Data Types and Variables

```
// Variables, can be changed. Type inference
var name = "John"
var age = 20
var weight = 200.5
var isOrganDonor = true
// data types can be called explicitly
var name: String = "John"
var age: Int = 20
var weight: Double = 200.5
var isOrganDonor: Bool = true
// constants, cannot be changed
let name = "John"
```

Functions

```
// function with no parameters
func sayHello() {
    print("Hello")
}
// function with parameters
func sayHello(name: String) {
    print("Hello \(name)")
}
// function with parameters and return type
func sayHello(name: String) -> String {
    return "Hello \(name)"
}
// omit parameter label
func sayHello(_ name: String) -> String {
    return "Hello \(name)"
}
```

Loops

```
// for loop
for i in 1...10 {
    print(i)
}
// while loop
var i = 1
while i <= 10 {
    print(i)
    i += 1
}
// repeat while loop
var i = 1
repeat {
    print(i)
    i += 1
} while i <= 10
```

ARRAYS

```
// arrays: used to store multiple values of the same type
var numbers = [1, 2, 3, 4]
// access by index
print(numbers[0])
// add to array
numbers.append(5)
// remove from array
numbers.remove(at: 0)
// iterate over array
for number in numbers {
    print(number)
}
```

Conditionals

```
// if statement
if age >= 21 {
    print("You can drink")
} else if age >= 18 {
    print("You can vote")
} else {
    print("You can't drink or vote")
}
// switch statement
switch age {
case 0...2:
    print("Infant")
case 3...12:
    print("Child")
case 13...19:
    print("Teenager")
default:
    print("Adult")
}
```

Tuples

```
// tuples: used to group multiple values in a single value
let credentials = (name: "John", password: "password")
```

Gaurd

```
// guard statement: used to exit early if a condition isn't met
func ageRequirement(age: Int?) {
    guard let age = age, age >= 21 else {
        print("You can't drink")
        return
    }
    print("You can drink")
}
```

Optionals

```
// optionals: used to indicate that a value may not exist
var age: Int? = nil
// force unwrap: can cause a crash if nil
print(age!)
// optional binding: if let
if let age = age {
    print(age)
} else {
    print("No age")
}
```

Enumerations

```
// enumerations: used to group related values
enum Weekend {
    case saturday
    case sunday
}
// create instance of enum
let today = Weekend.saturday
// switch statement
switch today {
case .saturday:
    print("Today is Saturday")
case .sunday:
    print("Today is Sunday")
}
```

Classes

```
// classes: used to create custom data types
// pass by reference,
class Person {
    var name: String
    var age: Int
    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }
    func sayHello() {
        print("Hello my name is \(name)")
    }
}
// create instance of class
let john = Person(name: "John", age: 20)
// access properties
print(john.name)
// call method
john.sayHello()
```

Structs

```
// structs: used to create custom data types, when you don't need inheritance
// pass by value ,
struct Person {
    var name: String
    var age: Int
    func sayHello() {
        print("Hello my name is \(name)")
    }
}
// create instance of struct
let john = Person(name: "John", age: 20)
// access properties
print(john.name)
// call method
john.sayHello()
```

Dictionaries

```
// dictionaries: used to store key value pairs
var person = ["name": "John", "age": 20]
// access by key
print(person["name"]!)
// add to dictionary
person["weight"] = 200.5
// remove from dictionary
person.removeValue(forKey: "age")
// iterate over dictionary
for (key, value) in person {
    print("\(key): \(value)")
}
```

Closures

```
// closures: used to group functionality together, like binding a function to a variable
let sayHello = {
    print("Hello")
}
// call closure
sayHello()
// closure with parameters
let sayHello = { (name: String) in
    print("Hello \(name)")
}
// call closure
sayHello("John")
```