

NOTEBOOK

HACKATRUCK

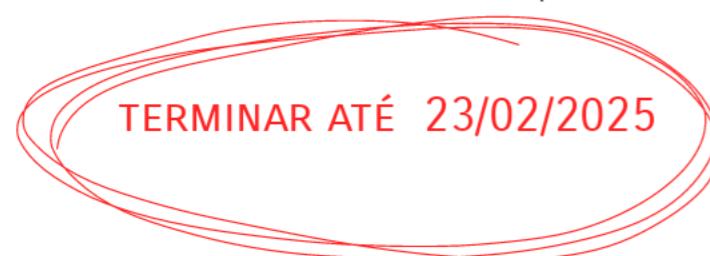
SITE HACKATRUCK PARA ESTUDOS: [HTTPS://HACKATRUCK.COM.BR/CURSOS/](https://hackatruck.com.br/cursos/)

ÚNICAS LINGUAGENS OFICIALMENTE SUPORTADAS PELA APPLE SÃO :
OBJECTIVE-C E SWIFT.

ALGUNS ARTIGOS PARA ESTUDO:

[HTTPS://HACKATRUCK.COM.BR/NOTICIAS/ARTIGOS-QUE-ENRIQUEM-AS-AULAS-DO-HACKATRUCK-INSTRUTOR-ANDRE-FORATTO/](https://hackatruck.com.br/noticias/artigos-que-enriquem-as-aulas-do-hackatruck-instrutor-andre-foratto/)

[HTTPS://HACKATRUCK.COM.BR/NOTICIAS/ARTIGOS-QUE-ENRIQUEM-AS-AULAS-DO-HACKATRUCK-INSTRUTOR-GABRIEL-THOMAZ/](https://hackatruck.com.br/noticias/artigos-que-enriquem-as-aulas-do-hackatruck-instrutor-gabriel-thomaz/)



Algoritmos em Swift (15 horas)	Orientação a Objetos (15 horas)	Swift (20 horas)
<ul style="list-style-type: none">• Conceitos Iniciais.• Comandos de Decisão.• Comandos de Repetição.• Funções.	<ul style="list-style-type: none">• Conceitos Iniciais.• Classes.• Encapsulamento.• Herança.• Polimorfismo.	<ul style="list-style-type: none">• Sintaxe Básica de Swift.• Estruturas de Dados em Swift.• Orientação a Objetos em Swift.

VARIÁVEIS E CONSTANTES

VARIÁVEL: VARIÁVEL (VAR) ARMAZENADOR QUE PODE SER ALTERADO.

VAR NOMEVARIAVEL = VALORMUTAVEL

CONSTANTE: CONSTANTE (LET) ARMAZENADOR QUE É FIXO, NÃO PODE SER ALTERADO.

LET NOMECONSTANTE = VALORMUTAVEL

NOMENCLATURAS DE VARIÁVEIS E CONSTANTES

```
var _teste = 123
var π = 3.14
var 🍎🍊🍋 = "Frutas"
var nome101 = "Joãozinho"
```

TODOS ESSES CASOS SÃO ACEITOS . DE REGRAS TEMOS QUE OS NOMES NÃO PODEM CONTER ESPAÇOS EM BRANCO, SÍMBOLOS MATEMÁTICOS E SETAS. TAMBÉM NÃO PODEM COMEÇAR COM NUMEROS.

TIPOS DE DADOS

INT: NÚMEROS INTEIROS
Ex: 2, 10, 478...

CHARACTER: UTILIZADO PARA UM UNICO CARACTERE.
Ex: A, B, 9, @, %...

STRING: É UTILIZADO PARA 2 OU MAIS CARACTERES.
Ex: PALAVRA, TIPO 2, UM...

FLOAT: PARA NÚMEROS DECIMAIS.
Ex: 1.9, 3.89778, 7.0

OBS: SE TENTARMOS ATRIBUIR 10.4999999, o SWIFT ATRIBUÍRAM A ESSE VALOR 10.5.

DOUBLE: PARA NÚMEROS DECIMAIS PORÉM COM MAIS PRECISÃO.
Ex: 2.99, 10.99...

OBS: SE ATRIBUÍRMOS O VALOR 123456789.12345678, o SWIFT IRÁ CONSIDERAR 123456789.1234568.

BOOL: PARA ARMAZENAR VERDADEIRO OU FALSO.
Ex: TRUE OU FALSE.

O SWIFT FAZ ESSA TIPIFICAÇÃO DE FORMA AUTOMÁTICA, QUANDO CRIAMOS UMA VARIÁVEL ATRIBUINDO VALOR A ELA.

VAR NUM = 10 → TIPIFICAÇÃO IMPLÍCITA

VAR NUM1: INT = 10 → TIPIFICAÇÃO EXPLICITA

VAR NUM2: INT? = 10 → OPCIONAL, CASO QUEIRA ATRIBUIR NIL A VARIÁVEL.

