

38

Structures de données

Tableau
Ensemble
Dictionnaire
Structure (struct)
Énumérations
...

10/02/2025

38

Tableau

39

- Un tableau est une collection ordonnée d'éléments du même type.

```
var nombres: [Int] = [1, 2, 3, 4, 5]
```

```
// Ajouter un élément  
nombres.append(6)
```

```
// Accéder à un élément  
let premierNombre = nombres[0]
```

```
// Parcourir le tableau  
for nombre in nombres {  
    print(nombre)  
}
```

```
// Supprimer un élément  
nombres.remove(at: 2)
```

10/02/2025

39

Les tableaux

40

Les tableaux

- Déclaration d'un tableau en swift
 - ▣ **let** tableau = [valeur1, valeur2, ..., valeurN]
 - ▣ **var** tableauEntiers: [Int] // il faut pas oublier l'initialization du tableau
 - ▣ **Var** tableauNoms:[String]
 - ▣ Var T=[Int]()

Exemple:

```
let prenom = ["Sara", "Ali", "Mohammed", "Maha"]
print(prenom[2])
```

10/02/2025

40

41

```
var prenom = ["Sara", "Ali", "Mohammed", "Maha"]
// On ajoute en fin de tableau
prenom = prenom + [ 'Nada']
print(prenom)
// Ou bien encore en début du tableau
prenom = ["Jean"] + prenom
print(prenom)
// pour concaténer plus rapidement en fin de tableau
prenom += [ 'Nada']
```

10/02/2025

41

42

□ Parcourir le tableau

```
let Tabprenoms=["sara","Maha","mohammed","Ali"]
var i:Int8
for i in 0..

```

```
for A in Tabprenoms{
  print(A)
}
```

10/02/2025

42

43

□ Quelques fonctions qui gèrent les tableaux:

```
var name=["sara","toto","pim"]
  name.insert("lala", at: 1)
  print(name)
  name.remove(at: 0)
  print(name)
  name.index(of: "sara") (nil s'il n exist pas)
```

10/02/2025

43

44

```

for i in 0..

```

10/02/2025

44

Ensembles (Set)

45

- Un ensemble est une collection non ordonnée d'éléments uniques.

```

var fruits: Set<String> = ["Pomme", "Banane", "Orange"]

// Ajouter un élément
fruits.insert("Kiwi")

// Vérifier l'existence d'un élément
if fruits.contains("Banane") {
    print("Banane est présent !")
}

// Parcourir l'ensemble
for fruit in fruits {
    print(fruit)
}

```

10/02/2025

45

Dictionnaires (Dictionary)

46

- Un dictionnaire est une collection de paires clé-valeur, où chaque clé est unique.


```
var capitales: [String: String] = [
    "France": "Paris",
    "Italie": "Rome",
    "Espagne": "Madrid"
]
// Ajouter ou modifier une valeur
capitales["Allemagne"] = "Berlin"
// Accéder à une valeur
if let capitaleFrance = capitales["France"] {
    print("La capitale de la France est \(capitaleFrance).")
}
// Parcourir le dictionnaire
for (pays, capitale) in capitales {
    print("\(pays): \(capitale)")
}
```

10/02/2025

46

47

Les dictionnaires


var dictionnaire: [type1: type2]
 // type1 peut être identique à type2

// Par exemple

```
var dictionnaire: [String: Int]
let dictionnaire :Dictionary<Int, String> = [:]
```

Dans le dictionnaire:

- les valeurs ne se rangeront plus dans un ordre spécifique (de 0 à n pour les indices du tableau).

- Une clé est unique.

10/02/2025

47

48

□ Parcourir le dictionnaire

```
let dic=["Nom":"xxx", "Prenom":"sara", "Adresse":"yyy  
45 yyy", "Ville":"Rabat"]  
for (cle,valeur) in dic{  
    print(cle + " : " + valeur)  
}
```

10/02/2025

48

49

□ Déclarer un dictionnaire vide

```
var monDictionnaireVide = [String: Int]()  
Ou  
var monDictionnaireVide: [String: Int] = [:]
```

