1. First Steps

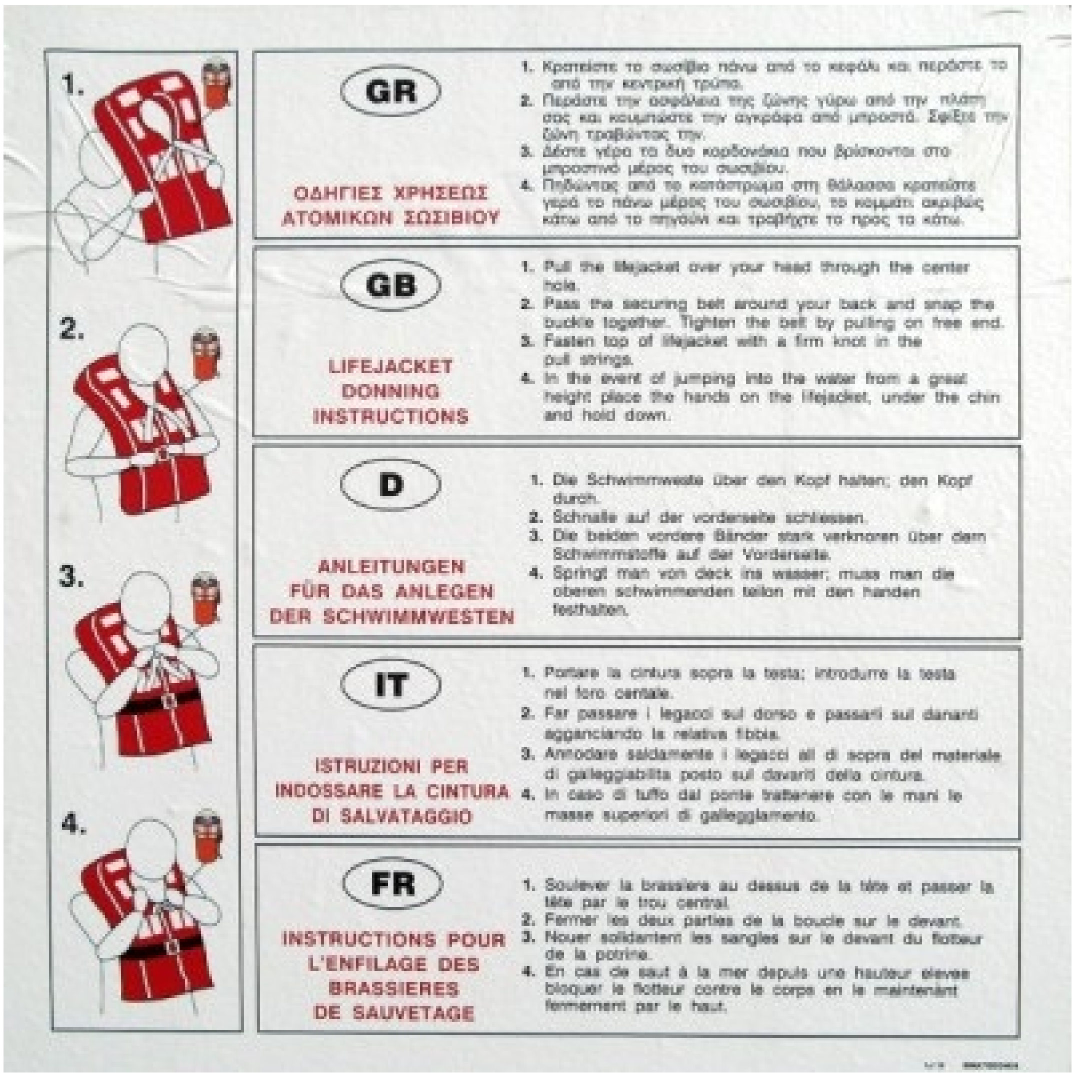
What is computer program?

