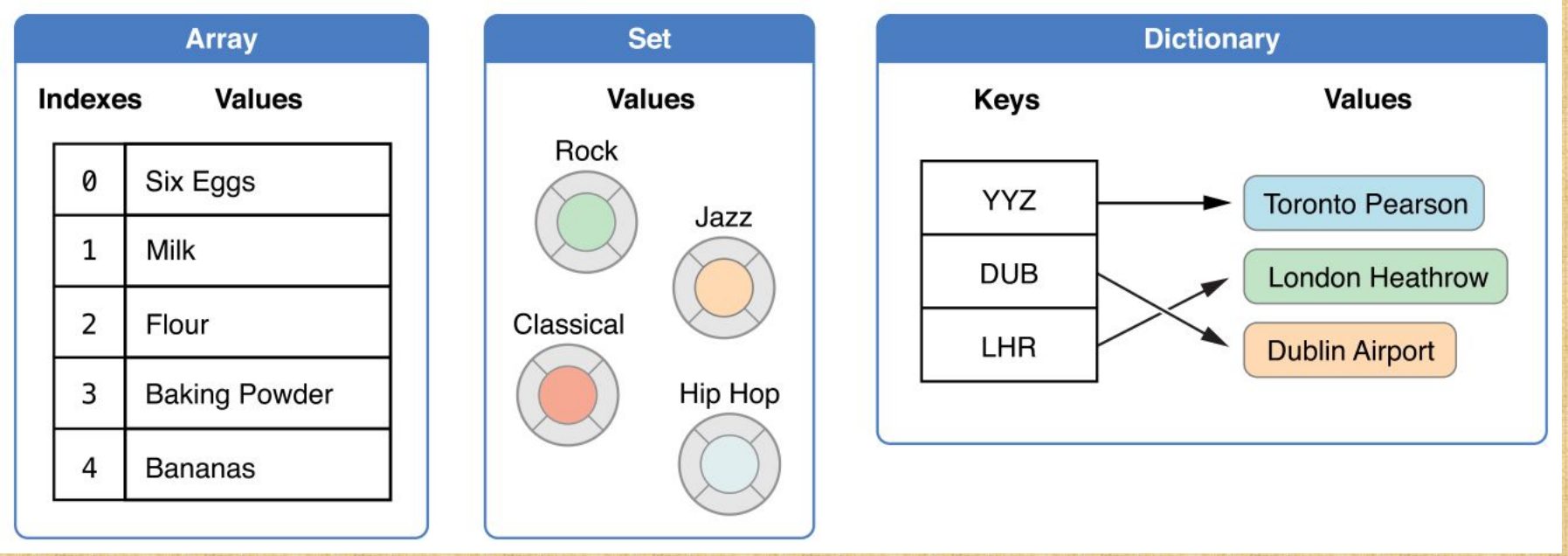
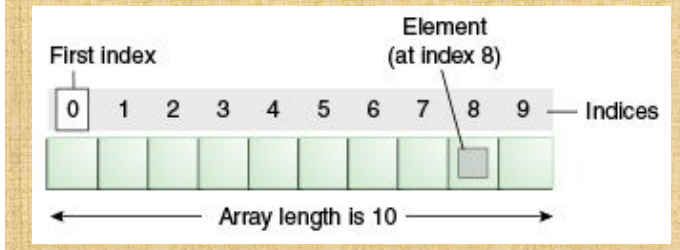
# iOS Programming

Lecture 4

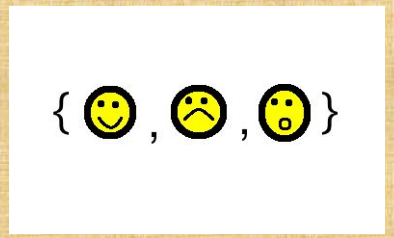
# Recap - Collections



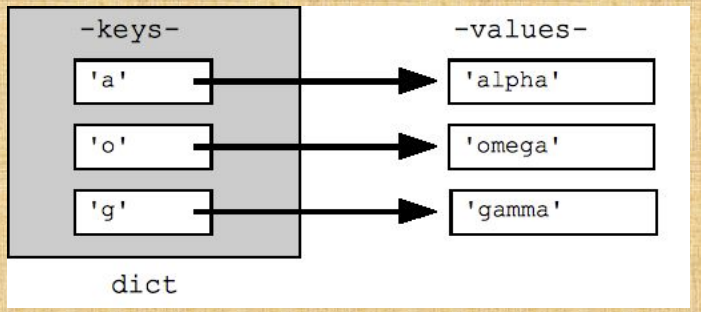

## Array




## Set



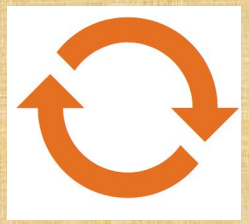
## Dictionary



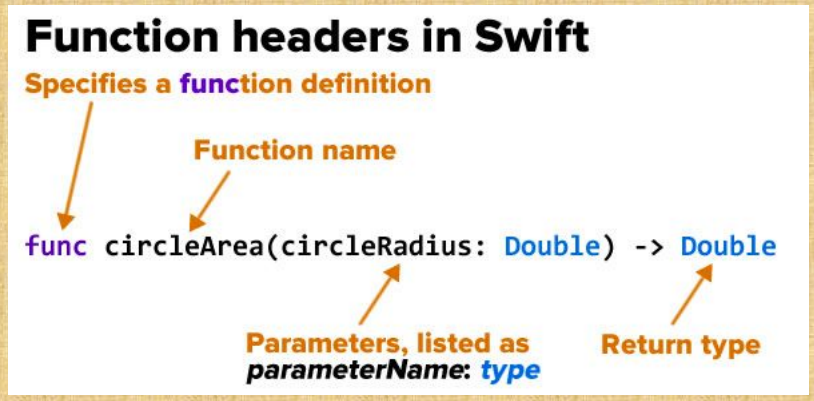
# Today

Conditional Statements

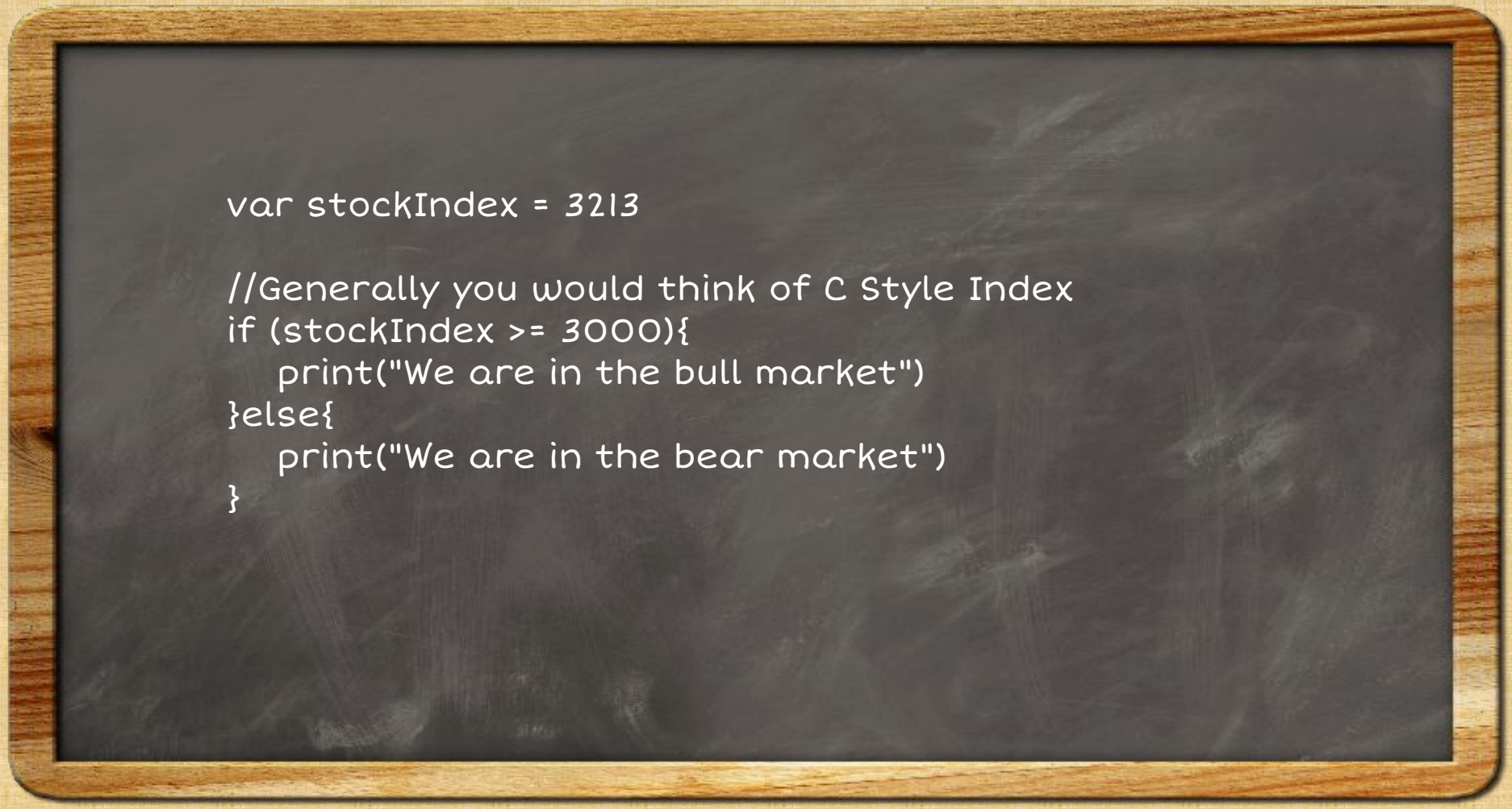

Loops



Functions



# Conditional Statements



```
var stockIndex = 3213

//Generally you would think of C Style Index
if (stockIndex >= 3000){
    print("We are in the bull market")
}else{
    print("We are in the bear market")
}
```

# If/else Swifty style

Drop the brackets

```
var stockIndex = 3213

//You drop the brackets
if stockIndex >= 3000{
//The { are needed even when there is only one line
   print("We are in the bull market")
}else{
   print("We are in the bear market")
}
```

Note: The condition must evaluate to be a Boolean. Unlike C Style languages where 0 can be considered false and any other number is considered true.

