UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Engenharia Civil, Arquitetura e Urbanismo

ALEXANDRE BARROZO DO AMARAL VILLARES

TAXONOMIA DE TEMAS PARA ENSINO DE PROGRAMAÇÃO EM CONTEXTO VISUAL

CAMPINAS

2019

ALEXANDRE BARROZO DO AMARAL VILLARES

TAXONOMIA DE TEMAS PARA ENSINO DE PROGRAMAÇÃO EM CONTEXTO VISUAL

Dissertação de mestrado apresentada à Faculdade de Engenharia Civil, Arquitetura e Urbanismo da Unicamp, para obtenção do título de mestre em Arquitetura, Tecnologia e Cidade, na área de Arquitetura, Tecnologia e Cidade.

Orientador: Prof. Dr. Daniel de Carvalho Moreira

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DEFENDIDA PELO ALUNO ALEXANDRE BARROZO DO AMARAL VILLARES ORIENTADO PELO PROF. DR. DANIEL DE CARVALHO MOREIRA.

CAMPINAS

2019

Agência(s) de fomento e nº(s) de processo(s): Não se aplica.
ORCID: https://orcid.org/0000-0001-7092-8654

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Elizangela Aparecida dos Santos Souza - CRB 8/8098

Villares, Alexandre Barrozo do Amaral, 1976-
V712t Taxonomia de tamas para ensino de programação em contexto visual / Alexandre Barrozo do Amaral Villares. – Campinas, SP : [s.n.], 2019.

Orientador: Daniel de Carvalho Moreira.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Civil, Arquitetura e Urbanismo.

1. Programação (Computadores) - Estudo e ensino. 2. Pyhton (linguagem de programação). 3. Tesauros. I. Moreira, Daniel de Carvalho, 1971-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Civil, Arquitetura e Urbanismo. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: A taxonomy of themes for teaching programming in a visual context
Palavras-chave em inglês:
Computer programming - Study and teaching
Python (programming language)
Thesaurus
Área de concentração: Arquitetura, Tecnologia e Cidade
Titulação: Mestre em Arquitetura, Tecnologia e Cidade
Banca examinadora:
Daniel de Carvalho Moreira [Orientador]
Hélio Pedrini
Luiz Ernesto Merkle
Data de defesa: 25-02-2019
Programa de Pós-Graduação: Arquitetura, Tecnologia e Cidade

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA CIVIL, ARQUITETURA E URBANISMO

TAXONOMIA DE TEMAS PARA ENSINO DE PROGRAMAÇÃO EM CONTEXTO VISUAL

Alexandre Barrozo do Amaral Villares

Dissertação de Mestrado aprovada pela Banca Examinadora, constituída por:

Prof. Dr. Daniel de Carvalho Moreira
Presidente e Orientador / UNICAMP

Prof. Dr. Hélio Pedrini
UNICAMP

Prof. Dr. Luiz Ernesto Merkle
UFTPR

A Ata da defesa com as respectivas assinaturas dos membros encontra-se no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 25 de fevereiro de 2019

AGRADECIMENTOS

Este trabalho só foi possível com o apoio, incentivo e colaboração de um grande número de pessoas e instituições. Os defeitos e limitações são de total responsabilidade minha, mas se alguma qualidade pode ser encontrada aqui, se deve sem dúvida a ajuda dessas pessoas, individual ou coletivamente. Agradeço minha família, minha esposa, pai, irmãos e em especial a minha mãe a quem dedico este trabalho. Agradeço profundamente meu orientador, Daniel. Agradeço diversos e tantos amigos que temo esquecer de mencionar algum: Décio Otoni de Almeida, Luciano Ramalho, Monica Rizzolli, Bernardo Fontes, João Antônio Ferreira, Marco Macarena, Pedro Guglielmo, Fabio Hirano, João Adriano, Mudrik, Allan, Omar Dalank, Rafael Otoni, Gustavo Chacon, Juliana Henno, Gabriele Landim, Erica Ide, Victor Scopacasa, Edu Bonini, Mike, Yumi, Juca Sanches, João Gaspar, Gil Barros e Leandro Veloso. Agradeço o carinho e acolhida de Carmita e Lorenzo. Agradeço meus amigos de infância que me ajudaram em momentos muito difíceis nos últimos anos. Agradeço meus amigos e colegas do Sesc Av. Paulista. Agradeço a numerosos bibliotecários e bibliotecárias, em especial das bibliotecas BAE-Unicamp, FAU-USP, IME-USP e Sesc Av. Paulista, Ludmila Almeida. Agradeço aos professores Maria Lucia Caira Gitahy, Clice Sanjar Mazzilli, Roberto Hirata Jr. e Ricardo Nakamura. Agradeço à Fundação Processing e seus múltiplos colaboradores, em especial Jonathan Feinberg (jdf). Agradeço aos dedicados professores, funcionários e colegas (tantos e tão generosos) da FEC-Unicamp.

RESUMO

O ensino de programação para artistas visuais, designers e arquitetos apresenta temas e abordagens particulares, especialmente em ambiente visual proporcionado por ferramentas de programação autônomas ou embutidas em programas de desenho e modelagem tridimensional. Partindo de uma análise da literatura do ensino introdutório de programação voltada ao público mencionado, identificando, comparando, decompondo e hierarquizando assuntos, este trabalho busca a caracterização de temas de programação, e exemplos de sua aplicação. Foi produzido um vocabulário controlado, ferramenta utilizada para assegurar uma linguagem comum a uma comunidade na organização, armazenagem e recuperação de documentos, com intuito de auxiliar professores e alunos a encontrar material didático apropriado, por meio da classificação de códigos-fonte que servem de exemplos didáticos. O vocabulário controlado, na forma de um tesauro, foi então aplicado na organização de exemplos disponíveis para uso com *Processing*, ferramenta largamente utilizada para a produção de resultados visuais com programação e que, acrescida do complemento *Python Mode*, permite a escrita e execução de código com sintaxe da linguagem *Python*. O tesauro e a coleção de exemplos classificados estão disponíveis para consulta, podendo futuramente ser ampliados, permitem identificar particularidades e lacunas nos temas atendidos pelos exemplos e servir de base para futuras pesquisas envolvendo material para ensino introdutório de programação em um contexto visual.

Palavras-chave:
ensino de programação (computadores); programação criativa; vocabulário controlado; *Processing* (linguagem de programação)*; Python* (linguagem de programação).

ABSTRACT

Teaching programming for visual artists, designers and architects involves a special set of themes and approaches, especially in a visual environment provided by stand-alone programming tools or embedded in design and 3D modeling software. Starting from a review of the literature of the introductory teaching of programming directed to the mentioned public, identifying, comparing, decomposing and laddering subjects, this work seeks to characterize programming themes, and examples of its aplicattion. A controlled vocabulary was produced, a tool to ensure a common language in a community for the organization, storage and retrieval of documents, in order to help teachers and students to find appropriate didactic material by classifying source code examples. The controlled vocabulary, in the form of a thesaurus, was then applied in the organization of examples available for use with Processing, a tool widely used for the production of visual results with programming and that, with the addition of Python Mode, allows the writing and execution of code with the Python programming language syntax. The thesaurus and the collection of classified examples are available for online consultation, and may be extended in the future, they allow one to identify particularities and gaps in the topics covered by the examples, and can serve as a basis for future research involving introductory teaching material in a visual context.

Keywords:
introductory programming, creative coding, controlled vocabulary, Processing, Python

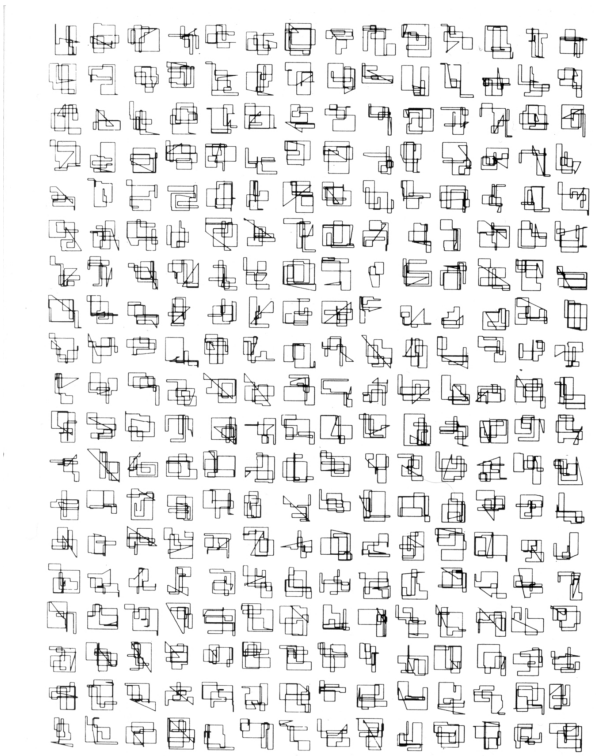
Sumário

1. Introdução

Artistas visuais, designers e arquitetos estão entre os que têm se beneficiado de novas ferramentas de programação. Para o ensino deste público, assim como para o uso profissional, estão disponíveis muitas linguagens de programação. São linguagens de uso geral e corrente em diversas áreas, algumas embutidas em programas de desenho e computação gráfica, como programas *CAD/BIM*[1] e modeladores 3D, e também linguagens e ferramentas de programação voltadas especialmente para as artes visuais e para o design.

Desde a década de 1960 este público tem experimentado (Figura 1.1) o uso de tecnologias computacionais em suas obras[2](VICTORIA AND ALBERT MUSEUM, 2018) e diversas linguagens surgiram com o objetivo de tornar a programação mais adequada às necessidades de pessoas com formações e interesses diversos (MELLO, 2015).

Figura 1.1 – Georg Nees, 23-Ecke (poklygon of 23 vertices), Computer-generated drawing, 1965



Fonte: Material suplementar a Nake (2018), disponível em : <https://www.mitpressjournals.org/doi/suppl/10.1162/leon_a_01325>

[1] *Computer Aided Design* e *Building Information Modeling.*

[2] *"Since the 1960s, artists and programmers have used computers to create prints, drawings, paintings, photographs and digital artworks."* (VICTORIA AND ALBERT MUSEUM, 2018)

Uma das primeiras tentativas de reduzir a barreira para o ensino de programação e alcançar pessoas com interesse em resultados visuais foi a linguagem *Logo* criada por Seymour Papert, Cynthia Solomon e Wally Feurzeig, que se destacava pelo resultado gráfico produzido com movimentação de um cursor gráfico tradicionalmente representado como uma tartaruga. Há relatos de que esta nomenclatura seria uma homenagem ao trabalho de Grey Walter com robôs tartaruga a partir de 1948 e uma das primeiras implementações robóticas de tartaruga com caneta (Figura 1.2), para uso com *Logo*, foi feita em 1971 por Tom Callahan (HOGGET, 2010).

Figura 1.2 - Imagem de capa do livro de *Mindstorms*

Fonte: HOGGET, 2010. Disponível em: <https://web.archive.org/web/20180508042337/http://cyberneticzoo.com/cyberneticanimals/1969-the-logo-turtle-seymour-papert-marvin-minsky-et-al-american/>.

Baseado originalmente na linguagem de programação *Lisp*[3], ainda hoje são desenvolvidas variantes de *Logo*, como o *NetLogo*[4] que explora a possibilidades de interação entre um número grande de agentes, o que pode ser ilustrado por um cenário de uma quantidade enorme de tartarugas. São influências de *Logo* o

[3] Uma das primeiras linguagens de programação de alto nível cuja família de variantes modernas incluem *Scheme*, *Racket* e *Common Lisp* <https://common-lisp.net/>.

[4] Utilizado por Coates (2010), tem como site oficial: <http://ccl.northwestern.edu/netlogo/>

módulo de desenho *turtle*[5] na linguagem *Python*, as ferramentas de "caneta" na linguagem de programação introdutória *Scratch*, ou ainda as tartarugas robô que podem ser adicionadas para fins educacionais ao jogo *Minecraft*[6], controladas pela linguagem de programação *Lua*.

Em Burry (2011), ao descrever sua experiência no ensino de arquitetura “infiltrada pela programação” em 1993, o autor identifica dois dos usos do código no contexto do design:

1. Para a criação de ferramentas que auxiliam na produtividade automatizando tarefas, viabilizando trabalhos que seriam impossíveis ou pouco práticos de outra maneira.
2. Para a experimentação em design com uso de programação.[7]

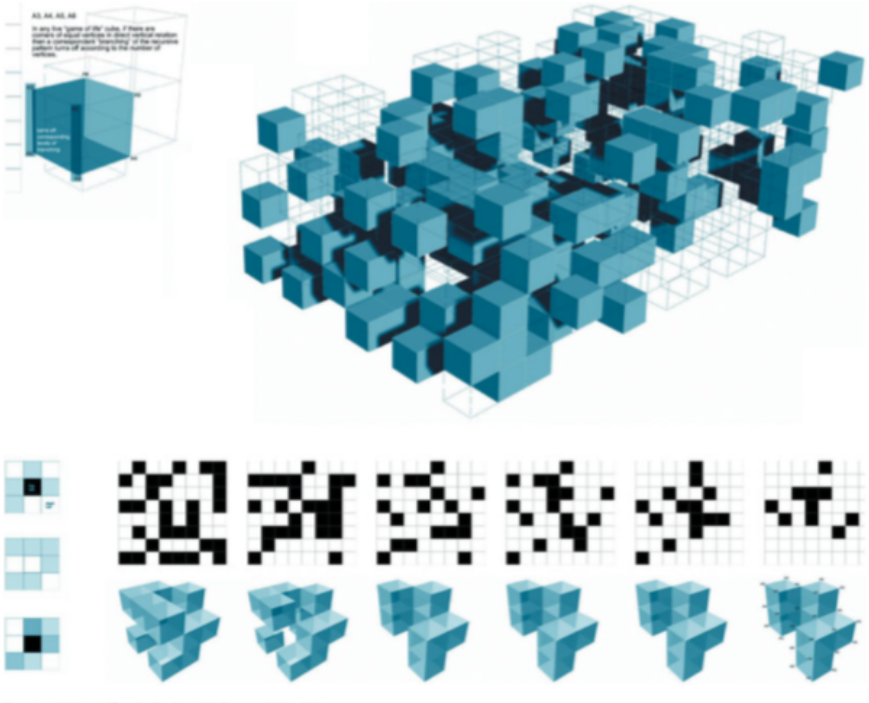
Um exemplo de experimentação em design com código é trazida por Silver (2006), que descreve como Brandon Williams e Studio Rocker (Figura 1.3) produzem volumes a partir da tradução tridimensional de padrões gerados pelos autômatos celulares do Jogo da Vida de Conway, um tema encontrado em diversas das obras pesquisadas.

[5] Documentado em <https://docs.python.org/3/library/turtle.html>

[6] Por meio da modificação *Computer Craft Edu* <http://computercraftedu.com/>

[7] No trecho "*[...] The two problems that had encouraged me to step outside my professional comfort zone of compliant passenger to become front-seat driver were stimulated by a need to rid myself of repetitive work [...]. Very quickly, however, I could see that a prime motivation for coding on top of software was to augment my design practice by allowing me to work in ways hitherto impractical, and so scripting became a medium of experimentation ahead of productivity gain. [...] Within a year, scripting had infiltrated my teaching, and in 1993 I instituted an elective course in which the participants had to come up with two pieces of code: a productivity tool [...] and a design experimentation script. [...]*" (BURRY, 2011, p.29-30)

Figura 1.3 - Brandon Williams/Studio Rocker, 3-D Game of Life, 2004



Fonte: SILVER. 2006, p. 24

Na presente década de 2010, o número de estudantes de artes nas novas mídias (*new media art*), design, arquitetura e mesmo nas ciências humanas, que usam programação em seus trabalhos aumentou substancialmente, afirma Manovich (2013), apontando também que uma das ferramentas que facilitou o acesso desses públicos teria sido o *Processing*, entre outras linguagens, que não necessariamente tornaram a programação em si mais fácil, mas sim permitiram gerar resultados interessantes de maneira mais eficiente.[8]

Para exemplificar o grande número de usos da programação no domínio do design e das artes visuais, há uma seleção de obras destacadas pela Fundação Processing[9], dentre as quais se pode observar os trabalhos da série *Surface:*

[8] No trecho "*In the present decade, the number of students in media art, design, architecture, and humanities who use programming or scripting in their work has grown [...] To a significant extent, this is the result of new programming and scripting languages such as JavaScript, ActionScript, PHP, Processing, and others.[...] These programming and scripting languages and APIs did not necessary made programming itself any easier. Rather, they made it much more efficient.*"
(MANOVICH, 2013, p.9)

[9] VISNJIC, Filip. Exhibition: A curated collection of projects created with Processing. Disponível em: <https://processing.org/exhibition/>. Acesso em: 29 dez. 2018.

Possible, Plausible, Potential (Figura 1.4) do designer e programador Nóbrega que afirma utilizar “o código tanto como sistema de representação como meio de expressão” e que escreve código de maneira a tornar suas ideias visíveis em desenhos"(NÓBREGA, 2015).

Os desenhos, que podem ser descritos como um encadeamento de volumes em perspectiva isométrica, são gerados por um programa desenvolvido por Nóbrega, se valendo de um mesmo conjunto de regras, mas produzindo desenhos únicos devido a algumas escolhas condicionais determinadas pseudo-aleatoriamente, e que são, finalmente, impressos por meio de uma plotter de caneta.

O número de artistas que escrevem o seu próprio software aumentou significativamente desde a introdução de computadores pessoais. Apesar das dificuldades técnicas e culturais que ainda persistiam, comenta Silver (2006), uma das barreiras seria um número maior de ferramentas voltadas para estas pessoas, uma vez que em geral as linguagens e ambientes de programação eram

Figura 1.4 - Surface: Possible, Plausible, Potential

Fonte: NÓBREGA, 2015. Disponível em: <http://superficie.ink/>

desenvolvidas por engenheiros de software e outros programadores com necessidades específicas diferentes das dos artistas. O movimento *open-source*, na sua visão, teria o potencial de servir de catalisador numa mudança do uso de software (em especial de linguagens de programação utilizadas pelos próprios artistas) nas artes[10].

A abordagem do ensino de programação em design, arte e arquitetura se dá de modo característico, uma vez que os objetivos e aplicações são distintos daqueles voltados a outras áreas.

> "Há uma diferença fundamental entre ensinar programação para alunos de ciência da computação e para alunos de arte e design. Espera-se que os estudantes de ciência da computação se tornem programadores profissionais que poderão programar aplicações de missão crítica no futuro. Estudantes de arte e design, por outro lado, são usuários de programação que precisam da programação para criar seus artefatos exploratórios, experimentais e artísticos. Não estão necessariamente interessados nos meandros da linguagem por si só. Talvez uma boa analogia seja a diferença entre ensinar uma língua estrangeira a um linguista e a um viajante. O viajante precisa da linguagem para poder se comunicar com as pessoas, explorar o novo ambiente e para sobreviver. O linguista precisa entender a sintaxe, a semântica e a pragmática da linguagem, mesmo que nunca precise se comunicar com um falante nativo dessa língua."(AMIRI, 2011, p.205)

Um panorama de ferramentas de programação para produzir resultados visuais foi preliminarmente apresentado em Villares e Moreira (2017). São ferramentas que se mostram adequadas para ensino de um público de "programadores alternativos", tradução de Mello (2015) para o termo de Schachman (2012),

[10] "*The number of artists writing their own software has increased significantly in the last 35 years, especially since the introduction of the personal computer. [...]One of the largest obstructions is the lack of tools for writing software for artists. Existing programming languages and environments have been written by software engineers to meet their specific needs, which have always been different from the needs of artists. The open-source software movement has provided a way for artists to collaborate on the production of their own tools. It is my hope these open-source art software initiatives will serve as a catalyst for a dramatic shift in the use of software within the arts.*" (SILVER, 2006, p.36)

constituído por aqueles que não pretendem se tornar engenheiros de software ou não têm a programação como foco principal da atividade profissional.

Partindo do pressuposto da importância do resultado visual no ensino introdutório de programação para este público, foi ampliada a noção de que é preciso produzir material didático com abordagem visual, escasso em português. Subsequentemente, considerações sobre a relevância dos temas escolhidos para a criação de uma compilação estruturada de exemplos didáticos acabaram por indicar novas investigações. Com isso, o foco da pesquisa passou a ser a estrutura de organização de exemplos de programação.

Este trabalho busca a identificação e caracterização de temas de programação, e exemplos de sua aplicação, que possam, por meio de um sistema de classificação, ajudar professores e alunos a encontrar códigos de exemplo úteis para suas aulas, experimentações e estudos, tarefa encontrada cotidianamente pelos que se dedicam ao ensino e aprendizado de programação em contexto de resultados visuais.

Um vocabulário controlado é uma lista de termos que se referem a conceitos ou assuntos. Por meio de uma taxonomia de classificação baseada em assuntos (*subject-based classification*) é possível organizar os termos de um vocabulário em uma hierarquia, permitindo assim a produção de um tesauro (GARSHOL, 2004).

Tesauros são vocabulários controlados utilizados para assegurar uma linguagem comum a uma comunidade na organização, armazenagem e recuperação de documentos, explicitando o conhecimento de um domínio, a serviço da indexação, recuperação e acesso da informação (PINHEIRO; FERREZ, 2014).

Deve-se evitar a confusão do termo “tesauro” com o termo de uso geral que se aplica a obras de referência que agrupam palavras pela similaridade de significado, contendo sinônimos e por vezes antônimos. Os tesauros aqui mencionados funcionam principalmente em conjunto com ferramentas de busca

e um padrão de anotação de metadados, com os quais uma coleção de documentos ou objetos pode ser organizada.

Esta pesquisa visa contribuir para indexação, recuperação e acesso de recursos educacionais abertos voltados ao ensino de programação no contexto das artes visuais e design, produzindo um vocabulário controlado para classificação de códigos-fonte de exemplos didáticos.

Um tesauro foi produzido e aplicado na organização de exemplos disponíveis para uso com o *Processing Python Mode*, opção do ambiente de desenvolvimento mantido pela Fundação Processing, o *Processing IDE*.

O trabalho a seguir será apresentado da seguinte maneira:

- Conceitos para o ensino introdutório de programação
- Material
  - Obras analisadas
  - Linguagens de programação nas obras
  - Códigos-exemplo de *Processing* modo *Python*
- Métodos
  - Observação e decomposição em elementos
  - Elaboração inicial da taxonomia e comparação entre obras
  - Interpretação, refinamento da taxonomia e construção do tesauro
  - Aplicação do tesauro na classificação de exemplos
- Resultados
  - Comparação entre obras
  - Tesauro de termos
  - Exemplos didáticos classificados
- Conclusão

De forma a registrar formalmente produtos da pesquisa, que podem ser consultados de maneira mais útil por meio de interfaces de software, a partir do tesauro proposto e do repositório de exemplos classificados foram gerados relatórios (listagens) que podem ser encontrados nos apêndices B e C.

2. Conceitos para o ensino introdutório de programação

São aqui apresentados conceitos identificados por uma revisão na literatura de ensino introdutório de programação. Da hierarquia apresentada em Luxton-Reilly et al (2017) foram destacados os assuntos mencionados, de maneira convergente, em um número significativo de fontes utilizadas por estes autores. Foi também feita a identificação de assuntos em três obras analisadas no primeiro momento da pequisa, estas últimas já incluem temas próprios do design e artes visuais no contexto do ensino de programação.

2.1 Luxton-Reilly et al. (2017)

Em uma revisão da literatura, procuraram estes autores encontrar uma hierarquia de conceitos em cursos introdutórios de programação, também conhecidos como CS1, sigla comum para disciplinas desta natureza em instituições de ensino anglófonas. Afirmando não terem obtido resultados na busca, passaram a interrogar a literatura a fim de identificar conceitos fundamentais do ensino introdutório de programação (*core concepts of CS1*) e expectativas chave para estes cursos. A revisão localizou sete artigos em que uma lista de conceitos foi construída a partir de diferentes fontes:

(1) Armstrong (2006) percorreu publicações em um recorte de quarenta anos a fim de estabelecer elementos essenciais (por ela nomeados "quarks") para o ensino de programação orientada a objetos, por fim identifica oito destes elementos de programação de computadores que separa nos grupos "estruturais" ou "comportamentais";

(2) Schulte e Bennedsen (2006) procuraram estabelecer quais conceitos os professores consideram mais importantes em um curso introdutório, utilizando listas compiladas por Dale (2005) e Milne e Rowe (2002). A lista de Dale (2005) foi produzida levantando entre professores quais tópicos haviam sido abordados em seus cursos. A lista de Milne e Rowe (2002) foi produzida pedindo a alunos que ordenassem conceitos de acordo com a dificuldade que neles percebiam;

(3) Tew e Guzdial (2010) analisaram a literatura e se valeram de quatro livros-texto dentre os mais utilizados em cursos introdutórios para estabelecer uma lista de conceitos fundamentais comuns, independentes de uma linguagem de programação específica, que foram em seguida utilizados para validar ferramentas de avaliação;

(4) Pedroni e Meyer (2010) examinaram a estrutura de dependências entre tópicos de programação orientada a objetos e como produto identificaram um conjunto de conceitos fundamentais de orientação a objetos que foram extraídos de material para preparação de cursos;

(5) Hertz (2010) construiu uma lista a partir do currículo modelo proposto pelo *Liberal Arts Computer Science Consortium,* LACS (2007)*,* porém combinando, generalizando, subdividindo e adicionando novos conceitos;

(6) Simon et al. (2012) examinou 20 provas de avaliação de cursos CS1 e para cada questão das provas identificou até três conceitos centrais avaliados por estas questões;

(7) Sanders et al. (2013) relataram o resultado do grupo de trabalho *ITiCSE 2013 Working Group* que desenvolveu um conjuto de 654 questões de múltipla escolha conhecido como *Canterbury QuestionBank*, abrangendo diversos tópicos de programação abordados em disciplinas CS1 e CS2 (cursos ministrados posteriormente a CS1). Cada questão foi marcada com até três tópicos de uma lista baseada em Simon et al. (2012) porém ampliada substancialmente para incluir tópicos de CS2 ou tópicos que surgiram no próprio processo de marcação.

Além das listas dos sete artigos acima mencionados, para garantir uma visão abrangente do que são considerados conceitos básicos de programação Luxton-Reilly et al. (2017) acrescentam ainda duas fontes ao material utilizado na construção de uma “lista mestre” de conceitos (Tabela 1):

(8) Lista de conceitos do LACS (2007), a mesma que serviu de ponto de partida para Hertz (2015);

(9) *Fundamental Programming Concepts and Fundamental Data Structures from the Software Development Fundamentals* (SDF) do *ACM Computing Curricula Task Force* (JOINT TASK FORCE ON COMPUTING CURRICULA; ASSOCIATION FOR COMPUTING MACHINERY; IEEE COMPUTER SOCIETY, 2013).

Luxton-Reilly et al. (2017) explicaram a construção de sua lista mencionando as seguintes considerações:

- Entradas em comum nas listas das fontes foram identificadas e mescladas ou divididas. Por exemplo, o conceito de "objetos e classes" encontrado em (2) Schulte e Bennedsen (2006) foi dividido em duas categorias separadas ("objetos" e "classes"), enquanto "iteração" e "estruturas de controle iterativo" (laços) foram fundidos em uma única categoria.
- Alguns conceitos foram eliminados da lista: "Fundamentos de Orientação a Objetos (OO)" foi considerado coberto por outros conceitos de nível inferior na hierarquia, e não parecia claro o suficiente quais conceitos de OO seriam considerado básicos (fundamentais).
- Entre os conceitos que foram eliminados estava por exemplo "Design por contrato" e "ética", pois, apesar de importantes, eram temas que não são diretamente relacionados à sintaxe e semântica e raramente são cobertos nos cursos CS1. Considerou-se que tópicos como "ética", embora interessantes em um curso CS1, normalmente não são cobertos em programas introdutórios.
- Finalmente qualquer tópico considerado pelas fontes que tipicamente fosse ensinado em um curso CS2 foi descartado e os demais conceitos foram agrupados em categorias. Algumas das categorias são elas mesmo um dos conceitos, mais amplo, identificado do exame da literatura.

Pode-se notar um certo grau de divergência entre as fontes, uma vez que dos 77 conceitos identificados por Luxton-Reilly et al. (2017), apresentados na Tabela 2.1, apenas 20 dos conceitos têm origem em 4 ou mais das 9 fontes. Todos os outros 57 conceitos foram relacionados em apenas uma, duas ou três fontes.

São listados a seguir os mencionados 20 conceitos que parecem apresentar maior consenso, e possivelmente maior relevância, no contexto do ensino introdutório de programação:

- Variáveis (*Variables);*
- Atribuição (*Assignment*);
- Expressões (*Expressions*);
- Escopo de variáveis (*Scope of variables*);
- Tipagem de dados (*Data types);*
- Cadeia de caracteres, dados alfanuméricos e símbolos (*Strings*);
- Arranjos (*Arrays);*
- Condicionais (*Conditional control structure*s);
- Laços de iteração *(Iterative control structures);*
- Funções, métodos e procedimentos *(Functions, methods & procedures);*
- *Recursão (Recursion);*
- Entrada/saída de dados (*Simple input/output);*
- Leitura e escrita de arquivos (*File input/output);*
- Objetos e instâncias (*Objects & instances);*
- Classes *(Classes);*
- Encapsulamento *(Encapsulation & information hiding);*
- Herança *(Inheritance);*
- Polimorfismo *(Polymorphism);*
- Normatização de estilos de programação e padronização *(Programming styles & standards);*
- Leitura de código (*Reading code)*

Tabela 2.1: Frequência de conceitos comuns em cursos introdutórios de ciência da computação (CS1)

Conceito	*Freq.*	*Fonte*
Variables & Assignment [Variáveis e atribuição]		
Variables	6	(9)(3)(5)(7)(2)(6)
Assignment	4	(9)(3)(4)(6)
Expressions	4	(9)(8)(5)(6)
Constants	3	(7)(2)(6)
Instance variables	2	(3)(2)
Scope of variables	6	(1)(3)(4)(7)(2)(6)
Lifetime	2	(7)(6)
Static & non-static variables	1	(2)
Data Types [Tipagem de dados]		
Data types	4	(8)(5)(7)(6)
Primitive data types	3	(9)(3)(4)
Integers	2	(3)(7)
Floating point	2	(3)(7)
Booleans	1	(3)
Strings	6	(8)(3)(5)(7)(2)(6)
Data Structures [Estruturas de dados]		
Arrays	6	(8)(3)(5)(4)(2)(6)
Matrices	1	(8)
Collections other than arrays	2	(7)(6)
Lists	1	(5)
Sets & relations	1	(8)
Structs & records	1	(7)
Linked list	3	(4)(7)(2)
Control Structures [Esturas de controle]		
Flow-control constructs	2	(8)(5)
Conditional control structures	5	(9)(3)(4)(2)(6)
Iterative control structures	4	(9)(7)(2)(6)
Loop (for)	1	(3)
Loop (while)	1	(3)
Loop (nested)	1	(3)
Events	2	(7)(6)
Exceptions	2	(8)(7)
Operations & Functions [Operações e funções]		
Arithmetic	2	(3)(7)
Relational operators	3	(3)(7)(6)
Logical operators	3	(3)(7)(6)
Boolean algebra	1	(8)
Functions, methods & procedures	6	(1)(9)(8)(5)(7)(6)
Function Definition	1	(3)
Parameters & param. passing	2	(3)(6)
Subroutines	2	(5)(6)
Accessor Methods	1	(3)
Mutator Methods	1	(3)
Return values	1	(3)
Static & non-static methods	1	(2)
Calling Functions		
Parameter passing	3	(3)(2)(6)
Dynamic binding	2	(1)(4)

Conceito	*Freq.*	*Fonte*
Recursion [Recursão]		
Recursion	7	(8)(3)(5)(4)(7)(2)(6)
Pointers & Memory Management [Ponteiros e gerenciam. memória]		
Pointers, references & memory management	2	(7)(2)
Input/Output [Entrada e saída]	2	(7)(6)
Simple input/output	5	(9)(3)(5)(7)(6)
File input/output	4	(9)(8)(7)(6)
Streams	2	(9)16
Graphical user interfaces	2	(7)(6)
Object-Oriented Concepts [Conceitos de orientação a objetos]		
Object-oriented concepts	2	(7)(6)
Objects & instances	4	(1)(3)(2)(6)
Classes	6	(1)(8)(3)(5)(2)(6)
Constructors	2	(3)(6)
Encapsulation & information hiding	4	(1)(3)(2)(6)
Message passing & object interaction	2	(1)(2)
Inheritance	5	(1)(8)(3)(5)(2)
Polymorphism	5	(1)(3)(4)(2)(6)
Interfaces (Java)	3	(8)(5)(7)
Abstract classes	1	(5)
Generics	2	(4)(2)
Instance variable types	1	(2)
Object identity	1	(6)
Libraries [Bibliotecas]		
Using language libraries	2	(7)(2)
Programming Process [Processo de Programação]		
Programming styles & standards **a**	4	(8)(5)(7)(2)
Reading code	4	(8)(5)(7)(6)
Debugging	2	(1)(2)
Design: algorithms	1	(2)
Design: classes **b**	2	(7)(2)
Design: methods **c**	2	(7)(6)
Design: programs	2	(1)(2)
Design: single class	1	(2)
Error handling	1	(2)
Testing	1	(7)
Abstract Programming Thinking		
Judgement **d**	1	(8)
Notional machine **e**	1	(8)
Problem solving strategies **f**	3	(5)(7)(6)

a Obedecendo a um conjunto de diretrizes ou padrões de estilo de programação.
b Identificando classes a partir de descrição de problemas. Schulte e Bennedsen (2) também incluem cartões CRC e "responsibility design" como categoria separada.
c Dadas as classes necessárias para resolver um problema, especifique os métodos requeridos.
d Escolher a estrutura de dados ou algoritmo adequado, e ser capaz de justificar a escolha.
e Modelo mental do computador.
f Incluído decomposição de um problema (dividir para conquistar) refinamento incremental e outras estratégias de solução de problemas.

