



Reskilling 4Employment Software Developer

Desenvolvimento de Aplicações Mobile - iOS

Bruno Santos

bruno.santos@cesae.pt

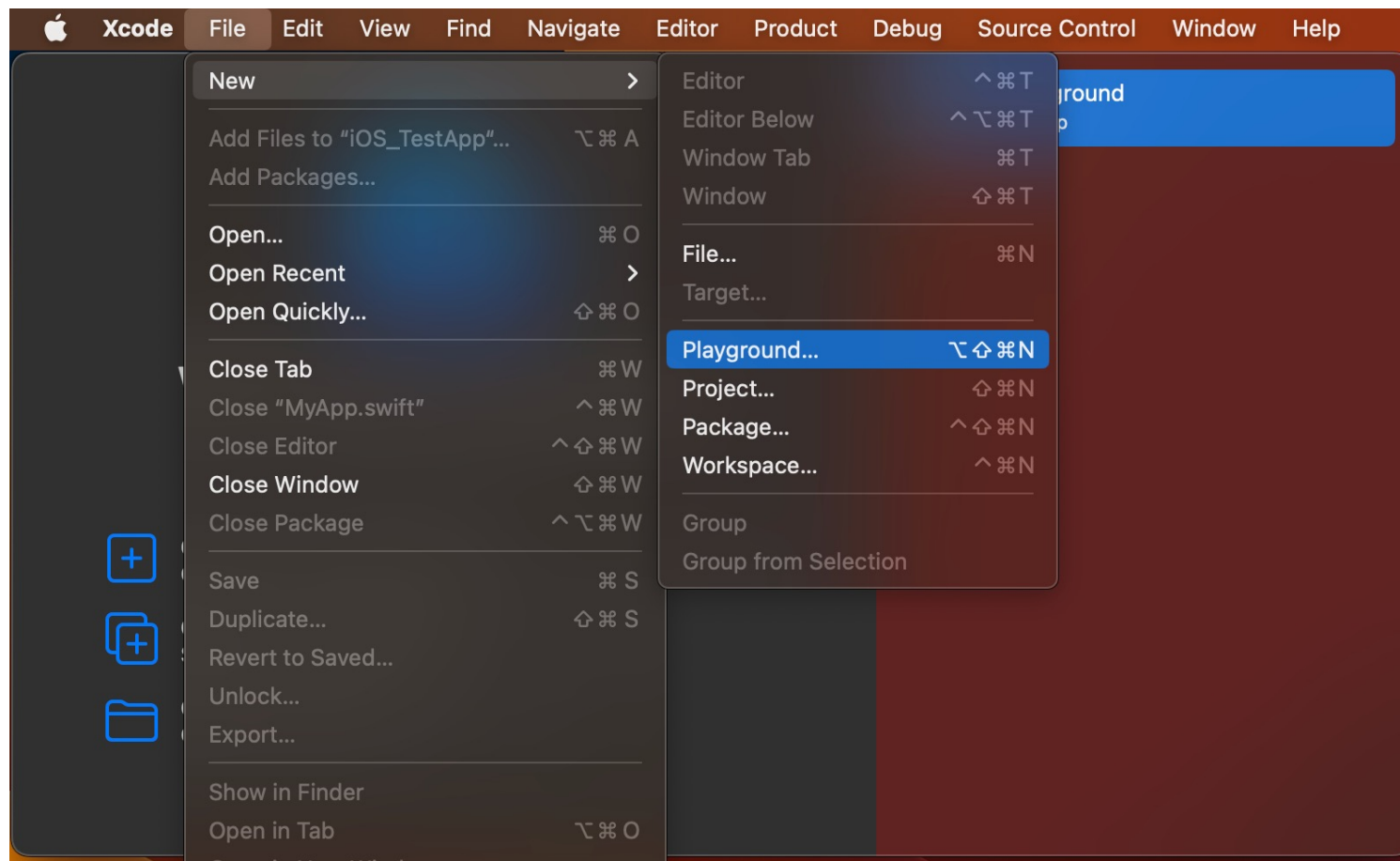
Swift

“Swift is a powerful and intuitive programming language for iOS, iPadOS, macOS, tvOS, and watchOS. Writing Swift code is interactive and fun, the syntax is concise yet expressive, and Swift includes modern features developers love. Swift code is safe by design and produces software that runs lightning-fast.”