NIL: NOMENCLATURA CADA EM SWIFT PARA DADOS NULO. (NULL, EM OUTRAS LINGUAGENS)

Operador	Operação
Adição +	É utilizado para somar
Subtração -	É utilizado para subtrair
Multiplicação *	É utilizado para multiplicar
Divisão /	É utilizado para dividir
Resto %	Retorna o resto da divisão entre dois números
Operador	Operação
pow()	É utilizado para elevar à potência de um número
sqrt()	É utilizado para calcular a raiz quadrada de um número

VAR RESULTADO: INT
RESULTADO = POW(3,2) } EQUIVALENTE A 3^2

VAR RESULTADO: INT
RESULTADO = SQRT(9) } EQUIVALENTE A $\sqrt{9}$

Operação	Saída
true && true	Nosso resultado é true
false && true	Nosso resultado é false
true && false	Nosso resultado é false
false && false	Nosso resultado é false

EX:
 VAR RESULTADO: BOOL
 RESULTADO = TRUE && TRUE
 PRINT("O RESULTADO DE TRUE && TRUE É \\"(RESULTADO)")

Operação	Saída
true true	Nosso resultado é true
false true	Nosso resultado é true
true false	Nosso resultado é true
false false	Nosso resultado é false

EX:
 VAR RESULTADO: BOOL
 RESULTADO = TRUE || TRUE
 PRINT("O RESULTADO DE TRUE || TRUE É \\"(RESULTADO)")

ORDEM DE PRIORIDADE PARA OS OPERADORES LÓGICOS

Ordem	Operador
1	Parênteses ()
2	NOT !
3	AND &&
4	OR

OPERADORES LÓGICOS

PODEMOS USAR ALGUNS OPERADORES LÓGICOS COMO o NOT(!), AND(&&) E OR(||)

Operação	Saída
!true	Nosso resultado é false
!false	Nosso resultado é true

OPERADORES DE COMPARAÇÃO

Operador	Como é interpretada
<code>==</code>	É utilizado para verificar igualdade.
<code>!=</code>	É utilizado para verificar desigualdade (diferente).
<code><</code>	É utilizado para verificar se um operando é menor que.
<code>></code>	É utilizado para verificar se um operando é maior que.
<code><=</code>	É utilizado para verificar se um operando é menor ou igual a.
<code>>=</code>	É utilizado para verificar se um operando é maior ou igual a.

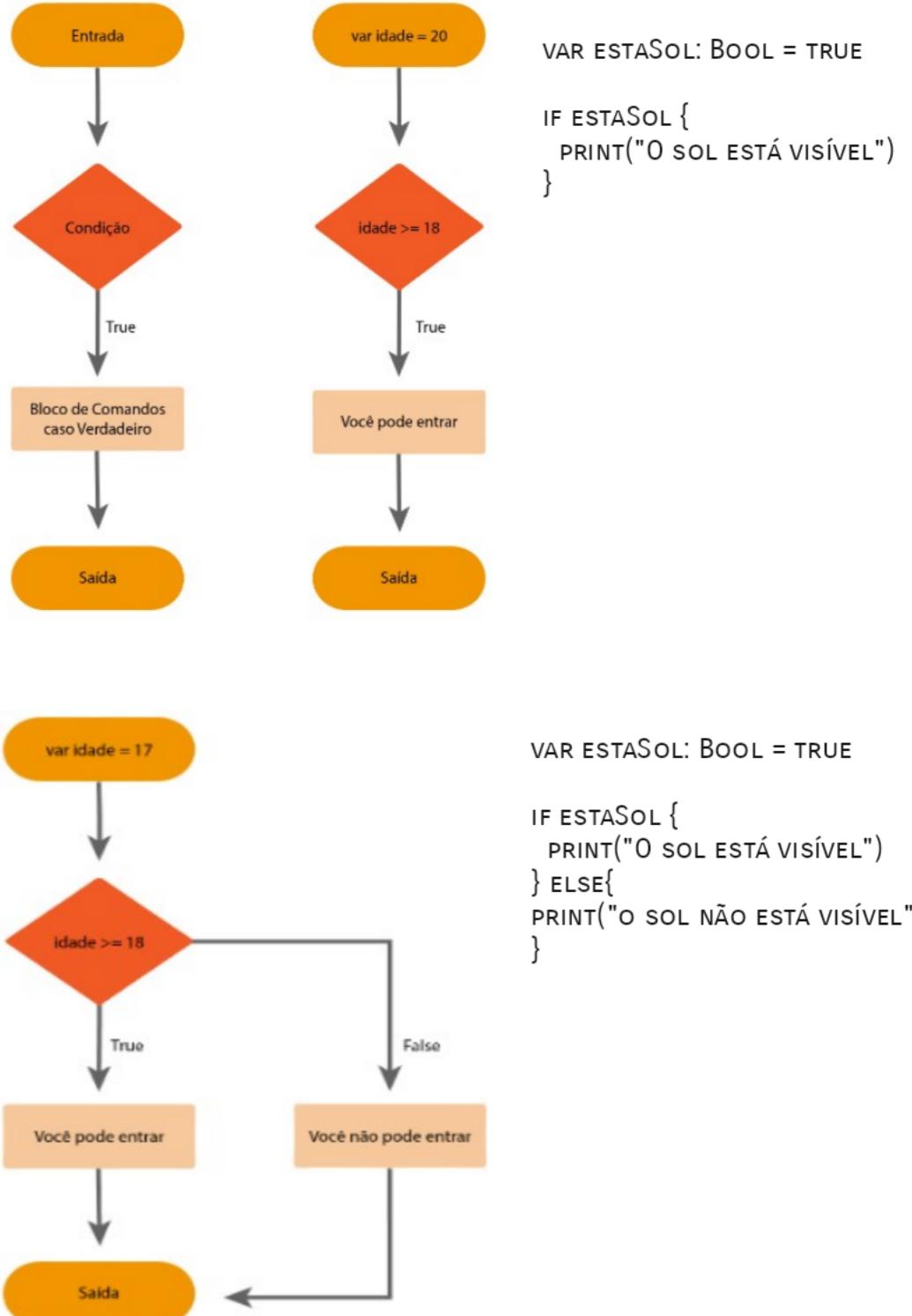
```

VAR SALARIOJOAO, SALARIOPEDRO, SALARIOMARCOS: DOUBLE } ATRIBUIÇÃO
VAR RESULTADO: BOOL } DAS VARIÁVEIS

SALARIOJOAO = 1300 } COLOCANDO VALORES
SALARIOPEDRO = 1000 } NAS VARIÁVEIS CRIADAS
SALARIOMARCOS = 1000 } VERIFICA COMO UM IF SE OS
                      } RESULTADOS SÃO IGUAIS
                      } RESULTADO RECEBE A ATRIBUIÇÃO
                      } DA IGUALDADE DOS 2 SALARIOS
RESULTADO = SALARIOJOAO == SALARIOPEDRO
PRINT ("O SALARIO DO JOÃO DE \$(SALARIOJOAO) É IGUAL O SALÁRIO DO PEDRO DE \$(SALARIOPEDRO) ? \$(RESULTADO)")

RESULTADO = SALARIOJOAO != SALARIOPEDRO
PRINT ("O SALARIO DO JOÃO É DIFERENTE DO SALÁRIO DO PEDRO? \$(RESULTADO)")
  
