U.PORTO
FACULDADE DE BELAS ARTES
UNIVERSIDADE DO PORTO

INTRODUÇÃO À PROGRAMAÇÃO GRÁFICA (USANDO PROCESSING)

42-15879077| RF| © ROYALTY-FREE/CORBIS, VALVES AND PIPELINES

MANUAL DAS SESSÕES

22, 29 de Novembro e 06 de Dezembro de 2006

Pedro Amado, 2006-06-08
Versão 1.61 (beta). Actualizada em 2007-04-12.

Qualquer correcão ou sugestão:
pamado(at)fba.up.pt

Para mais informações:
http://users.fba.up.pt/~pamado/

RESUMO

O presente manual serve de apoio às sessões de formação em Introdução à programação Gráfica usando Processing.

A formação pretende fornecer aos alunos/formandos uma oportunidade para contactarem com conceitos de algoritmia e conceitos gerais sobre linguagens de programação. O aluno/formando será incentivado a estruturar o pensamento de forma a resolver problemas através da programação do computador. No final da formação, pretende-se que os alunos/formandos possuam as ferramentas necessárias para se iniciarem na programação gráfica usando ferramentas de código como Processing ou complementando programas como Adobe Flash e ou Director em *scripting*, isto porque os conhecimentos adquiridos permitirão abordar outras linguagens de programação.

Porquê usar código para realizar tarefas que são (aparentemente) mais fáceis de realizar de modo tradicional? Como demonstrado muito brevemente, o conceito, ou melhor o código encontra-se na raiz da maior parte das criações artísticas. Da arte conceptual ao código digital usado nas plataformas actuais é apenas uma manifestação sobre diferentes meios, a formalização de um processo racional.

Depois de uma introdução teórica à programação e algoritmia introduzindo os conceitos chave (principais componentes da programação) introduz-se o Processing enquanto ferramenta e aborda-se o léxico e sintaxe do programa com breves exercícios. São apresentados exemplos, acompanhados por uma exposição multimédia, de alguns trabalhos dos autores mais influentes no que diz respeito ao desenho digital. John Maeda, Bem Fry, Casey Reas, Mariuz Watz, Lia, Golan Levin são alguns dos autores abordados. Na secção da Prática I e Prática II, bem como no final da introdução, são realizados exercícios práticos tendo em vista exercitar os principais conceitos apreendidos: Pong – Léxico e Sintaxe; Jogo da Vida – Manipulação de dados e algoritmia; Máquina de Desenho – Estruturação e prototipagem rápida de uma aplicação que gera gráficos parametrizados pelo utilizador.

As notas incluídas neste documento são relativas aos apontamentos pessoais de Pedro Amado, Técnico Superior de Design (FBAUP) compilados a partir das notas das aulas do Engenheiro Jorge Cardoso (Escola das Artes, UCP), pesquisa na Web (apontadores no final) e de investigação pessoal que realiza desde 2001 tendo por base a experiência pessoal enquanto Designer Gráfico (não-programador!).
Ver nota de crédito.

Por fim, este documento/formação surge por iniciativa pessoal de Pedro Amado no âmbito das dunções enquanto Técnico de Design da FBAUP. Enquanto formação, é complementar à formação académica da comunidade, mas como verificado por experiência pessoal, é crucial num desenvolvimento pessoal mais completo.
Espero que seja útil a todos!

INDICE

ÍNDICE DE QUADROS E FIGURAS

APRESENTAÇÃO

FIGURA 1 - PEDRO AMADO (FOTOGRAFIA)

Pedro Amado
Técnico Superior de Design, FBAUP
Licenciado em Design de Comunicação, FBAUP

Aluno de Mestrado Multimédia, FBAUP (Finalizar a dissertação)
Estatuto de Director no Movimento de Tempos Livres MOCAMFE

Realiza pequenas experiências em desenho digital, programação e construção *websites* desde 2001.

Desenvolve Design Tipográfico nos tempos livres – www.typeforge.net

Actualmente a implementar o DesignLab, página de apoio ao Gabinete do Técnico de Design, FBAUP: http://users.fba.up.pt/~pamado/designlab

INTRODUÇÃO

PÚBLICO-ALVO (PROGRAMAÇÃO GRÁFICA)

Alunos de Design de Comunicação (Designers);
Alunos de Artes Plásticas (Artistas);
Pessoas interessadas em Multimédia Digital (Toda a gente).

42-17042453| RF|© ROYALTY-FREE/CORBIS, BUSINESSMAN USING LAPTOP IN AUDITORIUM

FIGURA 2 - AUDITÓRIO

REQUISITOS

Conhecimentos básicos em utilização de um computador;
Motivação

42-17501678| RF|© ROYALTY-FREE/CORBIS, BUSINESSMAN HITTING GOAL

FIGURA 3 - REQUISITOS

BREVE NOTA DE CRÉDITO

Este documento foi construído essencialmente com base nos materiais da cadeira Programação Multimédia (2º Semestre - 2005/2006) leccionada pelo Eng.º Jorge Cardoso da Universidade Católica Portuguesa, que teve a gentileza de autorizar o uso para fins de formação académica. Mais uma vez, obrigado.

http://teaching.jorgecardoso.org/

O QUE É PROGRAMAR?

Entre muitas coisas possíveis, programar, criar um programa é, na sua essência, criar ou executar um algoritmo, ou um jogo de algoritmos que são construídos para resolver ou simplificar um problema ou uma operação.

Computer programming (often simply programming or coding) is the craft of writing a set of commands or instructions that can later be compiled and/or interpreted and then inherently transformed to an executable that an electronic machine can execute or "run". Programming requires mainly logic, but has elements of science, mathematics, engineering, and — many would argue — art.

In software engineering, programming (implementation) is regarded as one phase in a software development process.

The earliest programmable machine (that is, a machine that can adjust its capabilities based upon changing its "program") can be said to be the Jacquard Loom, which was developed in 1801. The machine used a series of pasteboard cards with holes punched in them. The hole pattern represented the pattern that the loom had to follow in weaving cloth. The loom could produce entirely different weaves using different sets of cards.

(retirado de http://en.wikipedia.org/wiki/Programming)

Assim pode entender-se que programar é especificar um conjunto de instruções para serem seguidas por uma pessoa ou máquina. As instruções são uma espécie de comando ou “passos” que a pessoa ou máquina têm que cumprir.

Se estivermos a programar para pessoas, o conjunto de instruções a seguir irá ser muito diferente do que para as máquinas.

EXEMPLO

1. Ligar o computador
2. Esperar que o SO arranque
3. Aceder ao Go do Finder
4. Escolher a opção Connect to Server
5. Esperar pela janela de login
6. Inserir login e password
7. Escolher o share igual ao login

No entanto, é preciso ter em conta que estas instruções são demasiado vagas para serem usados por um computador. Logo na primeira instrução seria preciso dizer que computador ligar, como o ligar, o que ligara exactamente, etc...

FIGURA 4 - JACQARD, TEAR (WIKIPÉDIA)

ALGORITMO

Um programa é uma implementação concreta de um algoritmo.
Afinal o que é isto de algoritmo?
Como vamos ver mais à frente, a palavra mais simples para descrever um algoritmo é "receita" (como em culinária).

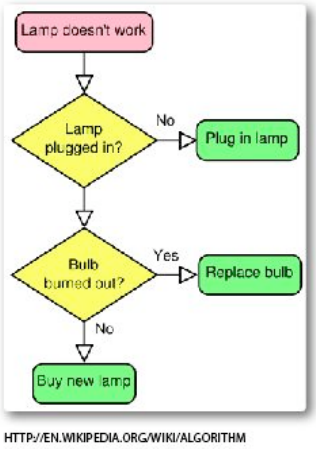


FIGURA 5 - ALGORITMO (WIKIPÉDIA)

Um algoritmo é um "conjunto de regras e operações que permitem resolver, num número finito de etapas, um problema" (http://www.infopedia.pt). A palavra algoritmo tem origem no nome do matemático persa Al-Khwarizmi - 780-850.

Um programa de computador é um "conjunto completo de instruções, em linguagem de código, que indica ao computador, passo a passo, como determinada tarefa deverá ser executada" (http://www.infopedia.pt). Dito de forma mais simples, um programa é uma implementação concreta de um algoritmo.

In mathematics and computing, an algorithm is a procedure (a finite set of well-defined instructions) for accomplishing some task which, given an initial state, will terminate in a defined end-state. The computational complexity and efficient implementation of the algorithm are important in computing, and this depends on suitable data structures.

Informally, the concept of an algorithm is often illustrated by the example of a recipe, although many algorithms are much more complex; algorithms often have steps that repeat (iterate) or require decisions (such as logic or comparison). Algorithms can be composed to create more complex algorithms.

The concept of an algorithm originated as a means of recording procedures for solving mathematical problems such as finding the common divisor of two numbers or multiplying two numbers. The concept was formalized in 1936 through Alan Turing's Turing machines and Alonzo Church's lambda calculus, which in turn formed the foundation of computer science.

Because an algorithm is a precise list of precise steps, the order of computation will almost always be critical to the functioning of the algorithm. Instructions are usually assumed to be listed explicitly, and are described as starting 'from the top' and going 'down to the bottom', an idea that is described more formally by flow of control.

(retirado de http://en.wikipedia.org/wiki/Algorithm)

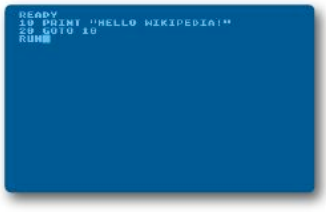
ALGORITMIA

Um programa, mais concretamente, um programa de computador é uma implementação correcta de um algoritmo. Normalmente os algoritmos (como no exemplo do que é programar) é descrito em linguagem natural, mas quando falamos de programas referimos-nos à esrita em código ou linguagem de programação tal como C, Java, Processing, Flash, PHP, HTML, etc.

LINGUAGENS DE PROGRAMAÇÃO

Um algoritmo pode ser descrito usando várias linguagens ou línguas:

- Português, podemos descrever um algoritmo como se estivéssemos a falar com outra pessoa;
- Pseudo-código, é uma linguagem entre a linguagem natural e uma linguagem de programação (permite-nos manter a legibilidade, ao mesmo tempo que nos aproxima da linguagem que vamos usar no programa de computador);
- Fluxogramas, descrição gráfica de um algoritmo;
- Linguagem de Programação (ex: Processing, Java, C, C++, Basic, PHP).



SCREENSHOT OF ATARI BASIC, ONE OF THE FIRST BASIC LANGUAGES FOR SMALL COMPUTERS.

FIGURA 6 - BASIC (WIKIPÉDIA)

Como sabemos existem várias linguagens. Algumas específicas de determinados ambientes ou plataformas (Flash), outras são universais (PHP)

- **HTML**:
  Linguagem de *Script* (não é verdadeiramente programação)
  Interpretado em tempo real (*render*)

Universal, Multiplataforma

- **PHP**:
  Linguagem de *Script* (não é verdadeiramente programação)
  Interpretado (executado pelo servidor)
  Universal, Multiplataforma

- **ActionScript**:
  Linguagem de *Script* (não é verdadeiramente programação)
  Multiplataforma mas dependente da aplicação/plataforma Flash/FlashPlayer.
  Compilado+Interpretado (Player)

- **Processing**:
  Baseado em JAVA
  Simplificado
  Multiplataforma mas dependente da aplicação/plataforma JAVA
  Compilado+Interpretado (executado pelo motor de JAVA)

- **JAVA**:
  Multiplataforma (independente)
  Compilado+Interpretado (executado pelo motor de JAVA)

- **C/C++:**
  Linguagem universal que tem que ser adaptadaàs plataformas tecnológicas
  Compilado (executado pelo SO ou pelo processador)

Java

The Processing environment is written in Java. Programs written in Processing are also translated to Java and then run as Java programs. Programs written in Java and Processing will usually run faster than programs based on scripting languages like ActionScript and Lingo, which is important for many graphics applications.

Large distinctions between Processing and Java are the Processing graphics library and a simplified programming style that doesn't require users to understand more advanced concepts like classes, objects, or animation and double-buffering (while still making them accessible for advanced users). Such technical details must be specifically programmed in Java, but are integrated into Processing, making programs shorter and easier to read.

Actionscript
ActionScript is the language written for Macromedia's Flash software. Flash was originally created as web animation software and ActionScript is integrated into a timeline representation. ActionScript is based on JavaScript and Processing is built on Java, so there are many similarities between these two languages.

Lingo
Lingo is the language written for Macromedia's Director software. Director was the dominant environment for designers and artists making CD-ROM projects, but has declining in popularity during the web era due to the success of Flash. It is still one of the most commonly used environments and it has excellent libraries of code for extending its functionality.

Python
Python is considered to be an excellent teaching language because of its clear syntax and structure. Python is typically used for non-graphic applications. It doesn't have a native graphics library, but graphics applications may be created by using graphics toolkits, some of which are cross-platform.

Design By Numbers
Design By Numbers (DBN) was developed for teaching general programming concepts to artists and designers with no prior programming experience. DBN is an extremely minimal language and environment, thus making it easy to learn but limited in its potential for creating advanced applications.

(do http://processing.org)

Basicamente a diferença que existe entre linguagem de programação e de *scripting* é a necessidade da segunda estar dependente de um programa externo para a correr – *players,* máquinas virtuais, etc. Enquanto a primeira é preparada para correr directamente na máquina de destino, uma linguagem compilada para ser lida naturalmente pelo executor final.

The programming language that a computer can directly execute is machine language (sometimes called "machine code"). Originally all programmers worked out every detail of the machine code, but this is hardly ever done anymore. Instead, programmers write source code, and a computer (running a compiler, an interpreter or occasionally an assembler) translates it through one or more translation steps to fill in all the details, before the final machine code is executed on the target computer. Even when complete low-level control of the target computer is

required, programmers write assembly language, whose instructions are mnemonic one-to-one transcriptions of the corresponding machine language instructions. People that do the programming are called computer programmers. Programmers must write, test, and give instructions to programs to perform a function.

Different programming languages support different styles of programming (called programming paradigms). Common languages are C++ and Java but there are many more. Part of the art of programming is selecting one of the programming languages best suited for the task at hand. Different programming languages require different levels of detail to be handled by the programmer when implementing algorithms, often in a compromise between ease of use and performance (a trade-off between "programmer time" and "computer time").
In an almost evolutionary order

- *FORTRAN is a general-purpose, procedural, imperative programming language that is especially suited to numeric computation and scientific computing. Originally developed by John Backus of International Business Machines Corporation (IBM) in the 1950s for scientific and engineering applications.*
- *C is a compiled procedural, imperative programming language made popular as the basis of Unix.*
- *Shell scripting, in particular using either a variant of the Bourne shell or the C shell, is popular among UNIX hackers. Although the exact implementation varies among different shells, the core principles remain intact: only providing facilities for program flow (also seen in C) while placing emphasis on using external programs, although most shells feature some other functions internally, known as builtins. Shell scripting is used primarily in systems administration, especially where tasks need to be automated and run at specific times (backups, monitoring, file system maintenance, among others). It is also used as a tool for rapid prototyping when the exact design of a program is not yet clear enough for a full implementation, often in a compiled language like C. Whilst most shell scripts are relatively simple it is possible to create complex and powerful programs in many implementations.*
- *SMALLTALK invented 1971 was the most important fundament for object oriented programming. It started a new programming paradigm, which influenced the whole art of programming significantly. Smalltalk is a pure object oriented (OO) language with a minimalistic syntax. This is possible because very consequent mostly everything is done inside the class library. Even standard control structures are implemented in the class library. There exists nothing else than objects. Not many other OO-languages have this clearness and simplicity.*
- *Pascal is a general-purpose structured language named after the famous mathematician and philosopher Blaise Pascal. It was very popular during the '80s and '90s. Whilst popularity of Pascal itself has waned (its principal use is in teaching of programming) languages derived from it (such as Object Pascal) are still in use.*

- *BASIC (Beginner's All purpose Symbolic Instruction Code) was invented by John Kemeny and Thomas Kurtz of Dartmouth College. It became the most widely used language when microcomputers first hit the market, in the '70s and '80s. Many dialects of BASIC have been produced. Because early dialects lacked important features such as strong data typing, procedures and functions, BASIC was typically seen as a language for learning programming and prototyping rather than for enterprise development. This is not true today since many BASIC compilers offer all of the structured programming advantages as other languages.*
- *Visual Basic is Microsoft's implementation of BASIC as an integrated development environment.*
- *C++ is a compiled programming language based on C, with support for object-oriented programming. It is one of the most widely-used programming languages currently available. It is often considered to be the industry-standard language of game development, but is also very often used to write other types of computer software applications. C++ was developed by Bjarne Stroustrup and was based on the programming language C. C++ retains the syntax and many familiar functions of C, but also adds various concepts associated with other programming paradigms, such as classes.*
- *C# is an object-oriented programming language developed by Microsoft as part of their .NET initiative. C# has a procedural, object oriented syntax based on C++ that includes aspects of several other programming languages (most notably Delphi, Visual Basic, and Java) with a particular emphasis on simplification (less symbolic requirements than C++, less decorative requirements than Java). Though developed by Microsoft, C# is standardized by the ECMA and International Standards Organization.*
- *Java is an object oriented interpreted programming language. It has gained popularity in the past few years for its ability to be run on many platforms, including Solaris, Linux, Microsoft Windows, Mac OS and other systems. It was developed by Sun Microsystems.*
- *Lisp is a family of functional, sometimes scripted, programming languages often used in AI.*
- *Object Pascal is an object oriented derivative of Pascal. Developed by Apple Computer in the 1980s, today it is primarily known as the language of Borland Delphi. It is also used with Kylix, Chrome and various open source object pascal implementations, such as FreePascal.*
- *The Delphi Language is the name of the Object Pascal derivative that is the primary language of later versions of Borland's Delphi Studio integrated development environment.*
- *Perl one of the first widely-used, cross-platform, interpreted languages, Perl owes much of its syntax and semantics to C and the Unix shell.*
- *Python is an interpreted, dynamically typed, object-oriented language that has some unique syntax features (like the significance of indentation). Though strictly speaking an interpreted language, its usage domain follows that of Java and C#.*
- *Ruby is very much like Python, though it features some constructs more closely related to those found in Perl and Lisp. It is also the basis of a very popular web application framework named Ruby on Rails.*
- *PHP is a newer programming language with focus on web design. It has a C-like syntax.*

(retirado de http://en.wikipedia.org/wiki/Programming e de http://en.wikipedia.org/wiki/Programming_language_timeline).

COMPILAR E EXECUTAR

No início da era da programação, os programas eram carregados directamente nas máquinas em "instruções máquina". À medida que as capacidades das máquinas aumentaram e que as instruções se complexificaram houve a necessidade de criar linguagens mais fáceis de usar para os programadores. Daí a necessidade de criar "tradutores" das linguagens dos programadores para as "linguagens máquina" – os compiladores. Um compilador não é mais do que um simples programa que converte o programa escrito em linguagem de alto nível (programador) para código máquina de forma a poder ser executado pelo *hardware* vulgo computador. Basta pensar nos primeiros teares mecânicos e nos actuais computorizados.

O programa escrito pelo programador é designado por código-fonte (*source*).

A maior parte dos programas tem de ser compilado para linguagem máquina
– O código que nós escrevemos é traduzido para código-máquina
– Só depois o computador (processador) pode executar o programa
Ex.: C, C++

Algumas linguagens são interpretadas
– Não é necessário compilar
– O programa associado lê as nossas instruções e traduz à medida que o programa é executado
– Não é o processador que executa directamente
Ex.: Java, Processing, Actionscript

DA ARTE AO CÓDIGO E DO CÓDIGO À ARTE – O DESENHO COMO UM PROCESSO RACIONAL

CONCEPT ART

Concept art is a form of illustration where the main goal is to convey a visual representation of a design, idea, and/or mood for use in movies, video games, or comic books before it is put into

the final product. This is a relatively new designation popularized by artists working in the automobile and video games industries. This term has been in use since the 1930's by the traditional animation industry who was describing drawn or painted images which illustrate the look, feel, design, colors, etc...of the animated movie to be made. Concept art is also referred to as "visual development" in traditional animation. The term was later adopted by the games industry. These illustrations became necessary for developed visual properties.

Concept art is the preliminary visual statement made in entertainment production. Before characters, worlds, or items are created, images are made to show what these things should look like, often based on a writers description. Concept Art is the illustrated visualization of ideas.

Conceptual art, sometimes called idea art, is art in which the concept(s) or idea(s) involved in the work take precedence over traditional aesthetic and material concerns. In some cases, Conceptual art may not entail any art object per se, but instead manifest solely as documentary evidence for an "art idea". In other, less extreme cases, Conceptual art may involve the construction of images and objects in a manner that frees the artist from their traditional role as a maker of aesthetic decisions. To give an example, many of the works of the artist Sol Lewitt may be constructed by anyone simply by following a set of written instructions.[1] This method was fundamental to Lewitt's definition of Conceptual art, the first to appear in print:

CONCEPTUAL ART

In conceptual art the idea or concept is the most important aspect of the work. When an artist uses a conceptual form of art, it means that all of the planning and decisions are made beforehand and the execution is a perfunctory affair.

***The idea becomes a machine that makes the art.** – Sol LeWitt, "Paragraphs on Conceptual Art", Art Forum, 1967.*

For the layman, this quotation highlights a key difference between a conceptualist installation and a traditional work of art - that the conceptualist work may involve little or no skill in its execution, whereas art in its traditional sense is distinguished by requiring a skill in its execution beyond the ability of the average person.

Conceptual art emerged as a movement during the 1960s. In part, it was a reaction against formalism

Conceptual Art - late 1960s New York, peak in 1970s, still evident today

A. *Roots and precursors*
   1. *Minimalism and Pop ironies (Stella, Warhol)*
   2. *Fluxus attitudes (George Brecht)*
   3. *Transcendental signifiers (Yves Klein, Piero Manzoni)*
   4. *Above all, the growing influence of Duchamp*

B. *Main characteristics of Conceptual Art*
   1. *Idea-based (form is incidental) – e nas tecnologias digitais? Talvez não incidental, mas definitivamente forma separada do conceito (instruções/dados)*
   2. *Anti-heroic and impersonal stance of artist (bureaucratic?)*
   3. *Impersonal execution (if at all), often industrial, often delegated*
   4. *Language a crucial component*
   5. *"The work can be made. The work does not have to be made to be art." (LeWitt)*

JOSEPH KOSUTH

The 'value' of particular artists after Duchamp can be weighed according to how much they questioned the nature of art."

One of his most famous works is "One and Three Chairs", a visual expression of Plato's concept of The Forms. The piece features a physical chair, a photograph of that chair, and the text of a dictionary definition of the word "chair". All three representations are merely physical abstractions of the one true idea of the chair, thus the piece is both the three physical representations of a chair, and the one universal notion of a chair. In this and other, similar works, Kosuth forwards tautological statements, where the works literally are what they say they are.

JOSEPH KOSUTH, ONE AND THREE CHAIRS (1965) TONY GODFREY, CONCEPTUAL ART, LONDON: 1998

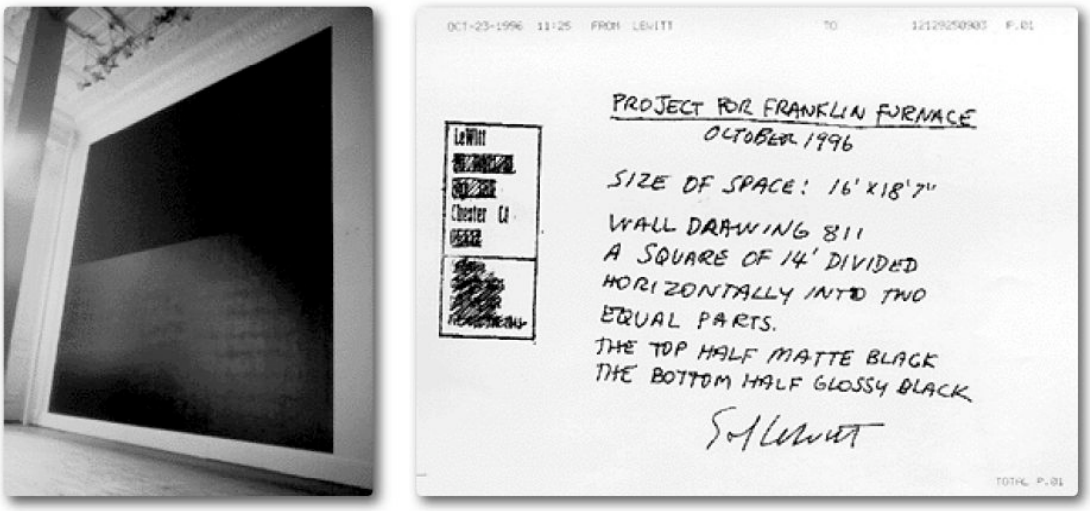
FIGURA 7 - JOSEPH KOSUTH, ONE AND THREE CHAIRS (1965)

'One and Three Chairs' (1965), by the U.S. artist, Joseph Kosuth. A key early work of Conceptual art by one of the movement's most influential artists. Tony Godfrey, conceptual Art, London: 1998 1965

SOL LEWITT

"The idea is the machine that makes the art" - Sol LeWitt

In a seminal text in written in 1967 titled "Paragraphs on Conceptual Art," LeWitt emphasized his view of art: "No matter what form it may finally have it must begin with an idea," and, "When an artist uses a conceptual form of art, it means that all of the planning and decisions are made beforehand and the execution is a perfunctory affair. The idea becomes a machine that makes the art."



LEWITT WALL DRAWING 811 AS DRAFTED AT FRANKLIN FURNACE OCT.1996" + INSTRUCTIONS FAXED BY LEWITT TO FRANKLIN FURNACE FOR DRAFTERS OF WALL DRAWING 811

FIGURA 8 - LEWITT WALL DRAWING 811 AS DRAFTED AT FRANKLIN FURNACE OCT.1996; INSTRUCTIONS FAXED BY LEWITT TO FRANKLIN FURNACE FOR DRAFTERS OF WALL DRAWING 811

“Banal ideas cannot be rescued by beautiful execution.
It is difficult to bungle a good idea.”

LÓGICA E SINTAXE

Um programa pode ser analisado segundo duas perspectivas:
–Sintaxe: o código está de acordo com as regras gramaticais da linguagem de programação utilizada?

–Lógica: o código executa aquilo que nós pretendemos?

Um programa pode estar sintacticamente correcto (regra geral o compilador detecta estes erros), mas logicamente estar errado.

Nível de Detalhe
1. Ligar o computador
2. Esperar que o SO arranque
3. Aceder ao Go do Finder
4. Escolher a opção Connect to Server
5. Esperar pela janela de login
6. Inserir login e password
7. Escolher o share igual ao login

Como se liga o computador??

Passo 1:
1. Procurar o botão com símbolo
2. Pressionar o botão
3. Se o computador n‹o ligou então
4. Verificar se os cabos de alimentação estão ligados
5. Pressionar novamente o botão

O passo Ligar o computador é, de facto, constituído por várias acções mais pequenas
Ao escrever um programa devemos tentar encontrar instruções de mais alto nível e agrupá-las;
Torna o programa mais fácil de ler e entender

1. Pressionar a tecla “p”
2. Pressionar a tecla “e”
3. Pressionar a tecla “d”
4. Pressionar a tecla “r”
5. Pressionar a tecla “o”

Podem ser agrupadas em
1. EscreverPedro

Quando lermos o programa sabemos logo que a instrução significa escrever a palavra “pedro”;

Se quisermos saber em detalhe como isso se faz podemos analisar o conjunto de instruções correspondente;

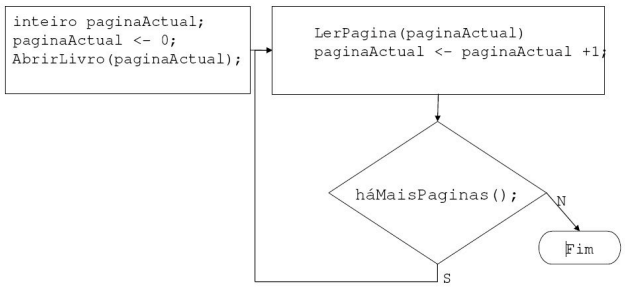
Por exemplo, um algorimo para alguém a ler um livro poderia ser descrito das seguintes formas:

Algoritmo em Português:
1. Abrir o livro na primeira página;
2. Ler uma página;
3. Passar à página seguinte;
4. Repetir os passos 2 e 3 até acabar o livro;

Algoritmo em pseudo-código:

```
Inteiro paginaActual;
    paginaActual <- 0;
    AbrirLivro(paginaActual);
    Fazer
          LerPagina(paginaActual);
          paginaActual <- paginaActual +1;
      Enquanto háMaisPaginas();
```

Algoritmo em Fluxograma:



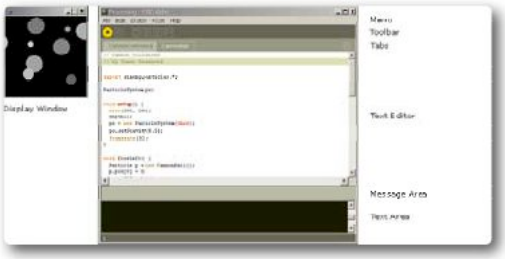
EXERCÍCIO

Pensar num programa que possa ser executado por uma pessoa da sala.

© ROYALTY-FREE/CORBIS, YOUNG WOMAN TRAINING HER DOG TO SIT

PRIMEIROS PASSOS

PROCESSING



SPLASH SCREEN DE PROCESSING E APLICAÇÃO EM PLENO USO

FIGURA 9 - PROCESSING SPLASH SCREEN

Processing is an open source programming language and environment for people who want to program images, animation, and sound. It is used by students, artists, designers, architects, researchers, and hobbyists for learning, prototyping, and production. It is created to teach fundamentals of computer programming within a visual context and to serve as a software sketchbook and professional production tool. Processing is developed by artists and designers as an alternative to proprietary software tools in the same domain.

The beta software for Processing 1.0 was released 20 April 2005 and can be downloaded here. Bug fixes are being made as we head toward the 1.0 release. Processing is free to download and available for GNU/Linux, Mac OS X, and Windows. Please help in releasing version 1.0!