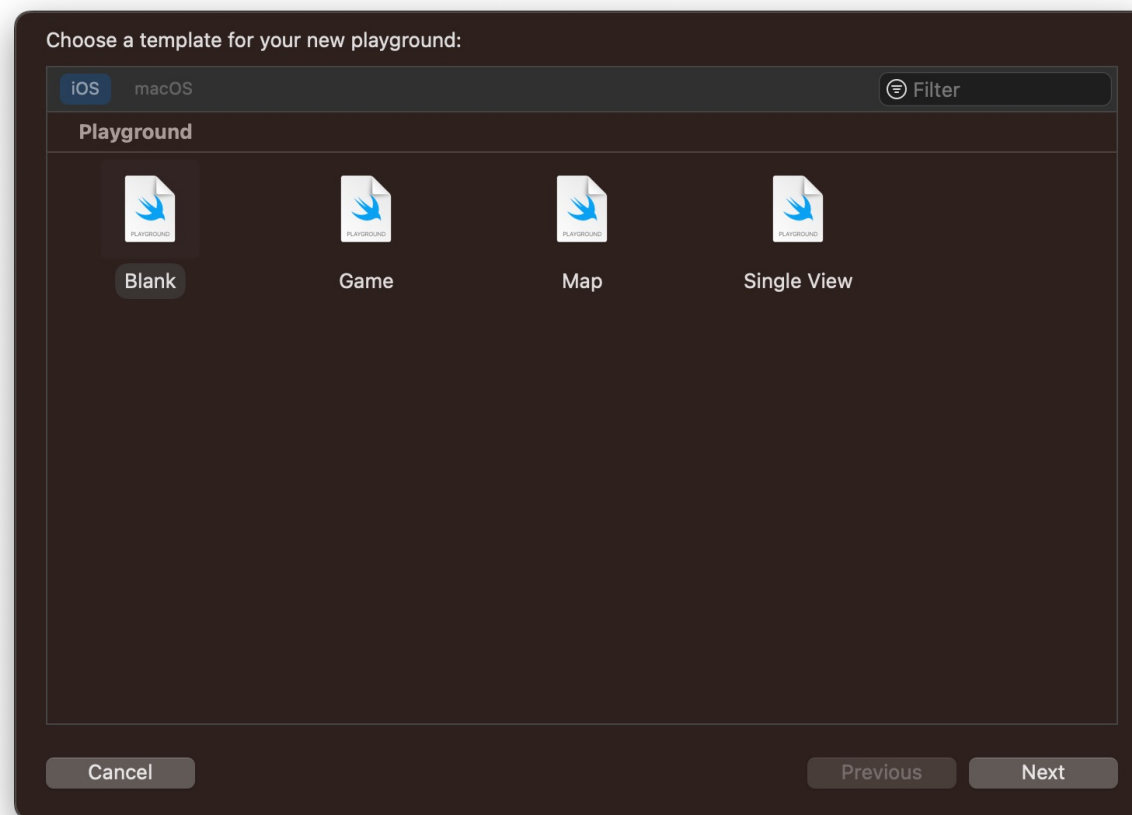
Xcode – Swift Playground

- O Swift Playground é o ambiente de desenvolvimento focado na linguagem Swift existente dentro do Xcode e que permite o desenvolvimento e teste de código Swift sem a necessidade da criação de uma interface gráfica (mobile/desktop).