A program is a list of instructions that are followed one after the other by a computer. You are most likely familliar with lists of instructions in everyday life. An example of a list of instructions would be a cooking recipe:

Fried eggs:

1. heat 2 tablespoons of butter in a non-stick frying pan.
2. break the eggs and slip into pan one at a time.
3. cook until whites are completely set and yolks begin to thicken.
4. carefully flip eggs. cook second side to desired doneness.
5. sprinkle with salt and pepper. serve immediately.

Another example is this list of instructions on how to put on a life jacket:



GR

ΟΔΗΓΙΕΣ ΧΡΗΣΕΩΣ ΑΤΟΜΙΚΩΝ ΣΩΣΙΒΙΟΥ

1. Κρατείστε το σωσίβιο πάνω από το κεφάλι και περάστε το από την κεντρική τρύπα.
2. Περάστε την ασφάλεια της ζώνης γύρω από την πλάτη σας και κουμπώστε την αγκράφα από μπροστά. Σφίξτε την ζώνη τραβώντας την.
3. Δέστε γερά τα δυο κορδονάκια που βρίσκονται στο μπροστινό μέρος του σωσιβίου.
4. Πηδώντας από το κατάστρωμα στη θάλασσα κρατείστε γερά το πάνω μέρος του σωσιβίου, το κομμάτι ακριβώς κάτω από το πηγούνι και τραβήξτε το προς τα κάτω.

GB

LIFEJACKET DONNING INSTRUCTIONS

1. Pull the lifejacket over your head through the center hole.
2. Pass the securing belt around your back and snap the buckle together. Tighten the belt by pulling on free end.
3. Fasten top of lifejacket with a firm knot in the pull strings.
4. In the event of jumping into the water from a great height place the hands on the lifejacket, under the chin and hold down.

D

ANLEITUNGEN FÜR DAS ANLEGEN DER SCHWIMMWESTEN

1. Die Schwimmweste über den Kopf halten; den Kopf durch.
2. Schnalle auf der vorderseite schliessen.
3. Die beiden vordere Bänder stark verknoren über dem Schwimmstoffe auf der Vorderseite.
4. Springt man von deck ins wasser; muss man die oberen schwimmenden teilon mit dan handen festhalten.

IT

ISTRUZIONI PER INDOSSARE LA CINTURA DI SALVATAGGIO

1. Portare la cintura sopra la testa; introdurre la testa nel foro centale.
2. Far passare i legacci sul dorso e passarli sul danant agganciando la relativa fibbia.
3. Annodare saldamente i legacci all di sopra del materiale di galleggiabilita posto sul davanti della cintura.
4. In caso di tuffo dal ponte trattenere con le mani le masse superiori di galleggiamento.

FR

INSTRUCTIONS POUR L'ENFILAGE DES BRASSIERES DE SAUVETAGE

1. Soulever la brassiere au dessus de la tête et passer la tête par le trou central.
2. Fermer les deux parties de la boucle sur le devant.
3. Nouer solidantent les sangles sur le devant du flotteur de la potrine.
4. En cas de saut à la mer depuis une hauteur élevee bloquer le flotteur contre le corps en le maintenant fermement par le haut.

The lists of instructions mentioned above are made to be executed by people. Computer programs are simillarly just lists of instructions but they are meant to be executed by computers. They are meant to be readable and understandable by humans but executing them would often be highly impracticable.

For example the program used for drawing a single screen in a modern game executes hundred of millions of mathematical operations like additions and multiplications. Executing such a list of instructions would take any person an embarasing amount of time, yet computers can happily do it 60 times per second.

Why do we need a programming language?

Lists of instructions like cooking recipes and putting on a life jacket are quite easy to understand for humans but they're incrediblly difficult to understand for a computer. Programming languages are designed as a way of giving a computer instructions that it can easily understand. That is because a programming language (like Swift) is much less ambigous than a language like english. Also it closely resembles the way in which a computer works.

In this book you'll learn the basics of programming using Swift. More importantly this will teach you about the kind of instructions that your computer understands and building programs for it to execute.

