

INTRODUÇÃO À PROGRAMAÇÃO

Conceitos básicos de programação e criação de programas simples de computador.

HOMENAGEM DA APPLE AOS PROGRAMADORES

Vídeo veiculado no Keynote do evento WWDC de 2016
homenageando os programadores de todo o mundo.

PAREM DE DIZER QUE APRENDER A PROGRAMAR É FÁCIL!

Artigo publicado originalmente por Scott Hanselman,
Engenheiro de Software da Microsoft em Junho de 2016

[Link](#)

Por que programar?

Software está em tudo.

Programação melhora
o mundo.

Atividade básica do
futuro.

*“Everybody in this country should
learn how to program a computer...
because it teaches you how to think.”*

- Steve Jobs



O que é programar?

Programar é dar instruções aos computadores.

Computadores são entidades com as quais conseguimos nos comunicar através de linguagens próprias, pautadas na matemática e na lógica computacional.

Computadores são máquinas digitais de repetições de programas, que surgem a partir de padrões de ações que precisamos executar.

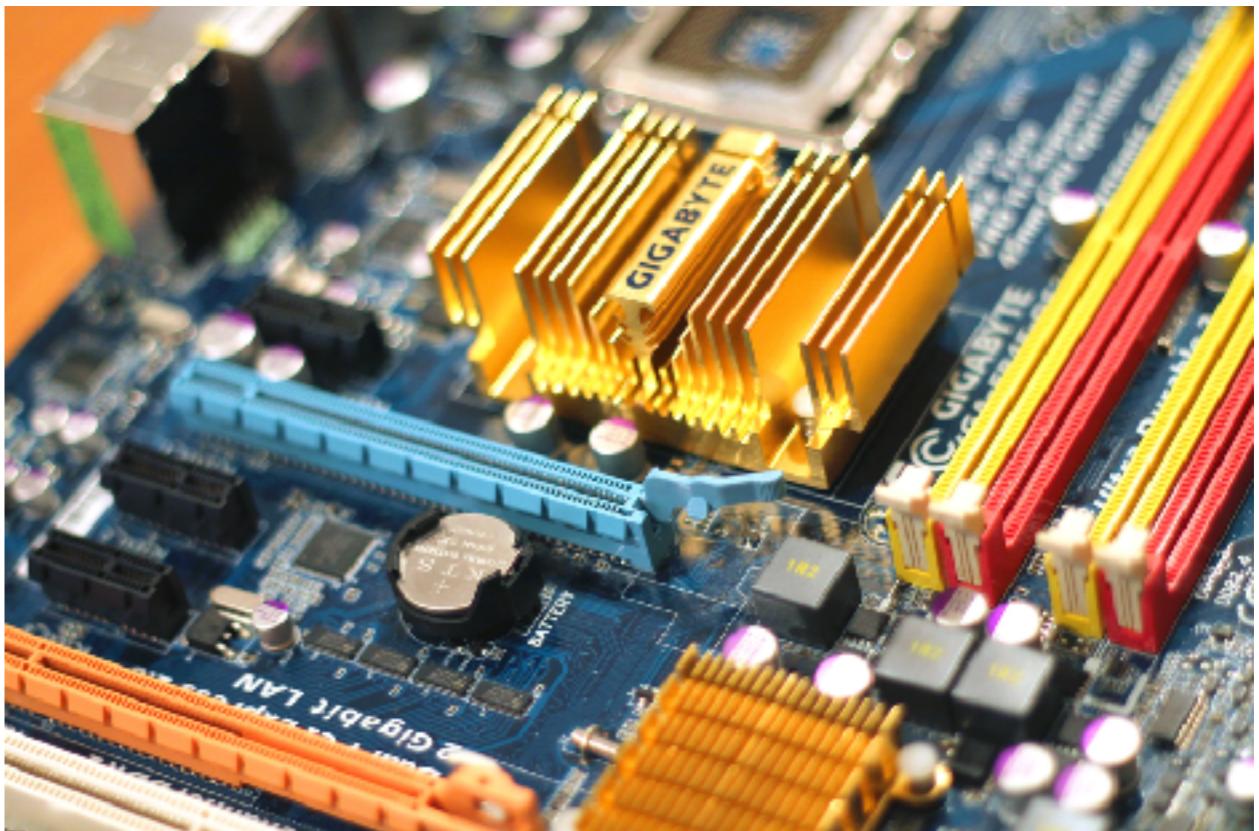
Computadores, em essência, são burros!

COMO FUNCIONA UM COMPUTADOR?



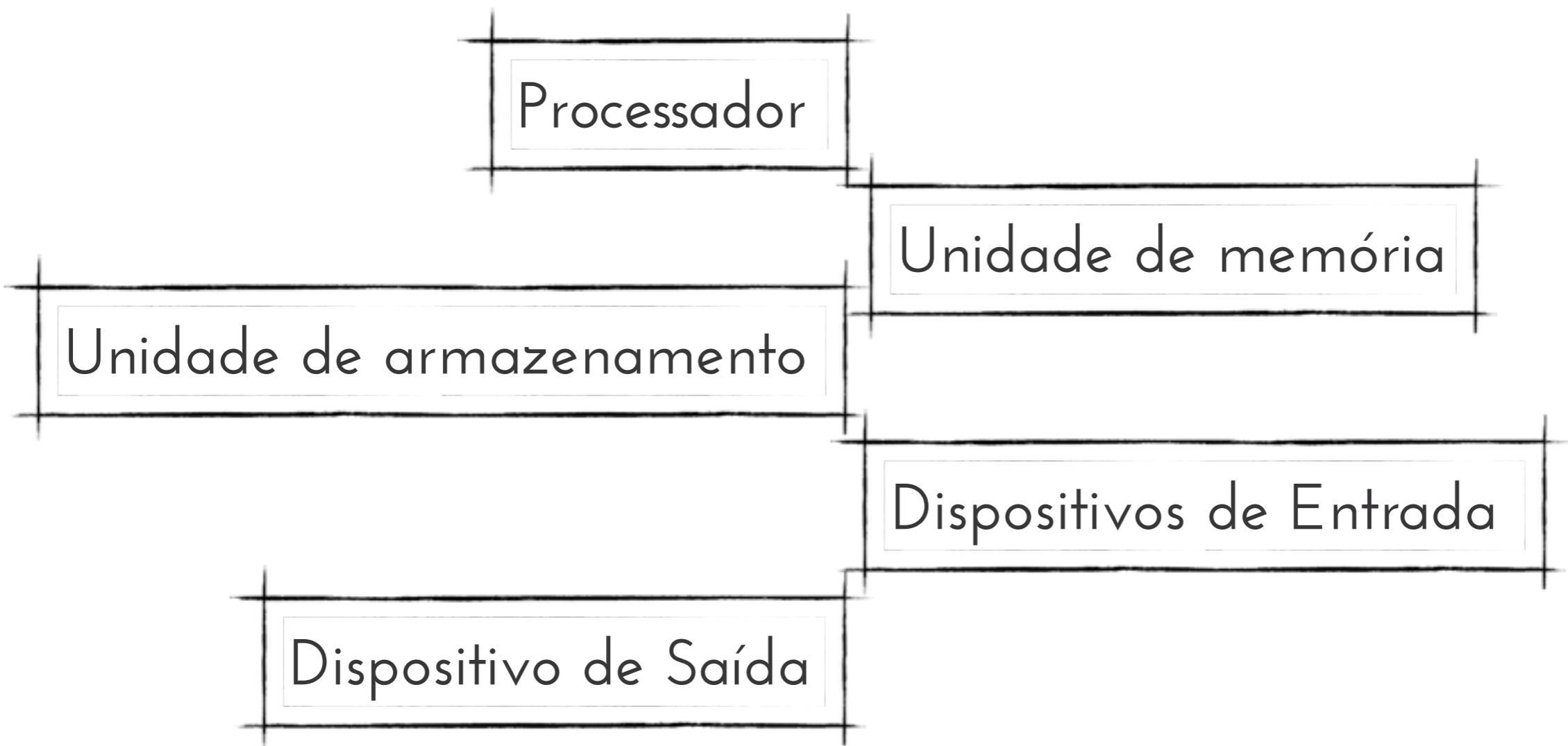
Como funciona um computador?

O computador é em essência uma máquina de fazer cálculos. A forma como os computadores modernos funciona em essência não difere de quando eles foram criados. A grande evolução da computação foi a possibilidade de criar circuitos e chips capazes de executar mais operações.



Componentes Básicos

Todo sistema computacional é composto, em base, pelos seguintes componentes:



Componentes Básicos

Processador

É o cérebro do computador, onde todas as operações acontecem.