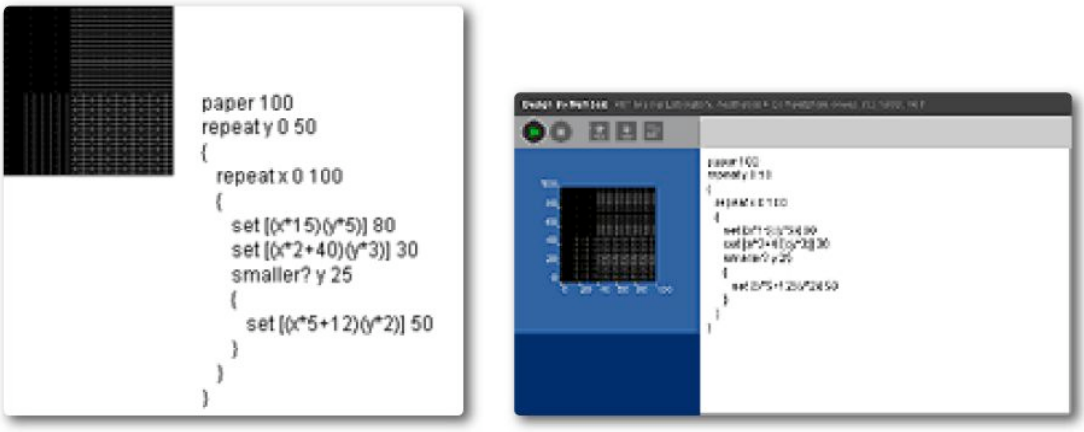
Processing is an open project initiated by Ben Fry (Broad Institute) and Casey Reas (UCLA Design | Media Arts). Processing evolved from ideas explored in the Aesthetics and Computation Group at the MIT Media Lab.

(do http://processing.org)

DESIGN BY NUMBERS

Design By Numbers was created for visual designers and artists as an introduction to computational design. It is the result of a continuing endeavor by Professor John Maeda to teach the "idea" of computation to designers and artists. It is his belief that the quality of media art and design can only improve through establishing educational infrastructure in arts and technology schools that create strong, cross-disciplinary individuals.

DBN is both a programming environment and language. The environment provides a unified space for writing and running programs and the language introduces the basic ideas of computer programming within the context of drawing. Visual elements such as dot, line, and field are combined with the computational ideas of variables and conditional statements to generate images.



DESIGN BY NUMBERS DE JOHN MAEDA EM PLENO USO

FIGURA 10 - DESIGN BY NUMBERS (DBN) A SAMPLE PROGRAM AND ITS RESULT.; DESIGN BY NUMBERS (DBN) PROGRAMS ARE WRITTEN IN THE RIGHT HALF OF THE ENVIRONMENT AND DISPLAYED IN THE LEFT.

DBN is not a general purpose programming language like C or Java, but was designed to familiarize people with the basic concepts of computational media. Studying DBN is a first step to take–not a final step. Its advantages are:

1. free to use and multiplatform
2. easy to understand syntax designed for beginners
3. immediately accessible on the web

There are three primary components to the Design By Numbers System:

1. Design By Numbers core software
This software contains the complete DBN environment and can be viewed from the web or downloaded to an individual's computer.
2. Design By Numbers book published by MIT Press
Takes readers step by step through the DBN language with explanation of examples.
3. Design By Numbers Courseware
A flexibly designed website generator built for educators who want to use DBN to teach computational design.

(retirado de http://dbn.media.mit.edu/)

WHY DO IT THE HARD WAY?

Because an artist is expected to understand his/her medium. A painter who refuses to be bothered with the technical details of painting -- like making sure the canvas is properly stretched over the frame -- does not get as far as an artist who understands the essence of craft. And yet in digital art, this separation from the basic issues of creation is not only tolerated, but re-enforced through interfaces that abstract the internal processes of the computer. Hence, many digital artists create a fair amount of work with very limited understanding of the medium.

Because this is true digital design. The programs we will be making in DBN bring us closer to the real workings of the computer. We will be creating compositions out of the basic graphic element of the computer, the pixel ou melhor os dados que o representam!!!. We start with the pixel like the traditional designer would start with pen and paper. We will build a basic language of line and shape and value. We begin to get a feel for how our calculations effect the visuals we are creating. We start to understand the relationships between the numbers and the images. We begin to have a conversation with these numbers, instead of using them as slaves to recreate images from the physical world. We even begin to learn from them.

Because we appreciate elegance. Put simply, programming beauty is like mechanical beauty: the fewer parts, the more elegant the machine. It is about the balance between simplicity and effect.

(retirado de http://dbn.media.mit.edu/)

WHY DO PEOPLE HAVE A FEAR OF PROGRAMMING?

[1] Stereotype of programers. Think war games or tron. People think of programming as a sort of reclusive breed of geeky guys. Maybe there's a stereotype for every profession. At any rate, know that programmers come in all shapes and colors and that you don't have to be a geeky guy to be a part of this community. Programmers are, as it turns out, incredibly helpful and love to share what they've learned. There are a ton of sites online and off where programmers learn from and help each other. Part of this fear involves the idea of doing solitary work late at night. In fact, the best way to do this homework is in groups, working together, sharing successes and failures. Unless your Rich Keeler and you're in Japan.

[2] Walls of books at bookstores. One of the most intimidating things about programming is finding the right resources. Part of the problem is the multitude of books available to help you "teach" yourself programming. These books which are as big as phonebooks and look incredibly tough are scary for the beginner. Part of the trick is learning how to pick a good book. The nice thing about DBN is that there is only one book, so you don't have to be intimidated at the choices. Later, the books in the bookstore look more like bread in the supermarket, and picking the right book becomes less stressful. For now, don't let the computer section frighten you.

[3] fear of math, math class. We probably all had the absent minded geometry teacher who left the chalk erasers in his pockets and had MC Escher drawings on the wall. For some people, this math world was a world that seemed cold and sterile. For others, it was a very powerful world of abstraction and exploration. Part of that fear that a lot of non-math folks had informs the fear of programming. DBN was designed to address that fear -- keeping numbers between 0 - 100, not pushing complex math ideas and so on. Stay fearful, but know that programming is not (but can be for sure) as complicated as most high school math classes..

[4] fear of unknown, hieroglyphics, There is a fear of foreign looking material. Because it looks difficult and doesn't make sense, we believe, then, it must be complicated and hard. In fact, the majority of code ideas are essentially very simple and can be boiled down to simple english (!) sentences. The computer is good at doing very very small things. It just can do them very fast.

The majority of programming is harnessing that brute force to do tons of things in a short amount of time. Coding is a lot closer to a natural way of talking and thinking then we commonly believe.

(retirado de http://dbn.media.mit.edu/)

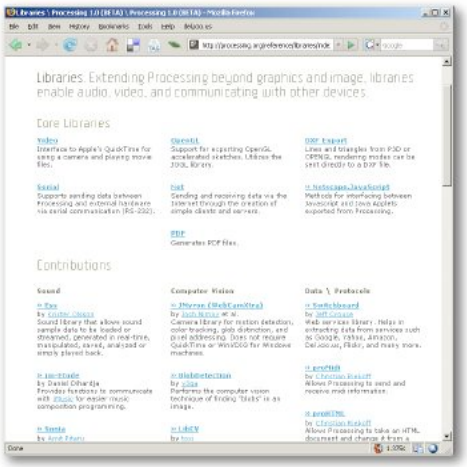
CONFIDENCE

Learning to program is hard work. Take small steps. Don't be afraid to break things or create things that don't work. We learn a lot by failure as well as success. Good luck!

(retirado de http://dbn.media.mit.edu/)

MOTIVOS PARA USAR O PROCESSING

1. Open Source;
2. Versão Beta (por enquanto)
   Situação ideal para começar a aprender;
3. Como se aprende?
   Há diversos websites dedicados à programação, alguns especialmente dedicados ao Processing – ver a secção de links;
4. Corre em que plataformas?!
5. Porquê a linguagem Java ou parecida?
   a. para simplificar;
   b. porque se adequa ao processo de ensino;
   c. é uma boa transição para outras linguagens;
   d. permite criar rapidamente aplicações independentes ou *applets* para a Web;
6. Não foi desenhado para substituir o Flash (sem timeline);
7. Compatível e Extensível



BIBLIOTECAS DISPONIVEIS NO SITE PARA AMPLIAR A ACÇÃO DO PROCESSING

FIGURA 11 - BIBLIOTECAS DE EXTENSÃO DO PROCESSING

As ideias que suportam o Processing são uma extensão das explorações do Aesthetics and Computation Group (ACG - http://acg.media.mit.edu/) dirigido pelo professor John Maeda no laboratório do MIT de 1996 – 2003(?). Design By Numbers de John Maeda (http://dbn.media.mit.edu/) é um antecessor directo do Processing. Este projecto originou em Outono de 2001 no ACG onde Ben Fry era um estudante de Doutoramento no Interaction Design Institute em Itália e onde Casey Reas era Professor. O projecto é actualmente dirigido por Ben Fry (Broad Institute) e Casey Reas (Departamento de Design | Media Arts do UCLA).

COMO COMEÇAR?

Duplo clique na aplicação Processing e seleccionar algum dos exemplos no Menu exemplos: File > Open > Examples.
Clicar em 'Run' (play)

Overview

The Processing Environment (Integrated Development Environment or IDE) is a simple and usable editor for writing and running programs. When the "run" button is clicked, the program compiles, the display window opens and the software runs inside. The toolbar provides functionality for running and stopping programs, creating a new sketch, opening, saving, and exporting. Additional commands are available through the menus. The message area gives feedback while saving and exporting and also shows the locations of errors when programs are

compiled. The text area presents messages and can be written to with the print() and println() programming functions.

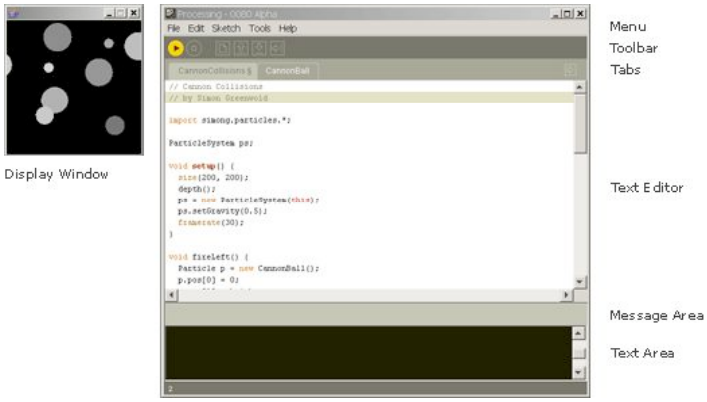


FIGURA 12 - IDE E APLICAÇÃO A CORRER

The toolbar provides access to six basic commands of Processing: Run, Stop, New, Open, Save, Export.

***Run**: Runs the code (compiles the code, opens the display window, and runs the program inside)*

***Stop**: Terminates a running program, but does not close the display window. Hitting the Escape (Esc) key also stops the program.*

***New**: Creates a new sketch (project)*

***Open**: Select and load a pre-existing sketch. A menu opens and you may choose from your own sketchbook, examples, or you can open a sketch from anywhere on your computer or network.*

***Save**: Saves the current sketch into its current location. If you want to give the sketch a different name, select Save As from the File menu.*

***Export**: Exports the current sketch into the sketchbook as a Java Applet embedded in an HTML file. The directory containing the files is opened. Click on the index.html file to load the software in the default web browser. There is more information about exporting below.*

Top Menu
Additional commands are found within the five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive which means only those items relevant to the work currently being carried out are available.
File

***New** (Ctrl+N)*
Creates a new sketch, named as the current date is the format "sketch_YYMMDDa".

Sketchbook
Gives the option to open a sketch from anywhere on the local computer or network, the sketchbook, or to open an example.

***Save** (Ctrl+S)*
Saves the open sketch in it's current state.

***Save as...** (Ctrl+Shift+S)*
Saves the currently open sketch, with the option of giving it a different name. Does not replace the previous version of the sketch.

***Export** (Ctrl+E)*
Be default, exports a Java Applet and creates and embeds it into an HTML file. After the files are exported, the directory containing the exported files is opened. There is more information about exporting below.

***Export Application** (Ctrl+Shift+E)*
Exports as a Java application as an executable file. Opens the directory containing the exported files.

***Page Setup** (Ctrl+Shift+P)*
(Not working yet)

***Print** (Ctrl+P)*
(Not working yet)

Preferences *(Ctrl+,)*
Allows you to change some of the ways Processing works.

Quit *(Ctrl+Q)*
Exits the Processing Environment and closes all Processing windows.
Edit

The Edit menu provides a series of commands for editing the Processing files.

Undo *(Ctrl+Z)*
Reverses the last command or the last entry typed. Cancel the Undo command by choosing Edit » Redo.

Redo *(Ctrl+Y)*
Reverses the action of the last Undo command. This option is only available, if there has already been an Undo action.

Cut *(Ctrl+X)*
Removes and copies selected text to the clipboard (an off-screen text buffer)

Copy *(Ctrl+C)*
Copies selected text to the clipboard.

Paste *(Ctrl+V)*
Inserts the contents of the clipboard at the location of the cursor, and replaces any selected text.

Select *All (Ctrl+A)*
Selects all of the text in the file which is currently open in the text editor.

Find *(Ctrl+F)*
Finds an occurance of a text string within the file open in the text editor and gives the option to replace it with a different text.

Find Next *(Ctrl+G)*
Finds the next occurance of a text string within the file open in the text editor.
Sketch

Run *(Ctrl+R)*
Runs the code (compiles the code, opens the display window, and runs the program inside)

Present *(Ctrl+Shift+R)*
Runs the code in the center of the screen with a neutral background. Click the "stop" button in the lower left to exit the presentation.

Stop
If the code is running, stops the execution. Programs written with the Basic Mode or using the draw() structure are stopped automatically after they draw.

Add File
Opens a file navigator. Select an image, font, or other media files to add it to the sketches "data" directory.

Import Library
Adds the necessary import statements to the top of the current sketch. For example, selecting Sketch » Import Library » video adds the statement "import processing.video.;" to the top of the file. These import statements are necessary for using the Libraries.*

Show Sketch Folder
Opens the directory for the current sketch.

Tools

Auto Format
Attempts to format the code into a more human-readable layout. Auto Format was previously called Beautify.

Create Font...
Converts fonts into the Processing font format and adds to the current sketch. Opens a dialog box which give options for setting the font, it's size, if it is anti-aliased, and if all characters should be generated. If the "All Characters" options is selected, non-English characters such as ü and Å are generated, but the font file is larger in size. The amount of memory required for the font is also determined by the size selected. Processing fonts are textures, so larger fonts require more image data.

Archive Sketch

Archives a copy of the current sketch in .zip format. The archive is placed in the same directory as the sketch.
Help

Environment
Opens the reference for the Processing Environment in the default Web browser.

Reference
Opens the reference in the default Web browser. Includes reference for the language, programming environment, libraries, and a language comparison.

Find in Reference *(Ctrl+Shift+F)*
Select a word in your program and select "Find in Reference" to open that reference HTML page.

Visit Processing.org (Ctrl+5)
Opens default Web browser to the Processing.org homepage.

About Processing
Opens a concise information panel about the software.
Top Sketchbook

All Processing projects are called sketches. Each sketch has it's own directory (folder) and inside there is the main program file which has the same name as the sketch. For example, if the name of the sketch is "Sketch_123", the directory for the sketch will be called "Sketch_123" and the main file will be called "Sketch_123.pde".

Sketches need other directories inside to contain additional media files and code libraries. When a font or image is added to a sketch by selecting the command "Add File..." from the "Sketch" menu, a "data" directory is created. All images, fonts, and other data/media files loaded within the sketch must be in this directory. Additional code libraries must be placed within a directory entitled "code". When a sketch is exported, all files from the "data" and "code" directories are exported into into a single .jar file with the same name as the sketch. For example, if the sketch is named "Sketch_123", the exported file will be called "Sketch_123.jar"

Sketches are all kept in the Processing directory, which will be in different places on your computer or network, depending if you use PC, Mac, or Linux and how you have your preferences set. To locate this directory, select the "Preferences" option in the "File" menu.

It is possible to have multiple program files in one sketch. These can be Processing text files (the extension .pde) or Java files (the extension .java). To add a new file, click on the arrow to the right of the file tabs. You can write functions and classes in new .pde files and you can write any Java code in files with the .java extension.
Top Exporting

Exporting creates a version of the sketch that can run within a Web browser. When code is exported from Processing it is changed into Java code and then compiled as a Java Applet. When a project is exported, a series of files are written to an "applet" directory which is created within the primary sketch directory. If the sketch is called "Sketch_123", the applet directory contains for following:

index.html
HTML file with the applet embedded and a link to the source code and the Processing homepage. Double-click to open this file in the default web browser.

Sketch_123.jar
Java Archive containing all necessary files for the sketch to run. Includes the Processing classes as well as those custom to the sketch and will also include media files (such as images) if they are a part of the sketch.

Sketch_123.java
The Java file generated by the pre-processor from the PDE file. This is the actual file which is compiled into the Applet by Jikes, the Java Compiler used in Processing.

Sketch_123.pde
The original program file. It is linked from the index.html file.

Every time a sketch is exported, all of the above files are written from scratch. Any changes made the the index.html file are lost.

When a sketch is exported, all files from the "data" and "code" directories are packed into a single JAR file, a Java Archive. Images and libraries not needed for the applet should be deleted before exporting to keep the files size small. For example, if there are many extra images in the "data" directory, they will be added to the JAR file, thus needlessly increasing the file size of the program. JAR files can be opened with programs such as WinZip so individual files can be seen and extracted.

In the future, it will be possible to export applications. This will allow programs to run without the Processing environment, to run at full screen, and to freely access data which is restricted by Applets (such as loading images through the Internet).

Top Coordinates

Processing uses a Cartesian coordinate system with the origin in the upper-left coer. If your program is 320 pixels wide and 240 pixels high, coordinate [0, 0] is the upper-left pixel and coordinate [320, 240] is in the lower-right. The last visible pixel in the lower-right corner of the screen is at position [319, 239] because pixels are drawn to the right and below the coordinate.

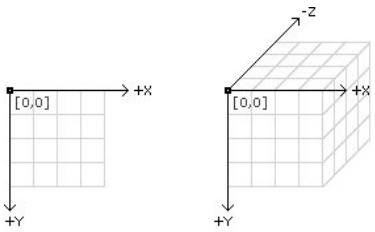


FIGURA 13 - SISTEMA DE COORDENADAS

Processing can also simulate drawing in three dimensions. At the surface of the image, the z-coordinate is zero, with negative z-values moving back in space. When drawing in simulated 3D, the "camera" is positioned in the center of the screen.
Top Programming Modes

Processing allows people to program at three levels of complexity: Basic Mode, Continuous Mode, and Java Mode. People new to programming should begin with the Basic Mode to learn about coordinates, variables, and loops before moving to Continuous and Java modes.

Basic

This mode is used drawing static images and learning fundamentals of programming. Simple lines of code have a direct representation on the screen. The following example draws a yellow rectangle on the screen:

```
size(200, 200);
background(255);
noStroke();
fill(255, 204, 0);
```

```
rect(30, 20, 50, 50);
```

Continuous

This mode provides a setup() structure that is run once when the program begins and a draw() structure which by default continually loops through the code inside. This additional structure allows writing custom functions and classes and using keyboard and mouse events.

This example draws four circles on the screen and utilizes a custom function called circles(). The circles() function is not a part of the Processing language, but was written for this example. The code in draw() only runs once because noLoop() is called in setup().

```
void setup()
{
  size(200, 200);
  noStroke();
  background(255);
  fill(0, 102, 153, 204);
  smooth();
  noLoop();
}

void draw()
{
  circles(40, 80);
  circles(90, 70);
}

void circles(int x, int y) {
  ellipse(x, y, 50, 50);
  ellipse(x+20, y+20, 60, 60);
}
```

This example draws rectangles that follow the mouse position (stored in the system variables mouseX and mouseY). Thedraw() section runs forever until the program is stopped, thus creating the potential for motion and interaction.

```
void setup()
{
  size(200, 200);
  rectMode(CENTER);
  noStroke();
  fill(0, 102, 153, 204);
}
void draw()
{
  background(255);
  rect(width-mouseX, height-mouseY, 50, 50);
  rect(mouseX, mouseY, 50, 50);
}
```

Java

This mode is the most flexible, allowing complete Java programs to be written from inside the Processing Environment. Writing in Java Mode removes the limitations of the Processing Libraries and gives access to the full Java programming language.

```
public class MyDemo extends PApplet {
  void setup()
  {
    size(200, 200);

    rectMode(CENTER);
    noStroke();
    fill(0, 102, 153, 204);
  }

  void draw()
  {
    background(255);
    rect(width-mouseX, height-mouseY, 50, 50);
    rect(mouseX, mouseY, 50, 50);
  }
}
```

(do http://processing.org)

Para mais informações sobre a estrutura e sintaxe do código:
http://processing.org/reference/index.html
http://processing.org/learning/index.html
http://processing.org/faq/index.html

RESUMINDO:

IDE;
Estrutura e Sintaxe;
Modos de programação

- Modo Básico
  Serve basicamente para desenhar objectos simples em ecrã;
- Modo Contínuo
  Fornece estrutura ao programa e permite desenhar de forma continua, actualizar e introduzir interactividade mais complexa;

Help

- Local
- On-line

Sistemas de coordenadas
Modo Básico

- Primeiras formas estáticas rect() e ellipse();
- Primerias formas dinâmicas mouseX, mouseY.

Comentários
Permite escrever notas e texto importante para o(s) programador(es)

- Simples (notas);
- Multi-linha (documentação/instruções);

Primeiro Programa
Modo Contínuo

- setup()
  size()
  background()
  fill()
  stroke()
  rectMode()

- draw()
  rect() interactivo

EXERCÍCIO (DEMONSTRAÇÃO):

1. Desenhar uma aplicação de tamanho 500x500 pixeis;
2. Desenhar um forma dinâmica em programação contínua que mude de tamanho, cor e forma.
3. Explorar o efeito de arrasto.

DEFINIR PARA CRIAR – DESENHAR OS PARÂMETROS DO DESENHO

A partir da apresentação de Petr van Blokland (Lisboa, 01 de Setembro de 2006). Blokland apresentou um conjunto de ideias retiradas do livro de Semiologia e criação Gráfica de Bertin aplicando-as à construção de programas – mais concretamente à sistematização do Design. Assim como vimos na Arte Conceptual a criação de um programa de computador não é uma ideia nova. O problema fulcral centra-se na parametrização do problema e na criação do conjunto de instruções a serem seguidas por quem irá executar o programa.

Jacques Bertin
Semiologie Graphique (1967)

Disponível na Biblioteca da FBAUP:
Bertin, Jacques
Semiologie graphique : les diagrammes-les réseaux-les cartes / Jacques Bertin. - 2ª ed. - Paris : Gauthier-Villars, 1973. - [4], 431p. : il. ; 28 cm
Cota 910 9

Também disponível:
Bertin, Jacques
La graphique et le traitement graphique de l'information / Jacques Bertin. - Paris : Flammarion, 1977. - 277p. : il. ; 22 cm
Gráficos / Informaço - Tratamento gráfico / Semiótica / Comunicaço
Cota 766 116

7 PARÂMETROS

Na criação gráfica podemos identificar um conjunto de 7 parametros que condicionam, ou compõe todo o tipo de criações gráficas:

1. Posição
1. Tamanho
2. Valor
3. Textura
4. Cor
5. Orientação
6. Forma

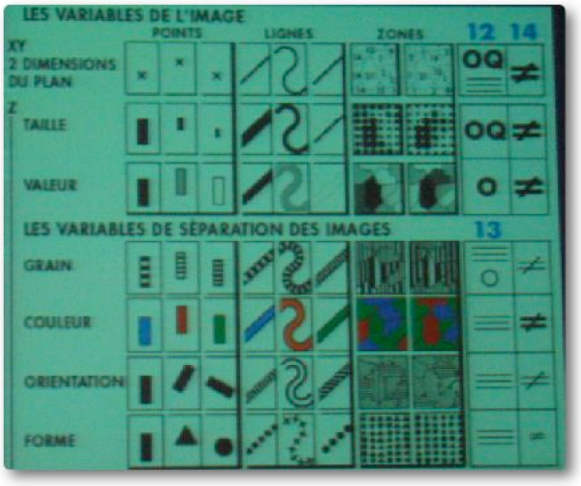


FIGURA 14 - JACQUES BERTIN - PARAMETROS

OS MEIOS DIGITAIS ACRESCENTAM MAIS 7 (PELO MENOS;)

1. Movimento
2. Crescimento
3. Adição (aumento)
4. *Moiré*
5. Espectro
6. Rotação
7. Mutação

A SOLUÇÃO É CRIAR GRUPOS DE PARÂMETROS

Formalmente o resultado pode ser bastante diferente, mas… A ideia que Blokland quiz apresentar é a de que os três resultados embaixo são iguais. Isto é, apresentam-se como uma solução válida para a resolução de uma capa de revista. A resolução é a mesma Títulos, subtítulos ouj informação extra e ilustrações. Isto apesar de estarem a ser representados por intermédio de diferentes grupos de parâmetros – Só texto, Texto e imagem, texto como ilustração, etc...

FIGURA 15 - PETR BLOKLAND - GRUPOS DE PARAMETROS

Há sempre um nível de abstracção que torna as coisas semelhantes. O “truque” está em descobrir, ou definir essas semelhanças e diferenças, essas metáforas para a representação. Tirando partido, claro, dos parâmetros do design gráfico tradicional, assim como os parâmetros introduzidos pelos meiios digitais interactivos.

Lembram-se da representação das 3 cadeiras de Kosuth?

JOSEPH KOSUTH, ONE AND THREE CHAIRS (1965) TONY GODFREY, CONCEPTUAL ART, LONDON: 1998

FIGURA 16 - KOSUTH, CADEIRA

EXEMPLOS

BEN FRY

http://benfry.com/

Ben Fry received his doctoral degree from the Aesthetics + Computation Group at the MIT Media Laboratory, where his research focused on combining fields such as Computer Science, Statistics, Graphic Design, and Data Visualization as a means for understanding complex data. After completing his thesis, he spent time developing tools for the visualization of genetic data as a postdoc with Eric Lander at the Eli & Edyth Broad Insitute of MIT & Harvard. For the 2006-2007 school year, Ben is teaching in Pittsburgh as the Nierenberg Chair of Design for the the Carnegie Mellon School of Design.

With Casey Reas of UCLA, he currently develops Processing, an open source programming environment for teaching computational design and sketching interactive media software that won a Golden Nica from the Prix Ars Electronica in 2005. In 2006, Fry received a New Media Fellowship from the Rockefeller Foundation to support the project.

His personal work has shown at the Whitney Biennial in 2002 and the Cooper Hewitt Design Triennial in 2003. Other pieces have appeared in the Museum of Modern Art in New York, at Ars Electronica in Linz, Austria and in the films "Minority Report" and "The Hulk." His information graphics have also illustrated articles for the journal Nature, New York Magazine, and Seed.

ANEMONE, 2005

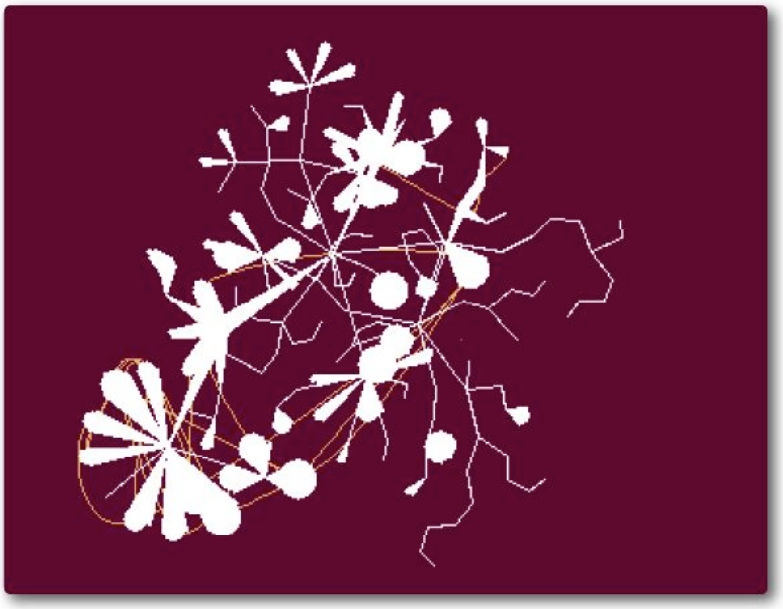


FIGURA 17 - ANEMONE, 2005 DE BEN FRY

What does a web site's structure look like? A site is made up of thousands of pages, all linked together in a tree-shaped structure. A 'map' of the structure can be drawn using illustration software, but the diagram quickly becomes tangled in anomalies because the site is not as hierarchical as would be expected. To further complicate matters, the contents of the site are continually changing. Web pages are added and removed daily, so by the time the map could be finished, it would already be inaccurate. How can these changes be represented visually?

How can a connection be made between the site's usage statistics and that structure? How can the paths that visitors take through the site be expressed? A number next to each page could keep track of the total number of visitors, but because the traffic patterns are continually changing, it becomes clear that the totals aren't as useful as hoped. How can the day to day and month to month changes in these numbers be expressed? How can the movement between pages be represented, making it apparent that some areas are more closely related than others?

Anemone is a project that uses the process of Organic Information Design to make this set of problems more approachable. Data from the Aesthetics and Computation Group's web site was used as input in the examples shown here. Rules for growth can govern the creation of new branches of structure within the site. Atrophy rules decay unused areas, eventually removing

them. Individual web pages can call attention to themselves as they are visited more rapidly than others. A set of rules governing movement can group related areas.

