SWIFT INTRO : CLOSURES

Closures

- Creates a function without a name, and assigns that function to a variable
- You can call that function using that variable, and even pass that function into other functions as parameters.

```
let driving = {
    print("I'm driving in my car")
}
```

```
driving()
```

Usage:

1. Running some code after a delay.
2. Running some code after an animation has finished.
3. Running some code when a download has finished.
4. Running some code when a user has selected an option from your menu.

Pros:

- Closures let us wrap up some functionality in a single variable, then store that somewhere.
- We can also return it from a function, and store the closure somewhere else.
-

Closure with parameters

Parameter are listed *inside* the open braces.
Then write **in** so that Swift knows the main body of the closure is starting.

```
let driving = { (place: String) in
    print("I'm going to \(place) in my car")
}
```

```
driving("London")
```

Return values in Closure

Similar to function, we need to use **-> String** before **in**, then use **return** to return values from closures

```
let drivingWithReturn = { (place: String) -> String in
    return "I'm going to \(place) in my car"
}
```

```
let message = drivingWithReturn("London")
print(message)
```

Closure as Parameter

If we wanted to pass that closure into a function, we would specify the parameter type as **() -> Void**. Meaning accepts no parameters, and returns **Void**

```swift
func travel(action: () -> Void) {
    print("I'm getting ready to go.")
    action()
    print("I arrived!")
}
```

```swift
travel(action: driving)
```

- **Traling closure syntax**

If the last parameter to a function is a closure, you can pass it directly after the function inside braces.

```swift
travel() {
    print("I'm driving in my car")
}
```

driving()

OR we can also do this if closure does not contain any parameters :

```swift
travel {
    print("I'm driving in my car")
}
```

Another example:

```
func animate(duration: Double, animations: () -> Void) {
    print("Starting a \(duration) second animation…")
    animations()
}
```

+ Call function WITHOUT a trailing closure

```
animate(duration: 3, animations: {
    print("Fade out the image")
})
```

+ Call function WITH a trailing closure

```
animate(duration: 3) {
    print("Fade out the image")
}
```

- **Passing A Closure with Parameter to function as a parameter:**

```
func travel(action: (String) -> Void) {
    print("I'm getting ready to go.")
    action("London")
    print("I arrived!")
}
```

```
travel { (place: String) in
    print("I'm going to \(place) in my car")
}
```

- **Passing A Closure with Parameter & Return Values to function as a parameter:**

```
func travel(action: (String) -> String) {
    print("I'm getting ready to go.")
    let description = action("London")
    print(description)
    print("I arrived!")
}
```

Calling function using trailing closure syntax

```
travel { (place: String) -> String in
    return "I'm going to \(place) in my car"
}
```

Example :

Add up all numbers in an array

```
func reduce(_ values: [Int], using closure: (Int, Int) -> Int) -> Int {
    // start with a total equal to the first value
    var current = values[0]

    // loop over all the values in the array, counting from index 1 onwards
    for value in values[1...] {
        // call our closure with the current value and the array element, assigr
        current = closure(current, value)
    }

    // send back the final current value
    return current
}
```

```
let numbers = [10, 20, 30]

let sum = reduce(numbers) { (runningTotal: Int, next: Int) in
    runningTotal + next
}

print(sum)
```

- **Shorthand parameter name**

```
func travel(action: (String) -> String) {
    print("I'm getting ready to go.")
    let description = action("London")
    print(description)
    print("I arrived!")
}
```

We can call **travel()** using something like this:

```
travel { (place: String) -> String in
    return "I'm going to \(place) in my car"
}
```

However, we can make it SHORTER :
+ Swift *knows* **the parameter** to that closure must be a string => we can remove it
+ It also knows the closure must **return** a string => we can remove that
+ This closure only has one line of code that must be the one that returns the value, so Swift lets us remove the **return** keyword too:

Therefore, we can write like this:

```
travel { place in
    "I'm going to \(place) in my car"
}
```

Moreover, we can let Swift provide **automatic names** for the closure's parameters. These are named with a dollar sign, then a number counting from 0.

We can remove **place in** and make it shorter like this:

```
travel {
    "I'm going to \($0) in my car"
}
```

- **Passing A Closure with Multiple Parameters to function as a parameter:**

```
func travel(action: (String, Int) -> String) {
    print("I'm getting ready to go.")
    let description = action("London", 60)
    print(description)
    print("I arrived!")
}
```

We use a trailing closure and shorthand closure syntax to call function like this:

```
travel {
    "I'm going to \($0) at \($1) miles per hour."
}
```

- **Return closures from functions:**

we're going to write a **travel()** function that:
+ accepts no parameters, but returns a closure.
+ The closure that gets returned must be called with a string, and will return nothing.

```swift
func travel() -> (String) -> Void {
    return {
        print("I'm going to \($0)")
    }
}
```

We can now call **travel()** to get back that closure, then call it as a function:

```swift
let result = travel()
result("London")
```

return a closure

call closure as a normal function

read more: https://www.hackingwithswift.com/quick-start/understanding-swift/returning-closures-from-functions

🟡 Closure Capturing

Closure capturing happens if we create values in **tourist5()** that get used inside the closure.
For example, we might want to track how often the returned closure is called:

```swift
151 func tourist5(person: String) -> (String) -> Void {
152     var counter = 0
153     print("\(person) visit different places")
154     return {
155         counter += 1
156         print("\(person) says Hello to \($0) \(counter) times")
157 
158     }
159 }
160 
161 let result2 = tourist5(person: "Jack") //print 1st line + return closure
162 result2("Jury") // counter = 1
163 result2("Jury") // counter = 2
164 result2("Jury") // counter = 3
```

it gets captured by the closure so it will still remain alive for that closure.

```
Jack visit different places      ← when create result2, this was printed out
Jack says Hello to Jury 1 times  ← call result2 1st time
Jack says Hello to Jury 2 times  ← call result2 2nd time
Jack says Hello to Jury 3 times  ← call result2 3rd time
```

read more: https://www.hackingwithswift.com/quick-start/understanding-swift/why-do-swifts-closures-capture-values