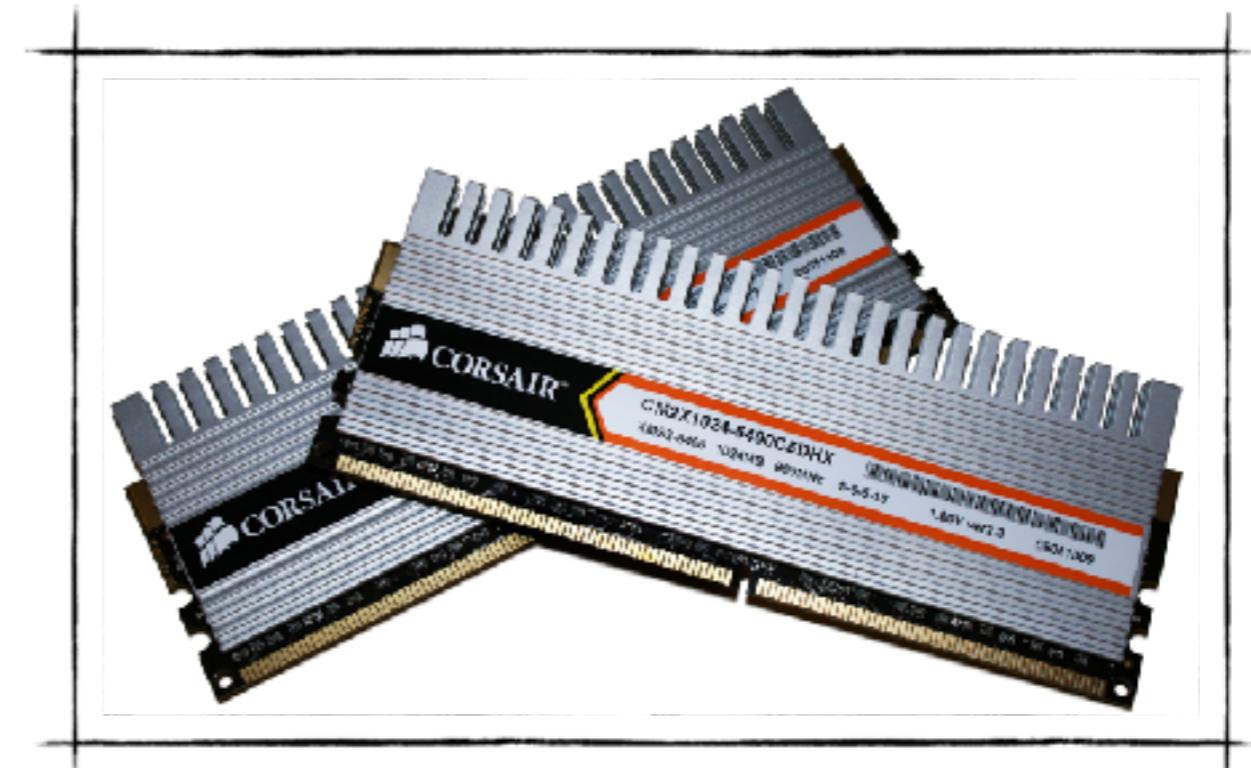
Processadores modernos chegam a processar bilhões de operações por segundo.



Componentes Básicos

Unidade de Memória

É o espaço de trabalho do processador. Podemos pensar na memória como se fosse um quadro branco onde os programas em execução vão anotando as informações necessárias para o processador efetuar suas operações, bem como onde o processador armazena o



Componentes Básicos



Unidade de armazenamento

É o espaço onde os dados são gravadas de maneira permanente.

Componentes Básicos

Dispositivos de entrada

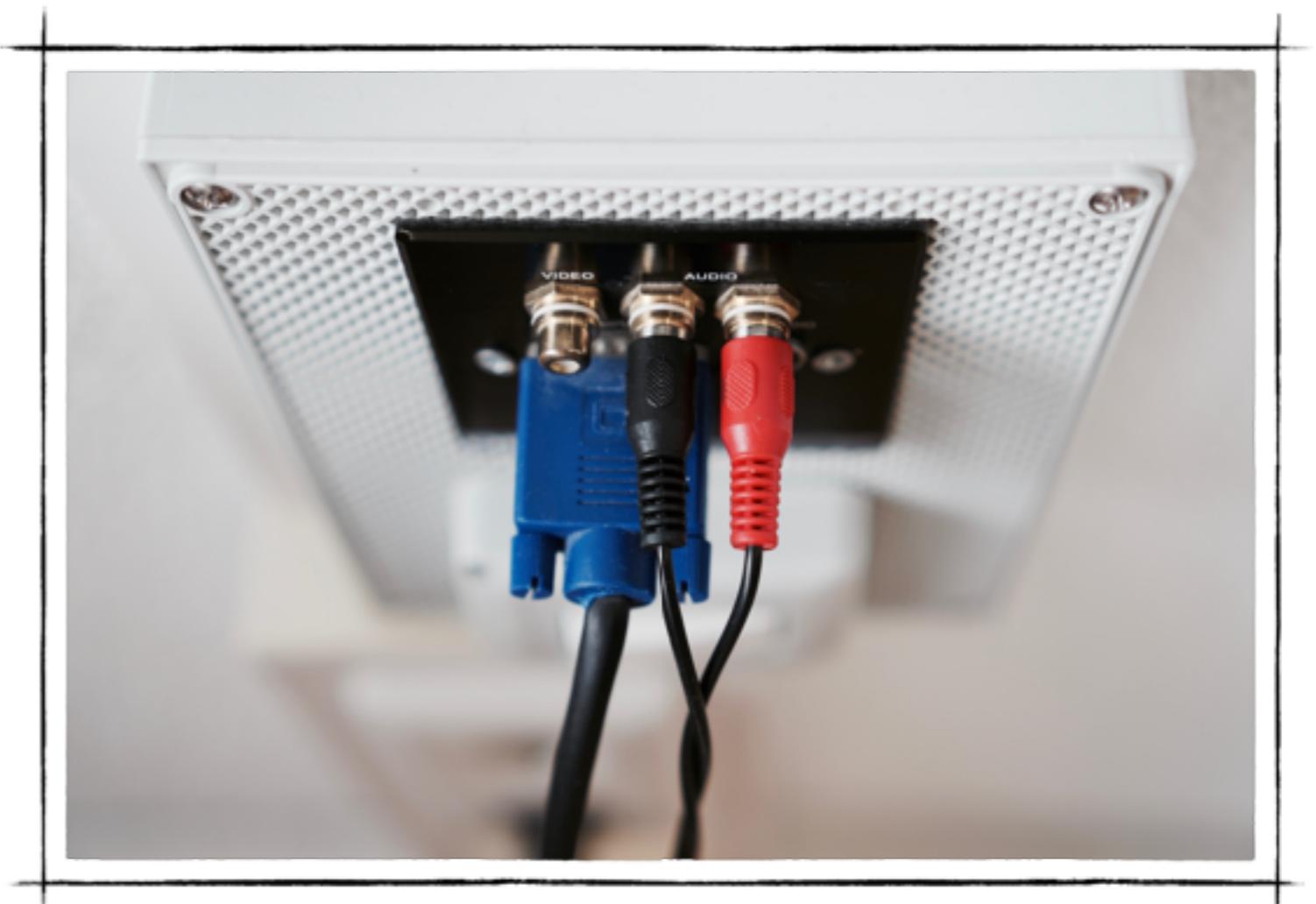
Historicamente eram os teclados e mouses. Hoje em dia, temos diversas formas de interagir com os computadores, desde telas sensíveis a toques e controles remotos, até diversos tipos de sensores que coletam informações que são processadas pelos programas.



Componentes Básicos

Dispositivos de saída

É todo dispositivo que produza algum tipo de saída, desde telas e caixas de som, até impressoras e dispositivos tátteis.



LINGUAGENS DE PROGRAMAÇÃO



66

As linguagens de programação são a forma que criamos para nos comunicarmos com nossos computadores e não receberam esse nome por acaso! Tal como as línguas humanas faladas ou escritas elas empregam regras de ortografia e sintaxe, que devem ser estritamente seguidas caso desejemos nos fazer claro para o nosso interlocutor: o computador!



66

Mas nunca podemos nos esquecer que essa conversa não é um monólogo, mas sim um diálogo que acontece entre você, o computador e todos os outros seres humanos que participam daquele projeto.

Quando pessoas participam de um intercâmbio é muito comum que elas encontrem no outro país pessoas de diversas regiões e culturas diferentes que para se comunicar fazem uso de uma língua comum (em geral o inglês). ■ ■

Nem todos são proficientes naquele idioma mas todos se esforçam para manter um bom nível de contato e isso só é possível graças aos padrões que essas línguas foram integrando ao longo do tempo. Um bom programador não encara sua linguagem de programação como um simples instrumento, ele tem respeito pelas estruturas e boas práticas daquela língua, tal como um acadêmico pelo idioma que emprega na escrita de um artigo científico.



66

Escreva seu código como se estivesse escrevendo uma monografia e não como se estivesse conversando no WhatsApp. Seus pares agradecerão.

Linguagens de programação são tão vivas quanto os idiomas falados, mudando frequentemente, empregando novas gírias e sotaques que surgem nos diversos meios aonde são utilizadas.



66

Esteja sempre atento ao que acontece com o seu meio e aberto as novidades, pois isso tenderá sempre a tornar seu trabalho mais produtivo.