Fonte: Adaptado de Luxton-Reilly et al. (2017), p.53

2.2 Mitchel, Ligget e Kvan (1986)

Esta obra, descrita em mais detalhes no Apêndice A, tem em sua terceira parte o texto de introdução a programação com *Pascal* (descrito em 3.3). Dos exemplos apresentados por Mitchel, Ligget e Kvan (1986) foram destacados por Celani (2008) no Quadro 2.1 os seguintes conceitos: Parâmetros; Repetição e variação; Condições; Hierarquia; Recursão; Transformação; e Analogia biológica.

Quadro 2.1 – Categorias de exemplos em Mitchell, Ligget e Kvan (1986)

Concept	Exercise
Parameters	"Write a parameterized procedure to generate the basic vocabulary element [for the plan Mies of Van der Rohe's 'Brick Country House']. Use this in a program to replicate the plan. Then use it in a program to produce variations on this theme" (p.198).
Repetition and variation	"It has often been argued that the aesthetic success of a composition is a matter of appropriate balance between 'unity' (which may be established by regular repetition) and 'variety' (which may be introduced by changing parameters from instance to instance). Test this proposition by generating repetitive compositions with different degrees of variation. Provide a critical analysis of your results" (p.250).
Conditions	"Many town plans consist of regular street grids interrupted at various points. Examine some plans of this type. What are the conditions in which the grid is interrupted? Write a set of conditional rules that could be used to produce plans of this type, and discuss their effects" (p.321)
Hierarchy	"All of the elements and subsystems of the Doric order have names. Draw a tree diagram that depicts this hierarchy. Then write a program, structured in the same way, that generates the order" (p.351)
Recursion	"Gothic tracery is often recursive. That is, a large pointed arch is subdivided into smaller pointed arches, each of which is further subdivided in the same way, and so on. Write a recursive procedure to generate such tracery designs" (p.353).
Transformation	"The plan and elevation compositions of the modern architectural masters Le Corbusier, Frank Lloyd Wright and Alvar Aalto rarely display rigid axial symmetry in the classical manner. But on careful inspection, they can usually be found to display less obvious symmetries and carefully broken symmetries. Take a composition that interests you, and the exceptions and distortions that are used to break symmetry. Using the insights that you gain from this analysis, write a concise, expressive program to generate the composition" (p.476).
Biologic analogy	"Examine the leaves along a twig from a plant. Can you characterize the pattern that you see in terms of type and regular repetition? What changes from instance to instance? What kinds of arithmetic and geometric sequences are involved? Write a brief, illustrated analysis" (p.250).

Fonte: CELANI, 2008, p. 9

2.3 Maeda (2001) [Reedição de Maeda (1999)]

A partir da análise da reedição do livro de Maeda, que registra o seu projeto didático para a linguagem de programação *Desgin By Numbers,* é possível, no Quadro 2.2, indicar os principais assuntos de cada capítulo, e com isso reconhecer tanto os temas de introdução à programação (como iteração/repetição, variáveis e condicionais) como os elementos básicos de desenho (linha, cor).

Aparecem, mas não proeminentes como títulos de capítulos, as discussões de interação, aleatoriedade e um elaborado exemplo de autômatos celulares de Wolfram.[11] A obra menciona também a noção de programação por meio de *sketches,* prática que se aproxima da prototipagem, e que tem continuidade na comunidade *Processing*, projeto iniciado por Reas e Fry, alunos e colaboradores de Maeda.

[11] <https://natureofcode.com/book/chapter-7-cellular-automata/>

Quadro 2.2 - Capítulos do livro Design by Numbers

capítulo	título	comentário
1	*Begin*	Faz uma discussão introdutória sobre a utilidade do livro.
2	*Commands*	São apresentados os atributos gráficos: Paper (cor de fundo) e Pen (cor de traço)
3	*Line*	Apresenta Line, a linha e o sistema de coordenadas.
4	*Lines*	Discute a repetição manual e introduz a sintaxe de comentários no código.
5	*Variables*	Apresenta as variáveis e o mecanismo de atribuição, Set para definir nomes atrelados a valores.
6	*Repeat*	Introduz os laços de repetição.
7	*Calculate*	Explica expressões e operadores matemáticos `( ) + - * /`
8	*Dot*	Como gerar um ponto/pixel, sintaxe `Set [X Y] N`, e leitura da cor do pixel com a sintaxe [X Y].
9	*Dots*	Discute a produção de curvas com pontos e o problema da divisão por zero.
10	*Nest*	Apresenta laços aninhados, que permitem grades e outros resultados gráficos.
11	*Question*	Com `Same?, NotSame?, Smaller?` é apresentada a ideia de execução condicional.
12	*Commands*	Definindo 'procedimentos' com command. Carregamento de módulos externos com Load
13	*Time*	Introduzindo animações. Limpeza do frame com Paper
14	*Paint*	Input do mouse para a criação de uma ferramenta de desenho, usando o laço Forever
15	*React*	Discussão da interação (ainda com o input do mouse)
16	*Touch*	Interação via teclado
17	*Network*	Acesso a um dado externo <Net N> compartilhado (variáveis compartilhadas remotamente)
18	*Change*	Transposições e manipulações de pixels, <Time N>, autômatos celulares de Wolfram e de Conway
19	*Numbers*	Funções com a sintaxe `Number NOME N{}` e invocação `<NOME N>` e discussão sobre aleatoriedade.
20	*End*	Considerações finais do autor
21	*Bibliography*	Referências

Fonte: Quadro elaborado pelo autor.

2.4 Reas e Fry (2007)

Nesta edição que antecede Reas e Fry (2014) os autores trazem um sumário resumido (*Contents*), um segundo sumário mais detalhado (*Extended Contents*) e uma lista dos capítulos ordenada por categorias (*Contents by category*), esta última lista é material que motivou procedimentos posteriores de identificação e comparação de temas em outras obras nesta pesquisa. O Quadro 2.3, com breve

anotação do assunto principal de cada capítulo, foi elaborado a partir da classificação transversal dos autores em *Contents by category*.

Reas e Fry (2007) apresentam entrevistas e contribuições de diversos educadores e artistas, entre eles Hernando Barragán (criador do *Wiring*, base da plataforma de programação de microcontroladores *Arduino*). Não foram incluídos na análise do quadro os capítulos introdutórios sobre o ambiente de desenvolvimento integrado (IDE) de *Processing*, prefácios e apêndices, mantendo apenas os capítulos que contém exemplos de código. Os capítulos de entrevistas foram mantidos no quadro, porém não foram utilizados na identificação de temas.

Quadro 2.3 - Categorias em Reas e Fry (2007)

categoria	**capítulo**	**assunto**
Color 1	*Color by Numbers*	cor, como tipo e como output
Color 2	*Components*	cor
Control 1	*Decisions*	condicionais
Control 2	*Repetition*	repetição
Data 1	*Variables*	variáveis
Data 2	*Text*	texto
Data 3	*Conversion, Objects*	tipagem
Data 4	*Arrays*	estruturas de dados
Development 1	*Sketching, Techniques*	estratégias de desenvolvimento
Development 2	*Iteration, Debugging*	estratégias de desenvolvimento
Drawing 1	*Static Forms*	formas
Drawing 2	*Kinetic Forms*	movimento/animação
Image 1	*Display, Tint*	imagens bitmap
Image 2	*Animation*	movimento/animação
Image 3	*Pixels*	imagens bitmap
Image 4	*Filter, Blend, Copy, Mask*	imagens bitmap
Image 5	*Image Processing*	imagens bitmap
Input 1	*Mouse I*	input [interface humana]
Input 2	*Keyboard*	input [interface humana]
Input 3	*Events*	input [interface humana], programação orientada a eventos
Input 4	*Mouse II*	input [interface humana]
Input 5	*Time, Date*	input [tempo]
Input 6	*File Import*	input [dados externos]
Input 7	*Interface*	input
Math 1	*Arithmetic, Functions*	operações matemáticas
Math 2	*Curves*	geometria
Math 3	*Trigonometry*	geometria
Math 4	*Random*	aleatoriedade

Quadro 2.3 - Categorias em Reas e Fry (2007), continuação

Motion 1	*Lines, Curves*	movimento/animação
Motion 2	*Machine, Organism*	simulação, movimento/animação
Output 1	*Images*	imagens
Output 2	*File Export*	output [exportação de dados]
Shape 1	*Coordinates, Primitives*	sistema de coordenadas
Shape 2	*Vertices*	formas, geometria
Shape 3	*Parameters, Recursion*	formas, recursão
Simulate 1	*Biology*	simulação
Simulate 2	*Physics*	simulação
Structure 1	*Code Elements*	Sintaxe básica, comentários
Structure 2	*Continuous*	loop principal (setup & main loop)
Structure 3	*Functions*	funções e encapsulamento
Structure 4	*Objects I*	Orientação a objetos
Structure 5	*Objects II*	Orientação a objetos
Synthesis 1	*Form and Code*	Colagens, ondas, grades e fractais
Synthesis 2	*Input and Response*	Interface humana e interação
Synthesis 3	*Motion and Arrays*	Estruturas de dados
Synthesis 4	*Structure, Interface*	Orientação a objetos
Transform 1	*Translate, Matrices*	transformações do sistema de coordenadas
Transform 2	*Rotate, Scale*	transformações do sistema de coordenadas
Typography 1	*Display*	texto/tipografia
Typography 2	*Motion*	tipografia
Typography 3	*Response*	tipografia
Interviews 1	*Print*	[não analisado]
Interviews 2	*Software, Web*	[não analisado]
Interviews 3	*Animation, Video*	[não analisado]
Interviews 4	*Performance, Installation*	[não analisado]
Extension 1	*Continuing . . .*	integração, outras linguagens e tecnologias
Extension 2	*3D*	Gráficos tridimensionais
Extension 3	*Vision*	Visão computacional
Extension 4	*Network*	Redes
Extension 5	*Sound*	Som
Extension 6	*Print*	Saídas gráficas vetoriais
Extension 7	*Mobile*	Aplicativos móveis
Extension 8	*Electronics*	computação física (sensores/atuadores)

Fonte: Quadro preparado pelo autor.

Uma vez estabelecida a possibilidade de se identificar um repertório de assuntos para o ensino introdutório de programação a partir da literatura, e vislumbrado um conjunto de temas que é específico para o público de design e artes visuais nos autores pioneiros da abordagem para este público, serão apresentadas a seguir as obras selecionadas como material da pesquisa.

3. Material

Como fontes primárias foram selecionadas obras voltadas ao ensino introdutório de programação para o público de artistas visuais, designers e arquitetos. Foi definida uma amostra limitada de um número considerável, disponível atualmente, de livros, publicações extensas e em diversos idiomas que atendem ao critério mencionado. Em seguida são descritas as linguagens de programação e ferramentas de desenvolvimento encontradas nas obras, e por fim é descrito o corpo de exemplos didáticos que foi classificado com o vocabulário controlado produzido.

3.1 Obras analisadas

A seleção foi refinada, após leituras exploratórias, em um recorte que optou pela inclusão apenas de livros que tratassem de ferramentas de programação textuais, multiplataforma e livremente acessíveis. Foi dada especial atenção a obras que tratam das ferramentas de programação do ecossistema de *Processing*, descrito posteriormente, e que é identificado como relevante para este público por diversos autores (SILVER, 2006; BURRY, 2011; ALENCAR, CELANI, 2014; MANOVICH, 2013; MELLO, 2015).

Também foi utilizado Downey (2015), livro introdutório de ensino de programação em *Python,* para o público geral, cuja tradução em português é Downey (2016). Um texto recomendado em diversos cursos, teve uma de suas edições anteriores usada como base para material de apoio de disciplinas introdutórias no IME-USP[12]. A inclusão de Downey (2015; 2016) permite a comparação dos assuntos abordados em obra estabelecida para o ensino introdutório, mas que não tem o foco no público de design, arquitetura ou artes visuais.

Foi feita uma leitura preliminar das obras listadas no Quadro 3.1 que permitiu a elaboração do diagrama na Figura 3.1, explicitando a distribuição cronológica das obras e algumas das relações entre as mesmas. Celani (2008) considera Mitchel, Ligget e Kvan (1987) uma obra pioneira, tendo a autora tratado ela mesmo de

[12] IME-USP**.** *Aulas de Introdução à Computação com Python*: Edição Interativa. 2015. Disponível em: <https://panda.ime.usp.br/aulasPython/static/aulasPython/introducao.html>. Acesso em: 1 out. 2017.

temas similares em Celani (2003), livro no qual recomenda a leitura de Maeda (1999), e tendo incentivado a conversão dos códigos-fonte apresentados como exemplos em Mitchel, Ligget e Kvan (1987) por integrantes de seu grupo de pesquisa, para linguagem *Visual Basic for Applications* (VBA) embutida no *Autocad* (CELANI; LAZARINI, 2007) e para *Processing* (ALENCAR; CELANI, 2014).

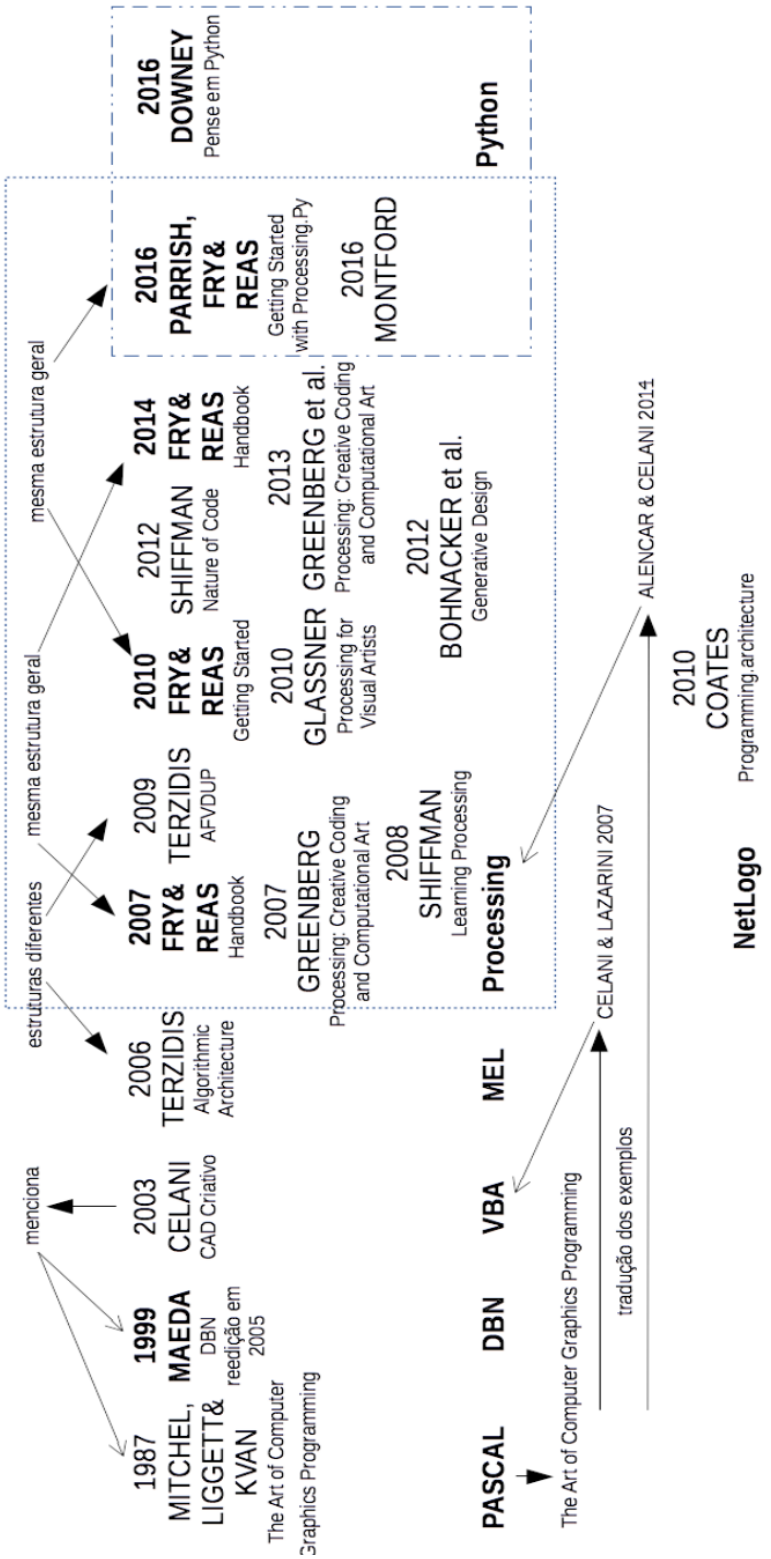
Quadro 3.1 – Primeira etapa de leituras

ano	**autores**	**título**	**linguagem de programação utilizada**
1987	MITCHELL; LIGGETT; KVAN	The Art of Computer Graphics Programming: A Structured Introduction for Architects and Designers	PASCAL
1999 [2001]	MAEDA	Design By Numbers [consultado na reedição de 2001]	DBN
2003	CELANI	CAD Criativo	VBA
2006	TERZIDIS	Algorithmic Architecture	MEL
2007	FRY; REAS	Processing: a programming handbook for visual designers and artists	Processing (Java)
2007	GREENBERG	Processing: creative coding and computational art	Processing (Java)
2009	TERZIDIS	Algorithms for Visual Design Using the Processing Language	Processing (Java)
2010	COATES	Programming.architecture	NetLogo
2010	FRY; REAS	Getting started with Processing	Processing (Java)
2010	GLASSNER	Processing for Visual Artists	Processing (Java)
2012	SHIFFMAN	Nature of Code	Processing (Java)
2012	BOHNACKER; GROSS; LAUB	Generative Design: Visualize, Program, and Create with Processing	Processing (Java)
2013	GREENBERG; XU; KUMAR	Processing: Creative Coding and Generative Art in Processing 2	Processing (Java)
2014	FRY; REAS	Processing: a programming handbook for visual designers and artists (2a ed.)	Processing (Java)
2016	PARRISH; FRY; REAS	Getting started with Processing Python Mode	Processing (Python)
2016	MONTFORT	Exploratory Programming for the Arts and Humanities	Python; Processing (Java); JavasScript

ano	**autores**	**título**	**linguagem**
2015	DOWNEY	Think Python: How to Think Like a Computer Scientist	Python
2016	DOWNEY	Pense em Python	Python

Fonte: Quadro elaborado pelo autor.

Figura 3.1 – Diagrama da cronologia, relações entre as obras e linguagens de programação



Fonte: Diagrama elaborado pelo autor

3.2 Linguagens de programação nas obras

As obras analisadas adotam diferentes linguagens de programação, indicadas no Quadro 3.1 anteriormente apresentado. Em ordem cronológica da sua primeira aparição no corpo de material pesquisado temos:

PASCAL, Design by Numbers, Visual Basic for Applications, MEL Script, Processing (Modo Java), Processing (Modo Python), JavaScript, Python.

Segue uma apresentação mais detalhada de quatro destas linguagens de maneira a permitir uma melhor compreensão das obras.

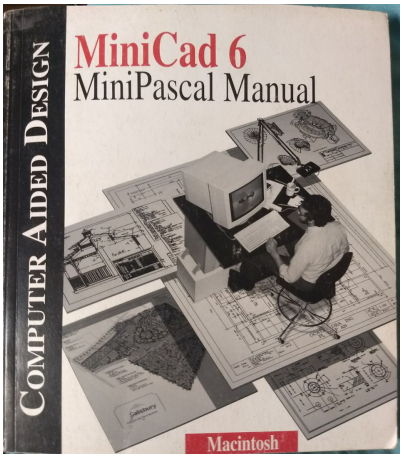
3.2.1 Pascal

A linguagem de programação *Pascal* foi criada por Niklaus Wirth no final da década de 60 e lançada originalmente em 1970 com intenção de ser usada educacionalmente (KRUGLYK; LVOV, 2012, ISO 7185:1990). Tendo sido largamente utilizada como linguagem introdutória nos cursos de ciência da computação e engenharia foi também adotada na indústria, ganhando variantes estendidas para uso com orientação a objetos na segunda metade da década de 1980, conhecidas como *Object Pascal*. Segundo Guzdial (2011), *Pascal* foi sucedida por *C++* e posteriormente por *Java* como a linguagem mais popular para os cursos introdutórios de programação nas disciplinas de graduação de ciência da computação nos Estados Unidos[13]. A descendente contemporânea, já também acrescida de recursos para orientação a objetos, é a linguagem e ambiente de programação *Delphi*.

Em 1988 o software CAD *MiniCAD*, na versão 1.0, incluía uma linguagem de programação baseada em Pascal denominada *MiniPascal* (Figura 3.2), ao que parece sem nenhuma relação com *Mini-Pascal*, de Benedict Løfstedt, derivada da variante *Pascal-S*. Posteriormente, passaria a ser conhecida como *VectorScript*, quando as versões mais novas do *MiniCAD* ganharam o nome *VectorWorks*, em 1998.

[13] <https://computinged.wordpress.com/2011/01/24/predictions-on-future-cs1-languages/>. Acesso em: 5 ago. 2018.

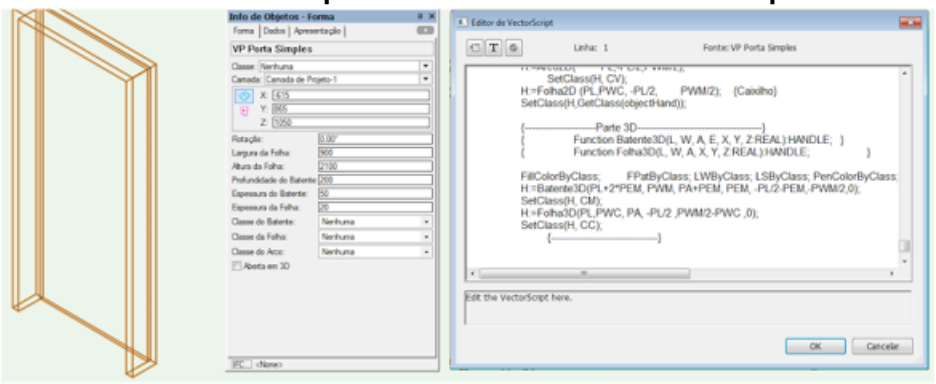
Figura 3.2 - Capa do manual de MiniPascal



Fonte: Foto da capa, acervo do autor.

A linguagem *VectorScript* foi utilizada para a criação de extensões do software CAD *VectorWorks* conhecidas também como ferramentas *Plug-in*, algumas das quais produziam "Objetos *Plug-in*" ou "Objetos Paramétricos" (Figura 3.3), criados tanto pelos desenvolvedores originais do programa como por usuários, em geral arquitetos, em contraste com o *Vectorworks SDK* (da sigla em inglês para "Kit de Desenvolvimento de Software"), ferramenta voltada aos desenvolvedores profissionais, utilizando a linguagem C++.

Figura 3.3- Exemplo de uma porta paramétrica desenvolvida pelo autor em 2001 em VectorScript



Fonte: Captura de tela produzida pelo autor.

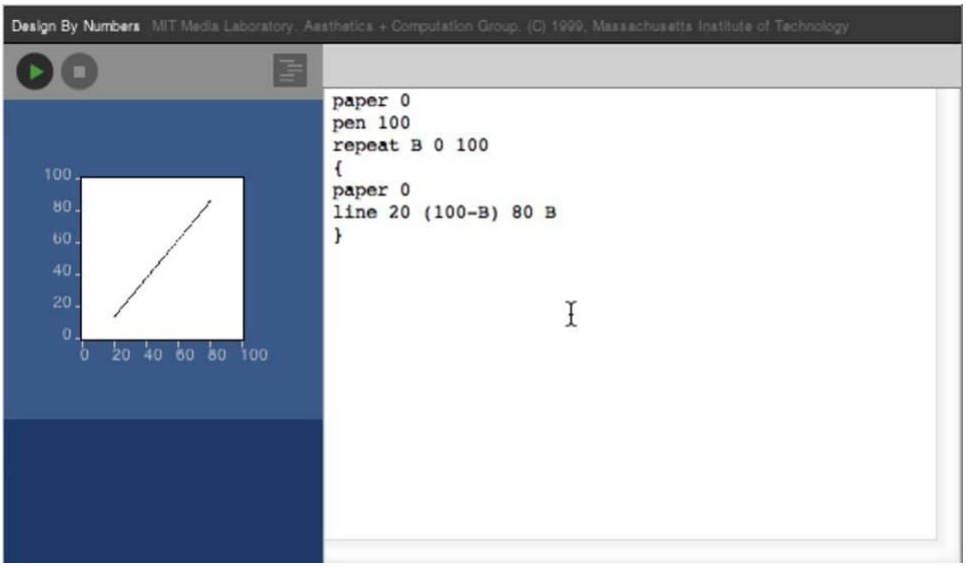
Desde 2015 o *Vectorworks* (cujo nome agora se escreve sem a maiúscula intermediária) inclui, além do *VectorScript* original, *Python* como opção moderna de linguagem de programação embutida. Em 2016 foi também introduzida a linguagem *Marionette*, ferramenta de programação descrita como visual, cuja interface de programação é baseada em nós e fios, emelhante a ferramentas como Bentley/GenerativeComponents, Rhino/Grasshopper, Revit/Dynamo ou Blender/Sverchok.

3.2.2. Design by Numbers

> "Com este livro, Maeda ensina tanto designers profissionais como amadores um processo de design que paradoxalmente tem um aspecto prático, quase de arte aplicada. Sua abordagem do design gráfico computacional não é diferente de uma abordagem para xilogravura. Destinada a exemplificar o desenho fundamental de um método de programação que se transforma em um processo de design visual este livro é uma inigualada tentativa de compartilhar conhecimento precioso. Conforme a consciência do novo meio continua a se disseminar, quanto mais designers aprenderem a disseminá-lo, mais designers aprenderão a incorporar o meio digital em suas práticas sem por ele ser incomodados." — Paola Antonelli no prefácio de Maeda (2001).

No *Massachusetts Institute of Technology* (MIT), John Maeda e alunos do grupo de pesquisa *Aesthetics + Computation* do *MIT Media Lab* desenvolveram a linguagem de programação *Design By Numbers* (abreviado *DBN*, Figura 3.4)[14] em 1999.

Figura 3.4 - Exemplo mínimo no Design By Numbers



Fonte: MASSACHUSETTS INSTITUTE OF TECHNOLOGY. *Design By Numbers*. 2001. Disponível em: <http://dbn.media.mit.edu/whatisdbn.html>.

Design by Numbers foi concebida como uma linguagem deliberadamente limitada, que oferecia comandos para desenhar em uma grade de 101 por 101 pixels, coordenadas de (0, 0) a (100, 100) e cores do branco ao preto em uma escala de cinzas (apesar de uma versão em certo momento oferecer o uso de cores em

[14] <http://dbn.media.mit.edu/>

caráter experimental). Permitia operações numéricas apenas com números inteiros. Desenvolvido ao longo do tempo com a ajuda de diversos colaboradores[15], *Design By Numbers* foi também um ambiente de desenvolvimento, um editor de código, que permitia a execução imediata dos programas escritos na linguagem.

A ambição de Maeda seria convencer os artistas gráficos a se tornarem programadores, segundo ele, a atitude de muitos artistas em relação às tecnologias da informação era, na melhor das hipóteses, ambivalente e, na pior das hipóteses, hostil[16] (BONNETT, 2000). Por conta disto teria buscado uma sintaxe simples que reduzisse as frustrações associadas à programação, em um processo de criação que fosse mais semelhante ao croquis (*sketch*) em um guardanapo, com iterações descartáveis e facilidade para se começar do zero, permitindo testar instantâneamente variações no código e sendo uma ferramenta livremente acessível em diversas plataformas de computadores (SURGUY, 2018).[17]

Design By Numbers, como linguagem, é uma antecessora direta de *Processing*: alunos de Maeda envolvidos no desenvolvimento e manutenção de DBN[18], Ben Fry e Casey Reas, em seguida criaram a nova plataforma, na qual a ideia de desenvolvimento por meio de *sketch,* um programa com característoca de esboço ou *croquis,* tem continuidade.

[15] *"DBN is the product of many people. Benjamin Fry created DBN 3 and 2. DBN 1.0.1 was created by Tom White. The original version DBN 1.0 was created by John Maeda. The courseware system was developed by Casey Reas, and translated to Japanese by Kazuo Ohno. Other people that have contributed to DBN development are Peter Cho, Elise Co, Lauren Dubick, Golan Levin, Jocelyn Lin, and Josh Nimoy."(Maeda, Home Page, 2001).*

[16] "Maeda's ambition in this work is straightforward. He wants to persuade graphic artists to become computer programmers. According to the author, the attitude of many artists towards information technologies has been at best ambivalent, and at worst hostile."

[17] <https://maxoffsky.com/research/research-essay-the-history-of-processing/>

[18] "*The then voluntary caretakers of the DBN system, Ben Fry and Casey Reas felt that they had the right designs for a new solution. It is called "Processing."And it is still going strong today in a way that continues to pleasantly astound me*."(Maeda; Antonelli, Design By Numbers , 2005).

3.2.3. Processing

> “É uma linguagem concebida especificamente para as artes visuais. Assim como diferentes materiais possibilitam diferentes resultados artísticos na escultura, na pintura ou na música, também na arte computacional diferentes linguagens de programação permitem diferentes resultados estéticos (REAS; MCWILLIAMS, 2010). O Processing apresenta funções nativas e uma estrutura de processamento interno otimizadas para a produção de imagens. Ademais, tornou-se uma linguagem amplamente usada em escolas de arte de todo o mundo no ensino de programação para artistas e designers." (MELLO, 2015)

Largamente empregado para o ensino em um contexto visual, *Processing* é utilizado para a criação de um grande número de obras nos mais diversos domínios das artes e do design, como peças gráficas, instalações interativas, projetos materializados por meio de fabricação digital entre outros exemplos. Manovich (2013) confirma:

> "Um bom exemplo de ambiente de programação contemporâneo, que é atualmente muito popular entre artistas e designers e que, na minha visão, é o mais próximo da visão de [Alan] Kay é o Processing. Construído sobre a linguagem de programação Java, Processing oferece um estilo de programação simplificado e uma extensa biblioteca de funções gráficas e de [manipulação de] mídias. Pode ser usado para desenvolver programas de mídias complexas e também rapidamente testar ideias"[19] (MANOVICH 2013, p.71)

Processing como ecossistema

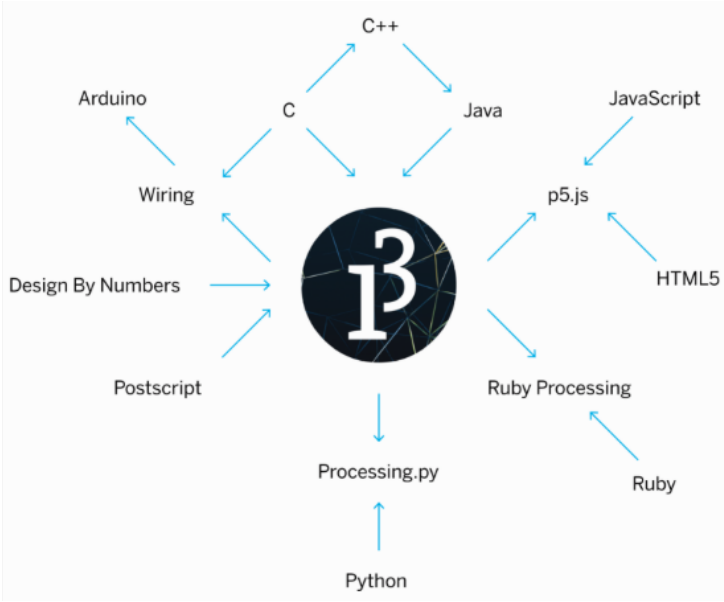
É desejável notar que o nome *Processing* acaba por se referir a muitas coisas. Uma das maneiras mais comuns de se mencionar *Processing* é (1) como uma linguagem de programação para artistas e designers, baseada em *Java*, ou ainda (2) como um

[19] "*A good example of a contemporary programming environment, which is currently very popular among artists and designers and which, in my view, is close to Kay's vision is Processing. Build on top of Java programming language, Processing features a simplified programming style and an extensive library of graphical and media functions. It can be used to develop complex media programs and also to quickly test ideas.*"

ambiente de desenvolvimento (*Integrated Development Environment* - IDE, da sigla em inglês para Ambiente Integrado de Desenvolvimento) que serve de (3) plataforma para outras ferramentas, linguagens e bibliotecas, (4) um conjunto de ideias (5) uma fundação (6) que a fomenta uma grande comunidade de pessoas que contribuem com este ecossistema de código aberto.

1. *Processing* como linguagem de programação é uma extensão de *Java,* uma linguagem desenvolvida inicialmente por James Gosling e lançada em 1995. Java é uma das mais utilizadas linguagens de programação para o desenvolvimento de software em geral, primeira no ranking *TIOBE Programming Community Index 2018*.[20] *Processing* é descrito por vezes como um superconjunto de *Java,* somando a esta um léxico gráfico inspirado no *OpenGL*, *Postscript* e *Design By Numbers* entre outras fontes (Figura 3.5). É possível incorporar código *Java* "convencional" em um *sketch* (um programa, em terminologia adotada pelos criadores, herdada do *Design By Numbers*) escrito em *Processing* e é possível acessar bibliotecas de terceiros escritas nessa linguagem.

Figura 3.5 - Família de linguagens relacionadas ao Processing



Fonte: Parrish; Reas; Fry, 2016, p.5

[20] <https://www.tiobe.com/tiobe-index/>

2. O *Processing IDE*, é um editor de código, um gerenciador de ferramentas e bibliotecas, com uma infraestrutura capaz de executar código escrito em *Processing* e exportar aplicativos "executáveis" independentes em diversas plataformas (Windows, Linux, macOS e Android). É possível utilizar a linguagem *Processing* em outros ambientes de desensenvolvimentos (IDEs), como por exemplo *Eclipse,* no entanto é o *Processing IDE* A ferramenta promovida preferencialmente pelos mantenedores do projeto, possibilita a utilização imediata da linguagem e pode ser estendido através da instalação dos chamados "modos" (em inglês *modes)* comentados a seguir.