# If/else – for unwrapping

```
var stockIndex: Int?
stockIndex = 3213

if let currentIndex = stockIndex {
    if currentIndex >= 3000 {
        print("We are in the bull market.")
    }else{
        print("We are in the bear market.")
    }
}else{
    print("Need the current stock index to share
market type.")
}
```

# If/else – Logical Operators

```
var stockIndex = 3213
var isFutureBright = false

//Logical and, Logical || options are available
if stockIndex >= 3000 && isFutureBright {
    print("We are in the bull market.")
}else{
    print("We are in the bear market.")
}
```

# If/else if/else

```
var language = "Swift"

if language == "C++" {
    print("You are learning C++")
} else if language == "Java" {
    print("You are learning Java")
} else if language == "Python" {
    print("You are learning Python")
} else if language == "Kotlin" {
    print("You are learning Kotlin")
} else if language == "Swift" {
    print("You are learning Swift")
} else {
    print("You are learning Unknown language")
}
```

# Switch – Better way

```
var language = "Swift"

switch language {
   case "C++" :
   print("You are learning C++")
   case "Java" :
   print("You are learning Java")
   case "Python" :
   print("You are learning Python")
   case "Kotlin" :
   print("You are learning Kotlin")
   case "Swift" :
   print("You are learning Swift")
   default :
   print("You are learning Unknown language")
}
```

# Switch – Must be exhaustive

```
var language = "Swift"

switch language {
   case "C++" :
   print("You are learning C++")
   case "Java" :
   print("You are learning Java")
   case "Python" :
   print("You are learning Python")
   case "Kotlin" :
   print("You are learning Kotlin")
   case "Swift" :
   print("You are learning Swift")
//   default :
//   print("You are learning Unknown language")
}
```

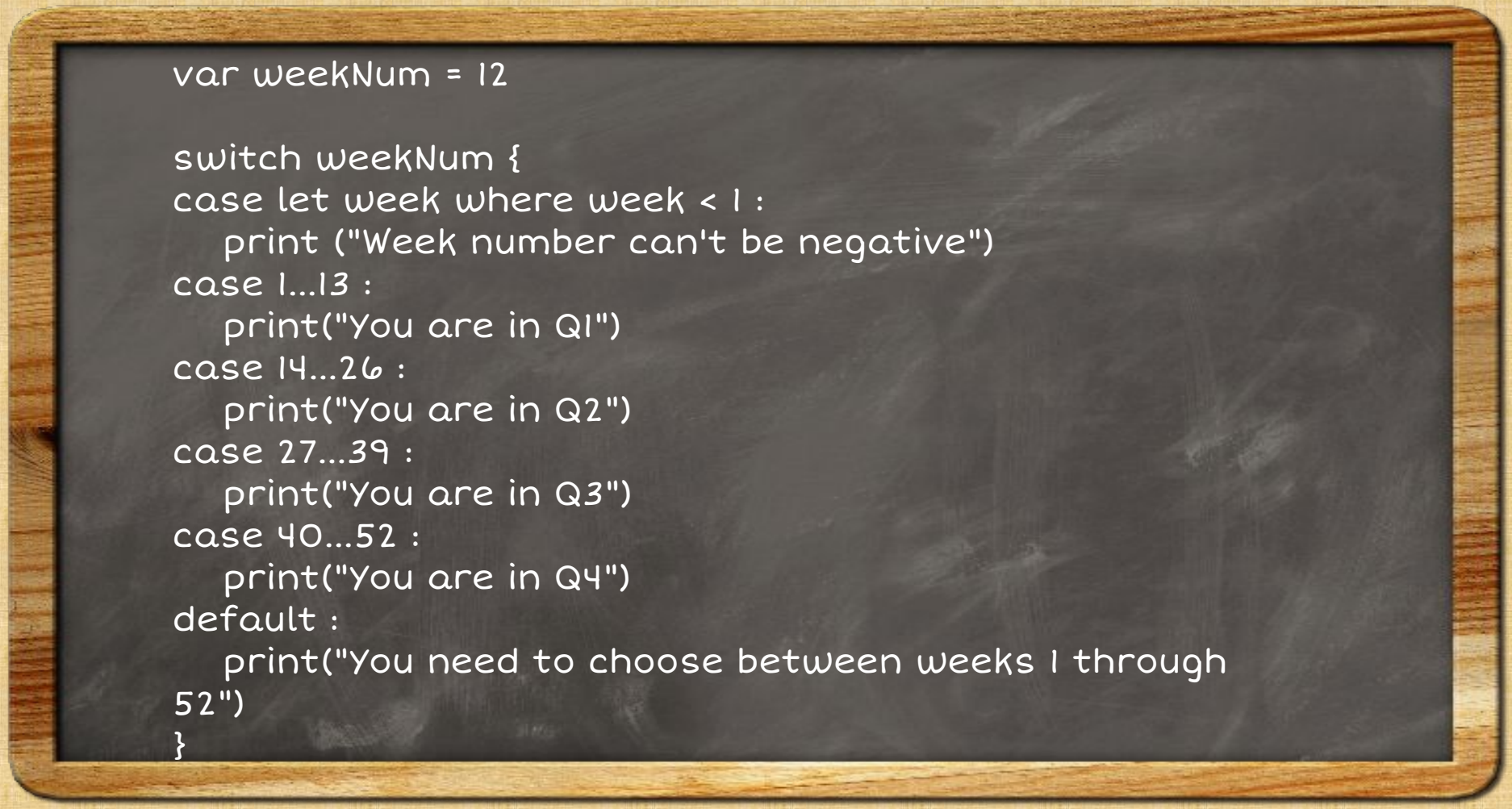
Switch must be exhaustive

Must be exhaustive

# Switch – Range Operator

```
var weekNum = 12

switch weekNum {
case 1...13 :
   print("You are in Q1")
case 14...26 :
   print("You are in Q2")
case 27...39 :
   print("You are in Q3")
case 40...52 :
   print("You are in Q4")
default :
   print("You need to choose between weeks 1 through
52")
}
```

# Switch – Less than check

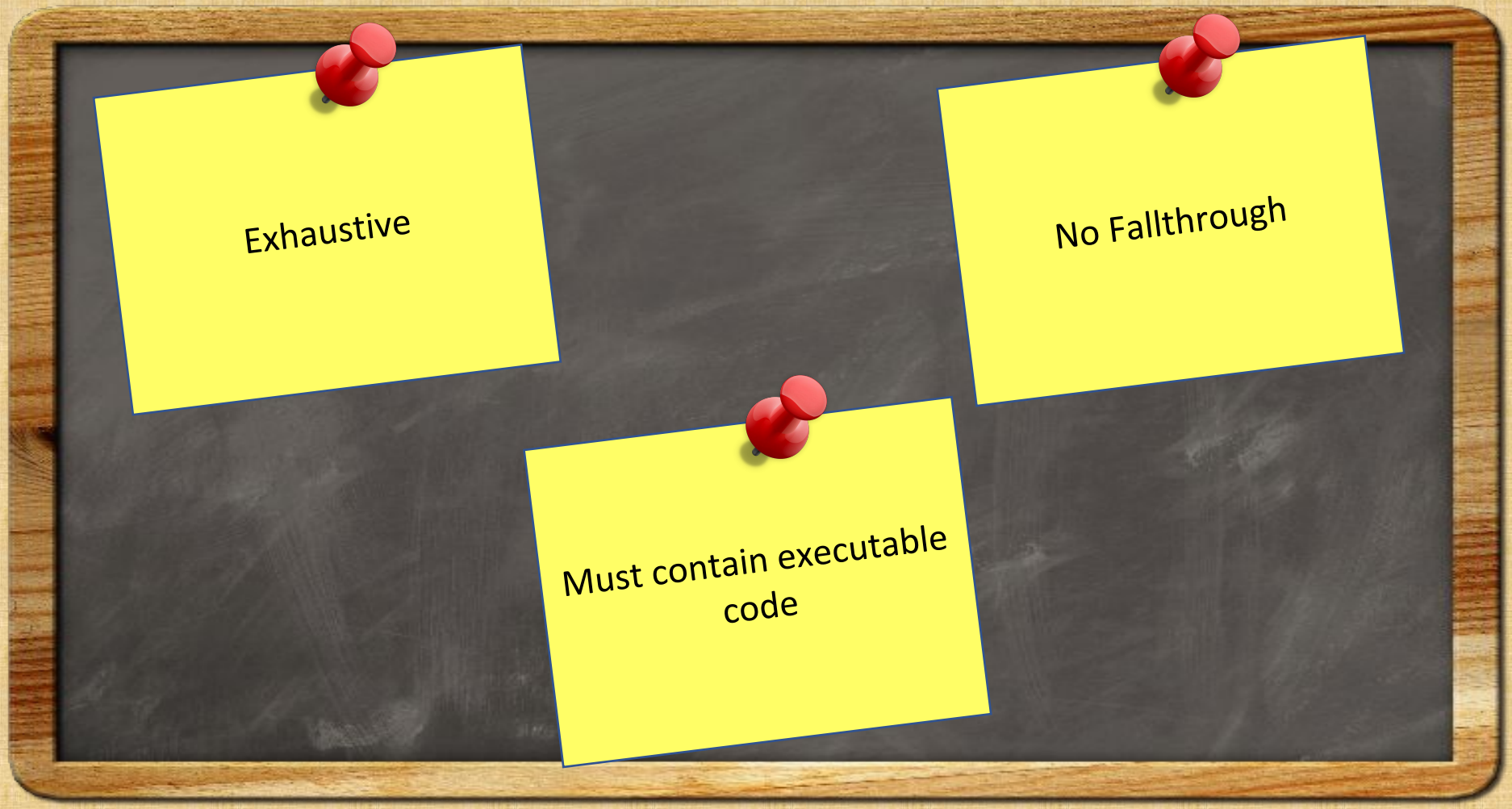


```
var weekNum = 12

switch weekNum {
case let week where week < 1 :
   print ("Week number can't be negative")
case 1...13 :
   print("You are in Q1")
case 14...26 :
   print("You are in Q2")
case 27...39 :
   print("You are in Q3")
case 40...52 :
   print("You are in Q4")
default :
   print("You need to choose between weeks 1 through
52")
}
```