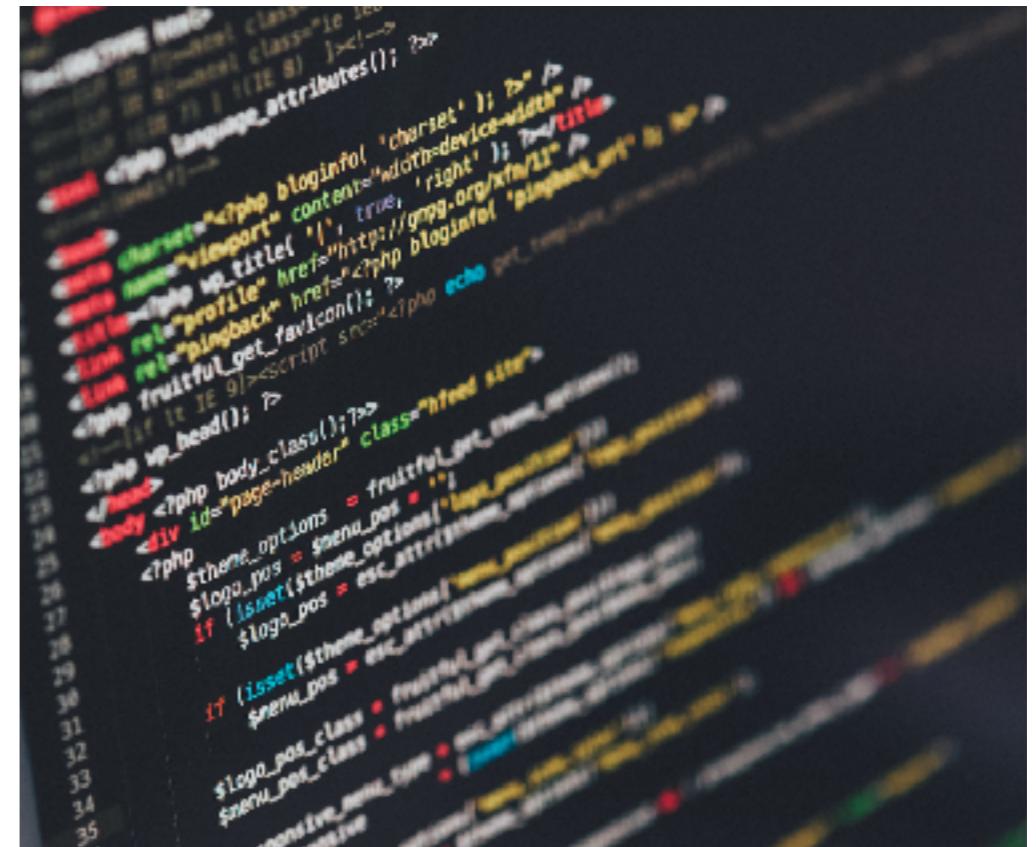


Linguagens de Programação

Existem literalmente centenas de linguagens de programação que foram criadas ao longo da história da computação.

Todos os anos novas linguagens nascem e outras caem em desuso e acabam morrendo.

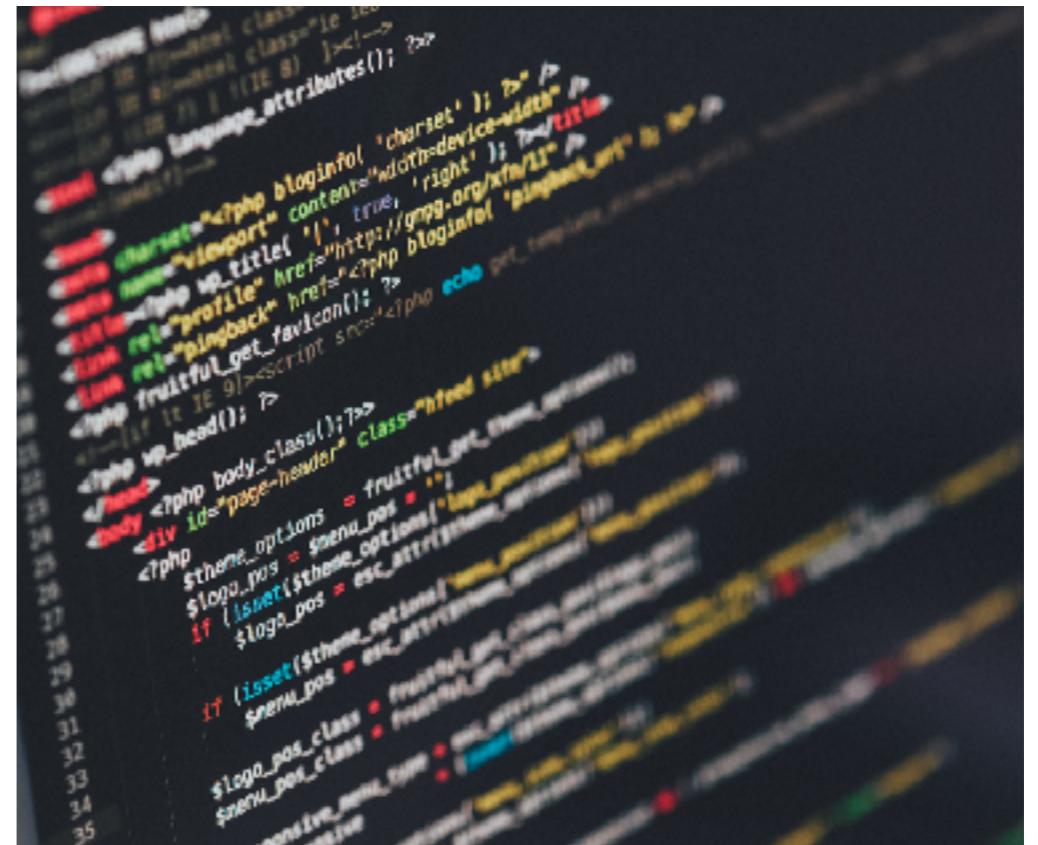
A Engenharia e as Ciências da Computação são entidades vivas e muito dinâmicas, sendo constantemente evoluídas e repensadas.



Linguagens de Programação

Linguagens de Programação não raramente nascem nos meios acadêmicos e começam a ganhar força nos mercados corporativos, onde são colocadas a prova.

Algumas delas surgem para atender nichos e necessidades muito específicas, outras se propõem a atender propósitos gerais.



TIPOS DE LINGUAGEM DE PROGRAMAÇÃO

Linguagens Lineares ou Procedurais

Linguagens Orientadas a Objetos

Linguagens de Domínio Específico

O processo de Compilação

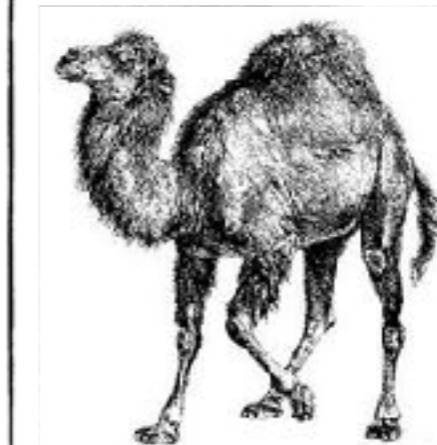
Linguagens que vamos estudar

Linguagens Lineares ou Procedurais

Os programas são estruturados como procedimentos ou funções, contidos em bibliotecas e consumidos de maneira linear em um programa.

Exemplos: C, Basic, Perl, Fortran, etc.

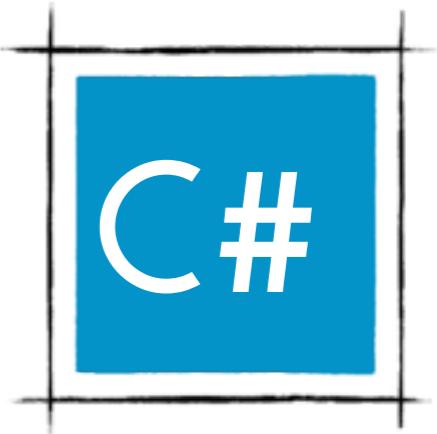
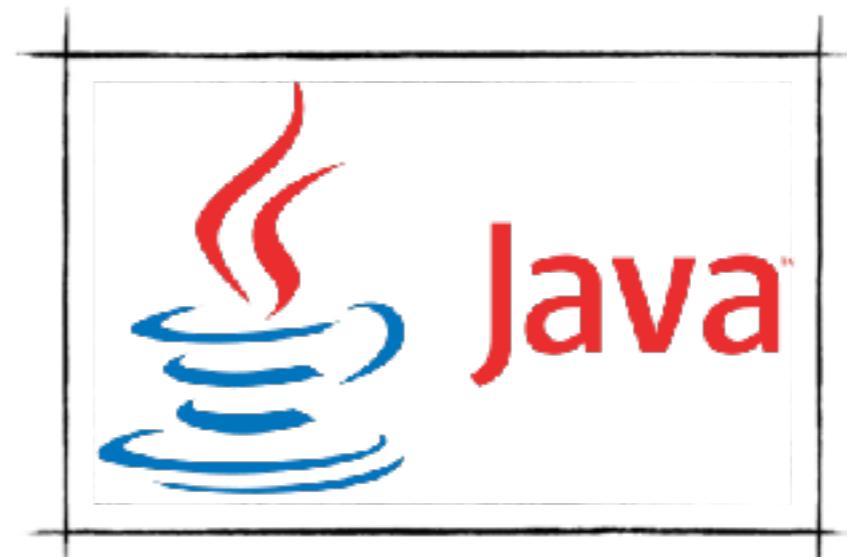
Fortran



Linguagens Orientadas a Objetos

Os programas são estruturados como classes e objetos que se relacionam entre si, oferecendo uma forma de se pensar mais próxima do mundo real.