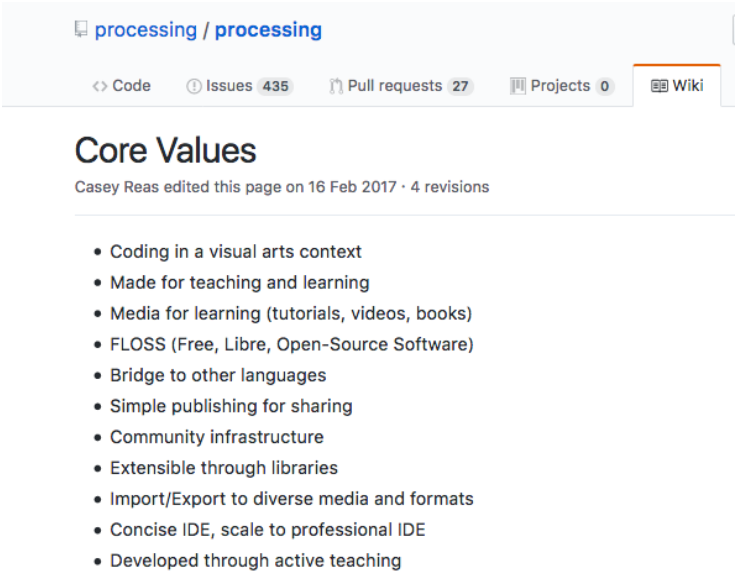
3. Ao ser executado pela primeira vez o IDE se inicia com o modo *Java* (*Java Mode)*, anteriormente conhecido como modo *Standard,* e permite a programação com sintaxe semelhante a *Java*. É possível então pelo próprio *IDE* baixar e instalar outros modos, como por exemplo o *Python Mode* que traz a sintaxe de *Python* e exemplos especialmente preparados. Por conta disso "tornou-se cada vez mais claro que o *Processing* não é uma linguagem única, mas sim uma abordagem orientada à arte de se aprender, ensinar e fazer coisas com código."[21]

4. *Processing* se tornou um conjunto de ideias e abordagens (Figura 3.6), não apenas uma linguagem ou linguagens em um *IDE*, sendo uma manifestação desta característica o projeto *p5.js*[22], uma biblioteca escrita na linguagem *JavaScript*, iniciada por Lauren McCarthy, hoje diretora da Fundação Processing. O *p5.js* re-implementa para o uso em páginas HTML, principalmente nos navegadores web, o vocabulário e as ideias de *Processing*.

[21] Disponível em: <https://py.processing.org/>. Acesso em: 5 jul. 2018. "*[...] it has become increasingly clear that Processing is not a single language, but rather, an arts-oriented approach to learning, teaching, and making things with code.*"

[22] <p5js.org>

Figura 3.6 - Valores Centrais de Processing



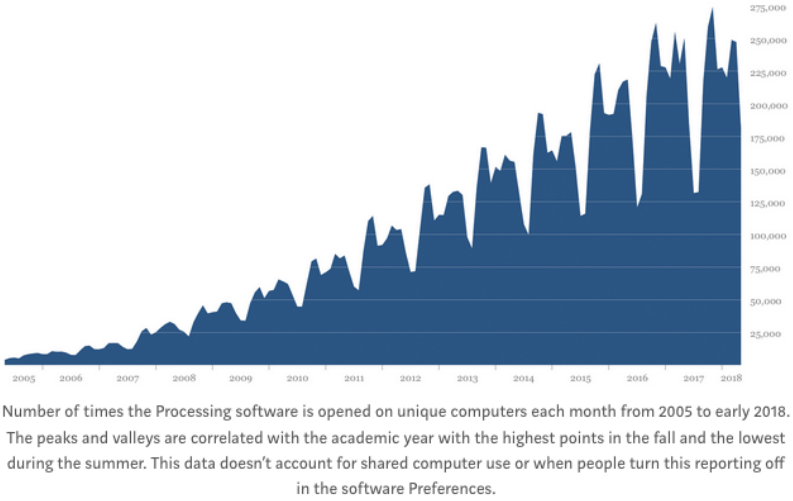
Fonte: <https://github.com/processing/processing/wiki/Core-Values>

5. A Fundação Processing foi iniciada em 2012 de maneira a manter e promover o projeto iniciado em 2001 por Reas e Fry. Sua missão é descrita sucintamente como sendo "[...] promover o letramento em software nas artes visuais e o letramento visual nos campos relacionados à tecnologia - assim como tornar esses campos acessíveis a diversas comunidades. Nosso objetivo é capacitar pessoas de todas os interesses e origens a aprender como programar e fazer trabalho criativo com código, especialmente aqueles que de outra forma não teriam acesso a essas ferramentas e recursos"[23].

6. Pode se considerar como a comunidade de *Processing* o numeroso conjunto de pessoas que utilizam (Figura 3.7), disseminam e contribuem com as ferramentas descritas até agora.

[23] *"Our mission is to promote software literacy within the visual arts, and visual literacy within technology-related fields —and to make these fields accessible to diverse communities. Our goal is to empower people of all interests and backgrounds to learn how to program and make creative work with code, especially those who might not otherwise have access to these tools and resources"* <processingfoundation.org>

Figura 3.7 - Número de vezes que IDE de *Processing* é aberto em computadores únicos



Fonte: PROCESSING FOUNDATION, 2018

Além das contribuições para o núcleo da linguagem e para o *IDE*, dentre ascontribuições mais essenciais da comunidade estão as bibliotecas, centenas delas: pacotes independentes de software que estendem a linguagem em diferentes direções, como por exemplo visualização de dados, simulações, vídeo e visão computacional. A maior parte das bibliotecas é desenvolvida independentemente por voluntários e o código fonte e exemplos são compartilhados para que todos possam usar e aprender com eles.[24]

Modos de Processing

Modos são complementos do IDE de Processing que permitem a programação usando linguagens (ou vocabulário e sintaxes) adicionais, em contraste ao modo anteriormente conhecido como *standard*, agora nomeado ***Java Mode***. Os modos também permitem suporte a plataformas de hardware específicas que dependem de infraestrutura ou bibliotecas específicas como o *Android Mode* e o *Raspbery Pi Mode.*

[24] <https://medium.com/processing-foundation/a-modern-prometheus-59aed94abe85>

Android Mode permite gerar aplicativos para dispositivos móveis com o sistema operacional Android, capazes de interagir com sensores e outras interfaces dos dispositivos móveis, tais como bússolas, giroscópios e acelerômetros. A sintaxe usada é a tradicional do *Processing Modo Java*, mas a instalação deste modo permite que o *IDE* interaja com o *Android SDK* produzindo os arquivos .apk que podem ser instalados pelo *Android*.

JavaScript Mode, atualizado pela última vez em 2015, é baseado no projeto *Processing.js* (ou *ProcessingJS*)[25], cuja última versão disponível é de março de 2014. *ProcessingJS* gerava a partir de código escrito com a sintaxe semelhante ao *Processing Java Mode*, um código *JavaScript* para execução em um navegador. Por muito tempo foi uma ferramenta popular para o compartilhamento de *sketches* (programas), utilizada em sites como *OpenProcessing*[26] e *HasCanvas*[27]. Este modo teve o seu uso reduzido e não é mais indicado como compatível com o *Processing IDE 3.0.*

P5.js Mode se vale da biblioteca *p5.js*, que tomou o lugar do *ProcessingJS* como principal forma de compartilhamento online de *sketches.* A biblioteca *p5.js* foi desenvolvida diretamente em *JavaScript* por Laure e fazendo uso do elemento *canvas*[28] é capaz de se beneficiar da *API WebGL*[29], as otimizações para desenho 3D no navegador. A biblioteca também é capaz de manipular a estrutura do documento HTML (DOM), na qual o *sketch* está inserido, e interagir com o navegador. *P5.js* normalmente é utilizada independentemente do *IDE Processing*, em IDEs variados, ou em seu site-editor[30], mas está disponível no modo *p5.js* dentro do *IDE,* que se torna capaz de gerar um arquivo HTML com as referências necessárias à biblioteca.

[25] <processingjs.org>

[26] <https://www.openprocessing.org/>

[27] <http://www.hascanvas.com/>

[28] <https://www.w3.org/wiki/Html/Elements/canvas>

[29] <https://get.webgl.org/>

[30] <https://editor.p5js.org>

Processing Python Mode foi desenvolvido principalmente por Jonathan Feinberg, com contribuições de James Gilles e Ben Alkov. O projeto, que também é conhecido como *Processing.py* e está documentado em um site da Fundação Processing[31], é baseado na implementação tradicional Java do Processing em conjunto com Jython. Jython é uma implementação de Python que é executada em uma Máquina Virtual Java (JVM, na sigla em inglês), permitindo acessar diretamente bibliotecas Java.

> "A vantagem desta estratégia é que os seus sketches Processing.py podem fazer praticamente qualquer coisa que um sketch Processing normal pode fazer, e versões novas do Processing.py aproveitam imeditamente as melhorias em performance ou novos recursos acrescentados na implementação Java. Quando você faz sketches com Processing.py, você tem acesso ao melhor de três diferentes plataformas e ecossistemas de software: Python, Java e Processing." (PARRISH, 2016)

Os exemplos, tutoriais e documentação de referência foram portados (traduzidos do modo Java) ou criados especialmente por James Gilles, Allison Parrish, e Miles Peyton, tendo o incentivo de Casey Reas, Ben Fry, Daniel Shiffman e Golan Levin. Créditos mais detalhados no site dedicado ao modo *Python* indicam ainda contribuições de outras pessoas assim como o papel que tiveram *Google*, *Carnegie Mellon University*, *National Endowment for the Arts* e outras instituições que apoiaram a iniciativa.

[31] <py.processing.org>

3.3.3. Python

A linguagem de programação *Python* foi concebida no final dos anos 1980 por Guido van Rossum como uma linguagem para o ensino e para automação de tarefas, uma chamada *scripting language* (VANDERPLAS, 2016, p.7). A terminologia *scripting language,* que tem sido questionada[32], designa principalmente linguagens interpretadas e que fazem integração entre outros componentes, aplicativos ou sistemas, estando associada também a linguagens de tipagem dinâmica (que não requerem a declaração dos tipos das variáveis), ambas características apontadas no sumário executivo[33] da *Python Software Foundation* (*PSF*). VanderPlas (2016) confirma que *Python* tem sido largamente adotada profissionalmente, em pesquisas científicas, de análises estatísticas à chamada ciência de dados, aprendizado de máquina, automação de processos, entre diversos outros usos.

Implementações de Python

A linguagem Python pode ser implementada de mais de uma maneira[34], e a implementação de referência, conhecida como *CPython*, é escrita na linguagem *C*. *IronPython* é uma implementação escrita na linguagem *C#* que permite estreita integração com o *.NET framework* da Microsoft. *Brython* e *Skulpt* são implementações na linguagem *JavaScript* que permitem a execução de *Python* em navegadores. *Jython* é uma implementação de Python que se vale da Máquina Virtual *Java* (JVM).

Python na educação

Guo (2014) contabilizou que Python já havia se tornado a linguagem introdutória mais utilizada nas principais universidades americanas em 2014. Tollervey (2015) reforçou a ligação inicial de Python com a educação comentando:

[32] <https://www.researchgate.net/post/Is_Python_a_Programming_language_or_Scripting_Language>

[33] <https://www.python.org/doc/essays/blurb/>

[34] <https://www.python.org/download/alternatives/>

> "Talvez isto explique o motivo de Python ser tão popular na educação: do início, Python é derivada de uma linguagem projetada para o ensino e mirando os programadores não-profissionais. No entanto, por se tornar uma plataforma aberta e extensível (Python é um projeto de código aberto), Python pôde crescer tornando-se a linguagem imensamente popular e flexível que é hoje, capaz de resolver efetiva e simplesmente muitos tipos diferentes de problemas computacionais." *(TOLLERVEY, 2015, p.1)*

Python e ferramentas para arquitetura e design

Villares e Moreira (2017) relata a presença de Python como uma linguagem embutida em um crescente número de ferramentas de desenho, modelagem 3D e ambientes de programação com suporte a produção de resultados visuais.

São exemplos descritos no referido artigo (Quadro 3.2) o modelador *Rhinoceros*, em que a linguagem *Python* foi acrescentada em substituição ao *RhinoScript*, o *Vectorworks* cuja linguagem *Vectorscript* (baseada em *Pascal*) foi substituida por *Python* e de maneira similar *Python* foi acrescentada ao *Maya*, *3D Max* e *Cinema 4D*.

Há possíveis formas de automação do *Revit* via *Python* (*Python Revit Shell* e pela criação de procedimentos no *Dynamo*). *Python* é acessível na interface das principais ferramentas de software livre para modelagem tridimensional *Blender* e *FreeCAD*, e também pode ser usado para controlar o *OpenSCAD*, com a biblioteca *solidpython*.

DrawBot[35], desenvolvido inicialmente por Just van Rossum em 2003, e posteriormente NodeBox[36], nele inspirado, permitem a exploração, com scripts em *Python*[37], de gráficos vetoriais 2D, especialmente para tipografia e design gráfico.

[35] <https://drawbot.com>

[36] <https://https://www.nodebox.net/>

[37] Apesar das versões mais recentes do NodeBox privilegiarem uma interface visual de nós e fios.

O projeto *Rosetta*[38]*,* do professor António Menezes Leitão, juntamente com seus colaboradores no Instituto Superior Técnico Universidade de Lisboa, permite a criação de modelos tridimensionais por meio de programação, controlando programas de modelagem utilizados por arquitetos, como *Rhinoceros* e *Autocad*, a partir de ambiente de desenvolvimento baseado no IDE *DrRacket.*[39] Pesquisadores do grupo implementaram para este ambiente a sintaxe de *Processing (P2R)*[40] e posteriormente uma variante de *Python (PyonR)*[41]*.* Atualmente pretendem avançar no desenvolvimento de uma interface em *Python* para modelagem tridimensional através da ferramenta *Khepri,* ainda em fase experimental[42].

Outros usos de Python com resultado visual

Para a produção de imagens com *Python* há um grande número de opções, muitas delas na própria biblioteca padrão[43], distribuídas como parte da implementação da linguagem, como os módulos *turtle*[44] (inspirado em Logo) e *tkinter*[45], no contexto de uma interface gráfica para o usuário (*GUI*) completa, ou ainda pacotes populares, que podem ser instalados separadamente, voltados para aplicações específicas, como por exemplo o *matplotlib* (visualização de dados) e *pygame* (programação de jogos).

[38] <http://web.ist.utl.pt/antonio.menezes.leitao/Rosetta/about/about.html>

[39] <https://racket-lang.org/>

[40] <http://web.ist.utl.pt/antonio.menezes.leitao/Rosetta/FinalReport/reports/HugoCorreia-Report.pdf>

[41]<http://web.ist.utl.pt/antonio.menezes.leitao/Rosetta/FinalReport/dissertations/HugoCorreiaThesis.pdf>

[41] http://web.ist.utl.pt/antonio.menezes.leitao/Rosetta/FinalReport/papers/ELS2014.pdf http://web.ist.utl.pt/antonio.menezes.leitao/Rosetta/FinalReport/dissertations/PedroRamosThesis.pdf

[42] <https://algorithmicdesign.github.io/tools.html>

[43] <https://docs.python.org/3/library/>

[44] <https://docs.python.org/3/library/turtle.html>

[45] <https://docs.python.org/3/library/tkinter.html>

Quadro 3.2 - Ferramentas com Python listadas em Villares e Moreira (2017)

ID	Evaluation	OS	License	Host/IDE	Language/Library	Syntax	Main uses
170	partial	Windows, Mac OS, Linux	Proprietary	Minecraft	[various]	blocks, Python, JavaScript, Lua	Education, games
180	partial	Windows, Mac OS, Linux	FOSS	NodeBox 3	NodeBox	nodes, Python	2D graphics, generative art, dataviz
71	yes	Windows, Mac OS, Linux	FOSS	Processing	Processing Python Mode	Python	2D and 3D graphics, intetactive art, generative art, dataviz
40	yes	Windows, Mac OS, Linux	FOSS	Blender	Python	Python	Plug-in tools, complex 3D geometry
60	yes	Mac OS	FOSS	Drawbot	Python	Python	2D graphics, generative art, dataviz
91	yes	Windows, Mac OS	Proprietary, educational at reduced cost	Rhinoceros	Python	Python	Plug-in tools, complex 3D geometry
122	yes	Windows, Mac OS	Proprietary, educational use at no cost	Vectorworks	Python	Python	Plug-in tools, parametric objects, complex 3D geometry
130	yes	online	-	Trinket.io (Browser based)	Python	blocks, Python, JavaScript	Education
140	partial	Windows	Proprietary	ArcGIS	Python	Python	Plug-in tools, GIS, dataviz
160	partial	Windows, Mac OS, Linux	FOSS	FreeCAD	Python	Python	Plug-in tools, complex 2D and 3D geometry
181	partial	Windows, Mac OS, Linux	FOSS	NodeBox for OpenGL	Python	Python	2D graphics, generative art, dataviz
200	partial	Mac OS (aplicativo ou módulo Python), Windows e Linux (módulo Python)	FOSS	PlotDevice	Python	Python	2D graphics, generative art, dataviz
210	partial	Windows, Mac OS, Linux	FOSS	QGIS	Python	Python	Plug-in tools, GIS, dataviz
260	no	Windows	Proprietary	3D Max	Python	Python	3D
270	no	Windows, Mac OS	Proprietary	Cinema 4D	Python	Python	3D
280	no		Proprietary	Maya	Python	Python	3D
220	partial	online	-	repl.it (Browser based)	Python [and many others]	Python [and others]	Education
320	no	Windows, Mac OS	Proprietary, educational use at no cost	Fusion360	Python, C++	Python, C-like	Plug-in tools, complex 3D geometry
230	partial	Windows, Mac OS, Linux	FOSS	DrRackett	Racket/ Rosetta	Lisp-like, Python, Java	Education, generative 3D geometry, complex 3D geometry
82	no	Windows	Proprietary, educational use at no cost	Revit, Vasari	RevitPythonShell	Python	Plug-in tools, complex 3D geometry

Fonte: Elaborado pelo autor.

No entanto nenhuma das ferramentas até aqui mencionadas aparentam ter a mesma simplicidade de uso e vocabulário amplo para exploração gráfica, que conjugados a implementação de um laço principal (como a estrutura *draw* do *Processing*), facilitam a criação de animações e programas interativos, ou ainda, e talvez principalmente, nenhuma delas tem o volume de documentação, bibliotecas e exemplos voltados às artes visuais e design encontrado em *Processing*.

O professor Claudio Esperança, que utiliza *Processing* em disciplinas na Universidade Federal do Rio de Janeiro - UFRJ, é autor de dois projetos relevantes: *ProcessingPy*[46] e *Skultpt IDE*[47]. O primeiro foi uma tentativa pioneira de recriar o ambiente de *Processing* com *Python*, entre 2009 e 2016. O segundo, *Skulpt IDE* provê um editor no navegador, com acesso a boa parte das funções de *Processing* em código com sintaxe de *Python*, sendo baseado em *Skulpt* e no *ProcessingJS,* ambos mencionados anteriormente. Em função destas características, não permite o uso nem das principais bibliotecas recentes de *Python*, nem das bibliotecas *Java* costumeiramente utilizadas pela comunidade *Processing*.

Mais recentemente, iniciado em 2017 no *Google Summer of Code* por Abhik Pal, o projeto *P5py*[48] pretende recriar a infraestrutura de *Processing* em *Python 3*, com o objetivo de se beneficiar de grande número de avanços em bibliotecas e ferramentas desenvolvidos pela comunidade *Python*, mas está ainda em estágio preliminar de desenvolvimento.

No presente momento, o *Processing Python Mode* traz a maior parte dos benefícios do ecossistema de *Processing* aqui previamente apresentado, tornando-se uma opção adequada para introdução de programação, aberta e livre, com muitas possibilidades de integração graças às comunidades ativas de *Processing* e *Python*.

[46] Baseado em Python 2 e Pyglet
https://github.com/esperanc/pyprocessing/blob/master/README.md

[47] <https://github.com/esperanc/skulptIde>

[48] <https://github.com/p5py>

Por ser baseado em *Jython*, que implementa a versão 2.7 de *Python*, o *Processing* modo *Python* não acessa bibliotecas que fazem uso de extensões em *C*, comumente usadas na implementação de referência da linguagem (o *CPython*), como por exemplo o popular pacote *numpy*. É possível, no entanto, importar bibliotecas descritas como "puro *Python*", que não usam extensões em *C,* ou bibliotecas escritas em *Java*.

3.3 Códigos-exemplo de *Processing* modo *Python*

Foi selecionado como material para classificação com o vocabulário produzido um conjunto de códigos-fonte disponíveis como exemplos didáticos no ambiente de desenvolvimento integrado (IDE) de *Processing*, mais especificamente com o acréscimo do *Python Mode* (ou modo *Python*).

O IDE de *Processing*, é um ambiente integrado de desenvolvimento, um editor capaz de executar código escrito em *Processing.* Os chamados modos (*modes*) são complementos do IDE que permitem a programação usando linguagens, vocabulário e sintaxe, adicionais, também permite suporte a plataformas de hardware específicas.

Em adição ao modo padrão do IDE, anteriormente conhecido como *standard,* agora nomeado *Java Mode,* e disponível pré-instalado, foi utilizado o *Python Mode*. O *Processing Python Mode,* também conhecido como *Processing.Py*[49], faz uso de *Jython*, uma implementação de *Python* capaz de acessar a infraestrutura e bibliotecas *Java* de *Processing*, e permite a escrita de *sketches* (programas) com sintaxe de *Python*.

Uma vez baixado o IDE de *Processing* e instalado o *Python Mode*[50] é possível acessar uma coleção de exemplos sob o menu Arquivo > Exemplos... (*File* >

[49] Documentado em <py.processing.org>.

[50] O IDE está disponível em <processing.org/downloads> e uma vez lançado apresenta no canto superior direito um menu que permite a instalação complementar do *Python Mode*. Os procedimentos de instalação são descritos em português em <https://abav.lugaralgum.com/como-instalar-o-processing-modo-python/>

Examples...). A coleção contém 249 códigos exemplo, é organizada em diretórios e subdiretórios (Figura 3.8) e incluí todos os arquivos necessários para estudo e execução do exemplo. Isto é, inclui código fonte (em arquivos .pyde e .py) e, quando necessário, outros arquivos: tabelas, texto, imagens, gráficos vetoriais ou fontes tipográficas.

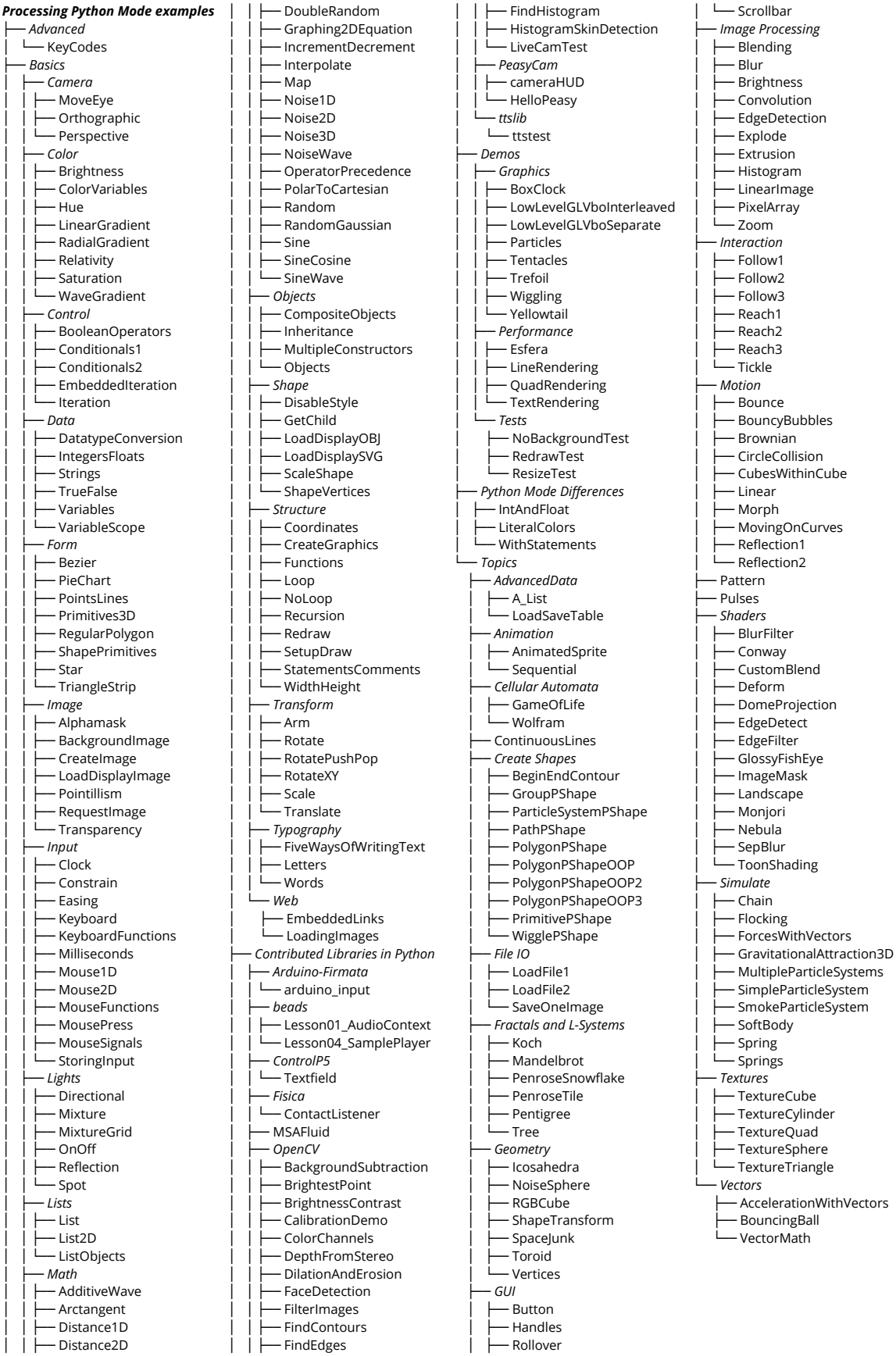
Os exemplos no *Python Mode* são em sua maior parte equivalentes aos disponíveis no *Java Mode*, "traduções" portadas para a linguagem *Python* dos exemplos originais, mas há alguns exemplos específicos do *Python Mode:* Três na categoria *Python Mode Differences* e 20 na categoria *Contributed Libraries in Python.*

Da mesma forma, há exemplos no *Java Mode* que não têm equivalente no *Python Mode.* É especialmente notável a ausência da categoria *Libraries* que agrupa bibliotecas padrão da Fundação Processing, mas há um esforço em andamento de voluntários para a conversão dos exemplos faltantes ao *Python Mode* no repositório público do projeto[51].

Apresentadas as obra analisadas, as linguagens de programação por elas utilizadas e o corpo de exemplos classificado com o vocabulário produzido, serão descritos os procedimentos adotados.

[51] Disponível em <https://github.com/jdf/processing.py>

Figura 3.8 - Exemplos do *Processing Python Mode*



Nota: os itens em itálico são diretórios e todos os outros itens são exemplos.

Fonte: Elaborada pelo pelo autor.

4. Métodos

A análise da literatura selecionada resultou em uma relação de termos que identificam os conceitos de programação abordados. O conjunto de termos foi então organizado como um vocabulário controlado, especificamente como um tesauro e um mapa de tópicos simples, que permite a exploração gráfica dos relacionamentos entre os termos do vocabulário. Em seguida foi aplicado o tesauro na classificação de 249 códigos-fonte exemplo do Processing Modo Python (apresentados nas seção 3.2).

Foram consideradas as práticas propostas por Gavrilova, Farzan e Brusilovsjy (2005), que interpretam ontologias educacionais como "conjunto estruturado de termos para descrever um domínio, que pode ser usado como esqueleto fundamental para uma base de conhecimento" e propõe um procedimento de cinco passos para produção dessas estruturas: 1. Desenvolvimento de glossário (*Glossary development*); 2. Escalonamento (*Laddering*); 3. Desintegração (*Disintegration*); 4. Categorização (*Categorization*) e 5. Refinamento (*Refinement*).

Foi também considerado o protocolo mínimo derivado do Estruturalismo descrito por Thiry-Cherques (2006): Observação; Decomposição em elementos; Conceitualização dos Elementos; Elaboração do Modelo; e Análise interpretativa.

Foram ainda analisadas as práticas de consolidação de listas de conceitos em Luxton-Reilly et al. (2017), apresentado no início deste trabalho, tendo então sido adotados os seguintes procedimentos:

- Observação e decomposição em elementos;
- Elaboração inicial da taxonomia e comparação entre obras;
- Interpretação, refinamento da taxonomia e construção do tesauro;
- Aplicação do tesauro na classificação de exemplos.

4.1. Observação e decomposição em elementos

Feita a leitura dos títulos descritos em Material - Obras Analisadas (3.1) houve um primeiro escrutínio de temas evidenciados pelos autores na organização de capítulos. Após a consideração de Mitchel, Ligget e Kvan (1987) e Maeda (2001) foi estudada a organização proposta em Fry e Reas (2007), a primeira edição da obra, que classifica transversalmente os capítulos em categorias, tendo sido consultada também a segunda, Fry e Reas (2014). Por fim foram selecionados para análise e comparação:

- Maeda (2001) [*Design By Numbers,* reedição da obra de 1999]
- Fry e Reas (2007) [*A Programming Handbook for Visual Designers and Artists*]
- Fry e Reas (2010) [*Getting Started With Processing*]
- Shiffman (2012) [*Nature of Code*]
- Downey (2016) [Pense em Python]

Em cada uma das obras selecionadas procurou-se um assunto ou tema que pudesse ser atribuído como principal para cada capítulo ao qual foi, na etapa seguinte, associado um conjunto único de letras, introduzido para facilitar a correlação de conteúdos entre diferentes obras.

A intenção inicial foi de atribuição de um único tema, predominante, a cada capítulo das obras analisadas, o que por vezes pareceu se mostrar insuficiente. A diferença de aprofundamento na subdivisão dos assuntos esconde, em alguns casos, a presença de um assunto ou subtema, que por sua vez foi relacionado em uma obra mas que não apareceu com proeminência no capítulo.

Por conta disto, foi realizada revisão na etapa de refinamento, ampliando, quando necessário, a lista de temas pelo exame dos subtópicos dos sumários nas obras que possuíam sumários detalhados.

A consolidação das anotações de identificação de temas das obras examinadas permitiu a construção do quadro de temas, e ao longo do processo, gerou revisões da categorização e codificação (reagrupamento em categorias, modificação das descrições e nomes dos temas).

4.2 Comparação entre obras e elaboração inicial da taxonomia

O quadro de codificação de temas é composto pelas colunas "código", "descrição", "categoria", "tema", "subtema". Na coluna código, um código de três letras (`aaa`) identificou as categorias (que podem ser consideradas também temas mais genéricos), códigos com seis e nove letras identificaram assuntos (temas) e subtemas (`aaa.bbb` e `aaa.bbb.ccc`, respectivamente). A coluna "descrição" dá nome ao item, que foi descrito abreviadamente nas colunas categoria, tema e, quando apropriado, subtema.

Em seguida, os assuntos são agrupados em categorias codificadas com três letras, e quando revelado conveniente subdivididos em subtemas codificados com três letras, como no exemplo:

categoria	assunto/tema	subtema	código	descrição
dat			**dat**	dados
dat	**var**		**dat.var**	variáveis
dat	**var**	**sco**	**dat.var.sco**	escopo de variáveis
dat	**typ**		**dat.typ**	tipagem

Os temas codificados podem então ser tratados como uma hierarquia, a lista expandida conforme a necessidade, na identificação de temas dos capítulos das outras obras selecionadas. Foi limitada a profundidade da taxonomia, com a intenção de reduzir a complexidade da estrutura, manter equilíbrio entre os ramos e nos subtemas optar por termos específicos mas de caráter ainda relativamente universal, evitando quando possível, por exemplo, detalhes de implementação ou palavras-chave específicas de uma linguagem.

Pela construção de uma matriz em que a primeira coluna identifica temas e subtemas por meio de seus códigos de seis e nove letras, nas colunas subsequentes, uma para cada obra examinada, foi indicado por meio do sinal "•" se o assunto foi encontrado com proeminência na estrutura de capítulos.

Construída com base na consolidação das anotações da etapa anterior de identificação de temas nas obras, a matriz destacou lacunas e sobreposições a ser

confirmadas e exploradas (nas categorias ou nas obras) e foi afetada pelas revisões do quadro de temas codificados.

4.3 Interpretação, refinamento da taxonomia e construção do tesauro

Nesta etapa foram acrescentadas obras cuja análise incluiu os subitens dos capítulos, foi feita uma revisão da taxonomia e em seguida a construção do vocabulário controlado em forma de tesauro.

4.3.1 Acréscimo de obras, análise de subitens de capítulos.

Foi acrescentada a análise das seguintes obras:

- Fry e Reas (2014)
  [*Processing: a programming handbook for visual designers and artists (2a ed.)*]
  Com atenção para os subtópicos dos capítulos e para diferenças entre a primeira edição, já examinada e esta segunda edição revisada.

- Parish, Fry e Reas (2016)
  [*Getting Started With Py.Processing*]
  Versão de Reas e Fry (2010) recriada para o Processing Python Mode, com atenção para diferenças da versão original (modo Java) já examinada.

- Downey *(2015)*
  [*Think Python*]
  Com atenção para os subtópicos e para as correspondências dos termos em inglês e português na edição em português, já examinada, informando a construção de termos equivalentes em inglês em um tesauro de referência.

Concomitante ao segundo escrutínio das obras, foi feita a revisão e adaptação da coleção de temas codificados e suas categorias.

4.3.2 Construção do tesauro

Tesauros, como definidos nas normas internacionais[52], estendem taxonomias de termos, nomes de assuntos organizados em uma hierarquia, tendo suas relações explicitadas por propriedades e anotações (GARSHOL, 2004).