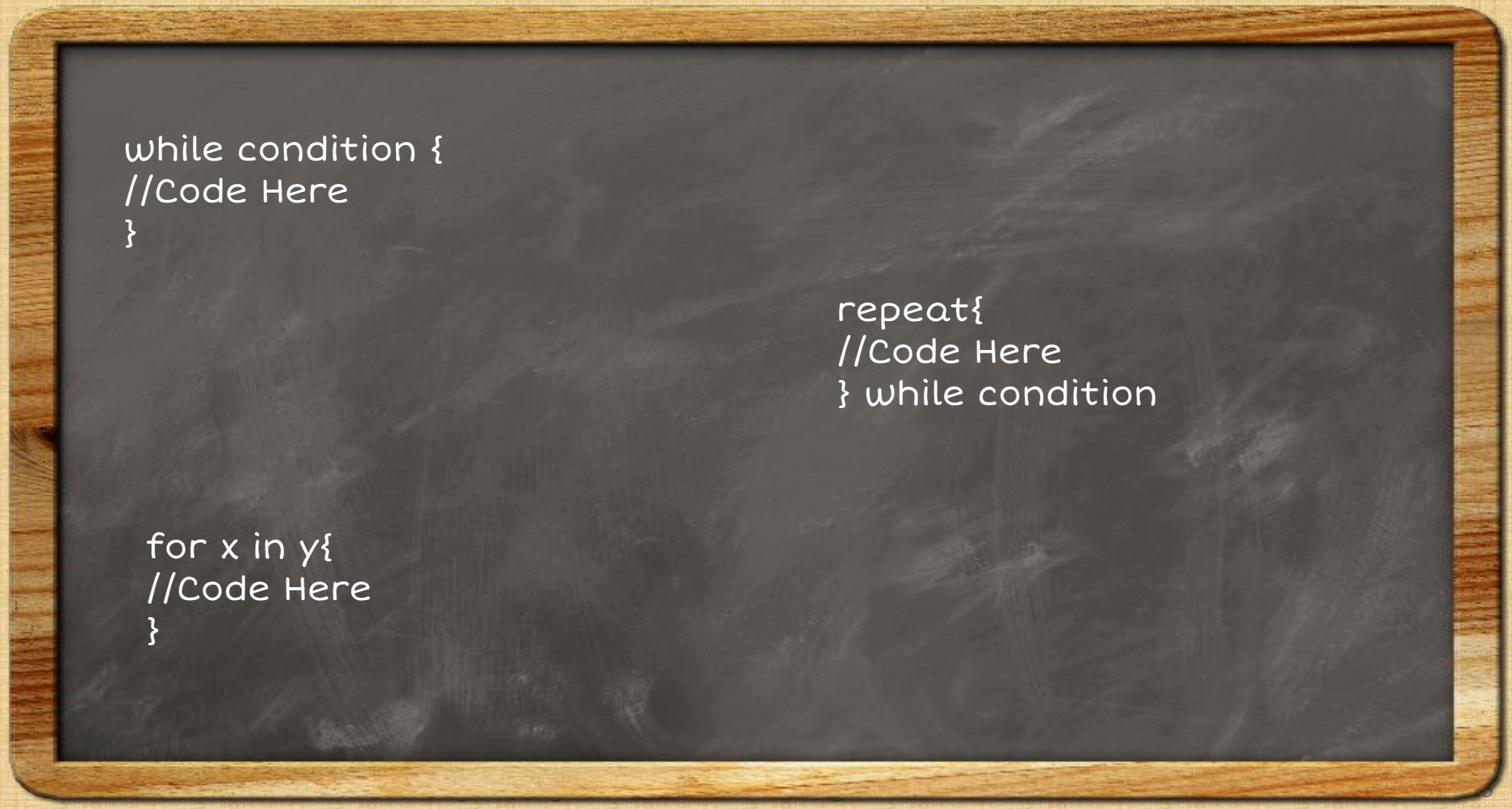
# Switch – Comma choices

```
var char: Character = "a"

switch char {
case "a", "A":
   print ("You are starting character")
case "b"..."z":
   print ("I am small case letter")
case "B"..."Z":
   print ("I am capital case letter")
case "0"..."9":
   print ("I am numeric character")
default:
   print ("I am not a alpha numeric character")
}
```

