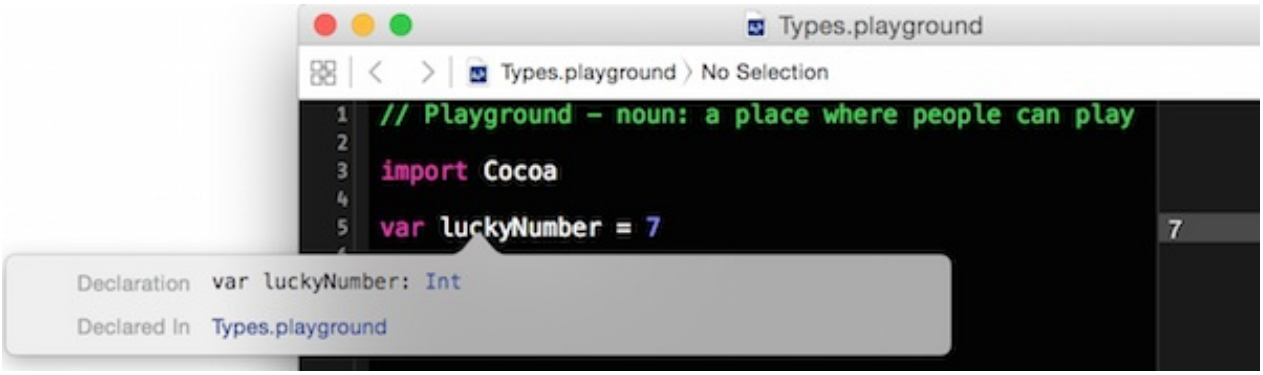
# 3. Types

## Introduction

All the values we've worked with so far have been integer numbers.
All variables and constants in Swift have a type. Think of the type as describing what kind of values a variable can have. The type for integer numbers is `Int`.

You can see the type of a variable in Xcode by option clicking on it's name (hold down option(⌥) and click on its name). A popup will appear were you can see the type of the variable.




Here you can see that our lucky number 7 is of type `Int`.

We often need variables of types that aren't `Int`. You already encountered a different type. The expresssions inside an if statement. These values have type `Bool` also known as `Boolean` named after mathematician George Boole who layed out the mathematical foundations of Logic.

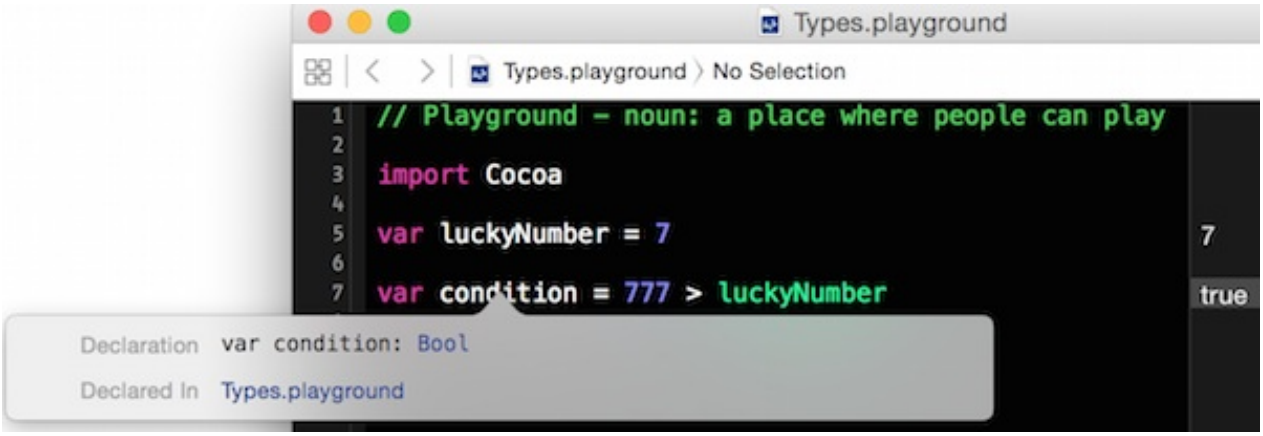
Comparision operators ( `<, <=, >, >=, ==, !=` ) produce values of type `Bool`. Boolean expressions can have only one of 2 types `true` or `false`

