Swift

Bases du langage

Variables

- Commentaires : // ou /* */
- ; facultatifs en fin d’instruction
- Constantes : let maximumNumberOfLoginAttempts = 10
  - let π = 3.14159
  - let 你好 = "你好世界"
  - let 🐶🐮 = "dogcow"
- Variables : var currentLoginAttempt = 0
  - var x = 0.0, y = 0.0, z = 0.0
- Type : var welcomeMessage: String
  - var red, green, blue: Double
  - String, Double, Float, Int32, Int64, UInt32, UInt64, Boolean, Character
- Wrapper : Int(), Double()…

Variables

- Tuple : let http404Error = (404, "Not Found")
- Optional : var serverResponseCode: Int? = 404
  - Null => nil

let possibleString: String? = "An optional string."

let forcedString: String = possibleString! // requires an exclamation point

let assumedString: String! = "An implicitly unwrapped optional string."

let implicitString: String = assumedString // no need for an exclamation point

  - ! : supprimer la nécessité de vérifier et de déballer la valeur de l'option facultative à chaque fois qu'elle y accède, car on peut supposer qu'elle a une valeur en permanence

- Gestion des erreurs : throws sur une fontion ou
  - try canThrowAnError() // no error was thrown } catch { // an error was thrown }
- Affichage : print(welcomeMessage)
  - Interpolation : print("The current value of welcomeMessage is \(welcomeMessage)")

Opérateurs

- let b = 10 var a = 5 a = b
- let (x, y) = (1, 2)
- if x = y {// This is not valid, because x = y does not return a value. }
- let rowHeight = contentHeight + (hasHeader ? 50 : 20)
- a != nil ? a! : b
  - Raccourci :
    - let defaultColorName = "red" var userDefinedColorName: String?
    - var colorNameToUse = userDefinedColorName ?? defaultColorName
- 1 + 2 // equals 3
- 5 - 3 // equals 2
- 2 * 3 // equals 6
- 10.0 / 2.5 // equals 4.0
- "hello, " + "world" // equals "hello, world”
- a += 2

Comparaison

- 1 == 1 // true because 1 is equal to 1
- 2 != 1 // true because 2 is not equal to 1
- 2 > 1 // true because 2 is greater than 1
- 1 < 2 // true because 1 is less than 2
- 1 >= 1 // true because 1 is greater than or equal to 1
- 2 <= 1 // false because 2 is not less than or equal to 1
- (1, "zebra") < (2, "apple") // true because 1 is less than 2; "zebra" and "apple" are not compared
- (3, "apple") < (3, "bird") // true because 3 is equal to 3, and "apple" is less than "bird"
- (4, "dog") == (4, "dog") // true because 4 is equal to 4, and "dog" is equal to "dog"

Range

- for index in 1...5 { print("\(index) times 5 is \(index * 5)") }
- let names = ["Anna", "Alex", "Brian", "Jack"]
- let count = names.count
- for i in 0..<count { print("Person \(i + 1) is called \(names[i])") }
- for name in names[2...] { print(name) } // Brian Jack
- for name in names[...2] {print(name)} // Anna Alex Brian
- for name in names[..<2] {print(name)} // Anna Alex
- let range = ...5
- range.contains(7) // false
- range.contains(4) // true
- range.contains(-1) // true

Logique

- let allowedEntry = false

if !allowedEntry { print("ACCESS DENIED") }

- let enteredDoorCode = true let passedRetinaScan = false

if enteredDoorCode && passedRetinaScan { print("Welcome!") } else { print("ACCESS DENIED") }

- let hasDoorKey = false let knowsOverridePassword = true

if hasDoorKey || knowsOverridePassword { print("Welcome!") } else { print("ACCESS DENIED") }

String

- let someString = "Some string literal value »
- let quotation = """

The White Rabbit put on his spectacles. "Where shall I begin,

please your Majesty?" he asked.

