iOS Workshop: Intro to Swift - Cheatsheet

Written by Adrian Wisaksana (dev@beingadrian.com) for Make School Singapore 2016
Adapted from the original by Jennifer Zhang (jjz@mit.edu) for MIT Blueprint 2015

Variables vs. Constants

- `let` — keyword in front of a name to denote that its a constant (value never changes)
- `var` — keyword in front of a name to denote that its a variable (value can change)

Basic Types

```
// Int – a number without precision (no decimal point; integer value)
let one: Int = 1

// Float – a number with more precision (has values beyond decimal point)
let one: Float = 3.14

// Bool – a value that is either true or false
let truth: Bool = true

// String – text (set of characters) that is incapsulated in quotes
var pokemon: String = "Pikachu"
```

Optionals

- Optionals are special types (indicated by the original type with a question mark) that allow the variable/constant to take the value `nil`

```
var maybe: String? = nil

/**
 * UNWRAPPING OPTIONALS
 */

// force unwrapping (unsafe)
print(maybe!) // if maybe is nil, the program would crash

// old method
if maybe != nil {
  // maybe is not nil here, but still has to be forcefully unwrapped
  print(maybe!)
}

// if let method
if let someValue = maybe {
  // if maybe is not nil, `someValue` is not nil here
  // `someValue` is a constant only accessible in this scope
```

```
} else {
  // maybe is nil
}

// guard let method
guard let someValue = maybe else { return }
// someValue is not nil here
print(someValue) // will only be executed if maybe is not nil
```

Enums

- An enumeration defines a common type for a group of related values and enables you to work with those values in a type-safe way within your code (*Apple Docs*)

```
enum EventType {
  case Sports
  case Formal
  case Casual
}

// shorten syntax (should follow this)
let eventType: EventType = .Sports

// longer syntax (under the hood)
let eventType: EventType = EventType.Sports
```

Classes

- A class is simlar to a blueprint that helps architects build individual objects. We use it to create instances of an object
- **Instance** — a particular concrete occurence of the class
    - i.e. the blueprint of a Car (class), we can build a particular car instance like a Ferarri
- **Properties** — variables or constants that only exist within and are particular to the class
    - i.e. the blueprint of a Car (class) states that cars should have the color red and have four wheels
- **Instance Methods** — functions that belong to a class
    - i.e. the bluprint of a Car (class) has the method `start()` to start the car
- **Initializer** — a special method that constructs an instance of the class
    - i.e. the blueprint of a Car (class) has a special section on how to put the car together

```
enum CarType {
  case Sedan
  case Sports
  case OffRoad
  case Family
}

class Car {

  // properties
  let name: String
  let wheelCount: Int
```

```
  let color: String
  let type: CarType

  // initializer
  init(name: String, wheelCount: Int, color: String, type: CarType) {
    self.name = name
    self.wheelCount = wheelCount
    self.color = color
    self.type = type
  }

}

// creating an instance
let car = Car(name: "Ferarri", wheelCount: 4, color: "Red", type: .Sports)
```

If/Else Statements

```
// the first line is called the condition
// it's what we check to see if it evaluates to be `true`
if name == "Adrian" {
  // if the condition is true, this block will be executed
} else {
  // if the condition is false, this block will be executed
}
```

Arrays

- Arrays are lists/collections of values of the same type
- An array is a SequenceType

```
let pokemons: [String] = ["Mudkip", "Pikachu", "Meowth", "Charizard"]

// Each value in the array can be obtained through the value's index
let firstPokemon = pokemons[0] // array indexes start from 0

// Getting the array count
let pokemonCount = pokemons.count

// Getting the last item of an array
let lastPokemon = pokemons[pokemons.count-1] // -1 because array indexes start form 0
```

Dictionaries

- Dictionaries are also collections of objects like arrays, but instead, we obtain a **value** in a dictionary through a **key**
- Dictionaries store things in **key-value** pairs

```
// a dictionary of words (keys) and definitions (values)
let dictionary: [String: String] = [
```

```
  "life": "the period between birth and death of a living thing",
  "death": "the end of the life of a person or organism"
]

let deathDefinition = dictionary["death"]
print(deathDefinition) // Prints: "the end of the life of a person or organism"
```

For Loops

- For loops run a chunk of code a certain number of times the programmer indicates

```
// Case 1: Code should run a fixed number of times
// 1...5 is a Range<Int> (range of integers) that is a SequenceType
for index in 1...5 {
  print(index)
}
// Prints:
// 1
// 2
// 3
// 4
// 5

// Case 2: Code should run depending on the number of a sequence type
// An array is a Sequence Type
for clothe in clothes {
  print(clothe)
}

// Alternative to for loops: forEach higher order function
clothes.forEach { (clothe) in
  print(clothe)
}
```

Extra Help

- Ask a mentor!
- Apple's Swift Documentation
    - https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/