**Simple Variable Types**

Int, Double, Bool

*Can be implicitly declared:*

let x = 42 // Int constant

var y = 3.1 // Double variable

*Explicit declaration:*

let z: Int = 42 // Int constant

let a: Double = 3 // Double constant

let cold: Bool = true //Boolean constant

**Common Operators**

**Arithmetic Operators**

+ Add // e.g. y = a+b

- Subtract

\* Multiply

/ Divide

% Remainder

**Arithmetic shorthand**

// e.g. y += 21 same as y = y+21

+= Add and assign

-= Subtract and assign

\*= Multiply and assign

/= Divide and assign

%= Remainder and assign

**Unary operators**

- Minus //e.g. y = -x Gives negative of a number

**Boolean tests - return true or false**

< Less than

<= Less than or equal

> Greater than

>= Greater than or equal

== Equal

!= Not equal

&& Logical AND

|| Logical OR

**Conditionals**

if comparison1 {

//called if comparison 1 is true

}

else if comparison2 {

//called if comparison 2 is true

}

else {

//called if nothing else is true

}

e.g.

let age = 27

if age >= 18 {

print( "allowed to vote")

} else {

print( "not old enough to vote")

}

Curly brackets are mandatory.

Round brackets around condition are optional.

**Switch Statement**

let age = 1

switch age {

case 0,1:

print( "baby" )

case 2...4:

print( "toddler" )

case 5..<18:

print( "child" )

default:

print( "adult" )

}

First applicable case is executed then exits.

Must have a default unless all values are accounted for by other cases. Can switch on more than one value.

let num = 15

switch (num%3, num%5) {

case (0, 0):

print( "FizzBuzz" )

case (0, \_):

print( "Fizz" )

case (\_, 0):

print( "Buzz" )

default:

print( num )

}

**Strings**

**Simple string expressions and printing out**

let emptyString = ""

let greeting = "Hello"

let friendlyGreeting = greeting + ", friend"

print( friendlyGreeting ) // prints to console

**String interpolation**

let temp = "Boiling water is \(100\*1.8 + 32)F"

**String tests**

if emptyString.isEmpty ...

if greeting.hasPrefix( "He" )...

if greeting.hasSuffix( "matey" )...

**Equality tests whether made up of same characters in same order**

if emptyString == "" //this is true

if greeting == "hello" // this is false as lower case

if "aardvark" < "apple" // lexically less, so true

**Useful string features**

greeting.count // returns string length (5)

greeting.lowercased() //returns "hello"

**Manipulating characters in a string**

let me = "Chris Price"

let space = me.firstIndex(of: " ") ?? me.endIndex

let name = me[..<space]

**Arrays**

//Declare empty array of strings

var myPets: [String] = [ ]

//Different way of doing same thing

var pets = [String]( )

myPets.count // will be zero

myPets.append( "Chaz the Dog" )

myPets.append( "Dave the Goldfish" )

myPets.count // will be two

//Declare constant array with one string in it

let yourPets = [ "Idris the Guinea Pig"]

// Combines the two string arrays into one string array

var ourPets = myPets + yourPets

ourPets.count // will be three

// Replace an element in a variable array

ourPets[0] = "Chaz the Bad Dog"

// Delete an element

let deadPet = ourPets.remove(at: 2)

ourPets.count // now only two pets

// deadPet will contain the element removed

// can add extra element at specific position

ourPets.insert( "Bob the Capybara", at: 0)

// Initialise fixed size with default values

// Makes array with 200 values of 0.0

var readings = [Double]( repeating: 0.0, count: 200)

readings.removeLast() // Can delete last element

readings.count // will be 199

**For Loops**

**for item in range { statements }**

var sum = 0

for i in 1...5 { sum = sum + i } // Gives the answer 15

sum = 0

for i in 1..<5 { sum = sum + i } // Gives the answer 10

**for item in collection { statements }**

var listOfAttendees = [ "Bill", "Jane", "Jim", "Fred", "Ann" ]

for name in listOfAttendees{

print( name )

} // Will print each name in the list to console

**Equivalent to 'for item in collection' using range**

for i in 0..<listOfAttendees.count {

print( listOfAttendees[i] )