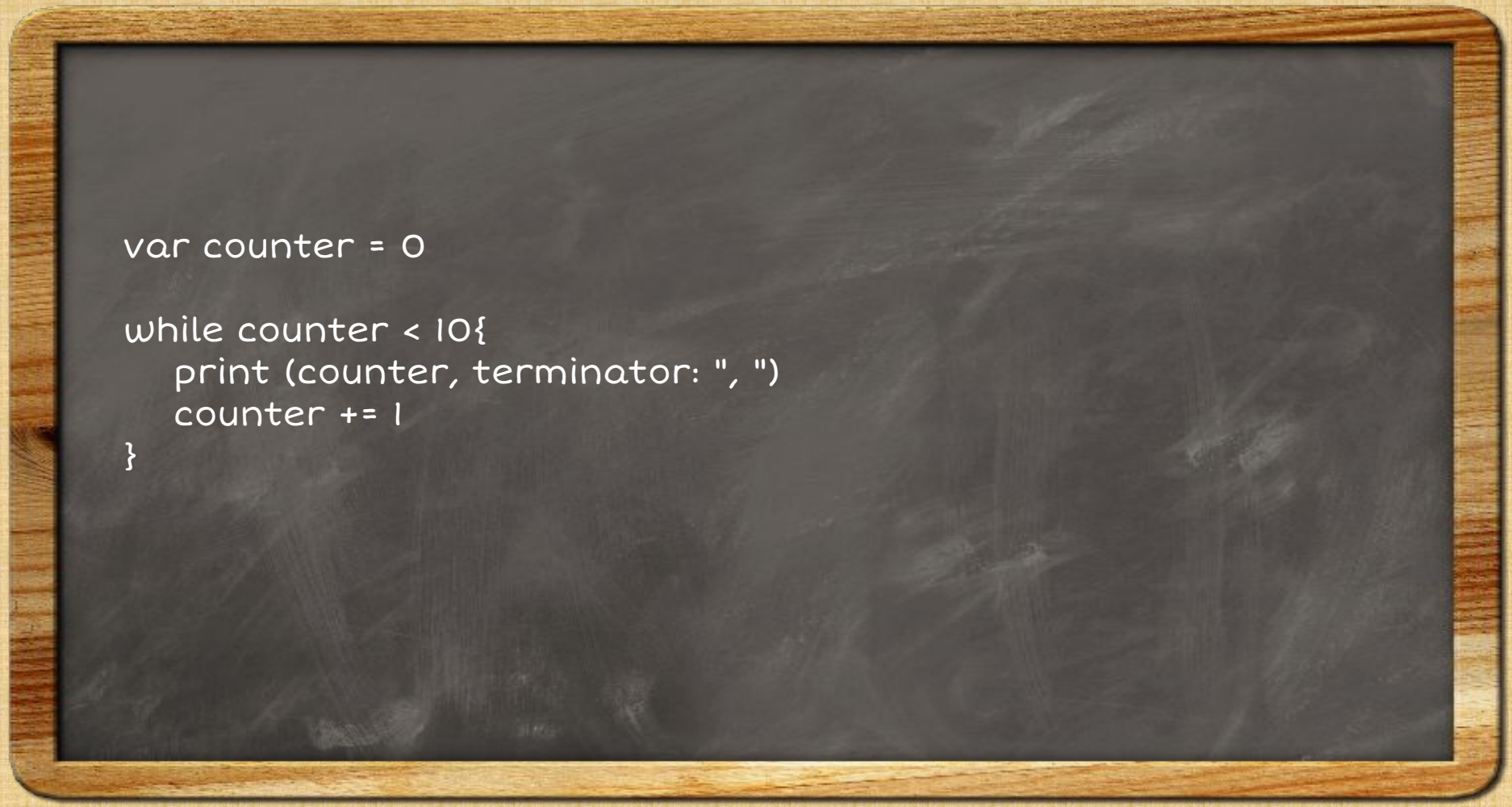
# Switch – Summary



# Loops

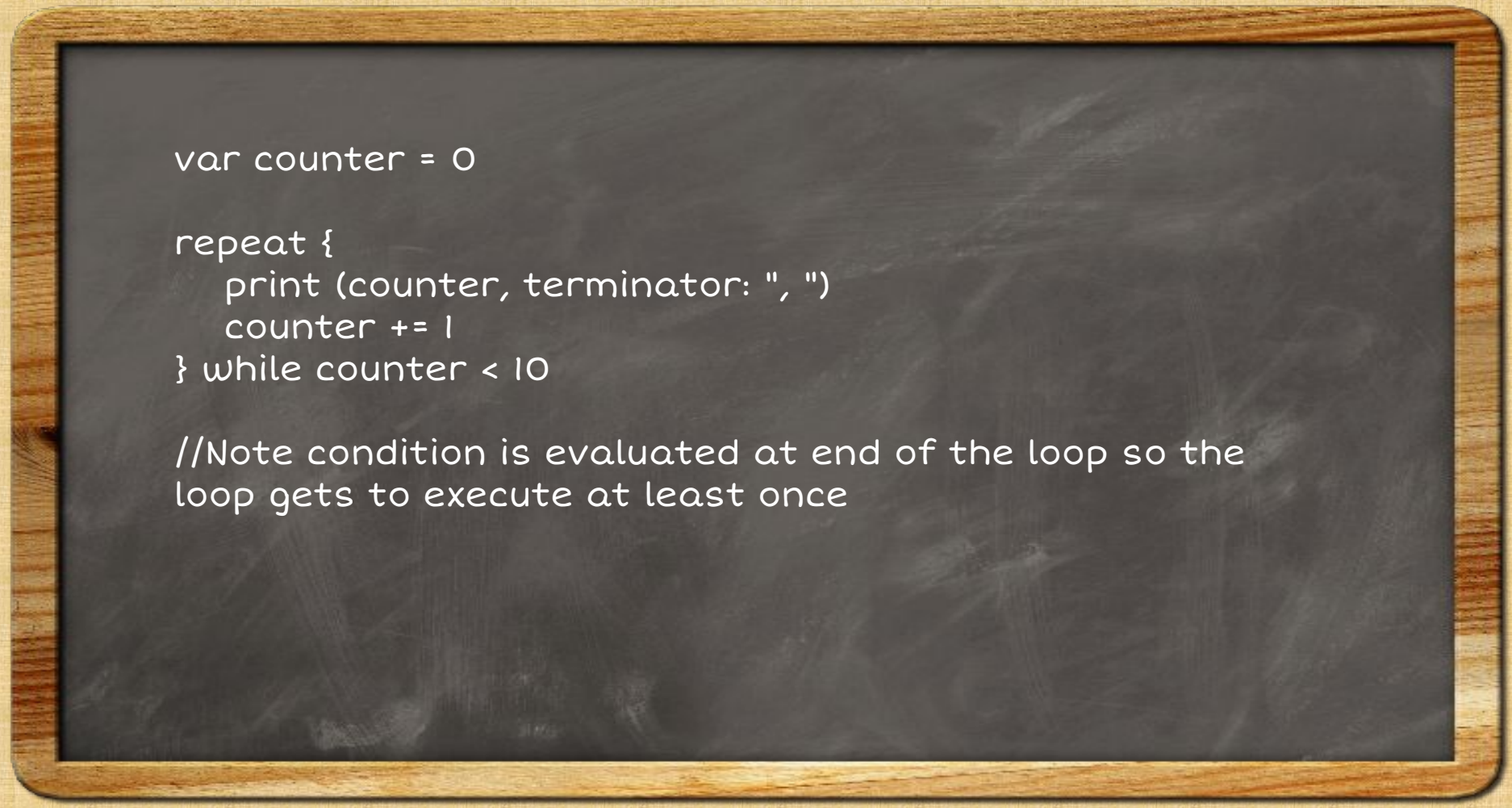

while condition {
//Code Here
}
repeat{
//Code Here
} while condition
for x in y{
//Code Here
}

# Loops – while loop



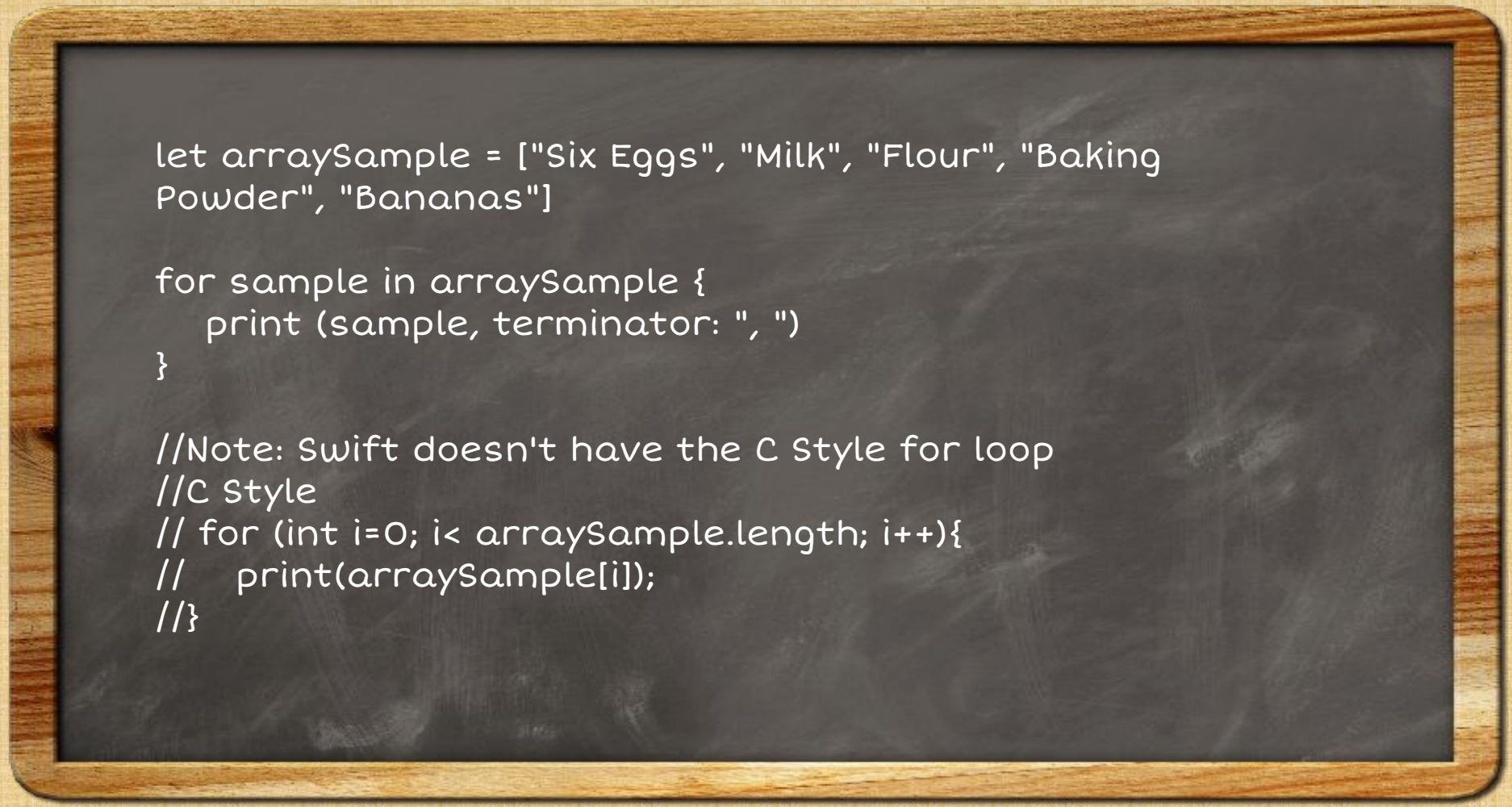
```
var counter = 0

while counter < 10{
   print (counter, terminator: ", ")
   counter += 1
}
```

# Loops – repeat while loop



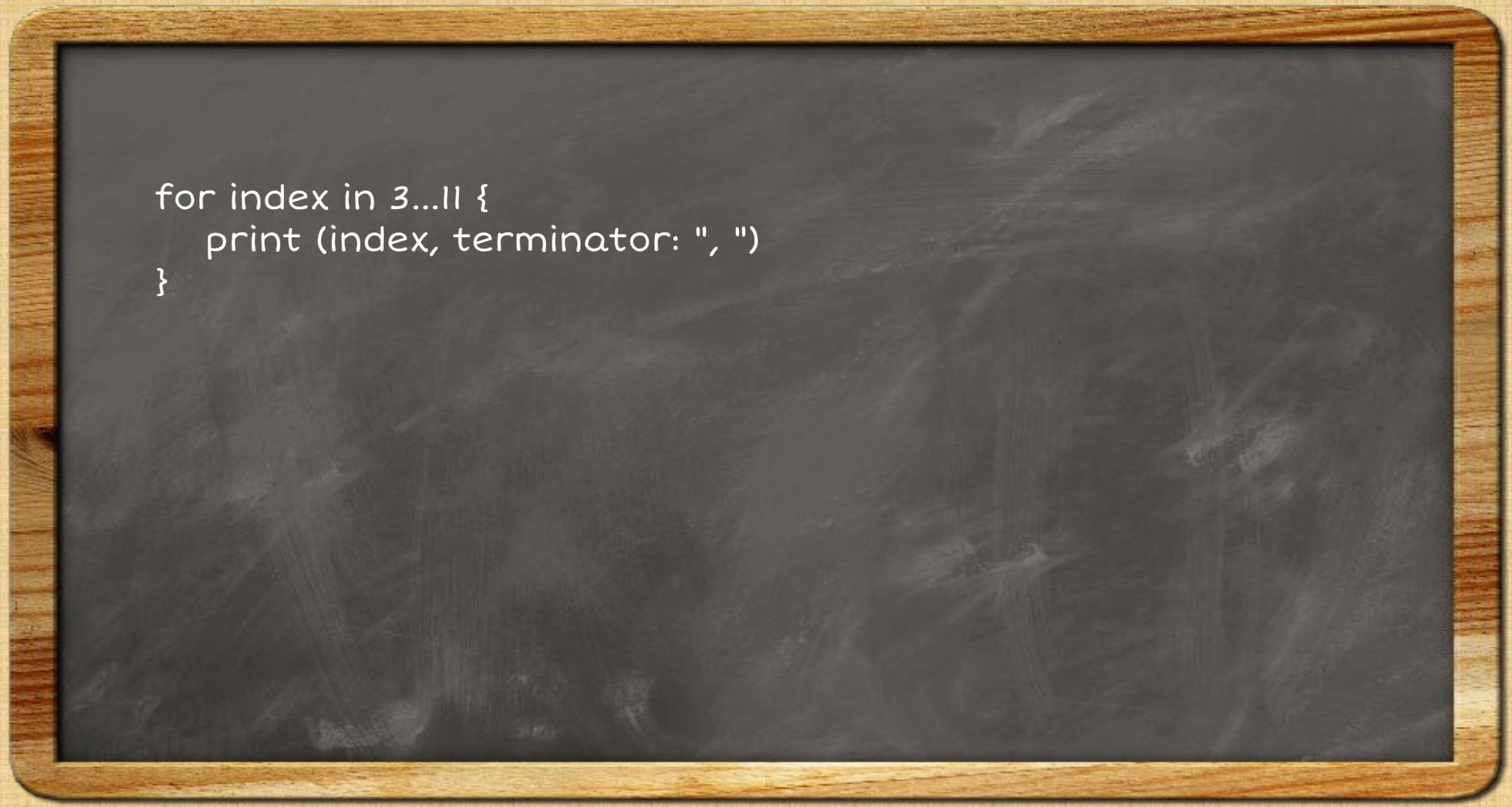
```
var counter = 0

repeat {
    print (counter, terminator: ", ")
    counter += 1
} while counter < 10

//Note condition is evaluated at end of the loop so the
loop gets to execute at least once
```

