Function

```
typealias operation = (Int, Int) -> (Int)
// parame ters: constant (pass by value)
// want to modify?
func modify Par ams(_ value: inout Int) {}
var value = 1
modify Par ams (& value)
// overlo ading external parameter names
func multip leOf(_ multip lier: Int, and value:
Int) {}
func multip leO f(m ult iplier: Int, value: Int) {}
// overlo ading return value
func getValue() -> Int
func getValue() -> String
let value: Int = getValue()
```

Basic types

```
// Range
let range1 = 0...2
let range2 = 0..<2
// Array
let arr1: [Int] = [1,2,3]
let arr2: Array< Int> = [1,2,3]
arr.first //=> Optional
arr[0] //=> value (maybe crash if wrong index)
arr[0...2] = [1,2,3]
arr[0...2] = [1,2,3 ,4,5] // still works
for item in arr {}
for (index, item) in arr.en ume rated() {}
// Dictionary
let dic [String: Int] = ["al pha ": 1, " bet a":
2]
let dic Dictio nar y<S tring, Int>
dic["un kno w"] //=> nil
for (key, value) in dic {}
for key in dic.keys {}
// Set
```

Basic types (cont)

```
> let set: Set<Int> = [1,2,3]
// Closure
var addClosure: (Int, Int) -> Int
// long form
addClosure = { (a: Int, b: Int) -> Int in
  return a + b
}
// short form use type inference
addClosure = { (a, b) in
  return a + b
}
// even shorter $0 = a, $1 = b
addClosure = {
  return $0 + $1
}
// shortest - omit "return" if immediately return
addClosure = {
  $0 + $1
}
// when closure is the last parameter
// common syntax
arr.filter { item -> Bool in
  return !!item
}
var stock = [1.5: 5, 2.5: 10]
let sum = stock.reduce(0) { result, pair -> Double in
  return result + (pair.key * Double(pair.value))
}
```


Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Optional

```
// either value or nil
var optional: String?
// some func return optional
var parseInt = Int("1") // 1
var parseInt = Int("so mething ugly") // nil
// maybe surprise
var optInt: Int? = 10
print( optInt) // Option al(10)
optInt + 1 // error. Must be unwrapped
// Avoid
// unwrapped with !
var unwrap ped OptInt = optInt! // 1 or crash if
optInt == nil
// Use this
// optional binding
if let unwrap ped OptInt = optInt {} else {}
// can shadow like this (temporary access)
if let optInt = optInt {} else {}
// multiple unwrap
if let optInt = optInt, let optStr = optStr {}
// (better) use guard
// also better for optimi zation <- don't know
why. find out later
let optFunc: () -> Int?
func takeMeIn() {
    guard let optFunc = optFunc() else {
         // when optFunc is nil
          return // guard must be returned or
throw
   }
   // when optFunc is Int
}
// Nil Coalescing
let result = optInt ?? 0
```


Sponsored by **Readable.com**
Measure your website readability!
https://readable.com