LÓGICA DE PROGRAMAÇÃO E ALGORITMOS

AULA 1

Prof. Vinicius Pozzobon Borin

CONVERSA INICIAL

O objetivo desta aula é introduzir os principais conceitos inerentes à esta disciplina. Iniciaremos compreendendo o que significam as palavras que compõem o nome de nossa disciplina: lógica e algoritmos. Investigaremos um pouco do raciocínio lógico necessário para o desenvolvimento de programas computacionais, bem como definiremos, em um contexto até mesmo fora da computação, o que são algoritmos.

Após essas definições, trataremos dos recursos necessários para que um sistema computacional execute algoritmos; portanto, veremos a arquitetura computacional base para a execução de programas, o *hardware*.

Veremos maneiras distintas de representar um algoritmo. Para tal, estudaremos a nomenclatura de pseudocódigo e de fluxogramas. Esses conceitos nos acompanharão ao longo de todo o nosso curso.

Por fim, encerraremos esta primeira aula compreendendo o que são linguagens de programação e como um sistema computacional a entende. Focaremos o entendimento da linguagem **Python**, a qual adotaremos para o estudo na disciplina.

Ao longo de toda a aula serão apresentados exemplos de algoritmos escritos tanto em pseudocódigo quanto em fluxogramas, e em linguagem de programação Python, que será conceituada e apresentada no decorrer de nosso curso. Todos os conceitos trabalhados nesta Aula 1 continuarão aparecendo de forma recorrente ao longo de todas as nossas próximas aulas.

TEMA 1 – INTRODUÇÃO À LÓGICA E AOS ALGORITMOS

Nossa disciplina é chamada de *Lógica de Programação e Algoritmos*. Portanto, nada melhor do que iniciarmos nossa primeira aula conceituando o nome de nossa disciplina. Afinal, o que é lógica? E o que são algoritmos? E como tudo isso se conecta na área de computação e programação? Compreenderemos isso nas próximas páginas.

1.1 INTRODUÇÃO À LÓGICA

Você já parou para pensar na maneira com que você racionaliza para realizar suas tarefas cotidianas? Por exemplo, ao acordar você segue, mesmo que de maneira não intencional, uma ordem na qual realiza suas atividades. Quando acorda você se veste, escova seus dentes, toma café, dentre outras tarefas. Cada pessoa segue sua rotina como prefere, mas todos temos uma sequência de passos que mais nos agrada.

Tudo o que fazemos e seguimos em nossas rotinas tem um motivo, o qual é baseado em nosso raciocínio lógico, desenvolvido ao longo da vida. Por que nos vestimos ao acordar? Bom, porque sair de pijama na rua está fora da convenção social. Por que tomamos café ao acordar? Porque precisamos de energia para o longo dia de trabalho que virá. E, assim, podemos questionar o porquê do que fazemos, e veremos que temos uma resposta racional mesmo que faça sentido somente para nós mesmos.

O **raciocínio lógico** adotado para estabelecer nossas tarefas é um pensamento que adotamos desde os nossos tempos primitivos. E ele rege a maneira como conduzimos o nosso dia a dia. Esse tipo de raciocínio provém da ciência da lógica. A **lógica** foi, pela primeira vez, conceituada na Grécia Antiga por Aristóteles (384-322 a.C.). A palavra lógica tem sua origem no termo grego *logos*, e significa **linguagem racional**. De acordo com o dicionário Michaelis (S.d.), lógica é: "Parte da filosofia que se ocupa das formas do pensamento e das operações intelectuais".

Trazendo o conceito de lógica para a área da computação e informática, novamente de acordo com o dicionário Michaelis (S.d.), o termo *lógica* associado a essa área é dado como: a "Maneira pela qual instruções, assertivas e pressupostos são organizados num algoritmo para viabilizar a implantação de um programa".

Certo; o conceito do dicionário nos trouxe uma nova palavra: **algoritmo**. Ainda não definimos o que é um algoritmo! Faremos isso na próxima subseção. O que é relevante neste momento inicial é que se compreenda que o uso de lógica é a base para o desenvolvimento de *softwares* computacionais. E é esse tipo de raciocínio lógico que desenvolveremos e trabalharemos ao longo desta disciplina.

Veremos que o raciocínio lógico inevitavelmente nos leva a resultados do tipo binário – **Verdadeiro** (V) ou **Falso** (F) – para toda e qualquer proposição, não havendo a possibilidade de o

resultado ser ambos, simultaneamente.[1] Trabalharemos bastante com esse tipo de análise, então, vejamos um exemplo matemático a seguir:

$$\begin{cases} \boldsymbol{p}: x \text{ é } maior\ ou\ igual\ a\ 10? \\ \boldsymbol{q}: se\ x \text{ é } maior\ ou\ igual, então\ V(p) = V. \\ \boldsymbol{r}: se\ x \text{ é } menor, então\ V(p) = F. \end{cases}$$

em que *p* é a preposição, *x* é um valor arbitrário, inteiro e positivo e *V(p)* é o resultado lógico da preposição, podendo ser V ou F. Nunca ambos simultaneamente, assim como nunca ficaremos sem uma resposta.

Caso *x* seja o valor 11, o resultado será *V(p)=V*, pois 11 é maior do que 10. Se *x* for o valor 10, *V(p) = V* também o será, pois ambos são iguais. O resultado de *V(p) = F* ocorrerá somente se *x* for, por exemplo, 9 ou menos.

A lógica pode ser dividida em indutiva e dedutiva (Puga; Rissetti, 2016). Em ambas, temos premissas que nos fazem chegar a uma conclusão. A **indutiva** é aquela em que as premissas não garantem que a conclusão será verdadeira. Já a **dedutiva** é bastante interessante para a nossa área, pois premissas verdadeiras garantem uma conclusão também verdadeira. O exemplo dado anteriormente é baseado em lógica dedutiva, e trabalharemos somente com esse tipo de lógica na área da programação, afinal, um programa computacional necessita sempre ter certeza do resultado que deve ser obtido.

1.2 INTRODUÇÃO AOS ALGORITMOS

Vimos que a lógica está intimamente conectada com o conceito de algoritmos na área da computação. Mas, afinal, o que são algoritmos?

Assim como ocorre com a lógica, o conceito de algoritmo não é especificamente atrelado ao universo da informática, e precede em alguns séculos o surgimento da era digital. Já no século XVII, filósofos e matemáticos como o alemão Gottfried Wilhelm Leibniz discutiam seu conceito. Nessa época, linguagens matemáticas simbólicas e máquinas universais de cálculo eram muito empregadas para auxiliar em cálculos complexos, como diferenciais e integrais. Utilizar dispositivos como a

calculadora de Leibniz[2] requer uma sequência correta de passos para atingir o resultado. Na matemática, resolver problemas é seguir etapas na ordem correta.

Assim, o conceito de algoritmo surge. **Um algoritmo é dado como uma sequência de passos a serem realizados para que uma determinada tarefa seja concluída, ou um objetivo, atingido.**

A ideia do uso de algoritmos antes da era da informatização foi fundamental na matemática, uma vez que naturalmente empregamos uma sequência de passos fixa e imutável para resolver equações algébricas. Por exemplo, qual seria a sequência de passos para resolvermos a equação $[(a + b) * c + d]$? Considere que *a, b, c* e *d* sejam números inteiros e positivos.

1. realizar o cálculo dentro dos parênteses $a + b$;

2. multiplicar o resultado de dentro dos parênteses por *c;*

3. por fim, somar com *d*.

Observe que qualquer outra ordem de execução desses passos resultaria em um resultado incorreto, uma vez que na matemática devemos respeitar a ordem de precedência dos operadores, ou seja, primeiro calculamos o que está dentro dos parênteses e, por fim, fazemos o cálculo da multiplicação antes da adição, sempre.

Vejamos outros exemplos de algoritmos, mas agora mais de nosso cotidiano. Um exemplo clássico de algoritmo é uma receita de bolo. Qualquer receita de bolo é uma sequência exata de passos que, caso não sejam seguidos corretamente, resultarão em um bolo, no mínimo, imperfeito.

Vejamos outro exemplo prático: você está em sua casa e abre a geladeira. Como faz tempo que não vai ao mercado, você só encontra dentro dela manteiga, queijo e presunto fatiados. Você também lembra que tem algumas fatias de pão de forma no armário e pensa: “Vou fazer um sanduíche!”.

Para construir seu sanduíche, mesmo involuntariamente, você faz um algoritmo! Duvida? Quer ver só? Tente imaginar (ou anote em seu caderno) a sequência de passos que você utiliza para montar o seu lanche. Depois, volte aqui e compare com as etapas que descreverei mais adiante. Observe que você deve utilizar somente os ingredientes citados em nosso exemplo (i.e., pão de forma, queijo fatiado, presunto fatiado e manteiga).

1. pegue uma fatia de pão de forma;

2. com a ponta faca, raspe duas vezes na manteiga dentro do pote;

3. com a mesma faca que contém a manteiga, espalhe uniformemente a manteiga em um dos lados do pão de forma;

4. no mesmo lado que você espalhou a manteiga, coloque uma fatia de queijo e uma de presunto, esta última em cima da de queijo;

5. em cima das fatias, coloque o outro pão de forma e pronto, seu sanduíche está pronto.