# Loops – for loop



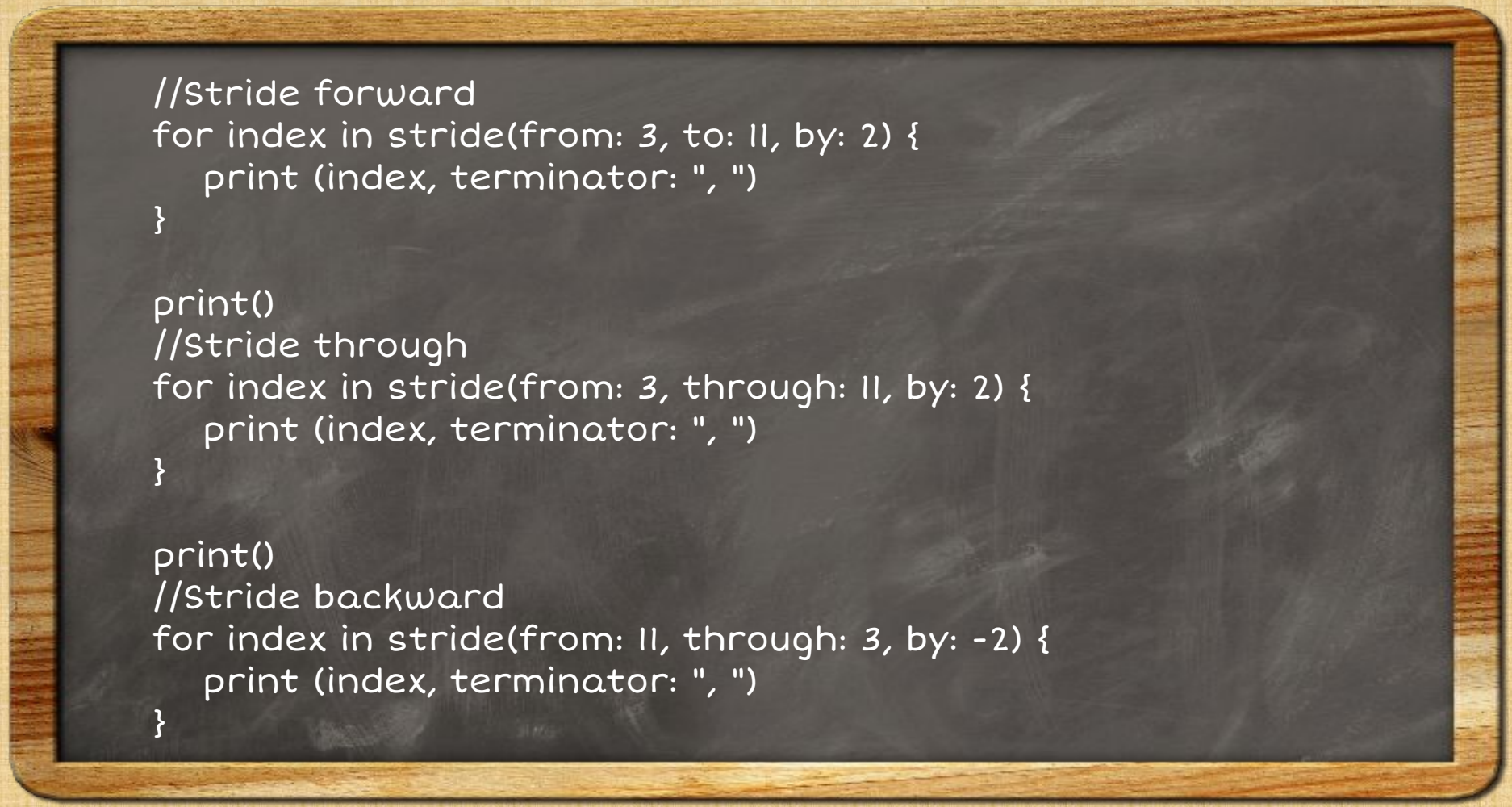
```
let arraySample = ["Six Eggs", "Milk", "Flour", "Baking
Powder", "Bananas"]

for sample in arraySample {
    print (sample, terminator: ", ")
}

//Note: Swift doesn't have the C Style for loop
//C Style
// for (int i=0; i< arraySample.length; i++){
//    print(arraySample[i]);
//}
```

