ITM103 iOS Application Development

2016 S1

Topic 2: Introduction to Swift - Part 1

Objectives

- By the end of the lesson, you will be able to:
  - Understand variables, data types, collections
  - Understand basic Swift language functions, control flow
  - Understand the Object-Oriented Concepts of classes, objects, functions and properties

Introduction to Swift Part 1

What is Swift?

What is Swift

Swift. A modern programming language that is safe, fast, and interactive.

Swift is a powerful and intuitive programming language for iOS, OS X, and watchOS. Writing Swift code is interactive and fun, the syntax is concise yet expressive, and apps run lightning-fast. Swift is ready for your next project — or addition into your current app — because Swift code works side-by-side with Objective-C.

Watch Apple's Swift announcement in Jun 2014:
https://youtu.be/oo_If2FX9eI

What is Swift

Swift 3

The powerful programming language that is also easy to learn.

Watch Apple's Swift 3 announcement in WWDC, Jun 2016:
https://youtu.be/Jmjlmn0jHbw

What is Swift

Swift 3

The powerful programming language
that is also easy to learn.

Online reference
https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/

Introduction to Swift Part 1

Identifiers and Data Types

Swift Identifiers

- Identifiers, used for variables, functions, classes, etc
- Rules:
  - Cannot contain whitespace, mathematical symbols, arrows, private-use (or invalid) Unicode points, or line- or box-drawing characters.
  - Cannot begin with a number.
  - The first character must be a letter or underscore.
  - Swift variables are case-sensitive.

```
let π = 3.14159
var 你好 = "你好世界"
```

Data Types

- **Implicit Typing** - compiler determines type:

```
let i = 1                       // Int type
let j = 0xFF                    // Int represented in hex
let k = 0o77                    // Int represented in octal
let bi = 0b1111                 // Int represented in binary

let testResult = true           // Bool type

let f = 4.5                     // Double type
let s = "This is a string"      // String type
```

- **Explicit Typing** - you determine type:

```
let i : Int = 1
let f : Float = 1
let s : String = "Explicitly-typed string"
```

Data Types

- Explicit Type Conversion

```
let i : Int = 1

let f1 : Float = i              // This is NOT allowed
let f2 : Float = Float(i)       // This is allowed
```

- No Implicit Type Conversion for Arithmetic Operations

```
let i = 1
let d = 2.2                                 // This is a double

let resultDouble = Double(i) + d            // Result = 3.2
let resultInt = i + Int(d)                  // Result = 3

let result : Double = i + f                 // NOT allowed
```

Operation not allowed on two different types.

Data Types

- Increment / Decrement

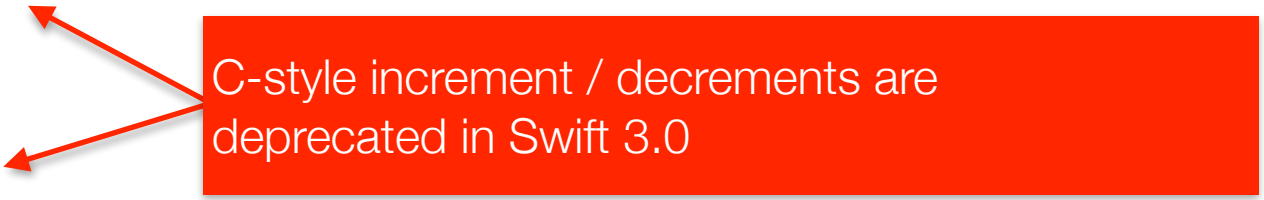
```
var i = 1

i++
print (i)

i--
print (i)

i = i + 1
print (i)

i = i - 1
print (i)
```



Data Types

- let vs var:
  - **let** declare a constant

    (a value that is assigned exactly once in its entire lifetime)

  - **var** declare a variable

```
let integerConstant = 1
// Cannot modify integerConstant

var integerVar = 2
integerVar = integerVar + integerConstant
```

Data Types

- Including Values in Strings:

```
let i : Int = 1
let j : Int = 2


let s1 : String = "The value of i = \(i)"
// s becomes "The value of i = 1"

let s2 : String = "The value of i = \(i + j)"
// s becomes "The value of i = 3"
```

Use \(…) to concatenate values into the strings

You can include expressions inside.

Data Types

- Commonly Used Data Types
  - **Int** integer (bit-length depends on platform)
  - **Float** 32-bit floating point
  - **Double** 64-bit floating point
  - **Bool** true/false
  - **String** string of characters

- Other types are available:
  - Int8, Int16, Int32, Int64, UInt8, UInt16, UInt32, UInt64, Float32, Float64, Float80

Data Types

- **Bool** type
  - Valid values are **true** / **false**

```
var isRunning = false

if (!isRunning)
{
    isRunning = true
}
```

Data Types

- **String** type:

```
var me = "I Love Swift"

print ("String length: \(me.characters.count)")

print ("To upper case: \(me.uppercased())")

print ("To lower case: \(me.lowercased())")


print ("First character: \(me[me.startIndex])")

print ("n-th character \(me[me.index(me.startIndex, offsetBy: 5)])")
print ("n-th character from last position \(me[me.index(me.endIndex,
offsetBy: -5)])")

var pos3 = me.index(me.startIndex, offsetBy: 3)
var pos5 = me.index(me.startIndex, offsetBy: 5)
print ("Substring of position 3 (incl) to 5 (excl): \(me[pos3 ..< pos5])")

for c in me.characters { print (c) }
```

Data Types

- print function is used to print to console

```
let a = 1
let b = 2.5
let c = "A";

print ("Int: \(a) Float: \(b) Char: \(c)")
```

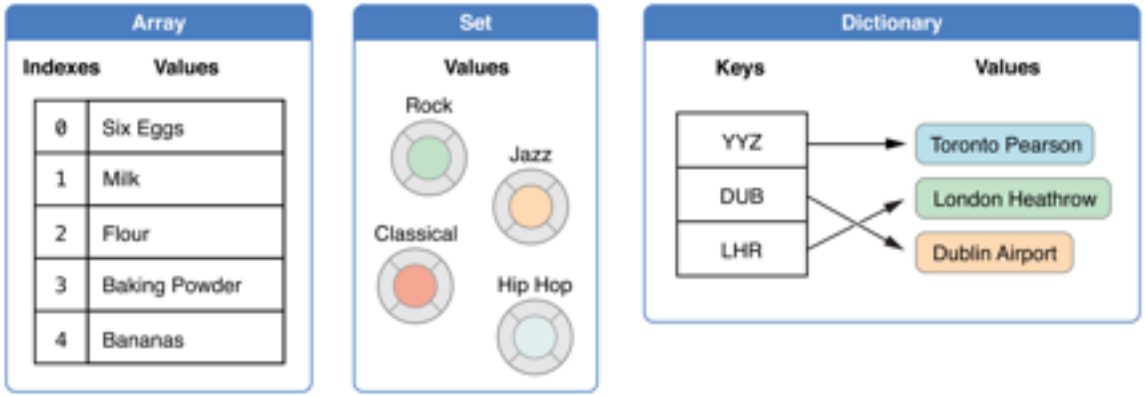
Introduction to Swift Part 1

Arrays and Collections

Arrays and Collections

- 3 types of collections:
  - **Array** standard numerical indexed array
  - **Set** items with no ordering
  - **Dictionary** key-value

    also known as: associative array / hash table



- https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/Swift_Programming_Language/CollectionTypes.html

