

# Programming Refresher

Helen Miles ([hem23@aber.ac.uk](mailto:hem23@aber.ac.uk))

## Session Plan

The session is planned for approximately 2 hours but you should work at the pace you are most comfortable with. The session will be structured roughly as follows:

<b>15 mins</b>	Introduction & Questionnaire 1
----------------	--------------------------------

<b>1.5 hours</b>	Playgrounds
------------------	-------------

<b>5 mins</b>	Questionnaire 2
---------------	-----------------

## Questionnaires

We want to know if this session is helpful, please take a couple of minutes to fill these in:

Q1: complete before the session – <https://goo.gl/forms/wXwLe9JOUnV0Hjc12>

Q2: complete after the session – <https://goo.gl/forms/KL5oYVj77qTJwcM93>


Thank you! 😊

---

## List of Playgrounds

Filename	Purpose
P0-HelloWorld	Welcome to Playgrounds
P1-Types	Declaring values
P2-Strings	A closer look at Characters and Strings
P3-ControlStructures	Controlling the flow of a program using loops
P4-EnumSwitch	Declaring enums and using switch statements
P5-Functions	Making programs more useful with functions
P6-Collections	Looking at array, sets and dictionaries

## Some Notes on Playgrounds

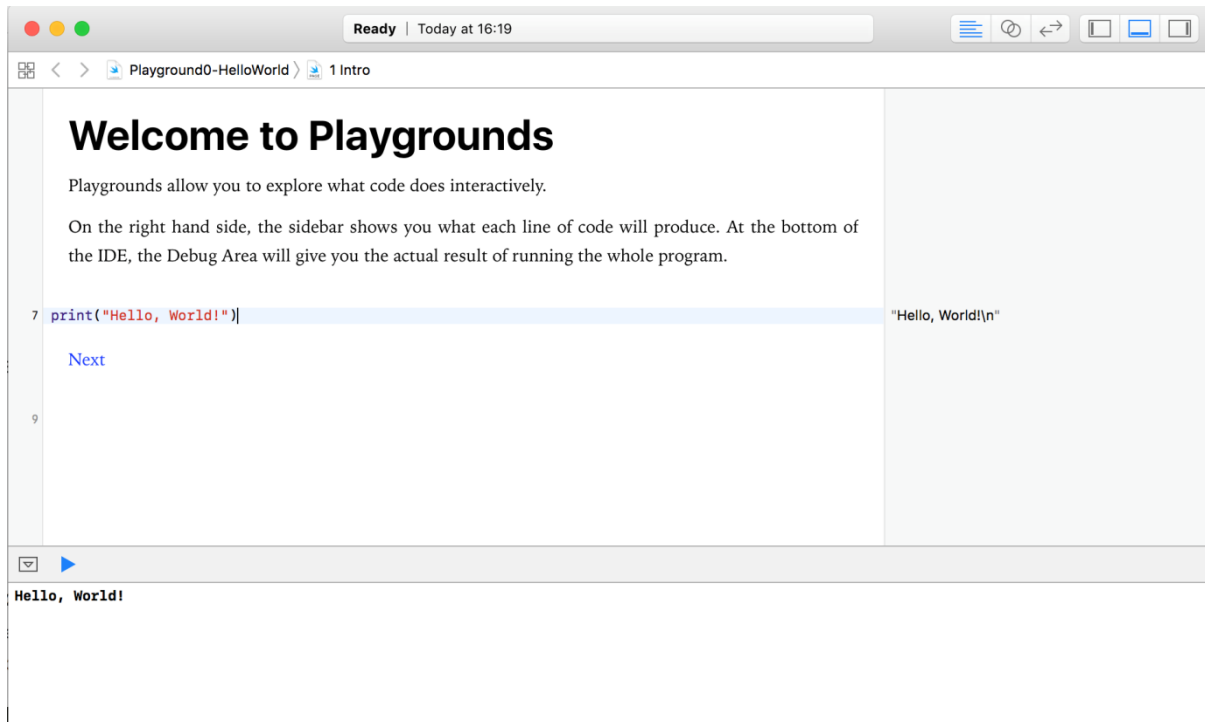
- Playgrounds are set up as either iOS, MacOS or tvOS
- You can only use the appropriate libraries
- Playground settings can be changed in Xcode (View > Utilities > Show File Inspector) or by clicking the following button in the top right  corner:
- Documentation is in Markup and can be shown as raw text or rendered (Editor > Show Raw Markup)
- Playgrounds are still a bit buggy, sometimes they will show an error that is no longer there
- If they are not showing any output, often because of a problem shown in console - turn on Assistant Editor

## Playground 0: Welcome to Playgrounds

Open P0-HelloWorld.playground in Xcode.