VALENCE, 1999

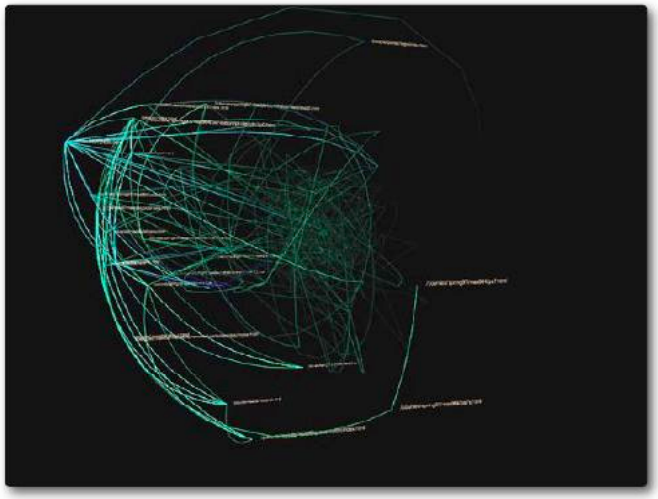


FIGURA 18 - VALENCE, 1999 DE BEN FRY

http://acg.media.mit.edu/people/fry/valence/

Valence is a set of software sketches about building representations that explore the structures and relationships inside very large sets of information.

Genome Valence is the most recent incarnation of this software. It visualizes biological data and was created for the Whitney Biennial in 2002.

I'm interested in building systems that create visual constructions from large bodies of information. The methods used in designing static chunks of data: charting, graphing, sorting and the rest (see the books by Tufte for the complete run-down) are well understood, but much interesting work remains in finding models and representations for examining dynamic sources of data, or very very large data sets. For this work, I'm employing behavioral methods and distributed systems which treat individual pieces of information as elements in an environment that produce a representation based on their interactions. Valence is a software experiment that addresses these issues.

An example

The image on this page is taken from a visualization of the contents of the book "The Innocents Abroad" by Mark Twain. The program reads the book in a linear fashion, dynamically adding each word into three-dimensional space. The more frequently particular words are found, they make their way towards the outside (so that they can be more easily seen), subsequently pushing less commonly used words to the center. Each time two words are found adjacent in the text, they experience a force of attraction that moves them closer together in the visual model.

The result is a visualization that changes over time as it responds to the data being fed to it. Instead of less useful numeric information (i.e. how many times the word 'the' appeared), the piece provides a qualitative feel for the perturbations in the data, in this case being the different types of words and language being used throughout the book.

Why is it effective?
The premise is that the best way to understand a large body of information, whether it's a 200,000 word book, usage data from a web site, or financial transaction information between two multinational corporations, is to provide a feel for general trends and anomalies in the data, by providing a qualitative slice into how the information is structured. The most important imformation comes from providing context and setting up the interrelationships between elements of the data. If needed, one can later dig deeper to find out specifics, or further tweak the system to look at other types of parameters.

CASEY REAS

http://reas.com/

Reas is an artist and educator living and working in Los Angeles. His work employs ideas explored in conceptual and minimal artworks as focused through the contemporary lens of software. He exhibits, performs, and lectures in the US, Asia, and Europe. As an associate professor in the Design | Media Arts department at UCLA, Reas interacts with undergraduate and graduate students. His classes provide a foundation for thinking about software as a medium for visual exploration.

TI, 2004

FIGURA 19 - TI, 2004 DE CASEY REAS

Aggregate layers of abstraction remove every trace of systemic complexity, revealing a living surface. Structured form emerges from the results of thousands of local interactions between autonomous elements.
Photo from TI installation at BANK gallery. Photo by Robert Downs TI,2004

PROCESS 4, 2005

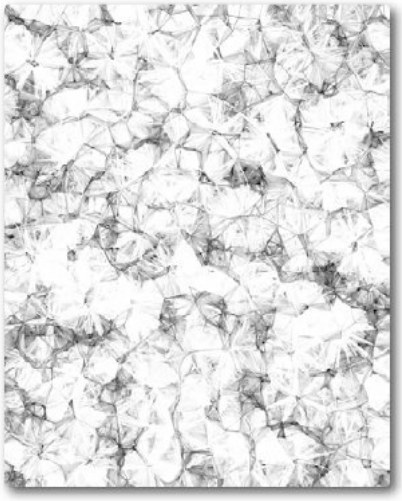


FIGURA 20 - PROCESS 5, 2005 DE CASEY REAS

Process 4 (Software 1) is a software implementation of the Process 4 instructions: "A rectangular surface filled with varying sizes of Element 1. Draw a line from the centers of two Elements when they are touching. Set the value of the shortest line to black and the longest to white, with varying grays between."

Process 4 2005 Software Variable size

ARTICULATE, 2004

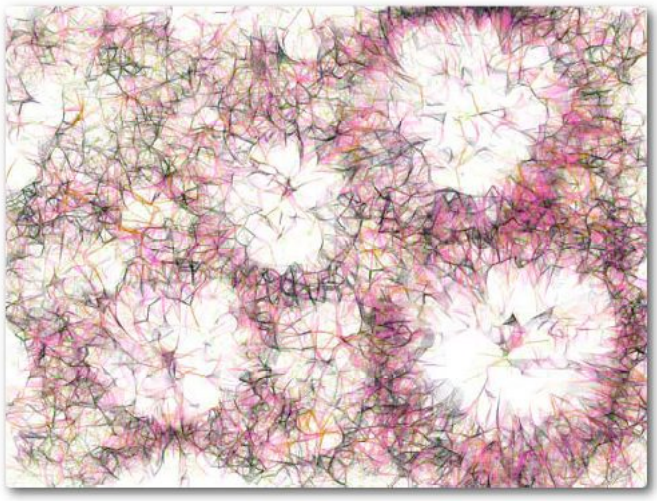


FIGURA 21 - ARTICULATE, 2004 DE CASEY REAS

Structure emerges through the interactions of autonomous software elements.
Still image from a live interaction with the software

Articulate 2004 CD-ROM 6, signed

MARIUZ WATZ

http://www.unlekker.net

ELECTROPLASTIQUE #1 (2005)

http://www.unlekker.net/proj/electroplastique/

FIGURA 22 - ELECTROPLASTIQUE #1, 2005 DE MARIUZ WATZ

4-screen temporal composition for the Territoires Electroniques festival 2005 at Le Fondation Vasarely in Aix-en-Provence. The starting point was the work of Victor Vasarely, the famous

father of Op-Art. I have always been inspired by his work with the transformation of shapes and grids, as well as his wonderful use of light and color to create depth. His Plastic Alphabet and the programmes he created for his images represent a generative approach to form which is strict and yet organic.

In ElectroPlastique #1 a regular grid is deformed and then used as the basis of a series of organic abstract systems that evolve over time (5 minutes). Due to the panoramic format of the 4-screen projection, the impression is much like a landscape. In the end the grid is exploded and disappears, hinting at a non-cartesian vector space.

AV.06: ILLUMINATIONS (2006)

http://www.unlekker.net/proj/illuminations/index.html

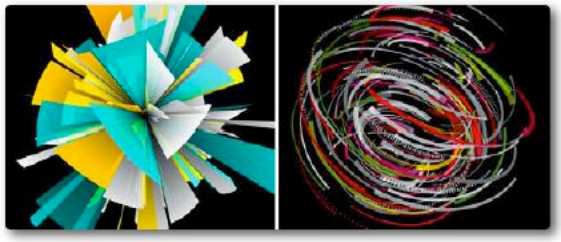


FIGURA 23 – ILLUMINATIONS, 2006 DE MARIUZ WATTZ

Commissioned by the AV.06 festival for the Sage Gateshead venue in Newcastle. Using a fixed projector and two DL2 projectors mounted on a moving base, projections were made to fit the organic space of the concert hall concourse (designed by Norman Foster).

3D form systems were chosen in response to the striking architecture, to maximize the sensation of forms "pushing through" the walls.

C_DRAWER (SYSTEM C, DRAWING MACHINE 1-12)

http://processing.org/exhibition/works/cdrawer/index.html
http://systemc.unlekker.net/

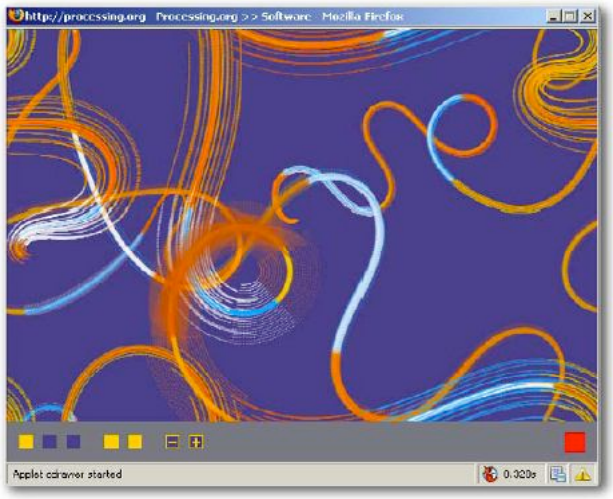


FIGURA 24 - C_DRAWER, 2004 DE MARIUZ WATZ

Like drawing with a bunch of crayons in one hand. Simple and messy, but fun. System C is a time-based drawing machine, a software system for the creation of rule-based images. Several autonomous agents move over a surface, making marks as they move, as a realtime two-screen projection.

The drawing process is modelled on a simple kinetic system. Each agent has a speed and direction that both smoothly change for every step it takes. When the speed or rate of rotation exceeds the minimum or maximum constraints, they start reversing so that their values tend toward the other extreme, causing a subtle oscillation between values over time. Different values for the constraints result in visually different but recognizably related images.

Each image takes a preset amount of time to complete. Once an image is done, the surface is cleared, the drawing system is set to a randomized initial state and the drawing process starts over. The images that are created are saved and uploaded to the web so that online users can observe the most recent activity. All completed images are saved with a time stamp indicating the time of their creation. An archive documents the whole period of the exhibition.Some physicists define glass not as a solid but rather as a slow-moving liquid, so viscous that we cannot observe its movement. System C draws its images slowly enough to be perceived as nearly static, yet is in reality in a process of slow development.This project uses the Open Source FTP library edtFTPj from Enterprise Distributed Technologies.

BORIS MÜLLER - POETRY ON THE ROAD, 2006

http://www.esono.com/
http://www.poetry-on-the-road.com/

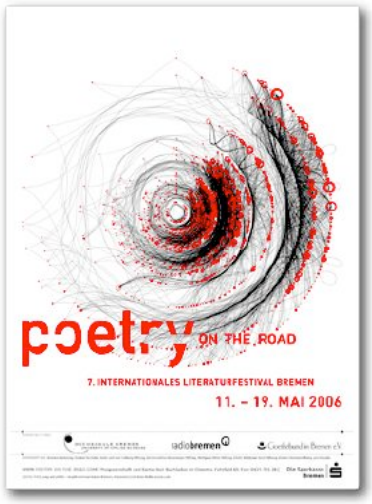


FIGURA 25 - POETRY ON THE ROAD, 2006 DE BORIS MÜLLER

Poetry on the Road is an international literature festival which is held every year in Bremen, Germany. Since 2002 I am commisioned to design a visual theme for the festival. While the theme itself is changing, the underlying idea for the visuals is always the same: All graphics are generated by a computer program that turns texts into images. So every image is the direct representation of a specific text. The design and the development process are a collaboration with the design agency jung und pfeffer.

AARON KOBLIN

http://www.aaronkoblin.com/

Aaron Koblin recieved his MFA from the Department of Design|Media Arts at UCLA and his BA in Electronic Art at the University of California, Santa Cruz. Aaron has studied in The Netherlands and Japan and his work has been shown internationally. Utilizing a background in the computer game industry he led a course in game design for the web at UCLA and has been working with data driven projects as an interactive designer and researcher for the Center for Embedded Networked Sensing (CENS).

FLIGHT PATTERNS

FIGURA 26 - FLIGHT PATTERNS, 2004 DE AARON KOBLIN

Air traffic as seen by the FAA.

The following flight pattern visualizations are the result of experiments leading to the project Celestial Mechanics by Scott Hessels and Gabriel Dunne. FAA data was parsed and plotted using the Processing programming environment. The frames were composited with Adobe After Effects and/or Maya.

LIA

http://lia.sil.at/

Lia has been working on digital art since 1995, after graduating from the High-school for Music Students in Graz, Austria. Living in Vienna since then, she splits her time between visual design, web art, video and realtime visual performances, apparently different activities that she manages to bind together trough her unique approach to creativity and production. Over the last years she has also been an invited teacher at the Fachhochschule Joanneum in Graz, the École Cantonalle d'Art de Lausanne, and the Fine Arts University in Oslo.

REMOVE.ORG

http://www.re-move.org/

FIGURA 27 - REMOVE, ? DE LIA ?

O.I.G.C. GENERATIVE INTERACTIVE AUDIO-VISUAL APPLICATION

for the FURTHER PROCESSING exhibition @ Medienturm in Graz, Austria.
http://flickr.com/photos/lia_lia/sets/72157594273344128/

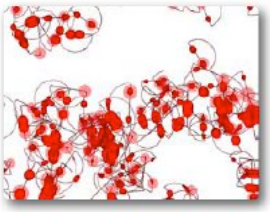


FIGURA 28 - O.I.C.G., 2006 DE LIA

WITHOUTTITLE

http://processing.org/exhibition/works/withouttitle/index.html

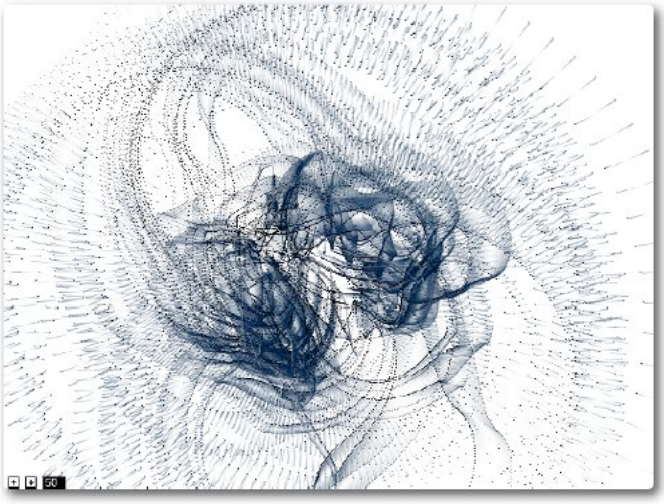


FIGURA 29 - WITHOUT TITTLE, 2006 DE LIA

A software machine exploring the boundaries of control.

GRASS, 2006 – THE BARBARIAN GROUP

http://portfolio.barbariangroup.com/nextfest/index.html

The Barbarian Group was asked to help Goodby, Silverstein & Partners with a multimedia installation for Wired Magazine's NextFest. The purpose of the installation was to promote the new Saturn hybrid cars. Goodby teamed up with Obscura Digital to make the installation space. Obscura is no stranger to convention installations and their expertise was invaluable throughout the entire process.

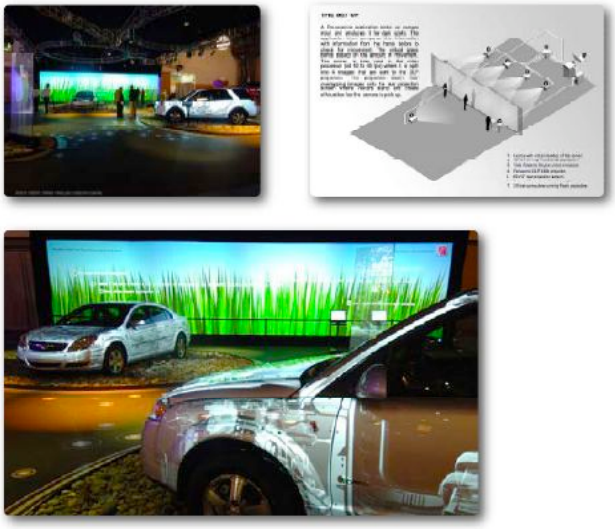


FIGURA 30 - GRASS, 2006 DE THE BARBARIAN GROUP

Our part in this involved making an interactive projection wall shown seamlessly by four DLP projectors on a 45'x12' rear projection screen. The concept was simple enough: 45 feet of grass swaying in a virtual breeze. Visitors to the installation can make the grass sway just by walking in front of it. Additionally, visitors can input text from either of the two kiosks positioned in front of the screen. This text appears in the projection attached to its own grass blade and sways along with the rest of the scene.

MANIFEST, 2005 - MICHAEL CHANG

http://users.design.ucla.edu/~mflux/manifest/

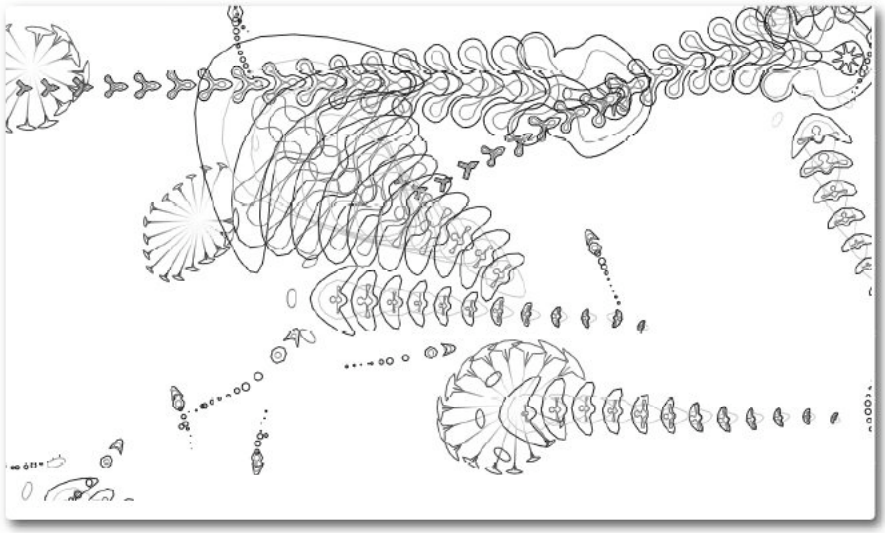


FIGURA 31 - MANIFEST, 2005 DE MICHAEL CHANG

Drawing program where different gestures create varying organisms animated through procedural animation and physics simulations.

This is a project written with Processing in the Programming Media II class taught by Casey Reas and Sean Dockray at the UCLA department of Design|Media Arts in the Spring of 2005. It is an interactive toy that allows one to draw and create organisms on the fly. Varying stroke lengths and loops create different organisms. The program is driven by procedural animation and physics simulations. All art assets were made in Adobe Illustrator and imported into Processing as SVG files (Scalable Vector Graphics format). Creature behaviors are driven by a rudimentary AI with some boid-like behavior. Inspiration came from Sodaplay and Presstube.

METROPOP DENIM, 2005-06 - CLAYTON CUBITT AND TOM CARDEN

Fashion Photography and Generative Artwork
http://processing.org/exhibition/works/metropop/

FIGURA 32 - METROPOP DENIM, 2005-06 DE CUBITT E CARDEN

THINKING MACHINES 4, 2001-04 - MARTIN WATTENBERG

http://www.turbulence.org/spotlight/thinking/

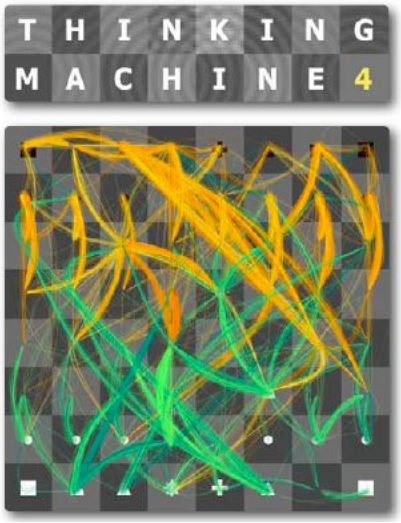


FIGURA 33 - THINKING MACHINES, 2001 DE MARTIN WATTENBERG

Thinking Machine 4 explores the invisible, elusive nature of thought. Play chess against a transparent intelligence, its evolving thought process visible on the board before you.

The artwork is an artificial intelligence program, ready to play chess with the viewer. If the viewer confronts the program, the computer's thought process is sketched on screen as it plays. A map is created from the traces of literally thousands of possible futures as the program tries to decide its best move. Those traces become a key to the invisible lines of force in the game as well as a window into the spirit of a thinking machine.

ALPHABOT, 2000 - NIKITA PASHENKOV

http://processing.org/exhibition/works/004/index.html

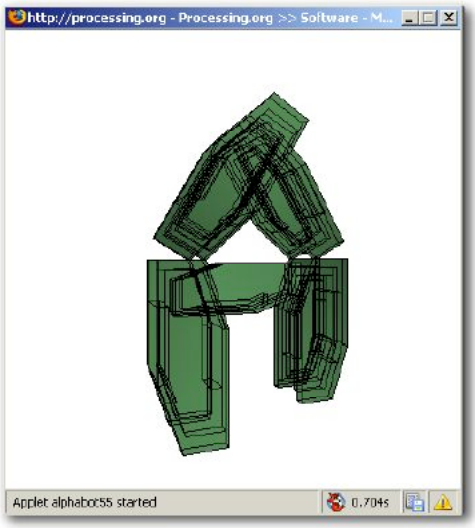


FIGURA 34 - ALPHABOT, 2000 DE NIKITA PASHENKOV

Alphabot is a virtual robot that is able to take the shape of any letter in the English alphabet as you type them on a keyboard.

The alphabot project was designed and executed as an (interactive) entry for a typography competition which took place in November 2000.

THE DUMPSTER, 2006 - GOLAN LEVIN ET AL.

http://artport.whitney.org/commissions/thedumpster/index.html

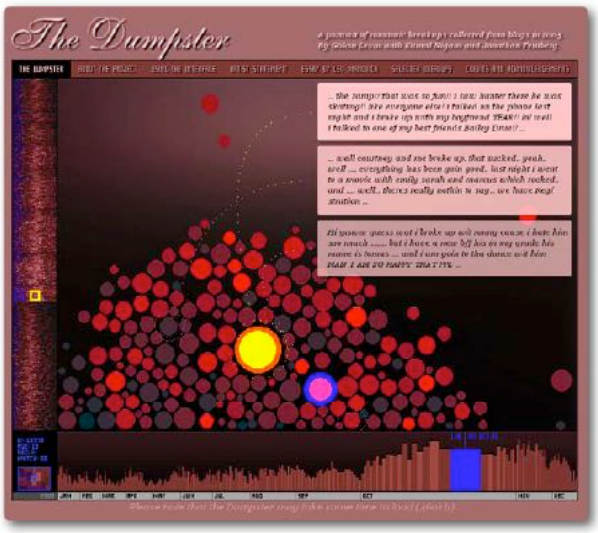


FIGURA 35 - THE DUMPSTER, 2006 DE GOLAN LEVIN, ET. AL.

An interactive online visualization that attempts to depict a slice through the romantic lives of American teenagers.
The Dumpster is an interactive online visualization that attempts to depict a slice through the romantic lives of American teenagers. Using real postings extracted from millions of online blogs, visitors to the project can surf through tens of thousands of specific romantic relationships in which one person has "dumped" another. The project's graphical tools reveal the astonishing similarities, unique differences, and underlying patterns of these failed relationships, providing both peculiarly analytic and sympathetically intimate perspectives onto the diversity of global romantic pain.

The Dumpster was created by Golan Levin, Kamal Nigam and Jonathan Feinberg and made possible by support from the Whitney Artport, the Tate Online, and Intelliseek. Version 1.0 of the Dumpster was built in Processing and launched on Valentine's day, 2006.

The Dumpster is the first in a series of three Net-based artworks co-commissioned by the Whitney Artport and the Tate Online. Critical texts and video interviews with the artists will accompany the works at the Tate Online web site. A high-resolution press image of the Dumpster can be found here. An essay about the Dumpster by media critic/historian Lev Manovich appears here.

NIKE ONE, 2005 - MOTION THEORY

http://dev.motiontheory.com/nikegolf/

FIGURA 36 - NIKE ONE, 2005 DE MOTION THEORY

Motion Theory used coding and design to visualize the thoughts and mechanics of the creative process of two engineers.

Nike 'One'
Motion Theory used coding and design to visualize the thoughts and mechanics of the creative process of two engineers behind Nikes' evolving golf line.

CONCEITOS GERAIS DE PROGRAMAÇÃO

O seguinte capítulo demonstra os conceitos chave da programação numa abordagem teórica. Pretende-se fornecer uma visão geral dos principais conceitos de programação. Ao contrário do capítulo seguinte, este não segue uma lógica de aprendizagem e serve apenas de *showcase* de elementos que irão ser utilizados gradualmente na aplicação prática dos conceitos e estruturas. A função deste capítulo é quase a de um glossário ou índice remissivo.

Os exemplos relativos ao Processing podem ser acedidos através das moradas fornecidas ou através dos exemplos fornecidos com a pasta de Sketches do próprio programa.

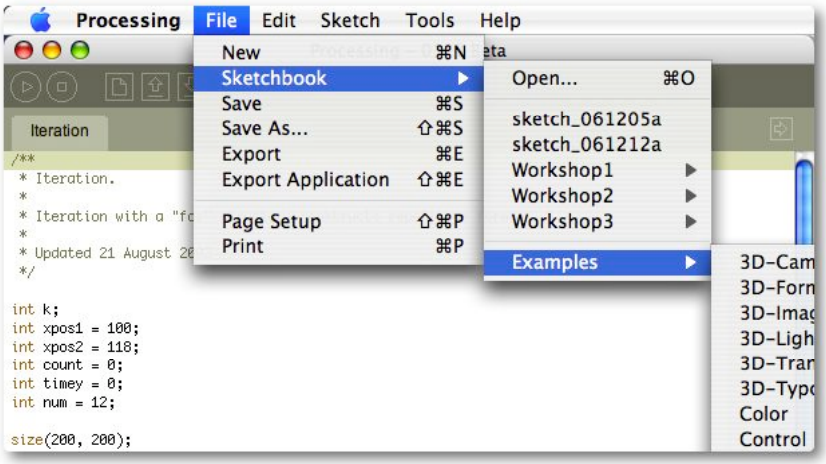


FIGURA 37 - EXEMPLOS INCLUIDOS COM O PROCESSING.

VARIÁVEIS (MEMÓRIA DO PROGRAMA)

Um conceito muito importante em algoritmia é a memória
– A memória permite-nos armazenar resultados intermédios do algoritmo
– No algoritmo anterior, armazenámos o número da página corrente

Os elementos de memória utilizados num algoritmo designam-se por variáveis.
Todo o tipo de elementos que vão ser reutilizados pelo programa mais tarde tem que ser guardados nas "gavetas" ou "contentores" que são variáveis. Para os reutilizarmos temos que "abrir" as gavetas. Para isso temos que dizer ao computador para abrir/chamar a gaveta/variável que queremos. Como é que o computador sabe qual queremos? Todas a variáveis tem que ter um nome – é um identificador que diz ao computador em que parte da memoria (geral) se encontra o valor que queremos. Tem que ser um nome contínuo e

não pode começar por números. Uma politica comum é usar a notação de "camelo" (camelCase) ou *underscores* (under_scores).

DECLARAR VARIÁVEIS

Declarar – Dizer ao programa que vamos usar um "contentor" de um certo tipo de dados – o processing, ao contrario do Flash ou PHP precisa que seja indicado o tipo de dados que a variável vai armazenar – números inteiros, reais, texto, etc.;
Inicialização - Atribuir valores ou dados às variáveis;
Declarar e inicializar ao mesmo tempo também é possível.

http://processing.org/learning/examples/variables.html

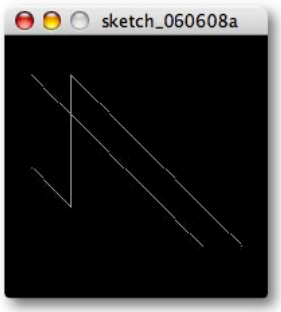


FIGURA 38 – VARIÁVEIS

```
size(200, 200);
background(0);
stroke(153);

int a = 20;
int b = 50;
int c = a*8;
int d = a*9;
int e = b-a;
int f = b*2;
int g = f+e;

line(a, f, b, g);
line(b, e, b, g);
line(b, e, d, c);
line(a, e, d-e, c);
```

VISIBILIDADE DAS VARIÁVEIS (*SCOPE*)

1. Variáveis Locais (declaradas dentro de métodos) – só são acessíveis "dentro" dos métodos e não são partilhadas por mais nenhuma parte do programa.

2. Variáveis Globais (declaradas no início e atribuídas a qualquer altura, acessíveis por todos os métodos a qualquer altura do programa).

(vamos ver o exemplo e aplicação do âmbito das variáveis e o funcionamento dos métodos)

TIPOS DE DADOS

Todos os dados armazenados em memoria e ou manipulados pelo computador tem uma natureza específica. Isto permite que sejam realizadas certo tipo de operações e outras não.

"Some o valor Pedro com o valor 15"
– Hmm… WTF?

Os valores/variáveis utilizadas nas operações (ver operadores) têm de ser compatíveis
– Têm de ser do mesmo tipo.

Quando utilizamos uma variável, essa variável tem, implícita ou explicitamente, um tipo. Mais uma vez, não estamos a falar de tipografia, mas tipo num sentido de género.