■ Ajouter et supprimer des éléments

```
var countries = ["FR": "France", "IT": "Italie", "UK": "United King"]  
// Je modifie la valeur, car la clé "UK" existe déjà  
countries["UK"] = "United Kingdom"  
// J'ajoute une valeur, car la clé "ES" n'existe pas  
countries["ES"] = "Espagne"
```

```
countries.removeValue(forKey: "ES")  
// L'association "ES": "Espagne" est supprimée 10/02/2025
```

49

50

```
Var D= [ "France": "Paris", "Chine":"Pékin", "
Maroc": " Rabat"]
let dictKeys = Array(D.keys)

■print(dictKeys)

["Chine", "Maroc", "France"]
```

10/02/2025

50

Structures (struct) et Classes (class)

51

```
struct Personne {
var nom: String
var age: Int

func sePresenter() {
print("Je m'appelle \ \(nom) et j'ai \ \(age) ans.")
}
}

let personne = Personne(nom: "Ali", age: 22)
personne.sePresenter()
```

10/02/2025

51

Structures (struct) et Classes (class)

52

```
class Animal {
    var nom: String
    var espece: String

    init(nom: String, espece: String) {
        self.nom = nom
        self.espece = espece
    }

    func faireDuBruit() {
        print("\(nom) fait un bruit !")
    }
}

let chien = Animal(nom: "Rex", espece: "Chien")
chien.faireDuBruit()
```

10/02/2025

52

Énumérations (enum)

53

- Les énumérations permettent de définir un type avec un ensemble de valeurs prédéfinies.

```
enum Direction {
    case nord
    case sud
    case est
    case ouest
}

let direction = Direction.nord

switch direction {
    case .nord : print("On va vers le nord !")
    case .sud  : print("On va vers le sud !")
    case .est, .ouest: print("On va vers l'est ou l'ouest !")
}
```

10/02/2025

53

Les tuples

54

- un tuple peut contenir plusieurs valeurs de n'importe quel type. Une variable ou une constante peut être un tuple.
 - ▣ `let` tuple = (valeur1, valeur2, ...)
 - ▣ `var` tuple = (valeur1, valeur2, ...)
- `let` JoursSemaine = ("lundi", "Mardi", ...)
- `var` tuple = (2.5, "bonjour", 2, ...)

10/02/2025

54

Les fonctions en Swift

55

```
func Nomfonction(parametre1: Type, parametre2: Type, ...)
{ // Instructions
}
```

```
func nomDeLaFonction(parametre1: Type, parametre2: Type, ...) ->
TypeRetour
{
    // Instructions

    return laValeurARetourner
}
```

10/02/2025

55

56

- Lorsqu'on appelle une fonction qui a plusieurs paramètres, on doit préciser le nom du paramètre utilisé lors de la déclaration, à partir du deuxième.

Nomfonction(valeur1, paramètre2: valeur2, ...)

10/02/2025

56

Les fonctions en Swift

57

Pour Swift on peut attribuer **2 mots** à un paramètre:

Exemple:

```
func disBonjour(A prenom: String) {
    print("Bonjour " + prenom + " !")
}
```

Pour appeler la fonction on l'appelle avec **le premier** nom du paramètre:
disBonjour(A: "Sara")

10/02/2025

57

58

On peut aussi **omettre** le nom du paramètre lorsqu'on appelle la fonction

```
func disBonjour(_ prenom: String) {
    print("Bonjour " + prenom + " !")
}
disBonjour( "Sara ")
```

10/02/2025

58

59

□ Fonction passer comme paramètre ou type de retourne

- Pour indiquer une fonction en tant que paramètre d'une fonction, il faut mettre sa déclaration dans les parenthèses de la fonction.

```
func nom_fonction(fonct : (type para .)->type retour)->type retour
```

Pour appeler la fonction:

```
No_fonction(nom_fonction_paramètre)
```

10/02/2025

59

Fonction de saisi

60

- // Fonction permettant de demander à l'utilisateur d'entrer un nombre et de le convertir en entier

```
func input() -> Int {
    let strData = readLine();
    return Int(strData!!)
}
```

```
var nombreEntre: Int
nombreEntre = input()
```

10/02/2025

60

61

- print("Please enter number 1")
- var num1 = Int(readLine()!!) print("Please enter number 2")
- var num2 = Int(readLine()!!)
- var sum = num1 + num2 print("The sum of \ (num1) and \ (num2) is \ (sum)")

10/02/2025

61

Les fermetures

62

- Une fermeture (closure) est un bloc d'instruction qui n'est pas nommé.
- C'est une fonction sans nom, qui possède des paramètres et une valeur de retour
- L'avantage est que ce bloc peut être passé en paramètre d'une fonction, c'est une seconde alternative au passage d'une fonction, en tant que paramètre, en Swift, ou vous pouvez la stocker comme une propriété d'un objet.