Exemplos: Java, C#, Swift, Objective-C, Smalltalk, etc.



Linguagens de Domínio Específico



São linguagens especializadas em resolver problemas específicos dentro de uma aplicação ou plataforma.

Exemplos: linguagens de Macros do Excel (VBScript), SQL, CSS, etc.

Tipos de Linguagem de programação

As Ciências da Computação catalogaram as linguagens de programação em centenas de categorias e, em muitos casos, uma linguagem se enquadra em mais de uma.

[Link](#)

O Processo de Compilação

Traduz os códigos escritos em uma linguagem de programação em uma codificação que é compreendida pelos compiladores.

Compõe os diversos arquivos e recursos que constituem um programa em um arquivo digital que pode ser executado dentro de uma plataforma.

É o intermediário entre as linguagens de médio e alto nível que utilizamos e o nível mais baixo em que o computador funciona.

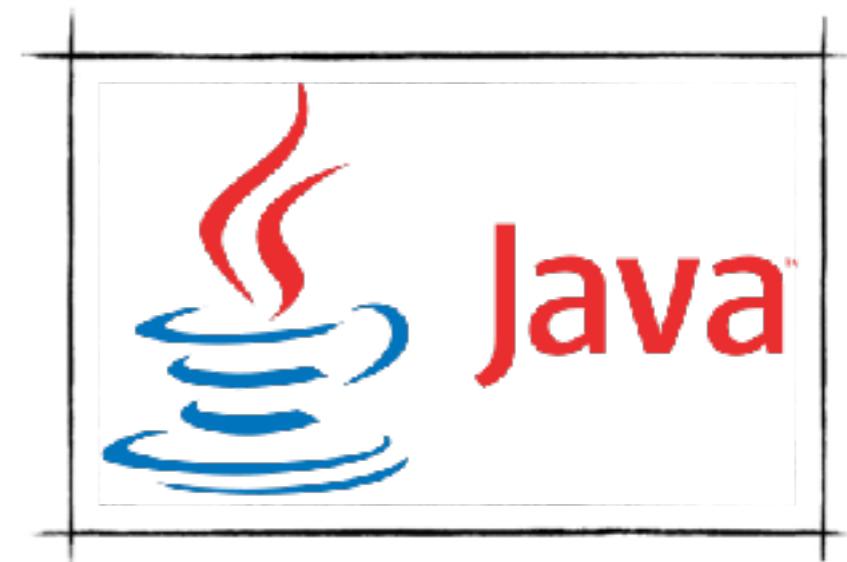
Linguagens que vamos estudar

Essa disciplina tem o objetivo de introduzir os **conceitos básicos de programação** com foco nas linguagens que estudaremos ao longo desse curso. Contudo, o bom entendimento dessas linguagens facilitará o aprendizado de outras que se façam necessárias ao longo sua jornada.

Vamos conhecer um pouco sobre as linguagens que nos acompanharão daqui por diante!



Swift



Java



O Java foi criado pela Sun Microsystems e lançado em 1995 como plataforma computacional onde seria possível criar programas uma única vez que pudessem executar em diferentes computadores e sistemas operacionais.

Java

A linguagem Java recebe o mesmo nome da plataforma e tem as seguintes características:

Inclui um conjunto rico de bibliotecas base para criação de programas chamadas de **Padrão Java**.

É fortemente tipada (strongly-typed), o que a torna mais previsível e segura.

Emprega o mecanismo de coleta de lixo (garbage collection) para gerenciamento automático de memória, que facilita a programação.



Java e Android

O Java foi escolhido como linguagem para criação de Apps para Android.

Não há resposta oficial do Google sobre a escolha da linguagem, mas acredita-se que seja devido a sua popularidade.

Na ocasião, o Java já era uma plataforma madura, bastante presente na comunidade Linux e com uma extensa comunidade Open Source.



Swift

A linguagem Swift foi criada pela Apple e lançada em 2014 em seu evento anual para desenvolvedores, o WWDC. O objetivo era criar uma linguagem com características mais que mantivesse as qualidades do Objective-C, que era usado anteriormente para desenvolvimento em sua plataforma.



Swift

O Swift conta com as seguintes características:

Moderna e segura

Orientada a Objetos,
mas com características
de linguagens funcionais.

Sintaxe simplificada; privilegia
a digitação mas sem perder a
legibilidade.



Swift



Em dezembro de 2015 a Apple abriu o código do Swift, passando a aceitar a colaboração de outros programadores que quisessem participar de seu desenvolvimento, bem como possibilitando que ela seja portada para outras plataformas.

Swift



Atualmente o Swift está disponível para Linux e há dezenas de projetos que permitem utiliza-la em outros tipos de aplicação fora dos sistemas Apple.

Há rumores de que o Google vem estudando substituir o Java pelo Swift no Android, embora isso não tenha se confirmado.

Swift



A 3^a versão do Swift foi lançada recentemente junto com o iOS 10, trazendo dezenas de novidades e transformando-a numa linguagem mais madura e produtiva.

Esse é o melhor momento para aprender Swift, há um potencial de mercado enorme na linguagem!

DEMO: UM PRIMEIRO PROGRAMA

Nessa demonstração vamos usar a ferramenta **Swift Playgrounds** do Xcode para escrever nosso primeiro programa em linguagem Swift!

CONCEITOS BÁSICOS



SINTAXE

Sintaxe

São regras que devem ser empregadas para escrita de programas em uma determinada, tal como as regras sintáticas, ortográficas e gramaticais das linguagens “humanas”.

Cada uma das centenas de linguagens de programação existentes tem suas próprias regras, portanto sua própria sintaxe.

Muitas linguagens tem antecessores comuns e não raramente é fácil identificar elementos de sintaxe entre as diversas linguagens.



Sintaxe

Exemplos:

Um programa básico para somar dois números escritos nas duas linguagens que estudaremos:

```
// Importa os pacotes necessários para o programa
import UIKit

let n1: Int = 10          // Declaração de uma constante com tipo explícito
let n2 = 20                // Declaração de uma constante com inferência de tipo

// Declaração de uma variável que recebe o resultado da operação de soma
var result = (n1 + n2)

// Imprime o resultado no terminal
print("O resultado da soma de \(n1) com \(n2) é \(result)")
```



Sintaxe

Exemplos:

Um programa básico para somar dois números escritos nas duas linguagens que estudaremos:

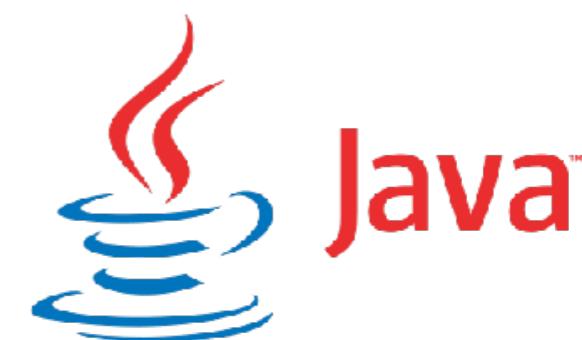
```
/*
 * Declara o pacote ao qual esse programa pertence
 */
package sumsample;

/**
 * A classe com o ponto de entrada do programa.
 */
public class SumSample {

    public static void main(String[] args) {
        // Declaração das variáveis que serão usadas
        int n1 = 10;
        int n2 = 20;

        // Declaração de uma variável recebendo o resultado da operação
        int result = (n1 + n2);

        // Imprime o resultado no terminal
        System.out.printf("O resultado da soma de %d com %d é %d.", n1, n2, result);
    }
}
```



VARIÁVEIS

Tipos de Dados
Estruturas de Dados
Alocação de Variáveis e Memória
Exercício
Operações e Operadores

Variáveis

Armazenam as informações que utilizamos em nosso programa.

São a estrutura básica que permite versatilidade em nossos programas.



Variáveis

São usadas para:

- Receber informações de entrada de um programa;
- Armazenar as informações de saída de um programa;
- Carregar informações armazenadas no computador em alguma fonte de dados como arquivo ou um banco de dados;
- Armazenar informações temporárias no processamento de um programa.

Variáveis

Em geral são especializadas em armazenar um tipo específico de informação.

São identificadas por nomes dentro dos programas.

Cada linguagem tem suas próprias regras para criação desses identificadores.

Tipos de Dados

São os tipos de informações que podemos armazenar em variáveis.

Em geral nos referimos a eles como **tipos**, um vocabulário comum em programação.



Tipos de Dados

Os ***tipos básicos*** ou ***primitivos*** estão presentes em praticamente todas as linguagens modernas:

Inteiros

Ponto Flutuante/Decimais

Caracteres

Tipos de Dados

Os **tipos complexos**, como o nome diz, representam informações mais complexas e relevantes para uso no mundo real, e são construídas pela composição de tipos básicos.