Ficou parecido com o seu? É claro que cada um tem suas preferências culinárias, como colocar ainda mais manteiga. Ou quem sabe colocar mais fatias de queijo? De todo modo, a ideia é bastante semelhante à que foi anteriormente apresentada.

Note também o nível de detalhes inseridos no algoritmo. Se pensarmos que um computador deverá executar esse algoritmo, quanto mais carecer de detalhes, menos a máquina saberá como proceder, resultando em um sanduíche errado. Quer ver um exemplo de erro? Na etapa 2 do algoritmo, se não fosse especificado que a ponta da faca deveria ser passada na manteiga, qualquer parte dela poderia ser usada, inclusive o cabo. E caso não tivesse sido dito para se usar uma faca, bom, aí seria possível usar até mesmo a sua própria mão para espalhar a manteiga. Portanto, é necessário informar a maneira mais minuciosa de construir o sanduíche.

Exercício de revisão:

Escreva uma sequência de passos (um algoritmo) para resolver a equação do delta (Δ) na fórmula de Bhaskara ($\Delta = b^2 - 4ac$).

Resposta: disponível em Puga e Rissetti (2016, capítulo 2, p. 10).

Exercício de fixação:

"Um cliente deseja fazer a consulta do saldo de sua conta corrente no computador pela internet. Suponha que o computador já esteja ligado e conectado à internet. A seguir estão os passos que poderiam ser utilizados, porém, foram colocados fora de ordem. Organize-os na ordem correta.

a. Inserir a senha;

b. Clicar no botão “OK” de acesso;

c. Selecionar a opção de saldo;

d. Encerrar a sessão;

e. Abrir o navegador;

f. Preencher dados (números de agência e conta);

g. Confirmar ou digitar o nome do usuário;

h. Fechar o navegador;

i. Confirmar o endereço do site do banco (Puga; Rissetti, 2016, p. 16, ex. 5).

TEMA 2 – SISTEMAS DE COMPUTAÇÃO

Vimos que algoritmos são a essência para o desenvolvimento de programas de computadores, denominados de *softwares*. Embora esta disciplina seja totalmente destinada às práticas de desenvolvimento de programas computacionais, nenhum *software* por si só tem utilidade se não existir um *hardware* para executar o programa que foi desenvolvido. Portanto, é necessário que vejamos a base de um sistema computacional necessário para executar programas.

O *hardware* é todo o aparato físico responsável por compreender o algoritmo desenvolvido e executá-lo. Vamos, portanto, compreender um pouco da história dos primeiros computadores digitais.

O grande estímulo para o surgimento do conceito do primeiro computador eletrônico do mundo veio com a Segunda Guerra Mundial. Nessa época, cálculos matemáticos complexos eram necessários para que se alcançasse diferentes objetivos. Um deles era o cálculo de rotas e trajetórias de mísseis de guerra.

Uma outra necessidade interessante era a codificação de mensagens contendo ordens, informações de combate, movimentações do inimigo etc., para as tropas. Mensagens como essas eram transmitidas via ar, ou mesmo água, e podiam ser interceptadas. Assim, era essencial que fossem enviadas em códigos. Desse modo, a necessidade de codificar mensagens que fossem

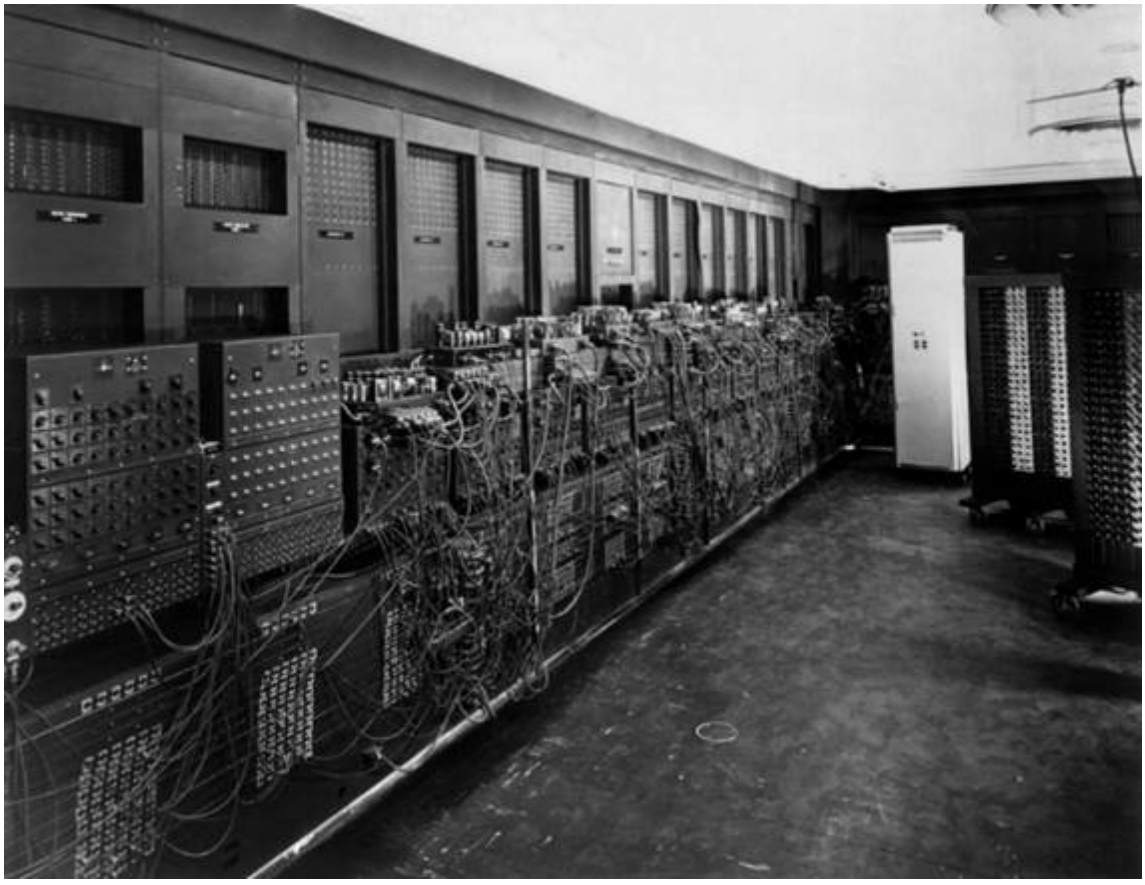
extremamente difíceis de ser decodificadas pelo inimigo era fundamental, e cálculos de alta complexidade eram exigidos.

O interesse pela decodificação vinha de ambos os lados. A própria nação emissora da mensagem precisava ser rápida em decodificar a sua mensagem, pois ordens de guerra precisam ser atendidas com urgência. Porém, o inimigo também tem grande interesse em interceptar e decodificar essa mesma mensagem, obtendo informações de guerra cruciais sobre o adversário.

Assim, uma corrida por máquinas capazes de realizar codificações e decodificações cada vez mais complexas e mais rapidamente se iniciou.

No início da guerra, tínhamos computadores construídos com milhares de válvulas e relés pesando toneladas e consumindo montantes gigantes de energia elétrica. Um dos mais famosos desse tipo foi o ENIAC (do inglês *Electronic Numerical Integrator and* Compute). A Figura 1 contém uma imagem desse imenso computador.

Figura 1 – Foto do ENIAC



Fonte: Everett Historical/Shutterstock.

O ENIAC começou a ser construído em 1943, e continha 18 mil válvulas, 1500 relés, pesava 30 toneladas e consumia 140 KW de potência. Tudo isso para ser capaz de manipular 20 unidades de memória (registradores), cada uma com um tamanho em decimal de 10 dígitos. O ENIAC só ficou pronto em 1946, após o término da guerra, e acabou nem sendo útil para tal propósito. Ainda, o ENIAC continha 6 mil interruptores que precisavam ser manualmente configurados para programar o computador (Tanenbaum, 2013).

Nessa época, começou a ficar evidente que máquinas como o ENIAC podiam não ser a melhor solução para o meio da computação devido à falta de praticidade em seu uso, e pesquisadores ao redor do mundo começaram a estudar novas maneiras de projetar e construir computadores.

Em 1946, o matemático húngaro John von Neumann, apoiando-se em trabalhos anteriores de pesquisadores como Alan Turing, propôs o que viria a ser de fato o primeiro computador digital. E não só isso, mas o húngaro criou a arquitetura computacional base empregada em todos os computadores modernos até os dias atuais.

Logo de início, von Neumann percebeu que programar computadores com imensas quantidades de interruptores é uma tarefa lenta, cara e manual demais, requerendo muita mão de obra só para acionar os interruptores. Ele também percebeu que fazer computadores trabalhar com aritmética decimal (a mesma à qual nós humanos estamos habitualmente familiarizados) não fazia sentido, uma vez que para representar um dígito eram necessárias 10 válvulas (1 acesa e 9 apagadas). Substituir por uma aritmética binária seria um caminho de mais fácil manipulação pelos computadores, caminho este adotado até hoje.

