{ … }

{ … }

Closures in Swift

() -> ()

Motivation

- Closures are **essential part of Swift**
- Very **important programming technique** in general

Agenda

1. Passing functions as arguments
2. Closures

Passing functions as arguments

Anatomy of a Function

signature

```
func add (value1: Int, value2: Int) -> Int {
    let result = value1+value2
    return result
}
```

body

Function Signature

name *argument list* *return type*

```
add (value1: Int, value2: Int) -> Int
```

Calling Functions

```
func add(value1: Int, _ value2: Int) -> Int {
    let result = value1+value2
    return result
}
```

```
add(2, 3)
```

✅

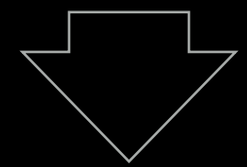
```
add("2", "3")
```

❌

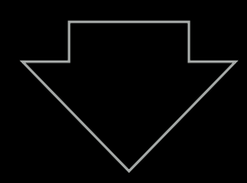
```
24  add("2", "3")   Cannot convert value of type 'String' to expected argument type 'Int'
```

The “Type” of a Function

```
func add (value1: Int, value2: Int) -> Int {
    let result = value1+value2
    return result
}
```



```
(value1: Int, value2: Int) -> Int
```



```
(Int, Int) -> Int
```

the type of the function ***add*** is from `(Int, Int)` to `Int`

Attention

What happens if we don't have arguments or a return value?

use `Void` or `()`

```
(String, String) -> ()
Void -> Bool
```

Quiz

Name the function types

```swift
// 1
func greet(greeting: String, names: [String]) {
    for name in names {
        print "\(greeting), \(name)")
    }
}

// 2
func generateRandomInteger() -> Int {
    let randomInteger = Int(arc4random())
    return randomInteger
}

// 3
func generateAndPrintData() {
  let generatedData = "this is a random string"
  print(generatedData)
}
```

```
(String, [String]) -> Void

(String, [String]) -> ()
```

```
func greet(greeting: String, names: [String]) {
    for name in names {
        print "\(greeting), \(name)"
    }
}
```

Void -> Int

() -> Int

```
func generateRandomInteger() -> Int {
    let randomInteger = Int(arc4random())
    return randomInteger
}
```

functions

Void -> Void

() -> ()

```
func generateAndPrintData() {
    let generatedData = "this is a random string"
    print(generatedData)
}
```

Passing Functions as Arguments

```
func doSomething(myFunction: (Int, Int) -> Int)
```

```
func add(value1: Int, _ value2: Int) -> Int {
    let result = value1+value2
    return result
}


doSomething(add)
```

`Closures` are **functions without names**

Anonymous Functions 

Function —> Closure

```
// function with name
func add(value1: Int, _ value2: Int) -> Int {
    let result = value1+value2
    return result
}
```

1. remove curly braces
2. add `in` keyword between argument list and function body
3. remove function name and `func` keyword
4. surround everything with curly braces

Example

```
// 1. remove curly braces
func add(value1: Int, _ value2: Int) -> Int
    let result = value1+value2
    return result
```

Example

```
// 2. add `in` keyword
func add(value1: Int, _ value2: Int) -> Int in
    let result = value1+value2
    return result
```

Example

```
// 3. remove `func` and function name
(value1: Int, _ value2: Int) -> Int in
    let result = value1+value2
    return result
```

Example

```
// 4. surround everything with curly braces
{ (value1: Int, _ value2: Int) -> Int in
    let result = value1+value2
    return result
}
```

Example

```
// passing function by name
doSomething(add)

// passing anonymous function
doSomething({ (value1: Int, _ value2: Int) -> Int in
    let result = value1+value2
    return result
})
```

More

- callbacks / completion handlers
- syntactic sugar for writing closures in Swift
- functional programming (map, filter, reduce)
- memory management pitfalls