Cada linguagem implementa seus próprios **tipos complexos**, embora alguns sejam comuns a maioria:

Strings

Datas e Calendários

Listas (Arrays ou Vetores)



Estrutura de Dados

A grande maioria das linguagens modernas oferece meios de criarmos **Estruturas de Dados**, que é a criação de tipos de dados compostos para representar um tipo específico de informação.



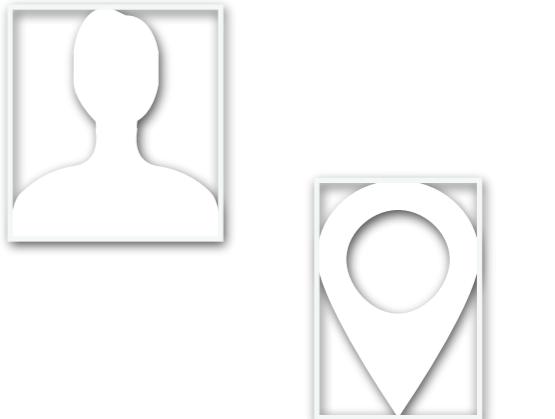
Estrutura de Dados

Exemplos:

A representação das informações do cadastro de um cliente (nome, CPF, data de nascimento, email, etc);



Dados de endereço e geográficos (endereço, cidade, estado, cep, latitude, longitude, etc.);



Um post em alguma rede social (conteúdo do post, data e horário, público, etc.).



Alocação de Variáveis e Memória

A memória do computador é o espaço que utilizamos para armazenar o conteúdo das variáveis.

Ao declarar uma variável, o computador aloca um espaço dentro da memória capaz de armazenar os valores possíveis para aquele tipo de informação.

A unidade base do armazenamento é o **bit**, um registro que pode conter um binário (0 ou 1, verdadeiro ou falso, ligado ou desligado, etc.)

Alocação de Variáveis e Memória

A unidade básica para medição do armazenamento é o **Byte**, um registro de 8 bits, que pode armazenar um número decimal entre 0 e 255.

As demais unidades de medida para dados são ordens de grande sobre os **Bytes**, conforme a tabela ao lado:

Unidade	Sigla	Tamanho
Byte	N/A	8 bits
Kilobyte	KB	1024 bytes
Megabyte	MB	1024 Kilobytes
Gigabyte	GB	1024 Megabytes
Terabyte	TB	1024 Gigabytes
Petabyte	PB	1024 Terabytes
Exabyte	EB	1024 Petabytes

Referência



Alocação de Variáveis e Memória

Diferentes tipos tem capacidades distintas de armazenamento que tem relação direta com o espaço em memória que é utilizado.

Exemplo

Os diversos tipos de dados numéricos suportados pela maioria das linguagens:

Tipo	Tamanho	Faixa de Valores
byte	8 bits	-128 a 127
short int	16 bits	-32.768 a 32.767
int	32 bits	-2.147.483.648 a 2.147.483.647
unsigned int	32 bits	0 a 4.294.967.295
long	64 bits	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

EXERCÍCIOS

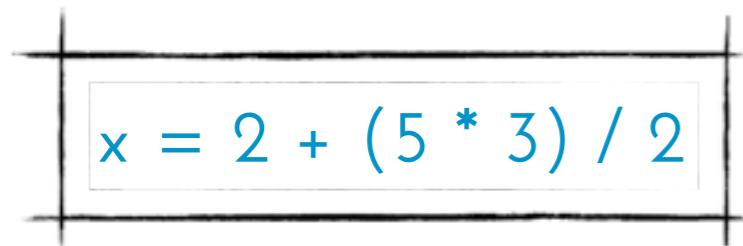
Declarando Variáveis

Operações e Operadores

Operações aritméticas e lógicas estão presentes em praticamente todas as linguagens.

São as operações básicas que o processador do computador consegue executar.

Os operadores são usados para criar expressões que geram um resultado que pode ser atribuído a variáveis, por exemplo:



```
x = 2 + (5 * 3) / 2
```

A variável **x** receberá o resultado da expressão *2 mais 5 vezes 3 dividido por 2*, chamado de **Atribuição**.

Operadores Aritméticos

Realizam operações aritméticas sobre valores ou variáveis numéricas.

Por padrão, a maior parte das linguagens suporta a maior parte das operações básicas: soma, subtração, multiplicação, divisão e resto.

Outras operações (exponenciação, raiz quadrada, fatorial, etc), estão disponíveis, em geral, através de bibliotecas de funções matemáticas.



Operadores Aritméticos

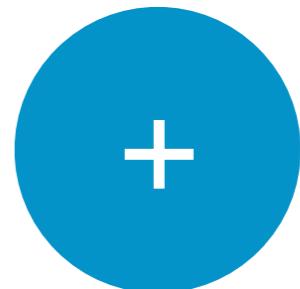
Lista de operadores
aritméticos comuns

Ex: $1 + 2$

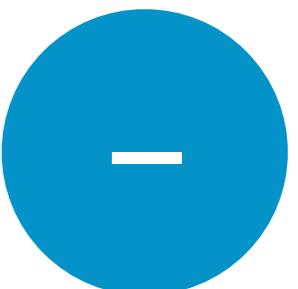
Módulo/ Resto



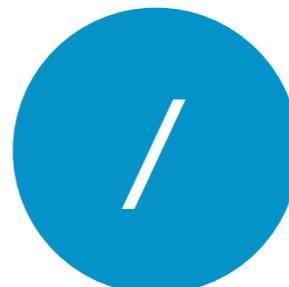
Adição/ Soma



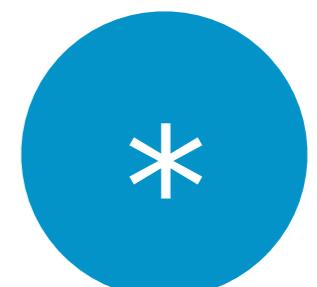
Subtração



Divisão



Multiplicação

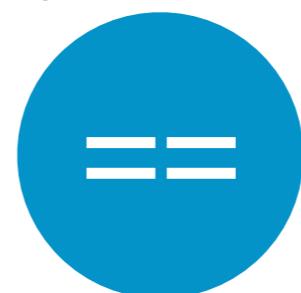


Operadores de Comparações

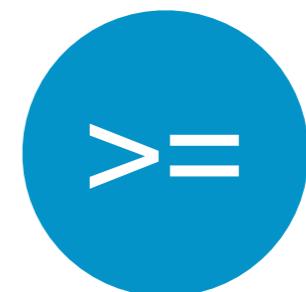
São usados para comparar dois valores numéricos nos critérios de igualdade, maior ou menor que.

Retornam um resultado booleano (verdadeiro ou falso).

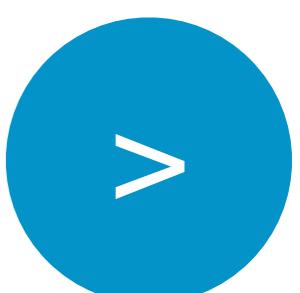
Igualdade



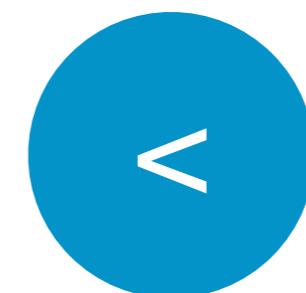
Maior ou igual



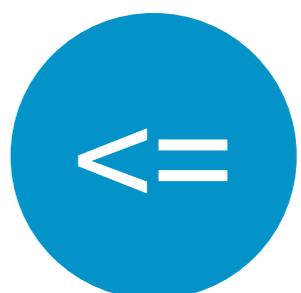
Maior que



Menor que



Menor ou igual



**Lista de operadores
de comparação**

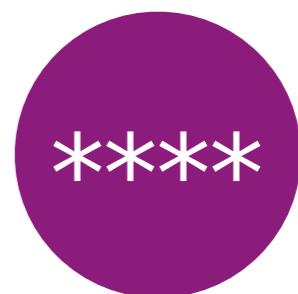
Operadores Lógicos

São usados para criar uma expressão composta por um ou mais resultados booleanos (como o resultado de uma operação de comparação).

E Lógico (AND)



OU Lógico (OR)



NÃO Lógico (NOT)



**Lista de operadores
lógicos comuns**



Operadores de Comparação e Lógicos produzem resultados booleanos, que são a base das expressões condicionais e criam as condições para que nossos programas possam tomar decisões.

EXERCÍCIO

Trabalhando com Operadores

ESTRUTURAS BÁSICAS DE PROGRAMAÇÃO

Construções Condicionais
Estruturas de Repetição

Construções Condicionais

Possibilitam verificar condições lógicas a fim de determinar o fluxo de um programa.

*Switch
Case
...
If...Else*

66

Fluxo do Programa: é o caminho que ele percorre durante sua execução. Conforme o algorítmico que estamos empregando para resolver um problema computacional, ele pode tomar direções diferentes de acordo com os dados de entrada.

A capacidade de trabalhar o fluxo do programa através de expressões condicionais é a pedra fundamental da programação.

IF...ELSE (Instrução SE...SENÃO)

If...Else

É a construção básica para criar condições em nossos programas.

Usam expressões booleanas (que retornam verdadeiro ou falso) para determinar se uma condição deve ou não ser executada.