TIPOS SIMPLES (PRIMITIVOS)

Tipos simples são tipos cujos valores são atómicos, i.e., não podem ser decompostos
Numérico

1. Inteiro (p.ex.: 2)
2. Real (p.ex.: 2,35 ou em linguagem do processing – 2.35);

Alfanumérico (cadeia de caracteres ou *string*)
"1" (*string*) é diferente de 1 (número)

Lógico (verdadeiro ou falso - *boolean*)

TABELA DE TIPOS DE DADOS

Keyword	Descrição	Tamanho/Gama
		(Números Inteiros)
byte	Byte	8-bit (1 byte) (-128 a 127)
short	Inteiro curto	16-bit (2 bytes) (-32768 a 32767)
int	Inteiro	32-bit (4 bytes) (-2147483648 a 2147483647)

long	Inteiro longo	64-bit (8 bytes) (-9223372036854775808 a ...807)
		(Números Reais)
float	Vírgula flutuante	32-bit IEEE 754 de precis‹o simples (4 bytes) (1.40129846432481707e-45 a 3.40282346638528860e+38) (positivo ou negativo)
double	Vírgula flutuante	64-bit IEEE 754 de dupla precis‹o (8 bytes) (4.94065645841246544e-324 a 1.79769313486231570e+308) (positivo ou negativo)
		(Outros)
char	Um caracter	16-bit Unicode (2 bytes)
boolean	Valor lógico	1-bit (true ou false)

http://processing.org/learning/examples/integersfloats.html

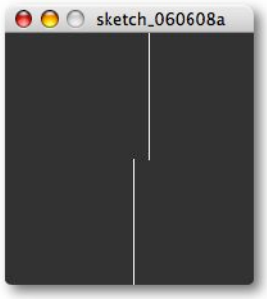


FIGURA 39 – TIPO DE DADOS: INT() E FLOAT()

```
int a = 0;      // Create a variable "a" of the datatype "int"
float b = 0.0;  // Create a variable "b" of the datatype "float"

void setup()
{
  size(200, 200);
  stroke(255);
  framerate(30);
}

void draw()
{
  background(51);

  a = a + 1;
  b = b + 0.2;
  line(a, 0, a, height/2);
  line(b, height/2, b, height);

  if(a > width) {
    a = 0;
  }
  if(b > width) {
    b = 0;
  }
}
```

TIPOS COMPLEXOS (COMPOSTOS)

Tipos complexos são tipos compostos por vários elementos simples:

Vector (*Array* em inglês – não confundir com a palavra inglesa *vector*): uma lista de elementos do mesmo tipo que podem ser acedidos via um índice.
Matriz: vector multi-dimensional;
Estrutura: agregação de vários tipos de dados;

http://processing.org/learning/examples/array.html

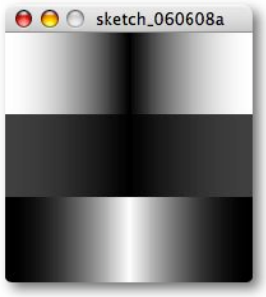


FIGURA 40 – VECTORES

```
size(200, 200);

float[] coswave = new float[width];

for(int i=0; i<width; i++) {
  float ratio = (float)i/(float)width;
  coswave[i] = abs( cos(ratio*PI) );
}

for(int i=0; i<width; i++) {
  stroke(coswave[i]*255);
  line(i, 0, i, width/3);
}

for(int i=0; i<width; i++) {
  stroke(coswave[i]*255/4);
  line(i, width/3, i, width/3*2);
}

for(int i=0; i<width; i++) {
  stroke(255-coswave[i]*255);
  line(i, width/3*2, i, height);
}
```

INSTRUÇÕES

Para atribuirmos valores a variáveis e controlarmos o fluxo do algoritmo precisamos de instruções:

– Declarar uma variável:

Inteiro idade;

Real preço;

– Atribuir um valor a uma variável:

```
xpto <- 25;
```

– Ler um valor introduzido pelo utilizador para uma variável

```
ler idade;
```

– Escrever texto/valor de uma variável no ecrã:

```
escrever "Valor de XPTO:", xpto;
```

– Alterar o fluxo do programa:

```
se (idade < 18) então
       podeConduzir <- falso;
senão
       podeConduzir <- verdadeiro;
fim se
```

CONDIÇÕES

As condições são uma parte importante nas nossas vidas. São acções ou decisões tomadas em função de determinadas condições. Vamos à praia se não estiver a chover, comemos se tivermos fome, vamos ao Belas se não gostarmos da comida da cantina, etc. Na programação também é preciso tomar decisões e executar acções condicionalmente, a este processo chamamos de *Controlo de Fluxo*. Este controlo do fluxo do programa é devido essencialmente a duas estruturas: `if e switch`

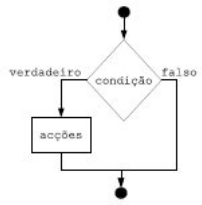


FIGURA 41 - DIAGRAMA DE UMA CONDIÇÃO SIMPLES

A ESTRUTURA IF...ELSE.

A estrutura if-else é a forma mais elementar de uma estrutura condicional. Se sim – fazemos - se não... sopas!

```
if (condição) {
acções;
}
```

Ou ainda

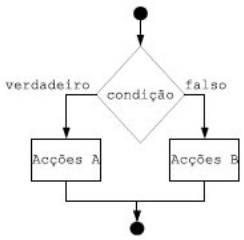


FIGURA 42 - DIAGRAMA DE UMA ESTRUTURA IF-ELSE

```
if (condição) {
acções 1;
} else if (condição) {
acções 2;
}
```

A estrutura if-else embora simples, pode dar origem a ramificações bastante complexas dependendo do número de condições a verificar ou ainda do numero de ramificações de cada condição (sub-condições).

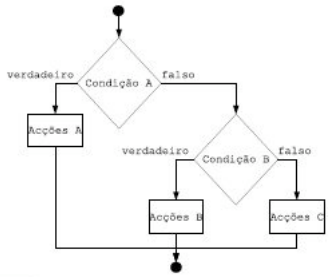
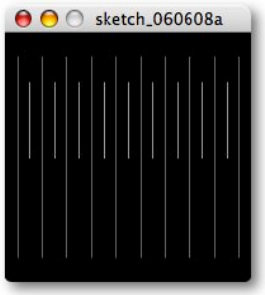


FIGURA 43 - DIAGRAMA DE UMA ESTRUTURA IF-ELSE COM RAMIFICAÇÕES

No entanto para casos de mais do que 2 ou três condições a verificar é aconselhado a usar a estrutura `switch`.




```
size(200, 200);
```

```
background(0);

for(int i=10; i<width; i+=10) {
  // If 'i' divides by 20 with no remainder draw the first line
  // else draw the second line
  if(i%20 == 0) {
    stroke(153);
    line(i, 40, i, height/2);
  } else {
    stroke(102);
    line(i, 20, i, 180);
  }
}
```

COMUTAR (ESTRUTURA SWITCH)

Como foi dito acima, a estrutura Switch é mais prática quando é preciso utilizar uma estrutura condicional onde é preciso fazer a verificação de 3 ou mais condições. Embora a estrutura if-else funcione, nestes casos revela-se mais eficiente tanto a nível da máquina como para escrever o código usar a estrutura switch.

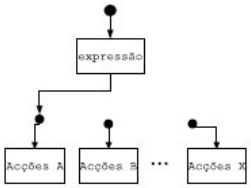


FIGURA 44 - DIAGRAMA DE FLUXO DA ESTRUTURA SWITCH

```
variável/condição x;

switch (condição x) {
  case 1:
    acções 1;
    break;

  case 2:
    acções 2;
    break;

  case 3:
    ...
}
```

http://processing.org/reference/switch_.html

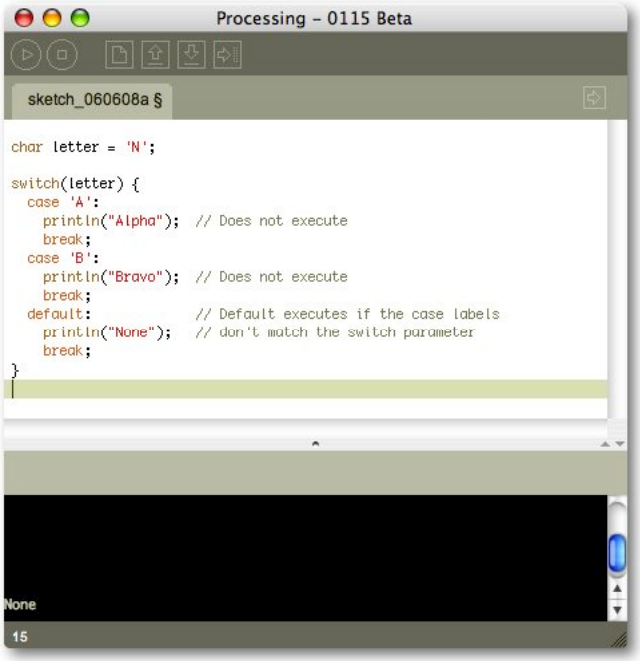


FIGURA 45 - SWITCH()

```
int num = 1;

switch(num) {
  case 0:
    println("Zero");  // Does not execute
    break;
  case 1:
    println("One");  // Prints "One"
    break;
}

char letter = 'N';

switch(letter) {
  case 'A':
    println("Alpha");  // Does not execute
    break;
  case 'B':
    println("Bravo");  // Does not execute
    break;
  default:              // Default executes if the case labels
    println("None");   // don't match the switch parameter
    break;
}

// Removing a "break" enables testing
// for more than one value at once

char letter = 'b';

switch(letter) {
  case 'a':
```

```
  case 'A':
    println("Alpha");  // Does not execute
    break;
  case 'b':
  case 'B':
    println("Bravo");  // Prints "Bravo"
    break;
}
```

OPERADORES

Para manipularmos os valores das variáveis precisamos de operadores
– Operadores aritméticos: `+, -, *, /`
– Operadores condicionais: `>, <, >=, <=, <>`
– Operadores lógicos: ou, e, negação

OPERADORES CONDICIONAIS

Só podem ser usados em estruturas condicionais visto o resultado produzido por esta verificação ser sempre uma variável do tipo boolean – verdadeiro ou falso – o que permite ou não verificar a condição. Os operadores condicionais são diferentes dos operadores de atribuição. Dizer que a = 10; é diferente que dizer a == 10;. O primeiro atribui o valor 10 à variável e o segundo verifica se a tem um valor numérico inteiro de 10 (devolvendo verdadeiro ou falso).

==	Igual a
>	Maior que
<	Menor que
>=	Maior ou Igual a
<=	Menor ou igual a
!=	Diferente ou negação (not)

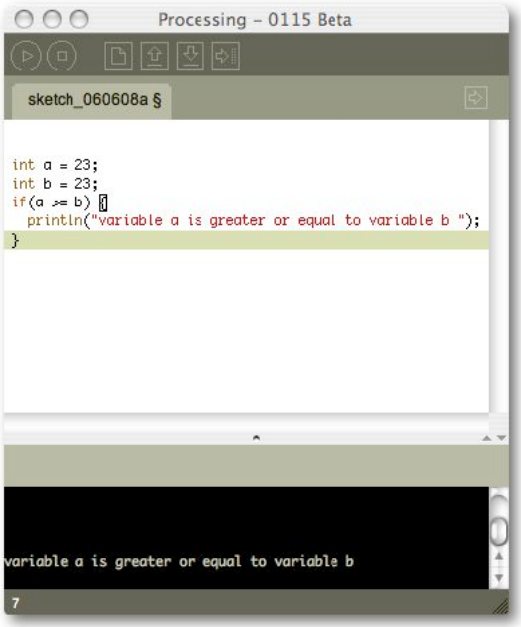


FIGURA 46 - OPERADORES CONDICIONAIS

```
int a = 23;
int b = 23;
if(a >= b) {
  println("variable a is greater or equal to variable b ");
}
```

OPERADORES LÓGICOS

Os operadores Lógicos operam sobre dados lógicos. Isto é, são como elementos de conjunção sintácticos que permitem executar condições complexas. Muitas vezes precisamos de verificar condições do género se não estiver a chover e fizer calor vamos à praia. Para verificar esta condição precisamos de um operador lógico para interligar os elementos.

&&	e
||	ou
!	não

|| – ("ou")

A	B	A || B
false	false	false
false	true	true
true	false	true
true	true	true

&& – ("e")

A	B	A && B
false	false	false
false	true	false
true	false	false
true	true	true

! – ("não")

A	!A
false	true
true	false

FIGURA 47 - TABELA DE EXEMPLOS DOS OPERADORES LÓGICOS

http://processing.org/learning/examples/logicaloperators.html

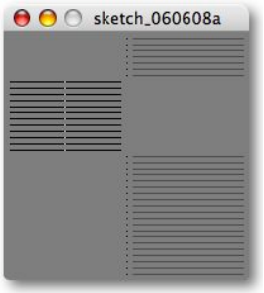


FIGURA 48 – OPERADORES LÓGICOS

```
size(200, 200);
background(126);

boolean op = false;

for(int i=5; i<=195; i+=5) {
  // Logical AND
  stroke(0);
  if((i > 35) && (i < 100)) {
    line(5, i, 95, i);
    op = false;
  }

  // Logical OR
  stroke(76);
  if((i <= 35) || (i >= 100)) {
    line(105, i, 195, i);
    op = true;
  }

  // Testing if a boolean value is "true"
  // The expression "if(op)" is equivalent to "if(op == true)"
  if(op) {
```

```
    stroke(0);
    point(width/2, i);
  }

  // Testing if a boolean value is "false"
  // The expression "if(!op)" is equivalent to "if(op == false)"
  if(!op) {
    stroke(255);
    point(width/4, i);
  }
}
```

CICLOS (CICLOS OU ITERAÇÕES)

Repetir várias vezes um conjunto de instruções para um dado conjunto de condições iniciais ou dinâmicas.

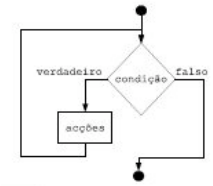


FIGURA 49 - DIAGRAMA DE FLUXO DE UM CICLO SIMPLES

CICLO PARA (*FOR*)

(*for* - repete um número predeterminado de vezes)

```
for (estado inicial; condição; mutação) {
  acções;
}
```

```
para i <- 1, i <= 10, i <- i + 1
  escrever i," ";
fim para
```

"i" é inicializado a "1".

O ciclo é repetido enquanto "i" for menor ou igual a "10". Em cada iteração, "i" é incrementado em 1 unidade.
Resultado: "1 2 3 4 5 6 7 8 9 10"

http://processing.org/learning/examples/iteration.html

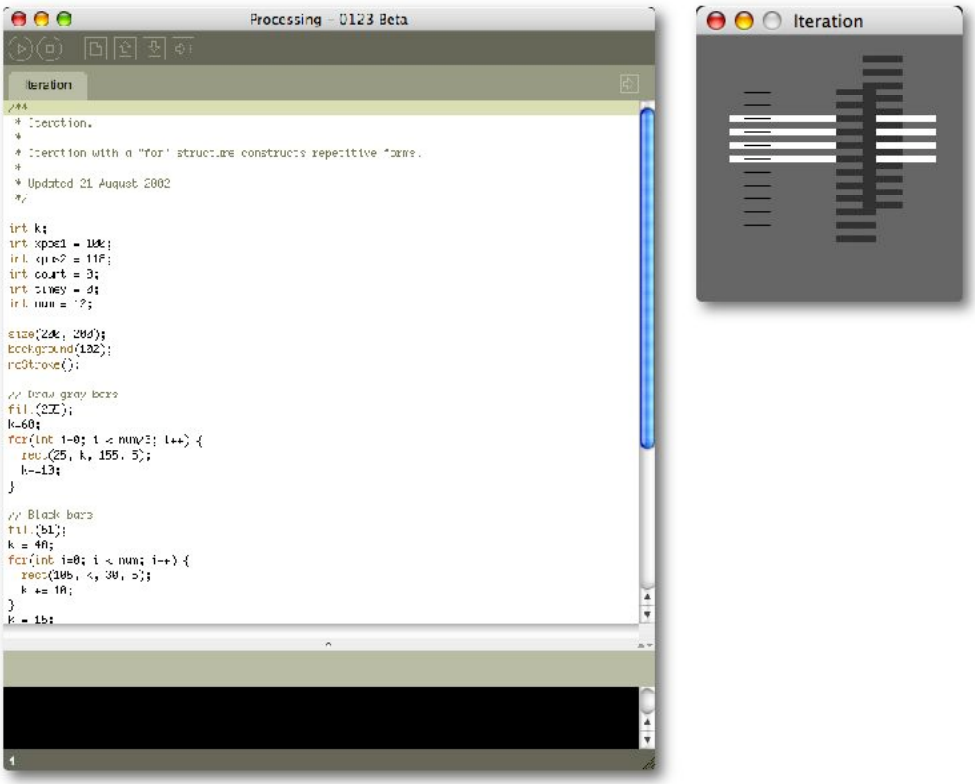


FIGURA 50 - ITERAÇÃO: CICLO FOR()

CICLO ENQUANTO (*WHILE*)

(*while* - repete um conjunto de instruções enquanto uma determinada condição for verdadeira)

```
while (condição) {
Acções;
}
```

```
i <- 2;
enquanto i < 10 fazer
       escrever i, " ";
       i <- i + 2;
fim enquanto
```

As instruções no interior são executadas apenas se i for inferior a 10
A variável i é incrementada 2 valores em cada iteração

Devemos garantir, no interior do ciclo, que a variável utilizada na condição seja eventualmente alterada de forma a que a condição seja falsa. De outro modo o ciclo não termina.
Resultado: “2 4 6 8 “

http://processing.org/reference/while_.html

FIGURA 51 – ITERAÇÃO: CICLO WHILE()

```
int i=0;
while(i<80) {
  line(30, i, 80, i);
  i = i + 5;
}
```

CICLO FAZER (*DO*)

(*do* - repete um conjunto de instruções enquanto uma determinada condição for verdadeira; **executa sempre pelo menos uma iteração, a primeira**)

```
do {
acções;
} while (condições);
```

```
i <- 2;
fazer
       escrever i, " ";
       i <- i + 2;
enquanto i < 10;
```

As instruções no interior são executadas apenas se i for inferior a 10
A variável i é incrementada 2 valores em cada iteração
Devemos garantir, no interior do ciclo, que a variável utilizada na condição seja eventualmente alterada de forma a que a condição seja falsa. De outro modo o ciclo não termina.
Resultado: "2 4 6 8 "

VECTORES E MATRIZES (VARIÁVEIS COMPLEXAS)

Até agora usamos variáveis simples. São variáveis (gavetas) que podem armazenar uma coisa de cada vez (contando que são de um tipo de cada vez).
Os vectores (*arrays*) por sua vez, são um tipo de variável dinâmica, como se de uma lista ou de um conjunto de gavetas se tratasse. Estes permitem-nos armazenar na mesma variável um conjunto de dados do mesmo tipo de uma só vez.

VECTORES (*ARRAYS*)

```
nomeVariavel = new tipoVariavel[tamanhoVariavel];

listaDeNumeros = new int[19];
// um vector para armazenar até 20 números

int lista[] = {1, 2, 4, 6, 8, 5};
//Declaração e atribuição simultânea de um vector com 6 posições
```

A melhor maneira de percorrer um vector inteiro será com um ciclo. Ao passo que aceder a uma só posição temos que usar métodos de acesso individualmente:

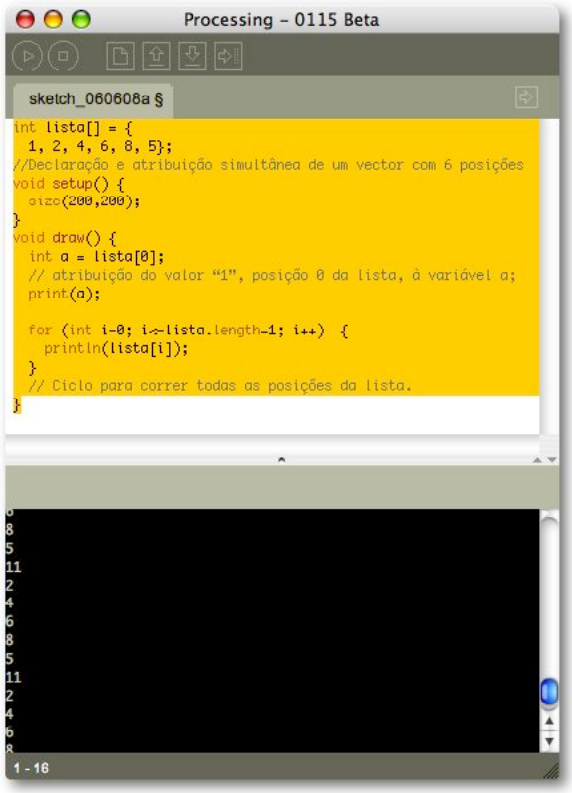


FIGURA 52 - VECTORES (ARRAYS)

```
int lista[] = {
  1, 2, 4, 6, 8, 5};
//Declaração e atribuição simultânea de um vector com 6 posições
void setup() {
  size(200,200);
}
void draw() {
  int a = lista[0];
  // atribuição do valor "1", posição 0 da lista, à variável a;
  print(a);

  for (int i=0; i<=lista.length-1; i++)  {
    println(lista[i]);
  }
  // Ciclo para correr todas as posições da lista.
}
```

MATRIZES

Se um Vector não é mais do que uma extensão de uma variável simples, porque não extender um vector? Uma Matriz é isso mesmo, é uma estrutura bi-dimensional. Se o primeiro – vector - pode ser entendido como um linha de uma tabela de Excel, o segundo – Matriz – pode ser entendido como a tabela inteira.

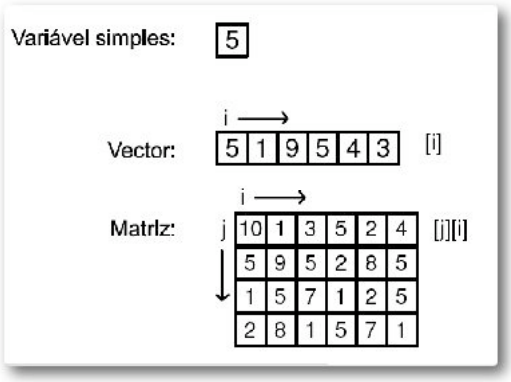


FIGURA 53 – MATRIZES

Declaram-se da mesma maneira que um vector:

```
nomeVar = new tipoVar [numLinhas][numColunas];
matriz = new int[2][4];
```

Percorrem-se da mesma maneira (2 ciclos):

```
matriz = new int[2][4];
for(int i=0; i<= 2; i++) {
  for(int j=0; i<=4; j++) {
       println(matriz[j]);
  }
}
```

http://www.processing.org/learning/examples/array2d.html

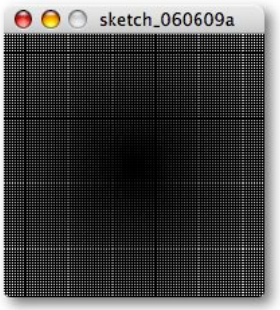


FIGURA 54 – MATRIZES

```
float[][] distances;
float maxDistance;

size(200, 200);
background(0);
maxDistance = dist(width/2, height/2, width, height);
distances = new float[width][height];
for(int i=0; i<height; i++) {
  for(int j=0; j<width; j++) {
    float dist = dist(width/2, height/2, j, i);
    distances[j][i] = dist/maxDistance * 255;
  }
}

for(int i=0; i<height; i+=2) {
  for(int j=0; j<width; j+=2) {
    stroke(distances[j][i]);
    point(j, i);
  }
}
```

FUNÇÕES E MÉTODOS

A palavra função e método tem o mesmo significado. Esta designa um acção a executar pelo programa, uma instrução. Provavelmente a diferença maior é que quando usamos uma função estamos a referir-nos a uma instrução conhecida pela linguagem (API) tal como rect(), ou fill(). Quando queremos que o programa desenhe por exemplo uma casa (casa, telhado duas janelas e uma porta) isto é executado por um método ou um conjunto de instruções para o efeito. Na realidade, são tudo métodos, isto porque para o programa executar uma função seja ela um rect() ou desenhar uma casa, esta tem que estar descrita em algum lado.

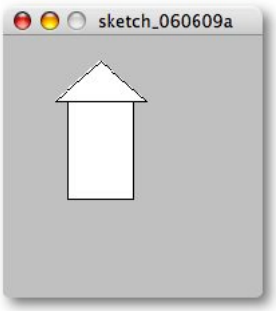


FIGURA 55 - MÉTODOS?

```
size(200,200);

int x = 50; //loc x
int y = 50; //loc Y
int w = 50; //largura
int h = 75; //altura
rect(x, y, w, h);
triangle(x-10, y, x+w+10, y, x+w/2, y-30);
```

A vantagem de usar um método é que o podemos chamar repetidas vezes poupando trabalho.

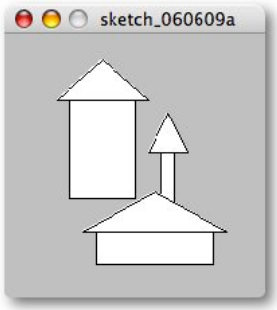


FIGURA 56 – MÉTODOS!

```
int x, y, w, h;

void setup() {
  size(200,200);
}

void draw(){
  x = 50; //loc x
  y = 50; //loc Y
  w = 50; //largura
  h = 75; //altura
  casa(); //invocar o método depois de definir as vars

  x = 120; //loc x
```

```
  y = 90; //loc Y
  w = 10; //largura
  h = 45; //altura
  casa(); //invocar o método depois de definir as vars

  x = 70; //loc x
  y = 150; //loc Y
  w = 90; //largura
  h = 25; //altura
  casa(); //invocar o método depois de definir as vars

}

void casa() {
  rect(x, y, w, h);
  triangle(x-10, y, x+w+10, y, x+w/2, y-30);
}
```

Ainda as vantagens de usar métodos personalizados é poder ajustar ou persolnalizar as nossas funções quando quisermos.

PARÂMETROS

Nos exemplos das funções e métodos usamos um método personalizado, mas para correrrmos o método dependíamos de duas variáveis globais serem redefinidas constantemente. E se exisatisse uma forma mais poupada para chamar o método? Já adivinharam! Com os parâmetros podemos poupar a escrita do código, melhorar o desempenho do programa (não usa tanto a memoria) e tornar o fluxo mais inteligível para o programador.

Os parâmetros não passam de valores passados ao método quando este é chamado:

```
metodoNovo(param1, param2,...);
```

Por exemplo, no médodo da cara() precisamos definir sempre a posição X e Y, logo poderíamos chamar o método de cada vez usando os novos valores assim:

```
casa(50, 50);
//Método cara(posX, posY);
```

Claro que o método teria que estar preparado para receber os parâmetros, por isso teria que ser escrito da seguinte forma:

```
void metodoNovo(param1, param2,...) {
  função(param1, param2);
}
```

Que no nosso exemplo resultaria:

```
Void casa(int posX, int posY) {
  Rect(posX, posY, 50, 75);
}
```

E por aqui fora. Poderíamos desenhar casas composições variáveis, cores variadas, tamanhos variados, etc.

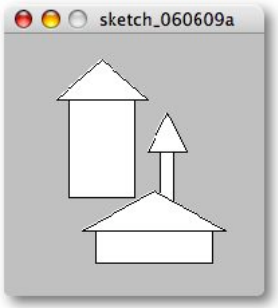


FIGURA 57 - MÉTODOS: PARÂMETROS

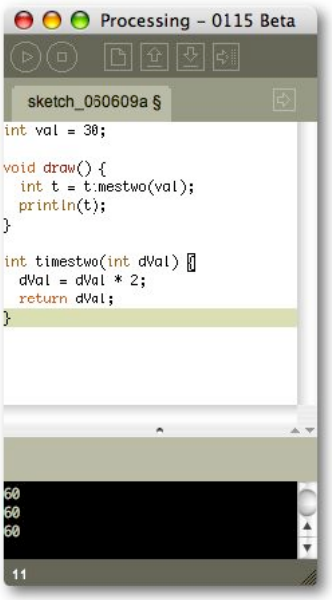
```
void setup() {
  size(200,200);
}

void draw(){
  casa(50, 50, 50, 75); //invocar o método e passar params
  casa(120, 90, 10, 45);
  casa(70, 150, 90, 25);
}

void casa(int x, int y, int w, int h) { // params = p1, p2...
  rect(x, y, w, h);
  triangle(x-10, y, x+w+10, y, x+w/2, y-30);
}
```

RETORNO

Um método também pode ser usado para devolver valores de retorno. Isto é útil quando precisamos de fazer muitas contas no programa principal. Para isso o método tem que ser inicializado de forma ligeiramente diferente:

```
int novoMetodo(int param1, int param2,...) {
  conta = param1/param2;
}
```

Logo para um programa que quiséssemos saber sempre o ponto médio entre duas localizações bastava atribuirmos este mátodo a uma variável para o guardar, funcionando da seguinte forma:




```
int a = novoMetodo(50, 107);
```

CALLBACKS

Como já vimos, quase tudo são métodos. No entanto, apesar dos métodos especificos que temos que programar de cada vez que os queremos utilizar, existe na sintaxe do Processing alguns métodos implementados de raiz que já têm o comportamento de retorno pelo que não os precisamos de invocar, basta implementar. São estes:

`setup()`

Invocado pelo próprio Processing antes do programa iniciar. É invocado só uma vez e serve para preparamos o programa – atributos como o tamanho e cor de fundo, framerate, atribuir valores a variáveis, etc.

`draw()`

Quando implemtentado, o draw() é invocado automaticamente pelo processing sempre que acaba de o executar ou logo a seguir ao setup(). Caso não haja nada em contrario, o Processing invoca o *loop* do draw() sempre que possível pelo computador.

```
keyPressed()
keyReleased()
mousePressed()
mouseMoved()
mouseDragged()
mouseReleased()
stop()
```

CLASSES E OBJECTOS

Fica para outra ocasião, mas, só em tom de nota, fica aqui um pequeno resumo do que são. Em Processing, assim como em quase todas as linguagens orientadas a objectos (OOP), existe para alem dos dados simples, dados complexos definidos pelo programador que são armazenados em *objectos*. Um Objecto é então uma estrutura de dados e funcionalidade. Ou seja, alem de um conjunto de dados (que podem ser de natureza diferente) pode ter associados métodos próprios. Em Processing a definição deste tipo de objectos é feita através de *Classes*. Para inicializarmos estes objectos (já definidos no programa) recorremos a construtores, a um em especial – `new`.

TEXTO

O texto - `String()` - é um tipo de dados ainda não abordado. Basicamente toda a informação que queremos representar ou manipular como informação temos que a manipular com a classe `String()`.

```
String meuNome = new String("Pedro");
Println(meuNome);
```

As *strings* de texto tem que ser escritas entre aspas, de resto comportam-se exactamente como qualquer outro tipo de dados.