### Exercise 1: Explore the GUI

When you open the file you should see something like this:



Playgrounds allow you to have a mix of code and markup. In this Playground the text at the top is all markup and the only line of code is the print statement `print("Hello,World!")`.

**Step 1:** Read the introduction – find the sidebar and console in the GUI.

**Step 2:** Change the text of the print statement from "Hello, World" to "Hello, Aber" and see what happens. Playgrounds should automatically re-run the code when changes are made, but if nothing changes within a few seconds you can prompt a re-run by clicking the play button in the lower left corner of the code area.

## Playground 1: Declaring Values

Open P1-Types.playground in Xcode.

### Exercise 1: Simple types

We are going to start by declaring some variables. You can see Swift's declaration structure in the variable `str`: `var str = "Hello, playground"`

**Step 1:** Use the same structure to declare the values suggested in the space just below the comment.

**Step 2:** Take a look at the types and output of your new variables.

### Exercise 2: Variables and constants

Variables hold values which can be changed at runtime (mutable), while constants hold values which are declared once and cannot be changed at runtime (immutable).

**Step 1:** You have a variable called `myStr`; change the declaration to make it a constant and see what happens to the next line of code.

**Step 2:** Use Swift's fix-it or manually change `myStr` back to a variable to fix the error.

### Exercise 3: Specific types

Swift allows you to declare values with and without stating a type (e.g. integer, string, float). Some languages require you to give the type while others will try to work it out based on the contents; both approaches have positives and negatives.

**Step 1:** Look at the examples; there's a mix of variables/constants and types.

**Step 2:** Declare a constant `Int myInt` without giving it a value. Why is that ok?

**Step 3:** Declare another constant `Int myOtherInt` and giving it the value of the constant you just created. What happens this time?

**Step 4:** You can fix the error by assigning a value to `myInt` before declaring `myOtherInt`.

### Exercise 4: Expressions

Expressions are the same as in most languages, see a list in the Swift Quick Reference provided. Type safety means that you sometimes have to specify a type in the expression.

**Step 1:** Declare a new variable called `newValue` and make it equal to `value + realValue`.

**Step 2:** You cannot add different types – you will need to **convert** the float to a Double. Add `Double()` around `value` to get it to work.

**Step 3:** Print `newValue` to the console.

**Step 4:** Display the contents of `newValue` in the sidebar.

---

*Warning: you need to be consistent with spaces in expression evaluation in Swift –*

<code>var age = 22 + 7</code>	works fine
<code>var age=22+7</code>	works fine
<code>var age = 22+ 7</code>	gives an error
<code>var age= 22 + 7</code>	gives an error

## Playground 2: Strings and Characters

Open P2-Strings.playground in Xcode.

### Exercise 1: Defining characters and string literals

You can create strings by providing a set of characters and/or by including other variables using string interpolation.

**Step 1:** Try using string interpolation and arithmetic with `value` to create a constant `answer` which will contain the string `"4.6 + 4.6 = 9.2"`

**Step 2:** Print `answer` to the console.

### Exercise 2: String operations

Swift allows programmers to do operations on Strings, try some of them on `politeGreeting` and look at the results.

**Step 1:** Try using `isEmpty`.

**Step 2:** Try using `hasPrefix` with the prefix `"friend"`.

**Step 3:** Try using `hasSuffix` with `"end"`.

**Step 4:** Try using `count` – how many characters are in the string?

**Step 5:** Add a new line after the declaration of `politeGreeting`:  
`politeGreeting = ""` – what are the results now?

### Exercise 3: String comparison

Comparing Strings is very useful and there are many ways to do this in Swift. Try comparing the following strings using common operators (use the Swift Quick Reference to help you):

**Step 1:** Is `"aardvark"` equal to `"penguin"`?

**Step 2:** Is `"aardvark"` equal to `"aard"` plus `"vark"`?

**Step 3:** Is `"aardvark"` less than `"penguin"`?

**Step 4:** Is `"quail"` less than `"penguin"`?

**Step 5:** Is `"penguin"` less than or equal to `"penguin"`?

**Step 6:** Think about what these results show. Try changing the strings and characters to explore this.

## Playground 3: Control Structures

Open P3-ControlStructures.playground in Xcode.

### Exercise 1: If statements

Construct an if statement to control your desk fan based on the current room temperature.

**Step 1:** Look at Swift's if statement structure in the example.

**Step 2:** Compare the provided variables (`temperature`, `comfyTemp` and `coldTemp`) using operators in the Swift Quick Reference.