Arrays and Collections

- Arrays
  - Declaring:

```
// Implicitly typed

var oddNumbers = [1,3,5,7,9]
var fruits = ["Apple", "Pear"]

// Explicitly typed

var evenNumbers : [Int] = [2,4,6,8,10]
var vegetables : [String] =
    ["Broccoli", "Spinach", "Cauliflower"]
```

Arrays and Collections

- Arrays
  - Iterating:

```
// Using a for-in loop
for vege in vegetables
{
    print (vege)
}

// Using a for ... loop
for i in 0 ... evenNumbers.count - 1
{
    print (evenNumbers[i])
}

for i in 0 ..< evenNumbers.count   // same as the loop above
{
    print (evenNumbers[i])
}

for i in (0 ..< evenNumbers.count).reverse()   // in reverse
{
    print (evenNumbers[i])
}
```

Arrays and Collections

- Arrays
  - Iterating:

```
// Using a for-in loop
for vege in vegetables
{
    print (vege)
}

// Using a for ... loop
for i in 0 ... evenNumbers.count - 1
{
    print (evenNumbers[i])
}

for i in 0 ..< evenNumbers.count   // same as the loop above
{
    print (evenNumbers[i])
}

for i in (0 ..< evenNumbers.count).reverse()   // in reverse
{
    print (evenNumbers[i])
}
```

Arrays and Collections

- Arrays
  - Modifying:

```
// Modifying specific item in array

evenNumbers[0] = 12

// Adding items to array

evenNumbers.append(14)
evenNumbers.insert(16, atIndex: 0)

// Removing items from array

var firstInt = evenNumbers.removeFirst()
var lastInt = evenNumbers.removeLast()
var intAt3rdPosition = evenNumbers.removeAtIndex(2)
evenNumbers.removeAll()
```

Arrays and Collections

- Arrays
  - Multi-dimensional:

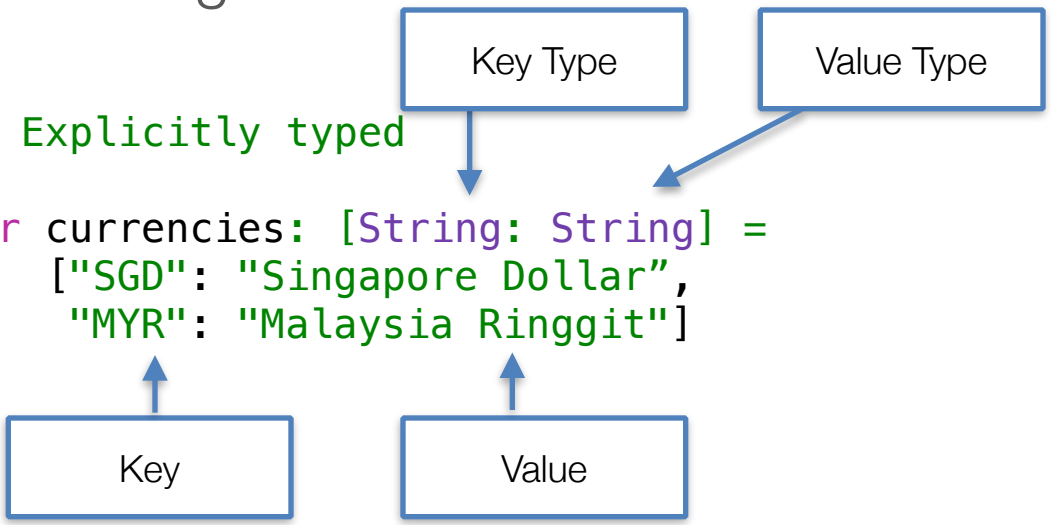
```
var ticTacToe : [[String]] = [];

for i in 0...2
{
    ticTacToe.append([".","X","."])
}

for i in 0...2
{
    var s = ""
    for j in 0...2
    {
        s = s + ticTacToe[i][j]
    }
    print (s)
}
```

Arrays and Collections

- Dictionary
  - Declaring:




```
// Implicitly typed

var currencies =
    ["SGD": "Singapore Dollar",
     "MYR": "Malaysia Ringgit"]
```

Arrays and Collections

- Dictionary
  - Iterating:

```
// Iterate through the keys
for currencyCode in currencies.keys
{
    print ("\(currencyCode) = \(currencies[currencyCode])")
}

// Iterate through the values
for currencyName in currencies.values
{
    print ("\(currencyName)")
}

// Iterate through both key and values
for (currencyCode, currencyName) in currencies
{
    print ("\(currencyCode) = \(currencyName)")
}
```

Arrays and Collections

- Dictionary
  - Modifying:

```
// Adds a key-value pair
currencies["AUD"] = "Australian Dollar"

// Removes the key-value pair
currencies["MYR"] = nil

// Clears everything
currencies.removeAll()
```

Arrays and Collections

- More on Collections:

  https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/CollectionTypes.html

Introduction to Swift Part 1

Control Flow - If, Loop, Switch

Control Flow

- The following are available for control-flow:
  - if { … } else { … }
  - for-in { … }
  - for { … }
  - while { … }
  - repeat { … } while
  - switch { case … : … }

Control Flow

- if { … } else { … }

```
let d = 1.2
if d < 2.0
{
    print ("d is less than 2.0")
}
else
{
    print ("d is greater than or equals to 2.0")
}
```

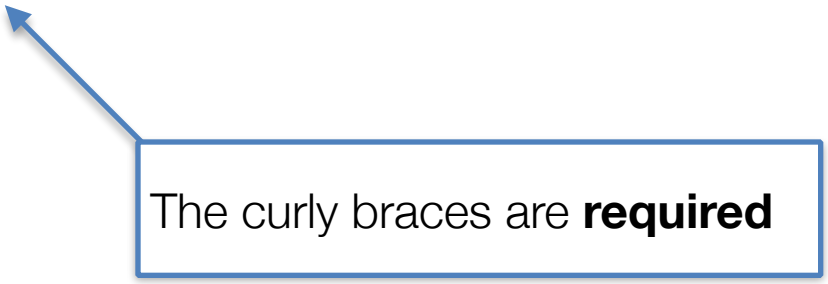
The round brackets are **not** required

The curly braces are **required**

Control Flow

- for-in { … }

```
let arr = [ 1, 3, 4, 8, 9 ]
for val in arr
{
    print (val)
}
```



Control Flow

- for { … }

```
let arr = [ 1, 3, 4, 8, 9]
for i in 0 ... arr.count - 1
{
    print (arr[i])
}

for i in 0 ..< arr.count
{
    print (arr[i])
}
```

Swift style, using "…"

~~for var i = 0; i < arr.count; i++~~
~~{~~
~~   print (arr[i])~~
~~}~~

Traditional C/C++ style, Deprecated in Swift 3.0

Control Flow

- while { … }

```
var n = 1
var sum = 0
while n < 10
{
    sum += n
    n += 1
}

print (sum)
```

Traditional while loop, without the brackets

The curly braces are **required**
even if there's only 1 line of code inside

Control Flow

- repeat { … } while

```
var n = 1
var sum = 0
repeat
{
    sum += n
    n += 1
} while n < 10

print (sum)
```

Traditional do-while loop, except in Swift it's **repeat-while**.