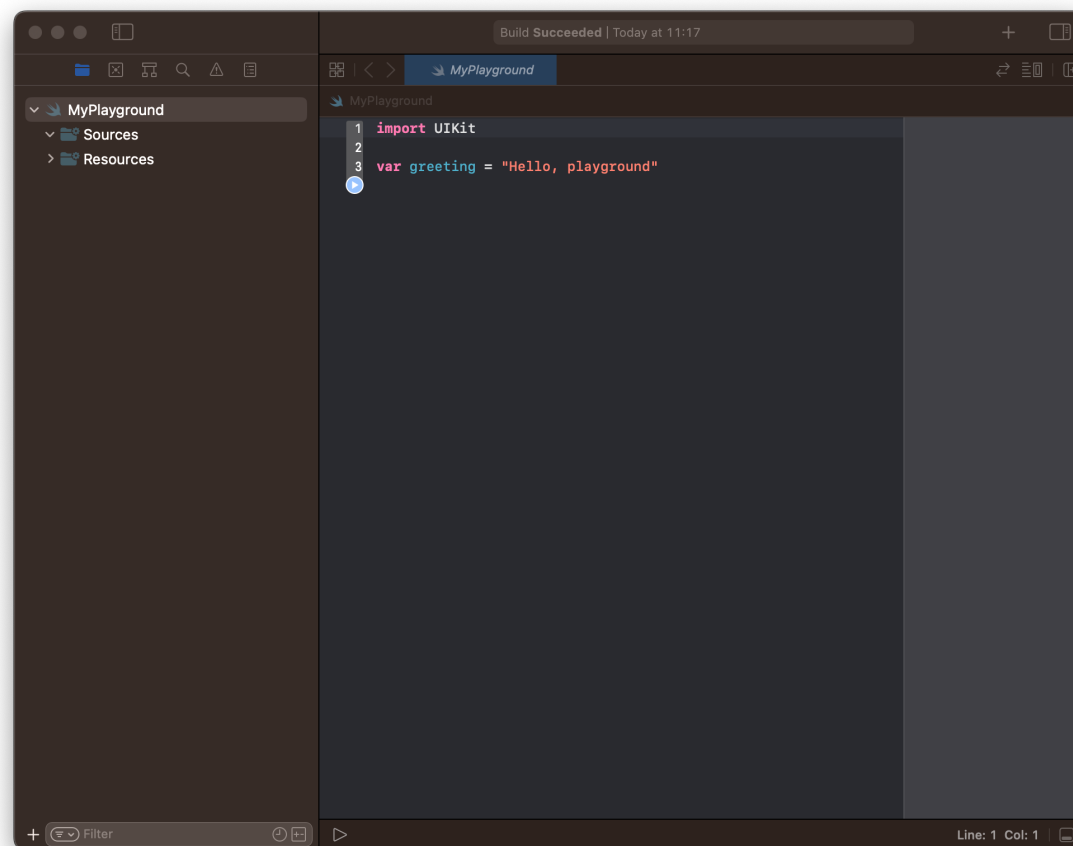
Xcode – Swift Playground



Xcode – Swift Playground



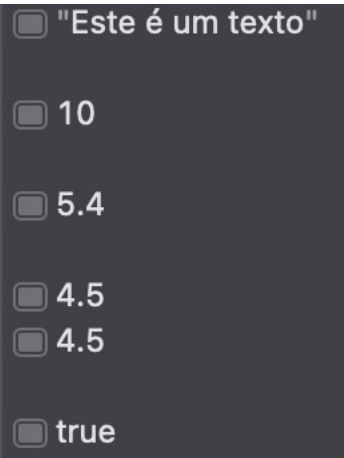
Xcode – Swift Playground



Swift – Tipos de Dados

- A definição de tipos de dados em Swift é feita por inferência, assim o tipo de dados nem sempre é necessário definir.

```
3 var texto = "Este é um texto"
4
5 var numeroInteiro = 10
6
7 var numeroDecimal = 5.4
8
9 var numeroDecimal1 : Double = 4.5
10 var numeroDecimal2 : Float = 4.5
11
12 var booleano = true
```




- Nota: Swift é uma linguagem que não obriga a utilização de ; no final de cada instrução

Swift – Variáveis vs Constantes

- Variável – valor pode ser alterado ao longo da execução do código.
- Constantes – valor não pode ser alterado após a primeira atribuição.
- As constantes têm a vantagem (comparativamente com as variáveis) de serem mais seguras (não altera tipos de dados de valor) e mais performáticas.
- Variáveis são definidas com “var” e constantes com “let”.