> "Na organização do conhecimento, os esquemas de representação tais como classificações, tesauros, taxonomias e ontologias cumprem uma função importante, pois fornecem terminologias com as quais podem ser modelados um ou mais domínios. Na medida em que são vocabulários representantes de determinada área, por meio da sistematização dos conceitos e das relações que se estabelecem entre estes de forma compartilhada e consensual, asseguram que em uma comunidade todos utilizem a mesma linguagem para organizar, armazenar e recuperar a informação. Os esquemas, portanto, além de explicitar o conhecimento de um domínio e permitir a construção de mapas de conhecimento, promovem a padronização e a reutilização de suas representações; compartilham um entendimento comum da estrutura da informação; possibilitam a criação de novos conhecimentos a partir do existente; e, sobretudo, viabilizam indexar, recuperar e acessar informação." (PINHEIRO; FERREZ, 2014)

Foi utilizado o software de edição e compartilhamento de vocabulários controlados *TemaTres*[53], software livre que pode ser instalado em um servidor web capaz de abrigar bancos de dados *MySQL* e ferramentas desenvolvidas na linguagem *PHP.* Uma vez instalado seguindo as orientações disponíveis na documentação do projeto, *TemaTres* foi configurado para a edição de um vocabulário controlado nomeado “ensino de programação em contexto visual” e um segundo “*teaching programming on a visual context*” para abrigar os termos de um vocabulário de referência em inglês.

[52] Normas ISO 2788 e ISO 5964.

[53] Disponível em <https://www.vocabularyserver.com/>.

Aos termos de um tesauro são associadas propriedades, também chamadas de campos. Para a construção do tesauro proposto foram adotados as seguintes propriedades, indicadas por meio de suas abreviaturas padronizadas:

- TG, termos mais abrangentes (Termo Genérico);
- TE, Termos mais Específicos ;
- TR, Termos Relacionados;
- NE, Nota Explicativa (ou Nota de Escopo);
- USE, Usar preferencialmente, em lugar deste, os seguintes termos;
- UP, É usado preferencialmente em lugar dos seguintes termos.
- EQ, Termo equivalente em um vocabulário de referência (em inglês)

Partindo dos temas codificados, foi elaborada uma lista de termos. Termos com origem em temas, são relacionados aos termos genéricos (TG) com origem em suas categorias. Exemplo:

VARIÁVEIS
TG: DADOS

Reciprocamente, são os termos específicos (TE) dos termos das categorias, e tem como termos específicos os produzidos a partir de seus subtemas. Exemplo:

DADOS
TE: VARIÁVEIS
TE: TIPAGEM (de dados)
TE: TEXTO

Temas da mesma categoria, ou subtemas de um mesmo tema, são considerados termos relacionados (TR). Opcionalmente, podem ser produzidas notas explicativas (NE). Exemplo:

VARIÁVEIS
NE: Nomes associados a valores na memória.
TG: DADOS
TE: ATRIBUIÇÃO
TR: TEXTO
TR: TIPAGEM

Também opcionalmente, podem ser feitas indicações de que certos termos são de uso preferencial em relação a outros (UP) ou indicação de que um termo é preferível (USE). Exemplos:

TIPAGEM
UP: TIPOS (de dados)
TIPOS (de dados)
USE: TIPAGEM

Neste processo foram acrescentados termos específicos que não apareceram ainda com na estrutura das obras examinadas, mas que se encaixam na estrutura. E para a criação de um tesauro multilíngue foram relacionados a cada termo um equivalente em inglês (EQ). Exemplos:

INTEIRO
NE: Tipo de valor de dados dos números inteiros (*integer*).
TG: TIPAGEM (de dados)
EQ: *Integer* [termo acrescentado ao vocabulário de referência em inglês]

Com o objetivo de evitar que a hierarquia de termos se tornasse demasiado profunda, e aceitando como consequência limitar a granularidade e número total de termos propostos, foi adotada a limitação de 3 níveis para os termos, que se reflete nos códigos alfabéticos de até nove letras separados por pontos, como na taxonomia preparada preliminarmente para a comparação de obras: aaa.bbb.ccc.

Uma primeira lista de termos foi importada pelo *TemaTres* a partir de um arquivo texto com um termo por linha. Uma vez no site foram acrescentados e revisados os termos, acrecentando códigos identificadores únicos (nos formatos aaa, aaa.bbb e aaa.bbb.ccc) assim como ajustados os campos e as relações entre os termos. Para a redação de Notas Explicativas (também conhecidas como notas de escopo) e inclusão de termos específicos, foi feita referência aos glossários em Reas e Fry (2014) e Downey (2016).

4.4 Aplicação do vocabulário

Foi aplicado um procedimento de classificação a cada um dos 249 itens, exemplos de código do *Processing Python Mode* (descritos em 3.2). As anotações resultantes foram armazenados em um repositório *Omeka*, em registros providos de campos definidos segundo o padrão *Dublin Core* para metadados.

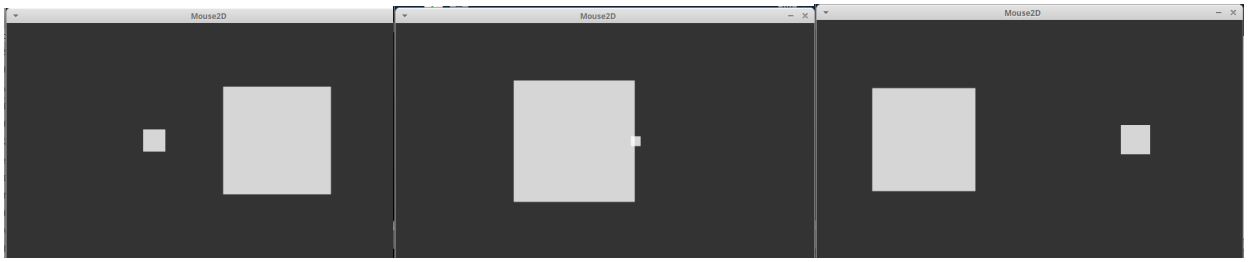
Abaixo é apresentada uma amostra de código-exemplo analisado, que resulta em animação interativa (Figura 4.1), exemplo que posteriormente será utilizado na demonstração do processo de classificação e anotação de metadados:

```
"""
Mouse 2D.
Moving the mouse changes the position and size of each box.
"""

def setup():
    size(640, 360)
    noStroke()
    rectMode(CENTER)

def draw():
    background(51)
    fill(255, 204)
    rect(mouseX, height / 2, mouseY / 2 + 10, mouseY / 2 + 10)
    inverseX = width - mouseX
    inverseY = height - mouseY
    rect(inverseX, height / 2, (inverseY/2) + 10, (inverseY/2) + 10)
```

Figura 4.1 – Frames capturados do exemplo Mouse2D



Fonte: Imagens capturadas pelo autor.

4.4.1 Classificação de exemplos didáticos (códigos-exemplo)

A classificação foi feita como descrito a seguir, tendo anotações pertinentes sido anotadas em uma planilha, cuja primeira coluna contém o nome do exemplo didático. Cada exemplo foi carregado no IDE de *Processing* e o código fonte foi examinado:

- Foi feita a leitura dos comentários iniciais, que em certos casos contém descrições, intenções dos autores e quando necessário instruções de interação. Por vezes os comentários sugerem experimentação com a mudança de constantes ou dados literais, no código;

- Foi feita uma leitura geral para a identificação da estrutura do código, observando a subdivisão em diferentes arquivos (que aparecem como abas no IDE) e opcionalmente reconhecendo funções, métodos e classes (no caso de código com orientação a objetos);
- O código é então executado e quando necessário há interação por meio dos dispositivos de entrada adequados. Em alguns casos foram feitas modificações no código, a título de experimentação;
- São eleitos e anotados até três termos únicos do tesauro proposto. O resultado desta classificação foi registrado no Apêndice C.

Foi feita a análise, breve descrição do exemplo e atribuídos os termos do tesauro proposto, anotados em um registro que posteriormente foi transferido para um repositório online, descrito a seguir, em que são considerados campos de metadados padronizados.

4.4.2 *Omeka* – repositório de coleções

Procurou-se selecionar uma ferramenta de software livre que fosse adequada para abrigar e dar acesso aos registros da classificação dos exemplos, que pudesse ser futuramente adaptada e que fosse compatível com modernos padrões de meta-dados.

O software *Omeka Classic*[54], é um gerenciador de coleções de itens utilizado por museus e diversas instituições de pesquisa, e que contém uma estrutura de campos para metadados compatível com o padrão *Dublin Core*, cumpre os requisitos mencionados e pode ser instalado em praticamente qualquer servidor web provido de banco de dados *MySQL* e capaz de executar a linguagem *PHP.*

A tabela produzida no processo de classificação de exemplos foi transferida para um repositório *Omeka.*

[54] Disponível em <https://omeka.org/>

4.4.3 *Dublin Core* – metadados

Para a criação dos formatos e registros do material classificado, foi consultado o padrão de anotação de metadados *Dublin Core Metadata Element Set* (DCMES)[55], parte do esquema de metadados para descrição de objetos digitais *Dublin Core*, promovido pela *Dublin Core Metadata Initiative* (DCMI).

O padrão *Dublin Core* é descrito como tendo dois níveis: Simples, com 15 elementos; e Qualificado, incluindo três elementos adicionais (Audiência, Proveniência e Detentor de Direitos), assim como um grupo de refinamentos de elementos (também chamados qualificadores), que refinam a semântica dos elementos de maneira que sejam úteis na descoberta de recursos.

Foram considerados neste procedimento quinze elementos do DCMES, aqui descritos e exemplificados:

1. Title: Título - Nome pelo qual o recurso é formalmente conhecido. Foi usado o nome do exemplo:

   Mouse2D

2. Creator: Autor - Pessoa ou entidade.

   `[Nem sempre disponível, não foi utilizado nesta ação.]`

3. *Subject*: Assunto / palavras-chave - Expresso com palavras-chave, descritores ou códigos de classificação que descrevem o tema do recurso. Foram utilizados até três dos termos do tesauro desenvolvido:

   (dat.inp.mou) mouse
   (dat.var) variáveis

4. *Description*: Descrição - A descrição pode incluir sumário do conteúdo, referências para uma representação de conteúdo ou um texto livre de

[55] Disponível em <https://tools.ietf.org/html/rfc5013>

relato do conteúdo. Foi produzido texto breve de descrição durante o processo de classificação:

Varia o desenho de dois retângulos com a movimentação do mouse.

5. *Publisher*: Editor - Inclui uma pessoa, uma organização ou entidade.

   Processing Foundation

6. *Contributor*: Contribuidor / colaborador - Pessoa, organização ou serviço

   [Não utilizado nesta ação.]

7. *Date*: Data - Data associada ao ciclo de vida do recurso, podendo se referir a criação ou disponibilização. Recomenda-se AAAA-MM-DD conforme norma ISO 8601.

   [Não utilizado nesta ação. Pode-se estudar a adoção de data disponível no repositório do mantenedor do projeto]

8. *Type*: Tipo do recurso – Natureza ou gênero do recurso. Recomenda-se utilizar vocabulário controlado[56]. Note que para descrever a forma de manifestação física ou digital do recurso deve-se usar o elemento Formato.

   Software

9. *Format*: Formato – Em geral descreve o tipo da mídia ou as dimensões do recurso, e pode ser usado para determinar o software, hardware ou outro equipamento necessário para mostrar ou operar o recurso.

   Processing Python Mode

10. *Identifier*: Identificador do recurso - Recomenda-se utilizar um sistema de identificação formal, como por exemplo uma URI. Foi utilizado o caminho do exemplo nas pastas, acrescido do nome do exemplo:

[56] Como o indicado em <http://dublincore.org/documents/dcmi-type-vocabulary/>

Basics>Input>Mouse2D

11. *Source*: Fonte - O recurso pode ser derivado, inteiro ou em parte, de uma fonte, preferencialmente indicada por um sistema de identificação formal. [Não foi utilizado, mas é intenção adotar URL do item no repositório público do *Python Mode.*]

12. *Language*: Idioma - A recomendação da RFC 1766 é a inclusão de um código de língua em 2 letras (padrão ISO 639), seguido opcionalmente pelo código do país em 2 letras (padrão ISO 3166). Foi usado para identificar o idioma dos comentários no código fonte:

    EN

13. *Relation:* Relacionado – Recurso relacionado.

    [Não foi utilizado nesta ação, mas poderia futuramente indicar exemplo análogo em outra ferramenta]

14. *Coverage:* Cobertura – Relevância jurisdicional, espacial ou temporal do recurso.

    [Campo não utilizado nesta ação]

15. *Rights:* Direitos – Informações a respeito de direitos autorais e licenciamento do recurso.

    [Não foi utilizado nesta ação, mas merece atenção posterior. Uma primeira avaliação indica licença livre permissiva (sem copyleft)]

Estabelecidos os procedimentos adotados de análise das obras, construção do tesauro a partir da taxonomia de temas e subsequente classificação do corpo de exemplos didáticos, serão em seguida descritos os resultados obtidos.

5. Resultados

Inicialmente serão apresentados os resultados dos procedimentos adotados de identificação e codificação de temas, uma coleção preliminar de temas significativos. Utilizando os temas, codificados como descrito na seção 4.2, são mostradas diferenças e sobreposições nos assuntos tratados em 5 obras, explicitadas por meio de uma matriz, ou quadro, de comparação.

For fim, notadas as especificidades na abordagem dos textos e acrescentada a análise das obras listadas na subseção 4.3.1, são propostas categorias, tendo em vista aspectos do ensino de programação em um contexto visual.

5.1 Comparação entre obras

Como exemplo de comparações que podem ser realizadas com o material produzido (Quadro 5.1), é possível observar que praticamente todos os temas tratados em Downey (2016) são abordados pelas obras introdutórias com *Processing*. Em destaque em cinza no Quadro 5.2 estão as exceções, temas não apresentados em outras obras, são eles: o tratamento mais detalhado de algoritmos de ordenação (csc.alg.ord) e a análise de algoritmos com notação “O” (csc.alg.bon), ambos temas de ciência da computação (`csc`)

Há também a discussão de Bancos de dados (`dat.dba`), detalhamento de questões de orientação a objetos (`pst.oop`) e sintaxe de comparação (`pst.ope.cmp`), assuntos presentes em Fry e Reas (2007), mas que não aparecem associados a capítulos específicos da obra e, por isso, não ganharam lugar na tabela de identificação de temas na etapa inicial da pesquisa.

Poucos são os temas que não aparecem em Fry e Reas (2007), além dos assuntos encontrados em Downey (2015) apontados a pouco, o que a torna, dentre as analisadas, a obra que cobre a maior variedade de temas.

Temas mais avançados aparecem em Shiffman (2012), que por sua vez se exime de tratar dos assuntos elementares, não apresentando nenhum dos temas destacados na estrutura de capítulos de Fry e Reas (2010), como pode ser visto no Quadro 5.3 A obra assume conhecimento de orientação a objetos (`pst.oop`) e Shiffman (2012) se aprofunda nas questões de simulações (`app.sim`), físicas e de agentes, análogos a entidades biológicas.

Fry e Reas (2010) mesmo só considerando em um primeiro momento os temas principais dos capítulos, sem o detalhamento de subtópicos, apresentam boa parte dos assuntos de Downey (2016), mas em especial deixam claro o imperativo dos assuntos gráficos (`gra`) e geométricos (`mat.geo`) em obras introdutórias para artistas e designers, temas presentes em Fry e Reas (2007; 2010, 2014) e Shiffman (2012), ausentes em Downey (2016).

Quadro 5.1 - Comparação de assuntos entre obras

código	Maeda (2001)	Fry e Reas (2007)	Fry e Reas (2010)	Shiffman (2012)	Downey (2016)
csc.alg.bon					•
csc.alg.ord					•
csc.rcr		•		•	•
dat.dba					•
dat.inp		•			
dat.inp.hid	•	•	•		
dat.inp.kbd	•	•			
dat.inp.mou	•	•			
dat.inp.tim		•			
dat.out.fil		•			•
dat.str		•	•		•
dat.txt		•			•
dat.typ		•	•		•
dat.var	•	•	•		•
dat.var.asg					•
dat.var.sco		•	•		•
dev.api		•			
dev.ide		•	•		
dev.tec		•	•		•
dev.tec.deb		•			•
dev.tec.ske		•			
dev.tec.suc		•			
exa.ele		•			
exa.gui		•			
exa.gui.eve		•			
exa.mga				•	
exa.mob		•			
exa.net	•	•			
exa.oca	•			•	
exa.ocv		•			
exa.pfo		•			
exa.sim		•		•	
exa.sim.bio		•		•	
exa.sim.neu				•	
exa.sim.phy				•	
exa.snd		•			
gra.clr	•	•			
gra.img	•	•	•		
gra.img.pix	•	•			
gra.mov	•	•	•		
gra.shp		•	•		
gra.tra	•	•	•		
gra.tri		•			
gra.tyg		•	•		
gra.vec		•			
mat.cur		•			
mat.geo	•	•	•		
mat.geo.trg		•		•	

Quadro 5.1 - Comparação de assuntos entre obras, continuação

código	Maeda (2001)	Fry e Reas (2007)	Fry e Reas (2010)	Shiffman (2012)	Downey (2016)
mat.geo.vec				•	
mat.geo.ver		•			
mat.ope	•	•			•
mat.rnd	•	•		•	
pst.cnd	•	•			•
pst.fun	•	•	•		•
pst.fun.arg		•			•
pst.fun.def	•	•	•		•
pst.fun.par		•			•
pst.fun.prc	•		•		•
pst.fun.rtn	•		•		•
pst.ite	•	•			•
pst.ite.nes	•	•			
pst.ite.sml		•			
pst.oop		•	•		•
pst.oop.atr					•
pst.oop.inh					•
pst.oop.met					•
pst.ope.cmp					•
pst.tok.com	•	•	•		•

Nota: Os códigos utilizado aqui já se referem aos do tesauro, quando da elaboração inicial, foi usada uma primeira versão da hierarquia.

Fonte: Quadro elaborado pelo autor.

Quadro 5.2 - Assuntos em Downey (2016)

códigos	Maeda (2001)	Fry e Reas (2007)	Fry e Reas (2010)	Shiffman (2012)	Downey (2016)
dat.dba					•
pst.cnd	•	•			•
pst.ite	•	•			•
csc.alg.bon					•
csc.alg.ord					•
dat.str		•	•		•
dat.txt		•			•
dat.typ		•	•		•
dat.var	•	•	•		•
dat.var.asg	•	•	•		•
dat.var.sco		•	•		•
dev.tec		•	•		•
dev.tec.deb		•			•
mat.ope	•	•			•
dat.out.fil		•			•
pst.fun	•	•	•		•
pst.fun.def	•	•	•		•
pst.fun.par		•			•
pst.fun.prc	•		•		•
pst.fun.rtn	•		•		•
pst.oop		•	•		•
pst.oop.atr					•
pst.oop.inh					•
pst.oop.met					•
csc.rcr		•		•	•
pst.fun.arg		•			•
pst.ope.cmp					•
pst.tok.com	•	•	•		•

Fonte: Quadro elaborado pelo autor.

Quadro 5.3 - Assuntos em Fry e Reas (2010)

códigos	Maeda (2001)	Fry e Reas (2007)	Fry e Reas (2010)	Shiffman (2012)	Downey (2016)
dat.inp.hid	•	•	•		
dat.str		•	•		•
dat.typ		•	•		•
dat.var	•	•	•		•
dat.var.asg	•	•	•		•
dat.var.sco		•	•		•
dev.ide		•	•		
dev.tec		•	•		•
gra.img	•	•	•		
gra.mov	•	•	•		
gra.shp		•	•		
gra.tra	•	•	•		
gra.tyg		•	•		
mat.geo	•	•	•		
pst.fun	•	•	•		•
pst.fun.def	•	•	•		•
pst.fun.prc	•		•		•
pst.fun.rtn	•		•		•
pst.oop		•	•		•
pst.tok.com	•	•	•		•

Fonte: Quadro elaborado pelo autor.

5.2 Tesauro

O tesauro de termos produzido é apresentado na forma de uma listagem sistemática de 136 termos e seus códigos únicos no Quadro 5.4. São 7 os termos genéricos no topo da hierarquia, ou termos superiores, sob os quais estão 51 termos mais específicos. Sob estes estão outros 78 termos. Não são contabilizados aqui os termos “não recomendados” (USE, usar no lugar deste, outro termo) e equivalentes em inglês do vocabulário de referência, que aparecem no relatório alfabético, registro completo do tesauro, no Apêndice B.

Quadro 5.4 - Listagem sistemática dos termos

csc	**ciência da computação**		
csc.alg		**algoritmos**	
csc.alg.ana			**análise de algoritmos**
csc.alg.bon			**notação "O"**
csc.alg.ord			**algoritmos de ordenação**
csc.rcr		**recursão**	
css.abt		**abstração**	
dat	**dados**		
dat.dba		**banco de dados**	
dat.inp		**entrada de dados**	
dat.inp.fil			**leitura de arquivos**
dat.inp.gra			**importação de arquivos gráficos**
dat.inp.hid			**dispositivos de interface humana**
dat.inp.kbd			**teclado**
dat.inp.mou		**mouse**	
dat.inp.tim			**tempo**
dat.out		**saída de dados**	
dat.out.fil			**escrita em arquivos**
dat.out.gra			**exportação de arquivos gráficos**
dat.str		**estruturas de dados**	
dat.str.arr			**arranjo**
dat.str.lst			**lista**
dat.str.mtr			**matriz**
dat.str.rcd			**registro**
dat.txt		**texto**	
dat.txt.enc			**codificação de caracteres**
dat.typ		**tipagem**	
dat.typ.boo			**valores booleanos**
dat.typ.con			**conversão de dados**
dat.typ.flo			**números de ponto flutuante**
dat.typ.int			**números inteiros**
dat.var		**variáveis**	
dat.var.asg			**atribuição**
dat.var.sco			**escopo de variáveis**
dev	**engenharia de software**		
dev.lib		**uso de bibliotecas**	
dev.api		**interface de programação de aplicação**	
dev.api.web			**interação com o navegador**
dev.dep		**implantação de software**	
dev.ide		**ambiente de desenvolvimento**	

dev.sys		**sistemas operacionais**	
dev.sys.var			**variáveis de sistema**
dev.tec		**estratégias de desenvolvimento**	
dev.tec.deb		**depuração**	
dev.tec.nam		**convenções de nomenclatura**	
dev.tec.ske			**sketching**
dev.tec.suc			**aproximações sucessivas**
exa	**exemplos de aplicação**		
exa.dat		**ciência de dados**	
exa.dat.viz			**visualização de dados**
exa.ele		**computação física**	
exa.gui		**interfaces gráficas**	
exa.gui.eve			**programação orientada a eventos**
exa.mga		**algoritmos genéticos**	
exa.mob		**aplicativos móveis**	
exa.net		**comunicação em rede**	
exa.oca		**autômatos celulares**	
exa.ocv		**visão computacional**	
exa.pfo		**formas parametrizadas**	
exa.sim		**simulações**	
exa.sim.bio			**simulações biológicas**
exa.sim.neu			**redes neurais**
exa.sim.phy			**simulações físicas**
exa.snd		**áudio**	
exa.snd.inp			**entrada sonora**
exa.snd.out			**saída sonora**
exa.ssf		**formas auto-similares**	
exa.ssf.fra			**fractais**
exa.ssf.lsy			**Sistema-L**
gra	**gráficos**		
gra.atr		**atributos gráficos**	
gra.atr.fll			**preenchimento**
gra.atr.stk			**traço**
gra.clr		**cor**	
gra.clr.spa			**espaços de cor**
gra.dro		**ordem de desenho**	
gra.img		**gráficos raster**	
gra.img.pix			**pixels**
gra.mov		**gráficos em movimento**	
gra.ren		**apresentação**	
gra.ren.gls			**shaders**
gra.ren.off			**buffer de apresentação**
gra.shp		**formas**	
gra.shp.ply			**polilinhas**

gra.tra		**transformações gráficas**	
gra.tra.rot			**rotação**
gra.tra.sca			**escala**
gra.tra.trn			**translação**
gra.tri		**gráficos tridimensionais**	
gra.tri.cam			**visão do observador**
gra.tri.lig			**luzes**
gra.tri.prj			**projeções**
gra.tri.tex			**texturas**
gra.tyg		**tipografia**	
gra.vec		**gráficos vetoriais**	
mat	**matemática**		
mat.cur		**curvas matemáticas**	
mat.geo		**geometria**	
mat.geo.coo			**sistemas de coordenadas**
mat.geo.trg			**trigonometria**
mat.geo.vec			**vetores**
mat.geo.ver			**vértices**
mat.irp		**interpolação**	
mat.irp.lrp			**Interpolação linear**
mat.num		**sistemas de numeração**	
mat.num.bin			**números binários**
mat.num.hex			**números hexadecimais**
mat.ope		**operações matemáticas**	
mat.rnd		**aleatoriedade**	
mat.rnd.dis			**distribuições aleatórias**
mat.rnd.pno		**ruído Perlin**	
pst	**estruturas de programação**		
pst.cnd		**condicionais**	
pst.cnd.alt			**execução alternativa**
pst.cnd.cha			**condicionais encadeadas**
pst.cnd.nes			**condicionais aninhadas**
pst.fun		**funções**	
pst.fun.arg			**passagem de argumentos**
pst.fun.def			**declaração de funções**
pst.fun.par			**parâmetros em funções**
pst.fun.prc			**procedimentos**
pst.fun.rtn			**retorno de funções**
pst.ite		**iteração**	
pst.ite.nes			**laços aninhados**
pst.ite.sml			**laço principal**
pst.oop		**orientação a objetos**	
pst.oop.atr			**atributos de objetos**
pst.oop.inh			**herança**

pst.oop.met	**métodos**	
pst.ope	**operadores**	
pst.ope.cmp		**comparações**
pst.ope.log		**operadores lógicos**
pst.ope.ord		**precedência de operadores**
pst.tok	**símbolos**	
pst.tok.com		**comentários**
pst.tok.res		**palavras reservadas**

Fonte: Quadro elaborado pelo autor.

5.2.1 Categorias (termos superiores)

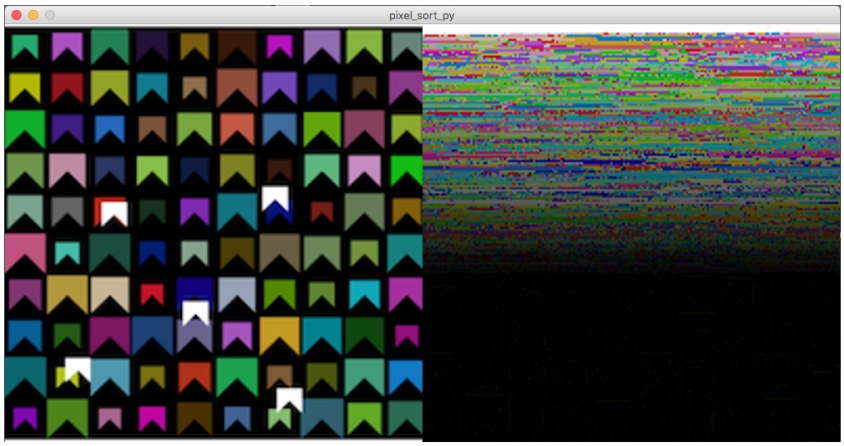
As categorias de temas de programação e os aspectos visuais dos exemplos de aplicação são aqui apresentados de maneira a explicitar algumas de suas inter-relações, no que podem contribuir para a compreensão das especificidades desta coleção de assuntos que serviram de base para o vocabulário controlado produzido.

Ciência da computação (csc)

Os conceitos mais específicos da ciência da computação aparecem bastante dispersos, sem destaque como capítulos nas obras, com a exceção dos mencionados na organização de Downey (2015; 2016). O tema recursão (csc.rcr) em Reas e Fry (2007), curiosamente aparece sob "forma" (*shape)* pelo interesse específico dos autores no resultado gráfico de processos com recursão, semelhantes a fractais e sistemas de Lindenmayer[57] Algoritmos de ordenação podem produzir visualizações interessantes como *pixel sorting* (Figura 5.1), e se enquadram nesta categoria (csc.alg.ord).

Figura 5.1 - Exemplo de pixels da imagem à esquerda ordenados por brilho (à direita)

[57] Como descritos em <http://mathworld.wolfram.com/LindenmayerSystem.html>



Fonte: Exemplo didático produzido pelo autor <https://github.com/villares/py.processing-play/tree/master/pixel_sort_py>.

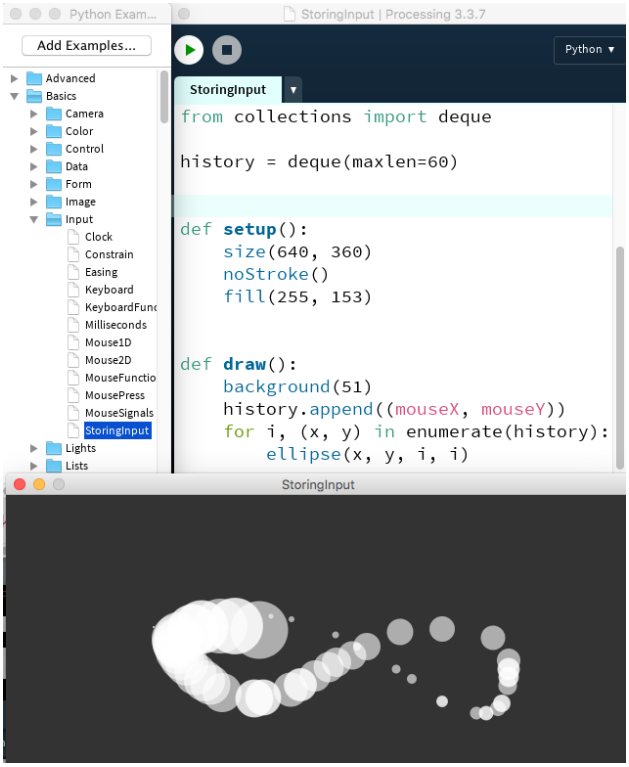
Exemplos de aplicação (exa)

Nesta categoria aparece uma grande diversidade de temas que reflete aplicações e servem de motivação para o aprendizado da programação. São os assuntos que tornam tangíveis os usos da programação, e como pode ser confirmado pelos quadros de comparação, Fry e Reas (2007) e Shiffman (2012) são as obras com maior número de exemplos de aplicações na amostra analisada. Aqui aparecem os Automatos Celulares (exa.oca), as formas auto-similares como fractais (exa.ssf.fra) que permeiam a literatura voltada para os resultados visuais, mas que já utrapassam a questão do vocabulário gráfico na programação apresentado na categoria gráficos (gra).

Dados (dat)

Dados, estruturas de dados e variáveis, são elementos fundamentais da programação. *Processing* possui tipos específicos para a representação de cor, muitos outros assuntos gráficos dependem do armazenamento de coleções de valores, listas ou *arrays* de pixels ou vetores, como por exemplo as aplicações de computação para visualização de dados, que frequentemente se valem de dicionários (*HashMap* em *Java* ou *dict* em *Python*) e outros tipos de mapeamento ou coleções de valores (Figura 5.2).

Figura 5.2 - Exemplo de uso da estrutura de dados *deque* (*double ended qeue*)



Fonte: Exemplos oferecidos pelo IDE de *Processing*, no modo *Python*

Os temas ligados a saída de dados (*output,* `dat.out`) numéricos e textuais tradicionais, em um console de texto ou para um arquivo em disco, aparecem em menor número, uma vez que os resultados dos programas se tornam predominantemente visuais, abarcados na categoria dos gráficos (`gra`), por outro lado, os temas de entrada (*input,* `dat.inp`) aparecem com proeminência, trazendo, além das entradas de dados numéricos ou textuais de arquivos, a interface humana: do teclado e mouse a questões mais gerais de interação pessoa-computador e sensores.

Engenharia de software (dev)

Esta categoria abrange desde as questões das ferramentas e ambientes de desenvolvimento até estratégias de programação, como o *sketching*, que é *modus operandi* característico de programadores em Processing, ou ainda a discussão de procedimentos de depuração (*debugging*).

Gráficos (gra) e Matemática (mat)

Nas obras analisadas, no que se refere aos resultados gráficos da computação em uma tela ou monitor, é possível correlacionar, simplificadamente, três formas de

se pensar em pontos ou posições no espaço, sem ainda entrar na questão das saídas vetoriais, como *plotters* e arquivos para fabricação digital por meio de controle numérico computadorizado (CNC):

1. O ponto ideal da geometria, no plano ou no espaço;
2. Uma posição representada por coordenadas na memória do computador;
3. O pixel na tela que representa um ponto graficamente.

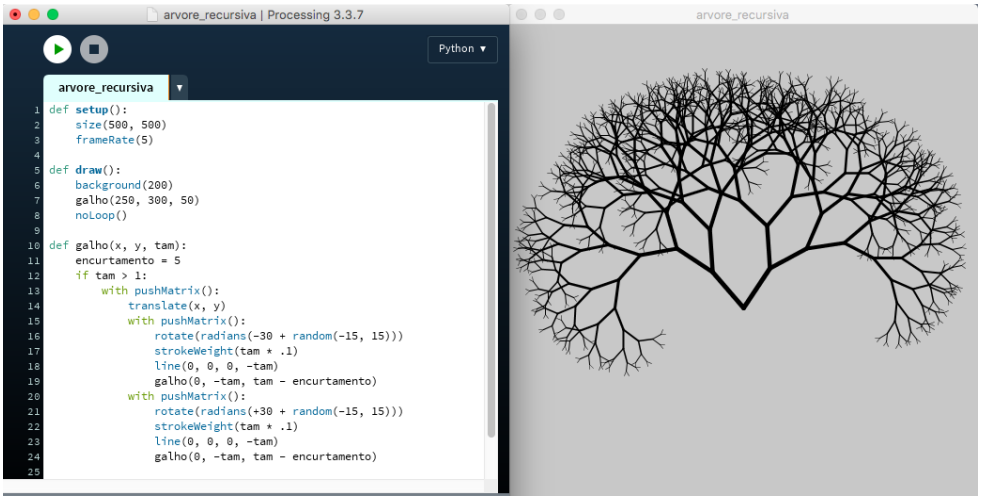
O ponto da geometria, com suas coordenadas, torna-se a posição de um objeto, o vértice de um polígono, ou ainda um ponto de controle de uma curva, podendo a posição ser descrita por um vetor ou por números armazenados na memória. Figuras, pontos, linhas, polígonos ou curvas, podem ser armazenadas como elementos gráficos, ditos vetoriais, mas em uma última etapa são, rasterizadas, isto é, transformadas em representações com pixels; estes têm suas coordenadas discretas, posições que podem ser descritas por números inteiros em uma grade na tela. Partindo das coordenadas dos pontos, idealizadas na geometria, passa-se para números de ponto flutuante (*float*), de precisão limitada, descrevendo vértices de figuras vetoriais na memória do computador e chega-se por fim a pixels em um monitor.