# Loops – Range Operator



```
for index in 3...11 {
    print (index, terminator: ", ")
}
```

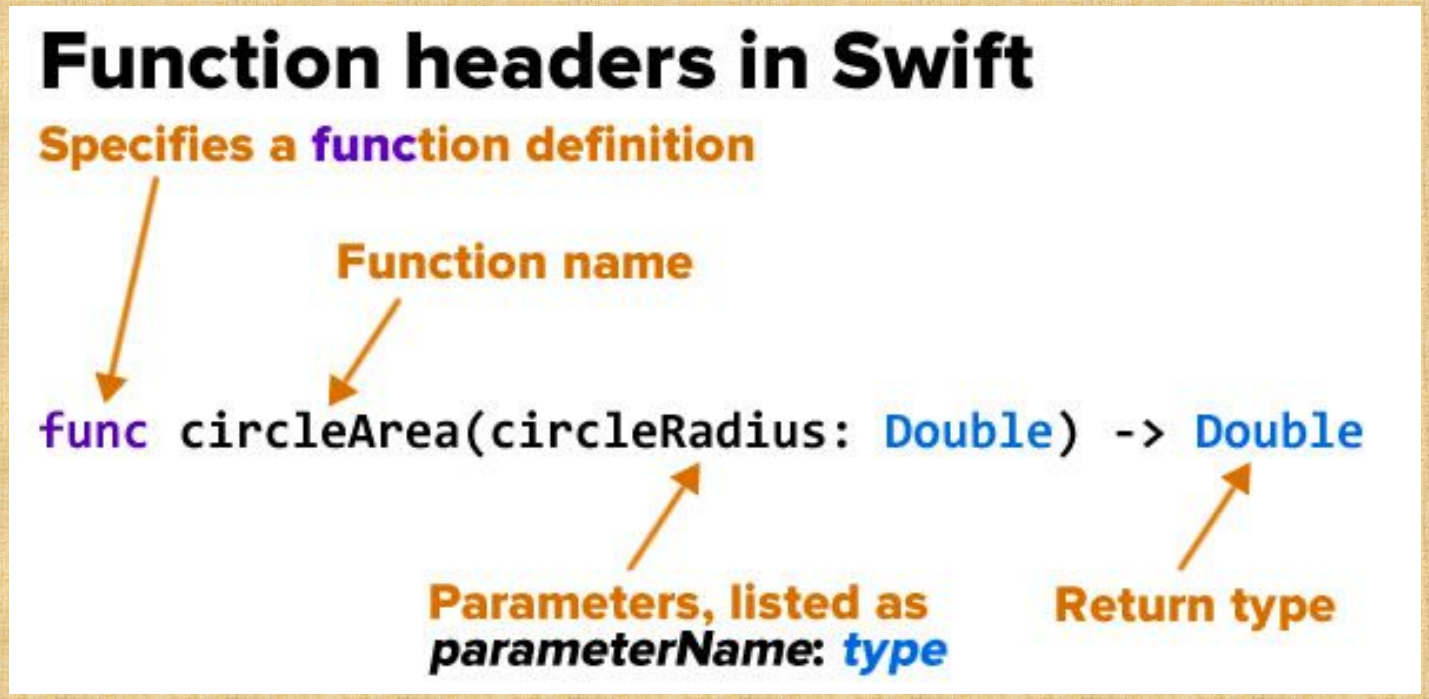
# Loops – for stride



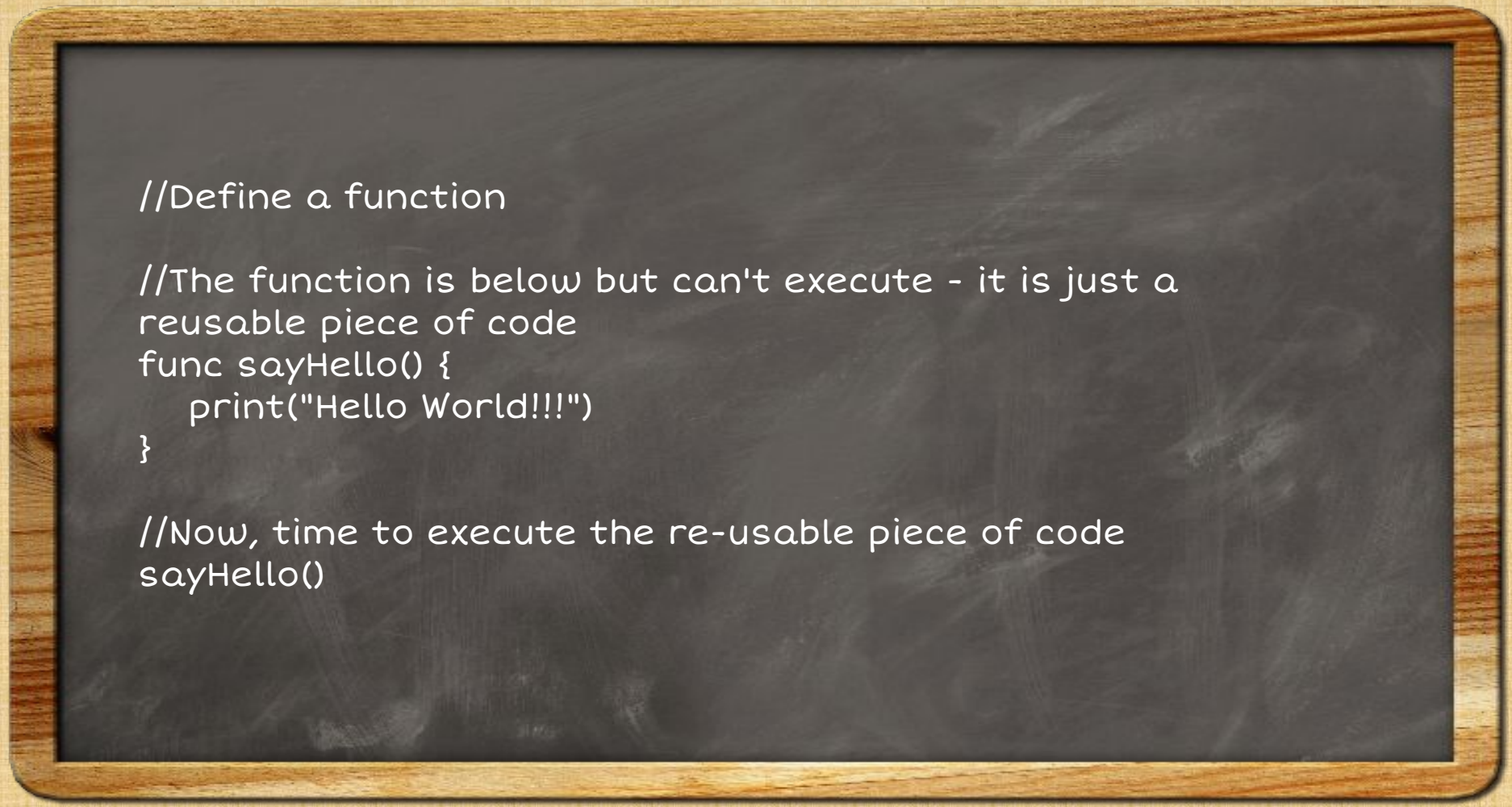
```
//Stride forward
for index in stride(from: 3, to: 11, by: 2) {
    print (index, terminator: ", ")
}

print()
//Stride through
for index in stride(from: 3, through: 11, by: 2) {
    print (index, terminator: ", ")
}

print()
//Stride backward
for index in stride(from: 11, through: 3, by: -2) {
    print (index, terminator: ", ")
}
```

# Functions

Functions provide re-usable pieces of code

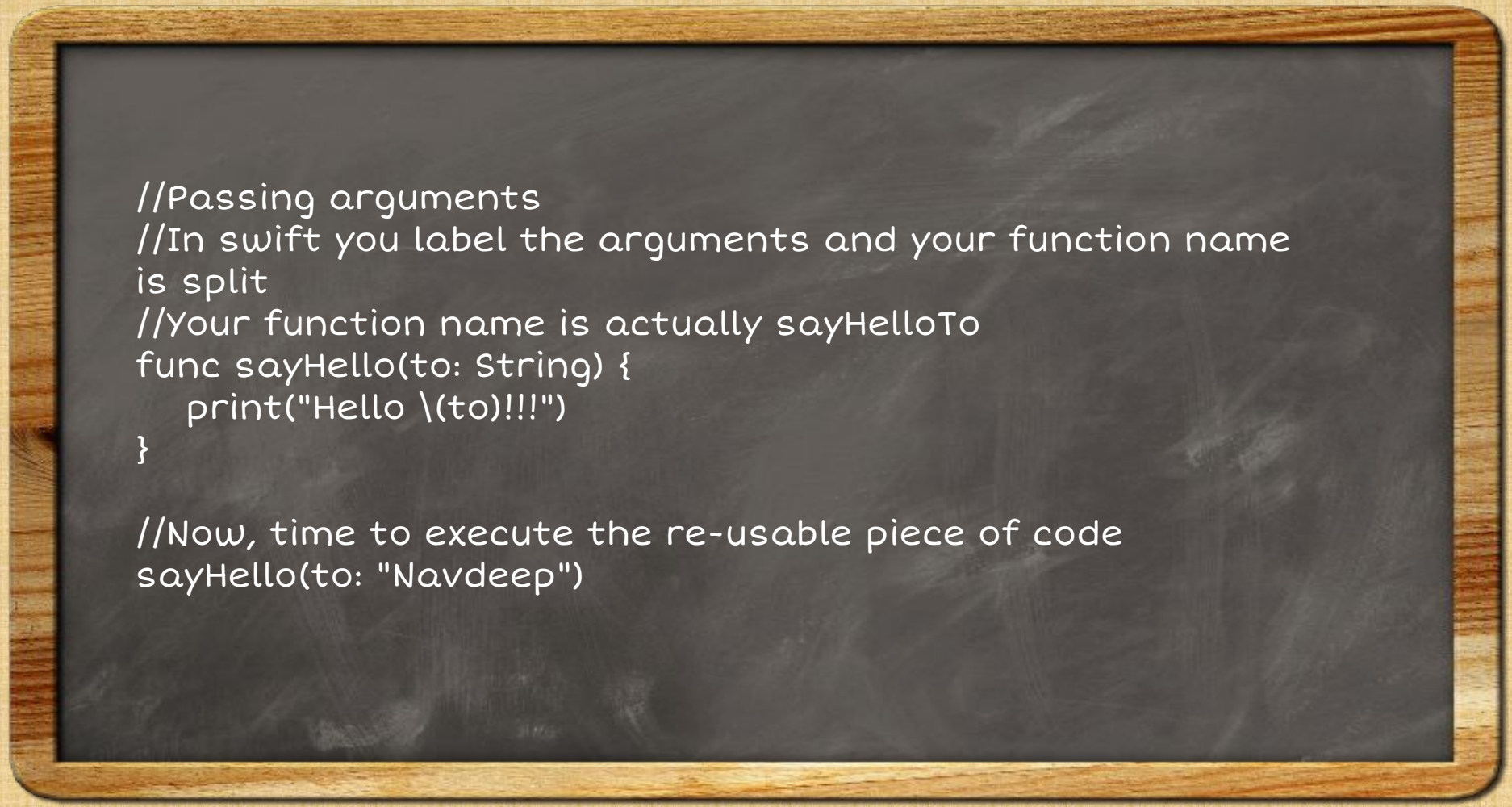


# Functions



```
//Define a function

//The function is below but can't execute - it is just a
reusable piece of code
func sayHello() {
    print("Hello World!!!")
}

//Now, time to execute the re-usable piece of code
sayHello()
```