O projeto, que ficou posteriormente conhecido como *máquina de von Neumann*, foi então o primeiro computador de programa armazenado criado.

2.1 A MÁQUINA DE VON NEUMANN

Dedicaremos um tempo para entender melhor o que von Neumann propôs. Afinal, essa é a arquitetura base de qualquer sistema computacional do século XXI. A máquina de von Neumann contém cinco elementos básicos, como ilustra a Figura 2.

Figura 2 – Diagrama da máquina de von Neumann

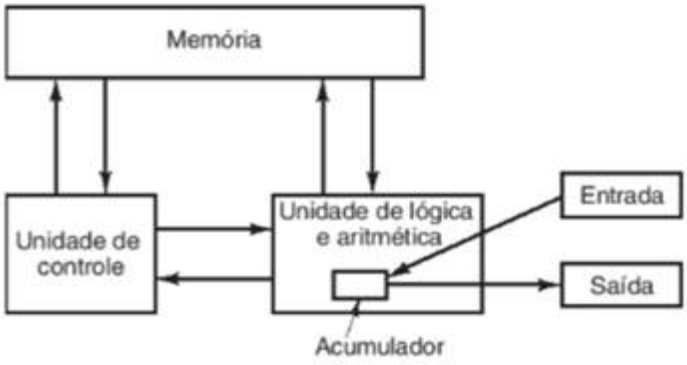


Figura 1.4
Máquina original de Von Neumann.

- **memória:** armazena o programa (algoritmo) em execução pela máquina, sendo capaz de armazenar dados usados pelo programa e instruções de controle. Tudo é armazenado em codificação binária. A memória que mantem os programas em execução é a memória RAM (*Random Acess Memory*);
- **unidade lógica e aritmética (ULA):** esta unidade é responsável por todos os cálculos aritméticos e lógicos do computador. Todas as operações são realizadas aqui. Uma máquina computacional é capaz de realizar somente cálculos aritméticos e lógicos dos mais simples possíveis, como somas, subtrações e operações relacionais (como relação de maior ou menor). Dentro dessa unidade, temos um registrador especialmente denominado de *acumulador*. O acumulador é responsável por armazenar temporariamente os resultados de operações dentro da ULA;
- **unidade de controle (UC):** gerencia o fluxo de execução dos programas dentro do computador. Define qual será a instrução seguinte a ser a executada e onde ela se encontra na memória;
- **dispositivos de entrada:** são as maneiras físicas pelas quais inserimos informações em nossos programas. Um dispositivo de entrada comum é o teclado em que digitamos, e as informações são transferidas ao programa;
- **dispositivos de saída:** são maneiras pelas quais o resultado de uma ação do programa informa o usuário. Podemos citar como dispositivos de saída tela/monitor, ou mesmo lâmpadas indicativas acessas.

A ULA e a UC formam o que é de chamado de processador, ou seja, a CPU (*Central Process Unit*) de um computador, e são sempre construídas juntas dentro de um único *chip*. Portanto, nos dias atuais, quando você compra um processor está adquirindo basicamente esses dois módulos.

Inicialmente, quando von Neumann propôs sua arquitetura, a memória não fazia parte do *chip* da CPU. Hoje, nas máquinas modernas, temos memórias de altíssimas velocidades inseridas dentro desse *chip* também, com o objetivo de melhorar o desempenho de processamento. Essas memórias inclusas no *chip* são chamadas de memória *cache*.

Em suma, a arquitetura de von Neumann, hoje considerada clássica dentro dos computadores modernos, é tida como o *hardware* mínimo necessário para um computador digital funcionar. Qualquer *software* de computador que aprenderemos a desenvolver ao longo desta disciplina requer uma máquina de von Neumann.

Desde a proposta de um computador digital feita por von Neumann em meados de 1945, as tecnologias necessárias para a sua construção evoluíram abruptamente. Na década de 1950, deu-se início à era da miniaturização dos computadores, com o surgimento dos primeiros dispositivos construídos à base de transistores, e não mais valvulados. Em 1960, evoluímos para projetos com circuitos integrados, reduzindo ainda mais os computadores e chegando a máquinas com tamanhos e custos compatíveis para vendas residenciais. Toda a história e a evolução dos computadores podem ser lidas em detalhes no livro *Organização estruturada de computadores* (Tanenbaum, 2013a).

De todo modo, o que todas essas gerações têm em comum é que a arquitetura que serviu como base está contida dentro de todas as nossas máquinas digitais, e não só computadores e *laptops*, mas também *smartphones* e até mesmo dispositivos eletrônicos embarcados, como forno de micro-ondas, lâmpadas inteligentes etc. Tudo isso tem von Neumann como ponto de partida.

2.2 A MENOR UNIDADE DE ARMAZENAMENTO: O *BIT*

Sabemos que um computador digital contém uma memória que manipula e armazena dados de maneira binária. Mas o que isso realmente significa?

Aprendemos, desde os primeiros anos de escola, a realizar operações aritméticas na denominada *base decimal*. Decimal, que vem de 10, nos indica que, para representarmos qualquer número, precisamos de 10 símbolos diferentes, nesse caso, números de zero até 9. Podemos realizar qualquer combinação de dígitos de 0-9 para representar o número que queremos em decimal.

O termo ***bit*** é originário de *binary digit* (dígito binário), de que escolheu-se pegar as duas primeiras letras da palavra *binary (bi)* e a última letra da palavra *digit* (t) para formar a palavra *bit*

(bi+t). Um *binary digit* nos indica que, diferentemente da base decimal, na base binária podemos representar números utilizando somente dois (bi) símbolos. Nesse caso, zero ou 1. Qualquer número que antes podíamos representar em decimal, podemos também representar em base binária a partir de combinações de zeros e uns, somente.

Um computador digital armazena, calcula e manipula com esse tipo de aritmética devido à facilidade que é para ele realizar operações nessa base. Como citado, a menor unidade de informação armazenada é zero ou 1, que também são chamados de nível lógico baixo (0) e nível lógico alto (1), ou então de **falso** (0) e **verdadeiro** (1).

Em aspectos físicos, um valor zero em uma memória poderá ser a ausência de um sinal elétrico (ou carga elétrica) naquele ponto, e o valor 1 pode ser representado pela existência de sinal elétrico (ou carga elétrica). Se pudéssemos enxergar a olho nu as informações armazenadas em uma memória, veríamos quantidades quase que infinitas de sequências de sinais (ou cargas elétricas) dentro dela.[3]

Se o computador trabalha com aritmética binária, talvez você esteja se perguntando: como é que para nós, usuários, as informações aparecem na tela em base decimal? Bom, o fato é que é possível converter números de uma base para a outra. Embora a aritmética binária e os cálculos de conversão de base não sejam objeto de estudo desta disciplina, caso tenha interesse em compreender como convertemos decimal em binário, ou vice-versa, recomendo a leitura dos capítulos 1 e 2 do livro *Sistemas digitais: princípios e aplicações* (Tocci; Widmer; Moss, 2013).

O que se deve compreender aqui é que todo e qualquer computador projetado com base na máquina de von Neumann é binário e, portanto, só compreende dígitos 0 e 1, o *bit*, nada além disso.

O *bit* é a menor unidade de armazenamento. Porém, é mais comum mensurarmos a memória em unidades maiores para facilitar nossa compreensão. Sendo assim, adotamos convenções de que um octeto, ou seja, **8 *bits***, é denominado de *Byte*. Assim como **1024 Bytes correspondem a 1 KiloByte.** Veja a tabela de nomenclatura a seguir:

Tabela1 – Nomenclatura

	Equivale à	Abreviação
8 *bits*	1 *Byte*	B
1024 *Bytes*	1 *KiloByte*	KB
1024 KB	1 *MegaByte*	MB
1024 MB	1 *GigaByte*	GB
1024 GB	1 *TeraByte*	TB
1024 TB	1 *PetaByte*	PB

Ao longo de nossa disciplina de programação e de outras vindouras na área de computação, estaremos sempre observando nossos dados armazenados na memória do computador em alguma das unidades apresentadas conforme a Tabela 1.

2.3 O SISTEMA OPERACIONAL

Considere a seguinte situação: você precisa desenvolver um *player* de música para rodar em um computador de mesa ou em um *laptop*. O *player* tem como característica acessar músicas salvas em uma memória da máquina e reproduzir em uma saída de áudio do computador.

Observe que o universo de configurações de *hardware* dos computadores tende ao infinito, pois temos centenas de marcas e modelos, com *hardwares* dos mais distintos, tecnologias diferentes de diversos fabricantes, capacidades de memória e processamento completamente distintos. Sem falar nas possibilidades de saídas de som que podemos ter para a aplicação do *player* (fone de ouvido, *subwoofer*, *home theater*).

Se um programador precisar se preocupar com o *hardware* disponível em cada máquina, será inviável desenvolver qualquer programa de computador nos dias de hoje. Ou o programador teria que considerar todos os *hardwares* existentes no mundo, algo impossível, ou então desenvolver o programa para uma única plataforma e uma única configuração, tornando o *software* limitado e comercialmente inviável.