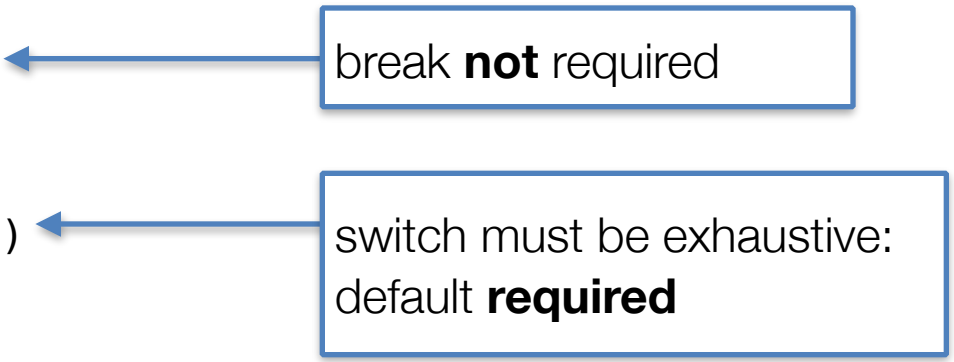
The curly braces are **required**

Control Flow

- switch { case … : … }

```
let n = 1

switch n
{
case 1: print ("Hello")
case 2: print ("World")
case 3: print ("From")
case 4: print ("Swift")
default: print ("Nothing")
}
```



Control Flow

- More on Control Flow

  https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ControlFlow.html

Introduction to Swift Part 1

Functions

Functions

- Also known as methods in other languages.

- Declaration and calling of a function differs from other languages. The arises due to a need for compatibility with Objective-C.

Functions

- Declaration and calling of a function:

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


print (add(x:1, y:2))
```

This means that this function returns a value (of type Int)

notice how the second parameter needs to be specified?

Functions

- Declaration and calling of a function:

```
func addAndPrint(x:Int, y:Int)
{
    print ("\(x) + \(y) = \(x+y)")
}


addAndPrint(x:1, y:5)
```

In Swift 3.0, the first parameter name is required.

This function does not return anything.

NOTE: It is **also** valid to declare
`func addAndPrint(…) -> Void`

Functions

- External parameter names:

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


print (add(x:1, yValue:2))
```

External parameter name
name of the parameter
(as seen by external callers)

outside the function, we must specify **yValue**, and not y

Functions

- Local parameter names:

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

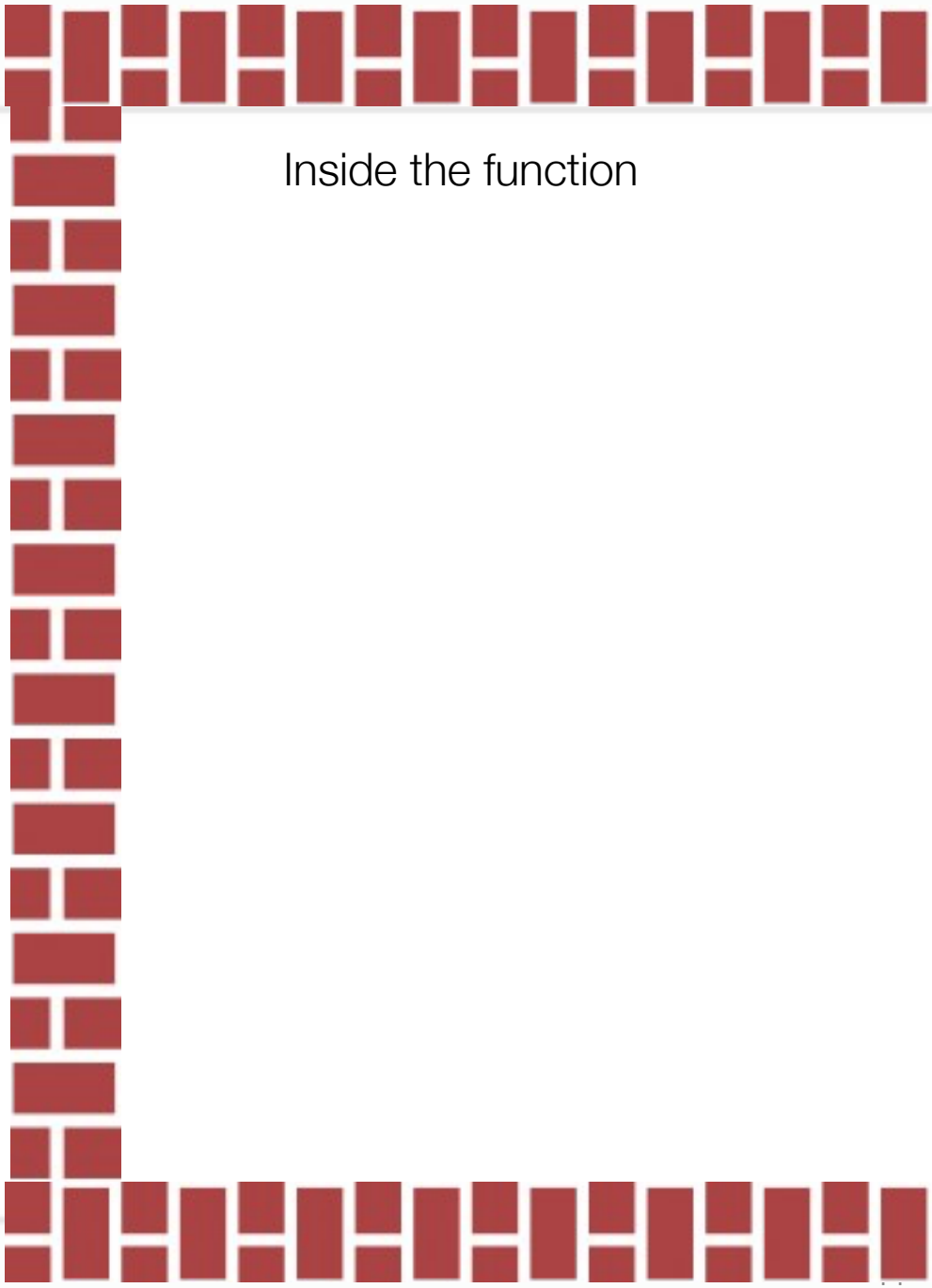
Local parameter name
name of the parameter
(as seen within the function)

inside the function we use **y,**
not yValue.

```
print (add(x:1, yValue:2))
```

