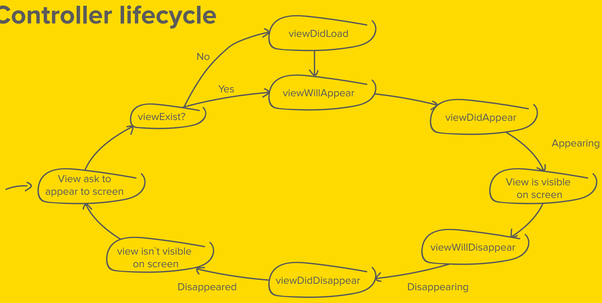
Swift
Cheat sheet

ASSIST Innovative Minds

View Controller lifecycle



Higher Order Function

Filter

Use **filter** to loop over a collection and return an **Array** containing only those elements that match an include condition.

Example:

```
// filter an array [1,2,3,4,3,3] -> [3,3,3]
let numbers = [1,2,3,4,3,3]
var filteredArray = [Int]()
for number in numbers {
    if number == 3{
        filteredArray.append(number)
    }
}
print(filteredArray)  //-> [3,3,3]
```

Or in a single line:

```
let filtered = numbers.filter({$0 == 3})
print(filtered)  //-> [3,3,3]
//filtered isn't mutable
```

Reduce

Use **reduce** to combine all items in a collection to create a single new value.

Example:

```
//calculate sum [1,2,3,4] -> 10
var sum1 = 0
for number in [1,2,3,4] {
  sum1 += number
}
print(sum1)    //-> 10
```

Or in a single line:

```
let sum2 = [1,2,3,4].reduce(0, {$0 + $1})
print(sum2)  //-> 10
```

Map

Use **map** to loop over a collection and apply the same operation to each element in the collection. The **map** function returns an array containing the results of applying a mapping or transform function to each item.

Example:

```
// transform [1,2,3,4] -> [3,6,9,12]
var transformArray = [Int]()
for number in [1,2,3,4] {
   transformArray.append(number * 3)
}
print(transformArray)   //-> [3, 6, 9, 12]
```

Or in a single line:

```
let transform = [1,2,3,4].map({$0 * 3})
print(transform)  //-> [3, 6, 9, 12]
```

Switch

You can apply a **switch** to multiple data types: **String, Enum, Int, Float**, and other types.

Example:

```
let someString: String = "first"
switch <value> {
    case <pattern>:
        <code>
    default:
        <code>
    }
```

Default parameter function

If you omit the second argument when calling this function, then the value of parameterWithDefault is 12 inside the function body.

Example:

```
func someFunction( parameterWithoutDefault: Int,
parameterWithDefault: Int = 12) { }
```

Guard

A guard statement is used to transfer program control out of a scope if one or more conditions aren't met. The condition in this example is an optional binding. The variable declared in the **guard** can be used outside the closure.

Example:

```
var age: Int?
        guard let checkedAge = age else {
                print("Age is nil")
                return
        }
        print("The age is \(checkedAge)")
```

Application enter in background

When the app enters the background, this method is called.

Example:

```
AppDelegate ->   func applicationDidEnterBackground
(_ application: UIApplication) { }
```

If the app returns from the background, this method is called.

```
AppDelegate ->   func applicationDidBecomeActive_ applica-
tion: UIApplication) { }
```

Defer

Use **defer** to write a block of code that is executed after all other code in the function, just before the function returns.

Example:

```
func printSomeString() {
   print("Hello start function!")
   defer{
      print("Hello inside the defer!")
   }
   print("Hello end function!")
}

printSomeString()    //->    Hello start function!
                             Hello end function!
                             Hello inside the defer!
```

Closure

Closures can capture and store references to any constants and variables from the context in which they are defined. This is known as closing over those constants and variables. Swift handles all of the memory management of capturing for you.

Example:

```
// FirstFile.swift
class Closure: UIViewController {
//Declare a variable
   var name: ((_ returnName: String)->Void)?
//call closure
   func returnName() {
      name?("John")
   }
}
//SecondFile.swift
let closure = Closure()
closure.name = { returnName in
  print(returnName)
}
closure.returnName()    //  -> John
```

Protocol

A **protocol** defines a blueprint of methods, properties and other requirements that suit a particular task or piece of functionality. The protocol can then be adopted by a class, structure or enumeration to provide an actual implementation of those requirements. Any type that satisfies the requirements of a protocol is said to conform to that protocol.

Example:

```
protocol CarProtocol {
   var model: String { get }
   var HP: Int { get }
   var type: String { get }
   func description() -> String
}

class Car: CarProtocol {
   var color: String
   let model: String
   let HP: Int
   var type: String
   init(model: String, HP: Int, type: String) {
      self.model = model
      self.HP = HP
      self.type = type
      self.color = "Red"
   }

   func description() -> String {
      return " Model: \(model) \n Color: \(color) \n Type: \(type )
\n Power: \(HP) HP \n"
   }
}

let car = Car(model: "Audi", HP: 120, type: "A8")
print(car.description()) // -> Model: Audi
                                Color: Red
                                Type: A8
                                Power: 120 HP
```

Reference material:

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