E, acredite, até meados da década de 1970 era assim que se desenvolviam programas computacionais: exclusivos para um determinado *hardware*. Porém, é importante ressaltar que tínhamos bem menos *hardwares* e tecnologias naquela época.

O surgimento do conceito de sistema operacional fez com que a vida do desenvolver melhorasse bastante. Um sistema operacional é também um *software*, mas bastante complexo e capaz de gerenciar a execução de outros programas executando em concorrência em uma mesma máquina. O sistema operacional, dentro de suas principais características, tem como objetivo permitir ou não a execução de um *software*, gerenciar o uso da memória e do processador por cada um dos programas e, por fim, abstrair por completo o *hardware* da máquina, tanto para o usuário quanto para o programador. Desse modo, ao desenvolver o seu *player* de música, você não irá desenvolvê-lo para um *hardware* específico, mas, sim, para um sistema operacional específico, o que facilita enormemente o processo de desenvolvimento.[4]

Talvez você esteja se perguntando se já utilizou algum sistema operacional ao longo de sua vida de usuário de computadores. A resposta é: com toda a certeza! Quer ver alguns exemplos?

- em computadores (*desktops* e *laptops*), os mais conhecidos são os da Microsoft, que iniciou com o MS-DOS, e foi seguido pelo Windows. Em contrapartida, temos os baseados em UNIX/Linux e suas diferentes distribuições (Ubuntu, Mint, Fedora etc.). Citamos também o sistema da Apple, o macOS. Dentre outros menos conhecidos, citamos o FreeBSD e Solaris;
- no meio dos dispositivos móveis, temos sistemas operacionais desenvolvidos exclusivamente para eles, e que se preocupam mais em atender necessidades desse tipo de mercado, como otimização máxima dos recursos de energia. Dentre os largamente adotados, citamos o Android, da Google, e o iOS, da Apple;
- diversos outros sistemas computacionais também contêm um sistema operacional. Videogames são um exemplo. No Playstation 4 da Sony, temos uma variação do FreeBSD chamada de *Orbis OS*. No ramo de dispositivos embarcados inteligentes, cada vez mais comuns hoje em dia, sistemas operacionais são empregados. O mais difundido é o FreeRTOS.

TEMA 3 – REPRESENTAÇÕES DE ALGORITMOS

Algoritmos são sequências de passos a serem seguidos para que uma tarefa seja executada com sucesso. Esses passos, os quais podemos chamar também de instruções, apresentam maneiras distintas de representação, especialmente na área da computação, em que já buscamos trazer uma aproximação com a linguagem computacional.

Neste tópico, portanto, aprenderemos três diferentes tipos de representações de algoritmos. Essas representações o acompanharão ao longo de toda esta disciplina, bem como de sua vida como programador. São elas: descrição narrativa, pseudocódigo e fluxograma.

3.1 DESCRIÇÃO NARRATIVA

A maneira mais simples e intuitiva de construirmos nossos algoritmos é a partir de uma **linguagem natural**. Descrever as instruções utilizando esse tipo de linguagem significa representar os passos a serem seguidos em modo texto, por meio de frases, como se estivéssemos simplesmente tendo uma conversa informal com alguém. É por isso que a descrição narrativa apresenta pouco formalismo e é bastante flexível, não contendo regras. O problema disso é que abre margem para ambiguidades e dupla interpretação, o que faz com que acabe não sendo utilizada na construção de algoritmos computacionais.

Você se lembra do nosso algoritmo de sanduíche, lá na Seção 1.2? Pois bem, ele está em forma de descrição narrativa. Note que muitas das frases lá construídas, por mais detalhadas que estejam, estão bastante dúbias, e o que a instrução nos diz para fazer nem sempre fica claro.

Vejamos outro exemplo de descrição narrativa. A seguir, temos um algoritmo que recebe dois valores numéricos (x e y), verifica se eles são iguais ou não e informa, por meio de uma mensagem, o resultado dessa validação.

1. Ler dois valores (x e y).

2. Verificar se x e y são iguais.

3. Se x for igual a y, mostrar a mensagem “Valores iguais!”.

4. Se x for diferente de y, mostrar a mensagem “Valores diferentes!”.

5. Fim

Observe a falta de formalismo nesse exemplo. Na etapa 2, por exemplo, é dito para verificar se os valores são iguais, mas como essa verificação é feita? Não temos a informação disso, porque na descrição narrativa estamos preocupados somente com a ideia geral da instrução, e não com a implementação dela em um código computacional.

É por esse motivo que não trabalharemos com essa representação ao longo de nosso curso. Mas é interessante que se tenha conhecimento dela, uma vez que pode ser útil para ajudá-lo a formular ideias antes de se sentar na frente do computador. Também é útil para explicar um algoritmo seu para pessoas que não sejam necessariamente de sua área.

3.2 PSEUDOCÓDIGO

O pseudocódigo é a representação de um algoritmo mais próxima que podemos ter de um programa computacional, mas sem nos preocuparmos com qual linguagem de programação será adotada. Pseudocódigo, que significa **falso código**, é uma linguagem estruturada,[5] com regras e padrões bem definidos e fixos. Também conhecido como **português estruturado**, é uma maneira de escrever algoritmos sem necessitar de um *software* de programação ou de um computador (você pode trabalhar em seu próprio caderno). A grande vantagem de se aprender pseudocódigo é que ele corresponde a uma linguagem genérica e, uma vez entendendo a lógica por trás de seu pseudocódigo, basta aplicá-lo a qualquer linguagem de programação existente.

Vejamos o mesmo exemplo apresentado em descrição narrativa, mas agora nos atentando às regras impostas pela representação em pseudocódigo.

Exemplo de algoritmo:

```
1. Var
2.     x, y: inteiro
3. Início
4.     Ler (x, y)
5.     Se (x = y) então
6.         Mostrar (“Valores iguais!”)
7.     Senão
8.         Mostrar (“Valores diferentes!”)
9.     Fimse
```

10. Fim

Não vamos nos ater ainda às nuanças e regras de construção do pseudocódigo. Faremos isso ao longo do curso. Porém, você consegue perceber como temos um conjunto de regras formais bem definidas?

Por exemplo, um algoritmo em pseudocódigo deve, sempre, iniciar com a palavra **Algoritmo** seguida de um nome dado ao seu código (linha 1), assim como sempre finalizar com a palavra **Fim** (linha 11). Qualquer outra nomenclatura ali utilizada quebra o formalismo do pseudocódigo, resultando em erro. Outras palavras como **Ler**, **Mostrar**, **Se**, **Então** etc., estão ali empregadas propositalmente e têm significados específicos que serão trabalhados no decorrer desta disciplina.

3.3 FLUXOGRAMA

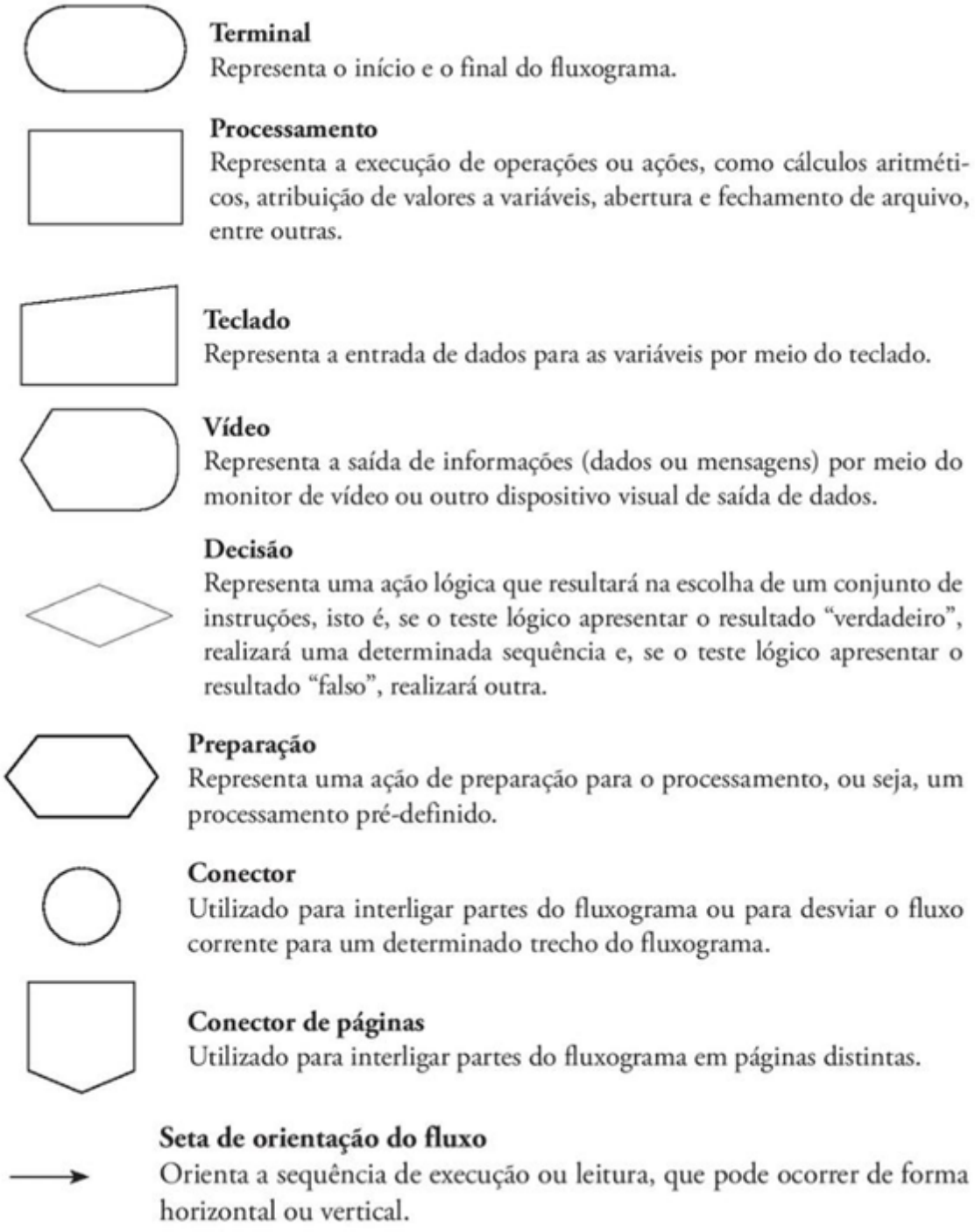
A última maneira que trabalharemos para representar algoritmos é o fluxograma. Esse tipo de representação serve para escrever um algoritmo de maneira gráfica e empregando símbolos.

