# Functions and Closures

## Functions

* **func doMath(operation:String, on a: Double, and b: Double) -> Double {**
* **return a + b**
* **}**
* All the parameters are actually copy and are not passed by reference.
* Input parameter as 2D array
  + **doMath("+", on: 1.0, and:1.0) // Notice that you do not require the parameter name of the first parameter.**
  + This is because it looks good because you say do math and then do + rather than saying operation:"+".
  + 'On' and 'and' are label of the parameters. Used to make the code much more expressive.
* You can return multiple values using a tuple
  + **func doMath(operation:String, on a: Double, and b: Double) -> (ops: String, result: Double) {**
  + **return (operation, a + b)**
  + **}**
* If let’s say you want to change the parameter. You cant do that as done above so you will have to declare the parameter as var.
  + **func hello(var image:[[Int]]) {**
  + **print(1)**
  + **}**
  + But in this case you are still getting copy of an image and not the actual image.
* If you want to pass something by reference and if you want any change in the function will be done to the original copy then use '**inout**'
  + **func hello(inout image:[[Int]]) {**
  + **print(1)**
  + **}**
  + Now the original image will be changed.
  + Now to call this function you have to use the '**&**' sign.
  + **hello(&image)**
  + This & sign will tell that the parameter going to the function can actually be changed.
* You can assign function to variables or constant
  + **let helloFunction = hello**
  + helloFunctions(image)
* You can have nested functions
* function can take another function as parameter
  + **func hasAnyMatches(list: [Int], condition: (Int) -> Bool) -> Bool {**
  + **for item in list {**
  + **if condition(item) {**
  + **return true**
  + **}**
  + **}**
  + **return false**
  + **}**
  + **func lessThanTen(number: Int) -> Bool {**
  + **return number < 10**
  + **}**
  + **var numbers = [20, 19, 7, 12]**
  + **hasAnyMatches(list: numbers, condition: lessThanTen)**
* Defer is used to write a block that is executed after all code is executed right after before returning.
  + **var fridgeIsOpen = false**
  + **let fridgeContent = ["milk", "eggs", "leftovers"]**
  + **func fridgeContains(\_ food: String) -> Bool {**
  + **fridgeIsOpen = true**
  + **defer {**
  + **fridgeIsOpen = false**
  + **}**
  + **let result = fridgeContent.contains(food)**
  + **return result**
  + **}**
  + **fridgeContains("banana")**
  + **print(fridgeIsOpen)**

## Closures

* Closures are like inline functions
* Arguments and return type from the body is separated using in.
* The following is a function:
  + **func performMagic(spell: String) -> String {**
  + **return spell**
  + **}**
* The following is a closure:
  + **var newMagicFunction = {**
  + **(spell: String) -> String in**
  + **return spell;**
  + **{**
* Now it can be called as follows:
  + **newMagicFunction("disappear")**
  + This will output disappear.