Functions

Outside the function

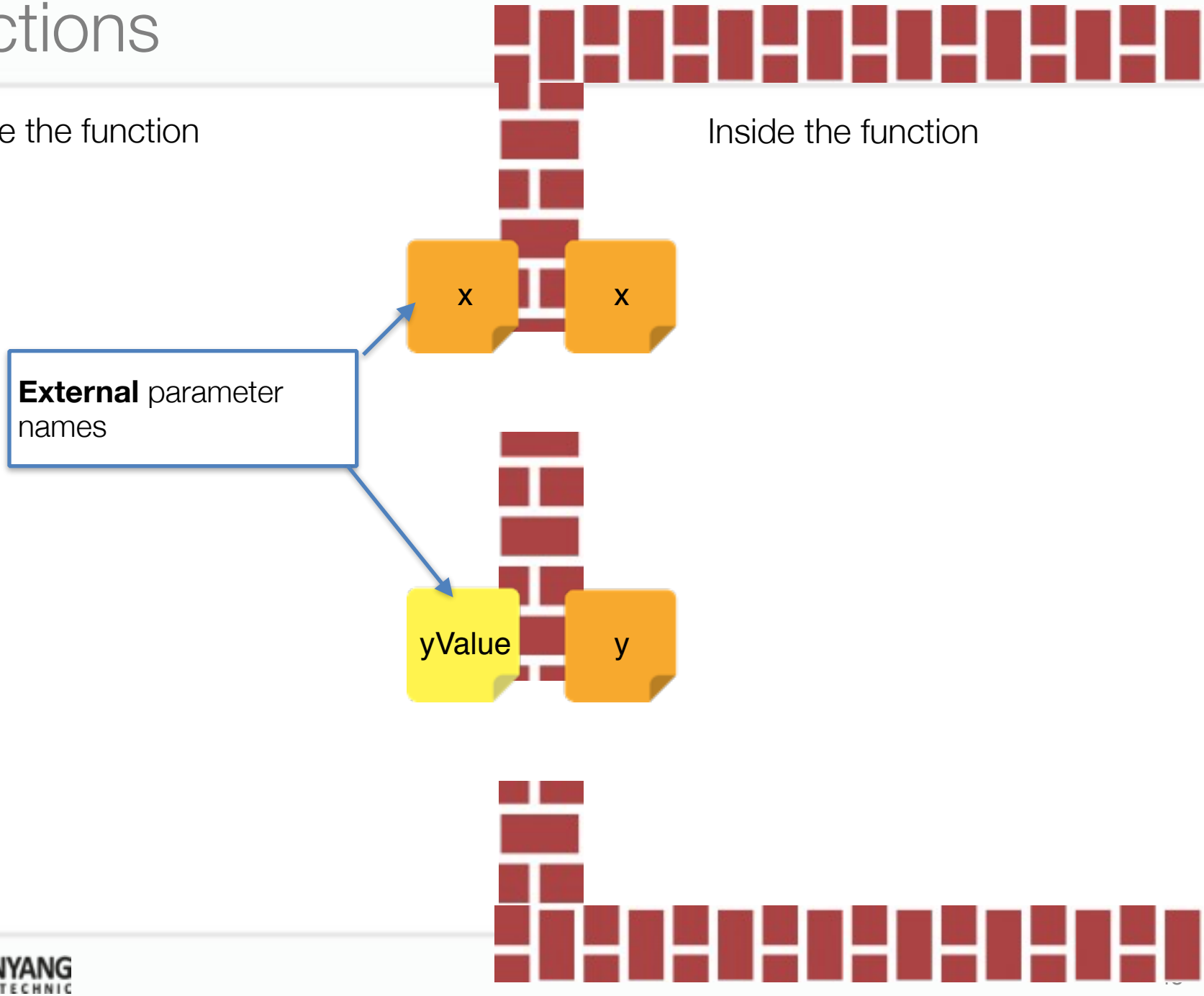


Inside the function

Functions

Outside the function

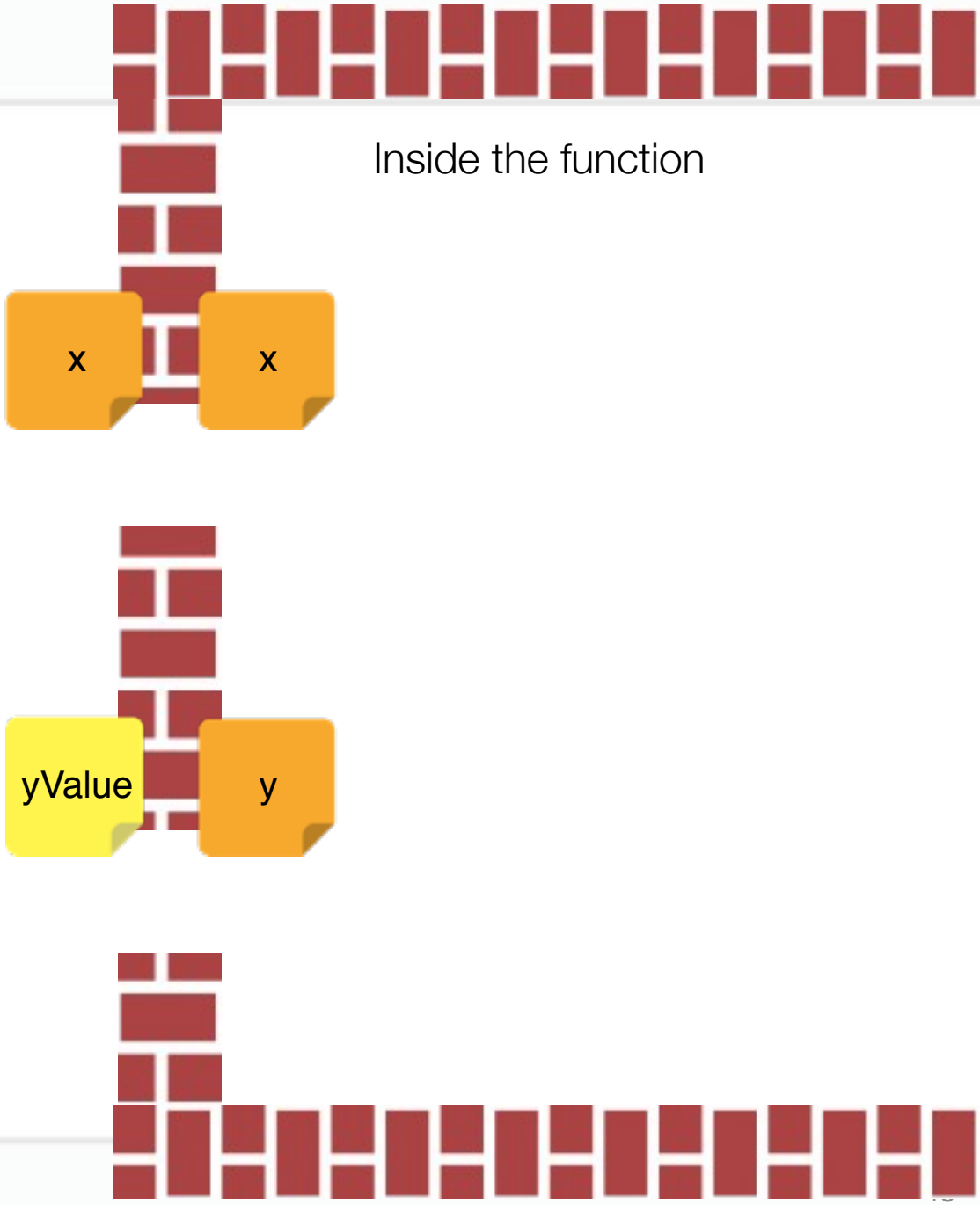
Inside the function



Functions

Outside the function

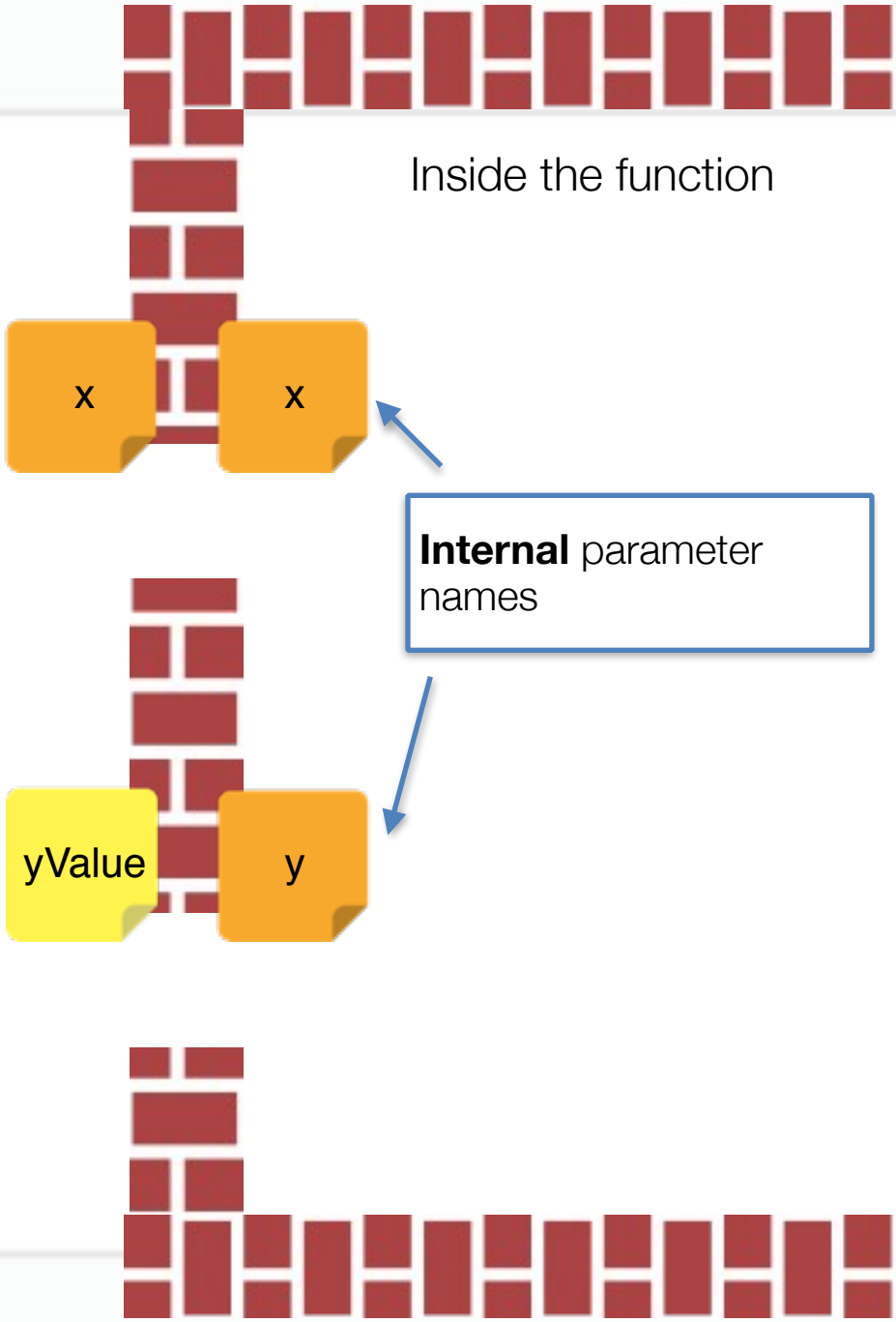