As questões dos pontos e suas coordenadas tornam-se portanto larga gama de assuntos que surgem sob diferentes abordagens, da geometria e manipulação do sistema de coordenadas, aliadas a outras transformações gráficas (`gra.tra`), da matemática vetorial (`mat.geo.vec`), dos gráficos vetoriais (`gra.vec`) ou ainda a abordagem da manipulação dos pixels (`gra.img.pix`).

A apresentação das transformações do sistema de coordenadas, operações de translação, rotação e escala (no *Processing* acionados com `translate()`, `rotate()` e `scale()`), que em geral são feitas no contexto de uma "pilha" que armazena e depois restaura estados do sistema de coordenadas (no *Processing* com os comandos `pushMatrix()` e `popMatrix()`), pode ser feita em conjunto com a discussão de questões de modularidade, da programação estruturada e encapsulamento (com ou sem orientação a objetos) em exemplos com resultado

visual. Por exemplo, cada galho de uma árvore desenhada com recursão é desenhado, pela função que o define, "reto" (alinhado a um eixo do sistema de cordenadas) e com início na origem, e esta é deslocada e girada sucessivamente a cada invocação da função (Figura 5.3).

Figura 5.3 - Exemplo de recursão com transformação do sistema de coordenadas



Fonte: Exemplo didático produzido pelo autor
<https://github.com/villares/py.processing-play/tree/master/arvore_recursiva>

Estruturas de programação (pst)

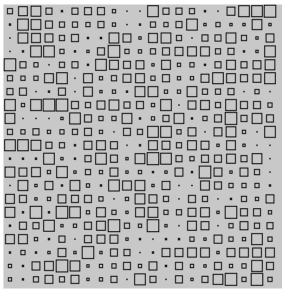
A categoria de temas "estrutura", ou como adotado "estruturas de programação" (pst), inclui as questões de sintaxe que precisam ser tratadas no contexto de uma linguagem específica. Na organização de Reas e Fry (2007), ao usarem o termo abrangente *structure* como categoria transversal aos capítulos do livro, incluiram na categoria o controle do fluxo de execução, que outros autores separam, e nela a abordam questões de símbolos (*tokens*) e sintaxe, que na taxonomia proposta, permaneceram da mesma maneira.

Foram inclusos nesta categoria: a discussão da orientação a objetos (pst.oop); funções e seus subtemas (pst.fun); o tratamento do controle do fluxo de execução, como condicionais (pst.con) e iteração (pst.ite).

Um tema que integra de maneira exemplar os aspectos visuais e de programação são as grades. Como a própria grade de pixels da tela, são utilizadas em diversas aplicações, de elementos gráficos à organização do espaço (de tabelas a pilotis) e que podem ser obtidas através de laços aninhados (Figura 5.4), também conhecidos como laços encaixados, em inglês *nested loops* (`pst.ite.nes`). O

tema é discutido em Maeda (2001), em Fry e Reas (2007; 2014), e também em Shifman (2012), onde são apresentados laços aninhados para a construção de uma grade para Autômatos Celulares (exe.oca), mas neste último caso o assunto não aparece na extração de temas por não ter proeminência na estrutura de capítulos apresentada por Shiffman (2012).

Figura 5.4 - Exemplo de grade com laços aninhados



```
spac_size = int(width / (grid_elem + 1))
for x in range(spac_size / 2, width, spac_size):
    for y in range(spac_size / 2, width, spac_size):
        # sorteia um tamanho (se o rand_size > 0)
        square_size = elem_size + rand_size * random(-1, 1)
        rect(x + rand_posi * random(-1, 1),  # desenha um quadrado
             y + rand_posi * random(-1, 1),
             square_size,
             square_size)
```

Fonte: Exemplo produzido pelo autor <https://github.com/villares/sketch-a-day/tree/master/s097>

5.2.2 Demais termos

A limitação adotada de 3 níveis hierárquicos para os termos, descrita em 4.3.2, levou ao reposicionamento de alguns temas que poderiam originalmente ser consideras em hierarquias distintas. Por exemplo, entrada por meio de teclado (dat.inp.kbd) e mouse (dat.inp.mou) que poderiam ser considerados termos específicos de dispositivos de interface humana (dat.inp.hid), foram promovidos como termo específico sob entrada de dados (dat.inp). Ou ainda em outro exemplo, o assunto da notação “O” (csc.alg.bon) poderia estar sob análise de algoritmos (csc.alg.ana) mas termina por ficar ao lado deste.

Outras acomodações foram feitas devido à proeminência de certos tópicos de caráter mais abrangente, como Texto (dat.txt), que acaba por absorver as questões que poderiam estar sob *Strings* em uma discussão de tipagem de dados (dat.typ). Da mesma maneira foi tratamento do assunto cor (gra.clr) que poderia estar sob atributos gráficos (gra.atr) mas teve sua posição elevada.

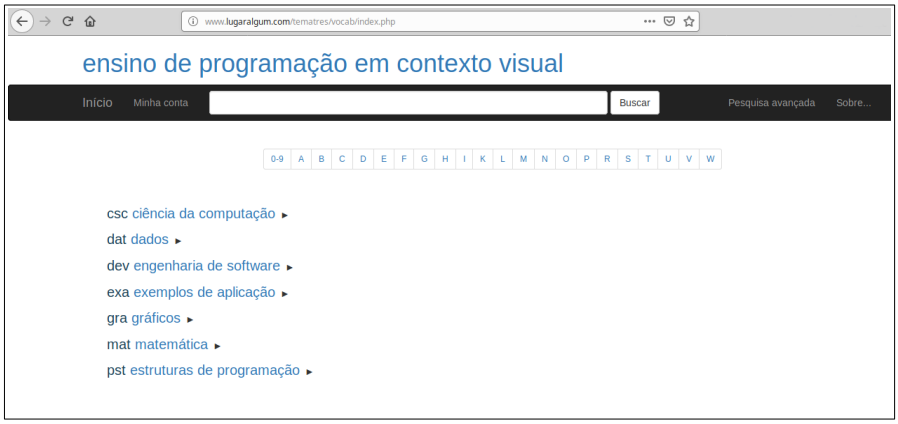
Uma versão do tesauro, registrado no Apêndice B, está disponível para consulta online[58] (Fiigura 5.5), podendo ser consultado em um navegador, com busca de termos e navegação por ligações entre os termos, ou ainda, ser interrogado pela

[58] No endereço <https://www.lugaralgum.com/tematres/>

Application Programming Interface (API) do servidor de vocabulários *TemaTres* no qual está hospedado, cuja resposta é dada nos formatos *JSON* ou *XML*.

A comunicação pela API permite a integração do vocabulário com formulários de outras aplicações, como ferramentas de busca ou repositórios de coleções.

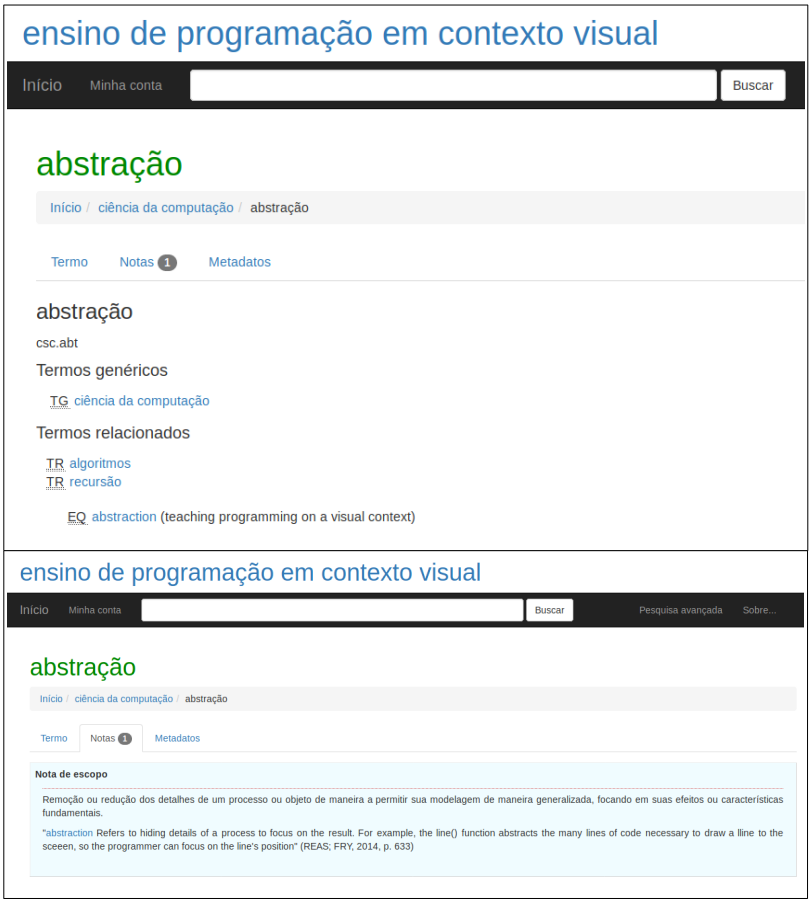
Figura 5.5 – Tela inicial do tesauro no *TemaTres* online



Fonte: Captura realizada pelo autor.

TemaTres também permite a exportação em diversos formatos para intercâmbio de vocabulários controlados e mapas de tópicos, entre eles *Skos-Core, TopiMapr* (XTM), *IMS Vocabulary Definition Exchange* (VDE). Os termos propostos possuem campos indicando hierarquia e relacionamento com outros termos, apontam para o termo equivalente em inglês e em alguns casos possuem uma aba com nota de escopo, como exemplificado na Figura 5.6.

Figura 5.6 – Exemplo de termo no tesauro no *TemaTres* online

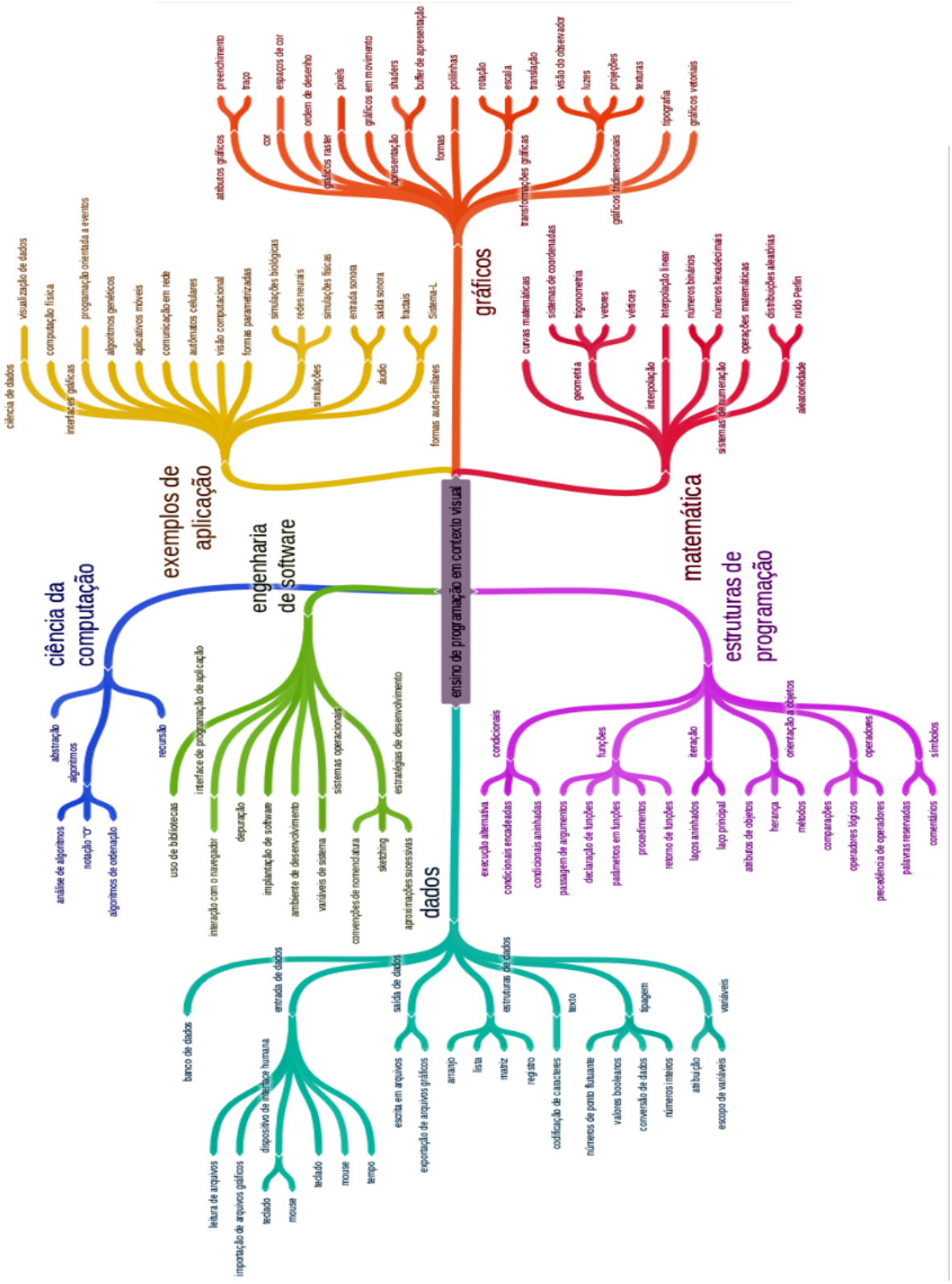


Fonte: Capturas realizadas pelo autor.

A exportação da coleção de termos permite o uso de outras ferramentas de visualização. Desta maneira, a taxonomia de termos pode ser consultada, por exemplo, por meio da ferramenta online *Coggle*[59], para edição e compartilhamento de mapas mentais e mapa de tópicos (Figura 5.7).

[59] Ferramenta em <coggle.it>, mapa de tópicos disponível em: <https://coggle.it/diagram/XCqHK58wsD6Q7T4a/t/ensino-de-programação-em-contexto-visual>

Figura 5.7 - Mapa de tópicos produzido com a ferramenta *Coggle* a partir do tesauro



Fonte: Elaborado pelo autor, disponível em <https://coggle.it/diagram/XCqHK58wsD6Q7T4a/t/ensino-de-programação-em-contexto-visual>

5.3 Exemplos didáticos classificados (códigos-exemplo)

Os 249 exemplos de código fonte, presentes no Processing IDE com modo Python e acessíveis conforme descrito na seção 3.2, foram analisados, e a classificação completa é apresentada no Apêndice C.

5.3.1 Coleção organizada para consulta

Dos 136 termos presentes no tesauro, 85 foram utilizados na classificação dos exemplos, que adotou o uso de até três termos por registro.

Pode-se observar ainda, no Quadro 5.5, que 10 termos foram indiretamente utilizados, uma vez que são conceitos gerais de termos específicos utilizados, incluindo os termos superiores da hierarquia.

É possível perceber também um grande número de exemplos identificados com os assuntos gráficos , sendo os termos mais utilizados sob esta categoria:

- gráficos tridimensionais (gra.tri);
- gráficos em movimento (gra.mov); e
- gráficos raster (gra.img).

Quadro 5.5 – Termos utilizados direta ou indiretamente na classificação

código	Frequência de utilização direta do termo na classificação	Termo ou termo específico deste termo genérico utilizado	código	Frequência de utilização direta do termo na classificação	Termo ou termo específico deste termo genérico utilizado	código	Frequência de utilização direta do termo na classificação	Termo ou termo específico deste termo genérico utilizado
csc		•	**exa**	**1**	•	**mat**	**1**	•
csc.alg			exa.dat		•	mat.cur	**6**	•
csc.alg.ana			exa.dat.viz	**1**	•	mat.geo	**8**	•
csc.alg.bon			exa.ele	**1**	•	mat.geo.coo	**3**	•
csc.alg.ord			exa.gui	**6**	•	mat.geo.trg	**17**	•
csc.rcr	**2**	•	exa.gui.eve	**2**	•	mat.geo.vec	**6**	•
css.abt			exa.mga			mat.geo.ver		
dat		•	exa.mob			mat.irp	**1**	•
dat.dba			exa.net			mat.irp.lrp	**2**	•
dat.inp		•	exa.oca	**3**	•	mat.num		
dat.inp.fil	**3**	•	exa.ocv	**14**	•	mat.num.bin		
dat.inp.gra	**1**	•	exa.pfo			mat.num.hex		
dat.inp.hid	**1**	•	exa.sim		•	mat.ope	**2**	•
dat.inp.kbd	**3**	•	exa.sim.bio	**1**	•	mat.rnd	**7**	•
dat.inp.mou	**19**	•	exa.sim.neu			mat.rnd.dis	**1**	•
dat.inp.tim	**3**	•	exa.sim.phy	**19**	•	mat.rnd.pno	**5**	•
dat.out		•	exa.snd	**2**	•	pst	**1**	•
dat.out.fil	**1**	•	exa.snd.inp			pst.cnd	**6**	•
dat.out.gra	**1**	•	exa.snd.out			pst.cnd.alt		
dat.str	**10**	•	exa.ssf	**1**	•	pst.cnd.cha	**1**	•
dat.str.arr			exa.ssf.fra	**2**	•	pst.cnd.nes		
dat.str.lst	**2**	•	exa.ssf.lsy	**3**	•	pst.fun	**2**	•
dat.str.mtr	**1**	•	**gra**	**8**	•	pst.fun.arg	**1**	•
dat.str.rcd			gra.atr	**1**	•	pst.fun.def	**1**	•
dat.txt	**7**	•	gra.atr.fll			pst.fun.par		
dat.txt.enc			gra.atr.stk			pst.fun.prc		
dat.typ	**1**	•	gra.clr	**10**	•	pst.fun.rtn		
dat.typ.boo	**1**	•	gra.clr.spa			pst.ite	**4**	•
dat.typ.con	**2**	•	gra.dro			pst.ite.nes	**2**	•
dat.typ.flo	**1**	•	gra.img	**24**	•	pst.ite.sml	**5**	•
dat.typ.int		•	gra.img.pix	**10**	•	pst.oop	**15**	•
dat.var	**5**	•	gra.mov	**25**	•	pst.oop.atr		
dat.var.asg	**1**	•	gra.ren		•	pst.oop.inh	**1**	•
dat.var.sco	**3**	•	gra.ren.gls	**14**	•	pst.oop.met	**1**	
dev		•	gra.ren.off			pst.ope		•
dev.lib	**22**	•	gra.shp	**15**	•	pst.ope.cmp		
dev.api		•	gra.shp.ply	**4**	•	pst.ope.log	**1**	•
dev.api.web	**1**	•	gra.tra	**3**	•	pst.ope.ord	**1**	•
dev.dep			gra.tra.rot	**4**	•	pst.tok	**1**	•
dev.ide			gra.tra.sca	**2**	•	pst.tok.com	**1**	•
dev.sys	**1**	•	gra.tra.trn	**1**	•	pst.tok.res		
dev.sys.var	**1**	•	gra.tri	**28**	•			
dev.tec			gra.tri.cam	**3**	•			
dev.tec.deb			gra.tri.lig	**6**	•			
dev.tec.nam			gra.tri.prj	**2**	•			
dev.tec.ske			gra.tri.tex	**6**	•			
dev.tec.suc			gra.tyg	**4**	•			
			gra.vec	**7**	•			

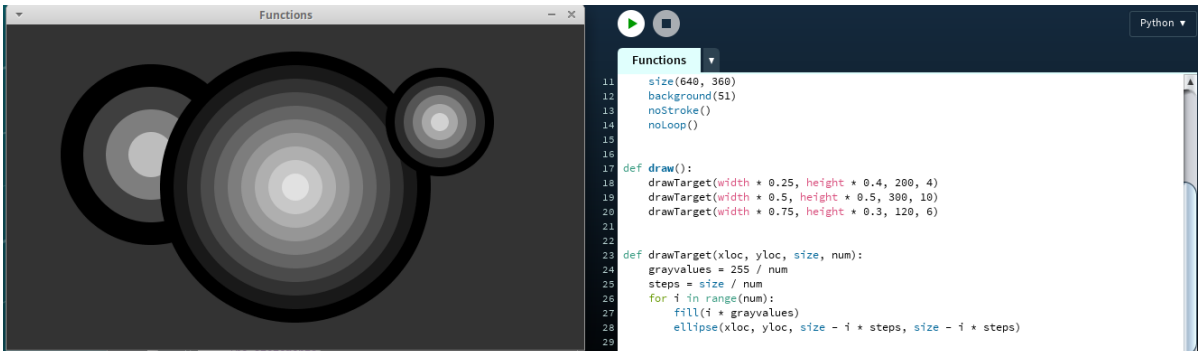
Total de termos: 136
Usados diretamente: 85
Usados direta ou indiretamente: 95

Fonte: Elaborado pelo autor.

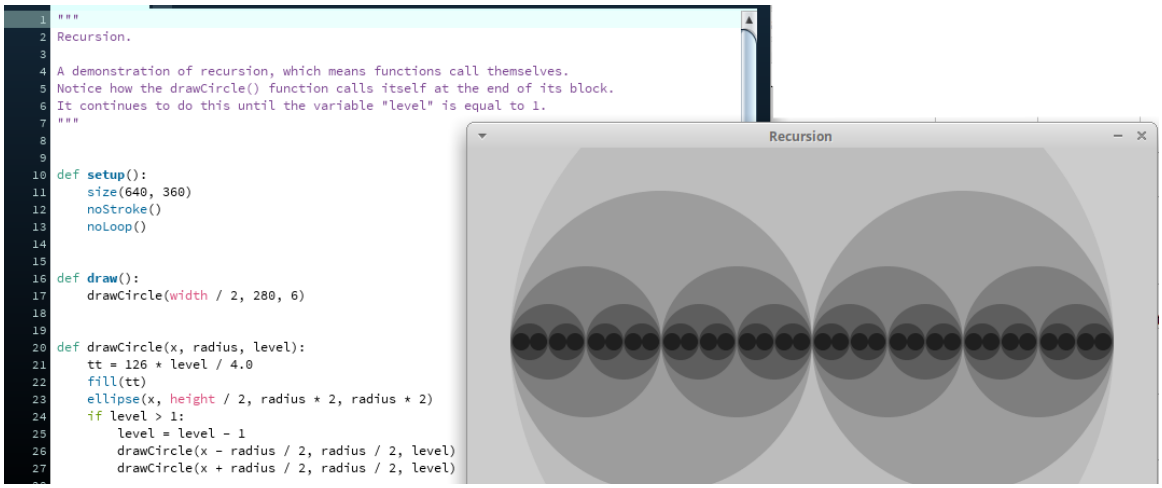
5.3.2 Amostra de exemplares da coleção

A coleção completa de registros dos exemplos classificados está disponível online[60] para navegação e buscas com o software *Omeka* (apresentado em 4.4.3). São abaixo apresentados a título de ilustração da coleção classificada, dois exemplares, na Figura 5.8, com imagens de referência, e seus campos Nome, Identificador, Assunto e Descrição.

Figura 5.8 – Illustração do registro de códigos-exemplo classificados



Nome: **`Functions`**	Identificador: **Basics>Structure>Functions**
Assunto: **declaração de funções (pst.fun.def);** **passagem de argumentos (pst.fun.arg);** **iteração (pst.ite)**	Descrição: **Exemplo elementar de declaração e chamada de uma função drawTarget() que desenha círculos concêntricos.**



Nome: **`Recursion`**	Identificador: **Basics>Structure>Recursion**
Assunto: **recursão (csc.rcr);** **funções (pst.fun.def)**	Descrição: **Exemplo mínimo de recursão com a definição de uma função que desenha círculos e esta recursivamente desenha círculos mais internos.**

Os resultados produzidos estão também registrados nos apêndices B e C.

[60]Podendo ser consultada em <http://lugaralgum.com/omeka-c/>.

6. Conclusão

A amostra das obras permitiu observar recorrência e sobreposição de temas tratados. Além do pressuposto tratamento de um vocabulário que permite resultados gráficos, os temas de simulação física e analogia biológica atravessam mais de 20 anos da literatura consultada (MITCHEL, LIGGET E KVAN, 1987; MAEDA, 2001; SHIFFMAN, 2012), são assuntos que tornam tangível o potencial da simulação e o poder multiplicador da geração de formas por meio de ferramentas computacionais que interessam ao público que se pretende atingir.

6.1 Taxonomia e tesauro

Comparado o vocabulário produzido com os conceitos levantados em Luxton-Reily (2017) é possível afirmar que há relação coerente das ideias fundamentais para o ensino de programação, o que pode ser confirmado de maneira simplificada mapeando os 20 conceitos com 4 ou mais fontes, apresentados no início deste trabalho.

Destes, 14 termos tem correspondência quase direta, como se verifica no Quadro 6.1, e quanto aos 6 outros, 4 são relacionados à orientação a objetos, que no tesauro proposto tem este assunto subdividido de maneira diferente, e poderiam sem prejuízo ser abarcados pelo termo geral orientação a objetos (pst.oop).

Restam o termo “Normatização de estilos de programação”, que poderia ser tratado em engenharia de software (dev), da mesma forma como já nesta categoria há “convenções de nomenclatura” (dev.tec.nam), e, por fim “leitura de código”, que é um conceito da lista de Luxton-Reily (2017) sob a categoria “Processo de programação“ na organização dos autores.

Leitura de código é uma habilidade técnica a ser desenvolvida pelo programador, e assim como depuração (dev.tec) não foi encontrada na coleção de códigos-exemplo analisada um exemplo satisfatório, mas poderia estar ao lado de “estilos de programação” (termo já adotado) ou ainda “legibilidade” (um potencial acréscimo).

Quadro 6.1 – Comparação de 20 termos em Luxton-Reilly (2017) e o tesauro produzido.

Lista de 20 conceitos com mais fontes em Luxton-Reilly (2017)	Mapeia a um termo do tesauro?	Poderia ser incorporado sob qual termo?
Variáveis	dat.var	
Atribuição	dat.var.asg	
Expressões	pst.ope	
Escopo de variáveis	dat.var.sco	
Tipagem	dat.typ	
Strings	dat.txt	
Etrada/saída de dados	dat.out; dat.inp	
Leitura e escrita de arquivos	dat.out.fil; dat.inp.fil	
Objetos e instâncias	**não**	pst.oop
Classes	**não**	pst.oop
Encapsulamento	**não**	pst.oop
Herança	pst.oop.inh	
Polimorfismo	**não**	pst.oop
Normatização de estilo de prog.	**não**	dev.tec
Leitura de código	**não**	dev.tec
Arrays	dat.str.arr	
Condicionais	pst.cnd	
Laços de iteração	pst.ite	
Funções, métodos e procedimentos	pst.fun	
Recursão	csc.rcr	

Fonte: quadro elaborado pelo autor.

6.2 Exemplos classificados

Um resultado não antecipado foi a identificação de tópicos com maior cobertura ou menor cobertura entre os exemplos. Termos sem exemplos na classificação, ou com poucos exemplos, levam a crer que mereceriam ter exemplos criados especificamente para sua melhor comunicação.

A aplicação do tesauro resultou em material de consulta e referência, para uso didático, funcional, apesar do caráter intrinsecamente subjetivo do processo de classificação dos exemplos. O material pode ter ainda sua utilidade ampliada com o compartilhamento em ferramenta online, como demonstrado com a software de gestão de repositórios *Omeka*.

É possível imaginar a ampliação deste material, tanto pelo enriquecimento do vocabulário (ampliação do tesauro) como pelo acréscimo de exemplos (aplicação da classificação a outros materiais didáticos), sendo inclusive possível a organização de exemplos de outras ferramentas de programação. Os exemplos

do *Processing Java Mode* poderiam ser, devido a sua similaridade, rapidamente classificados com o mesmo vocabulário e integrados com a coleção online.

A integração do vocabulário em ferramentas como o *TemaTres* e o *Omeka*, tornam o refinamento do vocabulário e organização do repositório de exemplos usando o tesauro desenvolvido mais acessível, e, se espera, facilitará a ampliação do material.

Das indagações suscitadas no desenrolar da pesquisa, muitas se beneficiariam de uma abordagem experimental com recursos e prazos maiores, munida já de uma estrutura de temas mas indo além da análise da literatura:

- Seria relevante buscar caraterizar e diferenciar as peculiaridades dos casos de uso de alunos ou estudantes e de instrutores ou professores na busca por exemplos didáticos;
- Uma classificação coletiva dos exemplos (*crowd-sourcing*) teria potencial para reduzir o viés na classificação dos exemplos registrados e ampliar mais rapidamente as coleções de exemplos, as tornando mais abrangentes e acessíveis;
- Seria possível introduzir uma análise do comprimento do código e alguma métrica de complexidade dos exemplos. Talvez fosse possível responder questões sobre o nível de conhecimento prévio para compreensão dos assuntos e exemplos, ou validar uma classificação por complexidade.

Todas estas possíveis direções de trabalho seriam auxiliadas pela existência de um vocabulário, taxonomia de temas, e coleção preliminar organizada de exemplos, produzida por este trabalho.

Referências

ALENCAR, Viviane; CELANI, Gabriela. The Art of Computer Graphics Programming: Translating Pioneer Programs. **Blucher Design Proceedings**, São Paulo, v. 1, n. 7, p. 500–504, dez. 2014. Disponível em: <http://pdf.blucher.com.br.s3-sa-east-1.amazonaws.com/designproceedings/sigradi2013/0096.pdf>. Acesso em: 29 dez. 2018.

AMIRI, Faramarz. Programming as Design: The Role of Programming in Interactive Media Curriculum in Art and Design. **International Journal of Art & Design Education**, v. 30, n. 2, p. 200–210, June 2011. Disponível em: <https://onlinelibrary.wiley.com/doi/epdf/10.1111/j.1476-8070.2011.01680.x>. Acesso em: 29 dez. 2018.

ARMSTRONG, Deborah J. The quarks of object-oriented development. **Communications of the ACM**, New York, v. 49, n. 2, p. 123–128, Feb. 2006. Disponível em: <https://www.researchgate.net/publication/220425366_The_quarks_of_object-oriented_development>. Acesso em: 3 jan. 2019.

BONNETT, John. Design By Numbers (John Maeda). **Journal of the Association for History and Computing**, Ann Arbor, v. 3, n. 1, Apr. 2000. Disponível em: <http://hdl.handle.net/2027/spo.3310410.0003.118>. Acesso em: 29 dez. 2018.

BURRY, Mark. **Scripting Cultures:** Architectural Design and Programming. Chichester: Wiley, 2011.

CELANI, Maria Gabriela Caffarena. **CAD criativo**. Rio de Janeiro: Campus, 2003.

_____. Teaching CAD Programming to Architecture Students. **Gestão & Tecnologia de Projetos**, São Carlos, v. 3, n. 2, p. 1–23, nov. 2008. Disponível em: <https://doi.org/10.4237/gtp.v3i2.73>. Acesso em 29 dez. 2018.

CELANI, Maria Gabriela Caffarena; LAZARINI, Kaya. Using CAD for generating architectural form: Reviewing and translating pioneer programs. In: MATHEMATICS & DESIGN INTERNATIONAL CONFERENCE, 5., 2007, Blumenau. Disponível em: <http://www.fec.unicamp.br/~lapac/papers/celani-lazzarini-2007.pdf>. Acesso em: 29 dez. 2018.

COATES, Paul. **Programming.architecture**. Abingdon: Routledge, 2010.

DALE, Nell. Content and Emphasis in CS1. **SIGCSE Bulletim**, New York, v. 37, n. 4, p. 69–73, Dec. 2005.

DOWNEY, Allen B. **Think Python**: How to Think Like a Computer Scientist. 2nd ed. Sebastopol: O'Reilly, 2015.

_____. **Pense em Python**: Pense como um cientista da computação. Tradução de Sheila Gomes. São Paulo, Novatec, 2016.

FRY, Benjamin Jothan. **Computational Information Design.** 2014. Tese (Doctor of Philosophy) – School of Architecture and Planning, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2004. Disponível em: <https://benfry.com/phd/dissertation-110323c.pdf>. Acesso em: 29 dez. 2018.

FULLER, Matthew (Ed.). **Software Studies**: A Lexicon. Cambridge, Massachusetts: The MIT Press, 2008.

GARSHOL, Lars Marius. Metadata? Thesauri? Taxonomies? Topic Maps! Making Sense of it all. **Journal of Information Science**, v. 30, n. 4, p. 378–391, Aug. 2004. Disponível em: <https://pdfs.semanticscholar.org/9905/a3cc8d6f50689953f30d6bd00c5e086ff336.pdf?_ga=2.5471539.1893636024.1546109508-819530073.1546109508>. Acesso em: 29 dez. 2018.

GAVRILOVA, Tatiana; FARZAN, Rosta; BRUSILOVSKY, Peter. **One Practical Algorithm of Creating Teaching Ontologies**. 2005. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=2B7CA01A9042A3DDD2DE71ACC7B54433?doi=10.1.1.93.3783&rep=rep1&type=pdf>. Acesso em: 29 dez. 2018.

GLASSNER, Andrew. **Processing for Visual Artists:** How to Create Expressive Images and Interactive Art. Natick, A K Peters, 2010.

GUO, Philip. **Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities.** 2014. Disponível em: <https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext>. Acesso em: 12 set 2017.