} // Will print each name in the list to console

// can get index and element by doing the following

for (index, name) in listOfAttendees.enumerated() {

print( "\(index+1). \(name) " )

}

**While Loops**

**while condition { statements }**

*e.g.*

var i = 0

while (i < 100)

{

print( "I told you so" )

i++  
} // will print "I told you so 100 times

**Functions**

// function with no parameters or result

func printHello() {

print( "Hello")

}

printHello() // usage

// function with one parameter, no result

func printMessage(message: String) {

print( message )

}

printMessage(message: "Well, hello") // usage

// To not need to name parameter in usage,

// would need to define printMessage as follows

// func printMessage(\_ message: String)

// function with two parameters, one result

func concat(s1: String, s2: String ) -> String {

return s1 + " and " + s2

}

let billAndBen = concat(s1: "Bill", s2: "Ben")

// function with no params, tuple as result

func returnTuple() -> (String, Int) {

return( "Page not found", 404 )

}

let result = returnTuple()

// String in result can be accessed as result.0

// Int in result can be accessed as result.1

**Structures**

struct Person{

// properties

let firstName: String

let lastName: String

// computed property

var fullName: String {

return firstName + " " + lastName

}

// Method - like function but on instance

func printName(){

print( fullName )

}

}

//Create an instance of a Person

let newPerson = Person(firstName: "Jim",

lastName: "Jones")

//Call method for instance newPerson

newPerson.printName()

let oldPerson = newPerson

//When you assign a structure to a new variable, //its value is COPIED (unlike classes).

// Structures have automatically generated //initialisers, but you can also write your own.

**Classes**

//Class syntax would be identical for the Person

//class, but need to write an initialiser.

class Person{

let firstName: String

let lastName: String

// Needs this initialiser

init(fName: String, lName: String) {

firstName = fName

lastName = lName

}

var fullName: String {

return firstName + " " + lastName

}

func printName(){

print( fullName )

}

}

let newPerson = Person(fName: "Jim",

lName: "Jones")

newPerson.printName()

let oldPerson = newPerson

//When you assign a class to a new variable,

//both variables point to the same structure

**Class inheritance**

// Can declared subclass of existing class

// Subclass inherits properties and methods

// Extra properties and methods can be declared,

// but extra properties must be initialised.

// Where methods are replaced by more specific

//versions, they need the qualifier 'overrides'

class Doctor: Person{

let specialism: String

// Needs this initialiser

init(fName: String, lName: String, spec: String) {

specialism = spec

super.init(fName: fName,lName: lName)

}

override func printName(){

print( "Dr " + fullName+" does "+specialism)

}

func seePatient( patientName: String ) {

print( "Hullo, " + patientName

+ ", I'm Dr " + fullName )

}

}

let doc = Doctor(fName: "Jane",

lName: "Jones", spec: "Pediatrics")

doc.printName()

doc.seePatient(patientName: "George")

**Enumerations**

enum TempType {

case degF

case degC

}

var tempType = TempType.degC

if tempType == .degF {

print("In Fahrenheit")

} else {

print("In centigrade")

}

// More complex example

enum DaysOfWeek {

case monday, tuesday, wednesday,

thursday, friday, saturday, sunday

}

var today = DaysOfWeek.monday

switch today{

case .saturday, .sunday:

print( "Chill - it's the weekend" )

default:

print( "Apply nose to grindstone" )

}

// Will print "Apply nose to grindstone"

today = .saturday

// We know type of today now,

// so don’t need to say “DaysOfWeek”

**Dictionaries**

//Create a dictionary containing our pets

var pets = ["Dave": "Fish", "Chaz": "Dog"]

pets["Idris"] = "Pig" //Adds Idris to pets

pets.isEmpty // returns false

pets.count // returns 3

pets["Idris"] // returns " Pig"

pets["Dog"] // returns nil

// Update value for Chaz

pets["Chaz"] = "Bad Dog"

// Loop over key/value pairs

for (name, animal) in pets {

print( "\(name) is a \(animal)" )

}

//Delete Chaz from dictionary

pets["Chaz"] = nil // Deletes Chaz entry

// Loop over keys

for name in pets.keys {

print( "\(name) is a pet" )

}

// Loop over values

for animal in pets.values {

print( "We have a \(animal)" )

}