Cheatography

Swift 4 Syntax (beginner) Cheat Sheet

by [deleted] via cheatography.com/66851/cs/16721/

If / else if / else

```
if var1 == var2 {
    print("Ok")
} else if var1 < var2 {
    print("Not ok")
} else {
    print("Maybe ok")
}
```

Switch

```
switch (var1, var2) {
    case (3, 8):
        print("Sure")
    case (1...15, 20..<25):
        print("Ranges")
    case (let localVar1, let localVar2) where
localVar1 + localVar2 > 30:
        print(localVar1, localVar2)
    default:
        print("Nothing")
}
```

while / repeat while

```
var digit = 5
while digit > 0 {
    digit -= 1
    print(digit)
}
repeat {
    digit -= 1
    print(digit)
} while digit >= 0
```

for in

```
for item in ["item1", "item2"] {
    print(item)
}
```

Guard

```
guard var1 >= 3 else {
   print("Lower than 2")
   continue
}
```

Optionals

```
var optional1: String?
var optional2: Int? = 10
```

Basic class

```
class Adventurer {
    var name: String
    var hp: Int
    let maxHealth: Int = 100

    static var credo = "Defend the helpless"

    var specialMove: String?

    var toFullHp: Int {
        return maxHealth - hp
    }

    var health: Int {
        get {
            return hp
        }
        set {
            hp = newValue
        }
    }

    init(name: String, hp: Int) {
        self.name = name
        self.hp = hp
    }

    convenience init(name: String) {
        self.init(name: name, hp: 100)
    }
}
```

Subclass

```
class Adventurer {
    var name: String
    static var credo = "Defend the helpless"

    init(name: String) {
        self.name = name
    }

    func attack(damage: Int) {
        print("Attacking for \(damage) damage.")
    }

    static func printCredo() {
        print(credo)
    }
}
class Ranger: Adventurer {

    var classAdvantage: String

    init(name: String, advantage: String) {
        self.classAdvantage = advantage
        super.init(name: name)
    }

    override func attack(damage: Int) {
        print("Ranger attack for \(damage)")
    }
}
var adventurer = Adventurer(name: "Harrison")
var ranger = Ranger(name: "Steven", advantage: "-
Stealth")
```

Array

```
var Array: Array<String> = []
var Array2 = Array<String>()
var Array3 = [Double]()
var Array4: [Int] = []
```

Dictionary

```
var dict1: Dictionary<Int, Int> = [:]
var dict2 = Dictionary<Int, String>()
var dict3 = [String : String]()
var dict4: [String : String] = [:]
```

Tuple

```
var tuple = (string: "str", int: 1, isBool: true)
var tuple2: (string: String, int: Int, isBool:
Bool)
tuple2 = ("str", 1, true)
var tuple3: (String, Int, Bool) = ("str", 1, true)
```

Set

```
var set = Set<Int>()
var set2: Set<Int> = []
```

Basic function

```
func myFunction(value: String) -> Bool {
    print("Basic function")
    return true
}
```

Complex function

```
func myFunction(value1: String = "Value1", value2:
Int = 2) -> (return1: String, return2: Int) {
   print("Complex function")
   return ("string", 3)
}
```

Functions as parameter

```
func superFunc(value: Int) -> Int {
    return value * 4
}
func myFunc(value: Int, anotherFunc: (Int) -> Int)
{
    let var1 = anotherFunc(value)
    print(var1)
}
myFunc(2, superFunc)
```

Empty Closure

```
var closureDeclaration: () -> () = {}
```

Basic shorthand

```
var closure = { (parameterString: String) -> Void
in
    print(parameterString)
}
```

Closure shorthand

```
var closure: (String) -> String = { value in
    return "\(value)"
}
```

Closures as function parameters

```
func myFunc(closure: ([String]) -> Void) {
    let array = ["value1", "value2", "value3", "-
value4"]
    closure(array)
}
myFunc { (values) in
    for value in values {
        print("\(value)")
    }
}
```