GUZDIAL, Mark. **Predictions on Future CS1 Languages Computing Education Research Blog**. 2011. Disponível em: <https://computinged.wordpress.com/2011/01/24/predictions-on-future-cs1-languages/>. Acesso em: 11 ago. 2018.

IME-USP**. Aulas de Introdução à Computação com Python**: Edição Interativa. 2015. Disponível em: <https://panda.ime.usp.br/aulasPython/static/aulasPython/introducao.html>. Acesso em: 1 out. 2017.

HERTZ, Matthew. What Do “CS1” and “CS2” Mean?: Investigating Differences in the Early Courses. In: ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 41., 2010, Milwaukee. **Proceedings...** New York: ACM, 2010, p. 199–

203. Disponível em: <http://doi.acm.org/10.1145/1734263.1734335>. Acesso em: 29 dez 2018.

HOGGET, Reuben. **1969 – The Logo Turtle – Seymour Papert et al (Sth African/American).** 2010. Disponível em: <https://web.archive.org/web/20180508042337/http://cyberneticzoo.com/cyberneticanimals/1969-the-logo-turtle-seymour-papert-marvin-minsky-et-al-american/>. Acesso em: 9 jan. 2018.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO 7185:1990: Information technology — Programming languages — Pascal.** Geneva, 1990. Disponível em: <https://www.iso.org/standard/13802.html>. Acesso em: 11 ago. 2018.

JOINT TASK FORCE ON COMPUTING CURRICULA; ASSOCIATION FOR COMPUTING MACHINERY; IEEE COMPUTER SOCIETY. **Computer Science Curricula 2013**: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. New York: ACM, 2013.

KRUGLYK, Vladyslav; LVOV, Michael. Choosing the First Educational Programming Language. In: ICTERI – INTERNATIONAL CONFERENCE ON ICT IN EDUCATION, RESEARCH AND INDUSTRIAL APPLICATIONS, 8., 2012, Kherson, Ukraine. **Proceedings...** Aachen: CEUR Workshop Proceedings, 2012, p. 188–198. Disponível em: <http://ceur-ws.org/Vol-848/ICTERI-2012-CEUR-WS-Volume.pdf>. Acesso em: 29 dez. 2018.

LIBERAL ARTS COMPUTER SCIENCE CONSORTIUM. A 2007 model curriculum for a liberal arts degree in computer science. **Journal on Educational Resources in Computing**, New York, v. 7, n. 2, June 2007. Disponível em: <http://doi.acm.org/10.1145/1240200.1240202>. Acesso em: 29 out 2018.

LUCAS, Elaine Rosangela de Oliveira; CORRÊA, Elisa Cristina Delfini; EGGERT-STEINDEL, Gisela. **As contribuições de Ranganathan para a Biblioteconomia:** reflexões e desafios. São Paulo, FEBAB, 2016.

LUXTON-REILLY, Andrew et al. Developing Assessments to Determine Mastery of Programming Fundamentals. In: THE 2017 ITICSE CONFERENCE ON WORKING GROUP REPORTS, 2017, Bologna. **Proceedings...** New York: ACM, 2017, p. 47–69. Disponível em: <http://dl.acm.org/citation.cfm?doid= 3174781.3174784>. Acesso em: 21 set 2018.

MAEDA, John; **Design by Numbers**. Prefácio de Paola Antonelli. Cambridge, Massachusetts: The MIT Press, 1999.

MANOVICH, Lev. **Software Takes Command**. New York: Bloomsbury, 2013.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY. **Design By Numbers**. 2001. Disponível em: <http://dbn.media.mit.edu/whatisdbn.html>. Acesso em: 29 dez. 2018.

MELLO, Patricia Oakim Bandeira de. **Arte e programação na linguagem Processing**. 2015. Dissertação (Mestrado em Mídias Digitais) – Pontifícia Universidade Católica de São Paulo, São Paulo, 2015. Disponível em: <https://tede2.pucsp.br/handle/handle/18199>. Acesso em: 29 dez. 2018.

MILNE, Iain; ROWE, Glenn. Difficulties in Learning and Teaching Programming—Views of Students and Tutors. **Education and Information Technologies**, New York, v. 7, n. 1, p. 55–66, Mar. 2002. Disponível em: <http://www.swisseduc.ch/informatik-didaktik/programmieren-lernen/docs/milne.pdf>. Acesso em: 3 jan. 2019.

MITCHELL, William J.; LIGGETT, Robin S.; KVAN, Thomas. **The Art of Computer Graphics Programming**: A Structured Introduction for Architects and Designers. New York: Van Nostrand Reinhold, 1987.

MONTFORT, Nick. **Exploratory Programming for the Arts and Humanities**. Cambridge, Massachusetts: The MIT Press, 2016.

NAKE, Frieder. The Pioneer of Generative Art: Georg Nees. **Leonardo**, Cambridge, Massachusetts, v. 51, n. 3, p. 277–279, June 2018.

NÓBREGA, Miguel. **Superfície**. 2015. Disponível em: <http://superficie.ink/>. Acesso em: 11 ago. 2018.

PARRISH, Allison; FRY, Ben; REAS, Casey. **Getting Started with Processing.py**: Making Interactive Graphics with Processing's Python Mode. San Francisco: Maker Media, 2016.

PINHEIRO, Lena Vania Ribeiro; FERREZ, Helena Dodd. **Tesauro Brasileiro de Ciência da Informação**. Rio de Janeiro: Instituto Brasileiro de Informação em Ciência e Tecnologia (IBICT), 2014.

PEDRONI, Michela; MEYER, Bertrand. Object-Oriented Modeling of Object-Oriented Concepts. In: ISSEP INTERNATIONAL CONFERENCE ON INFORMATICS IN SECONDARY SCHOOLS – EVOLUTION AND PERSPECTIVES, 4., 2010, Zurich. **Proceedings...** Berlin, Springer, 2010, p. 155–169. Disponível em: <https://www.researchgate.net/publication/221437590_Object-Oriented_Modeling_of_Object-Oriented_Concepts>. Acesso em: 3 jan. 2019.

PROCESSING FOUNDATION. **A Modern Prometheus**: The History of Processing by Casey Reas and Ben Fry. 2018. Disponível em: <https://medium.com/processing-foundation/a-modern-prometheus-59aed94abe85>. Acesso em: 29 dez. 2018.

_____. **Source code for the Processing Core and Development Environment (PDE)**. 2018. Disponível em: <https://github.com/processing/processing>. Acesso em: 29 dez. 2018.

REAS, Casey; FRY, Ben. **Getting Started with Processing**: A Hands-On Introduction to Making Interactive Graphics. Sebastopol: O'Reilly, 2010.

_____. **Processing**: A Programming Handbook for Visual Designers and Artists. Cambridge, Massachusetts: The MIT Press, 2007.

_____. **Processing**: A Programming Handbook for Visual Designers and Artists. 2nd ed. Cambridge, Massachusetts: The MIT Press, 2014.

SANDERS, Kate et al. The Canterbury QuestionBank: Building a Repository of Multiple-choice CS1 and CS2 Questions. In: ITICSE WORKING GROUP REPORTS CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION, 2013, Canterbury. **Proceedings...** New York: ACM, 2013, p. 33–52. Disponível em: <http://doi.acm.org/10.1145/2543882.2543885>. Acesso em: 29 dez 2018.

SCHACHMAN, Toby. Alternative Programming Interfaces for Alternative Programmers. In: ONWARD! ACM INTERNATIONAL SYMPOSIUM ON NEW IDEAS, NEW PARADIGMS, AND REFLECTIONS ON PROGRAMMING AND SOFTWARE, 2012, Tucson. **Proceedings...** New York: ACM, 2012, p. 1–10. Disponível em: <http://doi.acm.org/10.1145/2384592.2384594>. Acesso em: 25 set. 2017.

SCHULTE, Carsten; BENNEDSEN, Jens. What Do Teachers Teach in Introductory Programming? In: SECOND INTERNATIONAL WORKSHOP ON COMPUTING EDUCATION RESEARCH, Canterbury, 2006. **Proceedings...** New York: ACM, 2006, p. 17–28. Disponível em: <http://doi.acm.org/10.1145/1151588.1151593>. Acesso em: 27 fev 2018.

SHIFFMAN, Daniel. **The Nature of Code.** [S.l.]: Daniel Shiffman, 2012. Disponível em: <http://wtf.tw/ref/shiffman.pdf>. Acesso em: 30 dez. 2018.

SHIRKY, Clay. **Ontology is Overrated**: Categories, Links, and Tags. 2005. Disponível em: <http://shirky.com/writings/ontology_overrated.html>. Acesso em: 8 jun. 2018.

SIMON et al. Introductory Programming: Examining the Exams. In: FOURTEENTH AUSTRALASIAN COMPUTING EDUCATION CONFERENCE, 2012, Melbourne. **Proceedings...** Darlinghurst: Australian Computer Society, 2012, p. 61–70. Disponível em: <http://dl.acm.org/citation.cfm?id=2483716.2483724>. Acesso em: 29 dez 2018

SILVER, Mike (Ed.). **Programming Cultures**: Architecture, Art and Science in the Age of Software Development. Chichester: Wiley, 2006.

SOSNOVSKY, Sergey; GAVRILOVA, Tatiana. Development of Educational Ontology for C-Programming. **International Journal "Information, Theories & Applications"**, Sofia, v. 13, n. 4, p. 303–308, 2006. Disponível em: <http://www.foibg.com/ijita/vol13/ijita13-4.pdf>. Acesso em 30 dez. 2018.

TERZIDIS, Kostas. **Algorithmic Architecture**. Abingdon: Routledge, 2006.

_____. **Algorithms for Visual Design Using the Processing Language**. Indianapolis: Wiley, 2009.

TEW, Allison Elliott; GUZDIAL, Mark. Developing a Validated Assessment of Fundamental CS1 Concepts. ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 41., 2010, Milwaukee. **Proceedings...** New York: ACM, 2010, p. 97–101. Disponível em: <http://doi.acm.org/10.1145/1734263.1734297>. Acesso em: 29 dez 2018.

TOLLERVEY, Nicholas H. **Python in Education**: Teach, Learn, Program. Sebastopol: O'Reilly, 2015.

VICTORIA AND ALBERT MUSEUM. **Chance and Control**: Art in the Age of Computers. 2018. Disponível em: <https://www.vam.ac.uk/exhibitions/chance-and-control-art-in-the-age-of-computers>. Acesso em: 11 ago. 2018.

VILLARES, Alexandre B. A.; MOREIRA, Daniel de Carvalho. Python on the Landscape of Programming Tools for Design and Architectural Education. **Blucher Design Proceedings**, Volume 3, 2017, Pages 207-211, ISSN 2318-6968, Disponível em: <http://dx.doi.org/10.1016/sigradi2017-033>. Acesso em: 25/01/2019.

VANDERPLAS, Jake. **A Whirlwind Tour of Python**. Sebastopol: O'Reilly, 2016.

VISNJIC, Filip. **Exhibition**: A curated collection of projects created with Processing. Disponível em: <https://processing.org/exhibition/>. Acesso em: 29 dez. 2018.

Apêndice A - *The Art of Computer Graphics Programming: A Structured Introduction for Architects and Designers*

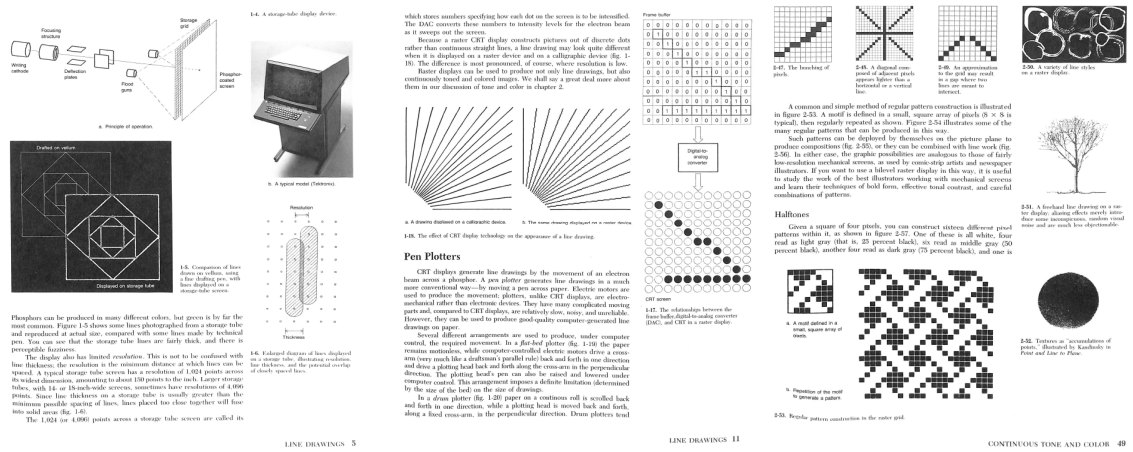
"Nós tratamos a programação de computadores para a produção de formas gráficas que (muito como na composição musical, dramaturgia e coreografia) envolve a distinção entre a especificação e performance do trabalho. Tal distinção não é tradição nas artes gráficas; desenhos e pinturas são normalmente executados diretamente. Mas em computação gráfica, um programa é como a partitura, o computador é como um pianista, e a produção da imagem é de uma performance meticulosa e fiel." (MITCHELL; LIGGETT; KVAN, 1987)

Um panorama inicial do uso de programação em ambiente computacional gráfico é oferecido por Mitchell, Liggett e Kvan (1987), derivada do material desenvolvido originalmente para os cursos de computação gráfica na *Graduate School of Architecture and Urban Planning* da *University of California, Los Angeles* - UCLA e na *Harvard Graduate School of Design*. A obra está dividida em três partes:

1. *Introduction to the Medium:* Descreve os equipamentos, computadores, e como o resultado gráfico produzido se compara às mídias tradicionais.
2. *Elementary Graphics Programs:* Introduz as primeiras estruturas necessárias para se escrever programas linguagem Pascal, e com eles produzir desenhos bidimensionais.
3. *Advanced Techniques:* Apresenta estruturas de dados, transformações do sistema de coordenadas e outros métodos de programação para a produção de "imagens mais ambiciosas", segundo os autores.

Além de uma discussão sobre o funcionamento geral dos computadores, a primeira parte explica especialmente os dispositivos de entrada e saída, suas características, o funcionamento das telas, de que maneira geometrias de desenho vetoriais são convertidas para uma grade de pixels e o efeito que isso tem nos traços e nas imagens (Figura A.1).

FIgura A.1 - Páginas de *Art Of Computer Graphics Programming*

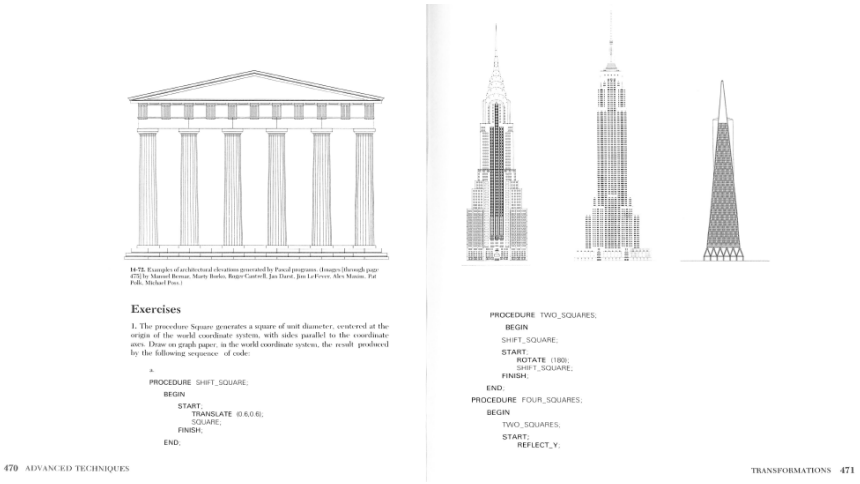


Fonte: MITCHELL; LIGGETT; KVAN,1987, p. 5, 11 e 49.

No prefácio afirmam os autores que "estamos aqui tão preocupados com questões da teoria do projeto (*design theory*) e estética visual quanto com a tecnologia de computadores" e logo em seguida "nosso objetivo central é lhe ensinar como escrever programas de computador concisos, elegantes, bem estruturados para gerar resultados gráficos" (MITCHELL; LIGGETT; KVAN, 1987, p. vii). Para tanto, na segunda parte, é apresentada a linguagem de programação *Pascal* (descrita em 3.2.1).

Na terceira parte do livro (Figura A.2) aparecem as estruturas de dados, *Arrays*, as transformações de sistemas de coordenadas e uma discussão sobre a construção de editores de desenho simples. Por fim, são apresentadas indicações de como o leitor pode seguir adiante em suas explorações da programação gráfica.

Figura A.2 - Páginas de *Art Of Computer Graphics Programming*



16-72. Examples of architectural elevations generated by Pascal programs. (Images [through page 472] by Manuel Bernat, Marty Burks, Bugsy Cantrell, Jan Darst, Jim LeFever, Alex Massin, Pat Pelli, Michael Pons.)

Exercises

1. The procedure Square generates a square of unit diameter, centered at the origin of the world coordinate system, with sides parallel to the coordinate axes. Draw on graph paper, in the world coordinate system, the result produced by the following sequence of code:

a.

```
PROCEDURE SHIFT_SQUARE;
    BEGIN
        START;
            TRANSLATE (0.6,0.6);
            SQUARE;
        FINISH;
    END;
```

470 ADVANCED TECHNIQUES

```
PROCEDURE TWO_SQUARES;
    BEGIN
        SHIFT_SQUARE;
        START;
            ROTATE (180);
            SHIFT_SQUARE;
        FINISH;
    END;
PROCEDURE FOUR_SQUARES;
    BEGIN
        TWO_SQUARES;
        START;
            REFLECT_Y;
```

TRANSFORMATIONS 471

Fonte: *Art Of Computer Graphics Programming*, 1987, p. 470 - 471.

As implementações das funções gráficas que permitem que os leitores produzam os resultados propostos, usando os ambientes de desenvolvimento *Turbo Pascal* e *Macintosh Pascal* (populares na época), são fornecidas em um apêndice do livro.

Apêndice B – Registro do Tesauro

Listagem dos termos do tesauro, produzida a partir de relatório gerado pelo editor de vocabulários controlados *TemaTres*, de maneira a registrar formalmente este produto da pesquisa. O tesauro também se encontra disponível para buscas e consultas no site indicado abaixo onde pode inclusive ser interrogado por meio da API do *TemaTres*.

URI:
http://www.lugaralgum.com/tematres/vocab/
Gerado em: TemaTres 3.0

3D graphics (EN)
EQ: gráficos tridimensionais

abstração
Código: csc.abt
Nota de escopo: Remoção ou redução dos detalhes de um processo ou objeto de maneira a permitir sua modelagem de maneira generalizada, focando em suas efeitos ou características fundamentais.
"abstraction Refers to hiding details of a process to focus on the result. For example, the line() function abstracts the many lines of code necessary to draw a lline to the sceeen, so the programmer can focus on the line's position" (REAS; FRY, 2014, p. 633)
TG: ciência da computação
TR: algoritmos
TR: recursão
EQ: abstraction (EN)

abstraction (EN)
EQ: abstração

aleatoriedade
Código: mat.rnd
Nota de escopo: Termo para números aleatórios, pseudo-aleatórios e distribuições aleatórias. O números aleatórios utilizados em programação são frequentemente pseudo-aleatórios, isto é são gerados de maneira deterministica mas aparentam aleatoriedade.
UP: aleatório
UP: pseudo-aleatório
TG: matemática
TE: distribuições aleatórias
TE: ruído Perlin
EQ: random (EN)

aleatório
USE: aleatoriedade

algorithm analysis (EN)
EQ: análise de algoritmos

algorithms (EN)
EQ: algoritmos

algoritmos
Código: csc.alg
Nota de escopo: Algoritmo é "[u]m processo geral para resolver uma categoria de problemas."(DOWNEY, 2016). Algoritmos são frequentemente descritos como receitas passo a passo para obter algum tipo de resultado.
TG: ciência da computação
TE: algoritmos de ordenação
TE: análise de algoritmos
TE: notação "O"
TR: abstração
TR: algoritmos genéticos
TR: recursão
EQ: algorithms (EN)

algoritmos de ordenação

Código: csc.alg.ord
TG: algoritmos
TR: algoritmos genéticos
TR: análise de algoritmos
EQ: sorting algorithms (EN)

algoritmos genéticos
Código: exa.mga
TG: exemplos de aplicação
TR: algoritmos
TR: algoritmos de ordenação
TR: aplicativos móveis
TR: áudio
TR: autômatos celulares
TR: computação física
TR: formas parametrizadas
TR: simulações
TR: visão computacional
EQ: genetic algorithms (EN)

alternate execution (EN)
EQ: execução alternativa

ambiente de desenvolvimento
Código: dev.ide
TG: engenharia de software
EQ: development environment (EN)
EQ: IDE (EN)

análise de algoritmos
Código: csc.alg.ana
TG: algoritmos
TR: algoritmos de ordenação
TR: notação "O"
EQ: algorithm analysis (EN)

animações

USE: gráficos em movimento

aplicativos móveis
Código: exa.mob
UP: apps
TG: exemplos de aplicação
TR: algoritmos genéticos
TR: áudio
TR: autômatos celulares
TR: computação física
TR: formas parametrizadas
TR: simulações
TR: visão computacional
EQ: mobile applications (EN)

application examples (EN)
EQ: exemplos de aplicação

application programming interface (EN)
EQ: interface de programação de aplicação

apps

USE: aplicativos móveis

apresentação
Código: gra.ren
TG: gráficos
TE: buffer de apresentação
TE: shaders
EQ: rendering (EN)

aproximações sucessivas
Código: dev.tec.suc
TG: estratégias de desenvolvimento
EQ: incrementalism (EN)

argument passing (EN)
EQ: passagem de argumentos

arranjo
Código: dat.str.arr
Nota de escopo: Arranjo é o nome em português para o conceito de coleções ordenadas e homogêneas de valores (de mesma tipagem) e uma das principais estruturas de dados em muitas linguagens

(array em inglês e na nomenclatura de C e Java).
TG: estruturas de dados
EQ: array (EN)

array (EN)
EQ: arranjo

ASCII
USE: codificação de caracteres

assignment (EN)
EQ: atribuição

atribuição
Código: dat.var.asg
Nota de escopo: "Uma instrução que atribui um valor a uma variável"(DOWNEY, 2016)
TG: variáveis
TR: escopo de variáveis
EQ: assignment (EN)

atributos de objetos
Código: pst.oop.atr
TG: orientação a objetos
EQ: fields (EN)

atributos gráficos
Código: gra.atr
TG: gráficos
TE: preenchimento
TE: traço
TR: cor
EQ: graphic attributes (EN)

áudio
Código: exa.snd
UP: som
TG: exemplos de aplicação
TE: entrada sonora
TE: saída sonora
TR: algoritmos genéticos
TR: aplicativos móveis
TR: computação física
TR: visão computacional
EQ: sound (EN)

autômatos celulares
Código: exa.oca
TG: exemplos de aplicação
TR: algoritmos genéticos
TR: aplicativos móveis
TR: computação física
TR: visão computacional
EQ: cellular autmata (EN)

banco de dados
Código: dat.dba
TG: dados
TR: texto
EQ: database (EN)

big "O" notation

big O notation (EN)
EQ: notação "O"

binary numbers (EN)
EQ: números binários

biological simulation (EN)
EQ: simulações biológicas

bitmap images (EN)
EQ: gráficos raster

bitmaps
USE: gráficos raster

booleans (EN)
EQ: valores booleanos

browser interaction (EN)

EQ: interação com o navegador

buffer de apresentação
Código: gra.ren.off
TG: apresentação
EQ: offscreen buffer (EN)

câmera

USE: visão do observador

camera settings (EN)
EQ: visão do observador

cellular autmata (EN)
EQ: autômatos celulares

chained conditionals (EN)
EQ: condicionais encadeadas

ciência da computação
Código: csc
Nota de escopo: Tópicos fundamentalmente independentes das questões de implementação da ferramenta de programação utilizada, provinientes da ciência da computação, icluindo a noção geral de algoritmos.
TE: abstração
TE: algoritmos
TE: recursão
EQ: computer science concepts (EN)

ciência de dados
Código: exa.dat
TG: exemplos de aplicação
TE: visualização de dados
EQ: data science (EN)

codificação de caracteres
Código: dat.txt.enc
UP: ASCII
UP: Unicode
TG: texto
EQ: text encoding (EN)

colour (EN)
EQ: cor

colour spaces (EN)
EQ: espaços de cor

comentários
Código: pst.tok.com
TG: símbolos
EQ: comments (EN)

comments (EN)
EQ: comentários

comparações
Código: pst.ope.cmp
TG: operadores
EQ: comparison (EN)

comparison (EN)
EQ: comparações

computação física
Código: exa.ele
UP: microcontroladores
TG: exemplos de aplicação
TR: algoritmos genéticos
TR: aplicativos móveis
TR: áudio
TR: autômatos celulares
TR: formas parametrizadas
TR: simulações
TR: visão computacional
EQ: physical computing (EN)

computer science concepts (EN)
EQ: ciência da computação

computer vision (EN)
EQ: visão computacional

comunicação em rede
Código: exa.net
UP: redes
TG: exemplos de aplicação
EQ: networking (EN)

condicionais
Código: pst.cnd
Nota de escopo: Condicionais são instruções que controlam o fluxo de execução, dependendo de alguma condição. A condição é uma expressão booleana que determina qual ramo deve ser executado.(DOWNEY, 2016)
UP: condicional
TG: estruturas de programação
TE: condicionais aninhadas
TE: condicionais encadeadas
TE: execução alternativa
EQ: conditionals (EN)

condicionais aninhadas
Código: pst.cnd.nes
TG: condicionais
EQ: nested conditionals (EN)

condicionais encadeadas
Código: pst.cnd.cha
TG: condicionais
EQ: chained conditionals (EN)

condicional

USE: condicionais

conditionals (EN)
EQ: condicionais

convenções de nomenclatura
Código: dev.tec.nam
TG: estratégias de desenvolvimento
EQ: naming conventions (EN)

conversão de dados
Código: dat.typ.con
TG: tipagem
TR: texto
EQ: type conversion (EN)

coordinate systems (EN)
EQ: sistemas de coordenadas

cor
Código: gra.clr
TG: gráficos
TE: espaços de cor
TR: atributos gráficos
EQ: colour (EN)

curvas matemáticas
Código: mat.cur
TG: matemática
EQ: curves (EN)

curves (EN)
EQ: curvas matemáticas

dados
Código: dat
Nota de escopo: Valores, números ou símbolos, que podem ser transmitidos, armazenados e manipulados por um computador.
TE: banco de dados
TE: entrada de dados
TE: estruturas de dados
TE: saída de dados
TE: texto
TE: tipagem
TE: variáveis
EQ: data (EN)

dados externos

USE: entrada de dados

data (EN)
EQ: dados

data science (EN)
EQ: ciência de dados

data structures (EN)
EQ: estruturas de dados

data visualization (EN)
EQ: visualização de dados

database (EN)
EQ: banco de dados

debugging (EN)
EQ: depuração

declaração de funções
Código: pst.fun.def
TG: funções
EQ: function definition (EN)

deployment (EN)
EQ: implantação de software

depuração
Código: dev.deb
TG: engenharia de software
TR: estratégias de desenvolvimento
TR: sistemas operacionais
EQ: debugging (EN)

desenvolvimento
USE: engenharia de software

development environment (EN)
EQ: ambiente de desenvolvimento

development strategies (EN)
EQ: estratégias de desenvolvimento

dispositivos de interface humana
Código: dat.inp.hid
Nota de escopo: Dispositivos de interface humana incluem, mas não estão limitados a teclado e mouse. Este termo pode ser usado para se referir a outros dispositivos, ou ainda a exemplos de uso de diversos dispositivos simultaneamente, incluindo teclado e mouse.
UP: interface de interação humana
UP: interface humana de interação
TG: entrada de dados
TE: mouse
TE: teclado
TR: leitura de arquivos
EQ: HID (EN)

distribuições aleatórias
Código: mat.rnd.dis
TG: aleatoriedade
EQ: random distributions (EN)

drawing order (EN)
EQ: ordem de desenho

engenharia de software
Código: dev
Nota de escopo: Teoria e prática da programação, ambientes, ferramentas e estratégias de desenvolvimento.
UP: desenvolvimento
TE: ambiente de desenvolvimento
TE: depuração
TE: estratégias de desenvolvimento
TE: implantação de software
TE: interface de programação de aplicação
TE: sistemas operacionais
TE: uso de bibliotecas
EQ: software development (EN)

entrada de dados
Código: dat.inp
UP: dados externos
TG: dados

TE: dispositivos de interface humana
TE: importação de arquivos gráficos
TE: leitura de arquivos
TE: mouse
TE: teclado
TE: tempo
TR: estruturas de dados
TR: saída de dados
TR: texto
TR: tipagem
TR: variáveis
EQ: input (EN)

entrada sonora
Código: exa.snd.inp
TG: áudio
EQ: sound input (EN)

escala
Código: gra.tra.sca
TG: transformações gráficas
EQ: scale (EN)

escopo de variáveis
Código: dat.var.sco
TG: variáveis
TR: atribuição
EQ: variable scope (EN)

escrita em arquivos
Código: dat.out.fil
UP: sáida em arquivos textuais
TG: saída de dados
EQ: write to file (EN)

espaços de cor
Código: gra.clr.spa
TG: cor
EQ: colour spaces (EN)

estratégias de desenvolvimento
Código: dev.tec
TG: engenharia de software
TE: aproximações sucessivas
TE: convenções de nomenclatura
TE: sketching
TR: depuração
TR: implantação de software
TR: interface de programação de aplicação
EQ: development strategies (EN)

estruturas de dados
Código: dat.str
TG: dados
TE: arranjo
TE: lista
TE: matriz
TE: registro
TR: entrada de dados
TR: saída de dados
TR: texto
TR: tipagem
TR: variáveis
EQ: data structures (EN)

estruturas de programação
Código: pst
Nota de escopo: Elementos que compõe as linguagens de programação, regras de sintaxe que governam seu funcionamento e suas relações.
"Syntax refers to the structure of a program and the rules about that structure" (DOWNEY, 2012, p. 3)
UP: sintaxe
TE: condicionais
TE: funções
TE: iteração
TE: operadores
TE: orientação a objetos
TE: símbolos
EQ: programming structures (EN)

event oriented programming (EN)
EQ: programação orientada a eventos

execução alternativa
Código: pst.cnd.alt
TG: condicionais
EQ: alternate execution (EN)

exemplos de aplicação
Código: exa
Nota de escopo: Aplicações de programação utilizadas como exemplo na literatura introdutória utilizada como baste para este vocabulário.
TE: algoritmos genéticos
TE: aplicativos móveis
TE: áudio
TE: autômatos celulares
TE: ciência de dados
TE: computação física
TE: comunicação em rede
TE: formas auto-similares
TE: formas parametrizadas
TE: interfaces gráficas
TE: simulações
TE: visão computacional
EQ: application examples (EN)

exportação de arquivos gráficos
Código: dat.out.gra
TG: saída de dados
EQ: graphic export (EN)

fields (EN)
EQ: atributos de objetos

file reading (EN)
EQ: leitura de arquivos

fill (EN)
EQ: preenchimento

floating point numbers (EN)
EQ: números de ponto flutuante

formas
Código: gra.shp
TG: gráficos
TE: polilinhas
EQ: shape (EN)

formas auto-similares
Código: exa.ssf
TG: exemplos de aplicação
TE: fractais
TE: Sistema-L
EQ: self-similarity (EN)

formas parametrizadas
Código: exa.pfo
TG: exemplos de aplicação
TR: algoritmos genéticos
TR: aplicativos móveis
TR: computação física
EQ: parametric shapes (EN)

fractais
Código: exa.ssf.fra
TG: formas auto-similares
EQ: fractals (EN)

fractals (EN)
EQ: fractais

função
USE: funções

funções
Código: pst.fun
UP: função
TG: estruturas de programação
TE: declaração de funções
TE: parâmetros em funções
TE: passagem de argumentos
TE: procedimentos
TE: retorno de funções
EQ: functions (EN)

function definition (EN)
EQ: declaração de funções

function parameters (EN)
EQ: parâmetros em funções

function return value (EN)
EQ: retorno de funções

functions (EN)
EQ: funções

genetic algorithms (EN)
EQ: algoritmos genéticos

geometria
Código: mat.geo
TG: matemática
TE: sistemas de coordenadas
TE: trigonometria
TE: vértices
TE: vetores
EQ: geometry (EN)

geometry (EN)
EQ: geometria

gráficos
Código: gra
Nota de escopo: Resultado gráficos ou visuais produzidos e elementos específicos das linguagens que visam produzir estes resultados.
TE: apresentação
TE: atributos gráficos
TE: cor
TE: formas
TE: gráficos em movimento
TE: gráficos raster
TE: gráficos tridimensionais
TE: gráficos vetoriais
TE: ordem de desenho
TE: tipografia
TE: transformações gráficas
EQ: graphics (EN)

gráficos em movimento
Código: gra.mov
UP: animações
TG: gráficos
EQ: motion graphics (EN)

gráficos raster
Código: gra.img
UP: bitmaps
TG: gráficos
TE: pixels
EQ: bitmap images (EN)

gráficos tridimensionais
Código: gra.tri
TG: gráficos
TE: luzes
TE: projeções
TE: texturas
TE: visão do observador
EQ: 3D graphics (EN)

gráficos vetoriais
Código: gra.vec
TG: gráficos
EQ: vector graphics (EN)

graphic attributes (EN)
EQ: atributos gráficos

graphic export (EN)
EQ: exportação de arquivos gráficos

graphic import (EN)
EQ: importação de arquivos gráficos

graphic transformations (EN)
EQ: transformações gráficas

graphics (EN)

EQ: gráficos

GUI (EN)
EQ: interfaces gráficas

herança
Código: pst.oop.inh
TG: orientação a objetos
EQ: inheritance (EN)

hexadecimal numbers (EN)
EQ: números hexadecimais

HID (EN)
EQ: dispositivos de interface humana

IDE (EN)
EQ: ambiente de desenvolvimento

implantação de software
Código: dev.dep
TG: engenharia de software
TR: estratégias de desenvolvimento
TR: sistemas operacionais
EQ: deployment (EN)

importação de arquivos gráficos
Código: dat.inp.gra
TG: entrada de dados
EQ: graphic import (EN)

incrementalism (EN)
EQ: aproximações sucessivas

inheritance (EN)
EQ: herança

input (EN)
EQ: entrada de dados

integers (EN)
EQ: números inteiros

interação com o navegador
Código: dev.api.web
TG: interface de programação de aplicação
EQ: browser interaction (EN)

interface de interação humana
USE: dispositivos de interface humana

interface de programação de aplicação
Código: dev.api
TG: engenharia de software
TE: interação com o navegador
TR: estratégias de desenvolvimento
TR: sistemas operacionais
EQ: application programming interface (EN)

interface humana de interação
USE: dispositivos de interface humana

interfaces gráficas
Código: exa.gui
TG: exemplos de aplicação
TE: programação orientada a eventos
EQ: GUI (EN)

interpolação
Código: mat.irp
TG: matemática
TE: Interpolação linear
EQ: interpolation (EN)

Interpolação linear
Código: mat.irp.lrp
TG: interpolação
EQ: linear interpolation (EN)

interpolation (EN)
EQ: interpolação

iteração

Código: pst.ite

Nota de escopo: "Execução repetida de um grupo de instruções, usando uma chamada da função recursiva ou um loop [(laço)]." (DOWNEY, 2016)

TG: estruturas de programação

TE: laço principal

TE: laços aninhados

EQ: iteration (EN)

iteration (EN)

EQ: iteração

keyboard (EN)

EQ: teclado

L-System (EN)

EQ: Sistema-L

laço principal

Código: pst.ite.sml

Nota de escopo: Muitos programas são estruturados em torno de processos iniciais (setup) seguidos de um laço principal em inglês main loop, o que facilita organizar o redesenho da tela (dandonome ao laço principal de Processing draw()) ou o tratamento de eventos, leitura de sensores ou outras entradas de dados (como o loop(), laço princiopal que os desenvolvedores de Arduino herdaram de Wiring, por sua vez inspirado no Processing) .