```

MÉTODO PARA PRINTAR
O VALOR DA VARIÁVEL



SWITCH CASE

O SWITCH É UMA FORMA DE NÃO ENTRARMOS EM GRANDES ESTRUTURAS DE IF ENCADEADAS E COMPLEXAS. OU SEJA, A LÓGICA É A MESMA DO IF, DIANTE DE UMA CONDIÇÃO FAÇA ALGO.

```
LET NUM = 8
```

```
SWITCH NUM {
CASE 0:
    PRINT("NUM TEM O VALOR 0")
CASE 1:
    PRINT("NUM TEM O VALOR 1")
DEFAULT:
    PRINT("NUM TEM OUTRO VALOR DIFERENTE DE 0 E 1")}
```

SERIA COMO SE FOSSE UM IF.
SERIA COMO SE FOSSE UM ELSE-IF.
SERIA COMO SE FOSSE UM ELSE

Operador	Operação
A ..< B	É utilizado para definir um intervalo entre um numero A e B excluindo B.
A ... B	É utilizado para definir um intervalo entre um numero A e B incluindo B.

Ex:

2..<6 ---> ESTÁ ENTRE 2 A 5

2...6 ---> ESTÁ ENTRE 2 A 6

COMANDOS DE REPETIÇÃO

WHILE: O COMANDO WHILE REPETE UM CONJUNTO DE OPERAÇÕES ENQUANTO UMA CONDIÇÃO FOR VERDADEIRA. SUA SINTAXE É A SEGUINTE:

```
VAR NUMERO = 1
VAR MENORQUECINCO = TRUE

WHILE MENORQUECINCO {
    IF NUMERO < 5{
        PRINT("O NÚMERO \((NUMERO) É MENOR QUE 5")
    }ELSE{
        MENORQUECINCO = FALSE
    }
    NUMERO += 1 //ESSA LINHA É IGUAL A ESCREVER NUMERO = NUMERO + 1
}
```

O WHILE FUNCIONARÁ ENQUANTO A VARIÁVEL BOOLEANA MENORQUECINCO FOR VERDADEIRA.

REPEAT-WHILE: O COMANDO REPEAT-WHILE É UMA VARIAÇÃO DO WHILE CUJA A CONDIÇÃO SÓ É VERIFICADA APÓS A PRIMEIRA EXECUÇÃO DOS COMANDOS NELE CONTIDOS.

VAR X = 5 —> IGUAL
VAR Y = 5 S

```
WHILE X != Y {
    PRINT("X É DIFERENTE DE Y")
}
```

O COMANDO FALA PARA PRINTAR A FRASE ENQUANTO X FOR DIFERENTE DE Y

```
REPEAT {
    PRINT("MESMO SEM VALIDAR A CONDIÇÃO SERÁ EXECUTADO AO MENOS
UMA VEZ")
} WHILE X != Y
```

REPEAT IRÁ FAZER UMA VEZ MESMO QUE O WHILE SEJA FALSO.

FOR: O COMANDO A SER EXECUTADO EM TODAS REPETIÇÕES ATÉ QUE ACABE O CONTADOR DEFINIDO.