# Functions – Passing Arguments

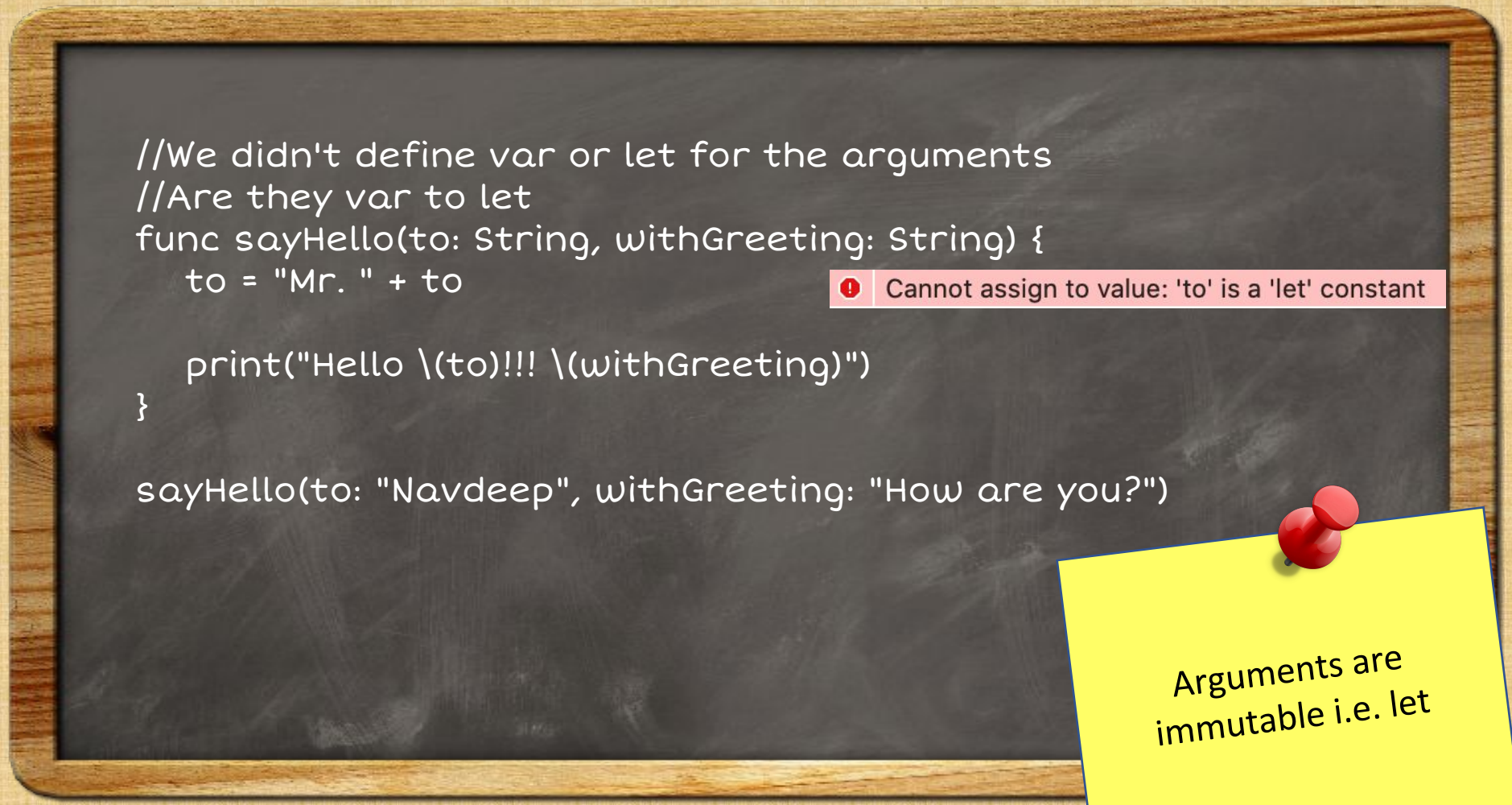


```
//Passing arguments
//In swift you label the arguments and your function name
is split
//Your function name is actually sayHelloTo
func sayHello(to: String) {
    print("Hello \(to)!!!")
}

//Now, time to execute the re-usable piece of code
sayHello(to: "Navdeep")
```

# Functions – Passing Multiple Arguments

```
//Passing Multiple arguments
//In swift you label the arguments and your function name is split
//Your function name is actually sayHelloToWithGreeting
func sayHello(to: String, withGreeting: String) {
    print("Hello \(to)!!! \(withGreeting)")
}

//Now, time to execute the re-usable piece of code
sayHello(to: "Navdeep", withGreeting: "How are you?")
```

# Functions – Immutable Arguments



# Functions – Default Argument Values

```
//Sometimes it makes sense to have default behavior, no
need to
//have two methods with one passing a constant to
second
func sayHello(to: String, withGreeting: String = "How is your
day so far?") {
    print("Hello \(to)!!! \(withGreeting)")
}

sayHello(to: "Navdeep", withGreeting: "How are you?")
sayHello(to: "Navdeep")
```

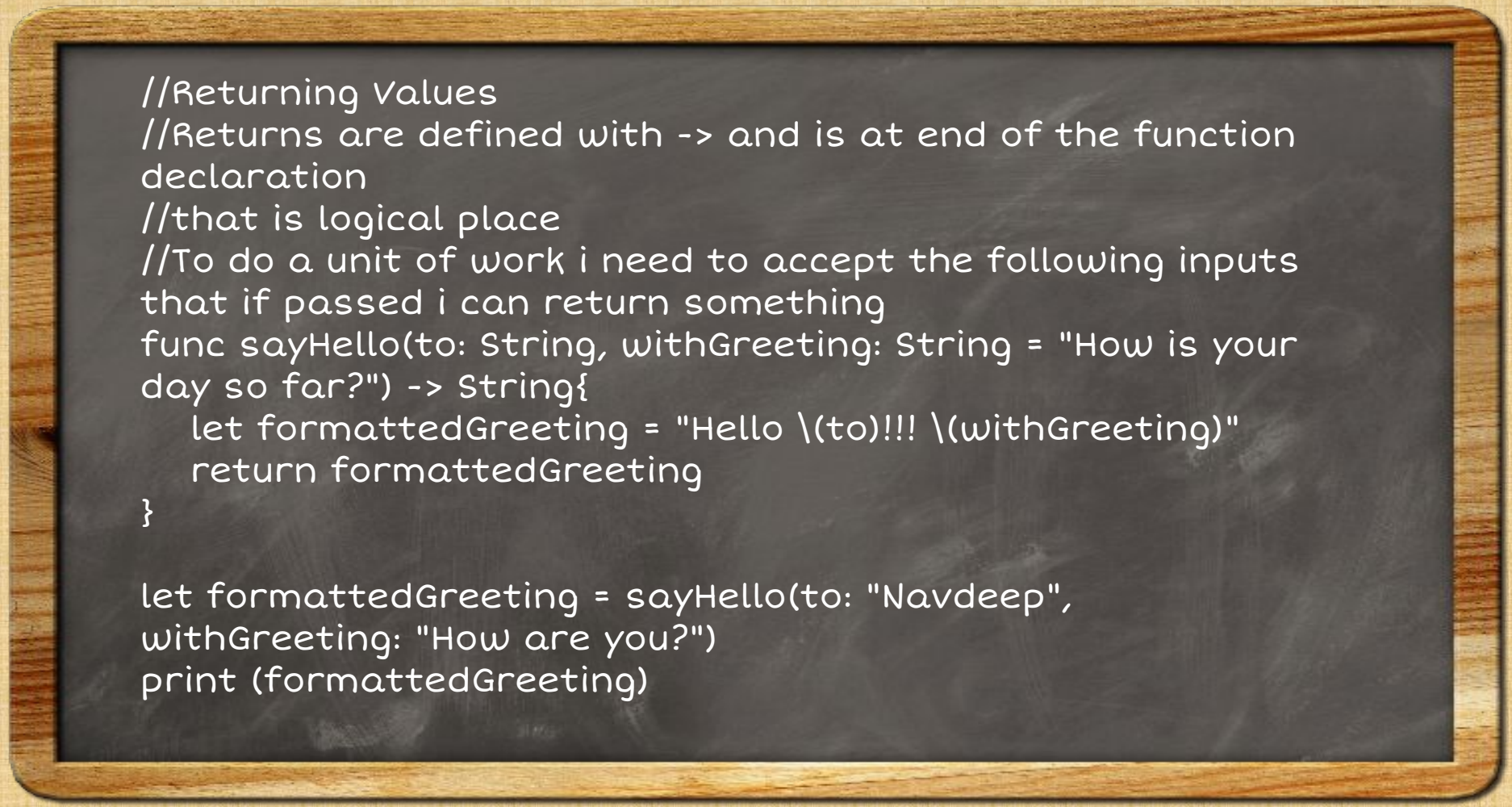
# Functions – Remove argument labels

```
//Say i don't want the callers to keep having to type to
//Simply prefix the parameter label with _
func sayHello(_ to: String){
    print ("Hello \(to)!!!")
}

sayHello("Navdeep")

//You can make any number and any combination of
argument labels extraneous by adding _
```

# Functions – Changing argument labels

```
//You can change the argument label while keeping the
parameter name same
//Note: This is useful when you need to change the labels
but then
//for changing labels you don't need to change the code
inside the function
func sayHello(toCustomerWithName to: String){
    print ("Hello \(to)!!!")
}

sayHello(toCustomerWithName: "Navdeep")
```

# Functions – Returning Values



```
//Returning Values
//Returns are defined with -> and is at end of the function
declaration
//that is logical place
//To do a unit of work i need to accept the following inputs
that if passed i can return something
func sayHello(to: String, withGreeting: String = "How is your
day so far?") -> String{
    let formattedGreeting = "Hello \(to)!!! \(withGreeting)"
    return formattedGreeting
}

let formattedGreeting = sayHello(to: "Navdeep",
withGreeting: "How are you?")
print (formattedGreeting)
```

# Functions – Swifty

```
func sayHello(to: String) -> Void{
    print("Hello \(to)!!!")
}

//Explicitly saying that method doesn't returns a value i.e.
is void
func sayHelloWithoutVoid(to: String){
    print("Hello \(to)!!!")
}

sayHello(to: "Someone")
sayHelloWithoutVoid(to: "Otherone")
```