UP: loop principal

TG: iteração

EQ: main loop (EN)

laços aninhados

Código: pst.ite.nes

TG: iteração

EQ: nested loops (EN)

leitura de arquivos

Código: dat.inp.fil

TG: entrada de dados

TR: dispositivos de interface humana

EQ: file reading (EN)

lights (EN)

EQ: luzes

linear interpolation (EN)

EQ: Interpolação linear

list (EN)

EQ: lista

lista

Código: dat.str.lst

Nota de escopo: Listas são sequencias ordenadas e mutáveis de valores, estrutura de dados implementadas em diversas linguagens de programação e "um dos tipos integrados mais úteis do Python" (DOWNEY, 2016)

TG: estruturas de dados

EQ: list (EN)

logic operators (EN)

EQ: operadores lógicos

loop principal

USE: laço principal

luzes

Código: gra.tri.lig

TG: gráficos tridimensionais

EQ: lights (EN)

main loop (EN)

EQ: laço principal

matemática

Código: mat

Nota de escopo: Conceitos, técnicas e operações matemáticas no contexto da programação.

TE: aleatoriedade

TE: curvas matemáticas

TE: geometria
TE: interpolação
TE: operações matemáticas
TE: sistemas de numeração
EQ: mathematics (EN)

mathematical operations (EN)
EQ: operações matemáticas

mathematics (EN)
EQ: matemática

matrix (EN)
EQ: matriz

matriz
Código: dat.str.mtr
TG: estruturas de dados
EQ: matrix (EN)

methods (EN)
EQ: métodos

métodos
Código: pst.oop.met
Nota de escopo: Uma função que está, em geral, associada a um objeto ou a uma classe, sendo definida em uma definição de classe.
TG: orientação a objetos
EQ: methods (EN)

microcontroladores

USE: computação física

mobile applications (EN)
EQ: aplicativos móveis

motion graphics (EN)
EQ: gráficos em movimento

mouse
Código: dat.inp.mou
TG: dispositivos de interface humana
TG: entrada de dados
EQ: mouse (EN)

mouse (EN)
EQ: mouse

naming conventions (EN)
EQ: convenções de nomenclatura

nested conditionals (EN)
EQ: condicionais aninhadas

nested loops (EN)
EQ: laços aninhados

networking (EN)
EQ: comunicação em rede

neural networks (EN)
EQ: redes neurais

notação "O"
Código: csc.alg.bon
UP: notação grande-"O"
TG: algoritmos
TR: análise de algoritmos
EQ: big O notation (EN)

notação grande-"O"

USE: notação "O"

number system (EN)
EQ: sistemas de numeração

números binários
Código: mat.num.bin
TG: sistemas de numeração
EQ: binary numbers (EN)

números de ponto flutuante

Código: dat.tyo.flo
TG: tipagem
EQ: floating point numbers (EN)

números hexadecimais
Código: mat.num.hex
TG: sistemas de numeração
EQ: hexadecimal numbers (EN)

números inteiros
Código: dat.typ.int
TG: tipagem
EQ: integers (EN)

object orientation (EN)
EQ: orientação a objetos

offscreen buffer (EN)
EQ: buffer de apresentação

operações matemáticas
Código: mat.ope
TG: matemática
EQ: mathematical operations (EN)

operadores
Código: pst.ope
TG: estruturas de programação
TE: comparações
TE: operadores lógicos
TE: precedência de operadores
EQ: operators (EN)

operadores lógicos
Código: pst.ope.log
TG: operadores
EQ: logic operators (EN)

operating system (EN)
EQ: sistemas operacionais

operator precedence (EN)
EQ: precedência de operadores

operators (EN)
EQ: operadores

ordem de desenho
Código: gra.dro
TG: gráficos
EQ: drawing order (EN)

orientação a objetos
Código: pst.oop
TG: estruturas de programação
TE: atributos de objetos
TE: herança
TE: métodos
EQ: object orientation (EN)

output (EN)
EQ: saída de dados

palavras reservadas
Código: pst.tok,res
TG: símbolos
EQ: reserved keywords (EN)

parametric shapes (EN)
EQ: formas parametrizadas

parâmetros em funções
Código: pst.fun.par
TG: funções
EQ: function parameters (EN)

passagem de argumentos
Código: pst.fun.arg
TG: funções
EQ: argument passing (EN)

Perlin noise (EN)
EQ: ruído Perlin

physical computing (EN)
EQ: computação física

physical simulation (EN)
EQ: simulações físicas

pixels (EN)
EQ: pixels

pixels
Código: gra.imp.pix
TG: gráficos raster
EQ: pixels (EN)

polilinhas
Código: gra.shp.ply
TG: formas
EQ: polylines (EN)

polylines (EN)
EQ: polilinhas

precedência de operadores
Código: pst.ope.ord
TG: operadores
EQ: operator precedence (EN)

preenchimento
Código: gra.atr.fll
TG: atributos gráficos
EQ: fill (EN)

procedimentos
Código: pst.fun.prc
TG: funções
EQ: procedures (EN)

procedures (EN)
EQ: procedimentos

programação orientada a eventos
Código: exa.gui.eve
TG: interfaces gráficas
EQ: event oriented programming (EN)

programming structures (EN)
EQ: estruturas de programação

projeções
Código: gra.tri.prj
TG: gráficos tridimensionais

prototipagem

USE: sketching

pseudo-aleatório

USE: aleatoriedade

random (EN)
EQ: aleatoriedade

random distributions (EN)
EQ: distribuições aleatórias

record (EN)
EQ: registro

recursão
Código: csc.rcr
Nota de escopo: Recursão ou recursividade pode definida como "[o] processo de chamar a função que está sendo executada no momento"(DOWNEY, 2016)
UP: recursividade
TG: ciência da computação
TR: abstração
TR: algoritmos
EQ: recursion (EN)

recursion (EN)
EQ: recursão

recursividade

USE: recursão

redes

USE: comunicação em rede

redes neurais
Código: exa.sim.neu
TG: simulações
EQ: neural networks (EN)

registro
Código: dat.str.rcd
Nota de escopo: Coleções ordenadas e imutáveis de itens, como as tuplas em Python
UP: tupla
TG: estruturas de dados
EQ: record (EN)

rendering (EN)
EQ: apresentação

reserved keywords (EN)
EQ: palavras reservadas

retorno de funções
Código: pst.fun.rtn
TG: funções
EQ: function return value (EN)

rotação
Código: gra.tra.rot
TG: transformações gráficas
EQ: rotation (EN)

rotation (EN)
EQ: rotação

ruído Perlin
Código: mat.rnd.pno
TG: aleatoriedade
EQ: Perlin noise (EN)

saída de dados
Código: dat.out
TG: dados
TE: escrita em arquivos
TE: exportação de arquivos gráficos
TR: entrada de dados
TR: estruturas de dados
TR: texto
TR: tipagem
TR: variáveis
EQ: output (EN)

sáida em arquivos textuais

USE: escrita em arquivos

saída sonora
Código: exa.snd.out
TG: áudio
EQ: sound output (EN)

scale (EN)
EQ: escala

self-similarity (EN)
EQ: formas auto-similares

shaders
Código: gra.ren.gls
TG: apresentação
EQ: shaders (EN)

shaders (EN)
EQ: shaders

shape (EN)
EQ: formas

símbolos
Código: pst.tok
TG: estruturas de programação
TE: comentários
TE: palavras reservadas
EQ: tokens (EN)

simulações
Código: exa.sim
TG: exemplos de aplicação
TE: redes neurais
TE: simulações biológicas
TE: simulações físicas
TR: algoritmos genéticos
TR: aplicativos móveis
TR: computação física
TR: visão computacional
EQ: simulation (EN)

simulações biológicas
Código: exa.sim.bio
TG: simulações
EQ: biological simulation (EN)

simulações físicas
Código: exa.sim.phy
TG: simulações
EQ: physical simulation (EN)

simulation (EN)
EQ: simulações

sintaxe
USE: estruturas de programação

sistema de Lindenmayer
USE: Sistema-L

Sistema-L
Código: exa.ssf.lsy
Nota de escopo: Sistema-L, também conhecido com sistema de Lindenmayer ou L-System é uma gramatica formal desenvolvida porAristid Lindenmayer que modela o crescimento de plantas.
UP: sistema de Lindenmayer
TG: formas auto-similares
EQ: L-System (EN)

sistemas de coordenadas
Código: mat.geo.coo
TG: geometria
EQ: coordinate systems (EN)

sistemas de numeração
Código: mat.num
TG: matemática
TE: números binários
TE: números hexadecimais
EQ: number system (EN)

sistemas operacionais
Código: dev.sys
TG: engenharia de software
TE: variáveis de sistema
TR: depuração
TR: implantação de software
TR: interface de programação de aplicação
EQ: operating system (EN)

sketching
Código: dev.tec.ske
UP: prototipagem
TG: estratégias de desenvolvimento
EQ: sketching (EN)

sketching (EN)
EQ: sketching

software development (EN)
EQ: engenharia de software

som
USE: áudio

sorting algorithms (EN)
EQ: algoritmos de ordenação

sound (EN)

EQ: áudio

sound input (EN)

EQ: entrada sonora

sound output (EN)

EQ: saída sonora

stroke (EN)

EQ: traço

system variables (EN)

EQ: variáveis de sistema

teclado

Código: dat.inp.kbd

TG: dispositivos de interface humana

TG: entrada de dados

EQ: keyboard (EN)

tempo

Código: dat.inp.tim

Nota de escopo: A questão do tempo é tratada na literatura tanto do ponto de vista da engenharia de software dev, pas principalmente como uma possível entrada de dados dat.inp (MAEDA, 1999; FRY e REAS, 2007)

TG: entrada de dados

EQ: time (EN)

text encoding (EN)

EQ: codificação de caracteres

text output

text strings (EN)

EQ: texto

texto

Código: dat.txt

Nota de escopo: Texto é um termo específico de dados que no entanto se refere a uma ampla gama de assuntos que se reclacionam com tipagem de dados, entrada de dados e saida de dados (nestes últimos casos, especialmnete com a leitura e escrita de arquivos textuais) ou ainda tipografia (em gráficos). Strings em muitas linguagens são tratados como objetos com métodos próprios e propriedades semelhantes a estruturas de dados.

TG: dados

TE: codificação de caracteres

TR: banco de dados

TR: conversão de dados

TR: entrada de dados

TR: estruturas de dados

TR: saída de dados

TR: tipagem

EQ: text strings (EN)

texturas

Código: gra.tri.tex

TG: gráficos tridimensionais

EQ: textures (EN)

textures (EN)

EQ: texturas

time (EN)

EQ: tempo

tipagem

Código: dat.typ

TG: dados

TE: conversão de dados

TE: números de ponto flutuante

TE: números inteiros

TE: valores booleanos

TR: entrada de dados

TR: estruturas de dados

TR: saída de dados

TR: texto

TR: variáveis

EQ: type (EN)

tipografia
Código: gra.tyg
TG: gráficos
EQ: typography (EN)

tokens (EN)
EQ: símbolos

traço
Código: gra.atr.stk
TG: atributos gráficos
EQ: stroke (EN)

transformações de coordenadas
USE: transformações gráficas

transformações gráficas
Código: gra.tra
UP: transformações de coordenadas
TG: gráficos
TE: escala
TE: rotação
TE: transĺação
EQ: graphic transformations (EN)

transĺação
Código: gra.tra.trn
TG: transformações gráficas
EQ: translate (EN)

translate (EN)
EQ: transĺação

trigonometria
Código: mat.geo.trg
TG: geometria
EQ: trigonometry (EN)

trigonometry (EN)
EQ: trigonometria

tupla
USE: registro

type (EN)
EQ: tipagem

type conversion (EN)
EQ: conversão de dados

typography (EN)
EQ: tipografia

Unicode
USE: codificação de caracteres

use of libraries (EN)
EQ: uso de bibliotecas

uso de bibliotecas
Código: dev.lib
TG: engenharia de software
EQ: use of libraries (EN)

valores booleanos
Código: dat.typ.boo
Nota de escopo: Dados booleanos, em homenagem a George Bool, contêm os valores "Verdadeiro" ou Falso (True ou False, em inglês) podendo ser representados também como 1 e 0. São o resultado de comparações, utilizados com operadores lógicos e em condicionais e outras estruturas de controle.
TG: tipagem
EQ: booleans (EN)

variable scope (EN)
EQ: escopo de variáveis

variables (EN)
EQ: variáveis

variáveis
Código: dat.var

Nota de escopo: Variáveis são nomes que fazem referência a valores (dados) na memória do computador.
TG: dados
TE: atribuição
TE: escopo de variáveis
TR: entrada de dados
TR: estruturas de dados
TR: saída de dados
TR: tipagem
EQ: variables (EN)

variáveis de sistema
Código: dev.sys.var
TG: sistemas operacionais
EQ: system variables (EN)

vector graphics (EN)
EQ: gráficos vetoriais

vectors (EN)
EQ: vetores

vertices (EN)
EQ: vértices

vértices
Código: mat.geo.ver
TG: geometria
EQ: vertices (EN)

vetores
Código: mat.geo.vec
TG: geometria
EQ: vectors (EN)

visão computacional
Código: exa.ocv
TG: exemplos de aplicação
TR: algoritmos genéticos
TR: aplicativos móveis
TR: áudio
TR: autômatos celulares
TR: computação física
TR: simulações
EQ: computer vision (EN)

visão do observador
Código: gra.tri.cam
UP: câmera
TG: gráficos tridimensionais
EQ: camera settings (EN)

visualização de dados
Código: exa.dat.viz
TG: ciência de dados
EQ: data visualization (EN)

write to file (EN)
EQ: escrita em arquivos

Apêndice C – Registro dos códigos-exemplo classificados

Listagem dos exemplos didáticos classificados, na forma de um quadro, de maneira a registrar formalmente este produto da pesquisa. O material também está disponível para buscas e consultas em um repositório *Omeka* em <www.lugaralgum.com/tematres>.

nome	identificador	descrição	assunto
KeyCodes	Advanced>KeyCodes>KeyCodes	Escreve no console o identificador das teclas pressionadas.	teclado (dat.inp.kbd)
MoveEye	Basics>Camera>MoveEye	Desenha um cubo, e o movimento do mouse altera altura do observador.	gráficos tridimensionais (gra.tri) visão do observador (gra.tri.cam)
Orthographic	Basics>Camera>Orthographic	Demonstra diferença entre projeção ortográfica e perspectiva em um cubo. Movimento do mouse muda volume de recorte.	gráficos tridimensionais (gra.tri) projeções (gra.tri.prj) visão do observador (gra.tri.cam)
Perspective	Basics>Camera>Perspective	Desenha dois cubos e demonstra alterações nos parâmetros da perspeciva com o movimento do mouse.	gráficos tridimensionais (gra.tri) projeções (gra.tri.prj) visão do observador (gra.tri.cam)
Brightness	Basics>Color>Brightness	Desenha faixas verticais de diversos matizes conforme a movimentação horizontal do mouse, variando o brilho conforme o movimento do mouse na vertical.	cor (gra.clr)
ColorVariables	Basics>Color>ColorVariables	Cria variáveis que guardam valores de cor e desenha quadrados sobrepostos em uma composição que homenageia Josef Albers.	cor (gra.clr) variáveis (dat.var)
Hue	Basics>Color>Hue	Desenha faixas verticais conforme a movimentação horizontal do mouse, variando o matiz conforme a posição vertical do mouse.	cor (gra.clr)
LinearGradient	Basics>Color>LinearGradient	Desenha gradientes linerares demonstrando a função de interpolação linear entre cores.	cor (gra.clr) Interpolação linear (mat.irp.lrp)
RadialGradient	Basics>Color>RadialGradient	Desenha gradientes radiais sorteando novas cores aleatórias a cada segundo.	cor (gra.clr)
Relativity	Basics>Color>Relativity	Demonstra a variação perceptiva das cores	cor (gra.clr)

		dependendo do contexto (justaposição)	
Saturation	Basics>Color>Saturation	Desenha faixas verticais conforme a movimentação horizontal do mouse, variando a saturação conforme a posição vertical do mouse.	cor (gra.clr)
WaveGradient	Basics>Color>WaveGradient	Preenche a área de desenho com uma distribuição gradiente de cores que varia nos dois sentidos	cor (gra.clr)
BooleanOperators	Basics>Control>BooleanOperators	Demonstra operadores lógicos alterando desenho de uma sequência de linhas e pontos gerados por um laço de repetição.	condicionais (pst.cnd) operadores lógicos (pst.ope.log) comparações (pst.ope.cmp)
Conditionals1	Basics>Control>Conditionals1	Demonstra o controle de execução condicional com "if" alterando o desenho de uma sequência de linhas quando o valor da abscissa é divisível por 20.	condicionais (pst.cnd) operações matemáticas (mat.ope)
Conditionals2	Basics>Control>Conditionals2	Demonstra o controle de execução com condicionais encadeadas, alterando o desenho de uma sequência de linhas .	condicionais (pst.cnd) condicionais encadeadas (pst.cnd.cha)
EmbeddedIteration	Basics>Control>EmbeddedIteration	Desenha uma grade de pontos conectados ao centro da tela	laços aninhados (pst.ite.nes)
Iteration	Basics>Control>Iteration	Desenha uma composição a partir de repetição de formas, unsando laços for.	iteração (pst.ite)
DatatypeConversion	Basics>Data>DatatypeConversion	Demonstra algumas conversões de valores entre diferentes tipos e apresenta como textos na área de desenho estes valores.	valores booleanos (dat.typ.con) texto (dat.txt) tipografia (gra.tyg)
IntegersFloats	Basics>Data>IntegersFloats	Compara o incremento de um número inteiro e de um número de ponto flutuante, animando duas linhas.	valores booleanos (dat.typ.con) conversão de dados (dat.typ.flo) números inteiros (dat.typ.int)
Strings	Basics>Data>Strings	Escreve um string de texto na tela que é ampliado conforme o usuário digita letras.	texto (dat.txt) tipografia (gra.tyg) teclado (dat.inp.kbd)
TrueFalse	Basics>Data>TrueFalse	Demonstra execução condicional com o desenho de uma sequencia de linhas horizontais ou verticais dependendo da condição de uma variável que descreve a posição ser	números de ponto flutuante (dat.typ.boo) condicionais (pst.cnd)

		maior que o valor numérico da metade da tela.	
Variables	Basics>Data>Variables	Demonstra o uso elementar de variáveis, deslocando o desenho repetido de um grupo de linhas pela alteração dos valores de algumas variáveis usadas em seu posicionamento na tela.	variáveis (dat.var) atribuição (dat.var.asg)
VariableScope	Basics>Data>VariableScope	Demontra atribuição de variáveis com escopo global e local.	escopo de variáveis (dat.var.sco)
Bezier	Basics>Form>Bezier	Demonstra o uso de curvas Bezier, desenhando uma sequencia de curvas que tem uma das extremidades alterada interativamente com o movimento do mouse.	formas (gra.shp) curvas matemáticas (mat.cur)
PieChart	Basics>Form>PieChart	Constroi um gráfico de pizza a partir de uma coleção de ângulos.	formas (gra.shp)
PointsLines	Basics>Form>PointsLines	Desenha linhas e pontos, em posições calculadas por meio de variáveis	formas (gra.shp)
Primitives3D	Basics>Form>Primitives3D	Desenha dois objetos tridimensionais primitivos, uma caixa cúbica e uma esfera.	gráficos tridimensionais (gra.tri)
RegularPolygon	Basics>Form>RegularPolygon	Desenha três polígonos regulares girando	formas (gra.shp)
ShapePrimitives	Basics>Form>ShapePrimitives	Demonstra em uma cmposição o desenho de triângulo, elipse, quadrilátero, retângulo e arco.	formas (gra.shp)
Star	Basics>Form>Star	Função quer desenha estrelas.	formas (gra.shp) trigonometria (mat.geo.trg) iteração (pst.ite)
TriangleStrip	Basics>Form>TriangleStrip	Desenha um anel a partir de vértices conectados por triângulos.	formas (gra.shp)
Alphamask	Basics>Image>Alphamask	Demonstra a mistura de uma máscara de transparência em uma imagem bitmap	gráficos raster (gra.img)
BackgroundImage	Basics>Image>BackgroundImage	Demonstra o uso de imagem como fundo, animando uma linha horizontal em movimento para baixo.	gráficos raster (gra.img)
CreateImage	Basics>Image>CreateImage	Cria um recurso de imagem, um buffer de pixels, com um gradiente.	gráficos raster (gra.img)
LoadDisplayImage	Basics>Image>LoadDisplayImage	Demonstra como exibir uma imagem bitmap na tela.	gráficos raster (gra.img)
Pointillism	Basics>Image>Pointillism	Lê pixels em posições aleatórias de uma	gráficos raster (gra.img) pixels (gra.img.pix)

		imagem e usa suas cores para desenhar círculos, gerando efrito de pontilismo.	
RequestImage	Basics>Image>RequestImage	Exemplo de carregamento asíncrono de imagens	gráficos raster (gra.img)
Transparency	Basics>Image>Transparency	Exemplo de manipulação da opacidade de uma imagem.	gráficos raster (gra.img)
Clock	Basics>Input>Clock	Simulação de um relógio	tempo (dat.inp.tim)
Constrain	Basics>Input>Constrain	Círculo que se move em direção ao mouse, limitado a uma área retangular.	mouse (dat.inp.mou) gráficos em movimento (gra.mov)
Easing	Basics>Input>Easing	Anima um círculo em direção a posição do ponteiro do mouse na tela, 5% da distância a cada ciclo, gerando o comportamento conhecido como “Easing” da chegada ao alvo.	mouse (dat.inp.mou) gráficos em movimento (gra.mov) matemática (mat)
Keyboard	Basics>Input>Keyboard	Gera diferentes faixas verticais cinza para cada tecla precionada no teclado. A barra de espaço limpa a tela.	teclado (dat.inp.kbd) condicionais (pst.cnd)
KeyboardFunctions	Basics>Input>KeyboardFunctions	Baseado na obra “Color Type Writer” de John Maeda, cria sequências de retângulos coloridos conforme se preciona teclas.	teclado (dat.inp.kbd) programação orientada a eventos (exa.gui.eve)
Milliseconds	Basics>Input>Milliseconds	Animação de faixas verticais em tons de cinza que variam conforme o resto da dividão do valor do tempo transcorrido em millisegundos.	tempo (dat.inp.tim) operações matemáticas (mat.ope)
Mouse1D	Basics>Input>Mouse1D	Varia o desenho de dois retângulos com o valor da posição X do mouse	mouse (dat.inp.mou) variáveis (dat.var)
Mouse2D	Basics>Input>Mouse2D	Varia o desenho de dois retângulos com a movimentação do mouse.	mouse (dat.inp.mou) variáveis (dat.var)
MouseFunctions	Basics>Input>MouseFunctions	Implementa o “arraste” de um retângulo com clique e arraste do mouse sobre o mesmo.	mouse (dat.inp.mou) programação orientada a eventos (exa.gui.eve)
MousePress	Basics>Input>MousePress	Demonstra o reconhecimento da presão do mouse alterando a cor das cruzes que são desenhadas em sobreposição sob o ponteiro do mouse.	mouse (dat.inp.mou) condicionais (pst.cnd)
MouseSignals	Basics>Input>MouseSignals	Três faixas rolantes da direita para esquerda registram gráficamente a	mouse (dat.inp.mou) iteração (pst.ite)

		posição X e Y do mouse, assim como os cliques.	
StoringInput	Basics>Input>StoringInput	Armazena dinamicamente um histórico das posições do mouse em uma estrutra de dados (deque) e desenha círculos a partir deste histórico.	mouse (dat.inp.mou) estruturas de dados (dat.str)
Directional	Basics>Lights>Directional	Demonstra em duas esferas a mudaça de direção de uma fonte de luz.	gráficos tridimensionais (gra.tri) luzes (gra.tri.lig)
Mixture	Basics>Lights>Mixture	Em um cubo que pode ser girado com a movimentação do mouse, a soma de luzes de três cores é demonstrada.	gráficos tridimensionais (gra.tri) luzes (gra.tri.lig)
MixtureGrid	Basics>Lights>MixtureGrid	Simulação de um relógio	gráficos tridimensionais (gra.tri) luzes (gra.tri.lig) laços aninhados (pst.ite.nes)
OnOff	Basics>Lights>OnOff	Demontra o a diferença do acionamento das luzes gerais com o clique do mouse.	gráficos tridimensionais (gra.tri) luzes (gra.tri.lig)
Reflection	Basics>Lights>Reflection	Demonstra em uma esfera a variação de um material especular com o movimento horizontal do mouse.	gráficos tridimensionais (gra.tri) luzes (gra.tri.lig) texturas (gra.tri.tex)
Spot	Basics>Lights>Spot	Altera com a movimentação do mouse uma luz direcional na superfície de uma efera.	gráficos tridimensionais (gra.tri) luzes (gra.tri.lig)
List	Basics>Lists>List	Demonstra o uso de uma lista na construção de três faixas de linhas com variação de cores.	estruturas de dados (dat.str) lista (dat.str.lst)
List2D	Basics>Lists>List2D	Demonstra o uso de listas aninhadas para produzir uma matrix de pontos de tamanho variável.	estruturas de dados (dat.str) matriz (dat.str.mtr) laços aninhados (pst.ite.nes)
ListObjects	Basics>Lists>ListObjects	O exemplo apresenta uma grade de pontos que se movem, produzido por objetos armazenados em uma lista.	estruturas de dados (dat.str) orientação a objetos (pst.oop) laços aninhados (pst.ite.nes)
AdditiveWave	Basics>Math>AdditiveWave	Animação com onda composta por círculos translúcidos baseada na soma de senoidais de diferentes frequências.	trigonometria (mat.geo.trg) gráficos em movimento (gra.mov) curvas matemáticas (mat.cur)
Arctangent	Basics>Math>Arctangent	Animação de instâncias de objetos em forma de olhos que miram o ponteiro do mouse (direção calculada pelo arcotangente)	trigonometria (mat.geo.trg) orientação a objetos (pst.oop)
Distance1D	Basics>Math>Distance1D	Animação de faixas verticais cuja velocidade é controlada pela	geometria (mat.geo) gráficos em movimento (gra.mov)

		distância do mouse ao centro da tela.	
Distance2D	Basics>Math>Distance2D	Grade de círculos cujo diâmetro varia conforme muda a distância para o ponteiro do mouse.	geometria (mat.geo) laços aninhados (pst.ite.nes)
DoubleRandom	Basics>Math>DoubleRandom	Distribuição de pontos baseada em uma faixas de números aleatórios definida também por uma função aleatória.	aleatoriedade (mat.rnd) iteração (pst.ite)
Graphing2DEquation	Basics>Math>Graphing2DEquation	Gráfico de sin(N * cos(r) + 5 * theta), onde N varia com a posição horizontal do mouse.	curvas matemáticas (mat.cur) trigonometria (mat.geo.trg)
IncrementDecrement	Basics>Math>IncrementDecrement	Animação de grandientes formados por linhas cuja posição é incrementada em variáveis globais.	gráficos em movimento (gra.mov) variáveis (dat.var) escopo de variáveis (dat.var.sco)
Interpolate	Basics>Math>Interpolate	Animação idêntica ao exitememplo “Easing” porém implementada com a função lerp()	interpolação (mat.irp) gráficos em movimento (gra.mov) mouse (dat.inp.mou)
Map	Basics>Math>Map	Círculo cujo diâmetro e cor tem os valores calculados a partir do movimento horizontal do mouse, traduzindo o valor da posição horizontal do ponteiro para uma nova faixa de valores com map() de Processing.	matemática (mat) gráficos em movimento (gra.mov) mouse (dat.inp.mou)
Noise1D	Basics>Math>Noise1D	Uma dimensão do Ruído Perlin é usada para deslocar um círculo na horizontal, cuja imagem em posição anterior desaparece gradualmente.	ruído Perlin (mat.rnd.pno) gráficos (gra)
Noise2D	Basics>Math>Noise2D	Demonstra o uso de duas dimensões do Ruído Perling para gerar uma textura. O movimento do mouse altera o parâmetro de “detalhamento”.	ruído Perlin (mat.rnd.pno) gráficos raster (gra.img)
Noise3D	Basics>Math>Noise3D	Demonstra o uso de três dimensïes do Ruído Perlin para criação de uma textura. Tratando a terceira dimensão como o tempo, e desta forma animando a textura.	ruído Perlin (mat.rnd.pno) gráficos (gra) gráficos raster (gra.img)
NoiseWave	Basics>Math>NoiseWave	Usa o Ruído Perlin para gerar um padrão ondulatório.	ruído Perlin (mat.rnd.pno)
OperatorPrecedence	Basics>Math>OperatorPrecedence	Código comentado com explicações sobre a precedência de operadores e uso de parenteses gera sequências de linhas e	precedência de operadores (pst.ope.ord)

		retângulos.	
PolarToCartesian	Basics>Math>PolarToCartesian	Animação de um círculo que gira em velocidade crescente em torno do centro da tela demonstra conversão simples de coordenadas polares em cartesianas.	sistemas de coordenadas (mat.geo.coo) trigonometria (mat.geo.trg)
Random	Basics>Math>Random	Exemplo de aleatoriedade. Campo de linhas verticais cujo valor de cinza é aleatório. Redesenhado a cada ciclo de draw().	aleatoriedade (mat.rnd)
RandomGaussian	Basics>Math>RandomGaussian	Desenha círculos cuja posição horizontal é definida por um número aleatório gerado em distribuissão Gaussiana.	distribuições aleatórias (mat.rnd.dis)
Sine	Basics>Math>Sine	Três círculos oscilam em tamanho suavemente segundo uma variação senoidal	trigonometria (mat.geo.trg) gráficos em movimento (gra.mov)
SineCosine	Basics>Math>SineCosine	Movimento linear de quatro círculos coloridos, controlados por seno se cossenos, nas laterais de um quadrado central.	trigonometria (mat.geo.trg) gráficos em movimento (gra.mov)
SineWave	Basics>Math>SineWave	Onda senoidal animada, desenhada com círculos.	trigonometria (mat.geo.trg) gráficos em movimento (gra.mov)
CompositeObjects	Basics>Objects>CompositeObjects	Exemplo de composição em orientação a objetos. Objetos "Ovo" com elemento "Anel"	orientação a objetos (pst.oop)
Inheritance	Basics>Objects>Inheritance	Exemplo de herança em orientação a objetos. Círculos e barra que giram em torno do centro da tela, implementadados por meio de Classes SpinArm e SpinSpot que herdam da classe Spin (um objeto que tem posição, ángulo e velocidade angular).	orientação a objetos (pst.oop) herança (pst.oop.inh)
MultipleConstructors	Basics>Objects>MultipleConstructors	Exemplo de construção de instâncias de um objeto com diferentes números/escolhas de argumentos passados.	orientação a objetos (pst.oop)
Objects	Basics>Objects>Objects	Exemplo de orientação a objetos. Classe que produz coleções retangulares de sequencias de linhas que se movimentam de maneira diferente conforme a movimentação do ponteiro do mouse.	orientação a objetos (pst.oop)
DisableStyle	Basics>Shape>DisableStyle	Demonstra a manipulação dos atributos de um gráfico	gráficos vetoriais (gra.vec) atributos gráficos (gra.atr) importação de arquivos

		vetorial importado.	gráficos (dat.inp.gra)
GetChild	Basics>Shape>GetChild	Demonstra o acesso a um subobjeto de um gráfico vetorial importado.	gráficos vetoriais (gra.vec) importação de arquivos gráficos (dat.inp.gra)
LoadDisplayOBJ	Basics>Shape>LoadDisplayOBJ	Demonstra a importação de um arquivo gráfico 3D e sua apresentação na tela.	gráficos vetoriais (gra.vec) gráficos tridimensionais (gra.tri) importação de arquivos gráficos (dat.inp.gra)
LoadDisplaySVG	Basics>Shape>LoadDisplaySVG	Demonstra a importação de um arquivo gráfico SVG e sua aplicação na tela.	gráficos vetoriais (gra.vec) importação de arquivos gráficos (dat.inp.gra)
ScaleShape	Basics>Shape>ScaleShape	Demonstra a ampliação na apresentação de um gráfico SVG importado.	gráficos vetoriais (gra.vec) escala (gra.tra.sca)
ShapeVertices	Basics>Shape>ShapeVertices	Itera pelos vértices de uma forma importade de um arquivo SVG permitindo a animação do contorno de um mapa do Reino Unido.	gráficos vetoriais (gra.vec) visualização de dados (exa.dat.viz)
Coordinates	Basics>Structure>Coordinates	Exemplo elementar do funcionamento do sistema de coordenadas, deshando pontos, linha e retângulo.	gráficos (gra) sistemas de coordenadas (mat.geo.coo)
CreateGraphics	Basics>Structure>CreateGraphics	Exemplo de criação de um objeto gráfico PGraphics, sobreposto ao desenho normal da tela. O mouse desenha círculos cheios sobre a tela e vazados no objeto criado.	gráficos vetoriais (gra.vec)
Functions	Basics>Structure>Functions	Exemplo elementar de declaração e chamada de uma função "alvo" que desenha círculos concêntricos.	declaração de funções (pst.fun.def) passagem de argumentos (pst.fun.arg) iteração (pst.ite)
Loop	Basics>Structure>Loop	Exemplifica o controle do laço principal draw() desativado com noLoop(), e reiniciado com loop() no evento de um clique do mouse.	laço principal (pst.ite.sml)
NoLoop	Basics>Structure>NoLoop	Exemplifica a situação em que o laço principal draw() é executado apenas uma vez devido a instrução noLoop()	laço principal (pst.ite.sml)
Recursion	Basics>Structure>Recursion	Exemplo mínimo de recursão com a definição de uma função que desenha círculos e esta recursivamente desenho círculos internos.	algoritmos de ordenação (csc.rcr) funções (pst.fun)
Redraw	Basics>Structure>Redraw	Exemplo de exemplo da repetição de draw() desativada e o redesenho sendo invocado por redraw() no evento do clique do mouse.	laço principal (pst.ite.sml)