For example

```
var luckyNumber = 7

var condition = 777 > luckyNumber
```

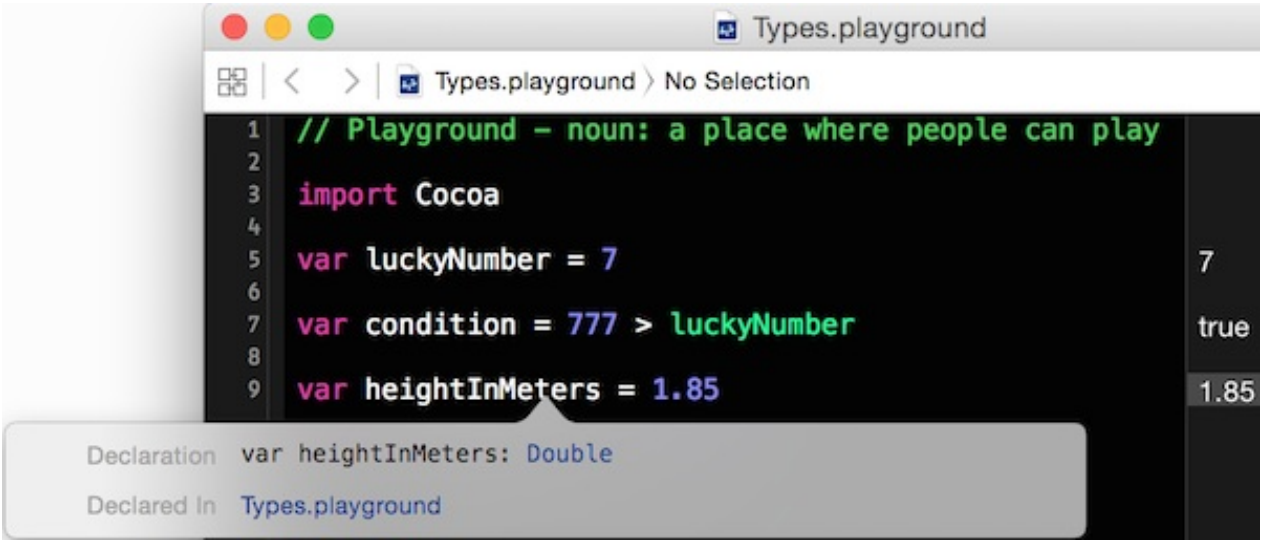
`condition` will be of type `Bool` and have a value of `true` as seen below:




## The Double type

What if we want to use numbers of the form ( `1.5, 1.7, 1.6,...` )? We can of course do that in Swift! If you declare a variable lets say `heightInMeters` with a value of `1.85` and check its type you'll see that its type is `Double`

```
var heightInMeters = 1.85
```




Variables of type `Double` hold fractional numbers and can be used for calculating fractional quantities. Any number of the form `X.Y` is a `Double` Examples: (35.67, 2.0, 0.5154, ...)

Doubles can be added, subtracted, multiplied and divided using the familiar operators ( `+, -, *, /` ).

There is no equivalent to the remainder( `%` ) operator for doubles.

```
var a = 3.5
var b = 1.25

print(a + b) // 4.75
```

```
print(a - b) // 2.25
print(a * b) // 4.375
print(a / b) // 2.8
```

[spoiler title='Why are fractional numbers called Double?(Optional)' collapse_link='true']
Numbers of type double have a limited precision. consider the following code

```
print(1.0 / 3.0) // 0.333333333333333
```

Mathematically speaking the number ( `1.0 / 3.0` ) should go on forever having an infinite number of 3s after the decimal `.` . Computers can't hold an infinite amount of information so they truncate the number at some point.

Representing decimal numbers in a computer is done via so called Floating Point Numbers. Represented by the type `Float` in Swift. The `Double` type is a kind of floating point number but compared to the `Float` type it can hold twice as many digits hence the name `Double` . [/spoiler]

## Declaring variables of a certain type

To explicitly declare a variable of a certain type you use the syntax:

**var** `variable` : `Type` = `...`

For example:

```
var integer:Int = 64
var boolean:Bool = false
var double:Double = 7.2
```

If you don't provide a type for a variable a type is automatically inferred for you using the value you provide

Examples:

```
// We don't declare a type for a it is implicitly Int
// because the value 7 is an Int
var a = 7

print(a / 2) // 3
```

```
// We don't declare a type for a it is implicitly Double
// because the value 7.0 is a Double
var a = 7.0
```

```
print(a / 2) // 3.5
```

If you explicilty declare a variable as having type `Double` then you can initialize it with an integer but the variable will hold a `Double`

```
var a:Double = 7 // We explicitly declare a type for a

print(a / 2) // 3.5
```

## Type Casting

Initializing a variable of type `Double` with an integer only works if you use a constant value. If you try initializing a variable of type `Double` with a variable of type `Int` then you'll get an error.

```
var a = 64
var b:Double = a // Error
```

To solve this problem we need to convert the value from `Int` to `Double`. Converting a value of some type to a different type is known as `type casting` or just `casting`.

To cast a variable to a certain type we use `TypeName(variableName)` or the `as` operator, `variableName as TypeName`. For example:

```
var a = 64
var b:Double = Double(a) // b = 64.0
var c:Double = a as Double // c = 64.0
```

Casting a `Double` to an `Int` discards all the digits after the decimal point. Note that this digits can't be recovered by casting the variable back to `Double`.

Example:

```
var number = 5.25
var integerNumber = Int(number) // 5
var doubleNumber = Double(integerNumber) // 5.0
```