Embora fluxogramas não sejam interpretados literalmente por nenhum *software* de programação, **o uso de fluxogramas é fundamental para representar a ideia de seu código e serve para organizar o seu raciocínio lógico**.

Também, fluxogramas são muito utilizados para mostrar algoritmos em trabalhos científicos, em apresentações acadêmicas e profissionais, uma vez que mostrar códigos completos em exposições orais tende a não ser uma boa prática didática. Assim, é fundamental que você saiba como construir fluxogramas de seus algoritmos, pois, em algum momento, precisará trabalhar com esse recurso, seja em outra disciplina, seja no mercado de trabalho.

Existe mais de uma nomenclatura de símbolos distintas para fluxogramas. A nomenclatura adotada nesta disciplina é a padronizada e documentada na norma ISO 5807:1985 e é também a adotada em nosso livro base (Puga; Rissetti, 2016). A seguir você encontrará a simbologia completa da ISO.

Figura 3 – Simbologia para fluxogramas ISO 5807:1985



Terminal
Representa o início e o final do fluxograma.

Processamento
Representa a execução de operações ou ações, como cálculos aritméticos, atribuição de valores a variáveis, abertura e fechamento de arquivo, entre outras.

Teclado
Representa a entrada de dados para as variáveis por meio do teclado.

Vídeo
Representa a saída de informações (dados ou mensagens) por meio do monitor de vídeo ou outro dispositivo visual de saída de dados.

Decisão
Representa uma ação lógica que resultará na escolha de um conjunto de instruções, isto é, se o teste lógico apresentar o resultado "verdadeiro", realizará uma determinada sequência e, se o teste lógico apresentar o resultado "falso", realizará outra.

Preparação
Representa uma ação de preparação para o processamento, ou seja, um processamento pré-definido.

Conector
Utilizado para interligar partes do fluxograma ou para desviar o fluxo corrente para um determinado trecho do fluxograma.

Conector de páginas
Utilizado para interligar partes do fluxograma em páginas distintas.

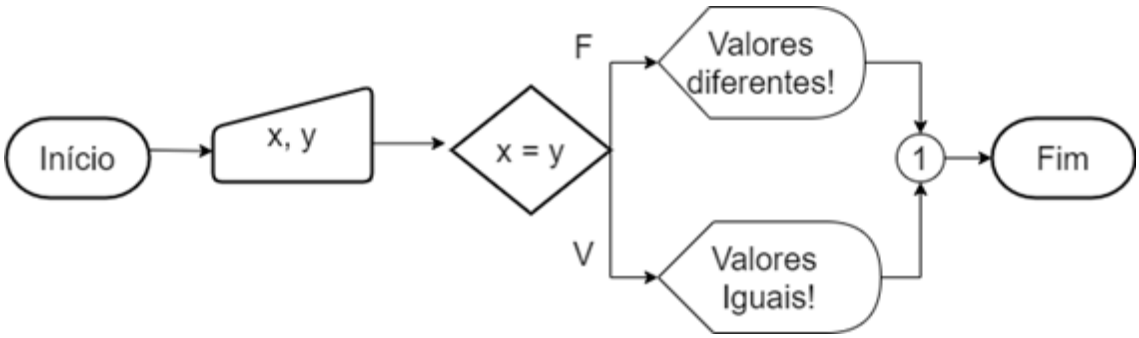
Seta de orientação do fluxo
Orienta a sequência de execução ou leitura, que pode ocorrer de forma horizontal ou vertical.

Fonte: Puga e Rissetti, 2016, p.

O mesmo algoritmo proposto na descrição narrativa e no pseudocódigo agora está representado em um fluxograma que pode ser visto na Figura 4.

Trabalharemos mais fluxogramas, em detalhes, no decorrer de nossa disciplina.

Figura 4 – Exemplo de fluxograma

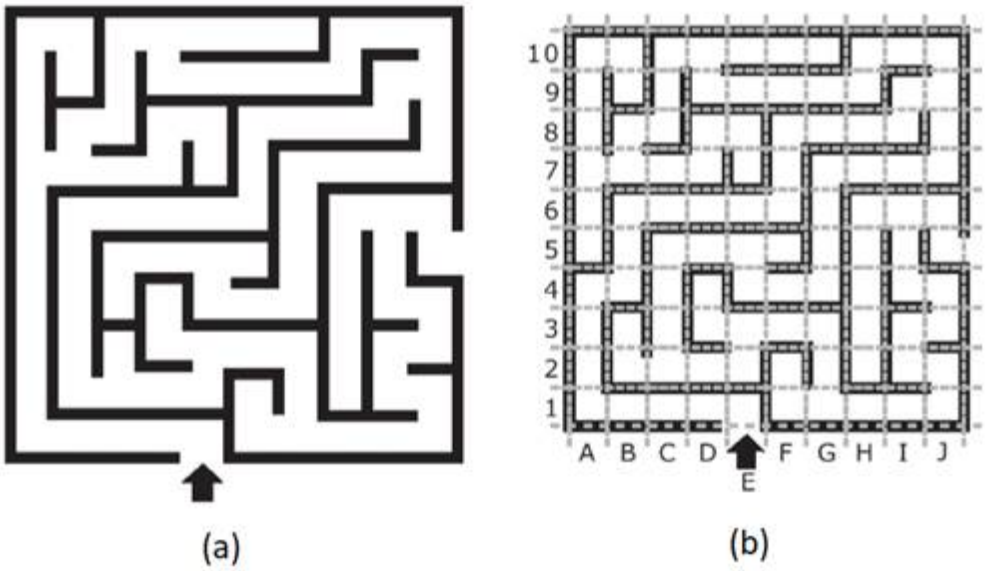


Exercício de fixação (adaptado de Puga e Rissetti, 2016, p. 16-17, exs. 3 e 9):

Quem nunca tentou encontrar o caminho correto em jogos de labirinto? Dependendo da complexidade do desenho isso pode tomar um tempo considerável, a não ser que exista um roteiro a ser seguido.

- tente elaborar um roteiro do caminho a ser seguido no labirinto, iniciando na seta da Figura 5a. Utilize a descrição narrativa para construir sua rota;
- tente indicar o caminho no labirinto da Figura 5b, agora usando a marcação feita para as linhas e colunas. O que é possível notar de diferente em seu algoritmo ao usar as marcações? As marcações o deixaram mais preciso?

Figura 5 – Labirintos do exercício: (a) sem marcações; (b) com marcações



Fonte: Puga e Rissetti, 2016, p.16-17.

