Swift 4 Cheat Sheet

by MerlinElMago (MerlinElMago) via cheatography.com/32585/cs/10104/

Variable declarations

let variable [: **type**] = *value*	Define a constant, providing a type.
let variable [: **type**] = *value*	Defina a variable, providing a type.

Swift inferes types based on the type of contents that is assigned at creation time. The type declaration is optional.

Comments

```
/* This is a multiline comment in Swift */
// This is a single line comment in Swift
```

Comments may be either single or multiline and may even be written after the last command on a line.

Data types

Int	Default Integer value depending on the running platform
Int8	1 Byte = -127...+127 (signed)
Int32	4 Byte = -2 147 483 648...+2 147 483 648
Int64	8 Byte = -9 223 372 036 854 775 808... +9 223 372 036 854 775 808
Float	(~6 digits precision) $1.2e^{-38}$...$3.4e^{+38}$
Double	(~15 digits precision) $2.3e^{-308}$...$1.7e^{+308}$
Bool	Can be either true or false.
Character	Only one character in length.
String	Same as character but unlimited in length
Optional	Can be either a value (of any type) or *no value*.

All Int values may be preceded by a capital U to indicate an unsigned range, which shifts min values to 0 and max values to their double.

Literals

Integer

`var myInt = 10`	Normal, decimal notation
`var myInt = 0xa`	*Hexadecimal* notation of 10
`var myInt = 0b1001`	*Binary* notation of number 10
`var myInt = 0o11`	*Octal* notation of number 10

Floating Point

`var myFloat = 12.345`	Normal, decimal number notation.
`var myFloat = 1.2345e1`	Exponential notation of the above number.
`var myFloat =`	Hexadecimal double (which I honestly don't use)

String

All characters should be enclosed in double quotes. We may use the following escape sequences to escape special characters.

`\0`	NULL Character
`\\`	Backslash character
`\b`	Backspace
`\f`	Formfeed
`\n`	Newline
`\r`	Carriage Return
`\t`	Horizontal Tab
`\v`	Vertical Tab
`\"`	Double Quote
`\'`	Single Quote
`\055`	Character represented by Octal number 55
`\x99`	Character represented by Hex-Number 99

Boolean

`true`	Value is true
`false`	Value is false
`nil`	There is no value

By **MerlinElMago**
(MerlinElMago)

cheatography.com/merlinelmago/

Not published yet.
Last updated 10th October, 2017.

Optionals

Optionals is actually a new datatype within Swift. As the use and understanding of optionals are of utmost importance, let's have a look at them in depth.

`var myInt : Int?` means that the variable named *myInt* **could** contain an integer value but **does not** have to.
If it **does** contain a value it will be of type *Some(Type)*, if it **doesn't** it will be *None*

`myInt!` in the code, means that the programmer will make sure that myInt contains a value of type *Some(T)*. This is called **Forced Unwrapping**.

`var myInt : Int!` allows a normal use (i.e. the use without ? nor !) of the variable in our code. This is called **Automatic Unwrapping**.

To check if an optional contains an actual value or not, we use a code block like this one:
`if myInt != nil { //code } else { //code }`

Another technique is called **Optional Binding**, and allows to check for a value directly during declaration of the variable:
`if let myInt = myUnce rta inInt { //code } else { //code }`

Operators

Arithmetic

`a * b`	**Multiply** a by b.
`a / b`	**Divide** a by b.
`a % b`	**Modulus** (remainder after /)
`a + b`	**Add** a to b.
`a - b`	**Substract** a from b.
`++a` or `a++`	Pre or post increment.
`--a` or `a--`	Pre or post decrement.

Comparision

`a == b`	A is **equal** to b.
`a != b`	A is **not** equal to b.
`a > b`	A is **greater** than b.
`a < b`	A is **less** than b.
`a >= b`	A is **greater** or equal to b.
`a <= b`	A is **less** or equal to b.

Logical

`a && b`	Is true while a **and** b are true.
`a || b`	Is true while a **or** b are true.

Operators (cont)

`a ! b`	Is true while **only one** of a or b is true.

Bitwise

`&`	Bitwise **and**.
`|`	Bitwise **or**.
`^`	Bitwise **xor**.
`~`	Bitwise ones **complement**.
`<<`	Bitwise **shl**.
`>>`	Bitwise **shr**

Assignment

`a = b`	Assign b to a.
`a += b`	Assign a+b to a.
`a -= b`	Assign a-b to a.
`a *= b`	Assign a*b to a.
`a /= b`	Assign a/b to a.
`a %= b`	Assign a%b to a.
`a <<= b`	Assign a shl b to a.
`a >>= b`	Assign a shr b to a.
`a &= b`	Assign a and b to a.
`a ^= b`	Assign a xor b to a.
`a |= b`	Assign a or b to a.

Range

`( 1..4 )`	Closed range, i.e. 1,2,3,4.
`( 1..<4 )`	Half open range, i.e. 1,2,3.