Inside the function



Functions

Outside the function

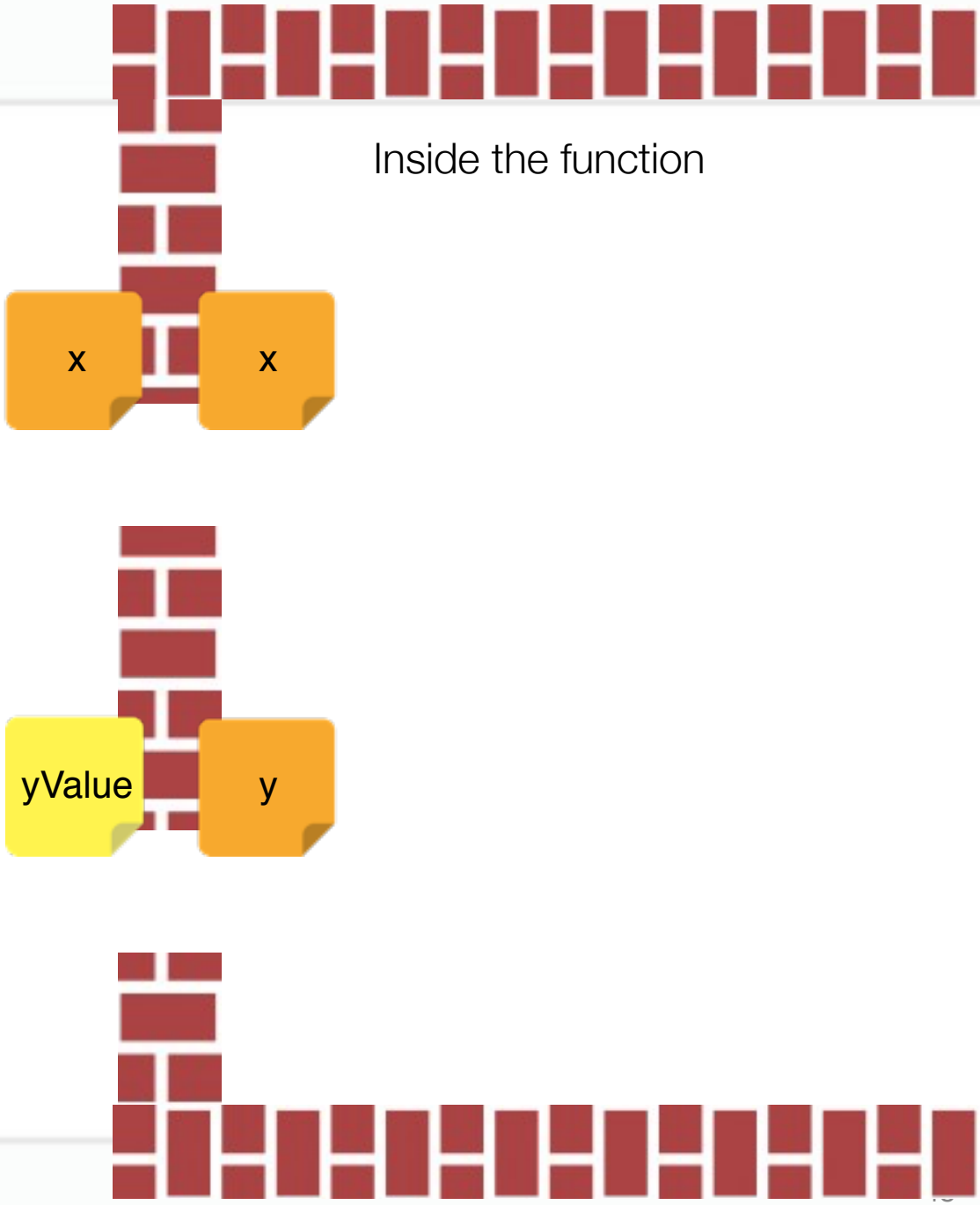
Inside the function



Functions

Outside the function

Inside the function

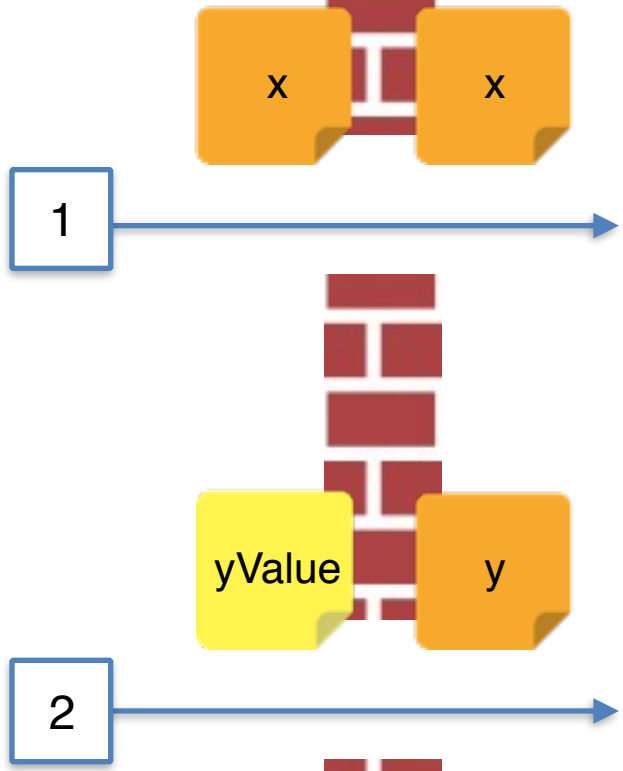


Functions

Outside the function

Inside the function