"""

- let wiseWords = "\"Imagination is more important than knowledge\" - Einstein"
- let dollarSign = "\u{24}" // $, Unicode scalar U+0024
- let blackHeart = "\u{2665}" // , Unicode scalar U+2665
- var emptyString = "" // empty string literal
- var anotherEmptyString = String() // initializer syntax
- if emptyString.isEmpty { print("Nothing to see here") }
- for character in "Dog!🐶" { print(character) }

String

- let catCharacters: [Character] = ["C", "a", "t", "!", "🐱"] let catString = String(catCharacters)
- let unusualMenagerie = "Koala 🐨, Snail 🐌, Penguin 🐧, Dromedary 🐪" print("unusualMenagerie has \(unusualMenagerie.count) characters")
- let greeting = "Guten Tag!"
- greeting[greeting.startIndex] // G
- greeting[greeting.index(before: greeting.endIndex)] // !
- greeting[greeting.index(after: greeting.startIndex)] // u
- let index = greeting.index(greeting.startIndex, offsetBy: 7)
- greeting[index] // a
- for index in greeting.indices { print("\(greeting[index]) ", terminator: "") } // Prints "G u t e n T a g ! "
- var welcome = "hello"
- welcome.insert("!", at: welcome.endIndex)
- // welcome now equals "hello!"
- welcome.insert(contentsOf: " there", at: welcome.index(before: welcome.endIndex))
- // welcome now equals "hello there!"

String

- welcome.remove(at: welcome.index(before: welcome.endIndex))
- // welcome now equals "hello there"
- let range = welcome.index(welcome.endIndex, offsetBy: -6)..<welcome.endIndex
- welcome.removeSubrange(range)
- // welcome now equals "hello"
- let greeting = "Hello, world!"
- let index = greeting.firstIndex(of: ",") ?? greeting.endIndex
- let beginning = greeting[..<index]
- // beginning is "Hello"
- // Convert the result to a String for long-term storage.
- let newString = String(beginning)
- let quotation = "We're a lot alike, you and I.”
- let sameQuotation = "We're a lot alike, you and I."
- if quotation == sameQuotation { print("These two strings are considered equal") }

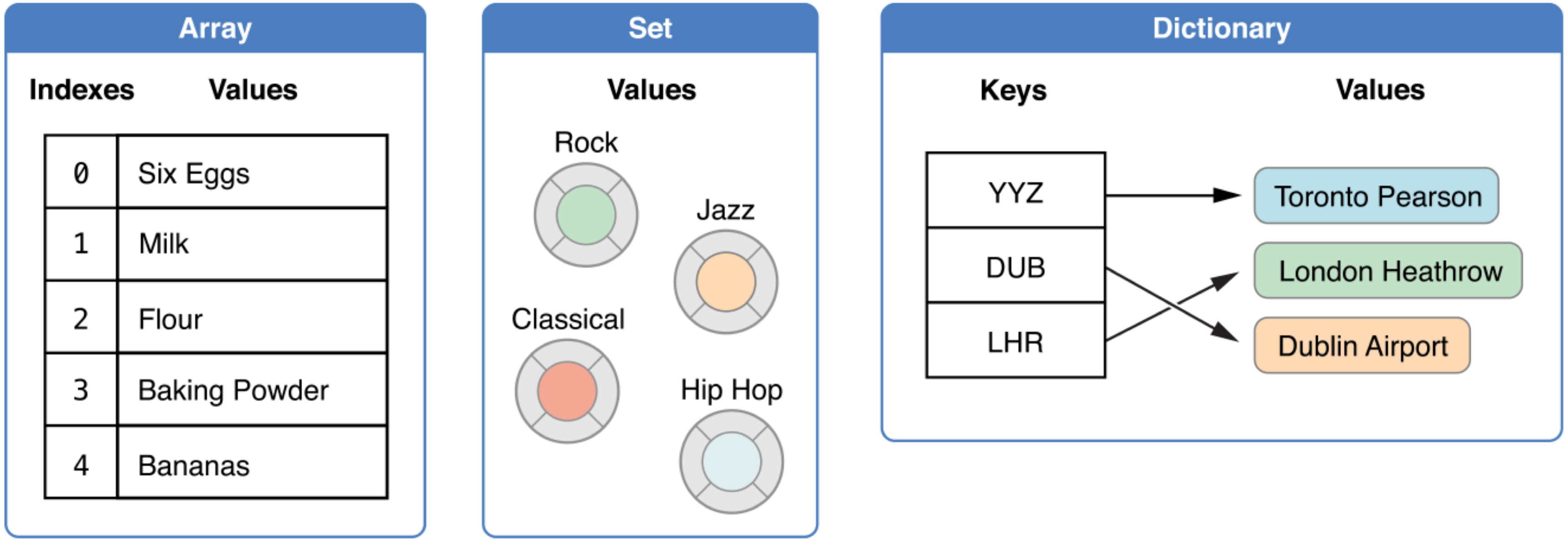
Fonctions

-
  ```
  func greet(person: String) -> String {
    let greeting = "Hello, " + person + "!"
    return greeting
  }
  ```

-
  ```
  func greeting(for person: String) -> String { "Hello, " + person + "!" }
  ```

-
  ```
  func minMax(array: [Int]) -> (min: Int, max: Int) {
    var currentMin = array[0]
    var currentMax = array[0]
    for value in array[1..<array.count] {
        if value < currentMin {
            currentMin = value
        } else if value > currentMax {
            currentMax = value
        }
    }
    return (currentMin, currentMax)
  }
  let bounds = minMax(array: [8, -6, 2, 109, 3, 71]) print("min is \(bounds.min) and max is \(bounds.max)")
  ```