FOR I IN 1...5 { —> CONTADOR DE 1 A 5

PRINT("BOM DIA ") —> IRÁ PRINTAR 5 "BOM DIA" NA TELA

}

OUTRA MANEIRA QUE PODEMOS UTILIZAR O FOR POR EXEMPLO SÃO EM TABUADAS COMO:

```
FOR I IN 1...10 {
    FOR J IN 1...10 {
        PRINT(" \((I) X \((J) = \((J * I)")
    }
    PRINT("_____")
```

EXECUTA A TABUADA COMPLETA DO 1 AO 10

1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
1 x 10 = 10
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20

O FOR DE I EXECUTA UMA VEZ E ESPERA O FOR DE J EXECUTADO POR COMPLETO, E FICA NESSA REPETIÇÃO ATÉ QUE O FOR DE I SEJA COMPLETO.

STRING

PODEMOS PERCORRER STRINGS COM UM FOR-IN. NESSE CASO, IREMOS ITERAR SOBRE OS CARACTERES DESSA STRING.

```
1 for c in "Entrada" {  
2     print(c)  
3 }
```



COMO PODEMOS ALTERAR UMA POSIÇÃO QUE NÃO SEJA A ÚLTIMA OU TROCAR UM ELEMENTO EM UM ARRAY? PARA ISSO, UTILIZAMOS O ÍNDICE DA SEGUINTE FORMA:

```
1 // Exemplo de indexação de Array  
2 var imparesMutaveis = [1, 3, 5, 7]  
3 let segundoImpar = imparesMutaveis[1]  
4 // Arrays vão de 0 até N-1 (onde N é o tamanho)  
5 for i in imparesMutaveis{  
6     print(i)  
7 }  
8 // Exemplo de alteração de elemento em determinado índice de um Array  
9 imparesMutaveis[0] = 9  
10 imparesMutaveis[1] = 11  
11 imparesMutaveis[2] = 13  
12 imparesMutaveis[3] = 15  
13 // imparesMutaveis agora é [9, 11, 13, 15]  
14 for i in imparesMutaveis{  
15     print(i)  
16 }
```

Run (Ctrl-Enter)

Output | Input | Comments (0)

```
1  
3  
5  
7  
9  
11  
13  
15
```

ARRAY

QUANDO QUEREMOS TRABALHAR COM VETORES, QUE SÃO COLEÇÕES DE DADOS INDEXADAS POR INTEIROS DE 0 À N-1 (ONDE N É O TAMANHO DA COLEÇÃO), UTILIZAMOS OS ARRAYS.

```
1 let pares: Array<Int> = [2, 4, 6, 8]  
2 let impares = [1, 3, 5, 7]
```

APPEND()

```
1 var imparesMutaveis = [1, 3, 5, 7]  
2 imparesMutaveis.append(9) // Agora, imparesMutaveis = [1, 3, 5, 7, 9]  
3  
4 for i in imparesMutaveis{  
5     print(i)  
6 }
```

input

PERCEBA A UTILIZAÇÃO DO COMANDO “APPEND” CHAMADO COM O USO DE UM “.” APÓS O NOME DE NOSSO ARRAY. CHAMAMOS ISSO DE ENVIO DA MENSAGEM “APPEND” (OU CHAMADA DO MÉTODO APPEND), RESPONSÁVEL POR

ADICIONAR AO FINAL DO ARRAY (NO CASO, IMPARESMUTAVEIS) O VALOR PASSADO COMO PARÂMETRO.

A JUNÇÃO DE ARRAYS TAMBÉM É VALIDA EM SWIFT, PODENDO CONCATENAR DOIS ARRAYS DE UM SIMPLES JEITO:

```
4 // E esse é outro jeito de se concatenar elementos:  
5 pares = [2, 4, 6, 8, 10]  
6 pares += [12, 14, 16, 18, 20]  
7 print(pares)  
8 // Será impresso: "[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]"
```

FUNÇÕES

PARA DEFINIR UMA FUNÇÃO UTILIZAREMOS O PREFIXO FUNC, APÓS ELE, UM NOME SEGUIDO DE () E POR FIM { } QUE ENGLOBA TODO O CÓDIGO CONTIDO NA FUNÇÃO.

```
1 |  
2 func corPredileta() {  
3     print("Minha cor predileta é laranja")  
4 }  
5  
6 corPredileta()
```

AQUI CRIAMOS A FUNÇÃO COM SUAS INSTRUÇÕES

AQUI CHAMAMOS A FUNÇÃO

PARAMETROS

PERMITE FORNECER UM VALOR DE ENTRADA QUE PODERÁ SER UTILIZADO NO CORPO DA NOSSA FUNÇÃO.

```
1 func corPredileta(cor: String) {  
2     print("Minha cor predileta é \(cor)")  
3 }  
4  
5 corPredileta(cor: "Azul")
```

PASSA A VARIÁVEL COR COMO PARÂMETRO

PODEMOS PASSAR QUANTOS PARÂMETROS QUISERMOS , CONTANDO QUE ESTEJAM PARAMETRIZADOS NA FUNÇÃO.

TIPOS DE RETORNO

```
4 func imprimeNome(nome: String) -> String{  
5     return "Meu nome é \(nome)"  
6 }  
7  
8 print(imprimeNome(nome: "Paulo"))
```

AQUI DIZEMOS QUE ESSA FUNÇÃO VAI RETORNAR UM VAR DO TIPO STRING, PODENDO VARIAR PARA INT DOUBLE, FLOAT E ETC.