Podem incluir diversas condições em uma única instrução.

IF...ELSE (Instrução SE...SENÃO)



Sintaxe do **if** em Swift

```
if [expressão booleana] {  
    // bloco de código a ser executado  
} else if [condição booleana] {  
    // bloco a ser executado caso a segunda condição seja satisfeita  
} else {  
    // bloco a ser executado se nenhuma das condições anteriores for satisfeita  
}
```

IF...ELSE (Instrução SE...SENÃO)

Considerações sobre o **if** em Swift:

O uso de parênteses nas expressões booleanas é permitido e opcional;

Condições **else if** e **else** são opcionais.

É possível incluir quantas condições **else if** forem necessárias



Exemplo de uso

```
// Determinar faixa etária
var idade = 19
if (idade < 12) {
    print("Criança")
} else if (idade < 18) {
    print("Adolescente")
} else if (idade < 50) {
    print("Adulto")
} else if (idade < 70) {
    print("Meia-idade")
} else {
    print("Idoso")
}
```

IF...ELSE (Instrução SE...SENÃO)



Sintaxe do **if** Java

```
if ([expressão booleana]) {  
    // bloco de código a ser executado  
} else if ([condição booleana]) {  
    // bloco a ser executado  
} else {  
    // bloco a ser executado se nenhuma das condições anteriores for satisfeita
```

IF...ELSE (Instrução SE...SENÃO)

Considerações sobre o **if** em Java:

O uso de parênteses nas expressões booleanas é obrigatório.

Condições **else if** e **else** são opcionais.

É possível incluir quantas condições **else if** forem necessários

É possível omitir as chaves do bloco de código de uma condição **({ e })** caso uma única instrução seja executada. Nesse caso ela pode inclusive ser colocada em linha com o if.

IF...ELSE (Instrução SE...SENÃO)



Exemplo de uso

```
// Como está o clima
int temperature = 27;
if (temperature < 0) {
    System.out.print("Congelando!!!");
} else if (temperature >= 0 && temperature < 12)
    System.out.print("Friozinho..");
else if (temperature >= 12 && temperature < 21)
    System.out.print("Clima fresco!");
else if (temperature >= 21 && temperature < 30)
    System.out.printf("Tempo quente!");
else {
    System.out.print("Quente demais!!!");
}
```

Instrução SWITCH...CASE

*Switch ...
Case*

Oferece uma forma simplificada de verificar quando uma entrada pode assumir um número pré-determinado de valores, que em cada caso ocasionarão na execução de diferentes instruções.

Instrução SWITCH...CASE



Sintaxe do **switch...case** em Swift

```
switch [algum valor] {  
    case [valor 1]:  
        [comandos do valor 1]  
    case [valor 2], [valor 3]:  
        [comandos do valor 2 ou 3]  
    default:  
        [caso nenhuma das condições anteriores seja satisfeita]  
}
```

Instrução SWITCH...CASE

Considerações sobre o **switch...case** em Swift:

O switch do Swift é extremamente versátil. Além de comparar valores é possível criar expressões mais complexas de comparação (consulte documentação).

O switch é exaustivo, o que significa que precisamos considerar todas as alternativas possíveis, ou usar um caso *default* para tratar as situações onde o valor não se aplica a nenhum caso previsto.

Instrução SWITCH...CASE



Exemplo de uso

```
// Como esta o clima em Swift com case
var temperature = -2

switch temperature {
case let x where x < 0:
    print("Congelando!!! ❄️")
case 0..<12:
    print("Frio... ")
case 12..<21:
    print("Clima fresco!")
case 21..<30:
    print("Tempo quente!")
default:
    print("Quente demais!!! 🔥")
}
```

Instrução SWITCH...CASE



Sintaxe do ***switch...case*** em Java

```
switch ([algum valor]) {  
    case [valor 1]:  
        [comandos do valor 1];  
        break;  
    case [valor 2]:  
        [comandos do valor 2];  
        break;  
    default:  
        [caso nenhuma das condições anteriores seja satisfeita];  
        break;  
}
```

Instrução SWITCH...CASE

Considerações sobre o **switch...case** em Java:

O switch...case do Java somente pode ser usado com valores inteiros e enumerações.

Ele não é exaustivo, portanto não é necessário prever todas as condições.

É obrigatório que ao final de cada instrução seja incluído a palavra reservada **break**.

O caso default é opcional.

Instrução SWITCH...CASE



Exemplo de uso

```
// Determinando o mês do ano
int month = 3;
switch (month) {
    case 1:
        System.out.print("Janeiro");
        break;
    case 2:
        System.out.print("Fevereiro");
        break;
    // Demais meses ...
    case 12:
        System.out.print("Dezembro");
        break;
    default:
        System.out.print("Mês inválido!");
        break;
}
```

ESTRUTURAS DE REPETIÇÃO

Estruturas de Repetição

Permitem realizar uma operação ou um conjunto de operações repetidas vezes conforme critérios.

São fundamentais para criação de algoritmos avançados, especialmente os que envolvem operações com listas.

*Laço WHILE
e DO...WHILE*

Laço FOR

Laço FOR

Laço FOR

São usados quando temos um número pré-definido de repetições que temos de executar.



Laço FOR



Sintaxe do **for** em Swift

```
for [variavel] in [numero inicial]...[numero final] {  
    [Bloco de Código a ser repetido]  
}
```

Laço FOR

Considerações sobre o **for** em Swift:

A instrução [numero inicial]...[numero final] é chamada de Closed Range Operator (operador de intervalo fechado), pois começa com o número inicial incluindo o final.

É possível usar também a instrução [numero inicial]..`\<`[numero final], chamada de **Half-Open Range Operator** (operador de intervalo semi-aberto), que inclui os números do inicial até um número antes do final.

Laço FOR



Exemplo de uso

```
// Imprimir tabuadas
for tabuada in 2...9 {
    print("Tabuada do \(tabuada):")
    for numero in 1...10 {
        var resultado = tabuada * numero;
        print("\t\(tabuada) x \(numero) = \(resultado)")
    }
}
```

Laço FOR



Sintaxe do **for** em Java

```
for ([inicializador]; [critério]; [incremento]) {  
    [Bloco de Código a ser repetido]  
}
```

Laço FOR

Considerações sobre o **for** em Java

É possível omitir as chaves **({ e })** caso o laço execute uma única instrução.

É possível incluir mais de uma variável no inicializador e no incremento.

É possível omitir o inicializador e o incremento se não forem necessários.

Laço FOR



Exemplo de uso

```
// Imprimir tabuadas
for (int tabuada = 2; tabuada <= 9; tabuada++) {
    System.out.printf("Tabuada do %d\n", tabuada);
    for (int numero = 1; numero <= 10; numero++) {
        int resultado = numero * tabuada;
        System.out.printf("\t%d x %d = %d\n", tabuada, numero, resultado);
    }
}
```

Laço WHILE e DO...WHILE

*Laço WHILE e
DO...WHILE*

São usados quando precisamos repetir uma operação enquanto (while) uma condição for verdadeira.

Laço WHILE e DO...WHILE



Sintaxe do **While** em Swift

```
while ([condição]) {  
    [bloco de código]  
}
```

Laço WHILE e DO...WHILE

Considerações sobre o **while** em Swift:

É possível usar a variação **repeat...while** para forçar a execução do loop pelo menos uma vez antes da condição ser verificada.

Laço WHILE e DO...WHILE



Exemplo de uso

```
// Calculo do Fatorial
let fatorial = 5
var numero = fatorial
var resultado: Int = 1

while (numero > 0) {
    resultado = resultado * numero
    numero = numero - 1
}
print("\(fatorial)! = \(resultado)")
```

Laço WHILE e DO...WHILE



Sintaxe do **While** em Java

```
while ([condição]) {  
    [bloco de código]  
}
```

Laço WHILE e DO...WHILE

Considerações sobre o **while** em Swift:

É possível omitir as chaves **({ e })** caso o laço execute uma única instrução.

É possível usar a variação **do...while** para forçar a execução do loop pelo menos uma vez antes da condição ser verificada.

Laço WHILE e DO...WHILE



Exemplo de uso

```
// Cálculo do Fatorial
int fatorial = 5;
int numero = fatorial;
int resultado = 1;

while (numero > 0) {
    resultado = resultado * numero;
    numero = numero - 1;
}
System.out.printf("%d! = %d", fatorial, resultado);
```

FUNÇÕES OU PROCEDIMENTOS

Funções ou Procedimentos

Um programa complexo é composto de muitas partes, de centenas ou milhares de pequenos programas que, quando combinamos, produzem uma solução (aplicação).

Usamos funções ou procedimentos para dividir os problemas de nossos programas em frações menores, mais fáceis de entender e que podem ser reutilizadas.



Funções ou Procedimentos

Todas as linguagens prevem alguma forma de criar rotinas, em geral considerando as seguintes práticas:

Dividem um problema grande em partes menores, mais fáceis de se entender a gerenciar.

Modularizam nosso programa, facilitando a compreensão das partes que o compõe.

Possibilitam o reaproveitamento de funções comuns, em diversos níveis.



PREMISSAS GERAIS

Premissas Gerais

Funções

Podem receber parâmetros de entrada para efetuar um processamento e produzir uma saída, que será retornada para quem chamou a função.