Collections



Array

Indexes	Values
0	Six Eggs
1	Milk
2	Flour
3	Baking Powder
4	Bananas

Set

Values

Rock

Jazz

Classical

Hip Hop

Dictionary

Keys

Values

YYZ

DUB

LHR

Toronto Pearson

London Heathrow

Dublin Airport

Arrays

- var someInts = [Int]()
- print("someInts is of type [Int] with \(someInts.count) items.") // Prints "someInts is of type [Int] with 0 items."
- someInts.append(3) // someInts now contains 1 value of type Int
- someInts = [] // someInts is now an empty array, but is still of type [Int]
- var threeDoubles = Array(repeating: 0.0, count: 3)
  - // threeDoubles is of type [Double], and equals [0.0, 0.0, 0.0]
- var anotherThreeDoubles = Array(repeating: 2.5, count: 3)
  - // anotherThreeDoubles is of type [Double], and equals [2.5, 2.5, 2.5]
- var sixDoubles = threeDoubles + anotherThreeDoubles
  - // sixDoubles is inferred as [Double], and equals [0.0, 0.0, 0.0, 2.5, 2.5, 2.5]
- var shoppingList: [String] = ["Eggs", "Milk"]
- print("The shopping list contains \(shoppingList.count) items.")
- if shoppingList.isEmpty { print("The shopping list is empty.") } else { print("The shopping list is not empty.") }
- shoppingList += ["Chocolate Spread", "Cheese", "Butter"]
- shoppingList[4...6] = ["Bananas", "Apples"]
- shoppingList.insert("Maple Syrup", at: 0)
- let mapleSyrup = shoppingList.remove(at: 0)
- for (index, value) in shoppingList.enumerated() { print("Item \(index + 1): \(value)") }

Exercices

Set

- var letters = Set<Character>()
- print("letters is of type Set<Character> with \(letters.count) items.")
- letters.insert("a")
- var favoriteGenres: Set = ["Rock", "Classical", "Hip hop"]
- if let removedGenre = favoriteGenres.remove("Rock") {
- print("\(removedGenre)? I'm over it.")
- } else { print("I never much cared for that.") }
- if favoriteGenres.contains("Funk") { print("I get up on the good foot.") } else { print("It's too funky in here.") }
- for genre in favoriteGenres.sorted() { print("\(genre)") }

Set

- let oddDigits: Set = [1, 3, 5, 7, 9]
- let evenDigits: Set = [0, 2, 4, 6, 8]
- let singleDigitPrimeNumbers: Set = [2, 3, 5, 7]
- oddDigits.union(evenDigits).sorted() // [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- oddDigits.intersection(evenDigits).sorted() // []
- oddDigits.subtracting(singleDigitPrimeNumbers).sorted() // [1, 9]
- oddDigits.symmetricDifference(singleDigitPrimeNumbers).sorted() // [1, 2, 9]
- let houseAnimals: Set = ["🐶", "🐱"]
- let farmAnimals: Set = ["🐮", "🐔", "🐑", "🐶", "🐱"]
- let cityAnimals: Set = ["🐦", "🐭"]
- houseAnimals.isSubset(of: farmAnimals) // true
- farmAnimals.isSuperset(of: houseAnimals) // true
- farmAnimals.isDisjoint(with: cityAnimals) // true