RECURSAO

UMA FUNÇÃO QUE CHAMA A SI MESMA É CONHECIDA COMO FUNÇÃO RECURSIVA.

RECURSIVIDADE EXEMPLO

```
func calcular(valorEntrada:Int) -> Int{  
    var valorSaida : Int;  
    if (valorEntrada > 1){  
        valorSaida = valorEntrada * calcular(valorEntrada: valorEntrada - 1)  
    } else {  
        valorSaida = 1  
    }  
    return valorSaida  
}  
  
print("Resultado: ", calcular(valorEntrada: 5))
```

A FUNÇÃO CALCULAR CHAMA ELA RECURSIVAMENTE ATÉ QUE O VALOR DE IF SEJA FALSA. OU SEJA, O VALOR DE ENTRADA SENDO 5:

1. 5 · CALCULAR (5-1) --> 5 · CALCULAR(4)
2. 5 · CALCULAR (4-1) --> 5 · CALCULAR(3)
3. 5 · CALCULAR (3-1) --> 5 · CALCULAR(2)
4. 5 · CALCULAR (2-1) --> 5 · CALCULAR(1)

$$5 \cdot (4 \cdot (3 \cdot (2 \cdot 1))) = 5! = 120$$

ORIENTAÇÃO A OBJETO

CONCEITUALMENTE, A ORIENTAÇÃO A OBJETOS CONSISTE NA IDENTIFICAÇÃO DOS OBJETOS E DE SEUS PROCESSAMENTOS. O TERMO OBJETO É UTILIZADO PARA REPRESENTAR ELEMENTOS DO MUNDO REAL, ONDE TUDO AO SEU REDOR SÃO EXEMPLOS DE OBJETOS: CARRO, MESA, JANELA, LIVRO, PESSOA, ETC.

CLASSES

AS CLASSES PODEM SER EXEMPLIFICADAS POR MEIO DE AGRUPAMENTOS QUE FAZEMOS POR SEMELHANÇAS.

```
1 class Carro {  
2  
3     // Seus atributos e metodos podem ser definidos aqui.  
4 }
```

OBJETO

UM OBJETO, DE FORMA BEM SUCINTA, PODE SER DEFINIDO COMO QUALQUER COISA QUE VOCÊ VÊ E QUE "CABE" EM UMA CLASSE ESPECÍFICA.

```
2 class Carro{  
3     // Seus atributos e metodos podem ser definidos aqui.  
4 }  
5  
6 let saveiro = Carro () // criamos um objeto de Person  
7 let gol = Carro () // criamos um objeto de Person
```

ATRIBUTOS E PROPRIEDADES

AS PROPRIEDADES SÃO OS ATRIBUTOS COMUNS DESSA CLASSE QUE SERÃO COMPARTILHADOS PARA CADA OBJETO CRIADO A PARTIR DELA

```
1 class Carro {  
2  
3 // 1  
4 var ano: Int? // Estas são algumas das propriedades da classe Carro  
5 var marca: String?  
6 var modelo: String?  
7 var versao: String?  
8 var cor: String?  
9  
10 }
```

ATRIBUTOS DA CLASSE CARRO

MÉTODOS/FUNÇÕES

MÉTODOS/FUNÇÕES SÃO OS COMPORTAMENTOS DOS OBJETOS DE UMA CLASSE.

DIGAMOS QUE TODO CARRO PODE: ANDAR, DAR RÉ, BUZINAR E ETC. ESSAS AÇÕES PODEM VIR A SER CHAMADAS DE MÉTODOS/FUNÇÕES DA CLASSE.

```
1 class Carro {  
2  
3 // 1  
4 var ano: Int? // Estas são algumas das propriedades da classe Carro  
5 var marca: String?  
6 var modelo: String?  
7 var versao: String?  
8 var cor: String?  
9  
10 func andar(){  
11     print("Andando para frente")  
12 }  
13  
14 func darRe(){  
15     print("Andando para trás")  
16 }  
17  
18 func buzinar(){  
19     print("BIIIIIP BIIIP BIIIP")  
20 }  
21  
22 }
```