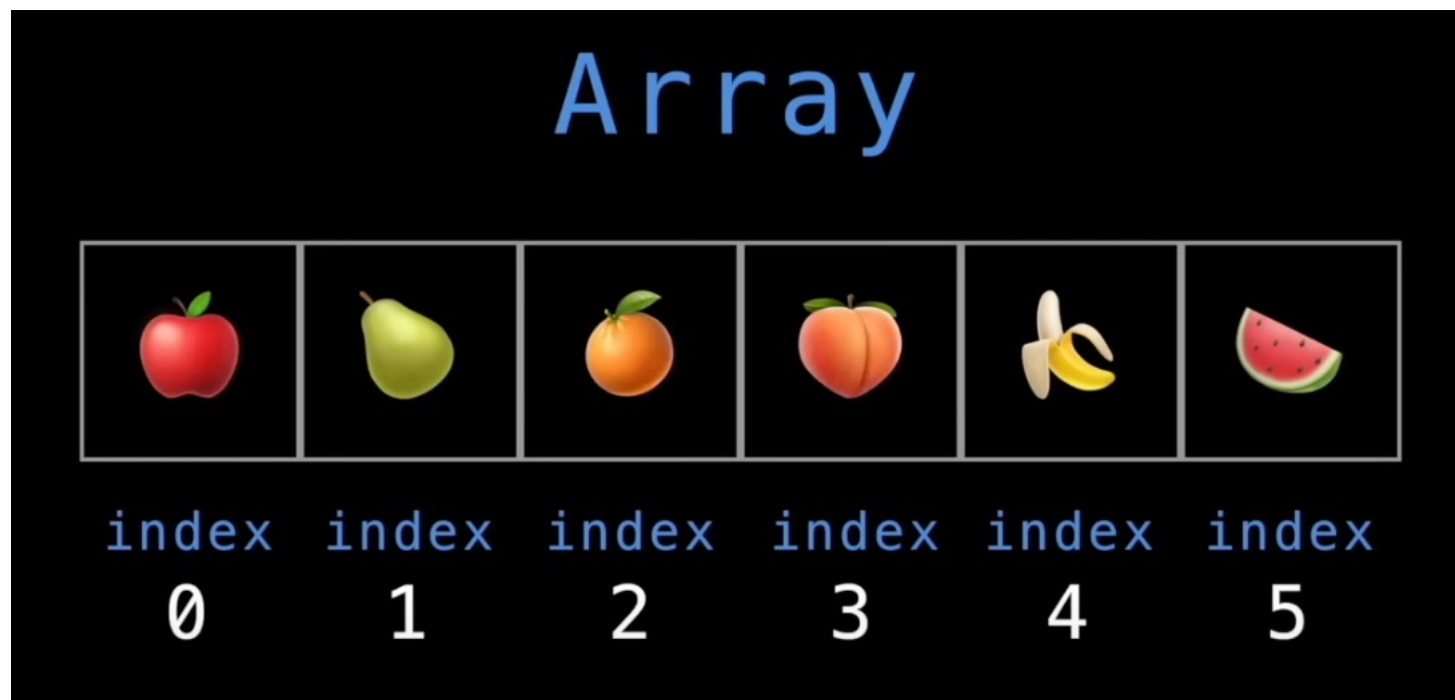
Swift – Variáveis vs Constantes

```
3 var nome = "Bruno"
4 nome = "Filipe"
5
6 let sobrenome = "Santos"
7 sobrenome = "Ramos"  Cannot assign to value: 'sobrenome' is a 'let' const...
```

Swift – Array

- Os arrays em Swift, como na generalidade das linguagens, são zero-indexados, ou seja, potencialmente guardam vários valores em posições e iniciam-se na posição 0.

Swift – Array



Swift – Array

```
4 var vazio: [Int] = []  
5  
6 var idades = [4,12,54,34,22,5,13,22]  
7  
8 vazio.count  
9 idades.count  
10  
11 idades.first  
12  
13 idades.last  
14  
15 vazio.last  
16
```

<input type="checkbox"/>	[]
<input type="checkbox"/>	[4, 12, 54, 34, 22, 5, 13, 22]
<input type="checkbox"/>	0
<input type="checkbox"/>	8
<input type="checkbox"/>	4
<input type="checkbox"/>	22
<input type="checkbox"/>	nil

Swift – Array

```
1 import UIKit
2
3
4 var vazio: [Int] = []
5
6 var idades = [4,12,54,34,22,5,13,22]
7
8 idades[3]
9
10 idades.append(10)
11
12 idades.insert(44, at: 0)
13
14 print(idades)
15
16 idades.sort()
17 print(idades)
```

☐ []

☐ [4, 12, 54, 34, 22, 5, 13, 22]

☐ 34

☐ [4, 12, 54, 34, 22, 5, 13, 22, 10]

☐ [44, 4, 12, 54, 34, 22, 5, 13, 22, 10]

☐ "[44, 4, 12, 54, 34, 22, 5, 13, 22, 10]\n"

☐ [4, 5, 10, 12, 13, 22, 22, 34, 44, 54]

☐ "[4, 5, 10, 12, 13, 22, 22, 34, 44, 54]\n"

☐ Line: 18 Col: 1

[44, 4, 12, 54, 34, 22, 5, 13, 22, 10]
[4, 5, 10, 12, 13, 22, 22, 34, 44, 54]