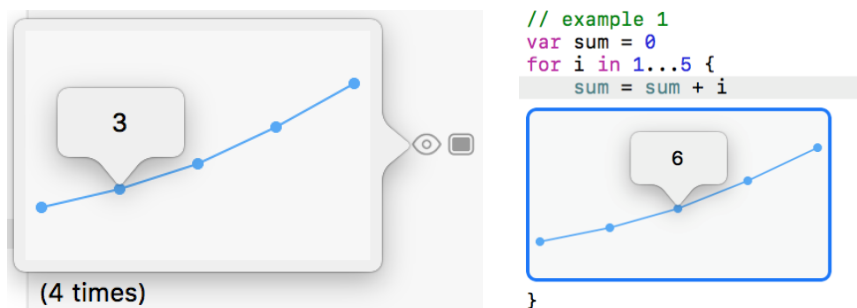
**Step 3:** Add print statements to give the right output for each condition.

### Exercise 2: For loops

Explore how Playgrounds will allow you to monitor loop variable progress.

**Step 1:** Find the Quick Look and Show Result buttons in the sidebar 

**Step 2:** Try each of the buttons with the different loops:



**Step 3:** Modify the loops to see how the progress changes.

### Exercise 3: While loops

**Step 1:** Explore the examples of while and repeat while loops provided.

**Step 2:** Try changing the values of `count` from 10 to 0. What happens?

### Exercise 4: Switch statements

Using the example switch statement provided, and the constant `petName`, create a switch statement to print relevant animal noises.

**Step 1:** Look at the example switch `someCharacter`.

**Step 2:** Write a new switch for `PetName`.

**Step 3:** Add cases for each of the pets specified in the list, and print the given sound.

### Exercise 5: Buzz

Write a loop that goes from 1 to 100 and prints "buzz" if the number is divisible by 3 and the number if it is not.

**Step 1:** Create a loop which will go from 1 to 100

**Step 2:** For every time the loop goes around, check whether the current number is divisible by 3. Hint: If `number % 3 == 0` then it is divisible by 3...

**Step 3:** If it is, print the word "buzz", otherwise just print the current number.

## Playground 4: Enums and Switch Statements

Open P4-EnumSwitch.playground in Xcode.

### Exercise 1: Enum syntax

Using the examples provided, write a new enum to hold the days of the week.

**Step 1:** Create an enum.

**Step 2:** Create a case for each of the 7 days.

### Exercise 2: Switch enums

Write a switch statement to work with the `Planet` enum.

**Step 1:** Look at the `Planet` enum and the example switch statement provided.

**Step 2:** Create a switch which will print the specified messages for Earth and Mars.

**Step 3:** Add a default case to catch the remaining cases.

### Exercise 3: Associated values

Create an enum to store library information.

**Step 1:** Create a new enum called `Library`.

**Step 2:** Add cases for the four items the library holds.

**Step 3:** Choose types to hold the associated values requested.

**Step 4:** Create two new `Library` variables to test your enum:

- a. A book: “The Martian” by Andy Weir, published 2011.
- b. A film: “The Martian”, released in 2015.

### Exercise 4: Raw values

Write a new version of your days of the week enum to give the days a raw numbered value, starting with Saturday as day number 1:

**Step 1:** Create a new enum with the appropriate type.

**Step 2:** Create cases for each of the days of the week.

**Step 3:** Add raw values to the cases, making sure Saturday is day 1.

**Step 4:** Try out your enum by getting the raw value of Tuesday – what is the result?

### Exercise 5: Week planner

Create a switch that will use your days of the week enum to return the activities of the day: during weekdays you are at work; on Tuesdays you also go to photography club; on Wednesday afternoons you leave early to go swimming; on weekends you relax.

**Step 1:** Create a switch to differentiate between your cases for days of the week.

**Step 2:** Create a ‘what to do’ string which will be updated depending on the day’s activities.

**Step 3:** Print the string after the switch statement.

## Playground 5: Functions

Open P5-Functions.playground in Xcode.

### Exercise 1: Declaring basic functions

Try creating your own functions that uses a loop to count from 10 down to 1, and return the String “10, 9, 8, 7, 6, 5, 4, 3, 2, 1, Done!” when it’s finished.

- Step 1:** Create a countdown function that will return a String.
- Step 2:** Create two variables inside the function: one to store the number counting down and one to store the string to return. Initialize them appropriately.
- Step 3:** Add a return statement to return the final string.
- Step 4:** Create a loop which will decrement the number to 0 and update the return string each time.
- Step 5:** Add an if statement that will update the string to return once the loop has completed.

### Exercise 2: Arguments, many arguments

Write a new version of `myFunction` that will deliver a personalized greeting message.

- Step 1:** Create a new function with the appropriate argument and return types.
- Step 2:** Use arguments and Strings to return a greeting, e.g. “Hello, Anna!”
- Step 3:** Test your function.