MÉTODOS

ENCAPSULAMENTO

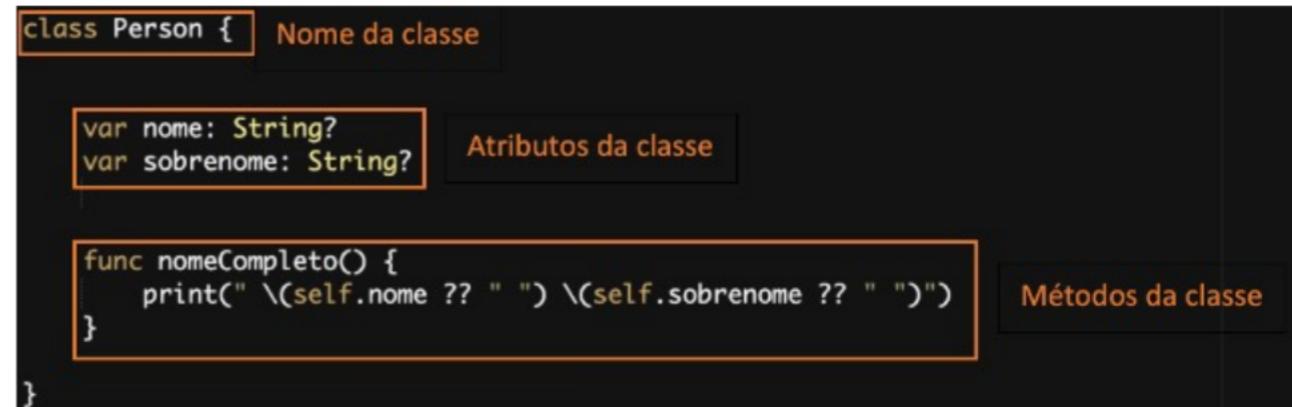
ATRAVÉS DO ENCAPSULAMENTO PODEMOS DEFINIR DIFERENTES NÍVEIS DE ACESSO PARA AS CLASSES, PROPRIEDADES E MÉTODOS. UTILIZAMOS DESTE CONCEITO QUANDO QUEREMOS CONTROLAR COMO NOSSAS CLASSES, OBJETOS E PROPRIEDADES SERÃO ACESSADOS POR OUTRAS CLASSES OU OBJETOS DENTRO DA APLICAÇÃO.

```
1 class Correntista {  
2  
3     var nome: String = "Leandro"  
4  
5     //Criamos a propriedade privada  
6     private var saldo: Double = 1000  
7  
8 }  
9  
10 //Criamos o objeto da classe  
11 var pessoal = Correntista()  
12  
13 pessoal.saldo = 2.50
```

É POSSÍVEL
OBSERVAR A
MENSAGEM DE ERRO
DIZENDO QUE NÃO
PODEMOS ALTERAR O
VALOR DA VARIÁVEL
POR SER
INACESSÍVEL E SER
PRIVADA.

CLASSES

A COMPOSIÇÃO DE UMA CLASSE SE DÁ POR 3 POSTOS-CHAVES, O NOME DA CLASSE, O CONJUNTO DE ATRIBUTOS DA CLASSE E POR FIM O CONJUNTO DE MÉTODOS DA CLASSE.



```
3 class Musica {  
4  
5     var nome: String?  
6     var artista: String?  
7     var album: String?  
8     var anoLancamento: Int?  
9     var duracao: Double?  
10    var rating: Int? //Nota de 0-5 por exemplo  
11    var linkToPlay: String?  
12  
13    func quemCanta(){  
14        print("O nome do artista é:" + artista!)  
15    }  
16  
17    func tocar(){  
18        print("Clique para ouvir:" + linkToPlay!)  
19    }  
20  
21    func feat( participante : String ) -> String {  
22        return "o artista \u201c" + artista! + "\u201d canta a música \u201c" + nome! + "\u201d com participação de \u201c" + participante + "\u201d"  
23    }  
24  
25    func nota() -> Int{  
26        return rating!;  
27    }  
28  
29 }  
30 }
```

ESSA É UM EXEMPLO DE UMA CLASSE COM SEUS ATRIBUTOS E MÉTODOS

```

1 class FiguraGeometrica {
2     init() {
3         print("O construtor da classe FiguraGeometrica foi chamado")
4     }
5 }
6
7 let quadrado = FiguraGeometrica()

```

input

```

O construtor da classe FiguraGeometrica foi chamado

```

EM SWIFT, O TIPO DE CONSTRUTOR MAIS BÁSICO QUE PODEMOS DEFINIR PARA UMA CLASSE É O QUE MOSTRAMOS NO EXEMPLO ANTERIOR, OU SEJA, COM A DEFINIÇÃO DA “FUNÇÃO” ESPECIAL NA CLASSE CHAMADA “INIT()”. É NOSSA RESPONSABILIDADE INICIALIZAR O ESTADO DO OBJETO DENTRO DESSA FUNÇÃO ESPECIAL.

```

1 class Bicicleta {
2     let rodas = 2
3     var dono: String
4
5     init(dono: String) {
6         // utilizamos "self.dono" para se referir a propriedade
7         // já que somente "dono" se refere ao parametro String
8         // do construtor
9         self.dono = dono
10    }
11 }
12
13 let bicicleta = Bicicleta(dono: "João") // Instanciamos a bicicleta de João.
14
15 print("A bicicleta de \(bicicleta.dono) tem \(bicicleta.rodas) rodas")

```

input

```

A bicicleta de João tem 2 rodas

```

ESSE CONSTRUTOR RECEBE UM PARÂMETRO CHAMADO “DONO” QUE É DO TIPO STRING E É ATRIBUÍDO À PROPRIEDADE “DONO” DO OBJETO QUE, PARA EVITAR A AMBIGUIDADE (E ERROS), DEVE SER REFERIDA COMO “SELF.DONO” DENTRO DO CONSTRUTOR.