TEMA 4 – LINGUAGENS DE PROGRAMAÇÃO E COMPILADORES

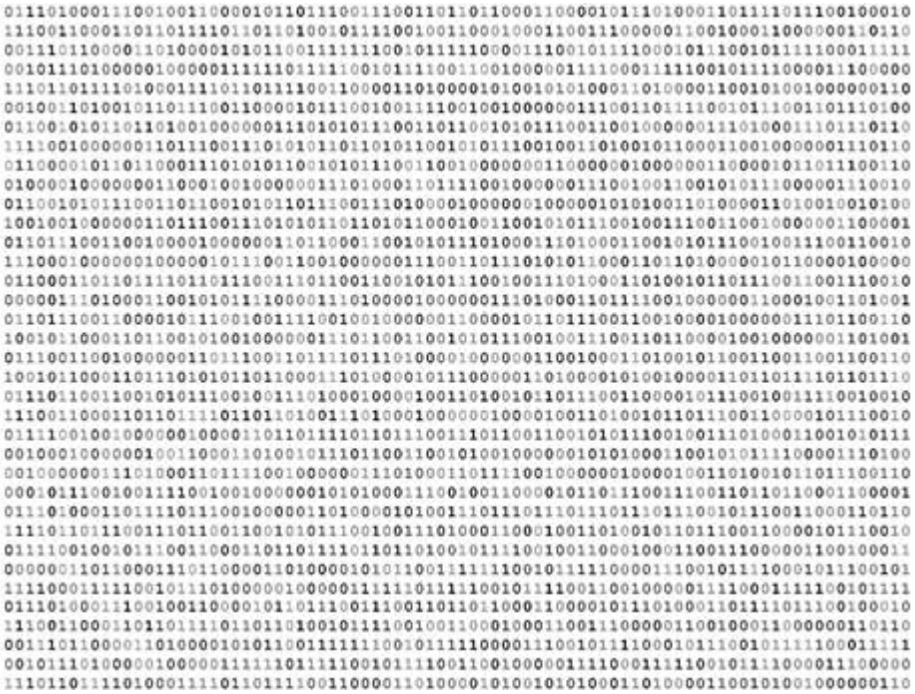
Aprendemos que pseudocódigo é uma maneira genérica de representar algoritmos em linguagem computacional, sendo o mais próximo que temos de uma linguagem de programação sem necessitar de um *software* para programar. Mas, afinal, o que é uma linguagem de programação?

4.1 LINGUAGEM DE PROGRAMAÇÃO

Conforme já aprendemos nesta aula, um computador trabalha com codificação binária e, portanto, não compreende mais nada além de *bits*. Agora, tente imaginar um cenário em que você, programador, se sente na frente de seu computador e comece a escrever o seu programa utilizando a linguagem que o computador compreende. O seu algoritmo se assemelharia ao ilustrado na Figura 6, ou seja, seria uma sequência quase infinita de zeros e uns.

Você consegue se imaginar escrevendo um código dessa maneira? Não? Pois bem, nem eu. Isso porque escrever em binário, embora não seja impossível, requer um esforço enorme, aumentando o tempo de desenvolvimento de um *software*.

Figura 6 – Exemplo de código em binário



Fonte : iunewind/Shutterstock.

Saiba que no passado já construímos códigos assim, em binário, mas naquela época desenvolvíamos programas bem mais simples e com pouquíssimos recursos. Um exemplo de máquina programada diretamente *bit* a *bit* foi o Altair 8800, criado em 1975, que funcionava com uma CPU Intel 8080. Todos os interruptores no painel frontal serviam para programar o computador.

Figura 7 – Painel frontal do Altair 8800

Fonte: Wikipédia, S.d. Disponível em: <https://pt.wikipedia.org/wiki/Altair_8800>.

O surgimento das linguagens de programação mudou para sempre a maneira como programamos. Com elas, passamos a escrever algoritmos de maneira muito mais simples e próxima à forma como nos comunicamos. Boas linguagens são pensadas para serem de fácil entendimento por um ser humano.

Não obstante, linguagens de programação apresentam outras facilidades, como escrever códigos que não dão margens a erros de interpretação pelo computador, graças à criação de conjuntos bem rígidos e específicos de regras que, se forem seguidos pelo programador, não resultarão em instruções ilegíveis pela máquina.

Uma linguagem de programação é, portanto, esse conjunto de regras, com palavras-chaves, símbolos e sequências específicas. A gama de linguagens que temos hoje no mercado é realmente grande e, em breve, comentaremos um pouco mais sobre algumas das mais importantes. Mas vamos primeiro a um breve histórico das linguagens.

4.2 HISTÓRICO DAS LINGUAGENS DE PROGRAMAÇÃO

Você se lembra do primeiro computador digital, o ENIAC? A grande máquina que só ficou pronta após o término da Segunda Guerra? O ENIAC, quando se tornou operacional em meados de 1946, contratou seis pessoas para trabalhar como seus programadores, todas elas mulheres.[6]

Naquela época, as programadoras começaram a desenvolver as primeiras técnicas de linguagens de programação com o objetivo de facilitar seu próprio trabalho. Simultaneamente, um outro grupo de programadores trabalhava em Harvard no computador Harvard Mark I. Nesse grupo, outra mulher ganhou destaque: Grace Hopper. Os esforços conjuntos de Hopper e das programadoras do ENIAC posteriormente deram origem à linguagem de programação COBOL (1959), ainda utilizada até os dias de hoje.

Outra linguagem de programação primitiva, surgida em 1954, foi a Fortran. Inventada pela IBM, foi a primeira linguagem de programação de propósito geral amplamente utilizada no mundo. A linguagem Fortran ficou bastante conhecida por ser uma linguagem bastante eficiente em desempenho e foi muito empregada em supercomputadores da época. Nos dias de hoje a linguagem é ainda utilizada, especialmente no meio acadêmico, em pesquisas que requerem alta complexidade matemática.

Nas décadas que se sucederam (1960 e 1970), com o COBOL e o Fortran se consolidando, os maiores paradigmas de linguagens de programação foram inventados. Nessa época, tivemos o surgimento da linguagem C, desenvolvida na Bell Labs. A linguagem C foi responsável por tornar popular o conceito de **biblioteca de funções**.

Você se lembra de quando falamos que programar se preocupando especificamente com um determinado *hardware* não é viável? Bom, essas bibliotecas continham centenas de códigos prontos para serem utilizados por programadores daquela linguagem. Dentro dessas bibliotecas era possível encontrar desde códigos para interfaceamento com *hardware* até códigos prontos para resolver problemas bastante comuns, como implementações de funções matemáticas não triviais.

Assim, o programador passou a se preocupar menos com pequenos detalhes, como *hardware*, podendo focar mais o seu algoritmo e seus recursos. O conceito de biblioteca é amplamente empregado até hoje e todas as linguagens de programação mais usadas no mercado contêm uma variedade bastante grande de bibliotecas pré-disponíveis.

Na década de 1980 tivemos a consolidação das linguagens imperativas, ou seja, linguagens de programação fortemente segmentadas, e o conceito de programação orientada a objetos estava então formalizado. Nesta disciplina de programação, não trabalharemos com esses paradigmas por se tratar de conceitos mais avançados de programação.[7]

Por fim, com o *boom* da internet em meados de 1990, tivemos o surgimento de uma gama bastante grande de linguagens para esse propósito, assim como o surgimento do conceito de linguagens funcionais, em que o foco é a produtividade do desenvolvedor, abstraindo cada vez mais aspectos técnicos e detalhes e agilizando o tempo de desenvolvimento.

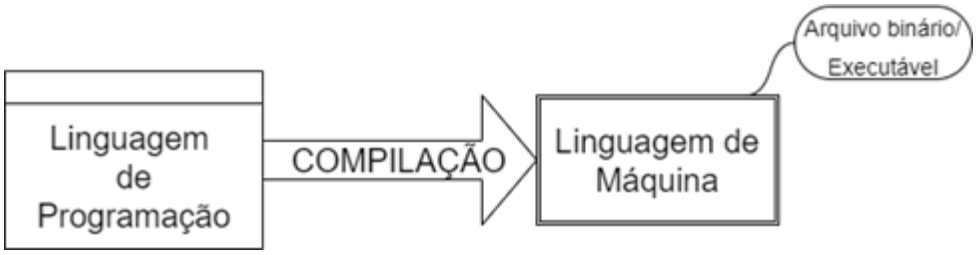
4.3 *SOFTWARE* DE COMPILAÇÃO

Descobrimos ao longo deste material que um computador trabalha e compreende somente a linguagem binária, também chamada de **linguagem de máquina** (vide Figura 6). Acabamos de descobrir que nós, programadores, escrevemos algoritmos computacionais não em binário, mas, sim, em uma linguagem de programação.

Se o computador compreende uma linguagem e você trabalha em outra, ilegível para ele, como então o computador entenderá o código feito por você?

Para resolver esse problema foram criados os *softwares* de compilação. Um **compilador** é capaz receber um código escrito em uma linguagem e realizar um processo chamado de **compilação**. A compilação é um processo bastante detalhado e complexo, o qual não será abordado nesta disciplina.[8] Porém, tudo que o *software* faz é transformar o código que você escreveu (também chamado de código de alto nível) em uma linguagem de máquina (linguagem baixo nível) compreendida pelo *hardware*. O resultado do processo de compilação normalmente é um arquivo do tipo binário (*binary file*), e em sistemas operacionais como Windows eles são do tipo executáveis (o famoso *.exe*). A Figura 8 ilustra esse processo.

Figura 8 – Processo de compilação



Fonte:

Cada linguagem de programação apresenta o(s) seu(s) respectivo(s) *software(s)* de compilação capaz(es) de compreendê-la. O *software* de compilação é também desenvolvido e dependente de um sistema operacional, ou seja, você acaba desenvolvendo programas específicos para uma só plataforma. Caso deseje trabalhar com mais plataformas, precisará realizar o processo de compilação novamente para cada plataforma, embora o código de alto nível praticamente não sofra alterações.