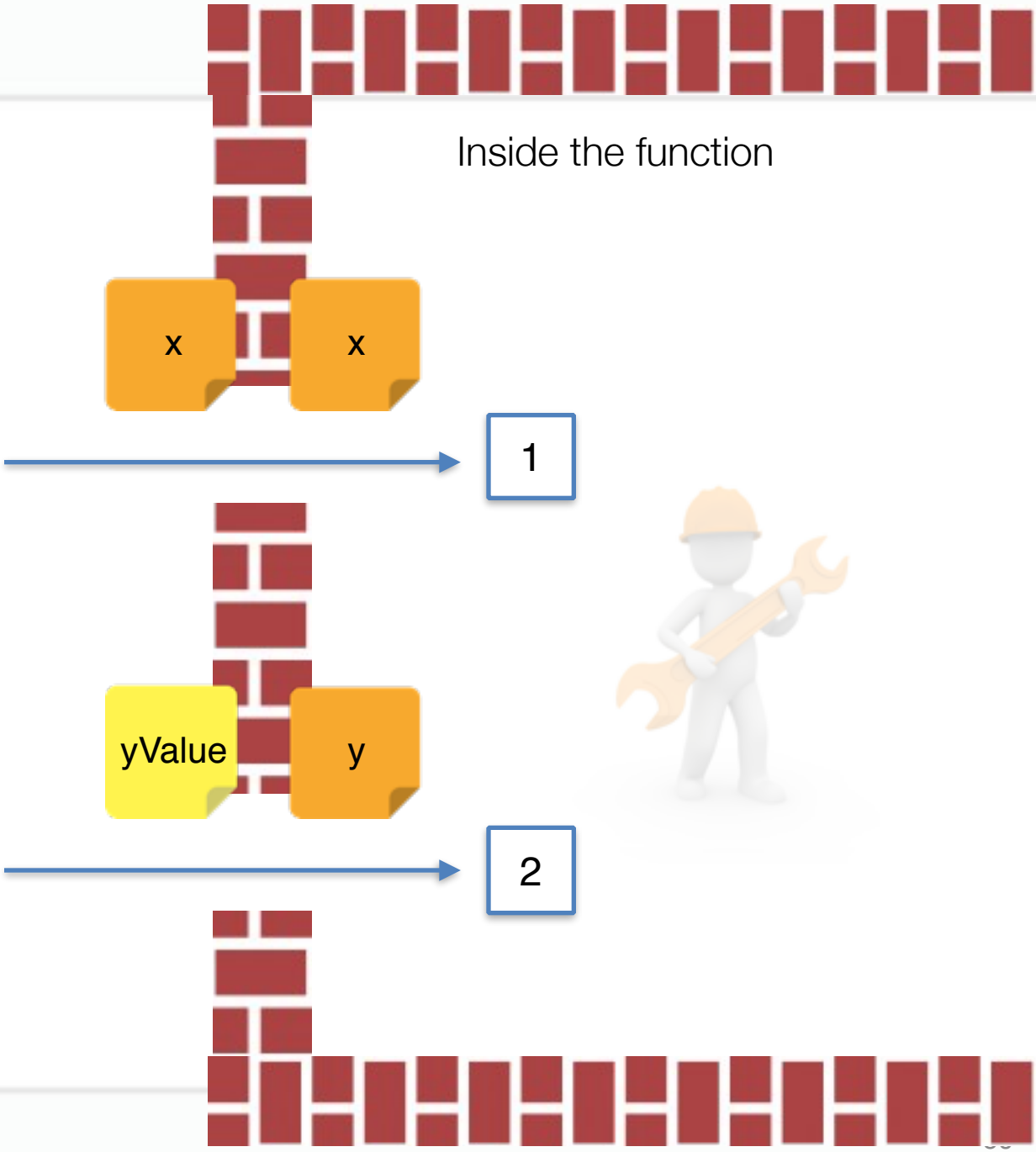
x = 1,
yValue = 2



Functions

Outside the function

Inside the function



Functions

Outside the function

Inside the function

x = 1,
y = 2,
ok, so x+y = 3

x

x

1

yValue

y

2

Functions

- External parameter names:

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


print (add(x:1, 2))
```

External parameter name
When you use an underscore, then there is no external parameter name.

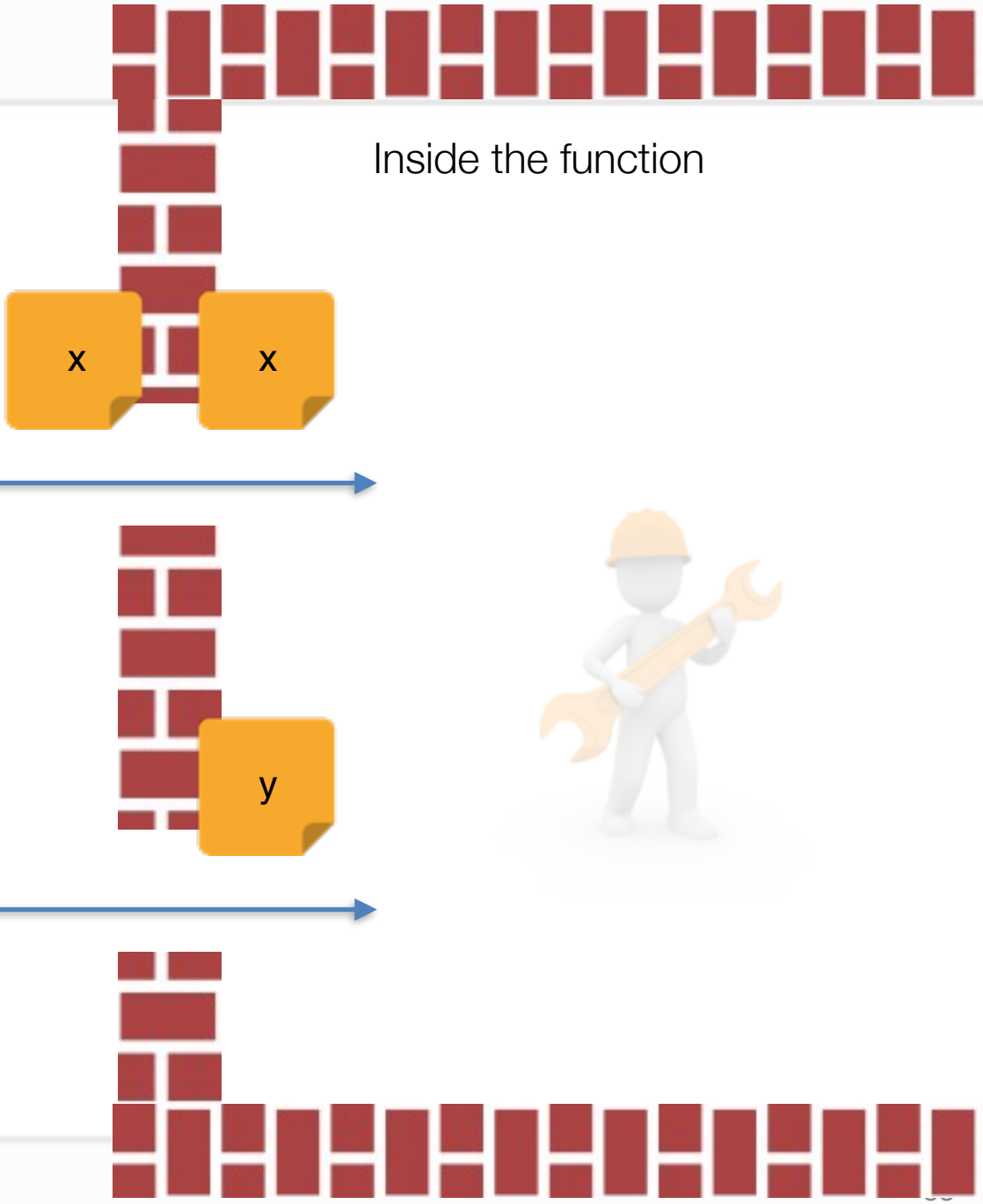
In this case the external parameter name can be omitted.

Functions

Outside the function

Inside the function

x = 1,
2nd param = 2



Functions

- **inout** parameters:

```
func swap(x:inout Int, y: inout Int)
{
    let t = x
    x = y
    y = t
}


var a = 5
var b = 6
swap(&a, &b)

print (a)
print (b)
```

inout
means the value of the parameter can change

To pass the reference of the external variable into the function, use '**&**'.

Functions

- More on Functions:

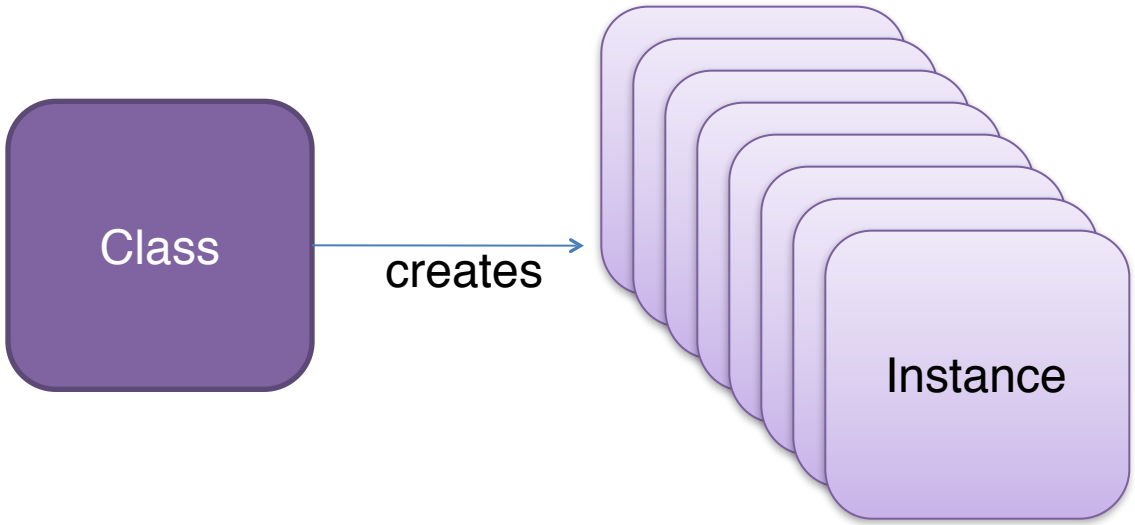
  https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Functions.html

Introduction to Swift Part 1

Object Oriented Programming

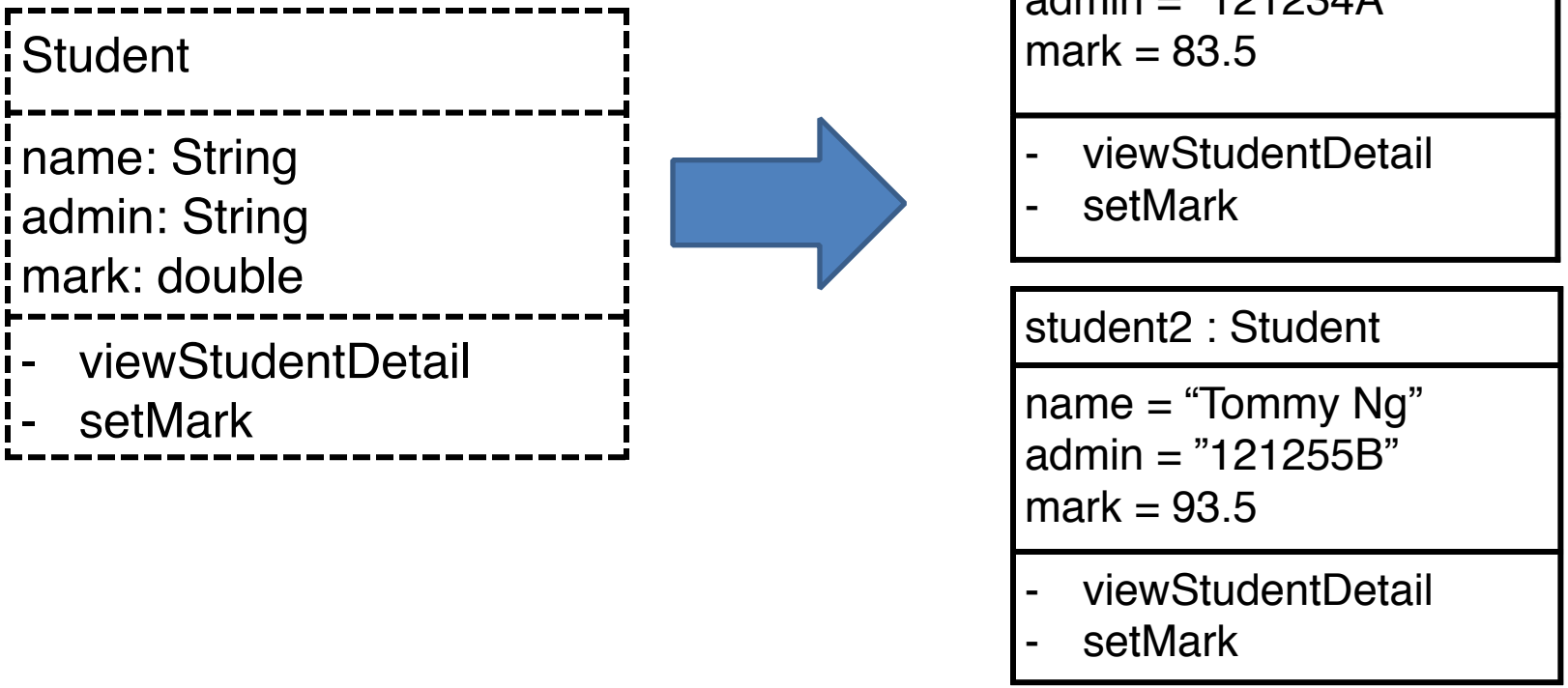
Object Oriented Programming

- Class is the blue print to create instances.



Object Oriented Programming

- Class is the blue print to create instances.



A class is a template of objects with same characteristics

Object Oriented Programming

- Class
  - Defines the grouping of data and code, or the blueprint of an object
- Instance (Object)
  - A specific allocation (instantiation) of a class
- Method
  - Function of an object, describe the behaviors
- Instance Variable
  - A specific piece of data of an object, describe the state

Object Oriented Programming

- Encapsulation
  - Keep implementation private and expose interface for public consumption
- Polymorphism
  - Different objects, same interface
- Inheritance
  - Hierarchical organization, share code, customize or extend behavior

Class Declaration

- Declaring a new class