Whether you want to build an app, a game or a website the basic principles remain the same. You have to write a program for the computer to execute and writing such a program is done using a programming language.

Why Swift?

The Swift programming language was introduced in June 2014 by Apple, since then it has grown immensly in popularity. Swift is primarly used for developing apps and games for the iPhone and the Mac and provides an easier and more enjoyable way of doing that.

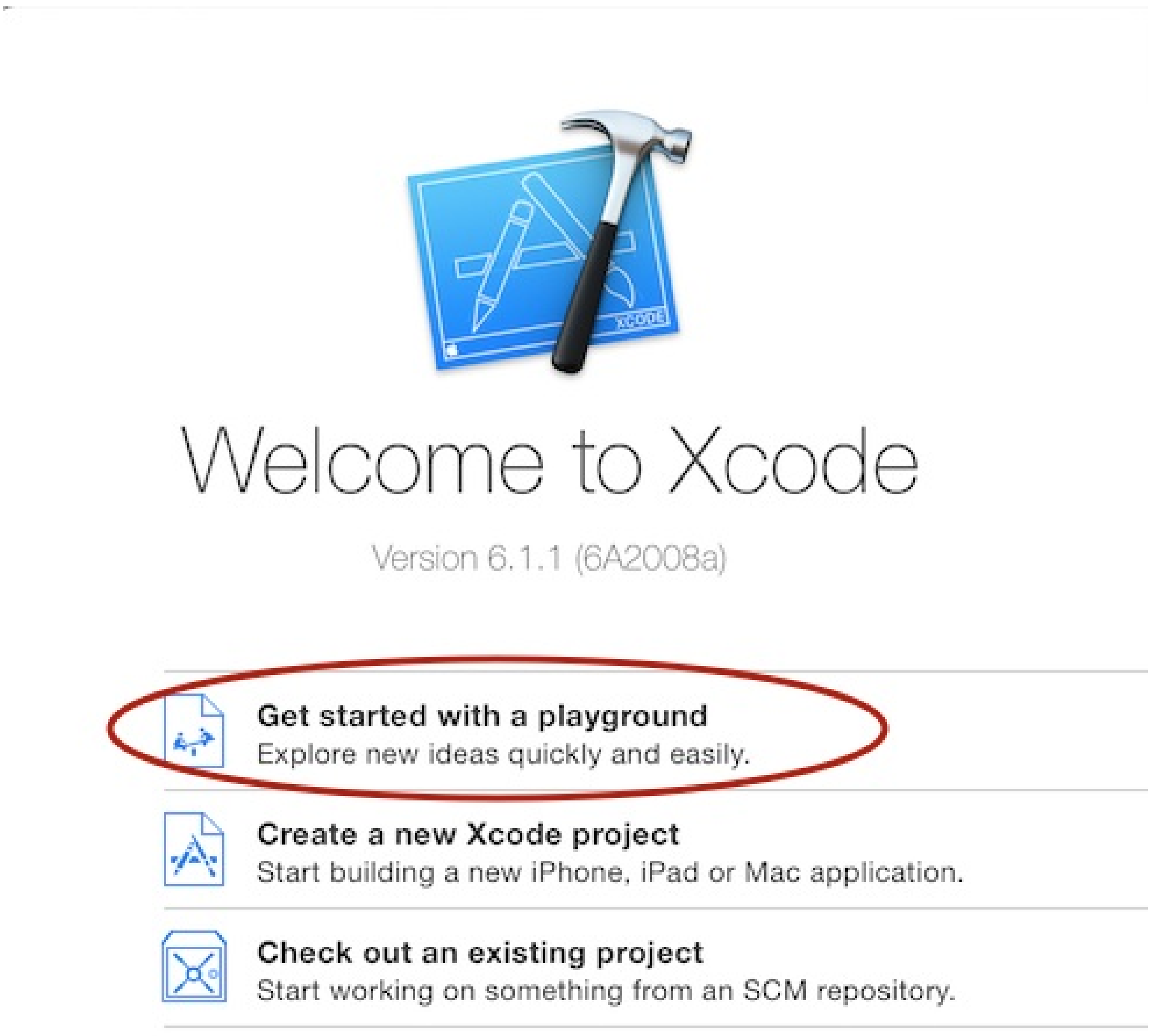
The great news is that Swift is also a great programming language for learning to code because of the **Playgrounds** feature described below.