Miscellaneous

`( a==b ) ? x : y`	Ternary operator. If a is equal to b then return x, if not return y.

Strings

`var myString = ""` `var myString = String()`	Both create an empty string.
`var myString = " one "` `myString += " two"`	Concatenates two strings, and returns "one two".

Cheatography

Swift 4 Cheat Sheet

by MerlinElMago (MerlinElMago) via cheatography.com/32585/cs/10104/

Strings (cont)

`var myString = "one " + " two "`	Another wat to concatenate two strings.
`var myName = "Tony"` `var myGreeting = "Hello \(myName)"`	
	This inserts a string into another string. Note that this is not limited to strings, but also works with numbers.
`var myAge = 18` `var myGreeting = "Age: \(myAge)"`	

String relevant functions and methods

`myStri ng.i sEmpty`	Returns true if empty and false if not.
`var myString = "two words"`	
`myString.localizedCapitalized`	Returns "Two Words"
`myStri ng.l oc ali zed Low er case`	Returns "two words" (this appears to be no change, but in fact, all characters are converted to lowercase.
`myStri ng.l oc ali zed Upp er case`	Returns "TWO WORDS"
`count( myString )`	Returns 9

Consult: Docs for further information.

Arrays

Arrays (cont)

`myArray[0]`	Retu the f elem of th arra

Multi-Dimensional Arrays

`var myArray:Array<Array<Type>> = Array(Array())`	Sam as abo but mult dim- ensi ally.
`var myArray:[[Type]] = [[]]`	
`var myArray = [[Type]]()`	

Array relevant functions and methods

`myArray.count`	Returns the number of items go an array.
`myArray.isEmpty`	Returns true if it is empty and false if not.
`myArray.first`	Returns first element of array
`myArray.last`	Returns last element of array
`myArray.append(value)`	Appends an element to an array (there are variations to this)
`myArray.insert(value at:X)`	Inserts *value* at position X
`myArray.remove(at: X)`	Removes the element at position X
`myArray.index(of: value)`	Returns the index (i.e. position) of *value* inside the array.

Consult: Docs for further information.

Dictionaries

`var myDict: [Type:Type] = [:]`	This creates an empty dictionary.
`var myDict = [key:value]`	Creates a dictionary with given *key:value* pairs.
`for (key, value) in myDict { //code }`	Iterate over a dictionary accessing *key:value* pairs in each iteration.

`var myArray:Array<Type> = Array()`	This creates an empty array.
`var myArray:[Type] = []`	Shorthand method to create an array with values X, Y and Z
`var myArray = [Type]()`	This is the shortest of the shorthands available.
`var myArray = Array(repeating: X, count: X)`	This allows to create an array with a default value and a given size.

By **MerlinElMago**
(MerlinElMago)

cheatography.com/merlinelmago/

Not published yet.
Last updated 10th October, 2017.

Dictionaries relevant functions and methods

`myDict.count`	Returns the amount of data pairs available.
`myDict.is Empty`	Returns true if the dictionary is empty.
`myDict.up dat eValue(value, forKey: ke y)`	Updates a *value* inside the dictionary.
`myDict.in dex (fo rKey: key)`	Returns the index where given *key* is located.
`myDict.re mov eVa lue (fo rKey: key)`	Removes a *key:value* pair.

Consult: Docs for further information.

Flow control (Decision making)

if..else

```
if a==b {
 //code
} else {
 //code
}
```

If a is equal to b, the first block of code is executed. If not, then the last block of code is executed. The *else* block may be present or not.

Switch

```
switch variable {
  case 1:
   //code1
  case 2:
   //code2
   fallthrough
  case 3:
   //code 3
  default:
   //code default
 }
```

The switch statement checks a variable for a given value. No break statement is needed. If we want a fallthrough to happen, we need to specify this in the code block. In this example, case 2 always gets executed together with case 3. The default statement catches all non matching cases.

Ternary oprerator

```
( condition) ? value1 : value2
```

Flow control (Decision making) (cont)

The ternary operator is described in the *Miscellaneous* section of the *Operators* block as well. It is basically a very condensed *if..then* statement.

Nil-Coalescing oprerator

```
( a ?? b )
```

Unwraps a and returns it, if it is NOT nil. If it is nil, then b is returned.

Flow Control (Looping)

for..in

```
for value in array {
  //code
 }
```

This loop, iterates over each of the elements within an array (or dictionary) and the variable *value* takes the value of an element, one at a time.

for loop

```
for initCond; chkCond; operator {
 //code
 }
```

The for loop basically a counts from an initial condition (*iniCond*) to an end condition (*!chkCond*) using an operator to change the value of the counter.

while loop

```
while condition {
 //code
 }
```

This code block is executed whenever the *condition* is true. The *condition* thus gets evaluated at the beginning.

do..while loop

```
do {
  //code
 } while condition
```

This is a special case of the while loop, as the evaluation is at the end, instead of the beginning. ! The code block is executed at least *once*.