Swift – Array

14 <code>print(idades)</code>	<input type="checkbox"/> "[44, 4, 12, 54, 34, 22, 5, 13, 22, 10]\n"
15	
16 <code>idades.sort()</code>	<input type="checkbox"/> [4, 5, 10, 12, 13, 22, 22, 34, 44, 54]
17 <code>print(idades)</code>	<input type="checkbox"/> "[4, 5, 10, 12, 13, 22, 22, 34, 44, 54]\n"
18	
19 <code>idades.reverse()</code>	<input type="checkbox"/> [54, 44, 34, 22, 22, 13, 12, 10, 5, 4]
20 <code>print(idades)</code>	<input type="checkbox"/> "[54, 44, 34, 22, 22, 13, 12, 10, 5, 4]\n"
21	
22 <code>idades.shuffle()</code>	<input type="checkbox"/> [13, 22, 5, 10, 34, 44, 4, 22, 54, 12]
23 <code>print(idades)</code>	<input type="checkbox"/> "[13, 22, 5, 10, 34, 44, 4, 22, 54, 12]\n"

Swift – Set

- Os Set são semelhantes aos Arrays, com duas principais diferenças: não aceita valores duplicados e não permitem ordenação.
- A grande vantagem é a performance!
- De notar que a apresentação dos valores é colocada de ordem aleatória.

Swift – Set

```
4 var idades = [4,12,12,54,34,22,12]
5
6 var idadesSet = Set(idades)
7
8 print(idadesSet)
9
10 idadesSet.contains(54)
```

☐ [4, 12, 12, 54, 34, 22, 12]

☐ {34, 4, 22, 12, 54}

☐ "[34, 4, 22, 12, 54]\n"

☐ true

Swift – Dicionários

- Dicionários são tipos de dados no formato chave-valor, o que permite atribuir a cada valor um identificador específico.

Swift – Dicionários

```
4 let pessoas: [String:Int] = [  
5     "Bruno" : 30,  
6     "Maria" : 24,  
7     "Joana" : 25,  
8     "Filipa": 42,  
9     "Marco" : 30  
10 ]  
11  
12 print(pessoas)  
13  
14 pessoas["Bruno"]
```

["Marco": 30, "Maria": 24, "Filipa": 42, "Joana": 25, "Bruno": 30]

["Marco": 30, "Maria": 24, "Filipa": 42, "Joana": 25, "Bruno": 30]\n"

30

Line: 14 Col: 15

["Marco": 30, "Maria": 24, "Filipa": 42, "Joana": 25, "Bruno": 30]

Swift – Funções

Sem parâmetros

```
4 func escreverOla(){  
5     print("Ola")  
6 }  
7  
8 escreverOla()|
```

Ola

Com parâmetros


```
4 func escreverNome(nome:String){  
5     print(nome)  
6 }  
7  
8 escreverNome(nome:"Bruno")
```

Bruno

Swift – Funções

```
4 func soma(primeiroNumero:Int, segundoNumero: Int) -> Int {  
5     return primeiroNumero + segundoNumero  
6 }  
7  
8 soma(primeiroNumero: 12, segundoNumero: 5)
```

17
17



Swift – Funções

- Em Swift é possível utilizar labels para os argumentos o que pode permitir uma leitura melhor do código, neste caso vamos nomear o segundo parâmetro da função como mais e veremos como fica uma leitura muito mais limpa.
- De notar que a label é utilizada na chamada à função e que o parâmetro é usado dentro da função

Swift – Funções

```
3  
4 func soma(primeiroNumero:Int, mais segundoNumero: Int) -> Int {  
5     return primeiroNumero + segundoNumero  
6 }  
7  
8 soma(primeiroNumero: 12, mais: 5)
```

17
17

Swift – Condições (if/else)

```
4 var isChecked = true
5
6 if isChecked {
7     print("Marcado")
8 } else {
9     print("Desmarcado")
10 }
```

☐

Marcado

```
3
4 var isChecked = false
5
6 if isChecked {
7     print("Marcado")
8 } else {
9     print("Desmarcado")
10 }
```

☐

Desmarcado

Swift – Condições (if/else)

```
4 var nota = 18
5
6 if nota < 10 {
7     print("Reprovado")
8 } else if(nota == 10) {
9     print("Aprovado com 10")
10 } else if(nota < 15) {
11     print("Aprovado")
12 } else {
13     print("Aprovado com boa nota: \(nota)")
14 }
```

☐

Aprovado com boa nota: 18

Interpolação do
valor da nota

Swift – Switch/Case

```
3 let codigo = 4
4
5 switch codigo{
6     case 1: print("Um")
7     case 2: print("Dois")
8     case 3: print("Três")
9     case 4: print("Quatro")
10    default: print("Outro valor")
11 }
12
```