Por fim, é interessante ressaltar a diferença entre compilador e interpretador. O **compilador**, conforme já comentado, gera um arquivo binário legível para a máquina. Após o processo de compilação, o código alto nível escrito desaparece, não sendo mais possível reavê-lo de maneira alguma (ao menos que você seja o desenvolver e tenha o código fonte). Um **interpretador**, por sua vez, nunca perde o seu código fonte, ele está sempre presente. O código não é convertido de uma só vez, mas, sim, executado instrução por instrução à medida que o programa o requisita.

4.4 LINGUAGENS DE PROGRAMAÇÃO MAIS POPULARES

A gama de linguagens de programação é bastante grande e ultrapassa a casa da centena. Já comentamos que o COBOL e a Fortran foram duas das primeiras linguagens, e que acabam por ser utilizadas até os dias de hoje em aplicações específicas.

É fato que não existe nenhuma linguagem ideal para todos os cenários de desenvolvimento. Cada uma contém suas peculiaridades e características que as tornam boas e populares em certos ambientes. O *IEEE Spectrum* anualmente lança seu *ranking* de linguagens de programação mais usadas e procuradas. Este *ranking* é criado com base em pesquisas de usuários na internet, mas também se baseia no uso dessas linguagens no meio científico, verificando bases de dados de artigos. O *ranking* do ano de 2019 pode ser visto na Figura 9.

Figura 9 – *Ranking* de linguagens de programação mais utilizadas no ano de 2019, segundo o *IEEE Spectrum*

Fonte: IEEE Spectrum.

O uso de *rankings* como esse são ótimos termômetros para sabermos em qual linguagem de programação devemos investir tempo de estudo, uma vez que a chance de nos depararmos com uma delas no mercado de trabalho será grande. Algumas das linguagens mais populares são:

- **linguagem Python:** é a linguagem mais popular dos últimos anos em praticamente todos os segmentos de desenvolvimento. É por esse motivo que adotaremos essa linguagem neste curso. Falaremos em detalhes sobre o Python na próxima seção;
- **linguagem Java:** a linguagem da Oracle é a mais popular atualmente no segmento de desenvolvimento para dispositivos móveis. O Java popularizou-se muito devido à inovadora forma de programar a que ele se propõe. Foi uma das primeiras linguagens que possibilitou o uso do mesmo código para compilar para qualquer plataforma, graças à proposta de virtualização de qualquer programa em Java utilizando uma máquina virtual pré-instalada. É orientada a objetos em seu cerne;
- **linguagem C:** talvez a mais antiga desta lista e que continua entre as mais utilizadas. A linguagem C, embora surgida na década de 1970, é muito usada até hoje e sofreu inúmeras atualizações com o passar do tempo. Considerada por muitos especialistas como linguagem de médio nível, devido à sua grande quantidade de detalhes, é ainda uma linguagem essencial,

especialmente para desenvolvedores na área de Engenharia e que trabalham com programação de *hardware* e dispositivos embarcados;

- **linguagem C++:** é uma linguagem derivada do próprio C. Porém, diferentemente dele, trabalha também com paradigmas de programação orientada a objetos;
- **linguagem C# (C Sharp):** é uma linguagem desenvolvida pela Microsoft como parte da plataforma .NET e deriva da linguagem C. É orientada a objetos e ganhou muito espaço devido às ótimas integrações com serviços Windows e suas bibliotecas de funções de fácil uso e versáteis;
- **linguagem Arduino:** certamente você já ouviu falar de Arduino.[9] Arduino é uma plataforma de desenvolvimento para eletrônicos muito popular, especialmente entre *hobbistas* e estudantes. O que talvez você não saiba é que a plataforma apresenta uma linguagem de programação própria, de mesmo, nome, bastante inspirada na linguagem C, porém, é mais simples e menos robusta. A linguagem Arduino é usada exclusivamente para a programação de *hardware* de Arduino;
- **linguagem HTML:** não é uma linguagem de programação, mas, sim, de marcação, e é o padrão utilizado na construção de páginas *web*. Não é uma linguagem compilada, ela é interpretada, e todos os navegadores são capazes de compreendê-la. O HTML evoluiu e se modernizou bastante desde o seu surgimento em 1991. Na versão mais recente, o HTML5, recursos foram implementados com o objetivo de eliminar *plugins* que antes era necessário instalar em navegadores, como o Flash (Adobe) e o Silverlight (Microsoft);
- **linguagem PHP:** é uma linguagem interpretada criada exclusivamente para desenvolvimento *web*. É uma linguagem muito empregada, pois apresenta uma ótima sinergia com HTML.

TEMA 5 – LINGUAGEM DE PROGRAMAÇÃO PYTHON

A linguagem escolhida para trabalharmos ao longo deste curso é o Python (visite a página: <https://www.python.org/>). Conforme vimos anteriormente, o Python é a linguagem mais empregada em quase todas as áreas de desenvolvimento, sendo fundamental que um programador nos dias de hoje tenha conhecimento dela. Porém, o que torna o Python tão popular, amado e utilizado? Vejamos suas principais características.

Primeiramente, é válido ressaltar que o Python é uma linguagem de propósito geral, ou seja, não foi criada pensando em um só setor de desenvolvido. O PHP, por exemplo, só é aplicável para programação *web*, e nada além disso. O Python pode ser empregado em qualquer área e em qualquer plataforma.

O que torna o Python tão versátil é a gama enorme de bibliotecas que existem para ele. Você quer desenvolver jogos com Python? Sem problemas, ele dá conta. Basta instalar as bibliotecas do *pygame* e aprender a usar. Desenvolvimento para dispositivos móveis, interfaces e até mesmo recursos de inteligência artificial e ciência de dados são facilmente localizados para uso no Python.

A linguagem caiu no gosto dos desenvolvedores por ser simples, fácil e intuitiva. Programadores iniciantes tendem a se dar muito bem com o Python, por exemplo. A linguagem é multiplataforma, ou seja, não é necessário adaptar seu código para rodar em sistemas diferentes. Um mesmo código deve ser capaz de funcionar tanto em ambientes Windows quanto Linux, ou até mesmo em dispositivos móveis, desde que compilados para cada plataforma, é claro.

A linguagem Python é orientada a objetos. Porém, não nos ateremos aos paradigmas de programação imperativa nesta disciplina, uma vez que o Python consegue trabalhar de maneira estruturada também.

O Python passa por atualizações constantes na linguagem para corrigir *bugs* e falhas de segurança, mas também para incluir novos recursos e funcionalidades. Atualmente, a linguagem Python está na versão 3. Ao entrar no *site* oficial da linguagem você encontrará todas as versões para *download* para diferentes plataformas (visite: <https://www.python.org/downloads/>).

Acerca da licença que rege a linguagem Python, conforme o *site* oficial, a linguagem Python é *Open Source*. O Python contém uma licença própria, mas compatível com a GPL.[10]

Por fim, como curiosidade, a linguagem Python contém inclusive uma espécie de filosofia da linguagem, chamada de *Zen do Python*. Esse mantra foi criado por um de seus programadores, Tim Peters, e é composto de 20 frases que, segundo Tim, todo programador deveria conhecer e seguir para se tornar um programador melhor. O zen pode ser encontrado no *site* oficial (disponível em: <https://www.python.org/dev/peps/pep-0020/>). A seguir, reproduzimos, em tradução livre feita pela Python Brasil (disponível em: <https://wiki.python.org.br/TheZenOfPython>), o mantra da linguagem por Tim Peters:

O Zen do Python, por Tim Peters

- Bonito é melhor que feio.
- Explícito é melhor que implícito.
- Simples é melhor que complexo.
- Complexo é melhor que complicado.
- Plano é melhor que aglomerado.
- Esparso é melhor que denso.
- Legibilidade faz diferença.
- Casos especiais não são especiais o bastante para quebrar as regras.
- Embora a praticidade vença a pureza.
- Erros nunca devem passar silenciosamente.
- A menos que sejam explicitamente silenciados.
- Diante da ambiguidade, recuse a tentação de adivinhar.
- Deve haver um – e preferencialmente só um – modo óbvio para fazer algo.
- Embora esse modo possa não ser óbvio à primeira vista a menos que você seja holandês.
- Agora é melhor que nunca.
- Embora nunca frequentemente seja melhor que *exatamente* agora.
- Se a implementação é difícil de explicar, é uma má ideia.
- Se a implementação é fácil de explicar, pode ser uma boa ideia.
- Namespaces são uma grande ideia – vamos fazer mais dessas!

5.1 HISTÓRIA DA LINGUAGEM PYTHON

Em 1982, em Amsterdã, no Centrum Wiskunde & Informatica (CWI), um programador de nome Guido van Rossum trabalhava no desenvolvimento de uma linguagem de programação chamada de ABC. Essa linguagem, embora não tenha se popularizado, até por carecer de recursos e funcionalidades, era bastante simples e intuitiva. A ABC se tornou uma inspiração para que von Rossum viesse a criar o conceito da linguagem Python nos anos subsequentes. Naquela época, a linguagem C dominava o mercado de desenvolvimento e, embora seja excelente, não é uma linguagem de fácil uso, especialmente para programadores iniciantes. Segundo ele, o Python veio como um contraponto ao C por ser simples e intuitivo.