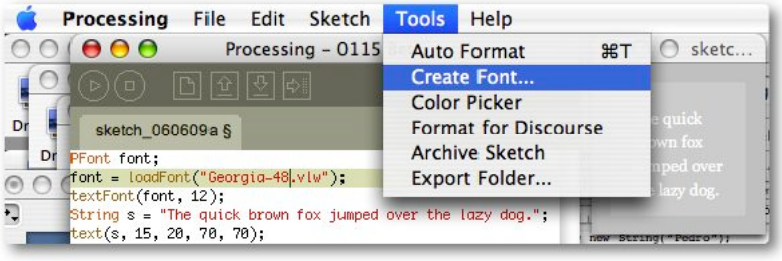


FIGURA 58 - CRIAR UMA FONTE (MENU)

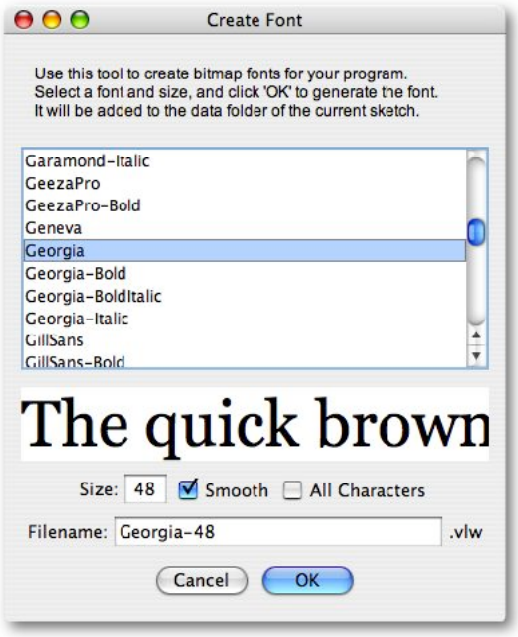


FIGURA 59 - CRIAR UMA FONTE (.VLW)

http://www.processing.org/reference/text_.html

FIGURA 60 – TEXTO

```
PFont font;
background(0);
font = loadFont("Georgia-48.vlw");
textFont(font, 12);
String s = "The quick brown fox jumped over the lazy dog.";
text(s, 15, 20, 70, 70);
```

COMENTÁRIOS E DOCUMENTAÇÃO

Os comentários são pedaços de texto no código que não afectam o programa. Existem porque quem programa não é uma máquina, mas sim pessoas – os comentários servem para ajudar as pessoas a programar fazendo referencias, explicando o código (que não é óbvio), documentando funções para outros, ou apenas para futura referencia. Existem duas formas de comentários:

SIMPLES OU DE LINHA.

`// <comentário>`

Tudo o que se segue nessa linha é ignorado pelo compilador.
Serve basicamente para pequenos apontamentos e descrições de funções.

```
println("comentário"); //imprime "comentário"
```

BLOCO OU MULTI-LINHA.

`/* <comentário> */`

Serve principalmente para reportar grandes explicações no código e é usado para documentação.

```
/*
Programa que escreve nomes.
Desenvolvido por Pedro Amado
2006.06.05

Basta mudar a variável nome para o nome que queremos.
*/

String nome = new String("Pedro"); // o meu nome
println(nome);
```

ESTRUTURA TÍPICA DE UM PROGRAMA

Descrição do programa
Declaração de variáveis
Acções

EXEMPLOS

(a executar oralmente)
Ligar um automóvel.
Calcular a média de idades da turma.
Determinar se um número é positivo, negativo, ou zero.

EXERCÍCIOS

1. Pensar num programa que possa ser facilmente implementado por um colega seu na sala de aula. Escrever esse programa. O programa deve ter mais de 10 instruções, deve ter pelo menos uma estrutura de selecção ou iteração e deve poder ser executado em menos de 5 minutos.

2. Escreva o programa anterior com um nível de detalhe maior – cada acção deve ser decomposta em acções mais pequenas.

3. Agrupar as novas instruções em conjuntos de instruções maiores e diferentes do original.

PONG

Comentários
Inicialização de variáveis
Posição e incremento
Setup
Dimensões
    Atribuição da variável posição e incremento
Draw
    Incremento mais posição
    Verificação de posição
    Condição de incremento

PRÁTICA I (EXPOSIÇÃO TEÓRICO-PRÁTICA)

Neste capítulo vamos desenvolver uma aplicação de maniulação de dados – Data APP (Modo de Texto). De seguida vamos converter essa mesma aplicação numa outra que use os dados recolhidos e os dados manipulados traduzindo-os em elementos visuais. Pretende-se com isto introduzir os elementos básicos do processing – Modo de utilização, Variáveis, Primitivas, Ciclos, Condições, Operadores, Vectores e utilização básica de Tipogafia (Objectos PFont).

Recapitulação e lista integral de funções, estrutura e sintaxe do Processing:
http://processing.org/reference/index.html
http://processing.org/learning/index.html
http://processing.org/faq/index.html

APLICAÇÃO DE DADOS (DATA APP - CÁLCULO DE MÉDIAS)

OBJECTIVOS

1. Criar uma aplicação que calcule a média das idades das pessoas presentes na formação – Dados brutos e individuais (*hardcode*);
    a. Dados (Data Types)
    b. Variáveis (declaração, inicialização, invocação e manipulação)
    c. Interacção com o developer (passar mensagens);
2. Fazer com que a aplicação represente esses dados graficamente através de formas simples;
    a. Primitivas simples (rect() e ellipse())
    b. Cores (fill() e color())
    c. Representação da posição do próprio utilizador
3. Optimização do processo da aplicação e criação de uma interface gráfica que revele a meta-informação subjacente à representação gráfica dos dados:
    a. Imagem de fundo da Interface (PImage);
    b. Nome da Aplicação (PFont);
    c. Data (date());
    d. Passagem de dados brutos a vectores (texto e integrais);
    e. Ciclos de desenho (for());
    f. Verificação da posição do próprio dentro da média de idades (condições – if() else(), case();
    g. Representação dos nomes relativos a cada dado/idade;

SINTAXE BÁSICA DO PROCESSING

MODO SIMPLES

Comentários Simples e Multi-linha;
print() e println();
size();
background();
fill(); vs. nofill();
stroke();
point();
line();
rect();
ellipse();

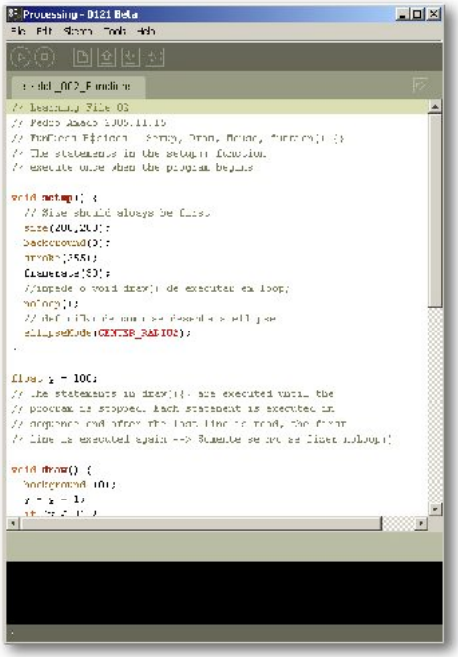


FIGURA 61 – ESTRUTURA E SINTAXE: SKETCH_001_ESTRUTURA_SINTAXE.PDE

```
// Learning File 01
// Pedro Amado 2005.11.15

// Estrutura e Sintaxe
// -------------------

// Exemplo de um Comentário Simples
// ---------------
```

```
// Interface com o próprio Processing
// ----------------------------------
//print("Olá");
println("Olá!");
print("Processing preparado para arrancar.");


// Basic Coordinates & Shapes
// --------------------------
size (300, 300);
background(0);
noFill();
stroke(255);
point(50,50);
line(55,55,100,100);
line(0,105,width,105);
/*
 Exemplo de um comentário multi-linha
 Aqui o width refere-se ao valor total do sketch.
 Logo o valor do right rect vai ser adicionado ao lecft rect
 Por isso para ser 10px de cada borda é somar 10 no início e subtrair 20 no
fim
 */
rect(10,125,width-20,150);
```

MODO CONTÍNUO

Callbacks;

Framerate();

Loop() vs. noLoop();

ellipseMode();

rectMode();

setup();

draw();

mousePressed();

Médodos Personalizados;

Parâmetros;

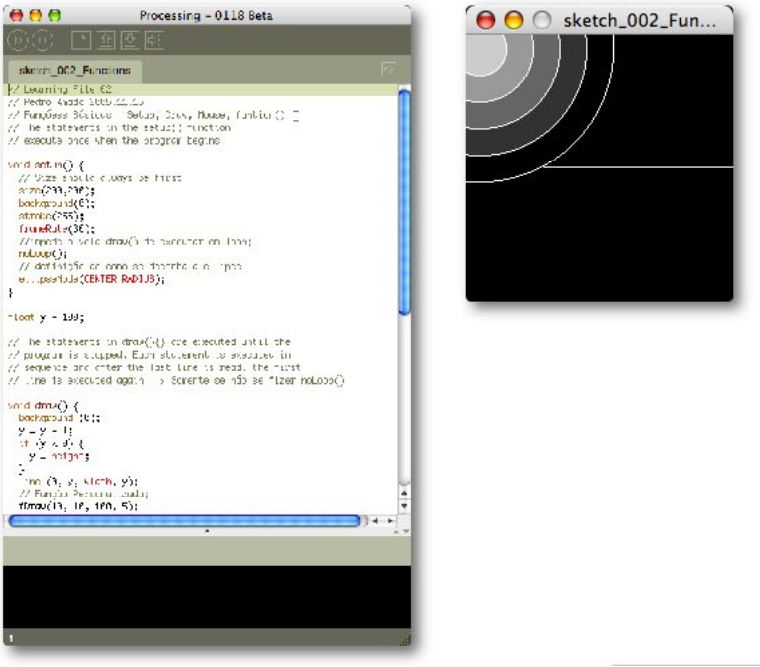


FIGURA 62 – MODO CONTÍNUO (*CALLBACKS*): SKETCH_002_FUNCTIONS.PDE

```
// Learning File 02
// Pedro Amado 2005.11.15
// Funçõess Básicas - Setup, Draw, Mouse, funtion() {}
// The statements in the setup() function
// execute once when the program begins

void setup() {
  // Size should always be first
  size(200,200);
  background(0);
  stroke(255);
  framerate(30);
  //impede o void draw() de executar em loop;
  noLoop();
  // definição de como se desenha a ellipse
  ellipseMode(CENTER_RADIUS);
}

float y = 100;
// The statements in draw(){} are executed until the
// program is stopped. Each statement is executed in
// sequence and after the last line is read, the first
// line is executed again --> Somente se não se fizer noLoop()

void draw() {
  background (0);
  y = y - 1;
  if (y < 0) {
    y = height;
  }
  line (0, y, width, y);
  // Função Personaliozada;
```

```
  fDraw(10, 10, 100, 5);
}

// Funções específicas(){};
void mousePressed() {
  //Isto anula o noLoop() invocado no void setup()
  //loop();
  // Executa a função (void draw()) uma única vez e não anula o noLoop();
  //redraw();
}

// Definição de Funções personalizadas que são chamadas previamente;
// Função(parametros, parametros,...)
void fDraw(int xloc, int yloc, int size, int num) {
  float grayvalues = 255/num;
  float steps = size/num;
  for(int i=0; i<num; i++) {
    fill(i*grayvalues);
    ellipse(xloc, yloc, size-i*steps, size-i*steps);
  }
}
```

DECLARAÇÃO E ATRIBUIÇÃO DE VARIÁVEIS

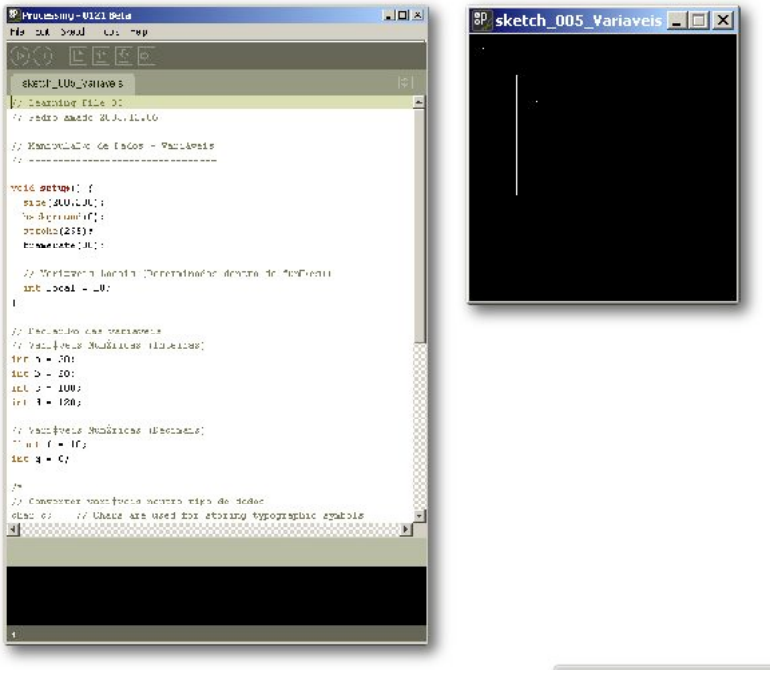
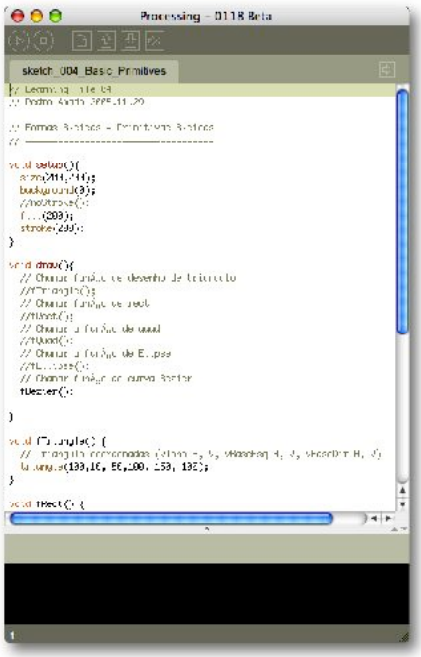


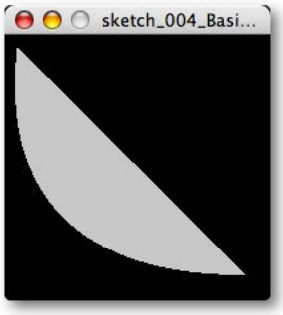
FIGURA 63 – VARIÁVEIS: SKETCH_005_VARIAVEIS.PDE

```
// Learning File 05
// Pedro Amado 2005.12.06

// Manipulação de Dados - Vari·veis
// --------------------------------
```

```
void setup() {
  size(200,200);
  background(0);
  stroke(255);
  framerate(30);

  // Variáveis Locais (Determinadas dentro de funções()
  int local = 10;
}

// Declarção das variaveis
// Variáveis Numéricas (Inteiras)
int a = 20;
int b = 20;
int c = 100;
int d = 120;

// Variáveis Numéricas (Decimais)
float f = 10;
int g = 0;

/*
// Converter variáveis noutro tipo de dados
char c;    // Chars are used for storing typographic symbols
float f;   // Floats are decimal numbers
int i;     // Ints are values between 2,147,483,647 and -2147483648
byte b;    // Bytes are values between -128 and 128
*/

// Variáveis Globais (Global Scope)
int global = 50;

void draw() {
background(0);
  //line(a, b/2, c*1.5, d+80);
  f = f + 0.1;
  g = int(f)*2;
  line(a+f, g, a+f, b+100);

  // Variáveis Locais (Determinadas dentro de funções()
  int local = 10;
  point(local, local);
  point(global, global);
}
```

PRIMITIVAS





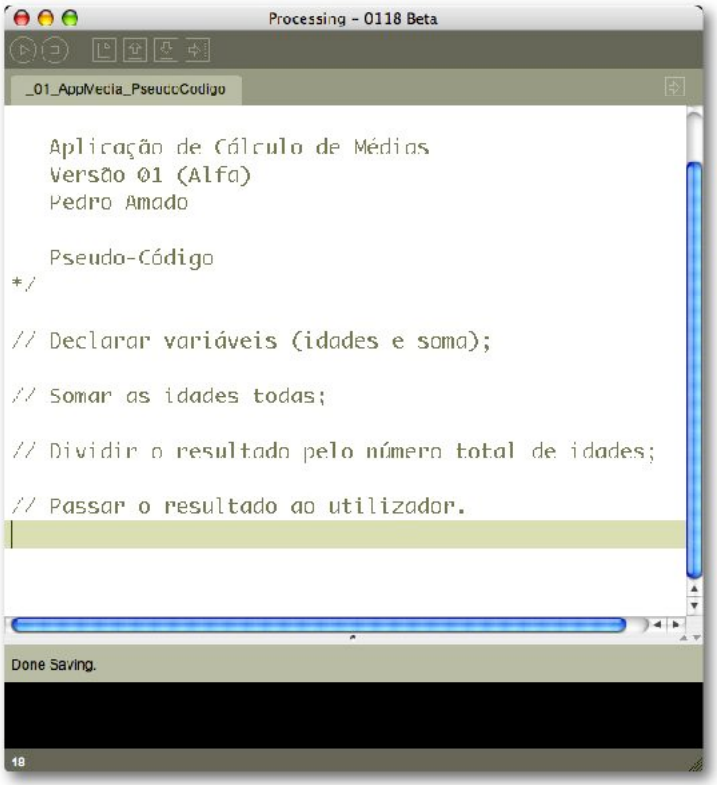

```
// Learning File 04
// Pedro Amado 2005.11.29

// Formas B·sicas - Primitivas B·sicas
// ------------------------------------

void setup(){
  size(200,200);
  background(0);
  //noStroke();
  fill(200);
  stroke(200);
}

void draw(){
  // Chamar funÁ„o de desenho de triangulo
  //fTriangle();
  // Chamar funÁ„o de rect
  //fRect();
  // Chamar a funÁ„o de quad
  //fQuad();
  // Chamar a funÁ„o de Elipse
  //fEllipse();
  // Chamar funÁ„o de curva Bezier
  fBezier();

}

void fTriangle() {
  // Triangulo coordenadas (vTopo H, V, vBaseEsq H, V, vBaseDir H, V)
  triangle(100,10, 50,100, 150, 100);
}

void fRect() {
  // Rect coordenadas (PontoSupEsq H, V, largura, altura)
```

```
  rect(45, 45, 100, 10);
}

void fQuad() {
  // Quad coordenadas (PontoSupEsq H, V, SupDir H, V, InfDir H, V, InfEsq H,
V)
  quad (50,100, 100,100, 100, 150, 40, 160);
}

void fEllipse() {
  // Ellipse coordenadas (Centro H, V, Raio H, V)
  ellipse(100, 50, 50, 150);
}

void fBezier() {
  // Bezier coordenadas (Ponto1 H, V, BCP1 H, V, BCP2 H, V, Ponto2 H, V)
  bezier(10, 10, 0, 100, 35, 180, 180, 180);
}
```

APLICAÇÃO DE DADOS (DATA APP - CÁLCULO DE MÉDIAS)

PSEUDO-CÓDIGO 1

(desenhar em modo contínuo, 500 x 300 px)



1. Declarar variáveis (idades e soma);

```
// Declarar variáveis
int tiago = 19; // declarar e inicializar
int maria = 27;
int ana = 23;
int soma; // só declarar
```

2. Somar as idades todas;

```
void setup() {
  size(500, 300);
  background(0);
  soma = 0;//inicializar a soma
  soma = tiago + maria + ana;// Somar as idades todas;
```

3. Dividir o resultado pelo número total de idades;

```
int nidades = 3;//total de idades
float media = soma/nidades;// Dividir o resultado pelo número to
```

4. Passar o resultado ao utilizador.

```
  // Passar os resultados ao utilizador
  println("Total de idades ="+ soma);
  println("Número de idades somadas = "+ nidades);
  println("-------------------------------");
  println("Média das idades = " + media);
}

void draw() {

}
```

CÓDIGO TOTAL (DATA APP – PRINTLN() TEXT MODE)

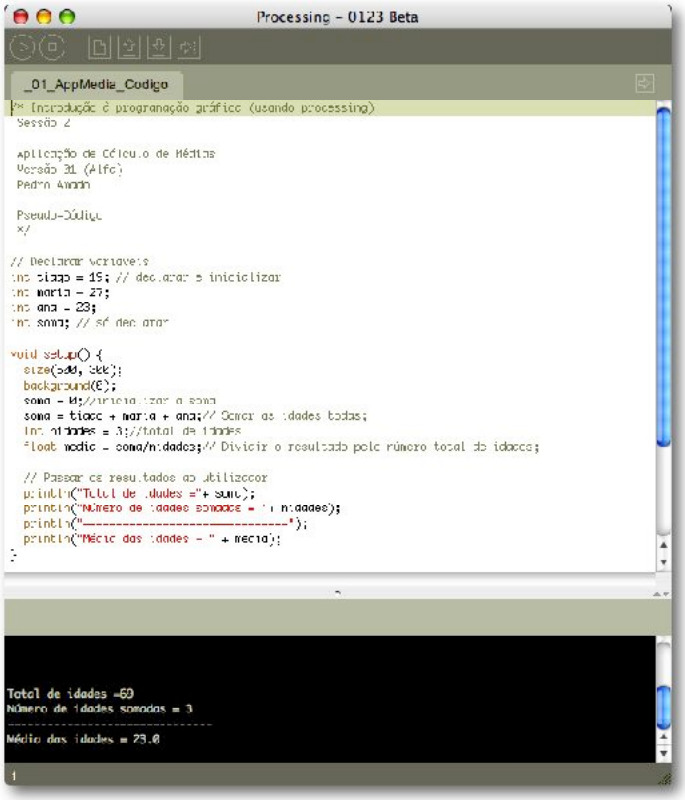
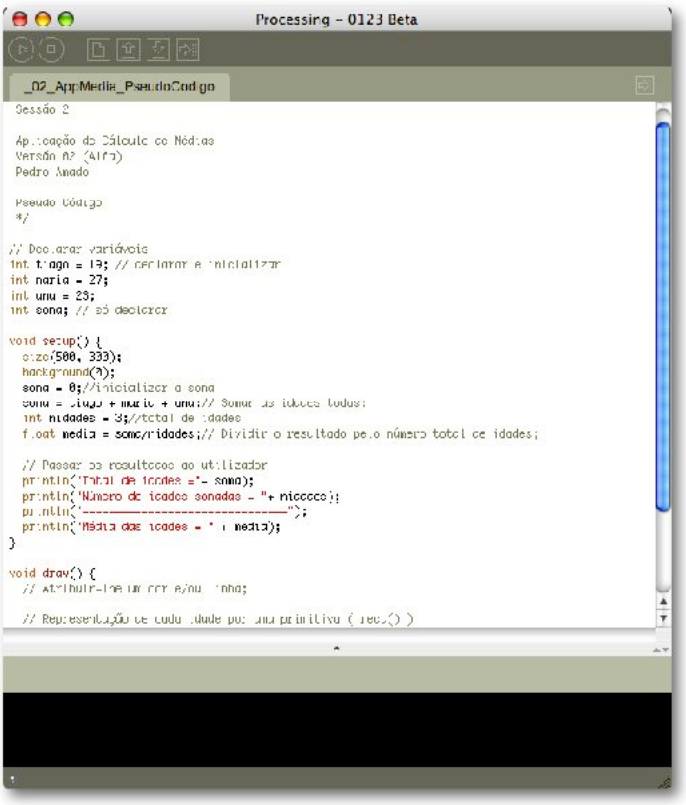


FIGURA 64 - _01_APPMEDIA_CODIGO

```
/* Introdução à programação gráfica (usando processing)
 Sessão 2

 Aplicação de Cálculo de Médias
 Versão 01 (Alfa)
 Pedro Amado

 Pseudo-Código
 */

// Declarar variáveis
int tiago = 19; // declarar e inicializar
int maria = 27;
int ana = 23;
int soma; // só declarar

void setup() {
  size(500, 300);
  background(0);
  soma = 0;//inicializar a soma
  soma = tiago + maria + ana;// Somar as idades todas;
  int nidades = 3;//total de idades
  float media = soma/nidades;// Dividir o resultado pelo número total de
idades;

  // Passar os resultados ao utilizador
  println("Total de idades ="+ soma);
```

```
  println("Número de idades somadas = "+ nidades);
  println("-------------------------------");
  println("Média das idades = " + media);
}

void draw() {

}
```

PSEUDO-CÓDIGO 2



(desenhar em modo contínuo, 500 x 300 px aproveitando a anterior)

```
void draw() {
  // Atribuir-lhe um cor e/ou linha;

  // Representação de cada idade por uma primitiva ( r

  // Representação do valor médio encontrado através d

  // Representação da própria idade através de uma pr
}
```

5. Atribuir-lhe um cor e/ou linha;

```
// Atribuir-lhe um cor e/ou linha;
stroke(255);
fill(100,100, 100, 150);
```

6. Representação de cada idade por uma primitiva (rect())

```
// Representação de cada idade por uma primitiva
rectMode(CENTER);
rect(tiago*10, 150, 5, 100);
rect(maria*10, 150, 5, 100);
rect(ana*10, 150, ana+5, 100);
```

7. Representação do valor médio encontrado através de uma primitiva de outra cor;

```
// Representação do valor médio encontrado através
int nidades = 3;//total de idades
float media = soma/nidades;// Dividir o resultado p
stroke(255, 0, 0);
line(media*10, 100, media*10, 200);
```

8. Representação da própria idade através de uma primitiva e de um texto diferente diferente;

```
// Representação da própria idade através de uma
ellipseMode(CENTER);
ellipse(tiago*10, 150, 10, 10);
```

CÓDIGO TOTAL (DATA APP – PRIMITIVAS)

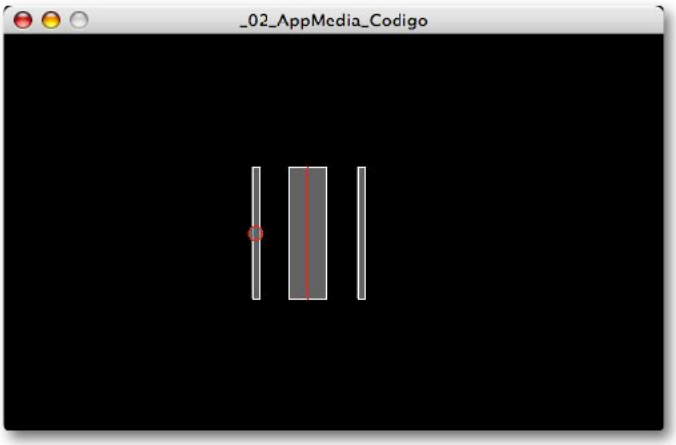


FIGURA 65 - DATA APP (PRIMITIVAS)

```
/* Introdução à programação gráfica (usando processing)
 Sessão 2

 Aplicação de Cálculo de Médias
 Versão 02 (Alfa)
 Pedro Amado

 Pseudo-Código
 */

// Declarar variáveis
int tiago = 19; // declarar e inicializar
int maria = 27;
int ana = 23;
int soma; // só declarar

void setup() {
  size(500, 300);
  background(0);
  soma = 0;//inicializar a soma
  soma = tiago + maria + ana;// Somar as idades todas;
  int nidades = 3;//total de idades
  float media = soma/nidades;// Dividir o resultado pelo número total de
idades;

  // Passar os resultados ao utilizador
  println("Total de idades ="+ soma);
  println("Número de idades somadas = "+ nidades);
  println("--------------------------------");
  println("Média das idades = " + media);
}

void draw() {
  // Atribuir-lhe um cor e/ou linha;
  stroke(255);
  fill(100,100, 100, 150);
  // Representação de cada idade por uma primitiva ( rect() )
  rectMode(CENTER);
  rect(tiago*10, 150, 5, 100);
  rect(maria*10, 150, 5, 100);
```

```
  rect(ana*10, 150, ana+5, 100);
  // Representação do valor médio encontrado através de uma primitiva de
outra cor;
  int nidades = 3;//total de idades
  float media = soma/nidades;// Dividir o resultado pelo número total de
idades;
  stroke(255, 0, 0);
  line(media*10, 100, media*10, 200);
  // Representação da própria idade através de uma primitiva e de um texto
diferente diferente;
  ellipseMode(CENTER);
  ellipse(tiago*10, 150, 10, 10);
}
```

SINTAXE BÁSICA DO PROCESSING

CICLOS OU ITERAÇÕES

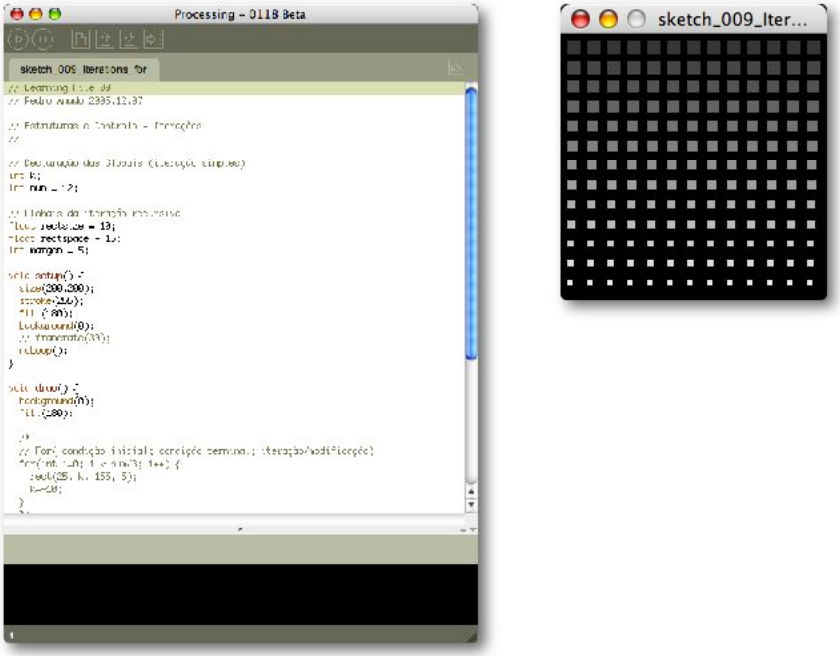


FIGURA 66 - ITERAÇÕES: SKETCH_009_ITERATIONS_FOR.PDE

```
// Learning File 09
// Pedro Amado 2005.12.07

// Estruturas e Controlo - Iterações
// ---------------------------------

// Declaração das Globais (iteração simples)
int k;
int num = 12;

// Globais da iteração recursiva
float rectsize = 10;
float rectspace = 15;
int margem = 5;

void setup() {
  size(200,200);
  stroke(255);
  fill(180);
  background(0);
  // framerate(30);
  noLoop();
}
```

```
void draw() {
  background(0);
  fill(180);
  k=60;

  /*
  // For( condição inicial; condição terminal; iteração/modificação)
  for(int i=0; i < num/3; i++) {
    rect(25, k, 155, 5);
    k+=10;
  }
  */

  // Exemplo de estrutura/iteração recursiva
  // Linha horizontal (primeira condição)
  for(int x = margem; x <= width-margem; x += rectspace){
    noStroke();
    fill(x+50);
    // Coluna vertical (repete para cada linha)
    for(int v = margem; v <= height-margem; v += rectspace){
      rect(v, x, rectsize, rectsize);
    }
    rectsize = rectsize - 0.5;
  }
}
```