☐

Quatro

```
3 let codigo = 15
4
5 switch codigo{
6     case 1..<10: print("Até Dez")
7     case 11..<20: print("Até Vinte")
8     case 21..<30: print("Até Trinta")
9     case 31..<40: print("Até Quarenta")
10    default: print("Maior que 40 ou Menor que 1")
11 }
12
```

☐

Até Vinte

Swift – Ciclos

```
4 var nomes = ["Bruno", "João", "Carla", "Mariana"]
5
6 for nome in nomes{
7     print(nome)
8 }
```



Bruno
João
Carla
Mariana

```
4 var nomes = ["Bruno", "João", "Carla", "Mariana"]
5
6 for nome in nomes where nome == "Bruno"{
7     print(nome)
8 }
```



Bruno

Swift – Ciclos

```
6 for i in 0...10{
7     print(i)
8 }
```

☐

0
1
2
3
4
5
6
7
8
9
10

```
6 for i in 0..<10{
7     print(i)
8 }
```

☐

0
1
2
3
4
5
6
7
8
9

Swift – Ciclos

```
4 var numerosAleatorios: [Int] = []
5
6 for _ in 0..<10{
7     let numero = Int.random(in: 0...100)
8     numerosAleatorios.append(numero)
9 }
10
11 print(numerosAleatorios)
```

□

[24, 5, 87, 57, 70, 6, 21, 69, 93, 55]

- De notar que a variável "i" foi substituída por _ devido a não ser usada.
- Em cada iteração do ciclo é gerado um número aleatório entre 0 e 100 e colocado no array.

Swift – Enum

```
3 enum Curso {
4     case WebDeveloper
5     case MobileDeveloper
6     case LowCodeDeveloper
7 }
8
9 func getDuracaoCurso(de curso:Curso){
10     if curso == .WebDeveloper{
11         print(1000)
12     } else if curso == .MobileDeveloper {
13         print(900)
14     } else if curso == .LowCodeDeveloper {
15         print(850)
16     } else {
17         print(0)
18     }
19 }
20
21 getDuracaoCurso(de: .WebDeveloper)
```

1000

- Enum permitem fazer uma validação inicial sobre um conjunto de valores, permitindo a seleção apenas de um valor dentro de um conjunto.

Swift – Enum

- É também possível associar valores a um enumerado (rawValue), o que pode permitir diminuir e melhorar a lógica de uma aplicação.

```
3 enum Curso : Int{
4     case WebDeveloper = 1000
5     case MobileDeveloper = 900
6     case LowCodeDeveloper = 850
7 }
8
9 func getDuracaoCurso(de curso:Curso){
10     print(curso.rawValue)
11 }
12
13 getDuracaoCurso(de: .MobileDeveloper)
```

900


Swift – Operadores

3	<code>let n1 = 10</code>	<input type="checkbox"/> 10
4	<code>let n2 = 15</code>	<input type="checkbox"/> 15
5		
6	<code>n1 + n2</code>	<input type="checkbox"/> 25
7	<code>n1 - n2</code>	<input type="checkbox"/> -5
8	<code>n1 * n2</code>	<input type="checkbox"/> 150
9	<code>n1 / n2</code>	<input type="checkbox"/> 0
10	<code>n1 % n2</code>	<input type="checkbox"/> 10
11		
12	<code>n1 > n2</code>	<input type="checkbox"/> false
13	<code>n1 <= n2</code>	<input type="checkbox"/> true
14	<code>n1 != n2</code>	<input type="checkbox"/> true
15	<code>n1 == n2</code>	<input type="checkbox"/> false

Swift – Optionals

- Um optional é um valor que pode assumir o valor nulo, por exemplo

```
2  
3 var idades: [Int] = []  
4 idades.sort()  
5  
6 let maisVelho = idades.last  
7
```



The image shows a snippet of Swift code in a dark-themed editor. To the right of the code, there is a vertical stack of three variable inspection boxes. The first box corresponds to the variable `idades` and shows its value as an empty array `[]`. The second box corresponds to the `sort()` method call and also shows `[]`. The third box corresponds to the variable `maisVelho` and shows its value as `nil`, illustrating that an array's `last` property returns `nil` when the array is empty.