Using Playgrounds

Playgrounds provide a fun and interactive way of writing code. Traditionally you would write a program and run it to see its results. With playgrounds you can see the results of your program imedially as you type it. This gives you a lot of opportunity for experimenting and makes learning faster.

If you have the companion app for this book than clicking on an exercise will open a playground for you to start coding.

If you don't have the companion app installed than you can open Xcode and create a new playground by clicking the "Get started with a playground" button. Select OS X As your Platform and choose a destination where you want to save the Playground.


Welcome to Xcode

Version 6.1.1 (6A2008a)

Get started with a playground
Explore new ideas quickly and easily.

Create a new Xcode project
Start building a new iPhone, iPad or Mac application.

Check out an existing project
Start working on something from an SCM repository.

Note: If you don't have Xcode installed, download the latest version from here

We'll start looking at basic concepts one by one now. We encourage you to experiment with the code we introduce by typing the statements into a playground and changing values around.

Variables and Constants

Use the `var` keyword to declare a variable and the `let` keyword to declare a constant. Variables and constants are named values. Variable can change their value over time and constants don't. To change the value of a variable you need to asign it a new one.

```
// declares a variable named a that has the value 1
var a = 1
// assigns the value 2 to the variable a
a = 2
// a has the value 2
// declares a constant named one with the value 1
let one = 1
one = 2 // this gives an error because we cannot change the value of a constant
```

> the text after `//` is called a comment. Comments are ignored by the computer when executing the program. They are usually used to explain parts of code

Naming Variables

Variables should usually be named using alphabetical characters. For example: `sum`, `number`, `grade`, `money`

If you want your variable's name to contain multiple words then you should start each word in the name with an uppercase letter except for the first one. For example you want a variable that holds the number of students in a class than you should name it `numberOfStudents` instead of `numberofstudents` because the first one is more readable.
This naming convention is called CamelCase.

It's recommanded to use descriptive names for variables. But don't overdo it, for example `numberOfStudents` is a reasonable name while `numberOfStudentsInTheCurrentClass` is too long. A good rule of thumb is to use at most 3 words for the name of a variable.

We could have used a way shorter name for the variable above, for example we could have called it `n`. The disadvantage with short variable names is that they're not expressive. If you read your code after 2 months you most likely won't remember what `n` means. But `numberOfStudents` is immediately obvious.

Generally its not a good idea to have variables that consist of a single letters but there are some exceptions.
When dealing with numbers that don't represent something it's ok to use single letter names.

Basic Operators

You can write arithmetic expressions using numbers, variables, operators and parentheses.

```
// The + operator returns the sum of two numbers
let sum = 1 + 2 // 3

// The - operator returns the difference of two numbers
let diff = 5 - sum // 5 - 3 = 2

// The * operator returns the product of two numbers
let mul = sum * diff // 3 * 2 = 6

// The / operator returns the numbers of times the  divisor(the number on
// the right side) divides into the dividend(the number on the left side)
// For example, when dividing 6 by 3, the quotient is 2, while 6 is called
// the dividend, and 3 the divisor.
// 13 divided by 5 would be 2 while the remainder would be 3.
let div = mul / diff // 6 / 2 = 3
```

```
// The remainder(modulo) operator returns the remainder of the division
let mod = 7 % 3 // 1 because 7/3 = 2 and remainder 1 (2 * 3 + 1 = 7)

// You can use parentheses to group operations
(1 + 1) * (5 - 2)

// Multiplication, division and remainder have higher precedence than
// addition and subtraction.
// For example: 5 + 2 * 3 = 5 + 6 = 11
```