10/02/2025

62

63

□ La flexibilité

Vous avez déjà appris que les fonctions peuvent être incroyablement puissantes et flexibles. Parce que les **fonctions sont des fermetures**, les fermetures sont aussi flexibles.

Cependant, toutes les fermetures ne sont pas une fonction

10/02/2025

63

64

- En Swift , une fermeture se déclare comme ceci:

```
{ (paramètres) -> Type Retour in
    // Vos instructions + return
}
```

- Une fermeture commence et se termine par un accolade, enroulant les paramètres, le type de retour et le corps de fermeture.
- Lorsqu'on utilise une fonction qui prend en paramètre une autre fonction. On peut directement mettre tous le bloc ci-dessous dans les paramètres.

10/02/2025

64

65

```
□ Exemple:

□ // Déclarez la variable myVar1, avec le type de données, et assignez la valeur.
□ var myVar1 : () -> () = {
□
□     print("Hello from Closure 1");
□ }
□
□ // Déclarez la variable myVar2, avec les types de données, et assignez une valeur.
□ var myVar2 : () -> (String) = { () -> (String) in
□
□     return "Hello from Closure 2"
□
□ }
□
□ // Déclarez la variable myVar3, avec les types de données, et assignez la valeur.
□ var myVar3 : (Int, Int) -> (Int) = { (a : Int, b: Int) -> (Int) in
□
□     var c : Int = a + b
□
□     return c
□ }
```

10/02/2025

65

66

```

□ func test_closure() {
□
□ // Exécutez de la fermeture (Closure).
□ myVar1()
□
□ // Exécutez de la Closure et obtenir la valeur de retour.
□ var str2 = myVar2()
□
□ print(str2)
□
□ // Exécutez de la Closure, passez (pass) les paramètres
□ // et obtenez une valeur de retour.
□ var c: Int = myVar3(11, 22)
□
□ print(c)
□
□ }

```

10/02/2025

66

67

□ Fonction avec fermeture (closure)

Fonction est un cas particulière de Closure. Fonction est une Closure nommé on peut dire que la Closure est une Fonction anonyme.

□ Exemple :

10/02/2025

67

68

```
func sum(a: Int, b: Int) -> Int {
  return a + b
}
```

Function

```
{ (a: Int, b: Int) -> Int in
  return a + b
}
```

Closure

```
func sum(a: Int, b: Int) -> Int {
  return a + b
}
```

```
var value = sum(1, 2)
```

```
var myClosure = { (a: Int, b: Int) -> Int in
  return a + b
}
```

```
var value = myClosure(1, 2)
```

10/02/2025

68

69

□ La fermeture anonyme

Quand on déclare une Fermeture, c'est pas nécessaire d'écrire le nom des paramètres. Ces paramètres peuvent être référencés par \$0, \$1,...

Exemple 1:

10/02/2025

69

70

```
1 import Foundation
2
3 // Déclarez une Closure de la manière habituelle.
4 var myClosure : (String, String) -> String
5
6     = { (firstName: String, lastName: String) -> String in
7
8         return firstName + " " + lastName
9     }
10
11 // Déclarez une Closure de manière anonyme (anonymous).
12 // (Ignorez les noms de paramètres).
13 var anonymousClosure : (String, String) -> String
14
15     = {
16         // En utilisant
17         // $0: Pour le premier paramètre
18         // $1: Pour le deuxième paramètre.
19         return $0 + " " + $1
20     }
21 }
```

10/02/2025

70

71

- Note: \$0, \$1,... sont des paramètres anonymes. Ils sont uniquement utilisés dans la Fermeture Anonyme. Si vous utilisez la Fermeture normal, vous recevrez un message d'erreur:
- Les arguments de fermeture anonymes ne peuvent pas être utilisés dans une fermeture comportant des arguments explicites