O nome da linguagem, Python, não foi dado em homenagem às cobras popularmente conhecidas como píton aqui no Brasil, como muitas pessoas pensam. O nome Python foi dado por von Rossum para homenagear o programa de TV britânico chamado de *Monty Python Flying Circus*, transmitido entre 1969 e 1974, e que até hoje arranca bastante risada do público ao redor do mundo (Figura 10).

Figura 10 – Caixa do filme *Monty Python and the Holy Grail*. No Brasil, o título foi traduzido como *Monty Python: em busca do cálice sagrado*

Fonte: Peter Gudella/Shutterstock.

Apesar de tudo, ficou difícil desassociar o nome da linguagem do da cobra, uma vez que em inglês a grafia é a mesma. Assim, quando os livros técnicos da área começaram a ser lançados, eles estampavam muitas vezes uma cobra na capa. Com o passar do tempo, a cobra acabou sendo adotada como mascote da linguagem, e até o logotipo dela hoje são duas cobras conectadas, uma azul e outra amarela (Figura 11).

Figura 11 – Logotipo da linguagem Python

Fonte: cash1994/Shutterstock.

Por fim, é importante ressaltar que com o crescimento da linguagem Python ao redor do mundo, foi criada em 2001 a *Python Software Foundation*. Essa fundação sem fins lucrativos é responsável por manter a linguagem Python e torná-la tão importante no mercado nos dias atuais, pois está sempre em contato com os programadores, ouvindo-os e trazendo melhorias para as novas versões da linguagem. Empresas como Microsoft e Google são apenas alguns dos exemplos de grandes nomes que fazem parte dessa fundação, mostrando o impacto que a linguagem Python tem no mundo da tecnologia.

5.2 APLICAÇÕES DO PYTHON

O Python é conhecido como uma linguagem realmente bastante versátil e aplicável em diferentes campos da computação. Vejamos alguns exemplos do uso do Python.

A maioria dos sistemas operacionais nos dias atuais vem com suporte para desenvolvimento em Python pré-instalado. Não só isso, mas parte desses sistemas operacionais também foi desenvolvida em linguagem Python. Todas as destruições Linux e o macOS contêm. Infelizmente, o Windows não tem o Python por padrão no sistema, mas é bastante simples instalá-lo, conforme veremos no decorrer deste curso.

Talvez você já tenha ouvido falar do minicomputador mais famoso do mundo, o RaspBerry Pi (Figura 12). Esse *hardware* tem em suas entranhas o suporte ao Python e normalmente é a linguagem adotada para desenvolver para esse dispositivo.

Figura 12 – Foto de uma RaspBerry Pi 3 Model B+

Fonte: goodcat/Shutterstock.

Serviços *web* são muitos desenvolvidos em Python. O Google tem muitas de suas aplicações desenvolvidas nessa linguagem. O Youtube e o próprio buscador do Google são dois exemplos de peso.

No ramo do entretenimento, um ótimo exemplo de uso de Python está na Industrial Light and Magical (ILM), empresa do cineasta George Lucas e responsável pelos efeitos visuais de filmes como *Star Wars*, *Jurassic Park e Terminator 2*. A linguagem Python está por trás de todos seus *softwares* de renderização, animação etc.[11]

Por fim, jogos são também desenvolvidos, parcialmente ou totalmente, em Python. Dois exemplos implementados completamente em Python são *Sid Meier's Civilization IV* (Firaxis Games/Take Two Interactive, 2005) e *Battlefield 2* (Digital Illusions CE/EA Games, 2005).

FINALIZANDO

Nesta aula, aprendemos os alicerces de nossa disciplina. Vimos o que é lógica de programação e algoritmos, e que ambas as definições são a base para construirmos programas computacionais.

Compreendemos a arquitetura de *hardware* necessária para desenvolvermos e executarmos *softwares*, e que computadores modernos são construídos com base na máquina de von Neumann.

A definição de linguagem de programação e como um computador, que trabalha em linguagem de máquina, consegue compreender o que escrevemos em programas foi apresentado durante esta

aula.

Por fim, conhecemos diferentes linguagens de programação do mercado e nos atentamos à linguagem Python, a qual será adotada e utilizada ao longo de toda esta disciplina.

É importante ter em mente que todos os conceitos teóricos desta aula devem estar bem consolidados para que você possa começar a praticar a partir da Aula 2. Ressaltando uma das frases escritas por Tim Peters no *zen do Python*: "Agora é melhor do que nunca". Comece a estudar hoje, não deixe para depois. Programação é prática, e prática leva ao aprendizado.

REFERÊNCIAS

>AHO, A. V. et al. **Compiladores**: princípios, técnicas e ferramentas. 2. ed. São Paulo: Pearson, 2013.

ARDUINO. Disponível em: <https://www.arduino.cc/>. Acesso em: 11 mar. 2020.

DEITEL, P.; DEITEL, H. **Java**: como programar. 10. ed. São Paulo: Pearson, 2017.

Download the latest version of Python. **Python**, S.d.(a) Disponível em: <https://www.python.org/downloads>. Acesso em: 11 mar. 2020.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de programação**: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Pearson, 2005.

HISTORY and license. **Python**, S.d.(b) Disponível em: <https://docs.python.org/3/license.html#terms-and-conditions-for-accessing-or-otherwise-using-python>. Acesso em: 11 mar. 2020.

IEEE SPECTRUM. Disponível em: <https://spectrum.ieee.org/>. Acesso em: 10 mar. 2020.

LÓGICA. In: **Michaelis.** Disponível em: <https://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/l%C3%B3gica/>. Acesso em: 10 mar. 2020.

PEP 20: The Zen of Python. **Python**, 19 ago. 2004. Disponível em: <https://www.python.org/dev/peps/pep-0020/>. Acesso em: 11 mar. 2020.

PERKOVIC, L. **Introdução à computação usando Python**: um foco no desenvolvimento de aplicações. Rio de Janeiro: LTC, 2016.

PUGA, S.; RISSETTI, G. **Lógica de programação e estrutura de dados.** 3. ed. São Paulo: Pearson, 2016.

PYTHON. Disponível em: <https://www.python.org/>. Acesso em: 10 mar. 2020.

Python Success Stories. **Python**, S.d.(c) Disponível em: <https://www.python.org/about/success/ilm/>. Acesso em: 11 mar. 2020.

STALLINGS, W. **Arquitetura e organização de computadores.** 10. ed. São Paulo: Pearson, 2017.

TANENBAUM, Andrew. **Organização estruturada de computadores.** 5. ed. São Paulo: Pearson, 2013a.

_____. **Sistemas operacionais modernos.** 3. ed. São Paulo: Pearson, 2013b.

TOCCI, R.; WIDMER, N. S.; MOSS, G. L. **Sistemas digitais**: princípios e aplicações. 10. ed. São Paulo: Pearson, 2007.

---

[1] Nesta disciplina estamos tratando da computação clássica, a qual todos os computadores e dispositivos móveis atuais estão inseridos e para a qual foram projetados. Porém, saiba que na computação quântica, ainda em estágios iniciais de estudo, ambos os estados podem coexistir (princípio da incerteza), mas esse assunto não será abordado neste curso.

[2] Sugestão: faça uma busca *on-line* pelas palavras "*Leibniz calculator*" ou "calculadora de Leibniz", e você verá a máquina manual construída por ele no século XVII.

[3] O nível e o tipo de sinal usado para representar o *bit* dependerá da tecnologia de memória empregada no armazenamento. Por exemplo, memórias RAM convencionais trabalham com capacitores, em que um capacitor carregado representaria nível lógico alto.

[4] O assunto de sistemas operacionais é bastante complexo e requer uma disciplina inteira para ele. Caso tenha interesse em se aprofundar no assunto, recomendo a leitura complementar do livro *Sistemas operacionais modernos* (Tanenbaum, 2013b).

[5] Programação estruturada é um paradigma de programação. Esse conceito foi criado na década de 1950, e linguagens que trabalham de maneira estruturada se utilizam de recursos como estruturas de sequência, de tomadas de decisão e de repetição.

[6] As programadoras do ENIAC: Jean Jennings Bartik, Betty Holberton, Marlyn Wescoff, Kathleen McNulty, Ruth Teitelbaum e Frances Spence. Existe um documentário sobre a história dessas mulheres, chamado de *The invisible computers: the untold story of the ENIAC programmers*.

[7] Sobre paradigmas de programação orientada a objetos, recomendo o livro *Java: como programar* (Deitel; Deitel, 2017).

[8] Sobre compiladores e seu processo de desenvolvimento e compilação, recomendo o livro *Compiladores: princípios, técnicas e ferramentas* (Aho et al., 2013).

[9] *Site* do Arduino para tutoriais, projetos e exemplos: <https://www.arduino.cc/>.

[10] A licença do Python 3 pode ser encontrada em:

<https://docs.python.org/3/license.html#terms-and-conditions-for-accessing-or-otherwise-using-python>.

[11]

Para saber um pouco mais sobre a história da ILM com o Python, acesse: <https://www.python.org/about/success/ilm/> (link em inglês).