CONDIÇÕES

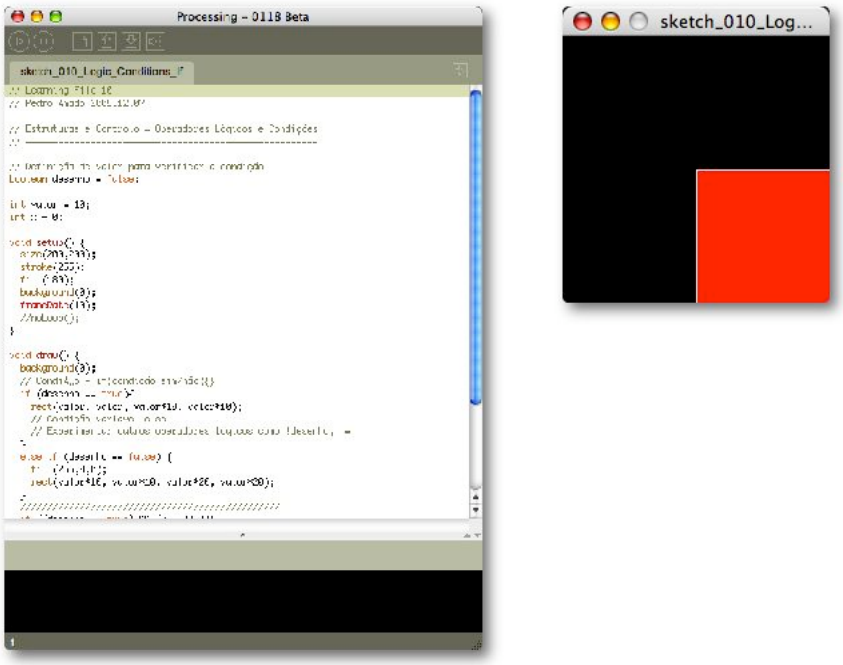


FIGURA 67 - CONDIÇÕES: SKETCH_010_LOGIC_CONDITIONS_IF.PDE

```
// Learning File 10
// Pedro Amado 2005.12.07

// Estruturas e Controlo - Operadores Lógicos e Condições
// ------------------------------------------------------

// Definiçãa de valor para verificar a condição
boolean desenho = false;
int valor = 10;
int x = 0;
void setup() {
  size(200,200);
  stroke(255);
  fill(180);
  background(0);
  framerate(30);
  //noLoop();
}

void draw() {
  background(0);
  // CondiÁ„o = if(condição sim/não){}
  if (desenho == true){
    rect(valor, valor, valor*10, valor*10);
    // Condição variavel else
    // Experimentar outros operadores logicos como !desenho, !=
  }
  else if(desenho == false) {
    fill(255,0,0);
    rect(valor*10, valor*10, valor*20, valor*20);
  }

  if ((desenho == true) && (x >=30 )){
    rect(110,110,10,10);
  }
  else if ((desenho == false) && ((x <=30 ) || (x>=60))) {
    rect(0,0,10,10);
  }
  x+=1;
}
```

VECTORES

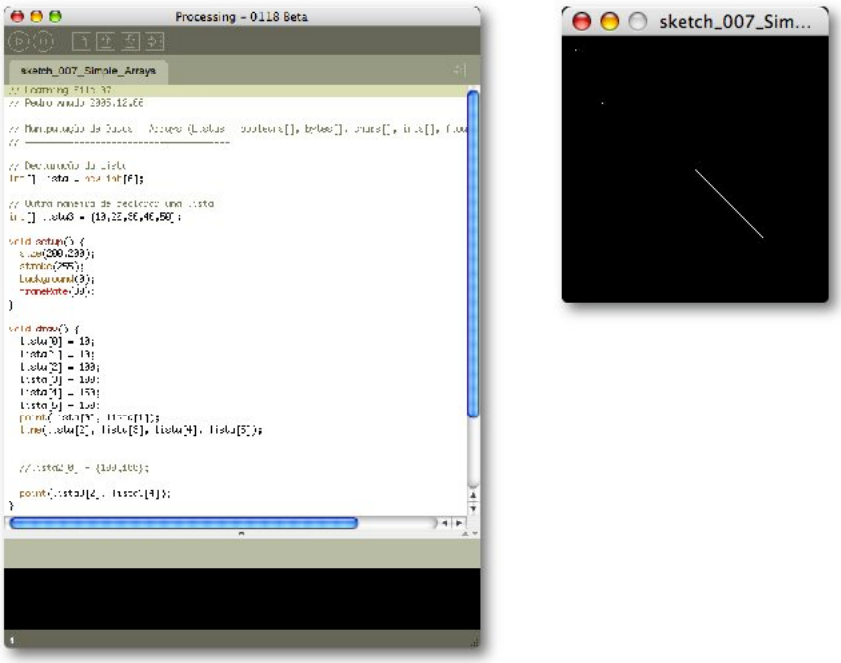


FIGURA 68 - VECTORES: SKETCH_007_SIMPLE_ARRAYS.PDE

```
// Learning File 07
// Pedro Amado 2005.12.06

// Manipulação de Dados - Arrays (Listas - booleans[], bytes[], chars[],
ints[], floats[], or Strings[])
// ---------------------------------------

// Declaração da Lista
int[] lista = new int[6];

// Outra maneira de declarar uma lista
int[] lista3 = {10,20,30,40,50};

void setup() {
  size(200,200);
  stroke(255);
  background(0);
  framerate(30);
}

void draw() {
  lista[0] = 10;
  lista[1] = 10;
  lista[2] = 100;
  lista[3] = 100;
  lista[4] = 150;
  lista[5] = 150;
  point(lista[0], lista[1]);
  line(lista[2], lista[3], lista[4], lista[5]);

  //lista2[0] = {100,100};
```

```
  point(lista3[2], lista3[4]);
}
```

OPERADORES MATEMÁTICOS

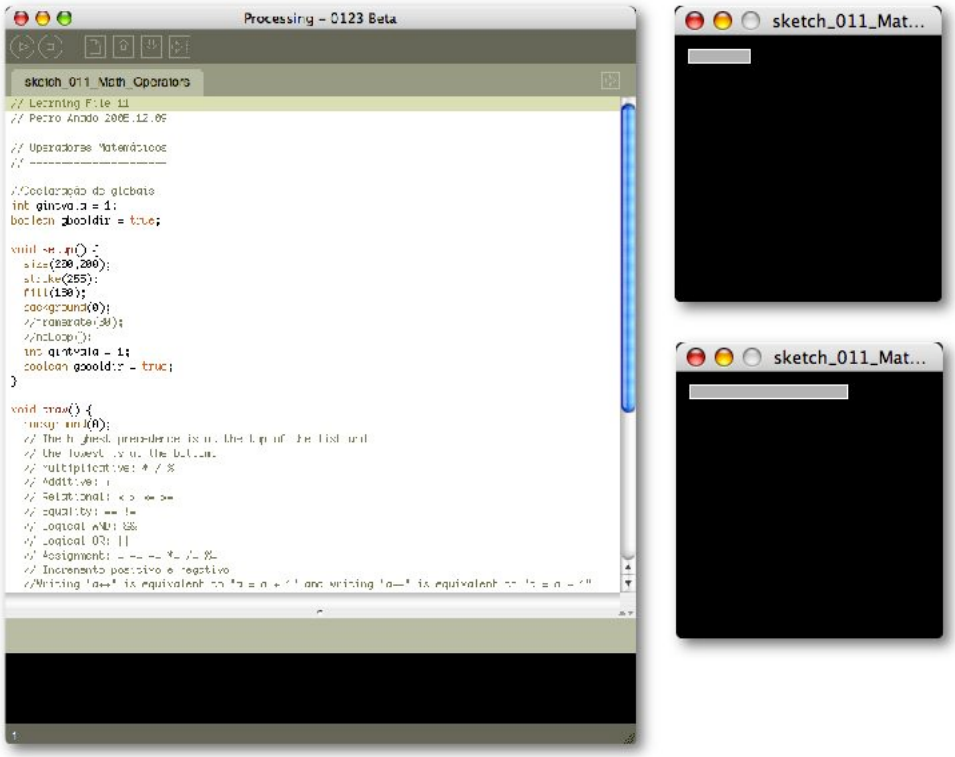


FIGURA 69 - OPERADORES MATEMÁTICOS: SKETCH_011_MATH_OPERATORS.PDE

```
// Learning File 11
// Pedro Amado 2005.12.09

// Operadores Matemáticos
// ----------------------

//Declaração de globais
int gintvala = 1;
boolean gbooldir = true;

void setup() {
  size(200,200);
  stroke(255);
  fill(180);
  background(0);
  //framerate(30);
  //noLoop();
  int gintvala = 1;
  boolean gbooldir = true;
}

void draw() {
  background(0);
```

```
  // The highest precedence is at the top of the list and
  // the lowest is at the bottom.
  // Multiplicative: * / %
  // Additive: + -
  // Relational: < > <= >=
  // Equality: == !=
  // Logical AND: &&
  // Logical OR: ||
  // Assignment: = += -= *= /= %=
  // Incremento positivo e negativo
  //Writing "a++" is equivalent to "a = a + 1" and writing "a--" is
equivalent to "a = a - 1"
  if(gbooldir == true) {
    gintvala++;
    // gintvala+=5;
  }
  else if(gbooldir == false){
    gintvala--;
  }
  if (gintvala > width){
    gbooldir = false;
  }
  else if (gintvala < 1) {
    gbooldir = true;
  }
  rect(10, 10, gintvala, 10);
}
```

TIPOGRAFIA BÁSICA

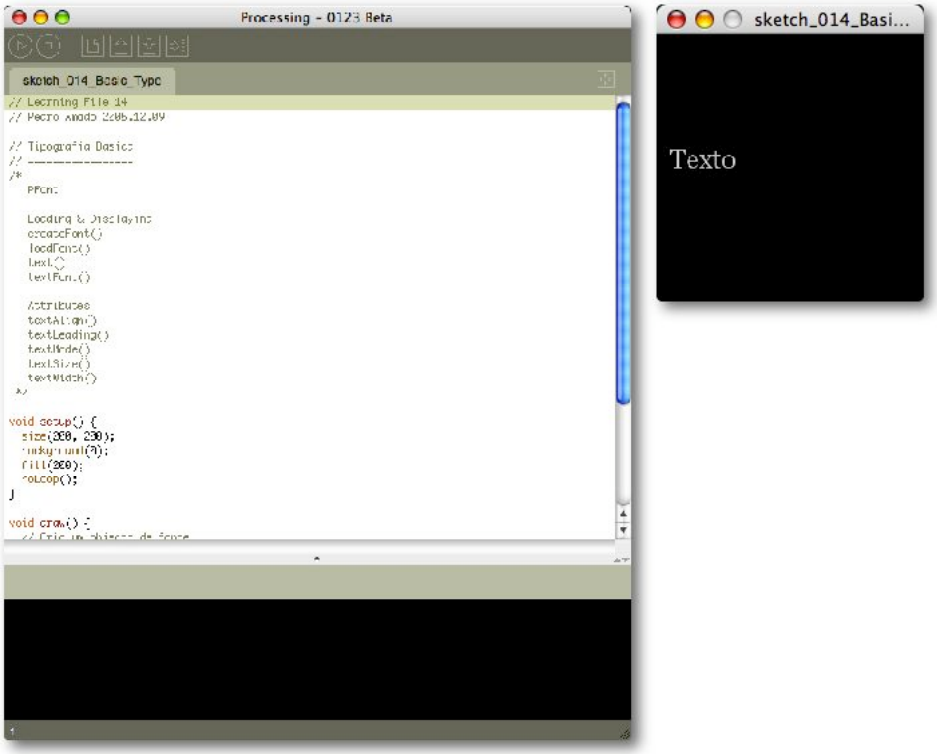


FIGURA 70 - TIPOGRAFIA BÁSICA: SKETCH_014_BASIC_TYPE.PDE

```
// Learning File 14
// Pedro Amado 2005.12.09

// Tipografia Básica
// -----------------
/*
   PFont

   Loading & Displaying
   createFont()
   loadFont()
   text()
   textFont()

   Attributes
   textAlign()
   textLeading()
   textMode()
   textSize()
   textWidth()
 */

void setup() {
  size(200, 200);
  background(0);
  fill(200);
  noLoop();
}
```

```
void draw() {
  // Cria um objecto de fonte
  PFont fgeorgia;
  // Invoca/load Fonte
  fgeorgia = loadFont("Georgia-48.vlw");
  textFont(fgeorgia);
  textSize(20);
  textAlign(LEFT);
  text("Texto", 10, 100);
}
```

APLICAÇÃO DE DADOS (DATA APP - CÁLCULO DE MÉDIAS)

PSEUDO-CÓDIGO 3

(re-desenhar em modo contínuo, 500 x 300 px)

```
// Declarar variáveis:
//   Soma e media de todas as idades;
//   Vector de todas as idades;
//   Vector de todos os nomes;
//   Meu nome/idade (posição nos vectores)
//   Data a usar
//   Nome da aplicação a usar;
//   Imagem a usar de fundo;
//Texto a desenhar em ecrã

void setup() {
  //inicializar variáveis;
  // Calcular a soma e a média e armazenar em memoria
  // Inicializar imagem (load image - data folder)
  // Inicializar texto (create font)

}

void draw() {
  // Desenhar a imagem de fundo;
  // Desenhar o Nome e Data da Aplicação;
  // Representar as idades todas e média (ciclos);
  //   Desenho de primitivas;
  //   Desenhar Área de gráfico
  //   Desenho das idades
    //   Verificação da idade ou nome próprio para o re
    //   Atribuição de nome a cada idade representada;
    //    Desenho da média
  //   Obter o resultado  médio da soma de todos os ele
  //   Desenho da média (primitivas);
  //   Atribuição de nome da média;
}
```

PRÉ-SETUP

1. Declarar variáveis:
    a. Soma e media de todas as idades;

```
// Declarar variáveis:
//   Soma e media de todas as idades;
int soma, media, total;
```

    b. Vector de todas as idades;

```
//   Vector de todas as idades;
int[] idades = {19, 21, 12, 22, 30, 17, 18, 45, 23, 23};
```

    c. Vector de todos os nomes;

```
//   Vector de todos os nomes;
String[] nomes = {"Ana", "Tiago", "Pedro", "Maria", "Susana", "Silvana",
```

    d. Meu nome/idade (posição nos vectores)

```
//   Meu nome/idade (posição nos vectores)
int meunome;
```

    e. Data a usar
    f. Nome da aplicação a usar;

```
//   Data a usar
//   Nome da aplicação a usar;
String data, nome;
```

2. Declarar objectos
    a. Imagem a usar de fundo;
    b. Texto a desenhar em ecrã

```
//   Imagem a usar de fundo;
PImage bg;
//Texto a desenhar em ecrã
PFont texto;
```

SETUP

1. Inicializar variáveis;

```
//inicializar variáveis;
soma = 0;
media = 0;
total = idades.length;
meunome = 5;
nome = "Aplicação Gráfica de Médias";
int dia = day();
int mes = month();
int ano = year();
data = "Porto, "+ ano +"-"+ mes +"-"+ dia;
//println(nomes[meunome]+idades[meunome]);
//println(data);
```

2. Calcular a soma e a média e armazenar em memoria (ciclos);

```
// Calcular a soma e a média e armazenar em memoria (ciclos);
for (int n = 0; n<total; n++) {
  soma = soma+idades[n];
}
media = soma/total;
//println(media);
```

3. Inicializar imagem (load image - data folder)*

```
/*
PImage b;
 b = loadImage("exemplo.jpg");
 image(b, x, y);
 */
bg = loadImage("background.png");
```

4. Inicializar texto (create font)*

```
/*
PFont font;
 // The font must be located in the sketch's
 // "data" directory to load successfully
 font = loadFont("exemplo.vlw");
 textFont(font, int size);
 text("exemplo", x, y);
 */

texto = loadFont("MyriadPro-Regular-14.vlw");
textFont(texto, 14);
```

DRAW

1. Desenhar a imagem de fundo;

```
// Desenhar a imagem de fundo;
image(bg, 0, 0);
```

2. Desenhar o Nome e Data da Aplicação;

```
// Desenhar o Nome e Data da Aplicação;
smooth();
fill(255);
text(("Pedro Amado - "+data), 10, 28);
```

3. Representar as idades todas e média (ciclos);
   a. Desenho de primitivas;
   b. Desenhar Área de gráfico

```
// Representar as idades todas e média (ciclos);
//   Desenho de primitivas;
//   Área de gráfico
noStroke();
rectMode(CORNER);
fill(255, 100);
rect(0, 200, 500, 100);
```

   c. Desenho das idades
   d. Verificação da idade ou nome próprio para o representar de forma diferente (condições)

```
//   Desenho das idades
rectMode(CENTER);
fill(255, 100);
noStroke();
for (int n = 0; n<total; n++) {
  //   Verificação da idade ou nome próprio para o representar de
  if (n == meunome) {
    fill(255, 255, 0, 100);
    rect(idades[n]*10, 250, 5, 100);
  }
  else {
    fill(255, 100);
    rect(idades[n]*10, 250, 5, 100);
  }
```

e. Atribuição de nome a cada idade representada;

```
//   Atribuição de nome a cada idade representada;
fill(255);
text(nomes[n], idades[n]*10, 200+n*10);
```

4. Desenho da média
   a. Obter o resultado médio da soma de todos os elementos;
   b. Desenho da média (primitivas);
   c. Atribuição de nome da média;

```
//    Desenho da média
//   Obter o resultado  médio da soma de todos os elementos;
//   Desenho da média (primitivas);
stroke(255, 0, 0);
strokeWeight(2);
strokeCap(SQUARE);
line(media*10, 200, media*10, 300);
//   Atribuição de nome da média;
fill(255,0,0);
text("Média de idades", media*10, 250);
```

CÓDIGO TOTAL (DATA APP – MODO GRÁFICO)

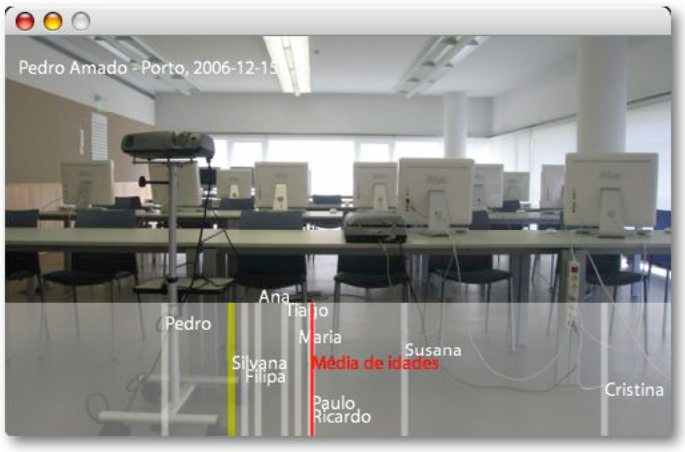


FIGURA 71 - DATA APP (MODO GRÁFICO)

```
/* Introdução à programação gráfica (usando processing)
 Sessão 2

 Aplicação de Cálculo de Médias
 Versão 03 (Alfa)
 Pedro Amado

 Código
 */

// Declarar variáveis:
//   Soma e media de todas as idades;
int soma, media, total;
//   Vector de todas as idades;
int[] idades = {
  19, 21, 12, 22, 30, 17, 18, 45, 23, 23};
//   Vector de todos os nomes;
String[] nomes = {
  "Ana", "Tiago", "Pedro", "Maria", "Susana", "Silvana", "Filipa",
"Cristina", "Paulo", "Ricardo"};
//   Meu nome/idade (posição nos vectores)
int meunome;
//   Data a usar
//   Nome da aplicação a usar;
String data, nome;
//   Imagem a usar de fundo;
PImage bg;
//Texto a desenhar em ecrã
PFont texto;

void setup() {
  size(500, 300);
  background(255);
  //inicializar variáveis;
  soma = 0;
```

```
  media = 0;
  total = idades.length;
  meunome = 5;
  nome = "Aplicação Gráfica de Médias";
  int dia = day();
  int mes = month();
  int ano = year();
  data = "Porto, "+ ano +"-"+ mes +"-"+ dia;
  //println(nomes[meunome]+idades[meunome]);
  //println(data);

  // Calcular a soma e a média e armazenar em memoria (ciclos);
  for (int n = 0; n<total; n++) {
    soma = soma+idades[n];
  }
  media = soma/total;
  //println(media);

  /*
  PImage b;
   b = loadImage("exemplo.jpg");
   image(b, x, y);
   */
  bg = loadImage("background.png");

  /*
  PFont font;
   // The font must be located in the sketch's
   // "data" directory to load successfully
   font = loadFont("exemplo.vlw");
   textFont(font, int size);
   text("exemplo", x, y);
   */

  texto = loadFont("MyriadPro-Regular-14.vlw");
  textFont(texto, 14);

}

void draw() {
  // Desenhar a imagem de fundo;
  image(bg, 0, 0);

  // Desenhar o Nome e Data da Aplicação;
  smooth();
  fill(255);
  text(("Pedro Amado - "+data), 10, 28);

  // Representar as idades todas e média (ciclos);
  //   Desenho de primitivas;
  //   Área de gráfico
  noStroke();
  rectMode(CORNER);
  fill(255, 100);
  rect(0, 200, 500, 100);

  //   Desenho das idades
  rectMode(CENTER);
  fill(255, 100);
  noStroke();
  for (int n = 0; n<total; n++) {
```

```
    //   Verificação da idade ou nome próprio para o representar de forma
diferente (condições)
    if (n == meunome) {
      fill(255, 255, 0, 100);
      rect(idades[n]*10, 250, 5, 100);
    }
    else {
      fill(255, 100);
      rect(idades[n]*10, 250, 5, 100);
    }
    //   Atribuição de nome a cada idade representada;
    fill(255);
    text(nomes[n], idades[n]*10, 200+n*10);
  }

  //    Desenho da média
  //   Obter o resultado  médio da soma de todos os elementos;
  //   Desenho da média (primitivas);
  stroke(255, 0, 0);
  strokeWeight(2);
  strokeCap(SQUARE);
  line(media*10, 200, media*10, 300);
  //   Atribuição de nome da média;
  fill(255,0,0);
  text("Média de idades", media*10, 250);
}
```

APLICAÇÃO INTERACTIVA (DESAFIO) – PONG (UMA TABELA)

FLUXOGRAMA E PSEUDO-CÓDIGO DA APLICAÇÃO;

Desenho dos objectos por dados e localizações
(mais fácil para actualizar posições e redesenhar bem como testar colisões)

1. Declaração de variáveis globais
    a. Incremento de direcção /localização da bola
    b. Orientação/direcção (positiva ou negativa)
    c. Raio da bola;
    d. Direcção horizontal
    e. Paddle Height e Width
    f. Margem de segurança para saída desenho do paddle e teste colisão com a bola
    g. Implementação de pausa (boolean)

2. Inicialização
3. Método Principal
    a. Refrescar o ecrã (background)
    b. Inicia o movimento da bola na vertical
        i. Posição da bola += incremento
        ii. Ver se a bola cai do ecra (passa para além da margem) e recoloca-a de novo no topo do ecrã aleatoriamente
    c. Constrain movimento do paddle à largura do ecrã
    d. Teste de colisão da bola com o paddle (float)
    e. Verificação se quando a bola está no limite, está dentro do paddle numa só expressão
        i. Verificação da posição do paddle em relação à bola
        ii. alteração da orientação horizontal

    f. inverte a direcção se bater no tecto < 0
    g. Inverte a dir_x cada vez que toca nas paredes < 0 || > width
    h. Desenha a bola (ellispe, locx, locy e size)
    i. Desenha a paddle (rect, locx, locy, size);

j. desliga o cursor

4. Métodos secundários
    a. Simple pause (boolean)

PRIMEIRA APLICAÇÃO INTERACTIVA – PONG

PSEUDO-CÓDIGO

```
// Learning File 18
// Pedro Amado 2005.12.15

// Simple Pong v0.1 Alpha

// Collision e Constrain e Random
// ------------------------------

// desenho dos objectos por dados e localizações
// (mais fácil para actualizar posições e redesenhar bem como testar
colisões)

// Declaração de variáveis globais
// incremento de direcção /localização da bola
// orientação/direcção (positiva ou negativa)
// raio da bola;
// direcção horizontal

// Paddle Height e Width

// Margem de segurança para saída desenho do paddle e teste colisão com a
bola

// implementação de pausa (boolean)

// Inicialização
void setup() {

}

void draw() {
  // refrescar o ecrã (background)

  // Inicia o movimento da bola na vertical
  // posição da bola += incremento

  //ver se a bola cai do ecra (passa para além da margem)
  // e recoloca-a de novo no topo do ecrã aleatoriamente

  // Constrain movimento do paddle à largura do ecrã

  // Teste de colisão da bola com o paddle (float)
  // limit = tamanho do ecra menos a margem de segurança onde se encontra o
paddle)
  // limit = zona de testes

  // verificação se quando a bola está no limit está dentro do paddle numa
só expressão
    // verificação da posição do paddle em relação à bola
    // alteração da orientação horizontal
```

```
      //pmouseX = last mouse loc

  // inverte a direcção se bater no tecto < 0

  // Inverte a dir_x cada vez que toca nas paredes < 0 || > width

  // Desenha a bola (ellispe, locx, locy e size)

  // Desenha a paddle (rect, locx, locy, size);

  // desliga o cursor
}


void mousePressed() {
  // Simple pause (boolean)
}
```

CÓDIGO TOTAL

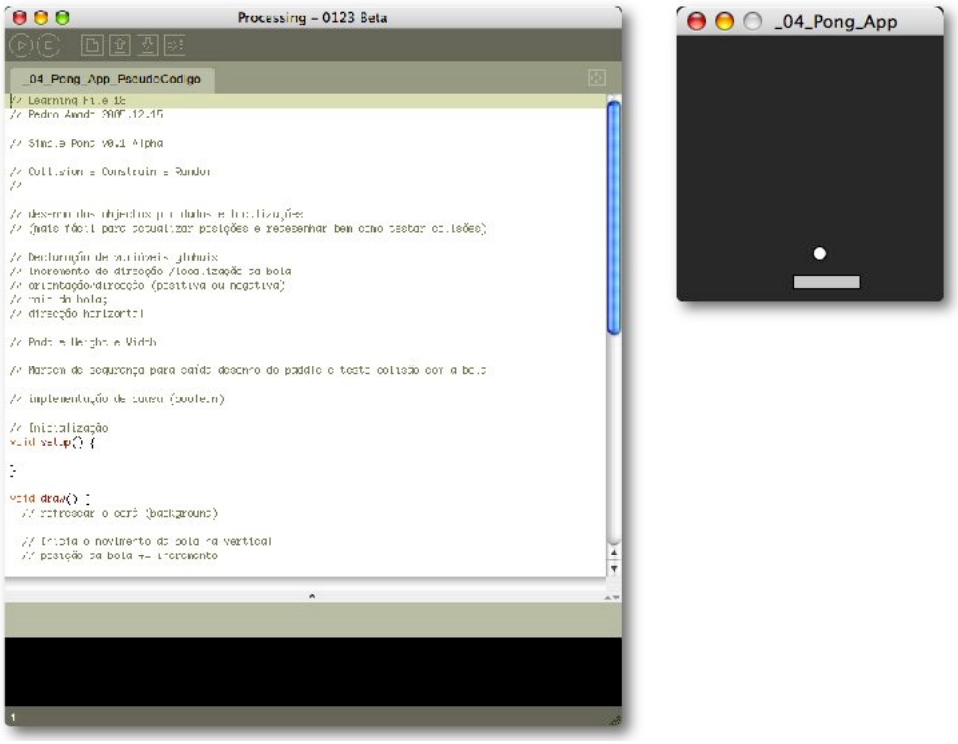


FIGURA 72 - PONG: SKETCH_018_CONSTRAIN_COLLISION_RANDOM_PONG_APP.PDE

```
// Learning File 18
// Pedro Amado 2005.12.15

// Simple Pong v0.1 Alpha
```

```
// Collision e Constrain e Random
// ------------------------------


// Declaração de variáveis globais
float bola_x, bola_y; // incremento de direcção
float bola_dir = 3;
float bola_size = 5; //raio da bola;
float dir = 0; //direcção

int paddle_w = 50; // Paddle Height e Width
int paddle_h = 10;

int dist_wall = 15;

void setup() {
  size(200, 200);
  background(0);
  fill(255);
  //noLoop();
  framerate(60);
  smooth();
  rectMode(CENTER);
  ellipseMode(CENTER_RADIUS);
  bola_x = width/2;
  bola_y = 1;

}

void draw() {
  background(40);

  // Inicia o movimento da bola na vertical
  bola_y += bola_dir*2;
  bola_x += dir;

  //ve se a bola cai do ecr„
  if (bola_y > height+bola_size) {
    bola_y = 1;
    bola_x = random(width);
    dir = random(-1.0,1.0);
  }

  /*
   random() Generates random numbers.
   Each time the random() function is called, it returns
   an unexpected value within the specified range. If one
   parameter is passed to the function it will return a
   float between zero and the value of the parameter.
   The function call random(5) returns values between 0 and 5.
   If two parameters are passed, it will return a float with a
   value between the the parameters. The function call
   random(-5, 10.2) returns values between -5 and 10.2.
   To convert a floating-point random number to an integer,
   use the int() function.
   */

  // Constrain movimento do paddle
  int paddle_x = constrain(mouseX, paddle_w/2, width-paddle_w/2);

  // Teste de colisão da bola com o paddle
  float limit = height - dist_wall - paddle_h - bola_size;
```

```
 //println(limit);

if (bola_y >= limit && bola_y <= limit+bola_dir*2 && bola_x > paddle_x-
paddle_w/2 && bola_x < paddle_x + paddle_w/2) {
    bola_dir *= -1;
    if (mouseX != pmouseX) {
      dir = (mouseX -pmouseX)/2.0;
      if (dir > 5) {
        dir = 5;
      }
      if (dir < -5) {
        dir = -5;
      }
    }
  }

  // inverte a direcção se bater no paddle ou no tecto
  if (bola_y < bola_size && bola_dir < 1) {
  //println(bola_dir);
    bola_dir *=-1;
  }

  // Inverte a dir_x cada vez que toca nas paredes
  if (bola_x > width - bola_size) {
    dir *= -1;
  }
  if (bola_x < bola_size) {
    dir *= -1;
  }

  // Desnha a bola
  fill(255);
  smooth();
  ellipse(bola_x, bola_y, bola_size, bola_size);

  // Desenha a paddle
  fill(200);
  noSmooth();
  rect(paddle_x, height - dist_wall, paddle_w, paddle_h);

  noCursor();
}


void mousePressed() {

}
```

PRÁTICA II (EXPOSIÇÃO TEÓRICO-PRÁTICA)

Neste capítulo vamos desenvolver uma aplicação de desenho (dinamico) a partir de dados estáticos de uma matriz – Draw APP. De seguida vamos converter essa mesma aplicação numa outra que exporte esses gráficos para um ficheiro utilizável posteriormente – PDF. Assim que a aplicação cumprir a sua função de desenho e exportação em PDF, vamos aumentar o número de objectos/desenhos a efectuar e tornar a aplicação (ligeiramente) interactiva com o ujtilizador. Pretende-se com isto introduzir os elementos mais avançados do processing – Sintaxe UI com o utilizador, Primitivas, Invocar Métodos (personalizados), Estrutura Condicional alternativa, Matrizes e utilização de Bibliotecas de expansão.