// SUPONHA QUE JOÃO VENDA SUA BICICLETA PARA MATHEUS, PODEMOS REPRESENTAR ISSO EM NOSSO PROGRAMA ALTERANDO O DONO DE BICICLETA. PERCEBA QUE NÃO ATRIBUÍMOS UMA NOVA BICICLETA À CONSTANTE, ALGO QUE OCASIONARIA UM ERRO, APENAS ALTERAMOS O ESTADO DO OBJETO BICICLETA, ALTERANDO SUA PROPRIEDADE NOME.

```

let bicicleta = Bicicleta(dono: "João") // Instanciamos a bicicleta de João.

print("A bicicleta de \(bicicleta.dono) tem \(bicicleta.rodas) rodas")

bicicleta.dono = "Matheus"

print("A bicicleta de \(bicicleta.dono) tem \(bicicleta.rodas) rodas")

```

PROTOCOLOS



Classe Carro

CADA OBJETO TEM SUAS PRÓPRIAS
“CÓPIAS” DO QUE FOI DEFINIDO NA
CLASSE, OU SEJA, CADA UM DELES TEM
SEUS PRÓPRIOS ATRIBUTOS E MÉTODOS.



Carro 1



Carro 2



Carro 3

DENTRO DA CLASSE CARRO, PODEMOS TER VÁRIOS MODELOS CORES MARCAS ANOS A PARTIR DELA.

```
1- class Carro {  
2  
3    var ano: Int? // Estas são algumas das propriedades da classe Carro  
4    var marca: String?  
5    var modelo: String?  
6    var versao: String?  
7    var cor: String?  
8  
9    func descricao(){  
10       print("O carro \(self.modelo!) da marca \(self.marca!), versão \(self.versao!) e ano \(self.ano!), é da cor \(self.cor!)")  
11    }  
12  
13}  
14  
15  
16 var carro1 = Carro() } CRIANDO OS OBJETOS  
17 var carro2 = Carro() } DA CLASSE  
18 var carro3 = Carro()  
19  
20  
21 carro1.cor = "Laranja"  
22 carro1.ano = 1980  
23 carro1.modelo = "Fuxca"  
24 carro1.versao = "Turbo Shift Auto"  
25 carro1.marca = "MM"  
26  
27 carro2.cor = "Azul"  
28 carro2.ano = 1980  
29 carro2.modelo = "Fuxca"  
30 carro2.versao = "Turbo Shift Auto"  
31 carro2.marca = "MM"  
32  
33 carro3.cor = "Verde"  
34 carro3.ano = 1980  
35 carro3.modelo = "Fuxca"  
36 carro3.versao = "Turbo Shift Auto"  
37 carro3.marca = "MM"  
38  
39 print(carro1.descricao())  
40 print(carro2.descricao())  
41 print(carro3.descricao())  
42
```

ATRIBUTO DAS
CÓPIAS

INSTANCIAR SIGNIFICA CRIAR UM OBJETO A PARTIR DE UMA CLASSE

NA PROGRAMAÇÃO ORIENTAÇÃO A OBJETOS DIZEMOS QUE UM OBJETO POSSUI UMA INTERFACE, OU SEJA, O QUE ELE CONHECE E O QUE SABE FAZER. POR MEIO DA INTERFACE É POSSÍVEL SABER QUAIS SERVIÇOS PODEM SER EXECUTADOS E TAMBÉM AS MENSAGENS QUE O OBJETO RECEBE.

ENCAPSULAMENTO

UTILIZAMOS ENCAPSULAMENTO QUANDO QUEREMOS DEFINIR COMO NOSSAS CLASSES, PROPRIEDADES E MÉTODOS SERÃO ACESSADOS POR OUTRAS CLASSES OU OBJETOS DENTRO DA APLICAÇÃO.

EXISTEM 3 NÍVEIS DE ENCAPSULAMENTO EM SWIFT:

PUBLIC: PERMITE ACESSO A QUALQUER OUTRO ELEMENTO E POR QUALQUER FUNÇÃO.

INTERNAL: PERMITE ACESSO APENAS DENTRO DA PRÓPRIA CLASSE E DE CLASSES HERDEIRAS.

PRIVATE: PERMITE ACESSO APENAS PELA PRÓPRIA CLASSE.

AQUI UM EXEMPLO DE COMO MUDA UM VALOR DE UM ATRIBUTO PRIVADO DE UMA CLASSE.

HERANÇA

PERMITE UMA MELHOR ORGANIZAÇÃO E REAPROVEITAMENTO DE CÓDIGO. POR MEIO DESSE CONCEITO, AS CLASSES FILHAS COMPARTILHAM OS ATRIBUTOS E MÉTODOS DA CLASSE MÃE.

NO EXEMPLO ABAIXO VEREMOS “CARRO ESTENDE VEICULO”, ONDE A CLASSE CARRO É A SUBCLASSE E A CLASSE VEICULO É A SUPERCLASSE.

DESSE MODO, CARRO TERÁ TODOS OS ATRIBUTOS PÚBLICOS DA CLASSE VEICULO, E TAMBÉM PODERÁ UTILIZAR SEUS MÉTODOS PÚBLICOS (OU ATÉ MESMO MODIFICÁ-LOS)

PORTANTO, O QUE É PRIVADO EM UMA CLASSE NÃO SERÁ OBSERVADO NAS SUAS SUBCLASSES.

GENERALIZAÇÃO: QUANDO PARTIMOS DE UMA CLASSE E CHAGAMOS A SUA SUPERCLASSE.

