Closures

Closure Syntax

```
{ (parameters) -> returnType in
   statements
}
```

Definition & Types

Closures: self-contained blocks of functionality that can be passed around & used in code

- global & nested functions are special cases of closures

3 Forms of Closures:

1. Global Functions: closures that have a name & don't capture any values
2. Nested Functions: closures that have a name & can capture values from their enclosing func
3. Closure Expressions: unnamed closures written in lightweight syntax that can capture value from surrouding context

Functions vs. Closures (Key Differences)

Function:

- has a name
- has ‘func’ keyword
- no ‘in’ keyword

Closure:

- no name
- no ‘func’ keyword
- has ‘in’ keyword

Defining & Calling Functions vs. Closures

```swift
// Function Definition
func giveAFunc () {
  // func body
}
```

```swift
// Closure Definition
var giveNoFunc = { () -> () in
  // closure body
}
```

- closure is stored inside of a var so we can call it
- closure itself starts w/ { and ends with }
- call is identical

Transforming a Function -> Closure

```swift
// Example Func - soon to become closure
func sayHello(name: String) -> String {
    return "Hello \(name)"
}
```

1. Replace 'func' keyword AND funcName w/ brace '{'
2. Replace the '{' after returnType w/ 'in' keyword

```swift
// Transformed Closure
{ (name: String) -> String in
    return "Hello \(name)"
} // gives error

// fix by assigning closure to a var & calling like a func
var sayHello = { (name: String) -> String in
    return "Hello \(name)"
}
sayHello("Abhi Singh") // prints "Hello Abhi Singh"
```

Closure Expressions

- **closure expressions**: a manner of writing inline closures in a brief, focused, syntax

Example 1: Refining a Method [sorted(by:)]

a) The Sorted Method

- upon sorting, method returns new array (same size, type) correctly sorted (og array not modified)

- method takes 2 arguments of same type as array's contents & returns Bool value stating whether 1st should appear before (T) or after (F) 2nd
    - therefore, sorting closure needs to be a func of type (String, String) -> Bool

```
let names = ["Brian", "Shek", "Abhi", "John"]
```

One Way to Provide Sorting Closure: write normal func of correct type & pass it in as argument to sorted(by:) method

```
func backward (_ s1: String, s2: String) -> Bool {
    return s1 > s2
}

var reversedNames = names.sorted(by: backward) // [S, J, B, A] - sorted lexiographically
```

b) Closure Expression Syntax

- parameters in c.e.s. can be **in-out** parameters but can't have default values
- **variadic** parameters can be used if named
- **tuples** can be used as parameter & return types

Closure Expression Version of backward(_:_:) Function

```
reversedNames = names.sorted(by: { (s1: String, s2: String) -> Bool in
    return s1 > s2
})
```

- parameters & return type written INSIDE {}
- 'in' introduces start of closure's body
- can even be written on a single line
- notice how call to sorted(by:) remains same
    - pair of parentheses surround entire argument; argument, however, is now **inline closure**

c) Inferring Type From Context