Integer Division

Addition, subtraction and multiplication behave pretty much as you expect. The tricky operations are division and remainder.
Take for example `5 / 2`. Normally you'd expect the result to be `2.5`. In Swift dividing two integers also produces an integer this is acomplished by discarding the part of the number after the decimal point. So `5 / 2 = 2`.
The remainder operator or modulo operator (%) is used to get the remainder of an integer division. `5 % 2 = 1`
For `5 / 2`:
`quotient = 5 / 2 = 2`
`remainder = 5 % 2 = 1`
`quotient * 2 + remainder = 5`

Generally speaking for two integers `a` and `b` this equations always hold
`quotient = a / b`
`remainder = a % b`
`b * quotient + remainder = a`

NOTICE: `remainder = a - b * quotient`
This implies that `remainder = a - b * (a / b)` and
`a % b = a - b * (a / b)`
You can view `a % b` as a shorthand way of computing `a - b * (a / b)`

NOTICE: if `a % b = 0` then `b` divides `a`, that is `a` is a multiple of `b`.
Example:
`15 / 5 = 3`
`15 % 5 = 0` ->
`15 = 5 * 3`

Order of statemets and more Playgrounds

The order of statements in a program matters. Like lists of instructions programs are executed from top to bottom.

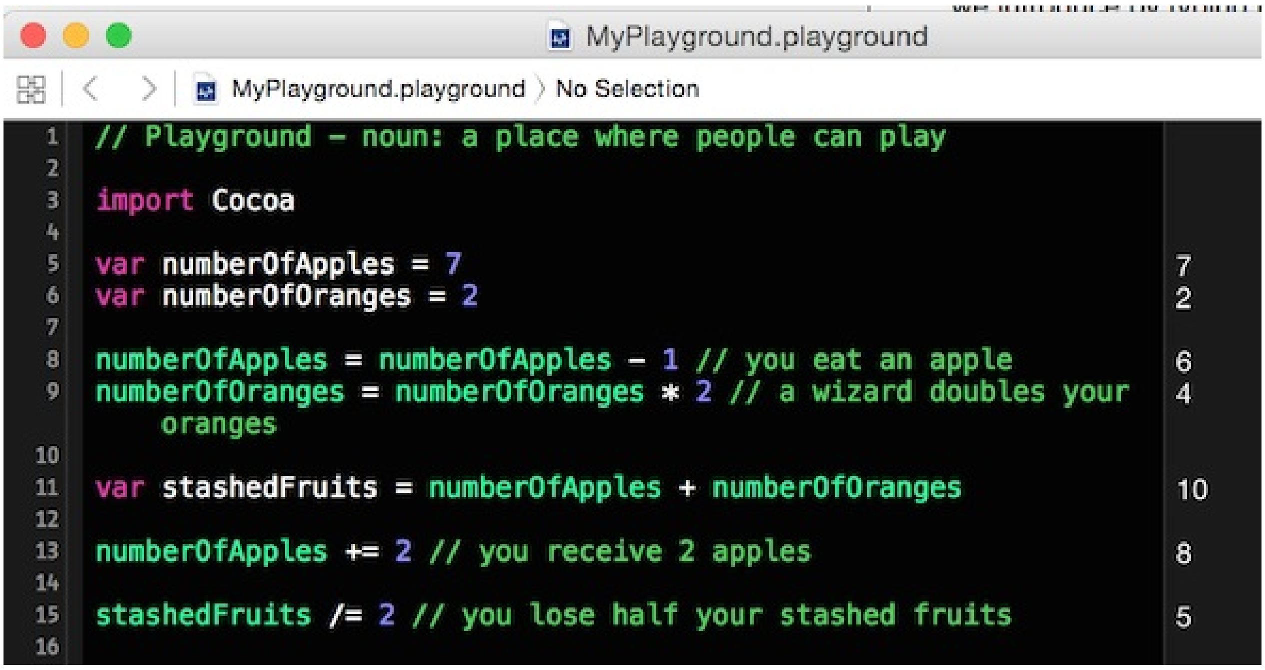
```
var numberOfApples = 7 // you have 7 apples
var numberOfOranges = 2 // you have 2 orages

// you eat an apple (numberOfApples = 6)
```

```
numberOfApples = numberOfApples - 1

// a wizard doubles your oranges (numberOfOranges = 4)
numberOfOranges = numberOfOranges * 2

var stashedFruits = numberOfApples + numberOfOranges // 10 (6 + 4)

// you receive 2 apples (numberOfApples = 8). stashedFruits remains unchanged!
numberOfApples += 2

stashedFruits /= 2 // you lose half your stashed fruits 5 (10 / 2)
```

