Simple Values

`let` for constants; `var` for variables

Constants: value doesn't need to be known at compile time, but must assign it a value exactly once

Rule for types: constant or variable must have the same type as the value you want to assign it.

Inferring Types: Type can be inferred if value provided when creating constant/var.

Specifying types: Write after the variable, separated by colon

```
let explic itD ouble: Double = 70
```

Values are never implicitly converted to another type. If conversion required, must be done explicitly

```
let lottoS tring = " Today's winning lotto number is "
 let lottoN umber = 94
 let lottoS entence =  lottoS tring + String (lo tto Numb
er)
```

Backslash string interpolation

```
let friend Count = 2
let friend Cou ntS tring = "I have \(frie ndC ount) numbe
r of
 friend s."
```

Use three double quotation marks for strings that take up multiple lines.

Create arrays and dictionaries using brackets, and access their elements by writing the index or key in brackets.

Simple Values (cont)

```
var toys = ["la bub u", " sim ski ", "furby"]
 toys[1] = " fug gle r"
 toys.a ppe nd( " jel lyc at")
 var bestIt emO fThing = [ " summer fruit" : " man g
 bestIt emO fTh ing ["summer fruit"] = " pea ch"
```

Arrays grow automatically.

For empty array, write `excell ent Fruits = []`. For an empty dict write `medioc reF ruits = [:]`.

If assigning empty array or dict to new var, you need to specify type.

```
let arrayO fGo odC off eeC afes: [String] = []
 let dictCa fes Wit hMa tch aAn dRa tings: [String:
 = [:]
```

Control Flow

Conditionals: `if`, `switch`

Loops: `for-in`, `while`, `repeat - while`

Parentheses around condition/loop variable are optional, braces are required

```
let croiss ant Scores = [1.2, 2.3, 2.2, 4]
 for score in croiss ant Scores {
     if score > 2.5 {
       print( " Great croiss ant !!")
     }
 }
```

In `if` statement, conditional must be a boolean expression. `if scor e { ... }` is an error, not implicit comparison to zero.

You can assign if conditions to an assignment

```
 let scoreD eco ration = if teamScore > 10 {
   " □"
 } else {
   " "
 }
 print( " Sco re: ", teamScore, scoreD eco ration)
```

Control Flow (cont)

Optionals: either contains a value or contains `nil` to indicate a value is missing.
A question mark after type of value marks it as optional.

```
var option alS tring: String? = " Hel lo"
 print( opt ion alS tring == nil)
 // prints " fal se"
```

Use `if` and `let` to work with missing values. If optional value is `nil`, conditional is false and code in braces is skipped. Otherwise, optional value is unwrapped and assigned to constant after `let`, which makes the unwrapped value available inside block of code.

```
var option alName: String? = "Big Dog"
 var greeting = " Hel lo! "
 if let name = option alName {
   greeting = " Hello, \(name )"
 } else {
     greeting = " Wow !"
 }
```

`??` operator: If optional value missing, default value used instead

```
let holida yInNov: String? = nil
 let holida yInDec: String = " Bal i"
 let welcomeMsg = " Enjoy \(holi day InNov ?? holida yIn D
ec )"
```

Switches: support any kind of data and variety of comparisons

After executing the code inside the switch case that matched, program exits from the switch statement. Execution doesn't continue to the next case, so you don't need to explicitly break out of the switch at the end of each case's code.

Control Flow (cont)

```
var perfume = "Eau Rose"
switch perfume {
case " Ele ctric Cherry ":
  print( " Jasmine sambac, ambret tolide musk")
case "On a Date", "Born In Roma":
  print( " These have a black currant note.")
case let x where x.hasS uff ix( " ros e"):
  print( "Is it a fragrance with \(x)?")
default:
  print( "A delicious choice.")
}
// Prints "Is it a fragrance with rose?"
```

`for-in`: Use to iterate over items in a dict by providing pair of names f key-value pair.

```
let intere sti ngN umbers = [
  " country codes from my holida ys": [66, 1, 86, 8
,
  " cricket player s": [49, 56, 26, 30],
]

var largest = 0
for (_, numbers) in intere sti ngN umbers {
  for number in numbers {
    if number > largest {
      largest = number
    }
  }
}
```

while: condition can be at end too, to ensure loop runs at least once

```
var croiss ant sLeft = 2
while croiss ant sLeft < 100 {
    croiss ant sLeft *= 2
}
 print( cro iss ant sLeft)
// prints " 128 "
```

```
var cakesLeft = 2
repeat {
    cakesLeft *= 2
} while cakesLeft < 0
print( cak esLeft)
```

Keep an index in a loop: Use `...` for a range that includes both values for range that omits upper val

Control Flow (cont)

```
var total = 0
for i in 0...4 {
    total += i
}
print( total)
// Prints " 10"
```

Functions and Closures

functions: Use `func` to declare a function. Call a function by following its name with a list of arguments in parentheses. Use `->` to separate the param names and types from the function's return type

```
func countr yGr eet ing (gr eeting: String, country: Str
ing)
        -> String {
    return " \(g ree ting), welcome to \(coun try )."
}
greet( gre eting: " Nam ast e", country: " Nep al")
```

By default, functions use their param names as labels for their arguments. You can write a custom arg label before the param name, or write _ to use no argument label.

```
func thank(_ person: String, for action: String)
        -> String {
    return " Thank you \(person), for \(acti on)."
}
thank( " Dil lin gsb y", for: " sco oping up the poo")
```

Use tuples for compound values: e.g. to return multiple values from a func. The elements of a tuple can be referred to either by name or number.

Functions and Closures (cont)

```
func doMath (sc ores: [Int])
      -> (min: Int, max: Int, sum:Int) {
  var min = scores[0]
  var max = scores[0]
  var sum = 0

  for score in scores {
    if score > max {
      max = score
    } else if score < min {
      min = score
    }
    sum += score
  }

  return (min, max, sum)
}

let results = doMath (sc ores: [5, 3, 100, 3, 9])
print( res ult s.sum)
// prints " 120 "
print( res ults.1)
// prints " 100 "
```

Nested functions: Functions can be nested. Nested functions have acce to variables declared in outer function.

Closures: A function can take another func as one of its arguments, and can also return another func as its value.

```
func hasAny Mat che s(list: [Int],
    condition: (Int) -> Bool) -> Bool {
  for item in list {
    if condit ion (item) {
      return true
    }
  }
  return false
}
func lessTh anT en( number: Int) -> Bool {
  return number < 10
}
var numbers = [20, 19, 7, 12]
hasAny Mat che s(list: numbers, condition: lessTh an
en)
```