```
class MyPoint : NSObject
{
    var x: Int
    var y: Int

    override init()
    {
        self.x = 0
        self.y = 0
    }

    func setCoordinates(x:Int, _ y:Int)
    {
        self.x = x
        self.y = y
    }
}
```

class is used to declare a class. Next comes the class name, and the super class name is after the colon.

For fields are declared without access level, Swift will automatically assign a default access level. It is **internal** in this case.

Non-optional fields must be initialised

Methods signature, more on that later.

Use self to access instance methods / variables.

Class Declaration

- Access levels in Swift:
  - **public** accessible from anywhere
  - **internal** accessible from same module
  - **private** accessible from same source file

- Implicit access levels:
  - **class** internal

  properties / methods inside:
  - public class internal
  - internal class internal
  - private class private

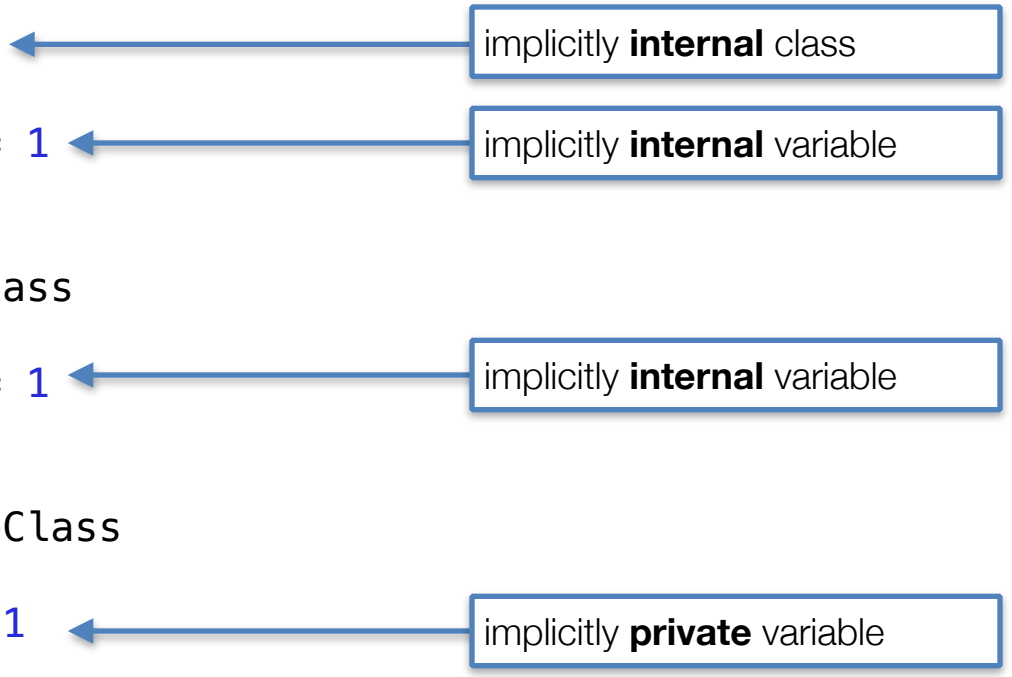
Class Declaration

- Implicit access levels:

```
class InternalClass
{
    var internalVar = 1
}

public class PublicClass
{
    var internalVar = 1
}

private class PrivateClass
{
    var privateVar = 1
}
```



Class Declaration

- Initialisers:
  - Constructors that initialises all fields in an object.

```
class Temperature {
    var temperature: Float
    var unit: String

    init(fromCelsius: Float) {
        self.temperature = fromCelsius;
        self.unit = "C"
    }

    init(fromFahrenheit: Float) {
        self.temperature = fromFahrenheit;
        self.unit = "F"
    }

}

var t = Temperature(fromCelsius: 30)
```

Class Declaration

- De-initialisers:
  - Destructors that release resources when an object is freed.

```
class Temperature {
    var temperature: Float
    var unit: String

    ...

    deinit
    {
        // Free any resources here
    }
}
```

Class Instantiation

- Instantiate a new object:

```
class MyPoint
{
    var x : Int
    var y : Int

    init(_ x: Int, _ y: Int)
    {
        self.x = x
        self.y = y
    }
}

// Initialize a constant p that cannot
// be assigned to another point object.
let p = MyPoint(1, 2)

// Initialize a variable v that can be assigned to
// another point later.
var v = MyPoint(3, 4)
```

Important: All non-optionals must be initialised with a value.

More on non-optionals later.

A "new" keyword is **not** required, unlike other modern languages.

Class Inheritance and Polymorphism

- How to inherit and override:

```
class MyShape
{
    func computeArea() -> Int
    {
        return 0
    }
}

class MyRect : MyShape
{
    var x: Int
    var y: Int

    override func computeArea() -> Int
    {
        return x * y
    }
}
```

MyRect inherits from MyShape

Polymorphism: This overrides the MyRect's computeArea function.

Class Downcasting

- Other special types:
  - **AnyObject**: an object of any class type
  - **Any**: any class / non-class type (incl func)

- Downcasting:

```
var m : MyShape

...

if (m is MyRect)
{
    var rect = m as! MyRect
    rect.x = 7
    rect.y = 6
    print ("Area = \(rect.computeArea())")
}
```

Use 'is' to test if an object belongs to a certain class.

Use 'as!' to downcast a variable of type MyShape to MyRect

Class Properties

- Getters / Setters allow greater control

```
class Rect {
    var origin = Point()
    var size = Size()
    var center: Point {
        get {
            let centerX = origin.x + (size.width / 2)
            let centerY = origin.y + (size.height / 2)
            return Point(x: centerX, y: centerY)
        }
        set {
            origin.x = newValue.x - (size.width / 2)
            origin.y = newValue.y - (size.height / 2)
        }
    }
}
```

Getters allow you control what happens during getting/setting values

newValue is the name of the new value to be set. You can change the variable name of newValue.

Object Oriented Programming

- More references:

Classes

https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html

Inheritance

https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Inheritance.html

Initialisation

https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Initialization.html

Deinitialisation

https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Deinitialization.html

Object Oriented Programming

- More references:

  Properties
  https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Properties.html

Summary

- Identifiers and Data Types
- Arrays and Collections
- Control Flow
- Functions
- Object-Oriented Programming Concepts