In the program above the variable `stashedFruits` gets a value only after all the previous statements are run. Also notice that the value assigned to a variable is computed at the time of assignment. Changing `numberOfApples` after declaring `stashedFruits` will not have any effect on the value of `stashedFruits`.

Looking at the code above in a playground will give you an idea of why playgrounds are incredibly helpful for visualizing how code behaves.

As you can see each line of code in a playground has the value of the expression on that line printed in the right area of the screen.

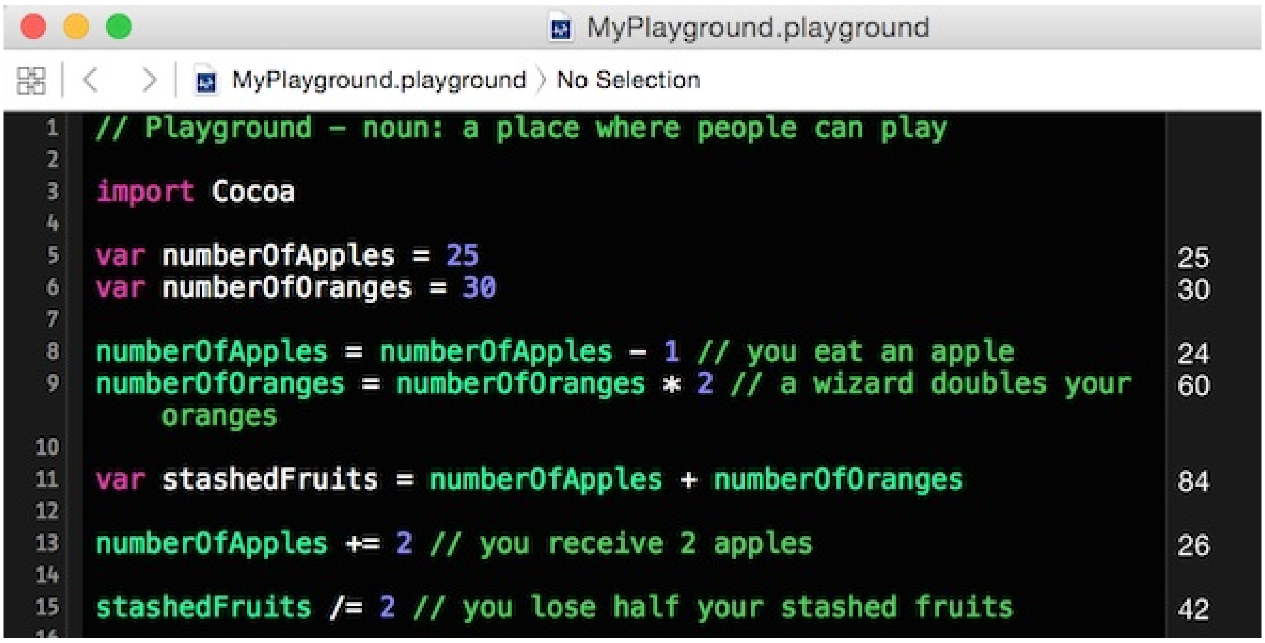


```
// Playground - noun: a place where people can play

import Cocoa

var numberOfApples = 7                                              7
var numberOfOranges = 2                                             2

numberOfApples = numberOfApples - 1 // you eat an apple             6
numberOfOranges = numberOfOranges * 2 // a wizard doubles your      4
    oranges

var stashedFruits = numberOfApples + numberOfOranges                10

numberOfApples += 2 // you receive 2 apples                         8

stashedFruits /= 2 // you lose half your stashed fruits             5
```

Now, the cool thing is that if we modify any values. All lines are imediatelly updated to reflect our changes. For example we modify:

```
var numberOfApples = 25
var numberOfOranges = 30
```

and everything is recalculated and displayed immediately.


