Apple Developer Academy ✦ POA

#Foundations2025

Swift

A new way to code

Felipe Elsner, Dev Mentor

Swift Content

#Foundations2025

Agenda

1. Constants and Variables ✅
2. Basic Types ✅
3. Decision Making ✅
4. Collections ✅
5. Loops ✅
6. Functions
7. Classes, Structs and Enums
8. Optionals

Functions

- Definition
  - A **block of code** that performs a **specific task** and can be called repeatedly throughout the program.

Functions

#Foundations2025

```
  1       2     A   3           4     B         5       A  C       6       D
func funcName(argument parameter: ParameterType) -> ReturnType {
       7
   (. . .)
D
}
```

1. **func** Key word
2. Name/Identifier
3. Argument
4. Parameter
5. Parameter Type
6. Return Type
7. Function Body

A. Parentheses **( )**

B. Colon **:**

C. Arrow **->**

D. Braces **{ }**

Functions


```
func funcName(_ parameter: ParameterType) -> ReturnType {
    (. . .)
}

func funcName(parameter: ParameterType) -> ReturnType {
    (. . .)
}
```

Functions


```
func funcName() -> ReturnType {
    (. . .)
}

func funcName() {
    (. . .)
}
```

Functions


```
func funcName() -> ReturnType {
    (. . .)
    return returnValue
}

func funcName() -> ReturnType {
    returnValue
}
```

Class & Struct

#Foundations2025

- **Class** -> Reference
  - Every time a class is called, it **copies the reference** from its original source.

- **Struct** -> Value
  - Every time a struct is called, it **copies the values** from its original source.

Class & Struct

#Foundations2025

```
class Name {

 var propertyName: Type = value
 (. . .)

 func funcName() {
 (. . .)
 }

 (. . .)

}
```

```
struct Name {

 var propertyName: Type = value
 (. . .)

 func funcName() {
 (. . .)
 }

 (. . .)

}
```

Class & Struct

#Foundations2025

Initializer

```
struct Name {

 var propertyName: Type = value
 (. . .)

 func funcName() {
 (. . .)
 }

 (. . .)

}
```

```
let example = Name()

example.funcName()
```

Class & Struct

#Foundations2025

Initializer

```
struct Name {

 var propertyName: Type
 (. . .)

 func funcName() {
 (. . .)
 }

 (. . .)

}
```

```
let example = Name(propertyName: value)

example.funcName()
```

Class & Struct

#Foundations2025

Initializer

```
class Name {

 var propertyName: Type
 (. . .)

 func funcName() {
 (. . .)
 }

 (. . .)

}
```

```
class Name {

 var propertyName: Type
 (. . .)

 init(propertyName: Type) {
 self.propertyName = propertyName
 }

 func funcName() {
 (. . .)
 }

}
```

Class & Struct

Initializer

```
class Name {

 var propertyName: Type
 (. . .)

 init(propertyName: Type) {
 self.propertyName = propertyName
 }

 func funcName() {
 (. . .)
 }

}
```

```
let example = Name(propertyName: value)

example.funcName()
```

Class & Struct

Initializer

```
class Name {

 var propertyName: Type
 (. . .)

 init(propertyName: Type) {
 self.propertyName = propertyName
 }

 func funcName() {
 (. . .)
 }

}
```

```
struct Name {

 var propertyName: Type
 (. . .)

 //implicit init

 func funcName() {
 (. . .)
 }

}
```

Class & Struct

Initializer

```
class Name {

 var propertyName: Type
 (. . .)

 init(propertyName: Type) {
 self.propertyName = propertyName
 }

 func funcName() {
 (. . .)
 }

}
```

```
struct Name {

 var propertyName: Type
 (. . .)

 //implicit init

 func funcName() {
 (. . .)
 }

}
```

Inheritance

- **What is inheritance used for?**
  - To make the code more concise by reusing code.

- **Structs** do not support inheritance.
- Only **classes** can inherit features from other **classes**.

Inheritance

Example

#Foundations2025

Classe

```swift
class Pessoa {
  let nome: String
  let sobrenome: String
  var idade: Int

  init(nome: String, sobrenome: String, idade: Int) {
      self.nome = nome
      self.sobrenome = sobrenome
      self.idade = idade
  }

  func saudar() {
    print("\(nome) diz: Olá!")
  }

}
```

Class - Inheritance

```swift
class PessoaFisica: Pessoa {
  let cpf: String

  init(nome: String, sobrenome: String, idade: Int, cpf: String) {
    self.cpf = cpf
    super.init(nome: nome, sobrenome: sobrenome, idade: idade)
  }

  override func saudar() {
    super.saudar()
    print("Meu CPF é \(cpf)")
  }
}
```

Enum

- **Enum**eration
  - A type that lets you define a group of related **values** in a **type-safe** way.
  - Is used when you have a set of **known, fixed options**.

Enum


```
enum Name {

 case name1
 case name2

 (. . .)
}
```

```
enum Name: Type {

 case name1 = value1
 case name2 = value2

 (. . .)
}
```

```
enum Name {

 case name1
 case name2

 (. . .)

 func funcName() {

 }

}
```

Optionals

#Foundations2025

A variable **might have a value**, or it **might be nil** (nothing)

- **Why to use?**
  - Accessing a missing value could cause a crash
  - Swift forces to handle with the **possibility of nil** safely.

Optionals


```
var varName: Type?

var varName: Type? = nil

var varName: Type? = value

var varNam  = nil
```

A variable needs a defined type

Optionals

Unwrapping

If let **Guard let** **If statement** **Forced Unwrapping**

Optionals

Unwrapping

If let

```
if let varName = optional {

(. . .)

}
```

Guard let

```
guard let varName = optional else {

(. . .)
return

}

(. . .)
```

Optionals

Unwrapping

If statement

```
if optional != nil {

(. . .)

}
```

Optionals
Unwrapping

Forced Unwrapping

```
optional!
```

Optionals

Unwrapping

`If let` `Guard let`

✅ Pros

- Safe Unwrapping
- Crash Free

⚠️ Cons

- More Verbose

`If statement`

✅ Pros

- Safe Checking
- Crash Free

⚠️ Cons

- Less Verbose, but still somewhat verbose
- The variable **remains optional**

`Forced Unwrapping`

✅ Pros

- Least Verbose

⚠️ Cons

- Unsafe Handling
- **High** risk of Crash

Swift Content

#Foundations2025

Agenda

1. Constants and Variables ✅
2. Basic Types ✅
3. Decision Making ✅
4. Collections ✅
5. Loops ✅
6. Functions ✅
7. Classes, Structs and Enums ✅
8. Optionals ✅

Apple Developer Academy ✦ POA

#Foundations2025

Swift

A new way to code

Felipe Elsner, Dev Mentor