Exemplo: função que recebe dois valores numéricos, calcula e retorna o valor da soma deles

Premissas Gerais

Procedimentos

Tal como funções, podem receber parâmetros de entrada, mas executam um processamento que não produz uma saída.

Exemplo: procedimento para imprimir uma sequência de caracteres no dispositivo de saída padrão.

Premissas Gerais

Métodos

Próprio das linguagens orientadas a objetos, são funções e procedimentos declarados no contexto de uma classe e que operam sobre instâncias de um objeto.

Exemplo: um método para calcular a média de notas de uma estudante.

Premissas Gerais

Vamos utilizar o termo **rotina** para abstrair a idéia de função, procedimento ou método.

O formato geral de um rotina, em pseudo-linguagem, é:

```
nomeDaRotina([lista_de_parametros], ...) {  
    instrução1  
    instrução2  
    [retorna valor]  
}
```

Premissas Gerais

Sempre teremos as seguintes partes:

nomeDaRotina - é o nome que a identifica, tal como identificamos variáveis com nomes.

lista_de_parametros - é uma lista opcional de parâmetros que são os dados de entrada da rotina.

bloco de código - as instruções que compõem aquela função.

instrução de retorno - finaliza a execução daquela rotina opcionalmente retornando um valor, quando a rotina é uma função.



Premissas Gerais

Em nosso programa chamamos uma função através de seu nome, passando os parâmetros adequados. Nesse momento o fluxo do programa é passado para a função, que executa as instruções opcionalmente retornando um valor e, assim, retomando a execução no local onde a rotina foi chamada.

ROTINAS EM SWIFT

Rotinas em Swift

O Swift permite tanto a criação de funções, disponível globalmente em nosso programa, como métodos que são declarados no contexto de classes ou estruturas.



Sintaxe da **Rotina** em Swift

```
func nomeDaRotina([parametro1: tipoDoParametro1], [parmetro2: tipoDoparametro2], [n  
parametros...]) [-> tipoRetorno] {  
    instrucao1  
    instrucao2  
    [return valor]  
}
```

Rotinas em Swift

Considerações:

A declaração de parâmetros é opcional.

É necessário determinar os tipos dos parâmetros.

Caso a rotina seja uma função e retorne um valor, o tipo de retorno deve ser declarado após o fechamento dos parênteses.

Rotinas em Swift

Considerações:

Procedimentos podem usar o comando return para encerrar a execução a qualquer instante.

Ao chamar uma função é necessário identificar os parâmetros por nome antes de passar o valor.

Caso o tipo de retorno seja omitido a rotina é considerada um procedimento e não retorna valor.

Rotinas em Swift



Exemplo de uso

```
// Função para cálculo da somatória de 2 números
func sum(a: Int, b: Int) -> Int {
    return a + b
}

var soma = sum(a: 10, b: 20)
```

Importante: podemos usar variáveis para passar seu valor para os argumentos de uma rotina.

ROTINAS EM JAVA

Rotinas em Java

Classes são a construção básica do Java, portanto a linguagem não conta com funções ou procedimentos, mas apenas com métodos, ou seja, nossas rotinas sempre serão declaradas no contexto de uma classe.



Sintaxe dos métodos em Java

```
class nomeDaClasse {  
    public [tipo_retorno] nomeMetodo([tipoParametro1 nomeParametro1], [tipoParametro2  
nomeParametro2], [n parametros...]) {  
        instrucao1;  
        instrucao2;  
        [return valor];  
    }  
}
```

Rotinas em Java

Considerações:

A declaração de parâmetros é opcional.

É necessário determinar os tipos dos parâmetros.

Caso a rotina não retorne nenhuma valor o tipo de retorno deve ser usar a palavra reservada **void**.

Procedimentos podem usar o comando **return** para encerrar a execução a qualquer momento.

Rotinas em Java



Exemplo de uso

```
// Uma calculadora simples
public class Calculator {
    public int Add(int a, int b) {
        return a + b;
    }

    public int Subtract(int a, int b) {
        return a - b;
    }
}

// Ponto de entrada
public class Sample {
    public static void main(String[] args) {
        // Cria uma instância da calculadora
        Calculator calc = new Calculator();

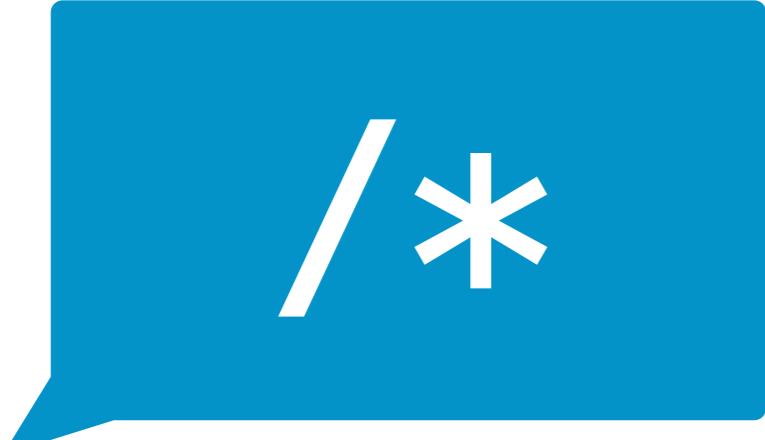
        // Calcula
        int addResult = calc.Add(10, 20);
        int subtractResult = calc.Subtract(34, 7);

        // Imprime os resultados
        System.out.printf("Resultado da soma %d", addResult);
        System.out.printf("Resultado da subtração %d", subtractResult);
    }
}
```

A construção mais próxima de uma função em Java são os métodos estáticos (static methods), que não dependem da existência de objetos para serem chamados.

COMENTÁRIOS

Comentários

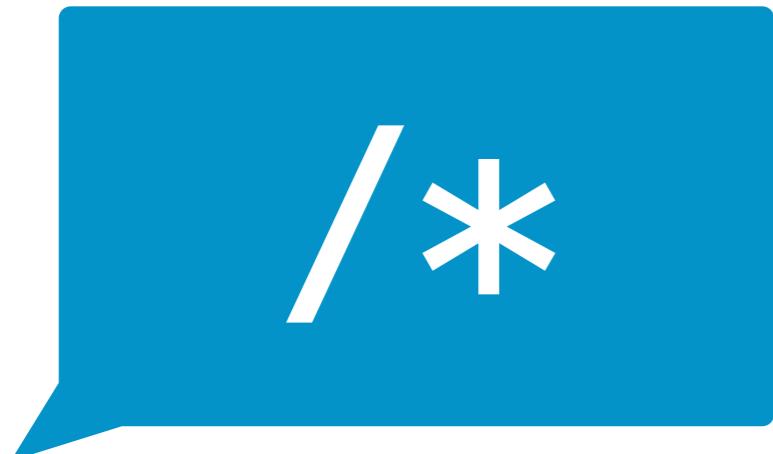


/*

Praticamente todas as linguagens de programação tem alguma forma de incluir comentários no código.

Usamos comentários para escrever considerações ou informações a respeito do que aquele programa faz, ou sobre a idéia que está sendo empregada naquele algoritmo.

Comentários



Comentários são fundamentais para ajudar a manter o código legível tanto para você quanto para seus pares.

Porém não abuse do uso de comentários, procure faze-lo de forma a complementar informações que não estejam óbvias pela própria leitura do código.

Comentários

Sintaxe dos comentários (igual para ambas linguagens)

```
// Comentários em linha
// Usados para criar comentários de uma única linha
    var x = 0          // Podem ser usados no final de uma instrução

/* Bloco de comentário
   Abrimos um bloco de comentário usando a sequência de caracteres barra e
asterisco.
   O bloco é fechado usando a sequência asterisco barra.
   Com isso podemos produzir um comentário mais extenso, seja para explicar a
idéia de um programa, seja para documentar o cabeçalho de um arquivo de código ou
função.
*/
```

ERROS EM PROGRAMAS

Erros em Programas

Os erros em programas são categorizados de maneira geral de duas formas:

Compile time: problemas que são identificados no momento de compilação do programa. Em geral, são problemas de sintaxe.



Runtime: problemas que acontecem durante a execução de um programa, em geral erros conceituais ou lógicos.

ESTRUTURAS DE COLEÇÕES DE DADOS

Estruturas de Coleções de Dados

O último passo dos conhecimentos introdutórios de programação, é o conhecimento das estruturas de dados presentes nas linguagens para trabalhar com estruturas de dados que armazenam coleções de informações.

Arrays (Matrizes / Vetores)

Possibilita trabalhar com uma coleção de dados, geralmente de um mesmo tipo.

Arrays são armazenados como uma lista de itens que podem ser acessados através de um índice.

Podem ter um tamanho fixo ou serem alocados dinamicamente.

Podem ser facilmente enumerados e a ordem com os elementos é acessado é previsível.