Recapitulação e lista integral de funções, estrutura e sintaxe do Processing:
http://processing.org/reference/index.html
http://processing.org/learning/index.html
http://processing.org/faq/index.html

SINTAXE BÁSICA DO PROCESSING

SINTAXE UI

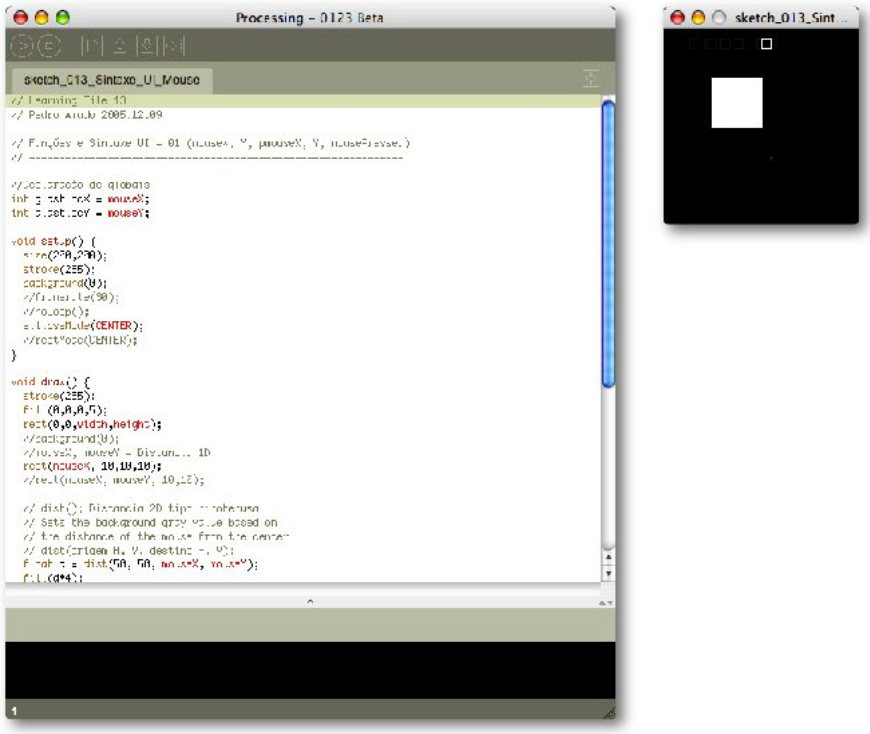


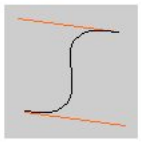
FIGURA 73 - SINTAXE UI (MOUSE): SKETCH_013_SINTAXE_UI_MOUSE.PDE

```
// Learning File 19
// Pedro Amado 2005.12.16

// Sintaxe UI Mouse UI 2
// ----------------------a

// Declaração de variáveis globais
float center_x, center_y;
boolean hover, drag;
int rect_size;
float dif_center_x, dif_center_y;

void setup() {
  size(200, 200);
  background(0);
  fill(255);
  //noLoop();
  framerate(30);

  center_x = width/2;
  center_y = height/2;
  rectMode(CENTER);
  rect_size = 25;
}

void draw() {
  background(0);
  noStroke();
```

```
  fill(150);

  // Teste para verificar o rollover do cursor em cima da caixa
  if (mouseX > center_x - rect_size && mouseX < center_x + rect_size
    && mouseY > center_y - rect_size && mouseY < center_y + rect_size) {
    hover = true;
    stroke(255);
    if (!drag) {
      fill(150);
    }
    else {
      hover = false;
      fill(255);
    }
  }

  // Desenha a caixa
  rect(center_x, center_y, rect_size, rect_size);
}

void mousePressed() {
  if (hover) {
    drag = true;
  }
  else {
    drag = false;
  }
  // Corrigir o clique do rato quando È diferente do centro do desenho a
mover
  // Não vale a pena preocupar com os valores negativos pois vão ser
corrigidos quando for para mover
  dif_center_x = mouseX-center_x;
  dif_center_y = mouseY-center_y;
}

void mouseDragged() {
  if (drag) {
    center_x = mouseX-dif_center_x;
    center_y = mouseY-dif_center_y;
  }
}

void mouseReleased() {
  drag = false;
}
```

FUNÇÃO BEZIER

bezier()



```
stroke(255, 102, 0);
line(85, 20, 10, 10);
line(90, 90, 15, 80);
stroke(0, 0, 0);
bezier(85, 20, 10, 10, 90, 90, 15, 80);
```

```
stroke(255, 102, 0);
line(30, 20, 80, 5);
line(80, 75, 30, 75);
stroke(0, 0, 0);
bezier(30, 20,  80, 5,  80, 75,  30, 75);
```

```
// Sintaxe
// bezier(x1, y1, x2 (BCP), y2(BCP), x3(BCP), y3(BCP), x4, y4);
stroke(150);
bezier(100, 200, 100, 200, mouseX, mouseY, mouseX, mouseY);
```

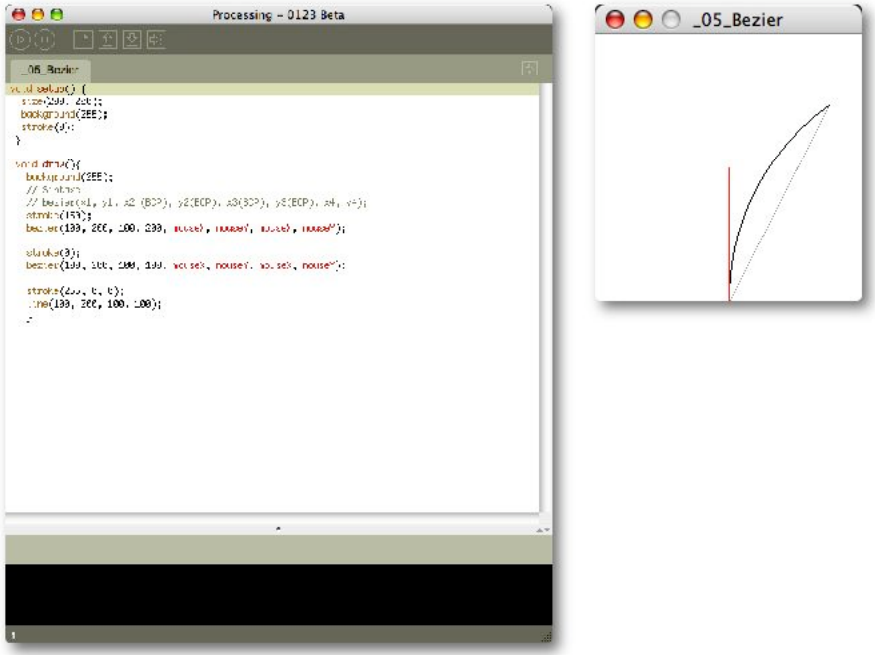


FIGURA 74 - _05_BEZIER.PDE

INVOCAR MÉTODOS

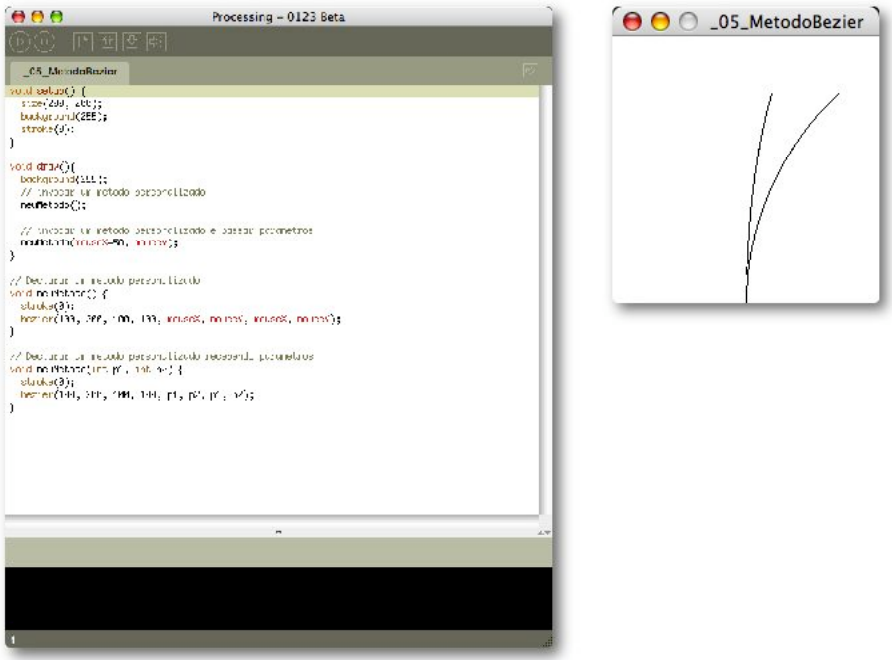


FIGURA 75 - _05_METODOBEZIER.PDE

```
// invocar um metodo personalizado
 meuMetodo();
```

```
// Declarar um metodo personalizado
void meuMetodo() {
  stroke(0);
  bezier(100, 200, 100, 100, mouseX, mouseY, mouseX, mouseY);
}
```

INVOCAR MÉTODOS PERSONALIZADOS E PASSAR PARÂMETROS

Um paremetro é uma instrução detalhada, isto é, uma personalização de uma instrução com maior nível de rigor ou detalhe. Voltando à metáfora do programa enquanto receita, podemos considerar que um método será um procedimento a cumprir do género “bater ovos”. Assim, podíamos dizer na receita, ou no método “bater ovos” simplesmente dizendo-o, ou então dizer “bater ovos 50 vezes”. Será mais ou menos equivalente a dizer desenha uma curva [predefinida, na localização fixa], ou desenha uma curva [predefinida, numa localização X], ou ainda melhor desenha uma curva [com arco AB, numa localização X]. Cada vez que invocamos o método desenhamos curvas diferentes. É quase como as primitivas, só que mais complexo.

```
// invocar um metodo personalizado e passar parametros
meuMetodo(mouseX+50, mouseY);


// Declarar um metodo personalizado recebendo parametros
void meuMetodo(int p1, int p2) {
  stroke(0);
  bezier(100, 200, 100, 100, p1, p2, p1, p2);
}
```

ESTRUTURA SWITCH (CONDIÇÕES)

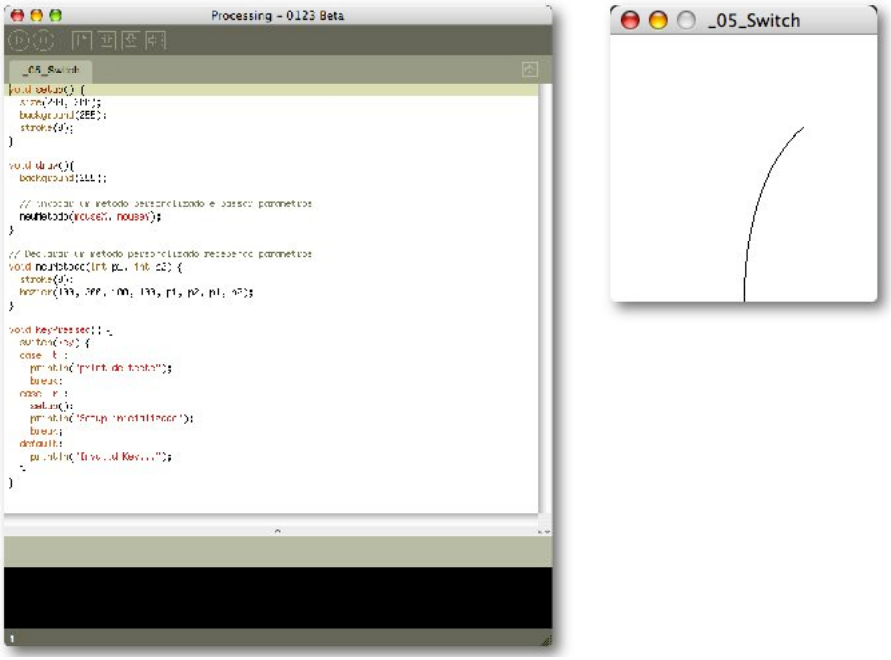


FIGURA 76 - _05_SWITCH.PDE

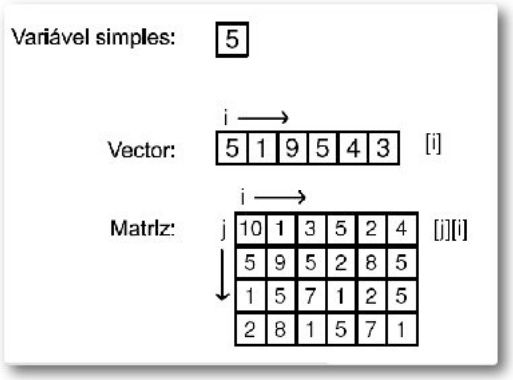
```
switch(expression)
{
  case label:
     statements
  case label:        // Optional
     statements      // "
  default:           // "
     statements      // "
```

}

```
void keyPressed() {
  switch(key) {
  case 't':
    println("print de teste");
    break;
  case 'r':
    setup();
    println("Setup inicializado");
    break;
  default:
    println("Invalid Key...");
  }
}
```

MATRIZES

São basicamente vectores de vetores... um vertical que contem uma data deles horizontais.




```
// Declarar e inicializar o nosso vector
int[] vector = {1, 7, 23, 444, 8};

void mousePressed() {
 println(vector[2]);
```

Matrizes… vector (vertical) de vectores (horizontais)

```
// Declarar a nossa matriz de dados
int[][] matriz;
```

```
// Inicializar linhas e colunas a zeros
matriz = new int[3][5];

void mousePressed() {
 println(vector[2]);
 print(matriz[0]);
```

Testar o programa…

```
// Declarar e inicializar o nosso vector
int[] vector = {1, 7, 23, 444, 8};
// Declarar a nossa matriz de dados
int[][] matriz;

void setup() {
  size(200, 200);
  background(255);
  stroke(0);
  // Inicializar linhas e colunas a zeros
  matriz = new int[3][5];
  matriz[1] = vector;
}
void draw(){
}
void mousePressed() {
 println(vector[2]);
 print(matriz[0]);
 println();
 print(matriz[1]);
}
```

A PARTIR DAQUI É PURA MANIPULAÇÃO DE CONCEITOS…

O QUE É QUE DEFINE UMA FLOR (PLANTA)?

1. 1 flor = 1 caule + 1 flor (bolbo);
    a. 1 caule = 2 pontos (base e altura) e uma espessura;
    b. 1 flor = 1 ponto (base), 1 flor (bolbo) e 1 pétala (ou mais);
        i. 1 pétala = 1 ponto, diâmetro e largura;

AS FLORES SÃO SENSÍVEIS AO AMBIENTE:

Vento como modificador ambiental/temporal.
É um movimento cíclico (que pode ser traduzido por uma função de seno sobre um incremento provocando um movimento oscilatório só na horizontal) - Interfere com a posição de cada flor… dada a resistência (espessura) do caule.

COMO É QUE ISTO SE PARAMETRIZA EM CÓDIGO/GRAFICAMENTE?

```
//importar bibliotecas

// Declaracao de Variaveis
// Tamanho dinamico de ecra
// Numero total de flores a desenhar
// Vento - float angulo em radianos - para fazer oscilacao funcao de sen
//     Incremento ao vento  - float PI
//     Total de incremento em pospara fazer oscilar - soma
// Declaracao da lista de flores a desenhar - Matriz
// Boolean para saber se imprimimos a imagem ou nao
// Declaracao de Objectos - Fonte
```

INICIALIZAÇÃO DO PROGRAMA:

```
//Inicializacao
void setup() {
  //Inicializacao das constantes do tamanho da app
  size(SW, SH);
  //size(SW, SH, OPENGL); // para desenhar muitos objectos
  //Inicializacao de flores Matriz
  //    Matriz
  //    VECTOR MANUAL da matriz - indice 0 - para teste da
  //    {locx, locy, altura, espessura, diametro petala, la
  //       locx
  //       locy
  //       altura
  //       espessura
  //       diametro da petala
  //       largura da petala
  //    inicializacao automatica - percorrer a matriz - for
  // inicializacao do vento - exemplo de circulo em metodo
  //    PI radianos = 180 graus incremento a dar ao valor v
  // Load Font Objects
  // boolean de impressão - evitar uso com muitos objectos
}
```

CICLOS DE DESENHO:

```
//Draw de objectos void draw
void draw() {
  //Verificar se vai ser um ciclo de print e iniciar a impr
  // Limpar
  // Nome da APP ou data ou assim... mensagens
  // invocar o metodo de flores personalizado em ciclo a pe
  // invocar um metodo de creditos personalizados
  // adicionar o incremento ao vento
  // calcular a oscilacao atraves do seno
  // se for um ciclo de impressao fechar a impressao
}
```

PREPARAÇÃO DO MÉTODO QUE VAI DESENHAR A FLOR:

```
// declarar o metodo personalizado de flores
void drawflower(int p0, int p1, int p2, int p3, int p4, int
  //desenho das petalas
  //localizacao da flor
  //verificacao da proximidade do rato e afastar... fica pa
  //desenho do Caule do chao para a flor
  //{locx, locy, altura, espessura, dpetalas, largpetala}
  // desenho da flor
  // desenho da petala
  //desenho dos BCPs  da base do caule
}

//Pause e restart
void mousePressed() {
  // restart
}

void keyPressed() {
  // print
}


// Tamanho dinamico de ecra
int SW, SH;

//numero total de flores a desenhar
int NFLORES;
```

No inicio vai ser igual a 1 para podermos testar o modelo/parâmetros de desenho e do próprio programa.

```
// Vento angulo em radianos para fazer oscilacao funcao de seno
float vento, incvento, inc;


// Declaracao da lista de flores a desenhar - Matrizes
int[][] flores;


//Boolean para saber se imprimimos a imagem ou nao
boolean p;

//Inicializacao
void setup() {
  SW = screen.width-100;
  SH = screen.height-300;
  size(SW, SH);
  background(255);
```

Desenhar só uma flor para testar o modelo de desenho.

```
 //Inicialização das flores
 NFLORES = 1;
```

Declaração da primeira flor na matriz, para testar a própria matriz, e o método de desenho:

```
//Inicializacao de flores MANUAL para teste da primeira
//{locx, locy, altura, espessura, diametro petala, largura petala}
flores = new int[NFLORES][6];
flores[0][0] = SW/2+SW/4; //locx
flores[0][1] = SH; //locy
flores[0][2] = SH/2; // altura
flores[0][3] = 1; // espessura
flores[0][4] = 10; // diametro da petala
flores[0][5] = 3; // largura da petala


//Draw de objectos void draw
void draw() {

// Limpar
background(255);

//{locx, locy, altura, espessura, diametro petala, largura petala}
drawflower(flores[0][0], flores[0][1], flores[0][2], flores[0][3], flores[0][4], flores[0][5]);
```

Personalizar/definir o método de desenho/objectos de desenho de flores ao máximo com estes 5 parametros…

```
// declarar o metodo personalizado de flores
//{locx, locy, altura, espessura, dpetalas, largpetala}
void drawflower(int p0, int p1, int p2, int p3, int p4, int p5) {
  //desenho das petalas
  //localizacao da flor
  int intflocx = int(p0+inc/p3); // loc horizontal + vento mais espesssura/resistencia do caule
  int intflocy = p2;

  //desenho do Caule do chao para a flor
  noFill();
  stroke(0, 100, 0, 100);
  strokeWeight(p3);
  //{locx, locy, altura, espessura, dpetalas, largpetala}
  bezier(p0, p1, p0, p2, intflocx, intflocy, intflocx, intflocy);

  // desenho da flor
  fill(230, 180, 0);
  noStroke();
  ellipse(intflocx, intflocy, 10, 10);

  // desenho da petala
  noFill();
  stroke(230, 180, 0, 100);
  strokeWeight(p5);
  ellipse(intflocx, intflocy, p4, p4);

  //desenho dos BCPs  da base do caule
  noFill();
  stroke(0, 200, 0);
  strokeWeight(1);
  line(p0, p1, p0, p2+p2/2);
  rectMode(CENTER);
  rect(p0, p2+p2/2, 5, 5);

}
```

A PARTIR DAQUI É PURA MANIPULAÇÃO DE DADOS…

VENTO - COMO MANIPULAR:

Setup()

```
// inicializacao do vento - exemplo de circulo em metodo europeu
vento = 0.0;
incvento = PI/100; //PI radianos = 180 graus incremento a dar ao valor vento para "circular"
```

Draw()

```
// adicionar o incremento ao vento
vento=(vento+incvento)%TWO_PI;
// calcular a oscilacao atraves do seno
inc = sin(vento)*100; // incremento da oscilacao do vento
```

Já deve dar uma flor a oscilar…

AGORA VAMOS AUTOMATIZAR O PROCESSO PARA DAR MUITAS!

Setup() – Aumentar o número de flores a desenhar

```
//Inicialização das flores
NFLORES = 10;
```

Draw() – criar um ciclo de desenho a partir do acesso à matriz em iteração.

```
// invocar o metodo de flores personalizado em ciclo a percorrer a matriz
for (int n = 0; n<flores.length; n++) {
  //{locx, locy, altura, espessura, diametro petala, largura petala}
  //drawflower(flores[0][0], flores[0][1], flores[0][2], flores[0][3], flores[0][4], flores[0][5]);
drawflower(flores[n][0], flores[n][1], flores[n][2], flores[n][3], flores[n][4], flores[n][5]);
}
```

SWITCH PRINT

Pré Setup()

```
//Boolean para saber se imprimimos a imagem ou nao
boolean p;
```

Testar o teclado… (sintaxe UI) – Callback keyPressed()

```
void keyPressed() {
  switch(key) {
  case 't':
    println("print de teste");
    break;
  case 'p':
    p = true;
    println("start printing pdf");
    break;
  case 'r':
    setup();
    break;
  }
}
```

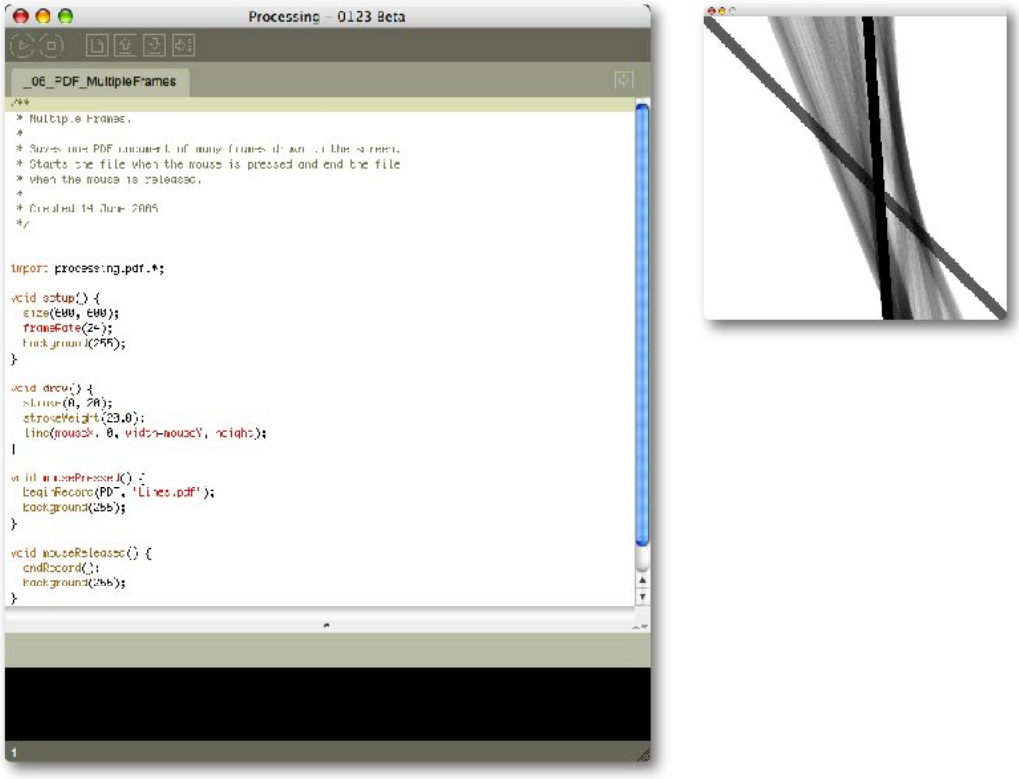


FIGURA 77 - _06_PDF_MULTIPLEFRAMES

Iniciar a impressão (um ciclo de cada vez) através da biblioteca de PDF

```
import processing.pdf.*;
```

```
void mousePressed() {
  beginRecord(PDF, "Lines.pdf");
  background(255);
}

void mouseReleased() {
  endRecord();
  background(255);
}
```

No inicio do Draw()

```
//Draw de objectos void draw
void draw() {
  //Verificar se vai ser um ciclo de print e iniciar a impressao
  if (p) {
    beginRecord(PDF, "printscreen####.pdf");
  }
```

Antes do final do draw()

```
 // se for um ciclo de impressao fechar a impressao
 if (p) {
   endRecord();
   p = false;
   println("----------------");
   println("close pdf");
 }
```

EXPERIMENTAR AGORA COM 1000+ FLORES…

A solução está no OpenGL da placa gráfica

pré Setup()

```
import processing.opengl.*;
```

Setup()

```
size(SW, SH, OPENGL); // para desenhar muitos objectos
//smooth(); // ligar com o modo OPENGL
```

CÓDIGO COMPLETO (DRAW APP – OPENGL + PDF EXPORT)

```
import processing.pdf.*;

import processing.opengl.*;

/*
Pedro Amado
 Learning File XX
 2006-12-05

 Drawing Aplication
 PDF Export APP
 */

// Declaracao de Variaveis
// Declaracao e inicializacao de variaveis do setup

// Tamanho dinamico de ecra
int SW, SH;

//numero total de flores a desenhar
int NFLORES;

// Vento angulo em radianos para fazer oscilacao funcao de seno e incremento
para fazer oscilar
float vento, incvento, inc;

// Declaracao da lista de flores a desenhar - Matrizes
int[][] flores;

//Boolean para saber se imprimimos a imagem ou nao
boolean p;

//Declaracao de Objectos
PFont f;

//Inicializacao
void setup() {
```

```
  SW = screen.width-100;
  SH = screen.height-300;
  //size(SW, SH);
  //background(255);

  size(SW, SH, OPENGL); // para desenhar muitos objectos
  background(255);
  //smooth(); // ligar com o modo OPENGL

  //Inicialização das flores
  //NFLORES = 500;

  //NFLORES = int (random(1,250));
  NFLORES = int (random(1,1250)); // OpenGL
  println("A desenhar "+NFLORES+" flores...");
  //Inicializacao de flores MANUAL para teste da primeira
  //{locx, locy, altura, espessura, diametro petala, largura petala}
  flores = new int[NFLORES][6];
  flores[0][0] = SW/2+SW/4; //locx
  flores[0][1] = SH; //locy
  flores[0][2] = SH/2; // altura
  flores[0][3] = 1; // espessura
  flores[0][4] = 10; // diametro da petala
  flores[0][5] = 3; // largura da petala

  //inicializacao automatica - percorrer a matriz
  for (int n=1; n < NFLORES; n++) {
    flores[n][0] = SW/2+int(random(-SW/4, SW/4));
    flores[n][1] = SH;
    flores[n][2] = SH/2+int(random(-100, 100));
    flores[n][3] = int(random(10)+1);
    flores[n][4] = int(random(50)+1);
    flores[n][5] = int(random(10)+1);
  }

  // inicializacao do vento - exemplo de circulo em metodo europeu
  vento = 0.0;
  incvento = PI/100; //PI radianos = 180 graus incremento a dar ao valor
vento para "circular"

  // Load Font Objects
  f = loadFont("Myriad-Web-24.vlw");
  textFont(f, 14);

  // boolean de impressão - evitar uso com muitos objectos e OpenGL
  p = false;
}

//Draw de objectos void draw
void draw() {
  //Verificar se vai ser um ciclo de print e iniciar a impressao
  if (p) {
    beginRecord(PDF, "printscreen####.pdf");
  }
  // Limpar
  background(255);
  fill(0, 10);
  //ellipse(mouseX, mouseY, 100, 100); // para a verificação de distâncias

  // Nome da APP ou data ou assim... mensagens
  fill(230, 180, 0);
  //textFont(f, 24);
```

```
  //text("Draw APP 1.0", 50, 140);

  // invocar o metodo de flores personalizado em ciclo a percorrer a matriz
  for (int n = 0; n<flores.length; n++) {
    //{locx, locy, altura, espessura, diametro petala, largura petala}
    //drawflower(flores[0][0], flores[0][1], flores[0][2], flores[0][3],
flores[0][4], flores[0][5]);
    drawflower(flores[n][0], flores[n][1], flores[n][2], flores[n][3],
flores[n][4], flores[n][5]);
  }

  // adicionar o incremento ao vento
  vento=(vento+incvento)%TWO_PI;
  // calcular a oscilacao atraves do seno
  inc = sin(vento)*100; // incremento da oscilacao do vento

  // se for um ciclo de impressao fechar a impressao
  if (p) {
    endRecord();
    p = false;
    println("----------------");
    println("close pdf");
  }
}

// declarar o metodo personalizado de flores
//{locx, locy, altura, espessura, dpetalas, largpetala}
void drawflower(int p0, int p1, int p2, int p3, int p4, int p5) {
  //desenho das petalas
  //localizacao da flor
  int intflocx = int(p0+inc/p3); // loc horizontal + vento mais
espesssura/resistencia do caule
  int intflocy = p2;

// verificar a posicao do rato e alterar a localizacao da flor
  float c = dist(mouseX, mouseY, intflocx, intflocy);
  if (c<100) {
    if (intflocx<mouseX) {
      intflocx = intflocx+(intflocx-mouseX)/2;
    }
    else if (intflocx>mouseX) {
      intflocx = intflocx-(mouseX-intflocx)/2;
    }
  }
  //desenho do Caule do chao para a flor
  noFill();
  stroke(0, 100, 0, 100);
  strokeWeight(p3);
  //{locx, locy, altura, espessura, dpetalas, largpetala}
  bezier(p0, p1, p0, p2, intflocx, intflocy, intflocx, intflocy);

  // desenho da flor
  fill(230, 180, 0);
  noStroke();
  ellipse(intflocx, intflocy, 10, 10);

  // desenho da petala
  noFill();
  stroke(230, 180, 0, 100);
  strokeWeight(p5);
  ellipse(intflocx, intflocy, p4, p4);
```

```
  //desenho dos BCPs  da base do caule
  noFill();
  stroke(0, 200, 0);
  strokeWeight(1);
  line(p0, p1, p0, p2+p2/2);
  rectMode(CENTER);
  rect(p0, p2+p2/2, 5, 5);

}
//Print Cycle

//Pause e restart
void mousePressed() {
  println(flores.length);
  println(flores[0].length);
  // restart
  setup();
}

void keyPressed() {
  switch(key) {
  case 't':
    println("print de teste");
    break;
  case 'p':
    p = true;
    println("start printing pdf");
    break;
  case 'r':
    setup();
    break;
  }
}
```

APLICAÇÃO INTERACTIVA (DESAFIO) - JOGO DA VIDA (SIMPLES)

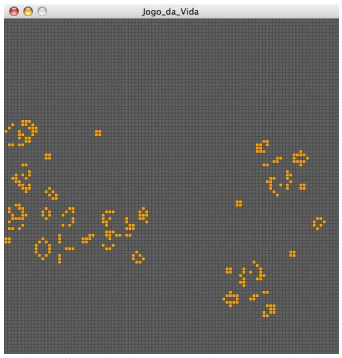


FIGURA 78 - JOGO DA VIDA

John Conway
http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

Autómatos Celulares:
A ideia é cada célula (unidade de desenho) alterar ou manter o seu estado consoante o estado dos seus vizinhos. Assim vão gerando padrões.