MyPlayground.playground

MyPlayground.playground 〉 No Selection

```
// Playground - noun: a place where people can play

import Cocoa

var numberOfApples = 25                                                     25
var numberOfOranges = 30                                                    30

numberOfApples = numberOfApples - 1 // you eat an apple                     24
numberOfOranges = numberOfOranges * 2 // a wizard doubles your              60
    oranges

var stashedFruits = numberOfApples + numberOfOranges                        84

numberOfApples += 2 // you receive 2 apples                                 26

stashedFruits /= 2 // you lose half your stashed fruits                     42
```

Try playing around!

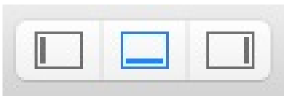
Print Statement

After making some computations you will want to show your results somehow. The simplest way to do it is with `print()` statement.

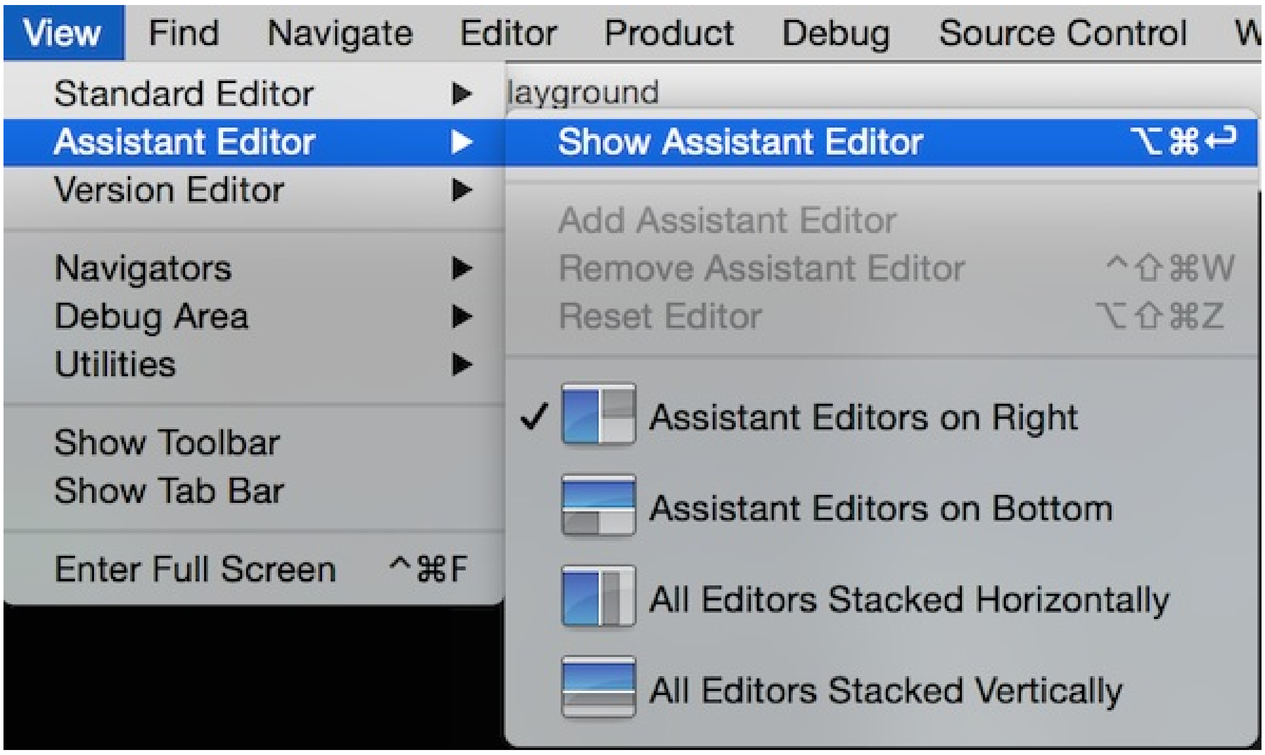
```
// will print Hello Swift! in the console
// you can print any text between quotes
print("Hello Swift!")

print(1 + 2) // will print 3

var ten = 10

print(ten) // will print 10
```