Especialização - Quando partimos de uma superclasse e chegamos na sua subclasse.

```
1+ class Veiculo{  
2+     var pneus: String?  
3+     var assentos: String?  
4+     var motor: String?  
5+  
6+     func desc() {  
7+         print("Número de rodas \$(self.pneus ?? " "), Número de Assentos \$(self.assentos ?? " "), Potencia Motor \$(self.motor ?? " ")")  
8+     }  
9+ }  
10+  
11+ class Carro: Veiculo {  
12+     var tipo: String?  
13+ }  
14+  
15+ class Caminhao: Veiculo {  
16+     var tipo: String?  
17+     var eixos: Int?  
18+ }  
19+  
20+ class Motocicleta: Veiculo {  
21+     var cilindradas: Int?  
22+ }  
23+  
24+ class Aviao: Veiculo {  
25+     var numMotores: Int?  
26+ }  
27+  
28+ }
```

AQUI TEMOS A CLASSE VEÍCULO E SUA SUB-CLASSES: CARRO, CAMINHÃO, MOTOCICLETA E AVIÃO.

SUBCLASSES TAMBÉM PODEM RECEBER ATRIBUTOS COMO VIMOS NO EXEMPLO ANTERIOR. MAS NÃO SE RESUME APENAS EM ATRIBUTOS, SUBCLASSES PODEM RECEBER NOVOS MÉTODOS ASSIM COMO UMA CLASSE.

```
1+ class Veiculo{  
2+     var pneus: String?  
3+     var assentos: String? → CLASSE PRINCIPAL  
4+     var motor: String?  
5+     var cidadeDeRegistro: String?  
6+  
7+     func desc() {  
8+         print("Número de pneus \$(self.pneus ?? " "), Número de Assentos \$(self.assentos ?? " "), Potencia Motor \$(self.motor ?? " ")")  
9+     }  
10+  
11+     func registro() {  
12+         print("Nossa cidade de registro é \$(self.cidadeDeRegistro ?? " ")")  
13+     }  
14+ }  
15+  
16+  
17+ class Motocicleta: Veiculo {  
18+     var cilindradas: Int?  
19+ → SUBCLASSE E SEU MÉTODO  
20+     func descansar(){  
21+         print("Farei paradas de 2h em 2h, o destino é Maresias. \n")  
22+     }  
23+ }
```

POLIMORFISMO

CONSEGUIMOS TRATAR AS CLASSES DE ACORDO COM SUA NATUREZA, POR MEIO DE MÉTODOS DE MESMA ASSINATURA (TIPO DE RETORNO, NÚMERO, ORDEM E TIPO DOS PARÂMETROS) E COMPORTAMENTOS DIFERENTES.

O POLIMORFISMO PERMITE QUE CLASSES HERDADAS TENHAM UM COMPORTAMENTO SIMILAR À CLASSE MÃE, PORÉM VIABILIZANDO CERTAS ESPECIALIZAÇÕES.

```
1+ class Veiculo{ CLASSE PAI  
2+     var modelo: String?  
3+     var marca: String?  
4+     var ano: Int?  
5+  
6+     func desc() { MÉTODOS  
7+         print(" \$(self.modelo ?? " "), marca \$(self.marca ?? " ") e ano \$(self.ano ?? 0)  
8+     }  
9+ }  
10+  
11+ func buzinar() { MÉTODOS  
12+     print(" Barulho emitido pelo respectivo veiculo")  
13+ }  
14+ }  
15+ class Motocicleta: Veiculo { CLASSE MOTO HERDADA DE VEICULO  
16+  
17+     override func buzinar() { MÉTODO HERDADO DA CLASSE PAI PORÉM COM POLIMORFISMO  
18+         print(" Barulho emitido pela \$(self.modelo ?? " ")")  
19+         print(" biiip biiip biiip \n")  
20+     }  
21+  
22+     init (ma: String, mo: String, a: Int)  
23+     {  
24+         super.init()  
25+         marca = ma  
26+         modelo = mo  
27+         ano = a  
28+     }  
29+ }  
30+  
31+ var mot = Motocicleta(ma:"Honda", mo: "GC" , a: 2013)  
32+ mot.desc()  
33+ mot.buzinar()
```

OVERRIDING

OCORRE QUANDO UMA CLASSE FILHA REDEFINE O COMPORTAMENTO DE UM MÉTODO DA CLASSE MÃE. ESTE TIPO DE POLIMORFISMO TAMBÉM É CHAMADO DE SOBRESCRITA DE MÉTODOS. NA REDEFINIÇÃO DE MÉTODOS, UM MÉTODO É MANTIDO COM A MESMA ASSINATURA, MAS SUA IMPLEMENTAÇÃO É ALTERADA NAS CLASSES FILHAS.

OVERLOADING

TAMBÉM CONHECIDO COMO OVERLOADING, ESTE CONCEITO TRATA DE MÚLTIPLAS DEFINIÇÕES DE UM MÉTODO NUMA MESMA CLASSE. A DIFERENCIACÃO SE DÁ POR MEIO DE ASSINATURAS DISTINTAS PARA CADA “VERSÃO” DO MÉTODO, OU SEJA, APESAR DOS MÉTODOS POSSUÍREM O MESMO NOME, CADA VERSÃO DELES RECEBE UMA LISTA DE PARÂMETROS DISTINTA.