Objectivos:
O jogo tem que criar um numero dinâmico de células que com mais ou menos resistência ao tempo de vida e aos vizinhos – desenhar um padrão que morre rápido e outro tipo fungo, que alastre muito.

Regras
http://pt.wikipedia.org/wiki/Jogo_da_vida
As regras definidas são aplicadas a cada nova "geração". Assim, a partir de uma imagem num tabuleiro bi-dimensional definida pelo jogador, percebem-se mudanças muitas vezes inesperadas e belas a cada nova geração, variando de padrões fixos a caóticos.

Conway escolheu suas regras cuidadosamente, após um longo período de testes, para satisfazer três critérios:

1. Não deve haver nenhuma imagem inicial para a qual haja uma prova de que a população pode crescer sem limite.
2. Deve haver imagens iniciais que aparentemente crescam sem limite.

3. Deve haver imagens iniciais simples que cresçam e mudem por um período de tempo considerável antes de chegar a um fim das possíveis formas:
    a. Sumindo completamente (por superpopulação ou por ficarem muito distantes)
    b. Estacionando em uma configuração estável que se mantem imutável para sempre, ou entrando em uma fase de oscilação na qual são repetidos ciclos infinitos de dois ou mais períodos.

Por outras palavras, as regras deviam tornar o comportamento das populações ao mesmo tempo interessante e imprevisível.

Na prática regras são simples e elegantes:

1. Qualquer célula viva com menos de dois vizinhos vivos morre de solidão.
2. Qualquer célula viva com mais de três vizinhos vivos morre de superpopulação.
3. Qualquer célula com exactamente três vizinhos vivos "nasce".
4. Qualquer célula com dois vizinhos vivos continua no mesmo estado.

É importante entender que todos os nascimentos e mortes ocorrem simultaneamente. Juntos eles constituem uma geração ou, como podemos chamá-los, um "instante" na história da vida completa da configuração inicial.

```
/*

Processing Learning File
Pedro Amado
2006.07.07

Jogo da Vida
Aplicação desenvolvida por Pedro e Nuno Amado

*/

int BOARD_SIZE = 100;
int CELL_SIZE  = 5;

boolean Board[][];

void setup() {
  int x,y;

  // Inicialização do palco
  size(BOARD_SIZE*CELL_SIZE,BOARD_SIZE*CELL_SIZE);
  //framerate(10);

  // Inicialização do tabuleiro (da bida)
  Board=new boolean[BOARD_SIZE][BOARD_SIZE];
  for(y=0; y<BOARD_SIZE; y++) {
    for(x=0; x<BOARD_SIZE; x++) {
      Board[y][x]=false;
```

```
    }
  }

  // Configuraçao da primal soup
  /*for (x=0;x<3;x+=1) {
    Board[2+x][3+x]=true;
    Board[3+x][2+x]=true;
    Board[3+x][3+x]=true;
    Board[3+x][4+x]=true;
    Board[4+x][3+x]=true;
  }*/
}

void draw() {
  int x,y;

  background(80);
  stroke(60);

  // Desenha a grelha
  for(y=0; y<height; y+=CELL_SIZE) {
    line(0,y,width,y);
  }
  for(x=0;x<width;x+=CELL_SIZE) {
    line(x,0,x,height);
  }

  //User Input
  if (mousePressed) {
    //framerate(999);
    //println("mouse");
    int coluna = mouseX/CELL_SIZE;
    if (coluna <=0) {
      coluna = 0;
    }
    else if (coluna >=BOARD_SIZE){
      coluna = BOARD_SIZE-1;
    }
    int linha = mouseY/CELL_SIZE;
    if (linha <=0) {
      linha = 0;
    }
    else if (linha >=BOARD_SIZE){
      linha = BOARD_SIZE-1;
    }
    //println(linha+", "+coluna);
    if (mouseButton == LEFT) {
      Board[linha][coluna] = true;
    }
    else if (mouseButton == RIGHT){
      Board[linha][coluna] = false;
    }
  }

  // Desenha as celulazinhas...
  fill(250,150,0);
  noStroke();
  for(y=0; y<BOARD_SIZE; y++) {
    for(x=0; x<BOARD_SIZE; x++) {
      if (Board[y][x]) {
        rect(x*CELL_SIZE+1,y*CELL_SIZE+1,CELL_SIZE-1,CELL_SIZE-1);
      }
```

```
    }
  }

  // Evolui a bida
  if (!mousePressed) {
    evolui();
  }
}

void evolui() {
  int x,y,n;
  boolean future[][];

  future=new boolean[BOARD_SIZE][BOARD_SIZE];
  for(y=0; y<BOARD_SIZE; y++) {
    for(x=0; x<BOARD_SIZE; x++) {

      n=contaVizinhos(x,y);
      if (n<2)
        future[y][x]=false;
      else if (n==2)
        future[y][x]=Board[y][x];
      else if (n==3)
        future[y][x]=true;
      else
        future[y][x]=false;
    }
  }

  Board=future;

}
int contaVizinhos(int x, int y) {
  int vx,vy,n;

  n=0;

  vx=x-1;
  vy=y-1;
  if (vx>=0&&vy>=0&&Board[vy][vx]) n++;

  vx=x;
  vy=y-1;
  if (vy>=0&&Board[vy][vx]) n++;

  vx=x+1;
  vy=y-1;
  if (vx<BOARD_SIZE&&vy>=0&&Board[vy][vx]) n++;

  vx=x+1;
  vy=y;
  if (vx<BOARD_SIZE&&Board[vy][vx]) n++;

  vx=x+1;
  vy=y+1;
  if (vx<BOARD_SIZE&&vy<BOARD_SIZE&&Board[vy][vx]) n++;

  vx=x;
  vy=y+1;
  if (vy<BOARD_SIZE&&Board[vy][vx]) n++;

  vx=x-1;
```

```
  vy=y+1;
  if (vx>=0&&vy<BOARD_SIZE&&Board[vy][vx]) n++;

  vx=x-1;
  vy=y;
  if (vx>=0&&Board[vy][vx]) n++;

  return n;
}

void mousePressed() {
  //noLoop();
  //redraw();
  //framerate(999);
}

void mouseReleased() {
  //smooth();
  //framerate(10);
  //loop();
}
```

PRÁTICA III

PROGRAMAR UMA PEÇA GRÁFICA.

O desafio consiste em desenhar uma aplicação para expor no Museu Virtual da FBAUP, que sirva de exemplo do potencial do Processing para desenhar gráficos.

OBJECTIVOS

- Criar padrões ou formas ilustradas;
- Capacidade para exportar gráficos em PDF;
- Interactividade com o utilizador.

INSTRUÇÕES PARA USO DO MODULO DE PDF

http://processing.org/download/revisions.txt

```
ABOUT REV 0100 - 13 January 2006

This is a pre-release of the PDF generating code. There will be a couple
other quick releases to follow in the coming days, and this shouldn't be
```

```
considered a "stable" release since it's not been tested heavily, but we
wanted to post it for the people who'd like to try it out.

To use it the pdf library, select Sketch -> Import Library -> pdf.
And then choose one of the following methods:

+ to render everything out to a file:

  void setup() {
    size(400, 400, PDF, "filename.pdf");
  }

  void draw() {
    // draw something good here
  }

  void mousePressed() {
    exit();  // important!
  }

  this will create a sketch that draws only to a file. successive frames
  will be drawn on top of one another. when you're finished, call exit(),
  for instance, inside mousePressed() or a keyPressed() event.
  this will still open a window of size 400x400 but not draw into it.
  future releases won't open the window, since it doesn't work properly
  with sizes larger than the screen size (as might often be the case when
  rendering to a file).

+ if you want to record just a single frame, use beginRecord() and
  endRecord() inside draw:

  void draw() {
    beginRecord(PDF, "frame-####.pdf");
    // your awesome drawing happens here.
    endRecord();  // saves the file
  }

  this can also be triggered by a mouse press. the #### will cause a
  four digit "frame" number to be inserted into the filename.

+ if you want to record the entire thing to a file at the same time as
  drawing to the screen (i.e. save out an interactive drawing), do this:

  void setup() {
    size(400, 400);
    // etc
    beginRecord(PDF, "everything.pdf");
  }

  void draw() {
    // do your worst
  }

  // this needn't be mousePressed, but you need to call this somewhere
  // when you're done, to ensure that the pdf file properly closes.
  void mousePressed() {
    endRecord();
  }

Caveats with PDF rendering:

+ anything bitmapped works very poorly. so it's no good for images.
```

```
+ use createFont() with a truetype font (some opentype also work) to
  make fonts work properly. any font that shows up in PFont.list()
  should work. if you don't use createFont(), they'll likely show up
  bitmappy and gross like images. the shape of the characters may be
  weird in some cases, there seems to be a java bug with cubic vs.
  quadric splines. looking into a workaround if necessary.

+ rendering from 3D isn't yet supported, but coming soon.

+ exit() is really important when using PDF with size()
```

IMPLEMENTAÇÃO DOS MÉTODOS PAUSERECORD() E RESUMERECORD()

Autor: marc_h

http://processing.org/discourse/yabb_beta/YaBB.cgi?board=Suggestions;action=display;num=1147209867

```
/*
 Forum do Processing

http://processing.org/discourse/yabb_beta/YaBB.cgi?board=Suggestions;action=
display;num=1147209867

 Autor
 mark_h

 pauseRecord() and resumeRecord()
« on: May 9th, 2006, 11:24pm »

Pedro Amado
2006.06.08

 */

 import processing.pdf.*;

void setup() {
  size (400, 400);
}

int fileno=1;

void draw() {

  pauseRecord(); // if we are recording stop!
  // Things not to appear in the pdf
  background(255);
  stroke((int)random(0,255));
  rect(width/2,height/2,mouseY,mouseX);

  resumeRecord();
  // This will all appear in the pdf
  stroke((int)random(0,255));
  ellipse (width/2, height/2, mouseX, mouseY);
}

void mousePressed() {

   beginRecord(PDF, "foo" + fileno++ + ".pdf");
   background(255); // set the intial state of the pdf
}

void mouseReleased() {

   endRecord(); // finish up this pdf
}

// Overriden methods from processing.core.PApplet
```

```
PGraphics container; // temporary storage for recorder

  public PGraphics beginRecord(String renderer, String filename) {
    filename = insertFrame(filename);
    PGraphics rec = createGraphics(width, height, renderer, filename);
    beginRecord(rec);
    return rec;
  }


  public void beginRecord(PGraphics recorder) {
    this.recorder = recorder;
    recorder.beginFrame();
  }


  public void endRecord() {
    if (recorder != null) {
 recorder.endFrame();
 recorder.dispose();
 recorder = null;
 container = null;
    }
  }

  public void pauseRecord() {
    if (recorder != null) {
 container = recorder;
 recorder = null;
    }
  }


  public void resumeRecord() {
    if (container != null) {
 beginRecord(container);
    }
  }
```

EXEMPLO DE UMA PEÇA GRÁFICA

Máquina de Desenho de Árvores

FIGURA 79 - MÁQUINA DE DESENHO

```
import processing.pdf.*;

/*
Pedro Amado
 2006.06.16
 ----------------------------------------
 Ficheiro para a Formação de 07 de Julho
 FBAUP
 ----------------------------------------
 Aplicação de desenho de gráficos
 Ecrã Full screen em formato A4
 Export Hi-Res PDF
 ----------------------------------------
 Desenhar o Layout no quadro! ;)
 */

//Declaração Variaveis
int SCREEN_H, SCREEN_W; //tamanho da janela da app
int MARGEM; //margem para "aparar" o desenho no ecrã
int w1, w2, w3, w4; //definição da área útil
int h1, h2, h3, h4;
color rgb_1, rgb_2, rgb_3; //cores a usar na comp

PFont tipo_1;

boolean mpress;

int click[] = new int[5]; //guardar os clicks - x1, y1, x2, y2, stroke

int linhas = 1000; //numero de objectos a desenhar
int rectas[][] = new int[linhas][4]; //lista de objectos (locs) em cena
int objects = 0; //verifica e conta os objectos desenhados (indice da lista)
float stroke_w;
```

```
float stroke_ws[] = new float[linhas]; //armazena a espessura dos traços

boolean trail = false;
boolean var_w = false;

boolean recordpdf = false;

// Definição da árvore
boolean arvore = false;

//Tronco = x1, x2, y1, y2, width, color ********************************
int tronco[] = new int[4];
float tronco_d; // Tamanho dist do tronco
float tronco_w;
color tronco_c;


//Ramos = nº de ramos (indice)  ****************************************
//nº de galhos max (iterações) para cada ramo
// Ramos = locs x1, x2, y1, y2, widths color
int nramos        = 10;                 //numero max de ramos
int r;                                  //indice
int ramos_g[]     = new int[nramos];    // numero de iterações(nós de galho)
por ramo
int ramos[][]     = new int[nramos][4]; // numero max de ramos por tronco 4
locs
float ramos_ang[] = new float[nramos];    // armazena o angulo de
crescimento de cada um
float ramos_w[]   = new float[nramos];  //
color ramos_c[]   = new color[nramos];  //

// Galhos
*****************************************************************
int ngalhos        = 5;                             // numero max galhos
int galhos_i;                                       // indice da matriz
int galhos[][]     = new int[nramos*ngalhos][4];    //matriz sequencial de
valores de posição
float galhos_ang[] = new float[nramos*ngalhos];     //armazena o ang de
crescimento de cada um
float galhos_w[][] = new float[nramos][ngalhos];
color galhos_c[][] = new color[nramos][ngalhos];

////////////////////////////////////////////////////////////////////////
//Atribuição e Setup
void setup() {
  MARGEM   = 50;
  SCREEN_H = screen.height-MARGEM; //definição dinâmica do tamanho para
poder mudar para versão web
  SCREEN_W = SCREEN_H*210/297-MARGEM; //regra de 3 simples para A4

  //Janela da App
  size(SCREEN_W, SCREEN_H);

  background(255);
  stroke(100);
  strokeWeight(1);

  creditos();

  //Marcas de corte line(x1, y1, x2, y2)
  w1 = MARGEM/2-5;
  w2 = MARGEM/2;
```

```
  w3 = SCREEN_W-MARGEM/2;
  w4 = SCREEN_W-MARGEM/2+5;

  h1 = MARGEM/2-5;
  h2 = MARGEM/2;
  h3 = SCREEN_H-MARGEM/2;
  h4 = SCREEN_H-MARGEM/2+5;

  marcascorte();

  rgb_1 = color(255, 0, 0);   //Red
rgb_2 = color(0,0,0);         //Green
rgb_3 = color(0, 0, 0, 50);   //Blue

  colorchooser();

  mpress = false;
}

////////////////////////////////////////////////////////////////////////////

//Metodo principal - Draw
void draw() {
  if (mpress) {

    if (!trail) {
      //trail
      noStroke();
      fill(255);
      rect(w1, h1, SCREEN_W-w1*2, SCREEN_H-h1*2);
    }

    stroke(rgb_1);

    tronco_d = dist(click[0], click[1], mouseX, mouseY);
    stroke_w = SCREEN_W/(tronco_d/10);

    if (var_w) {
      stroke_w = SCREEN_W/(tronco_d/20);
    }
    else if (!var_w) {
      stroke_w = 1;
    }

    if (stroke_w < 1) {
      stroke_w = 1;
    }
    else if (stroke_w > 300) {
      stroke_w = 300;
    }



    if (!trail) {
      mira(click[0], click[1]);
      strokeWeight(stroke_w);
      line(click[0], click[1], mouseX, mouseY);
    }
  }

  if (arvore) {
    //Tronco ************************************************************
```

```
    tronco();
    //RAMOS ************************************************************
    ramos();
    no(click[2], click[3], stroke_w);
    // GALHOS ***********************************************************
    galhos();
    // NÓS **************************************************************
    for (int l = 0; l<r; l++) {
      no( ramos[l][2], ramos[l][3], tronco_w);
    }

    // Flores ***********************************************
    for (int k = 0; k<galhos_i; k++) {
      flor(galhos[k][2], galhos[k][3], tronco_w);
    }
    arvore = false;
  }
}

////////////////////////////////////////////////////////////////////////////

void mousePressed() {
  mpress = !mpress;
  // preview
  click[0] = mouseX;
  click[1] = mouseY;
  //Tronco ************************************************************
  tronco[0] = mouseX;
  tronco[1] = mouseY;

  mira(click[0], click[1]);

}

////////////////////////////////////////////////////////////////////////////

void mouseReleased() {
  galhos_i = 0; //reset do indice da matriz de galhos

  mpress = !mpress;
  //preview
  click[2] = mouseX;
  click[3] = mouseY;

  // Arvore ************************************************************

  //Tronco ************************************************************
  tronco[2] = mouseX;
  tronco[3] = mouseY;

  //RAMOS *************************************************************
  r = int(random(nramos));                // numero de ramos (pelo menos 1)
  for (int i = 0; i<r; i++) {    // numero de iterações (galhos por ramo)
    ramos_g[i] = int(random(3));
    ramos[i][0] = tronco[2];
    ramos[i][1] = tronco[3];

    ramos_ang[r] = random(PI) *-1; //+- 180º //colocar em listas
    ramos[i][2] = tronco[2] + int(cos(ramos_ang[r])*(tronco_d/2)); //
Triangulo h, v, y; cos(a) = h/y; h = cos(a)*y;
    ramos[i][3] = tronco[3] + int(sin(ramos_ang[r])*(tronco_d/2));
```

```
    // GALHOS POR RAMO ************************************************
    ramos_g[i] = int(random(ngalhos)+1);
    for (int j=0; j<ramos_g[i]; j++) {
      galhos_ang[galhos_i] = random(PI) *-1; //+- 180º //colocar em listas
      galhos[galhos_i][0] = ramos[i][2];
      galhos[galhos_i][1] = ramos[i][3];
      galhos[galhos_i][2] = ramos[i][2] +
int(cos(galhos_ang[galhos_i])*(tronco_d/4));
      galhos[galhos_i][3] = ramos[i][3] +
int(sin(galhos_ang[galhos_i])*(tronco_d/4));

      galhos_i++;

    }
  }


  if (!arvore) {
    arvore = true;
  }

}

////////////////////////////////////////////////////////////////////////////

void marcascorte() {
  //Marcas de corte line(x1, y1, x2, y2)
  stroke(0);

  line(0, h2, w1, h2);
  line(0, h3, w1, h3);
  line(w4, h2, SCREEN_W, h2);
  line(w4, h3, SCREEN_W, h3);

  line(w2, 0, w2, h1);
  line(w2, h4, w2, SCREEN_H);
  line(w3, 0, w3, h1);
  line(w3, h4, w3, SCREEN_H);
}

////////////////////////////////////////////////////////////////////////////

void colorchooser() {
  //noStroke();
  stroke(0);

  int c = 10;

  fill(rgb_1);
  rect(w2+c*1, 5, c, c);

  fill(rgb_2);
  rect(w2+c*2, 5, c, c);

  fill(rgb_3);
  rect(w2+c*3, 5, c, c);
}

////////////////////////////////////////////////////////////////////////////

void creditos() {
  //background(255);
```

```
  stroke(100);
  strokeWeight(1);
  //Créditos Individuais
  tipo_1 = loadFont("Georgia-48.vlw");
  textFont(tipo_1, 10);
  fill(0);
  text("Pedro Amado © 2006", 75, 15);
}

////////////////////////////////////////////////////////////////////////////

void flor(int p1, int p2, float p3) {
  // Para o caso da grossura do tronco ser variavel
  if (p3<10) {
    p3=10;
  }
  strokeWeight(1);

  stroke(255, 0, 0);
  fill(255, 0, 0);
  ellipse(p1, p2, p3/8, p3/8);

  stroke(255,0,0, 50);
  fill(255, 255, 255, 50);
  ellipse(p1, p2, p3/3, p3/3);

  stroke(255,0,0);
  noFill();
  ellipse(p1, p2, p3, p3);
}

////////////////////////////////////////////////////////////////////////////

void no(int p1, int p2, float p3) {
  // Para o caso da grossura do tronco ser variavel
  if (p3<10) {
    p3=10;
  }

  noStroke();

  fill(255, 255, 255);
  ellipse(p1, p2, p3/8, p3/8);

  strokeWeight(2);
  stroke(255,0,0);
  noFill();
  ellipse(p1, p2, p3, p3);
}

////////////////////////////////////////////////////////////////////////////

void tronco() {
  stroke(tronco_c);
  tronco_w = stroke_w;
  strokeWeight(tronco_w);
  line(tronco[0], tronco[1], tronco[2], tronco[3]);
}

////////////////////////////////////////////////////////////////////////////
```

```
void ramos() {

  for (int i = 0; i<r; i++) {
    strokeWeight(0.75*tronco_w);
    line(ramos[i][0], ramos[i][1], ramos[i][2], ramos[i][3]);
  }

}

////////////////////////////////////////////////////////////////////////////

void galhos() {
  for (int j = 0; j< galhos_i; j++) {
    stroke(tronco_c);
    strokeWeight(0.5*tronco_w);
    line(galhos[j][0], galhos[j][1], galhos[j][2], galhos[j][3]);
  }

}

////////////////////////////////////////////////////////////////////////////
void keyPressed() {
  switch (key) {
  case 't':
    trail = !trail;

    break;
  case 'w':
    var_w = !var_w;

    break;
  case 'd':
    diagnose();
    break;
  case 'r':
    if (!recordpdf) {
      recordpdf = true;
      //PDF
      beginRecord(PDF, "Sketch.pdf");
      background(255);
      creditos();
      marcascorte();
      colorchooser();
    }
    else if (recordpdf) {
      endRecord();
      recordpdf = false;
    }
    break;
case 'c': //Clear
background(255);
      creditos();
      marcascorte();
      colorchooser();
  }
}

////////////////////////////////////////////////////////////////////////////

void mira(int p1, int p2) {
  stroke(255,0,0);
  strokeWeight(1);
```

```
  int a, b, c, d;
  a = p1-5;
  b = p2-5;
  c = p1+5;
  d = p2+5;
  line(a, p2, c, p2);
  //line(p1, b, p1, d);
}

//////////////////////////////////////////////////////////////////////////

void diagnose() {
  println("Variable width = "+var_w);
  println("Stroke = "+stroke_w);
  println("Trail = "+trail);
}

//////////////////////////////////////////////////////////////////////////

void stop() {
  endRecord();
}
```

BIBLIOGRAFIA

CARDOSO, Jorge – **Sebenta de Programação Multimédia** [Em linha]. UCP: Porto, 2006. 16 Fev 2006 Disponível na WWW: URL:http://teaching.jotgecardoso.org/pm.

FELLEISEN, Bruce; FINDLER, Robert; et. al. - **How to Design Programs: An Introduction to Programming and Computing** [Em linha]. MIT Press: Cambridge, MA, 2001 [Actual. em Set. 2003] Disponível na WWW: <URL: http://www.htdp.org/>. ISBN-10 0-262-06218-6, ISBN-13 978-0-262-06218-3

MENDES, António José; MARCELINO, Maria José – **Fundamentos de Programação em Java 2**. FCA: Lisboa, [s.d.]. ISBN 972-722-423-7

NETO, Joao Pedro - **Programação, Algoritmos e Estruturas de Dados**. Escolar: Lisboa, 2004. ISBN 972-592-179-8

A PUBLICAR

GREENBERG, Ira - Processing: Creative Coding and Computational Art. Friends of Ed: Berkley, 2007. ISBN: 159059617X (http://www.friendsofed.com/book.html?isbn=159059617X)

REAS, Casey; FRY, Ben - Processing: A Programming Handbook for Visual Designers and Artists. MIT Press: Cambridge, 2007. ISBN: 0-262-18262-9 (http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=11251)

Sugestões:
pamado(at)fba.up.pt

LINKS

RECURSOS EDUCATIVOS

Cadeira leccionada pelo Engº Jorge Cardoso da UCP
http://teaching.jorgecardoso.org/

Website do Processing:
http://processing.org/

Website de Marius Watz:
http://workshop.evolutionzone.com/

Website de Zachary Lieberman:
http://www.thesystemis.com/

Website da UCLA (Casey Reas e associados):
http://classes.design.ucla.edu/

Website de Daniel Shiffman
http://www.shiffman.net/

Fórum de discussão e dicas sobre programação em Português:
http://www.portugal-a-programar.org/forum/

EXEMPLOS E *SITES* VARIADOS

http://acg.media.mit.edu/people/fry/tendril/ (Tendril)
http://acg.media.mit.edu/people/fry/valence/ (Valence)
http://benfry.com/
http://dbn.media.mit.edu/
http://en.wikipedia.org/wiki/Algorithm
http://en.wikipedia.org/wiki/Concept_art
http://en.wikipedia.org/wiki/Conceptual_art
http://en.wikipedia.org/wiki/Functional_programming

http://en.wikipedia.org/wiki/Timeline_of_programming_languages
http://en.wikipedia.org/wiki/Programming
http://lawsofsimplicity.com/category/laws?order=ASC (Último livro de Maeda)
http://medienturm.at/mt.php?id=2&sid=225&k=full&_pid=412
http://plw.media.mit.edu/people/maeda/
http://processing.org/
http://processing.org/exhibition/works/metropop/
http://processing.org/faq/index.html
http://processing.org/learning/index.html
http://processing.org/reference/index.html
http://reas.com/
http://simplicity.media.mit.edu/ (Simplicity Blog, Maeda)
http://thesystemis.com/dbn/ (Design by Numbers – Parsons NY)
http://www.aaronkoblin.com/
http://www.flong.com/
http://www.franklinfurnace.org/history/flow/lewitt/lewitt.html
http://www.generatorx.no/?m=200609
http://www.maedastudio.com/index.php
http://www.re-move.org/
http://www.sodaplay.com/
http://www.theaustrianabstracts.net
http://www.tom-carden.co.uk/p5/2004/05/attractors-and-particles.php
http://www.ubu.com/
http://en.wikipedia.org/wiki/4%E2%80%B233%E2%80%B3
http://en.wikipedia.org/wiki/John_Cage
http://pt.wikipedia.org/wiki/Sol_LeWitt
http://upload.wikimedia.org/wikipedia/commons/f/f6/Duchamp_Fountaine.jpg
http://www.esono.com/boris/projects/poetry06/
http://en.wikipedia.org/wiki/Timeline_of_programming_languages
http://en.wikipedia.org/wiki/Comparison_of_programming_languages
http://en.wikipedia.org/wiki/Programming_language_implementation
http://en.wikipedia.org/wiki/Scripting_language
http://en.wikipedia.org/wiki/Programming_language_timeline
http://en.wikipedia.org/wiki/Timeline_of_computing
http://en.wikibooks.org/wiki/Computer_programming
http://en.wikipedia.org/wiki/Programming_languages
http://www.htdp.org/

ÍNDICE REMISSIVO

A construir.

Sugestões:
pamado(at)fba.up.pt

FAQ (PERGUNTAS FREQUENTES)

A construir.

Sugestões:
pamado(at)fba.up.pt

PARA O FUTURO

Classes

(Classes Dinâmicas)

Objectos:

PSound

PImage (dinâmico)

Vídeo

Bibliotecas:

ESS – Áudio e FFT

Quicktime – Vídeo Export

MySQL Database integration

HTML Parser

WACOM Tablet Interaction

Sugestões:
pamado(at)fba.up.pt