Swift Cheatsheet

A comprehensive guide to Swift programming language syntax, features, and best practices, designed for quick reference and efficient learning.

Swift Basics

Variables and Constants

`var`	Declares a variable (mutable).
	`var name = "John"`
	`name = "Jane" // Allowed`

`let`	Declares a constant (immutable).
	`let pi = 3.14159`
	`// pi = 3.0 // Error: Cannot assign to value: 'pi' is a 'let' constant`

Type Annotation	Explicitly specify the type of a variable or constant.
	`var age: Int = 30`
	`let message: String = "Hello, Swift!"`

Type Inference	Swift automatically infers the type.
	`var score = 100 // Infers Int`
	`let greeting = "Hi" // Infers String`

Optional Variables	Variables that may or may not have a value. Defined with `?`.
	`var optionalString: String?`
	`optionalString = "Hello"`
	`print(optionalString!) // Forced Unwrapping`

Optional Binding	Safely unwrap optionals using `if let` or `guard let`.
	`if let str = optionalString {`
	`    print(str) // Safe Unwrapping`
	`}`

Data Types

Basic Data Types:

- `Int`: Integer numbers (e.g., `10`, `-5`).
- `Double`: 64-bit floating-point numbers (e.g., `3.14`).
- `Float`: 32-bit floating-point numbers (e.g., `3.14`).
- `Bool`: Boolean values (`true` or `false`).
- `String`: Textual data (e.g., `"Hello"`).
- `Character`: A single character (e.g., `"A"`).

Collection Types:

- `Array`: An ordered collection of values of the same type.
- `Set`: An unordered collection of unique values of the same type.
- `Dictionary`: An unordered collection of key-value pairs.

Tuples:

- Groups multiple values into a single compound value.

```
let point = (x: 10, y: 20)
print(point.x) // 10
```

Operators

Arithmetic Operators	`+`, `-`, `*`, `/`, `%` (remainder)
Comparison Operators	`==`, `!=`, `>`, `<`, `>=`, `<=`
Logical Operators	`&&` (AND), `||` (OR), `!` (NOT)
Assignment Operators	`=`, `+=`, `-=`, `*=`, `/=`, `%=`

Range Operators	`...` (closed range), `..<` (half-open range)
	`for i in 1...5 { print(i) }`
	`// 1 2 3 4 5`
	`for i in 1..<5 { print(i) }`
	`// 1 2 3 4`

Control Flow

Conditional Statements

`if` Statement

```
let temperature = 20
if temperature > 18 {
    print("It's warm.")
} else {
    print("It's cold.")
}
```

`switch` Statement

```
let day = "Monday"
switch day {
case "Monday":
    print("Start of the week")
case "Friday":
    print("End of the week")
default:
    print("Another day")
}
```

Loops

`for-in` Loop	Iterates over a sequence (e.g., array, range).
	`let numbers = [1, 2, 3, 4, 5]`
	`for number in numbers {`
	`    print(number)`
	`}`

`while` Loop	Executes a block of code as long as a condition is true.
	`var count = 0`
	`while count < 5 {`
	`    print(count)`
	`    count += 1`
	`}`

`repeat-while` Loop	Similar to `while`, but executes the block at least once.
	`var i = 0`
	`repeat {`
	`    print(i)`
	`    i += 1`
	`} while i < 5`

Control Transfer Statements

`break`

Terminates the execution of a loop or switch statement.

`continue`

Skips the rest of the current iteration and starts the next iteration.

`fallthrough`

In `switch` statements, it transfers control to the next case (without checking the case condition).

`return`

Exits from a function or method.

Functions and Closures

Functions

Function Definition

```
func greet(name: String) -> String {
    return "Hello, " + name + "!"
}

print(greet(name: "World")) // "Hello, World!"
```

Function Parameters

```
func add(x: Int, y: Int) -> Int {
    return x + y
}

print(add(x: 5, y: 3)) // 8
```

Function with Multiple Return Values (Tuples)

```
func minMax(array: [Int]) -> (min: Int, max: Int)? {
    if array.isEmpty { return nil }
    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)
}

if let result = minMax(array: [8, -6, 2, 109, 3, 71]) {
    print("Min is \(result.min), and max is \(result.max)")
}
```

Closures

Closure Expression Syntax

```
{ (parameters) -> return type in
    statements
}
```

Example: Using a closure to sort an array

```
let numbers = [20, 19, 7, 12]
let sortedNumbers = numbers.sorted { (a: Int, b: Int) -> Bool in
    return a < b
}

print(sortedNumbers) // [7, 12, 19, 20]
```

Trailing Closures

If a closure is the last argument to a function, it can be written after the function call's parentheses.

```
let sortedNumbers = numbers.sorted { a, b in a < b }
```

Shorthand Argument Names

Swift automatically provides shorthand argument names `$0`, `$1`, etc.

```
let sortedNumbers = numbers.sorted { $0 < $1 }
```

Structures and Classes

Structures

Structure Definition

```
struct Point {
    var x: Int
    var y: Int
}

let myPoint = Point(x: 10, y: 20)
print(myPoint.x) // 10
```

Structures are Value Types

When a structure is assigned to a new variable, a copy of the structure is created.

```
var point1 = Point(x: 1, y: 1)
var point2 = point1
point2.x = 5
print(point1.x) // 1
print(point2.x) // 5
```

Classes

Class Definition

```
class Dog {
    var name: String
    init(name: String) {
        self.name = name
    }
    func bark() {
        print("Woof!")
    }
}

let myDog = Dog(name: "Buddy")
print(myDog.name) // "Buddy"
myDog.bark()       // "Woof!"
```

Classes are Reference Types

When a class instance is assigned to a new variable, a reference to the original instance is created.

```
let dog1 = Dog(name: "Buddy")
let dog2 = dog1
dog2.name = "Max"
print(dog1.name) // "Max"
print(dog2.name) // "Max"
```

Inheritance

Classes can inherit properties and methods from other classes.

```
class Poodle: Dog {
    override func bark() {
        print("Poodle Woof!")
    }
}

let myPoodle = Poodle(name: "Lucy")
myPoodle.bark() // "Poodle Woof!"
```

Protocols

Protocol Definition

```
protocol Animal {
    var name: String { get }
    func makeSound()
}

struct Cat: Animal {
    let name: String
    func makeSound() {
        print("Meow!")
    }
}

let myCat = Cat(name: "Whiskers")
myCat.makeSound() // "Meow!"
```