Flow Control (Statements)

`fallthrough`	As seen before, the fall through statement is used within the *switch* block. It allows tho execute the next case of the matching section.
`continue`	Continue allows to skip the rest of a iteration and
`break`	This statement allows to break out of a loop. It skips the rest of a iteration and aborts all subsequent loops.

Functions

```
func fName(parameter:Type)->Type {
 //code
 return Value
}
```

The function has to return a value of the same type that has been declaren in it's header. If we don't declare a return value, none has to be returned. Parameters are optional as well. A function without return value nor parameters, could look like this:

```
func fName() {
 //code
}
```

The function parameters may have a label. This would look like this (only declaration):

```
func fName(label parameter:Type)->Type {
```

Per definition, all function parameters are passed as constants, so they are immutable. If we had to make changes to those parameters inside our function, and have this changes reflected outside as well, we'd use in/out parameters.

```
func fName(label parameter:inout Type)->Type {
```

Variadic parameters allow us to use an undefined amount of parameters (of the same type) inside our function.

```
func fName(parameter:Type...)->Type {
```

And lastly, if we wanted a default value to be assumed for a given parameter, we'd use:

```
func fName(label parameter:Type=value)->Type {
```

Consult: Docs for further information.

Closures

```
{( Parameters ) -> Type in //code }
```

Same as a function but without a name. Can be passed around in func

```
let sum= {( n1:Int, n2:Int)->Int in
 return n1+n2
}
```

This would return the sum of two numbers when called like this:

```
let myResult = sum(2,3)
```

If the type of variables can be inferred, this is true for the sort(by:) funct writing...

```
let myElements = ["two","three","four"]
let mySort edE lements =
myElements.sorted(by: {( prm1: String, prm2: String
 unt})
```

...we could write (only closure part)...

```
let mySort edE lements =
myElements.sorted(by: { prm1, prm2 in return prm1.c h
```

This would return *"three","four","two"*. But it doesn't stop here. To write

```
let mySort edE lements =
myElements.sorted(by: { prm1, prm2 in prm1.c har act
```

Instead of declaring variables, names we can also use shorthands for t

```
var myTest: (Int, Int)->Int
myTest = { $0 + $1 }
myTest(1,2) //This would return 3
```

Swift is even able o infer most of the closure and reduce it to the minim

```
let mySort edE lements =
myElements.sorted(by: > )
```

Occasionaly, if a closure results to be very long, it can be written at the

Cheatography

Swift 4 Cheat Sheet

by MerlinElMago (MerlinElMago) via cheatography.com/32585/cs/10104/

Closures (cont)

```
let mySort edE lements =
myElements.sorted() {
  $0.cha rac ter s.count < $1.cha rac ter s.count
 }
```

Consult Docs for further information.

Enums

```
enum Name {
 case Label1 [= Value]
 case Label2 (Type)
}
```

Enumeratios are a way of labelling certain values to be used in the code.

```
enum Weekdays {
 case Mon, Tue, Wed, Thu, Fri, Sat, Sun
}
```

The way of referring to it is...

```
var myWeekDay = Weekda ys.Mon
```

Consult Docs for further information.

Structs

```
struct PointS truct {
  var X:Int,
  var Y:Int
}
```

Structs may be used to encapsulate data. They can even have a *initializer* to provide the struct with data.

```
struct PointS truct {
  var X:Int,
  var Y:Int
  init( x: Int, y: Int) {
    self.X = x
    self.Y = y
  }
 }
```

To declare a variable with this *struct* we would write...

```
let myPoint = PointS truct( x: 1, y: 2 )
```

Structures are similar to classes with a few *important* differences, being the most important one that their instances are always **passed by value**.

Consult Docs for further information.

Classes

Classes (cont)

Objects	Can be enumerations, variables or constants. Declaration can be preceded by: *lazy*, *static*, *private* or *public*.
Methods	The methods *init()* and *deinit()* are special methods which are called upon instantiation and destruction of the class. The method declaration can be preceded by one of the options: *private*, *public*, *mutating*, *class* or *static*
Protocols	*Protocols* do specify a certain blueprint a class conforms to. See below ↓

Protocols

```
protocol MyProtocol {
  var myVariable : Type { get set }
  func myMethod ( parameter: Type)->Type
}
```

A protocol defines a blueprint of *methods*, *properties* and other requirements that suit a particular task or piece of functionality. The protocol can then be adopted by a *class*, *struct*, or *enum* to provide an actual implementation of those requirements. Any type that satisfies the requirements of a protocol is said to conform to that protocol.

Consult: Docs for further information.

```
class MyClass : MyProtocol {
  var X:Int,
  init() {
  //code
  }
  func myMethod() {
  //code
  }
  deinit() {
  //code
  }
}
```

By **MerlinElMago**
(MerlinElMago)

cheatography.com/merlinelmago/

Not published yet.
Last updated 10th October, 2017.