Swift – Optionals

- Para resolver a situação temos as opções:
 - if let
 - nil coalescing
 - guard statement
 - force unwrap (perigoso, a evitar)

Swift – Optionals (if let)

```
3 var idades: [Int] = []
4 idades.sort()
5
6 if let maisVelho = idades.last {
7     print("O mais velho é \(maisVelho)")
8 }else{
9     print("Não foram definidas idades")
10 }
11
```

Debugger Output:

- Line 3: []
- Line 4: []
- Line 9: "Não foram definidas idades\n"

Console Output:

Não foram definidas idades

Swift – Optionals (nil coalescing)

```
2  
3 var idades: [Int] = []  
4 idades.sort()  
5  
6 let maisVelho = idades.last ?? 999  
7
```



Swift – Optionals (guard statement)

```
2
3 var idades: [Int] = []
4 idades.sort()
5
6 func getMaisVelho(){
7     guard let maisVelho = idades.last else { return }
8
9     print("O mais velho é \(maisVelho)")
10 }
11
12 getMaisVelho()
```

```
3 var idades: [Int] = [1,4,12,3]
4 idades.sort()
5
6 func getMaisVelho(){
7     guard let maisVelho = idades.last else { return }
8
9     print("O mais velho é \(maisVelho)")
10 }
11
12 getMaisVelho()
```

O mais velho é 12

Swift – Optionals (force unwrap)

```
3 var idades: [Int] = []  
4 idades.sort()  
5  
6 let maisVelho = idades.last!
```

error: Execution was interrupted,

```
2  
3 var idades: [Int] = [3,12,4]  
4 idades.sort()  
5  
6 let maisVelho = idades.last!
```

[3, 12, 4]
[3, 4, 12]
12


Swift – Classes

- Classes permitem a criação de objetos com informação específica de cada um. No primeiro caso vamos definir o objeto Pessoa com os parâmetros “nome”, “email” e “idade”. Criamos também o construtor para atribuir os valores à Pessoa quando criada.

Swift – Classes

```
1 import UIKit
2
3 class Pessoa {
4     var nome:String
5     var email:String
6     var idade:Int
7
8     init(nome:String, email:String, idade:Int){
9         self.nome = nome
10        self.email = email
11        self.idade = idade
12    }
13 }
14
15 let p1 = Pessoa(nome: "Maria", email: "maria@email.com", idade: 30)|
```

nome "Maria"
email "maria@email.com"
idade 30

 Pessoa

Swift – Classes

- Para permitir criar um objeto sem parâmetros (construtor por omissão) colocamos os parâmetros do mesmo como opcionais e indicamos o construtor vazio.

Swift – Classes

```
1 import UIKit
2
3 class Pessoa {
4     var nome:String?
5     var email:String?
6     var idade:Int?
7
8     init(){}
9
10    init(nome:String, email:String, idade:Int){
11        self.nome = nome
12        self.email = email
13        self.idade = idade
14    }
15 }
16
17 let p1 = Pessoa(nome: "Maria", email: "maria@email.com", idade: 30)
18 let p2 = Pessoa()
```

nil
nil
nil

☐ Pessoa
☒ Pessoa

Swift – Classes

- Podemos ainda aceder a cada um dos parâmetros para leitura ou escrita

Swift – Classes

```
16
17 let p1 = Pessoa(nome: "Maria", email: "maria@email.com", idade: 30)
18 p1.nome = "Joana"
19
20 print(p1.nome)
```

Expression implicitly coerced from 'String?' to 'Any'

- ☐ Pessoa
- ☐ Pessoa
- ☐ "Optional("Joana")\n"

```
17 let p1 = Pessoa(nome: "Maria", email: "maria@email.com", idade: 30)
18 p1.nome = "Joana"
19
20 if let nome = p1.nome {
21     print(nome)
22 }
```

Joana

- ☐ Pessoa
- ☐ Pessoa
- ☐ "Joana\n"

Opcional na
classe;

If let para o
valor

Swift – Classes

- É também possível criar funções específicas para cada classe

Swift – Classes

```
2
3 class Pessoa {
4     var nome:String?
5     var email:String?
6     var idade:Int?
7
8     init(){}
9
10    init(nome:String, email:String, idade:Int){
11        self.nome = nome
12        self.email = email
13        self.idade = idade
14    }
15
16    func escreverNome(){
17        print(nome)
18    }
19
20 }
21
22 let p1 = Pessoa(nome: "Maria", email: "maria@email.com", idade: 30)
23
24 p1.escreverNome()
```

⏮

Optional("Maria")

⚠ Expression implicitly coerced from 'String?' to 'Any'

Swift – Herança

- Herança permite a criação de objetos com a especificação de atributos e/ou funções herdando elementos de uma classe principal.