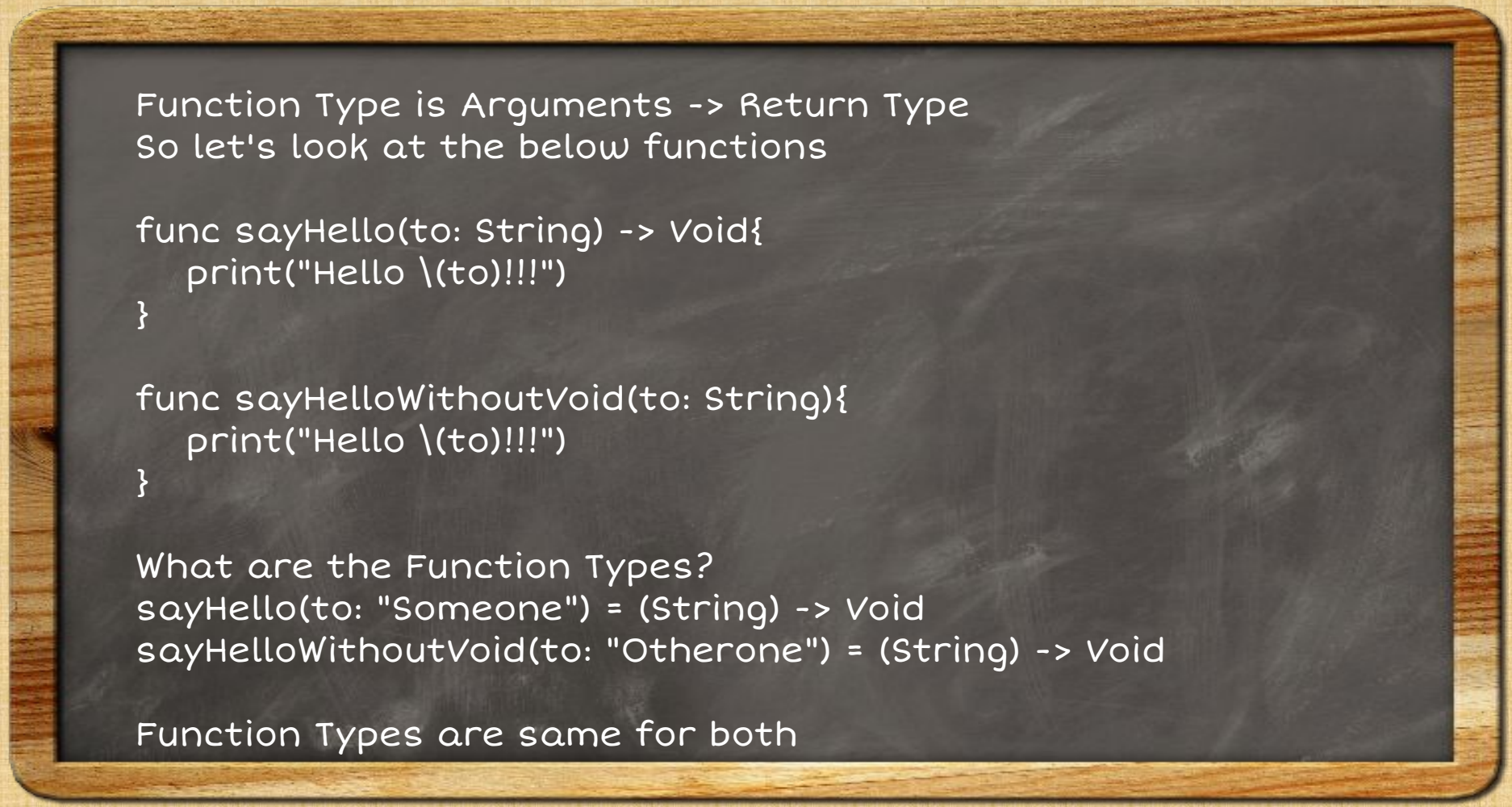
Add only if needed

# Functions – Swifty

```
func sayHello(to: String, withGreeting: String = "How is your
day so far?") -> String{
    let formattedGreeting = "Hello \(to)!!! \(withGreeting)"
    print (formattedGreeting)
    return formattedGreeting
}

//In real iOS Application, you put _ to show intent that you
don't want to care for returned value
_ = sayHello(to: "Navdeep", withGreeting: "How are you?")
```

Be explicit

# Functions Types



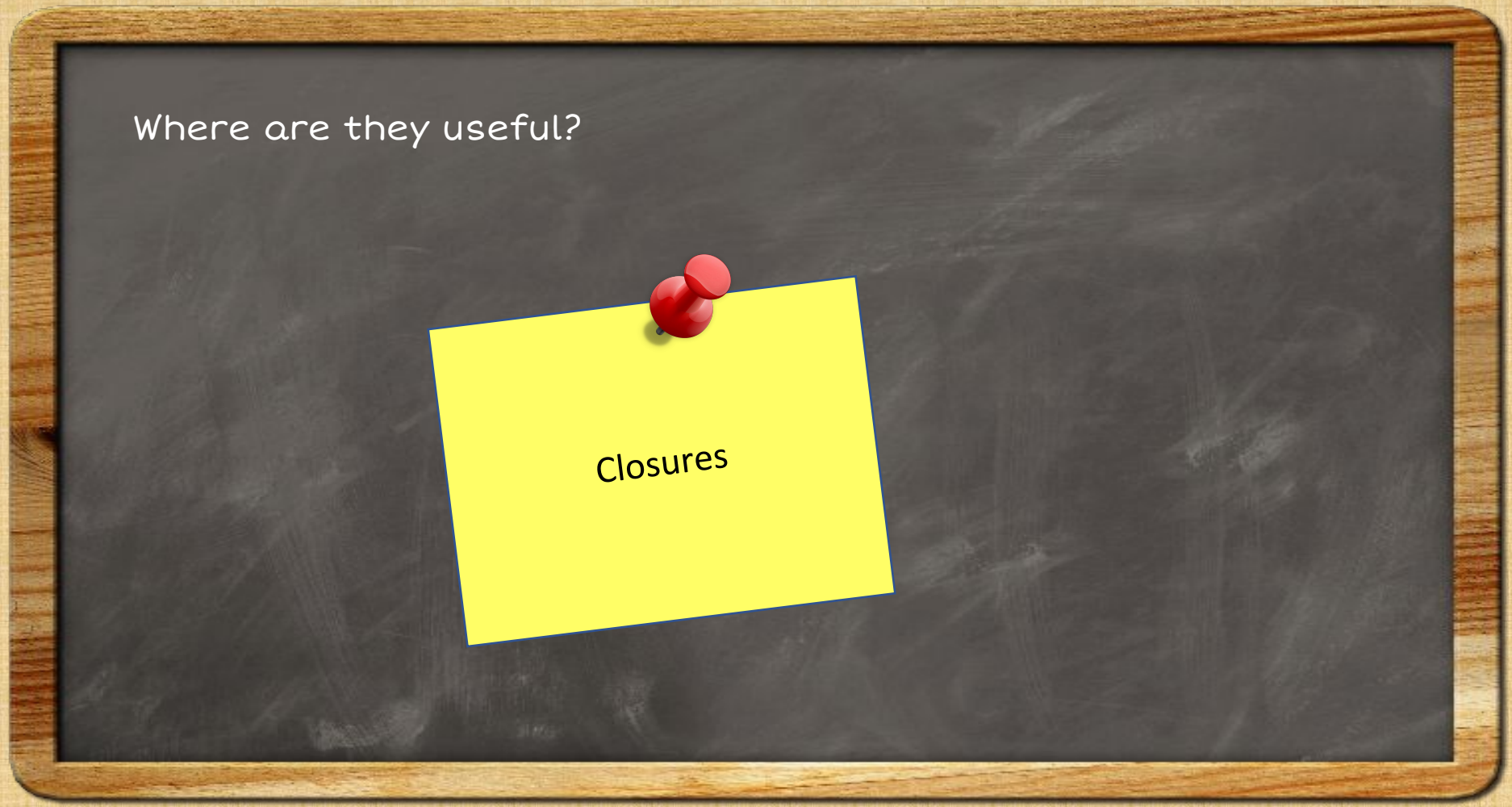
Function Type is Arguments -> Return Type
So let's look at the below functions

```
func sayHello(to: String) -> Void{
    print("Hello \(to)!!!")
}

func sayHelloWithoutVoid(to: String){
    print("Hello \(to)!!!")
}
```

What are the Function Types?
sayHello(to: "Someone") = (String) -> Void
sayHelloWithoutVoid(to: "Otherone") = (String) -> Void

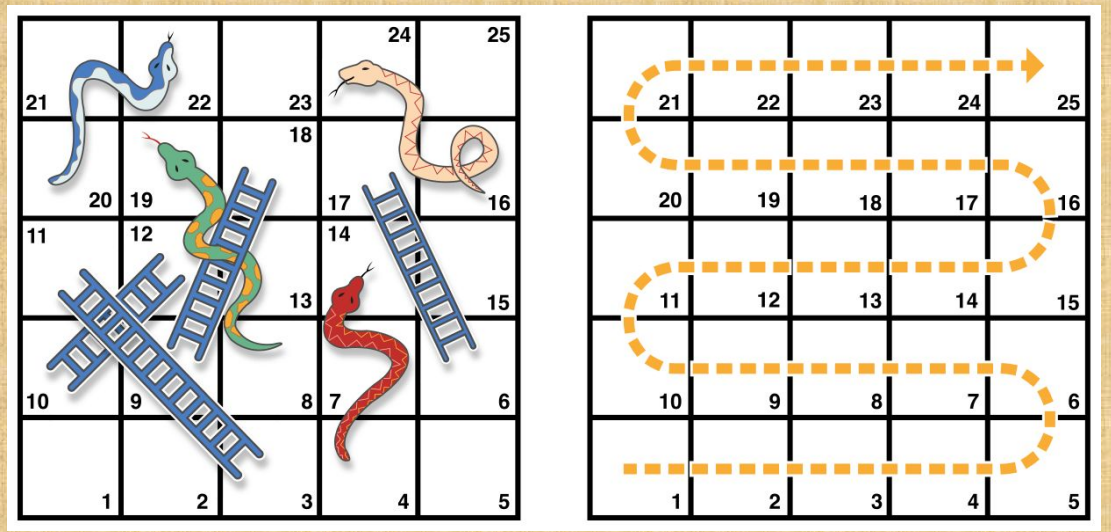
Function Types are same for both

# Functions Types



# Parting Notes

Conditional Statements




Practice:

- If else
- switch

- While, do while, for
- Create Functions
- Write functions to do addition, subtraction, multiplication and division.