10/02/2025

71

72

□ Exemple 2:

```
import Foundation

func test_anonymousClosure() {
    // Déclarez une variable du type de Closure.
    var mySum : ( Int, Int ) -> (Int)

    // Assignez une Closure anonyme pour lui.
    // $0: Pour le premier paramètre
    // $1: Pour le deuxième paramètre.
    mySum = {
        return $0 + $1
    }

    var value = mySum(1, 2)
    print(value)
}
```

10/02/2025

72

Les énumérations

73

- Une énumération permet de définir un type commun pour une liste de valeurs personnalisées.
- De la même manière que la déclaration **switch**, **enum's** dans Swift peut à première vue ressembler à une variante légèrement améliorée de la déclaration bien connue **enum**
- les énumérations de Swift lui permettent d'être utilisées dans un éventail de scénarios pratiques beaucoup plus large que les énumérations ordinaires. En particulier, ce sont d'excellents outils pour manifester clairement les intentions de votre code.

10/02/2025

73

74

- "Les énumérations déclarent des types avec des ensembles finis d'états possibles et de valeurs correspondantes. Avec l'imbrication, les méthodes, les valeurs associées et la correspondance de modèle, les énumérations peuvent définir des données organisées de manière hiérarchique."

10/02/2025

74

75

- Définir les énumérations de base

```
enum NomEnumeration{  
    case valeur1  
    case valeur2  
    ...  
}
```

```
enum NomEnumeration{  
    case valeur1, valeur2, ...  
}
```

10/02/2025

75

76

- Pour créer des variable de type énumération:

- * `Var V=NomEnumeration.valeur`

- * `V=.valeur`

On peut l'utiliser dans `switch case`:

```
switch V{
    case.valeur1:action1
    case.valeur2:action2
    ...
}
```

10/02/2025

76

77

Exemple:

- Voir exemple1 énumération

```
enum Mouvement{
    case gauche
    case droit
    case haut
    case bas
}
```

10/02/2025

77

78

Exemple:

- ▣ Voir exemple1 énumération

```
enum IOSType{
    case iPhone
    case iPad
    case iWatch
}

Var myDevice=IOSType.iPhone
```

10/02/2025

78

79

- ▣ Si le type d'une énumération est connu ou peut être déduit, vous pouvez utiliser la syntaxe à points pour les membres.

```
// dans ce cas le type est connu
var myDevice: IOSType = .iPhone
```

```
// dans ce cas le type peut être déduit
if myDevice == .iPhone { print("'ai un iPhone!") }
```

10/02/2025

79

80

- Les énumérations Swift peuvent stocker des valeurs associées de tout type et le type de valeur peut être différent pour chaque membre. Par exemple, vous souhaitez peut-être stocker un modèle d'appareil pour iPhone et iPad (par exemple, "mini" pour l'iPad ou "6 Plus" pour l'iPhone).

10/02/2025

80

81

```
enum IOSType{  
    case iPhone  
    case iPad  
    case iWatch  
}
```

10/02/2025

81

Les valeurs associées

82

Exemple:

```
enum IOSType{  
    case iPhone(String)  
    case iPad(String)  
    case iWatch  
}
```

```
var myDevice = IOSType.iPhone("6")
```

10/02/2025

82

83

- Vous pouvez obtenir les valeurs associées en utilisant une instruction switch:

```
var myDevice = IOSType.iPhone("6")
```

```
switch myDevice {  
    case .iPhone(let model): print("iPhone \(model)")  
    case .iPad(let model): print("iPad \(model)")  
    case .iWatch: print("iWatch")  
    default: print("not an iOS device")  
}
```

10/02/2025

83

Les valeurs associées

84

- Les valeurs associées sont un moyen fantastique d'attacher des informations supplémentaires à une énumération.

```
enum codes {
    //code-bar
    case UPCA(Int,Int,Int,Int)
    case Qrcode(String) }

Var code=codes.UPCA(3,98987,1,0)
code=.Qrcode(" http//.....")
```

10/02/2025

84

Valeurs Enum

85

- On peut affecter des valeurs aux **cas** des énumérations: **valeurs brutes:**

- ▣ Il faut définir un type natif unique pour tous l' énumération
- ▣ Chaque cas doit avoir une seule et unique valeur
- ▣ La valeur sera de même type pour tous les cas

(voir exemple 2)

10/02/2025

85