To see the console output in Playground make sure to show the `Debug Area`.

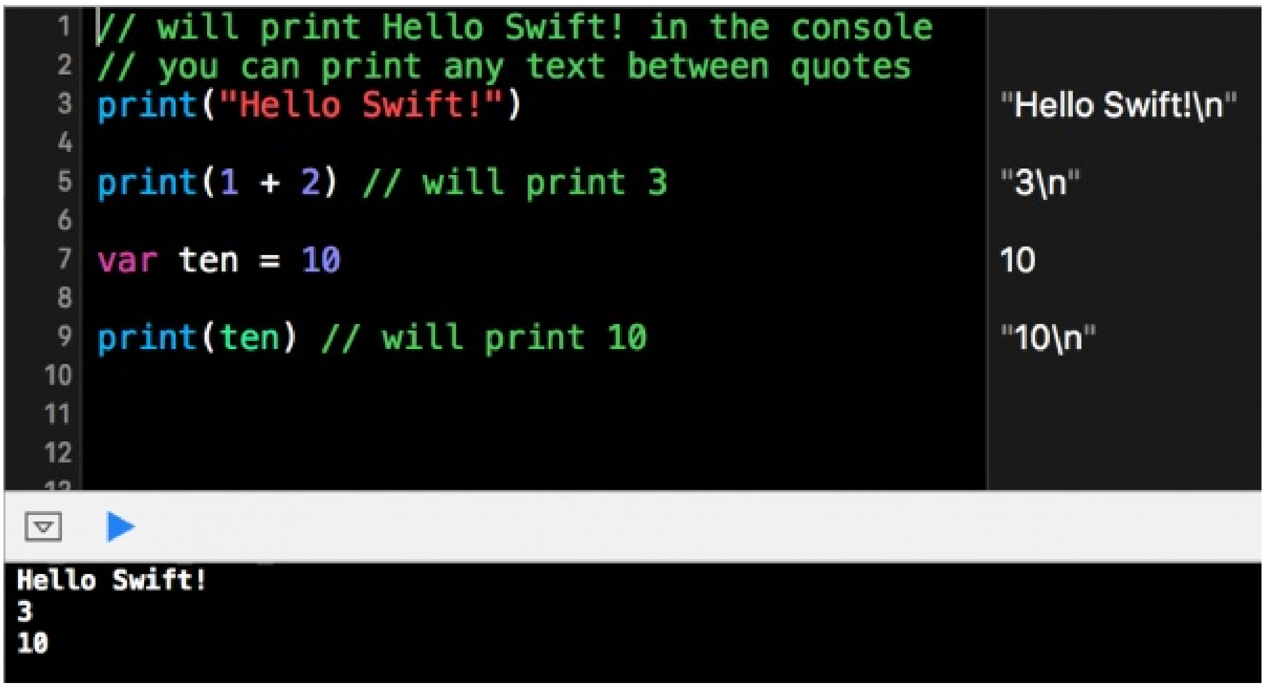
You can do that by pressing the middle button from the top right corner of the Playground.



Or from the menu:



This is how the code from the example would look with the `Debug Area` visible:



```
// will print Hello Swift! in the console
// you can print any text between quotes
print("Hello Swift!")                          "Hello Swift!\n"

print(1 + 2) // will print 3                   "3\n"

var ten = 10                                   10

print(ten) // will print 10                    "10\n"
```

```
Hello Swift!
3
10
```