### Exercise 3: Temperature conversions

Create a function which will convert temperatures given in Fahrenheit to Celsius using the formula provided.

- Step 1:** Create a new function with the appropriate argument and return types.
- Step 2:** Examine the formula provided and implement it within your function.
- Step 3:** Test your function with the values provided – do you get the correct results?

### Exercise 4: Multiple return values

Read through this section on how to implement functions with multiple return types.

- Step 1:** Try changing the values requested to see the results of the function change.
- Step 2:** Experiment with the values of the function.

### Exercise 5: Weather reporting

Using the previous exercises we will create a function to give short weather reports.

- Step 1:** Read the requirements carefully; we will need several variables of different types.
- Step 2:** Create a new function with the appropriate argument and return types.
- Step 3:** Using a control structure based on the temperature, return the appropriate weather report.
- Step 4:** Use another control structure to determine whether there is a weather warning active or not.
- Step 5:** Test your function – do you get the correct results?



## Playground 6: Collections

Open P6-Collections.playground in Xcode.

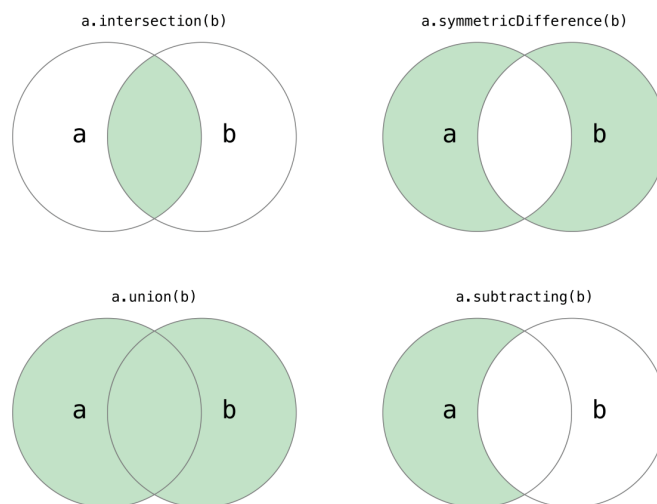
### Exercise 1: Arrays

Create a shopping list, add items to it and print the list out.

- Step 1:** Create an array called `shoppingList` with two initial elements: Eggs and Milk.
- Step 2:** Add Baking powder to the list using `append()`.
- Step 3:** Add Chocolate spread, Cheese, and Butter using the assignment operator.
- Step 4:** Add Maple syrup to the beginning of the list using `insert()`.
- Step 5:** Write an if statement to check whether the list is empty.
- Step 6:** If not, use a while loop to print the items on the list.

### Exercise 2: Sets

Take a look at the set operation diagram below; try each operation with the sets provided.



- Step 1:** Find the results of the following:
  - a. `oddDigits union evenDigits`
  - b. `oddDigits intersections evenDigits`
  - c. `oddDigits subtracting singleDigitPrimeNumbers`
  - d. `oddDigits symmetric difference singleDigitPrimeNumbers`
- Step 2:** Look at the results. Try using `sorted` if they are hard to read.
- Step 3:** Take a look at the three further operations – what do you think they do? Play around with them to find out more.

### Exercise 3: Dictionaries

Create a dictionary of pets containing the pet names and species: Dave the goldfish, Chaz the dog, and Idris the guinea pig.

- Step 1:** Set up a new dictionary containing the appropriate values stored as Strings.
- Step 2:** Add a new guinea pig called Murphy to the list.
- Step 3:** Print a list of the pets in the following format: “name is a animal”.
- Step 4:** Try also printing the dictionary in the following formats:
  - a. “name is a pet”
  - b. “We have a animal”

## Playground 7: Try Out Your New Swift Skills!

Start a new Playground in Xcode.

### Exercise: Prototype a Wildlife App

A company wants to make a mobile app to track the wildlife of Wales. Users can report what kinds of animals they've spotted around Wales, including animals on land, and in the seas and skies. Develop some prototype functionality for the system running the app:

- Record animals spotted by users
- Store relevant data about where/when they were spotted
- Animal identifier – a system to help the user identify what they've seen
- List information, e.g.:
  - What animals are found in Wales?
  - How many of each type?
  - Where is the best place to go to see a particular animal?

**Note:** the prototype doesn't need to take user input or store data for now, you can use variable to demonstrate how your functions will work.



## Swift Quick Reference

Binary Operators	
+	Add
-	Subtract
*	Multiply
/	Divide
%	Remainder

Unary Operators	
-	Minus, e.g. $x = -y$
--	Decrement
++	Increment

Binary Shorthand	
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Remainder and assign

Boolean Tests	
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal
!=	Not equal
&&	Logical AND
	Logical OR