```
// C comments work
/* So do C++/Java comments /* which are nestable! /* yay! */ */ */

//let e = 2.7
var e : Double = 2.7 // we'll see later, Double isn't actually necessary
e += 0.01828 // this wasn't legal with let definition, which declares a constant

//log(e) // this is a problem appearing before Foundation, but a warning

import Foundation // the super-useful big library of all the things

log(e) // this gave a warning appearing before Foundation

//var i : Int = 6
var i = 6 // type annotation isn't necessary, 6 is an Int literal
//var c =  "c" // this is a string!
var c : Character = "c" // type annotation IS necessary here
var s : String = "string" // back to being unnecessary here

var t = true

if ( t ) { print("true") } // print tacks a newline on automatically

print(i++)
print(++i)

/* //completely incorrect error message
func myxor (fst:Bool, snd:Bool) {
    return fst != snd
} */

/* // much more helpful error message
func myxor (fst:Bool, snd:Bool) {
    let result =  fst != snd
    return result
} */

func myxor (fst:Bool, snd:Bool) -> Bool { // correct return type declaration
    let result =  fst != snd
    return result
}

// myxor(true, true) // this, unexpectedly, doesn't work
myxor(true, snd: true) // all but first argument need the parameter label

func myand (fst: Bool, _ snd:Bool) -> Bool { // the _ here is how we prevent that
    return fst && snd
}

myand(true, true) // this works as expected from other languages

let torf = myxor(true, snd: false)
// we can embed variables, expressions, directly into strings:
print("\torf is false") // but be careful: you do need parens
print("\(torf) is false") // that's what we wanted
let string2 = "\(1+1) is 2"
```

```
print("\(myand(true, true)) is true")

s += " concat" // string concat with +
for c in s.characters.reverse() { // iterate through characters of string
    print(c)
}

var iSum = 0
for i in 3...5 { // range [3,5]
    iSum += i
}
print(iSum) //3 + 4 + 5

iSum = 0
for i in 3..<5 { // range [3,5)
    iSum += i
}
print(iSum) //3+4

s.characters.count // length of string (really length of array of characters)

//let cArr = ["a", "b", "c"] // again, this is an array of Strings
let cArr : [Character] = ["a","b","c"] // this is now an array of Characters

// optional parameters with default values
func myplus(one:Int, _ two:Int = 0, _ three:Int = 0) -> Int {
    return one + two + three
}

// myplus () // doesn't work. one isn't optional
myplus(1) // 1 + 0 + 0
myplus(1,2) // 1 + 2 + 0
myplus (1, 2, 3) // 1 + 2 + 3 (duh)

// variable number of parameters of the same type: ... type decorator
// effectively the same as taking an array of them
func mysum(nums:Int...) -> Int {
    var sum = 0
    for i in nums {
        sum+=i
    }
    return sum
}

mysum() // equivalent to []
mysum(1,2) // equivalent to [1,2]
mysum(1,2,3,4,5) // equivalent to [1,2,3,4,5]
```