Swift – Herança

```
3 class Pessoa {
4     var nome:String?
5     var email:String?
6     var idade:Int?
7
8     init(){}
9
10    init(nome:String, email:String, idade:Int){
11        self.nome = nome
12        self.email = email
13        self.idade = idade
14    }
15
16    func escreverNome(){
17        print(nome)
18    }
19 }
20
21 class Aluno: Pessoa {
22     var numero: Int?
23
24     override func escreverNome(){
25         print("\(nome) - \(numero)")
26     }
27 }
28
29 let p1 = Aluno(nome: "João", email: "joao@email.pt", idade: 25)
30 p1.numero = 1
31
32 p1.escreverNome()
```

Expression implicitly coerced from 'String?' to 'Any'

String interpolation produces a debug description of an optional value

Swift – Struct

- Quando criamos um objeto a partir de um outro estes ficam interligados (reference type), no entanto se quisermos que o novo objeto tenha apenas uma cópia do primeiro (value type) necessitamos de criar o objeto como uma Struct

Swift – Struct

```
3 class Pessoa {  
4     var nome:String?  
5     var email:String?  
6     var idade:Int?  
7  
8     init(){}  
9  
10    init(nome:String, email:String, idade:Int){  
11        self.nome = nome  
12        self.email = email  
13        self.idade = idade  
14    }  
15  
16    func escreverNome(){  
17        print(nome) ⚠ Expression implicitly coerced from 'String?' to 'Any'  
18    }  
19 }  
20  
21 var p1 = Pessoa(nome: "João", email: "joao@email.pt", idade: 25)  
22  
23 var p2 = p1  
24  
25 p1.nome = "Manuel"  
26 p2.nome
```

☐ Pessoa
☐ Pessoa
☐ Pessoa
☐ "Manuel"

```
3 struct Pessoa {  
4     var nome:String?  
5     var email:String?  
6     var idade:Int?  
7  
8     init(){}  
9  
10    init(nome:String, email:String, idade:Int){  
11        self.nome = nome  
12        self.email = email  
13        self.idade = idade  
14    }  
15  
16    func escreverNome(){  
17        print(nome) ⚠ Expression implicitly coerced from 'String?' to 'Any'  
18    }  
19 }  
20  
21 var p1 = Pessoa(nome: "João", email: "joao@email.pt", idade: 25)  
22  
23 var p2 = p1  
24  
25 p1.nome = "Manuel"  
26 p2.nome
```

☐ Pessoa
☐ Pessoa
☐ Pessoa
☐ Pessoa
☐ "João"

Swift – Extension

- Quando queremos adicionar uma nova funcionalidade a um elemento específico dentro da nossa aplicação podemos utilizar uma extension. Neste caso vamos adicionar ao tipo de dados String a possibilidade de remover os espaços em branco da mesma

Swift – Extension

```
3 extension String{
4     func removerEspacosBranco() -> String {
5         return components(separatedBy: .whitespaces).joined()
6     }
7 }
8
9 let nome = "Bruno Filipe Santos"
10
11 print(nome.removerEspacosBranco())
```



BrunoFilipeSantos

Exercícios

- Crie um Swift Playground ou utilize um Replit (<https://replit.com/>) para os seguintes exercícios:
 1. Crie um algoritmo que lê o nome de uma pessoa e escreve “Olá” seguido do nome da pessoa.
 2. Crie um algoritmo que após ler dois números inteiros apresenta a sua soma.
 3. Pretende-se lendo a base e altura de um triângulo calcular a sua área.

Exercícios

- Crie um Swift Playground ou utilize um Replit (<https://replit.com/>) para os seguintes exercícios:
4. Faça um programa que receba 3 valores que representarão os lados de um triângulo e verifique se os valores formam um triângulo e classifique esse triângulo como:
- Equilátero (3 lados iguais);
 - Isósceles (2 lados iguais);
 - Escaleno (3 lados diferentes).

De notar que para formar um triângulo:

- Nenhum dos lados pode ser igual a zero;
- Um lado não pode ser maior do que a soma dos outros dois;

Exercícios

- Crie um Swift Playground ou utilize um Replit (<https://replit.com/>) para os seguintes exercícios:
5. Dado uma série de 20 valores inteiros, faça um algoritmo que calcule e escreva a média aritmética destes valores.
 6. Faça um programa que receba 10 valores inteiros e os coloque em um vetor. Em seguida exiba-os em ordem inversa à ordem de entrada.