variable names should be one word, starting with a letter

```
var
```

adding strings together

```
“string” + var + “string”
```

booleans

```
true
false
```

```
if
else if
else
```

```
for number in 1...3{
}
```

string interpolation

```
print(“\(var) + \(var)”)
```

checking if it’s empty

```
var.isEmpty
```

length of a string

```
var.count
```

converting int to string

```
String(int)
```

arrays can only hold one value, you can set the value to a type

```
var array = [String]()
```

    the () initializes the array

adding elements to an array

```
array.append(element)
```

indexing

```
[0]
```

```
func greeting (variable_name: type, variable_name: type) -> return-type
{
}
```

adding ‘_’ before variable name so you don’t have to cqll it every time you call a function

```
_variable-name
```

initialize a variable without a value
```
var variable_name = variable_type
```

constant variables, you can’t re-assign them
```
let
```

or: `||`
and: `&&`

dictionaries: unordered lists of one type
```
var dictionary_name = [String: String]
```

    initialize it
```
    [String: String]()
```

```
var myDict = [Int: String]
myDict = [1: “a”, 2: “b”]
myDict[3] = “c”
myDict.removeValue(forKey: 1)
myDict.updateValue(“d”, forKey: 2)
```

classes:
```
class Human{
    var name = String
    var age = Int
    init(name: String, age: Int){
        self.name = name
        self.age = age
    }
    func say_hi(){
        print(“hello”)
    }
}

var me = Human()
me.name = “Bella”

var a = Human(name: “Joe”, age: 26)
var b = a.say_hi()
```

optional: it can have a value of its type or be nil
```
var name: String?
```

implicitly unwrapped optional: demands it has the correct value or it brakes
```
var name = String!
```

inheriting a class or being a subclass:
```
class A {
}
class B: A{
}
```

a subclass can override a parent function

```
override func name(){
}
```

    you can still access it after overriding it with

```
    super.name()
```

```
class Animal{
    var airborne = false
}

class Bird: Animal{
    override init(){
        self.init()
        self.airborne = true
    }
}
```

structures are similar to classes but they are meant for simpler data and they have a default initializer. Hence, initializers are optional.

converting to int, become optional int (Int?)

```
Int(variable)
```

```
var dictionary = [1: “a”, 2: “b”]
let value = dictionary[3] ?? “z”
```

value will have the value of dictionary[3] if that has no value then it will be assigned the value of “z”

checking an optional, optional binding

```
class Human {
    var name: String?
}
var him = Human()
him.name = “Jon”
if let name = him.name {
    print(name)
}
```

enumerated

```
enum Level{
    case beginner
    case intermediate
    case advanced
}
let myLevel: Level
myLevel = Level.intermediate
```

```
enum Season{
    case spring, summer, fall, winter
    func description() -> String{
        return “a time of the year”
    }
}
var season = Season.fall
var season2 = Season.winter
print(season.description())
print(season2.description())
a time of the year
a time of the year
```

switch statements

```
var x = 2
switch x {
    case 0:
        print(“x = 0”)
    case 1:
        print (“x = 1”)
    default:
        print(“x is neither 0 nor 1”)
}
```

every type has a rawValue

```
let a: (Int) -> Int
a = { $0 * $0}
print(a(9))
```

create functions with closures

```
b = {(n: Int) -> Int in return n * n}
```

iterate an entire array

```
map(array)

let myArray = [1,2,3]
print(myArray.map({$0 * 2})
[2,4,6]
```

    first index: $0
    second index: $1
    ...