Arrays (Matrizes / Vetores)



Arrays em Swift

```
// Cria uma lista vazia de elementos de um Tipo
var nomeLista: [Tipo] = [Tipo]()

// Cria uma lista com 3 elementos pré-definidos
var lista2 = [ elemento1, elemento2, elemento3 ]

// Operações com Listas
nomeLista.count      // Retorna a quantidade de elementos
nomeLista.isEmpty    // Retorna verdadeiro se a lista estiver vazia
nomeLista[2]          // Acessa o terceiro elemento da lista
nomeLista.append(elemento) // Adiciona elemento na lista

// Enumerando elementos
for elemento in elementos {
}
```

Arrays (Matrizes / Vetores)



Exemplo de uso

```
// Uma estrutura representando os dados de uma pessoa
struct Person {
    var name: String                                // Nome
    var birthday: Date                            // Data de Nascimento
}

// Declara Uma lista de pessoas
var people = [
    Person(name: "João", birthday: getDate(1975, 10, 21)),
    Person(name: "Maria", birthday: getDate(1981, 5, 12)),
    Person(name: "Cecília", birthday: getDate(1979, 7, 11)),
    Person(name: "José", birthday: getDate(1985, 2, 23))
]

// Adiciona uma pessoa a lista
people.append(Person(name: "Izabel", birthday: getDate(1982, 3, 13)))
// Verifica se a lista não esta vazia e enumera
if (!people.isEmpty) {
    for person in people {
        let age = getDifferenceInYearsFromNow(person.birthday)
        print("\(person.name) tem \(age!) anos.")
    }
}
```

Arrays (Matrizes / Vetores)

Considerações:

Arrays empregam o conceito de mutabilidade no Swift:

Quando declaradas como constantes (com a keyword let), elas são consideradas não

mutáveis, e seus elementos não podem ser alterados.

Quando declaradas como variáveis (com a keyword var), elas são consideradas mutáveis, e

seus elementos podem ser modificados.



Arrays (Matrizes / Vetores)

Considerações:

Um **Array** em Swift é homogêneo, o que significa que todos os elementos nele contidos devem ser de um mesmo tipo.

O Swift consegue determinar o tipo de um **Array** por inferência, quando inicializamos ela com um conjunto de elementos de um mesmo tipo.



Arrays (Matrizes / Vetores)



Arrays e Listas em Java

```
// Arrays com dimensões fixas
TipoDeDado[ ] nomeVariavel = new TipoDeDado[Tamanho];
TipoDeDado[2] = valor;          // Atribui um valor ao segundo elemento da lista

// ArrayLists para listas com dimensões variáveis
ArrayList<TipoDeDado> nomeVariavel = new ArrayList<TipoDeDado>();
nomeVariavel.add(valor);

// Enumerando elementos
for (TipoDeDado variavel : lista) {  
}
```

Arrays (Matrizes / Vetores)



Exemplo de uso

```
// Arrays com dimensões fixas
Person[] people = new Person[5];
people[0] = new Person("João", Utils.getDate(1975, 10, 21));
people[1] = new Person("Maria", Utils.getDate(1981, 5, 12));
people[2] = new Person("Cecília", Utils.getDate(1979, 7, 11));
people[3] = new Person("José", Utils.getDate(1985, 7, 11));
people[4] = new Person("Isabel", Utils.getDate(1982, 3, 13));

// Verifica se a lista não é vazia e enumera
if (people.length > 0) {
    Date now = new Date();
    for (Person person : people) {
        int age = Utils.getDifferenceInYears(person.getBirthday(), now);
        System.out.printf("%s tem %d anos.\n", person.getName(), age);
    }
}

// Listas
ArrayList<Person> people = new ArrayList<>();
people.add(new Person("João", Utils.getDate(1975, 10, 21)));
people.add(new Person("Maria", Utils.getDate(1981, 5, 12)));
people.add(new Person("Cecília", Utils.getDate(1979, 7, 11)));
people.add(new Person("José", Utils.getDate(1985, 7, 11)));
people.add(new Person("Isabel", Utils.getDate(1982, 3, 13)));

// Verifica se a lista não é vazia e enumera
if (!people.isEmpty()) {
    Date now = new Date();
    for (Person person : people) {
        int age = Utils.getDifferenceInYears(person.getBirthday(), now);
        System.out.printf("%s tem %d anos.\n", person.getName(), age);
    }
}
```



Arrays (Matrizes / Vetores)

Considerações:

Os **Arrays** são uma construção mais antiga, e trabalham com um número fixo de elementos.

ArrayLists são uma construção mais moderna que permite trabalhar com listas de tamanho dinâmico.

Assim como em Swift, **Arrays** e **Listas** no Swift trabalham de forma homogênea, ou seja é preciso determinar o tipo de dado que ela vai conter.



Dictionaries (Dicionários)

Possibilitam trabalhar com uma coleção de elementos indexada a partir de um outro elemento (geralmente uma string), chamados de chaves.

Podemos facilmente enumerar tanto os elementos quanto as chaves do dicionário, porém a ordem em que eles são acessados é imprevisível.

Ideal para trabalhar com coleções de elementos que são identificadas por algum tipo específico (como um dicionário que tem uma palavra e sua definição).



Dictionaries (Dicionários)



Dictionaries em Swift

```
// Cria um dicionário com uma chave e um valor
var nomeDict: [TipoChave : TipoValor] = [TipoChave : TipoValor]()

// Cria uma lista com 3 elementos pré-definidos
var dict = [
    chave1: valor1,
    chave2: valor2,
    chave3: valor3
]

// Operações com Listas
nomeDict.count      // Retorna a quantidade de elementos no dicionário
nomeDict.isEmpty    // Retorna verdadeiro se o dicionário estiver vazio
nomeDict[chave]     // Acessa um elemento a partir de uma chave
nomeDict[chave] = valor // Atribui um valor a chave

// Enumerando elementos
for (chave, elemento) in dict {
```

Dictionaries (Dicionários)



Exemplo de uso

```
// Declara um dicionário contendo palavras-chave e uma breve explicação
var swiftKeywords = [
    "let": "Usada para criação de constantes ou Arrays não mutáveis.",
    "var": "Usada para criação de variáveis ou Arrays mutáveis.",
    "class": "Usada para declarar classes.",
    "func": "Usada para declarar funções ou métodos.",
    "protected": "Modificador para campos, classes e métodos os tornando de acesso
protegido."
]

// Adiciona uma palavra-chave
swiftKeywords["for"] = "Usada em Loops que enumeram elementos."

// Remove uma palavra-chave que não existe em Swift
swiftKeywords.removeValue(forKey: "protected")

// Verifica se o dicionário está vazio e enumera seus elementos
if !swiftKeywords.isEmpty {
    // Imprime um cabeçalho
    print("Palavras-chave do Swift.")
    print("-----\n")

    for (key, value) in swiftKeywords {
        print("- \(key): \(value)")
    }
}
```

Dictionaries (Dicionários)

Considerações:

Dictionaries empregam o conceito de mutabilidade no Swift:

Quando declaradas como constantes (com a keyword let), elas são consideradas não mutáveis e seus elementos não podem ser alterados.

Quando declaradas como variáveis (com a keyword var), elas são consideradas mutáveis e seus elementos podem ser modificados.

Dictionaries (Dicionários)

Considerações:

Um Dictionary em Swift é homogêneo, o que significa que todas as chaves devem ser de um mesmo tipo e todos os valores também devem obedecer a um tipo pré-determinado.

O Swift consegue determinar os tipos de chave e valor de um Dictionary por inferência, quando inicializamos ela com um conjunto de elementos.



Dictionaries (Dicionários)



Arrays e Listas em Java

```
// ArrayLists para listas com dimensões variáveis
HashMap<TipoChave, TipoValor> nomeVariavel = new HashMap<TipoChave, TipoValor>();
nomeVariavel.put(chave, valor);

// Enumerando elementos
for (TipoChave chave : nomeVariavel.keySet()) {
    nomeVariavel.get(chave);           // Obtém a chave
}
```

Dictionaries (Dicionários)



Exemplo de uso

```
// Declara e carrega uma lista de palavras-chave e uma breve explicação.  
HashMap<String, String> javaKeywords = new HashMap<>();  
javaKeywords.put("public", "Modificador para campos, classes e métodos os tornando de  
acesso públicos.");  
javaKeywords.put("class", "Usada para declarar classes");  
javaKeywords.put("func", "Usada para declarar funções ou métodos.");  
javaKeywords.put("protected", "Modificador para campos, classes e métodos os tornando  
de acesso protegidos.");  
javaKeywords.put("static", "Permite a criação de campos ou métodos estáticos.");  
  
// Remove uma palavra-chave que não existe em Java  
javaKeywords.remove("func");  
  
// Verifica se o dicionário não está vazio e enumera os elementos  
if (!javaKeywords.isEmpty()) {  
    // Imprime um cabeçalho  
    System.out.println("Palavras-chave em Java");  
    System.out.print("-----\n");  
  
    for (String key : javaKeywords.keySet()) {  
        System.out.printf("- %s: %s\n", key, javaKeywords.get(key));  
    }  
}
```

Dictionaries (Dicionários)

Considerações:

A classe que representa um Dictionary em Java se chama **HashMap**. Map é um sinônimo para dicionário em computação, e o prefixo Hash tem a ver com o funcionamento interno dela.

Um Dictionary em Java é homogêneo, o que significa que toda as chaves devem ser de um mesmo tipo, e todos os valores também devem obedecer a um tipo pré-determinado.