SetupDraw	Basics>Structure>SetupDraw	Exemplo elementar da estrutura setup() /draw() animando um linha horizontal. Equivalente ao exemplo Linear.	laço principal (pst.ite.sml) escopo de variáveis (dat.var.sco) gráficos em movimento (gra.mov)
StatementsComments	Basics>Structure>StatementsComments	Exemplo elementar que define tamanho da tela e um fundo de cor caramelo em código que também exemplifica o uso de comentários.	palavras reservadas (pst.tok.com) funções (pst.fun) gráficos (gra)
WidthHeight	Basics>Structure>WidthHeight	Exemplo elementar que desenha faixas entrelaçadas de retângulos em diferentes cores para exemplificar o uso das variáveis de sistema width e height. Se mostra também um bom exemplo de iteração.	variáveis de sistema (dev.sys.var) iteração (pst.ite) gráficos (gra)
Arm	Basics>Transform>Arm	Braço articulado de dois segmentos com ângulos controlados independentemente pela posição vertical e horizontal do mouse.	rotação (gra.tra.rot) gráficos em movimento (gra.mov) sistemas de coordenadas (mat.geo.coo)
Rotate	Basics>Transform>Rotate	Exemplo elementar de rotação de um quadrado em torno do seu ponto central, em vibração aleatória	rotação (gra.tra.rot) sistemas de coordenadas (mat.geo.coo) aleatoriedade (mat.rnd)
RotatePushPop	Basics>Transform>RotatePushPop	Rotação encavalada de vários cubos, demonstra o uso da pilha de contextos de coordenadas (pushMatrix())	gráficos tridimensionais (gra.tri) rotação (gra.tra.rot) sistemas de coordenadas (mat.geo.coo)
RotateXY	Basics>Transform>RotateXY	Animação da rotação cumulativa de dois planos no espaço.	transformações gráficas (gra.tra) rotação (gra.tra.rot) sistemas de coordenadas (mat.geo.coo)
Scale	Basics>Transform>Scale	Exemplifica o uso do comando scale() com animação oscilante de dois quadrados.	escala (gra.tra.sca) trigonometria (mat.geo.trg)
Translate	Basics>Transform>Translate	Exemplifica o funcionamento de translate() construindo uma animação com dois quadrados. Demonstra o caráter cumulativo das translações (o segundo retângulo se move duas vezes mais rápido).	transformações gráficas (gra.tra) transĺação (gra.tra.trn)
FiveWaysOfWritingText	Basics>Typography>FiveWaysOfWritingText	Exemplo elementar de apresentação de texto na tela.	tipografia (gra.tyg) texto (dat.txt)
Letters	Basics>Typography>Letters	Exemplifica o a presentação de texto na tela.	tipografia (gra.tyg) texto (dat.txt)
Words	Basics>Typography>Words	Exemplifica o a presentação de texto na	tipografia (gra.tyg) texto (dat.txt)

		tela.	
EmbeddedLinks	Basics>Web>EmbeddedLinks	Demonstra o acionamento de uma URL no navegador com o clique em uma região retangular.	interação com o navegador (dev.api.web) interfaces gráficas (exa.gui)
LoadingImages	Basics>Web>LoadingImages	Carrega imagem a partir de um servidor.	gráficos raster (gra.img) importação de arquivos gráficos (dat.inp.gra)
BoxClock	Demos>Graphics>BoxClock	Caixa com arestas pintadas conforme a orientação em relação aos eixos tem suas dimensões definidas pelo horário do dia.	tempo (dat.inp.tim) gráficos tridimensionais (gra.tri)
LowLevelGL	Demos>Graphics>LowLevelGL	Demonstra chamadas de baixo nível a infraestrutura OpenGL (girando um triângulo colorido no espaço)	gráficos tridimensionais (gra.tri)
Particles	Demos>Graphics>Particles	Demonstra sistema de partículas que carregam um recurso de imagem (“sprite”)	simulações físicas (exa.sim.phy) orientação a objetos (pst.oop) gráficos raster (gra.img)
Tentacles	Demos>Graphics>Tentacles	Animação de agentes alongados formados por segmentos se deslocando em movimentos sinuosos em espaço esférico.	gráficos tridimensionais (gra.tri) orientação a objetos (pst.oop) gráficos em movimento (gra.mov)
Trefoil	Demos>Graphics>Trefoil	Texturas se acumulam em superfície revelando o volume de um nó "trefoil"	gráficos tridimensionais (gra.tri) texturas (gra.tri.tex)
Wiggling	Demos>Graphics>Wiggling	Faces com furos redondos formando um cubo gira no espaço. A tecla "w" permite que uma randomização movimente cada um dos vértices da estrutura	gráficos tridimensionais (gra.tri)
Yellowtail	Demos>Graphics>Yellowtail	Partindo do gesto do usuário ao arrastar o mouse precionado, anima agentes que saem e e retornam a tela.	mouse (dat.inp.mou) vetores (mat.geo.vec) estruturas de dados (dat.str)
Esfera	Demos>Performance>Esfera	Esfera formada por inúmeras linhas direcionadas ao centro.	gráficos tridimensionais (gra.tri) geometria (mat.geo) aleatoriedade (mat.rnd)
LineRendering	Demos>Performance>LineRendering	Acúmulo de grande número de linhas distribuidas aleatoriamente (para testar o motor de apresentação).	gráficos (gra) aleatoriedade (mat.rnd)
QuadRendering	Demos>Performance>QuadRendering	Acúmulo de grande número de retângulos distribuidos aleatoriamente (para testar o motor de apresentação).	gráficos (gra) aleatoriedade (mat.rnd)
TextRendering	Demos>Performance>TextRendering	Acúmulo de grande número de palavras “HELLO” distribuidas	gráficos (gra) aleatoriedade (mat.rnd)

		aleatoriamente (para testar a apresentação de texto).	
NoBackgroundTest	Demos>Tests>NoBackgroundTest	Um exemplo mínimo de desenho com o mouse de círculos sobrepostos	gráficos (gra)
RedrawTest	Demos>Tests>RedrawTest	Um exemplo mínimo do congelamento do laço principal de desenho (draw) que passa a ser acionado pontualmente pelo evento de uma tecla precionada.	gráficos (gra) laço principal (pst.ite.sml)
ResizeTest	Demos>Tests>ResizeTest	Habilita a mudança do tamanho da área de desenho. [Não estava executando em 1 de dezembro de 2018]	gráficos (gra) sistemas operacionais (dev.sys)
IntAndFloat	Python Mode Differences>IntAndFloat	Demonstra diferenças nos tipos Ingeger e Float no Python Mode.	tipagem (dat.typ)
LiteralColors	Python Mode Differences>LiteralColors	Demonstra a notação literal de cores alternativa.	símbolos (pst.tok) cor (gra.clr)
WithStatements	Python Mode Differences>WithStatements	Exemplifica o gerenciamento de contexto “with” de Python	estruturas de programação (pst) transformações gráficas (gra.tra)
A_List	Topics>AdvancedData>A_List	Permite que objetos Ball sejam criados com o clique do mouse e adicionados a uma lista. As bolas ao terminarem a simulação de queda são removidas da estrutura de dados.	estruturas de dados (dat.str) lista (dat.str.lst) orientação a objetos (pst.oop)
LoadSaveTable	Topics>AdvancedData>LoadSaveTable	Carrega dados de um arquivo com uma tabela CSV e com ela desenha círculos nomeados. Cliques do mouse criam novos círculos que são salvos na tabela.	escrita em arquivos (dat.out.fil) leitura de arquivos (dat.inp.fil)
AnimatedSprite	Topics>Animation>AnimatedSprite	Exemplo de animação “sprite”	gráficos em movimento (gra.mov) gráficos raster (gra.img)
Sequential	Topics>Animation>Sequential	Demonstra a apresentação de animação a partir de uma coleção de imagens carregadas.	gráficos em movimento (gra.mov) gráficos raster (gra.img)
GameOfLife	Topics>Cellular Automata>GameOfLife	Automato celular de Conway (Game of Life).	autômatos celulares (exa.oca) estruturas de dados (dat.str)
Wolfram	Topics>Cellular Automata>Wolfram	Automato celular de Wolfram.	autômatos celulares (exa.oca) estruturas de dados (dat.str)
ContinuousLines	Topics>ContinuousLines	Uma ferramenta de desenho elementar que produz linhas conectadas com o arraste do mouse	mouse (dat.inp.mou) gráficos (gra)
BeginEndContour	Topics>Create Shapes>BeginEndContour	Demonstra a criação de uma forma poligonal com um furo (animada	formas (gra.shp) polilinhas (gra.shp.ply) rotação (gra.tra.rot)

		girando sobre o próprio centro)	
GroupPShape	Topics>Create Shapes>GroupPShape	Demonstra como agrupar objetos gráficos PShape	formas (gra.shp) gráficos em movimento (gra.mov)
ParticleSystemPShape	Topics>Create Shapes>ParticleSystemPShape	Sistema de partículas que se vale de um PShape para a apresentação	gráficos raster (gra.img) orientação a objetos (pst.oop)
PathPShape	Topics>Create Shapes>PathPShape	Cria um objeto gráfico PShape de uma curva senoidal.	formas (gra.shp) curvas matemáticas (mat.cur) trigonometria (mat.geo.trg)
PolygonPShape	Topics>Create Shapes>PolygonPShape	Demonstra a construção de um objeto PShape poligonal.	polilinhas (gra.shp.ply) trigonometria (mat.geo.trg)
PolygonPShapepst.oop	Topics>Create Shapes>PolygonPShapepst.oop	Animação que demonstra objetos com com formas PShape como atributo.	formas (gra.shp) orientação a objetos (pst.oop)
PolygonPShapepst.oop2	Topics>Create Shapes>PolygonPShapepst.oop2	Animação que demonstra objetos com com formas PShape como atributo.	formas (gra.shp) orientação a objetos (pst.oop)
PolygonPShapepst.oop3	Topics>Create Shapes>PolygonPShapepst.oop3	Animação que demonstra objetos com com formas PShape como atributo.	formas (gra.shp) orientação a objetos (pst.oop)
PrimitivePShape	Topics>Create Shapes>PrimitivePShape	Elipse desenhada como uma “forma primitiva” em um objeto PShape.	formas (gra.shp) métodos (pst.oop.met)
WigglePShape	Topics>Create Shapes>WigglePShape	Movimenta vértices de um objeto PShape individualmente.	formas (gra.shp) polilinhas (gra.shp.ply)
LoadFile1	Topics>File IO>LoadFile1	Carrega uma coleção de posições x e y de um arquivo e com estas desenha pontos.	leitura de arquivos (dat.inp.fil) texto (dat.txt) sistemas de coordenadas (mat.geo.coo)
LoadFile2	Topics>File IO>LoadFile2	Carrega dados de um arquivo e mostra na forma de uma lista. Cliques do mouse mudam a “página” de itens da lista.	leitura de arquivos (dat.inp.fil) texto (dat.txt) Sistema-L (exa.ssf.lsy)
SaveOneImage	Topics>File IO>SaveOneImage	Salva uma imagem	exportação de arquivos gráficos (dat.out.gra)
Koch	Topics>Fractals and L-Systems>Koch	Curva de Koch.	fractais (exa.ssf.fra) curvas matemáticas (mat.cur)
Mandelbrot	Topics>Fractals and L-Systems>Mandelbrot	Fractal de Mandelbrot.	fractais (exa.ssf.fra) curvas matemáticas (mat.cur)
PenroseSnowflake	Topics>Fractals and L-Systems>PenroseSnowflake	Demonstração de uma classe que implementa um L-System e cobre uma região do plano com um padrão.	Sistema-L (exa.ssf.lsy) estruturas de programação (pst)
PenroseTile	Topics>Fractals and L-Systems>PenroseTile	Demonstração de uma classe que implementa um L-System e cobre uma região do plano com um padrão.	Sistema-L (exa.ssf.lsy) estruturas de programação (pst)
Pentigree	Topics>Fractals and L-Systems>Pentigree	Demonstração de uma classe que implementa	Sistema-L (exa.ssf.lsy) estruturas de programação

		um L-System e cobre uma região do plano com um padrão.	(pst)
Tree	Topics>Fractals and L-Systems>Tree	Árvore recursiva elementar.	formas auto-similares (exa.ssf) algoritmos de ordenação (csc.rcr)
Icosahedra	Topics>Geometry>Icosahedra	Icosaedro.	geometria (mat.geo)
NoiseSphere	Topics>Geometry>NoiseSphere	Distribuição de linhas como se saindo de uma esfera com pequenas variações aleatórias na direção	geometria (mat.geo) ruído Perlin (mat.rnd.pno) orientação a objetos (pst.oop)
RGBCube	Topics>Geometry>RGBCube	Cubo cuja superfície apresenta transições de cor.	cor (gra.clr) gráficos tridimensionais (gra.tri)
ShapeTransform	Topics>Geometry>ShapeTransform	Animação de primitivas tridimencionais que podem ser alteradas com o teclado.	gráficos tridimensionais (gra.tri) geometria (mat.geo) teclado (dat.inp.kbd)
SpaceJunk	Topics>Geometry>SpaceJunk	Coleção de caixas de dimensões aleatórias giradas e distribuidas no espaço formando uma esfera. O mouse perime aproximar e afastar.	gráficos tridimensionais (gra.tri) aleatoriedade (mat.rnd)
Toroid	Topics>Geometry>Toroid	Animação interativa de um toroide com diversos ajustes.	gráficos tridimensionais (gra.tri) geometria (mat.geo)
Vertices	Topics>Geometry>Vertices	Define função que desenha tronco de cone (ao redor do qual é possível orbitar a câmera)	gráficos tridimensionais (gra.tri) geometria (mat.geo) vértices (mat.geo.ver)
Button	Topics>GUI>Button	Exemplifica a implementação de dois botões, um retangular e outro circular.	interfaces gráficas (exa.gui) mouse (dat.inp.mou)
Handles	Topics>GUI>Handles	Exemplifica a implementação simples de alças de deformação com a classe Handle.	interfaces gráficas (exa.gui) mouse (dat.inp.mou) orientação a objetos (pst.oop)
Rollover	Topics>GUI>Rollover	Demonstra a detecção da sobreposição do mouse sobre uma área retangular e outra circular.	interfaces gráficas (exa.gui) mouse (dat.inp.mou)
Scrollbar	Topics>GUI>Scrollbar	Exemplifica a implementação de duas barras de rolagem que deslocam imagens carregadas.	interfaces gráficas (exa.gui) mouse (dat.inp.mou) gráficos raster (gra.img)
Blending	Topics>Image Processing>Blending	Demonstra os diversos modos de fusão (blend) possíveis na sobreposição de imagens.	gráficos raster (gra.img)
Blur	Topics>Image Processing>Blur	Exemplo de filtro de manipulação de imagem (blur).	gráficos raster (gra.img)
Brightness	Topics>Image Processing>Brightness	Exemplo de filtro de manipulação de imagem,	gráficos raster (gra.img) pixels (gra.img.pix)

		altera o brilho tornando visível apenas a região próxima ao mouse.	
Convolution	Topics>Image Processing>Convolution	Exemplo de implementação de um filtro de manipulação de imagem (convolution)	gráficos raster (gra.img) pixels (gra.img.pix)
EdgeDetection	Topics>Image Processing>EdgeDetection	Exemplo de implementação de um filtro de manipulação de imagem (sharpen)	gráficos raster (gra.img) pixels (gra.img.pix)
Explode	Topics>Image Processing>Explode	Exemplo de filtro de manipulação de imagem	gráficos raster (gra.img) pixels (gra.img.pix) gráficos tridimensionais (gra.tri)
Extrusion	Topics>Image Processing>Extrusion	Exemplo de filtro de manipulação de imagem	gráficos raster (gra.img) pixels (gra.img.pix)
Histogram	Topics>Image Processing>Histogram	Desenha o gráfico de histograma, com a frequência de distribuição dos tons, sobreposto a uma imagem.	gráficos raster (gra.img) pixels (gra.img.pix)
LinearImage	Topics>Image Processing>LinearImage	Animação baseada na ampliação em colunas de uma única fila de pixels de uma imagem.	gráficos raster (gra.img) pixels (gra.img.pix)
PixelArray	Topics>Image Processing>PixelArray	Exemplo de acesso direto aos pixels de uma imagem. Cada pixel é percorrido e sua cor preenche toda a tela.	gráficos raster (gra.img) pixels (gra.img.pix) estruturas de dados (dat.str)
Zoom	Topics>Image Processing>Zoom	Exploração tridimensional dos valores dos pixels de uma imagem.	gráficos raster (gra.img) pixels (gra.img.pix) gráficos tridimensionais (gra.tri)
Follow1	Topics>Interaction>Follow1	Manipulação de um segmento com o mouse,	mouse (dat.inp.mou) trigonometria (mat.geo.trg) rotação (gra.tra.rot)
Follow2	Topics>Interaction>Follow2	Manipulação de múltiplos segmentos com o mouse.	estruturas de dados (dat.str) trigonometria (mat.geo.trg) rotação (gra.tra.rot)
Follow3	Topics>Interaction>Follow3	Manipulação de múltiplos segmentos com o mouse.	estruturas de dados (dat.str) trigonometria (mat.geo.trg) rotação (gra.tra.rot)
Reach1	Topics>Interaction>Reach1	Animação de braço com dois segmentos, que se projeta em direção ao mouse	mouse (dat.inp.mou) trigonometria (mat.geo.trg) rotação (gra.tra.rot)
Reach2	Topics>Interaction>Reach2	Animação de braço com múltiplos segmentos, que se projeta em direção ao mouse	estruturas de dados (dat.str) trigonometria (mat.geo.trg) rotação (gra.tra.rot)
Reach3	Topics>Interaction>Reach3	Animação de braço com múltiplos segmentos, que se projeta em direção a uma bola.	gráficos em movimento (gra.mov) trigonometria (mat.geo.trg) rotação (gra.tra.rot)
Tickle	Topics>Interaction>Tickle	Animação interativa de uma palavra que vibra sob o mouse.	ruído Perlin (mat.rnd.pno) mouse (dat.inp.mou) tipografia (gra.tyg)
Bounce	Topics>Motion>Bounce	Exemplo elementar de um círculo em deslocamento que	gráficos em movimento (gra.mov) gráficos em movimento

		inverte a direção ao tocar as bordas da tela.	(gra.mov) gráficos em movimento (gra.mov)
BouncyBubbles	Topics>Motion>BouncyBubbles	Bolas que colidem entre si e são limitadas pelas bordas da área de desenho.	gráficos em movimento (gra.mov) simulações físicas (exa.sim.phy) orientação a objetos (pst.oop)
Brownian	Topics>Motion>Brownian	Um exemplo de "Random walker" (caminhante aleatório) que produz uma linha contínua.	gráficos em movimento (gra.mov) aleatoriedade (mat.rnd)
CircleCollision	Topics>Motion>CircleCollision	Animação com colisão entre círculos de diferentes tamanhos com cálculo realístico da física de mudança de velocidades.	gráficos em movimento (gra.mov) simulações físicas (exa.sim.phy) orientação a objetos (pst.oop)
CubesWithinCube	Topics>Motion>CubesWithinCube	Caixa aramada cúbica limita uma explosão de pequenos cubos coloridos.	gráficos em movimento (gra.mov) simulações físicas (exa.sim.phy) gráficos tridimensionais (gra.tri)
Linear	Topics>Motion>Linear	Exemplo elementar de animação de uma linha horizontal por meio da atualização de uma variável global.	gráficos em movimento (gra.mov) escopo de variáveis (dat.var.sco) condicionais (pst.cnd)
Morph	Topics>Motion>Morph	Anima a transformação de um quadrado em círculo e de volta a um quadrado usando interpolação linear.	gráficos em movimento (gra.mov) Interpolação linear (mat.irp.lrp)
MovingOnCurves	Topics>Motion>MovingOnCurves	Animação de um círculo ao londo de trajetória curva até o ponto clicado pelo mouse.	gráficos em movimento (gra.mov) curvas matemáticas (mat.cur)
Reflection1	Topics>Motion>Reflection1	Animação de um círculo que rebate em uma superfície inclinada (que é recriada em novo ângulo para cada nova colisão) Uma linha laranja indica a normal da superfície no ponto de contato.	gráficos em movimento (gra.mov) simulações físicas (exa.sim.phy) vetores (mat.geo.vec)
Reflection2	Topics>Motion>Reflection2	Animação de círculo em colisão com uma superfície multisegmentada.	gráficos em movimento (gra.mov) simulações físicas (exa.sim.phy) orientação a objetos (pst.oop)
Pattern	Topics>Pattern	Ferramenta de desenho que varia o tamanho de círculos conforme a velocidade de deslocamento do mouse.	mouse (dat.inp.mou)
Pulses	Topics>Pulses	Ferramenta de desenho que varia periodicamente o tamanho dos elementos pictóricos acionados de forma independente do gesto	mouse (dat.inp.mou) trigonometria (mat.geo.trg)
BlurFilter	Topics>Shaders>BlurFilter	Exemplo de uso de shader.	shaders (gra.ren.gls) gráficos raster (gra.img)
Conway	Topics>Shaders>Conway	Exemplo de uso de shader (que implementa	shaders (gra.ren.gls) autômatos celulares (exa.oca)

		um Jogo da Vida de Conway)	
CustomBlend	Topics>Shaders>CustomBlend	Exemplo de uso de shader.	shaders (gra.ren.gls)
Deform	Topics>Shaders>Deform	Exemplo de uso de shader.	shaders (gra.ren.gls)
DomeProjection	Topics>Shaders>DomeProjection	Exemplo de uso de shader.	shaders (gra.ren.gls)
EdgeDetect	Topics>Shaders>EdgeDetect	Exemplo de uso de shader.	shaders (gra.ren.gls)
EdgeFilter	Topics>Shaders>EdgeFilter	Exemplo de uso de shader.	shaders (gra.ren.gls)
GlossyFishEye	Topics>Shaders>GlossyFishEye	Exemplo de uso de shader.	shaders (gra.ren.gls)
ImageMask	Topics>Shaders>ImageMask	Exemplo de uso de shader.	shaders (gra.ren.gls)
Landscape	Topics>Shaders>Landscape	Exemplo de uso de shader.	shaders (gra.ren.gls)
Monjori	Topics>Shaders>Monjori	Exemplo de uso de shader.	shaders (gra.ren.gls)
Nebula	Topics>Shaders>Nebula	Exemplo de uso de shader.	shaders (gra.ren.gls)
SepBlur	Topics>Shaders>SepBlur	Exemplo de uso de shader.	shaders (gra.ren.gls)
ToonShading	Topics>Shaders>ToonShading	Exemplo de uso de shader.	shaders (gra.ren.gls)
Chain	Topics>Simulate>Chain	Simulação deu ma corrente.	simulações físicas (exa.sim.phy)
Flocking	Topics>Simulate>Flocking	Simulação escrita por Shiffman de revoada baseada no programa Boids de Craif Rainold.	simulações biológicas (exa.sim.bio) orientação a objetos (pst.oop)
ForcesWithVectors	Topics>Simulate>ForcesWithVectors	Exemplo de simulação com gravidade e resistência de deslocamento em fluido, usando vetores.	simulações físicas (exa.sim.phy) vetores (mat.geo.vec)
GravitationalAttraction3D	Topics>Simulate>GravitationalAttraction3D	Simulação de orbita de esferas em torno de um corpo central.	simulações físicas (exa.sim.phy) vetores (mat.geo.vec) gráficos tridimensionais (gra.tri)
MultipleParticleSystems	Topics>Simulate>MultipleParticleSystems	Ao clique do mouse, elementos formados por círculos e linhas são produzidos e precipitam. Implementado com sistemas de partículas.	simulações físicas (exa.sim.phy) estruturas de programação (pst)
SimpleParticleSystem	Topics>Simulate>SimpleParticleSystem	Exemplo de sistema de partículas com gravidade.	simulações físicas (exa.sim.phy) estruturas de programação (pst)

SmokeParticleSystem	Topics>Simulate>SmokeParticleSystem	Simulação de fumaça direcionada pelo vento (indicado pelo mouse) usando sistemas de partículas.	simulações físicas (exa.sim.phy) orientação a objetos (pst.oop)
SoftBody	Topics>Simulate>SoftBody	Polilinha que simula dinâmica de corpos macios se desloca em direção ao mouse.	simulações físicas (exa.sim.phy) polilinhas (gra.shp.ply)
Spring	Topics>Simulate>Spring	Simulação de deformações de ama mola com dois retângulos. A barra horizontal deve ser arrastada com o mouse.	simulações físicas (exa.sim.phy) interfaces gráficas (exa.gui)
Springs	Topics>Simulate>Springs	Três objetos circulares são instanciados. Quando arrastados com o mouse, retornam a posição inicial por meio de da simulação de molas com parâmetros ligeiramente diferentes em cada um.	simulações físicas (exa.sim.phy) orientação a objetos (pst.oop)
TextureCube	Topics>Textures>TextureCube	Demonstração de aplicação de textura em um cubo.	gráficos tridimensionais (gra.tri) texturas (gra.tri.tex)
TextureCylinder	Topics>Textures>TextureCylinder	Demonstração de aplicação de textura em um cilindro.	gráficos tridimensionais (gra.tri) texturas (gra.tri.tex)
TextureQuad	Topics>Textures>TextureQuad	Demonstração de aplicação de textura retangular mapeada em um quadrilátero plano no espaço.	gráficos tridimensionais (gra.tri) texturas (gra.tri.tex)
TextureSphere	Topics>Textures>TextureSphere	Demonstração de aplicação de textura mapeada em uma esfera.	gráficos tridimensionais (gra.tri) texturas (gra.tri.tex)
TextureTriangle	Topics>Textures>TextureTriangle	Demonstração de aplicação de textura retangular mapeada em um triângulo.	gráficos tridimensionais (gra.tri) texturas (gra.tri.tex)
AccelerationWithVectors	Topics>Vectors>AccelerationWithVectors	Círculo que orbita em torno do ponteiro do mouse, demonstra conceitos básicos de simulação de movimento com aceleração e vetores.	simulações físicas (exa.sim.phy) vetores (mat.geo.vec) orientação a objetos (pst.oop)
BouncingBall	Topics>Vectors>BouncingBall	Demonstra o uso de matemática vetorial básica para animar o deslocamento de um círculo sob ação simulada da gravidade.	simulações físicas (exa.sim.phy) vetores (mat.geo.vec)
VectorMath	Topics>Vectors>VectorMath	Demonstra operações de geometria vetorial básica, como subtração, normalização e multiplicação (escalar) ao desenhar uma linha a partir do centro da tela que aponta para o ponteiro do mouse.	vetores (mat.geo.vec)

arduino_input	Contributed Libraries in Python>arduino_input	Exemplo de comunicação serial entre o Processing e Arduino com protocolo Firmata. Os pinos digitais do Arduino são indicados como retângulos preenchidos (HIGH) ou não (LOW), e os pinos analógicos tem a tensão convertida em números entre 0 e 1023 que por sua vez são mostrados como círculos de diferentes diâmetros.	dispositivos de interface humana (dat.inp.hid) computação física (exa.ele) uso de bibliotecas (dev.lib)
Lesson01_AudioContext	Contributed Libraries in Python>Lesson01_AudioContext	[não foi possível executar este exemplo, foi Inferido o tema a partir da biblioteca de áudio utilizada.]	áudio (exa.snd) uso de bibliotecas (dev.lib)
Lesson04_SamplePlayer	Contributed Libraries in Python>Lesson04_SamplePlayer	[não foi possível executar este exemplo, foi Inferido o tema a partir da biblioteca de áudio utilizada.]	áudio (exa.snd) uso de bibliotecas (dev.lib)
Textfield	Contributed Libraries in Python>Textfield	Simulação de deformações de ama mola com dois retângulos. A barra horizontal deve ser arrastada com o mouse.	gráficos vetoriais (gra.vec) uso de bibliotecas (dev.lib)
ContactListener	Contributed Libraries in Python>ContactListener	Exemplo de uso de biblioteca de simulação física com múltiplos círculos em queda interagindo com um objeto poligonal que pode ser arrastado com o mouse.	simulações físicas (exa.sim.phy) uso de bibliotecas (dev.lib)
MSAFluid	Contributed Libraries in Python>MSAFluid	[Não foi possível executar este exemplo. A biblioteca de que depende não estava mais disponível] É possível inferir que se trata de um de simulação física.	simulações físicas (exa.sim.phy) uso de bibliotecas (dev.lib)
BackgroundSubtraction	Contributed Libraries in Python>BackgroundSubtraction	Subtração de partes invariáveis.	visão computacional (exa.ocv) uso de bibliotecas (dev.lib) gráficos raster (gra.img)
BrightestPoint	Contributed Libraries in Python>BrightestPoint	Localização de áreas claras	visão computacional (exa.ocv) uso de bibliotecas (dev.lib) gráficos raster (gra.img)
BrightnessContrast	Contributed Libraries in Python>BrightnessContrast	Manipulação de brilho e contraste.	visão computacional (exa.ocv) uso de bibliotecas (dev.lib) gráficos raster (gra.img)
CalibrationDemo	Contributed Libraries in Python>CalibrationDemo	Demonstração de ajustes da biblioteca.	visão computacional (exa.ocv) uso de bibliotecas (dev.lib) gráficos raster (gra.img)
ColorChannels	Contributed Libraries in Python>ColorChannels	Separação de cores.	visão computacional (exa.ocv) uso de bibliotecas (dev.lib) gráficos raster (gra.img)
DepthFromStereo	Contributed Libraries in Python>DepthFromStereo	Detecção de profundidade por análise de imagens estereoscópicas.	visão computacional (exa.ocv) uso de bibliotecas (dev.lib) gráficos raster (gra.img)

DilationAndErosion	Contributed Libraries in Python>DilationAndErosion	Manipulação de traços de desenho por meio de “dilatação” e “erosão”.	visão computacional (exa.ocv) uso de bibliotecas (dev.lib) gráficos raster (gra.img)
FaceDetection	Contributed Libraries in Python>FaceDetection	Detecção de rostos.	visão computacional (exa.ocv) uso de bibliotecas (dev.lib) gráficos raster (gra.img)
FilterImages	Contributed Libraries in Python>FilterImages	Filtros de imagem.	visão computacional (exa.ocv) uso de bibliotecas (dev.lib) gráficos raster (gra.img)
FindContours	Contributed Libraries in Python>FindContours	Detecção de contornos.	visão computacional (exa.ocv) uso de bibliotecas (dev.lib) gráficos raster (gra.img)
FindEdges	Contributed Libraries in Python>FindEdges	Detecção de bordas.	visão computacional (exa.ocv) uso de bibliotecas (dev.lib) gráficos raster (gra.img)
FindHistogram	Contributed Libraries in Python>FindHistogram	Demonstra a extração de histogramas de uma imagem.	visão computacional (exa.ocv) uso de bibliotecas (dev.lib) gráficos raster (gra.img)
HistogramSkinDetection	Contributed Libraries in Python>HistogramSkinDetection	Usa da extração de histogramas para detectar pele em uma imagem.	visão computacional (exa.ocv) uso de bibliotecas (dev.lib) gráficos raster (gra.img)
LiveCamTest	Contributed Libraries in Python>LiveCamTest	Teste de captura de vídeo.	visão computacional (exa.ocv) uso de bibliotecas (dev.lib) gráficos raster (gra.img)
cameraHUD	Contributed Libraries in Python>cameraHUD	Exemplo de como desenhar elementos na tela em posição fixa (em relação ao movimento causado pelo giro do observador), como em um "Heads Up Display".	visão do observador (gra.tri.cam) uso de bibliotecas (dev.lib)
HelloPeasy	Contributed Libraries in Python>HelloPeasy	Exemplo mínimo do uso da biblioteca, o arreste do mouse faz o observador "orbitar" em torno de dois cubos	visão do observador (gra.tri.cam) uso de bibliotecas (dev.lib)
ttstest	Contributed Libraries in Python>ttstest	[Não foi possível executar este exemplo. A biblioteca de que depende não estava mais disponível] É possível inferir que se trata de um exemplo de sintetização de fala (Text to Speech)	exemplos de aplicação (exa) uso de bibliotecas (dev.lib)