Hello World

```
print(“Hello, World!”)
print(123456)
print(3.1415)
print([0,1,2])
print(true)
```

```
Hello, World!
123456
3.1415
[0,1,2]
true
```

Basic Types

“Hello, World!”	String
123456	Int
3.1415	Double
[0,1,2]	Array<Int> / [Int]
true	Bool

Declare Constant

```
let PI: Double = 3.1415

let greeting = “hello”

let unknown: Int
```

```
let name: type = value

let name = value

let name: type
```

Use Constant to make your code more readable

```
print("Hello, World!", 123456, 3.1415)

let integer = 123456, PI = 3.1415

print("Hello, World!", integer, PI)

let myStudentID = 1234567

print("My student ID number is ", myStudentID)
```

Declare Variable

```
var PI: Double = 3.1415

var greeting = “hello”

var unknown: Int
```

```
var name: type = value

var name = value

var name: type
```

Type Casting

```swift
var PI: Double = 3.1415
```

```swift
var integer: Int = PI
```

```swift
var integer: Int = Int(PI)
```

Basic Operations

- **Addition (+)**
- **Subtraction (-)**
- **Multiplication (*)**
- **Division (/)**
- **Complementation：求余数 (%)**

The basic Operations are similar to most programming language

Boolean & Logical Operation

- Two values: true, false
- Boolean Operation: `!(not), &&(and), ||(or)`
- Others: `==, !=(not equal), >, >=, <, <=`
- Further, like mathematics using `()` to make expressions longer and readable

Conditional Statements

- ★ If statement
- ■ Switch statement
- ■ Guard statement

Similar to if-Statement, but make your code more readable

Array

Array's type: `[Type]` or `Array<Type>`

The `Type` before can be any type, Because of generic

1. Declare an array: `var array: [SomeType] = [SomeValue]`
2. Read data: `array[index]` (the type of data is someType)
3. Write data: `array[index] = yourValue`
4. Others: `append`, `insert`, `remove`…

You can even add your personal array method

Loop Statements

- For statement
- While statement
- Repeat-While statement

Repeat-While Statement will run at least once

★ Function Statement

```
func name(_ first: Type, outName second: Type) -> Type {
    Statements
    return someValue
}
```

```
assignedValue = name(value_1, outName: value)
```

Then assignedValue will equal to someValue

If the function do not have return value:

```
func name(_ first: Type, outName second: Type) -> Type {}
```