Dictionnaire

- var namesOfIntegers = [Int: String]() // namesOfIntegers is an empty [Int: String] dictionary
- namesOfIntegers[16] = "sixteen" // namesOfIntegers now contains 1 key-value pair
- namesOfIntegers = [:] // namesOfIntegers is once again an empty dictionary of type [Int: String]
- var airports: [String: String] = ["YYZ": "Toronto Pearson", "DUB": "Dublin"]
- Count, isEmpty, .removeValue(forKey: "DUB")
- for (airportCode, airportName) in airports { print("\(airportCode): \(airportName)") }
- for airportCode in airports.keys { print("Airport code: \(airportCode)") }
- for airportName in airports.values { print("Airport name: \(airportName)") }

Enumération

```
enum CompassPoint {
   case north
   case south
   case east
   case west }
var directionToHead = CompassPoint.west
```

```
enum Beverage: CaseIterable { case coffee, tea, juice }
let numberOfChoices = Beverage.allCases.count
print("\(numberOfChoices) beverages available")
for beverage in Beverage.allCases { print(beverage) }
```

```
enum Barcode { case upc(Int, Int, Int, Int) case qrCode(String) }
var productBarcode = Barcode.upc(8, 85909, 51226, 3)
productBarcode = .qrCode("ABCDEFGHIJKLMNOP")
```

```
enum ASCIIControlCharacter: Character { case tab = "\t" case lineFeed = "\n" case
carriageReturn = "\r" }
```

Structure et classes

- struct SomeStructure { // structure definition goes here }
- struct Size {

  var width = 0.0, height = 0.0 }

let twoByTwo = Size(width: 2.0, height: 2.0)

- class SomeClass { // class definition goes here } //self pour this et : pour héritage
- struct Resolution {

  var width = 0

  var height = 0 }
- class VideoMode {

  var resolution = Resolution()

  var interlaced = false

  var frameRate = 0.0

  var name: String? }
- let someResolution = Resolution()
- let someVideoMode = VideoMode()

if tenEighty === alsoTenEighty { print("tenEighty and alsoTenEighty refer to the same VideoMode instance.") }

Propriétés

- class DataManager {
- lazy var importer = DataImporter()
- var data = [String]()
- // the DataManager class would provide data management functionality here
  }
- let manager = DataManager()
- manager.data.append("Some data")
- manager.data.append("Some more data") // the DataImporter instance for the importer property has not yet been created

Propriétés

```
struct Rect {
  var origin = Point()
  var size = Size()
  var volume: Double { return width * height * depth } //read only
  var center: Point {
        get { let centerX = origin.x + (size.width / 2) let centerY = origin.y + (size.height / 2) return Point(x: centerX, y: centerY) }
        set(newCenter) { origin.x = newCenter.x - (size.width / 2) origin.y = newCenter.y - (size.height / 2) } } }


var square = Rect(origin: Point(x: 0.0, y: 0.0), size: Size(width: 10.0, height: 10.0))
```

Constructeur

- Structure et objet : méthode init()

```
struct Fahrenheit {
    var temperature: Double
    init() { temperature = 32.0 } }
var f = Fahrenheit()
```

- Type optionnel

```
class SurveyQuestion {
    var text: String
    var response: String?
    init(text: String) { self.text = text }
    func ask() { print(text) } }
let beetsQuestion = SurveyQuestion(text: "How about beets?")
beetsQuestion = SurveyQuestion("How about beets?")
```

Destructeur

```
class Player {
        var coinsInPurse: Int
        init(coins: Int) {
                coinsInPurse = Bank.distribute(coins: coins)
        }
        func win(coins: Int) {
                coinsInPurse += Bank.distribute(coins: coins)
        }
        deinit {
                Bank.receive(coins: coinsInPurse)
        }
}
```

Exercices