

Tutorial Pack 8

(90 minutes)

(To be completed during LW8 tutorial)

LEARNING OBJECTIVES

- To introduce students to the Python programming language

LEARNING OUTCOMES

- By the end of this tutorial students will have.
 - Setup a Python IDE (repl.it, Google Collab or Jupyter)
 - Built up a basic understanding of key Python programming concepts
 - Practiced creating and modifying variables
 - Familiarised themselves with key Python data types and structures
 - Developed experience using arithmetic, logical and membership operators
 - Practiced using control statements
 - Reviewed the fundamentals of program and flow control

RESOURCES AND TOOLS REQUIRED

- Google Apps account to use repl.it and Google Collab
- Jupyter can be installed through Anaconda on AppsAnywhere

IMPORTANT:

This pack is designed for you to go at your own speed. At the end there is a series of practice questions – you should attempt to answer **all questions**.

Any questions you do not complete today should be completed before your next tutorial.

You may find this tutorial pack and the exercises useful preparation for the coursework.

1. Python

1.1. Introduction

Python is a powerful, open source, dynamic object-orientated, programming language. **Python** has several key features that make it attractive, particularly for carrying out data analysis.

- The Python syntax is simple and concise (no semicolons or curly braces).
- Thousands of external modules for analysis and modelling.
- Python has a large and active data science community.
- Excellent compatibility with different data types and formats (JSON, XML, XLS, SQL, CSV etc)

One of the best ways to learn any programming language is by **practicing**. Before we get started however, let me introduce some important terminology.

1.1. Python Terminology

Term	Definition
<i>Python Language</i>	Special sets of predefined words and statements that we use to develop our Python programs
<i>Python Interpreter</i>	A program that understands Python syntax. Its job is to read the Python syntax that you write and perform the relevant tasks
<i>Python Package</i>	A collection of Python programming code that performs a task or set of tasks relating to a similar topic.
<i>Python Notebook</i>	A web-based editor for Python programs
<i>Source Code</i>	The written code that makes up a (Python) program

1.2. Setting up the development environment

Python programs are typically developed using an IDE (integrated development environment). This is a special piece of software that provides specific functionality, such as the ability to test code and correct common syntax errors, that many developers find useful. In the case of Python there are multiple possible IDEs you can use, including some that are commercial and others that are available for free.

For this tutorial we are going to use Google Collab, a web-based IDE that uses Python Notebooks.

<https://colab.research.google.com/>

The tutor will demonstrate:

- I. How to create a new notebook
- II. Rename a notebook.
- III. Adding text and code cells.
- IV. Running code inside a cell.

1.3.Creating and using variables

An important feature of a programming language is to enable us to retrieve, manipulate and store data. Typically, this is achieved using **variables**. Whenever we create, a new variable Python will try to determine what its datatype should be based on its contents.

The choice of datatype will affect future use of the variable.

Table 1 Available data types in Python

Data type	Purpose
str	For storing alphanumeric text
int	For integers (whole numbers)
bool	For storing true or false values
float	For floating point numbers
dict	A dictionary of key and value pairs
list	A collection of items (like an array)

Variables in Python are created by using the assignment (equals =) operator. Since Python is **dynamically typed**, there is no need to specify the data type used for a variable since it will be inferred by Python based on the content of the variable itself.

Do you remember the calculation for the area of a circle? πr^2

Example 1. Creating a variable called pi and assigning the value 3.14 to it

```
pi = 3.14
```

Example 2. Creating a variable called r and setting it to 5

```
r = 5
```

Example 3. Calculating the area of a circle

```
pi * (r**2)
```

In order to print our result to the console window we need to use the **print()** function with the variable we would like to print passed as a parameter

In this case I have assigned the result to a new variable called result

Example 4. Assigning the result of a mathematical calculation to a variable

```
result = pi * (r**2)
```

Next we add the print() function and specify the variable we would like to print out

Example 5: Printing out the result of our previous calculation

```
print(result)
```

So far, we have not explicitly set the data type for each of the variables created previously. Recall this was due to Python automatically determining what a variable's type should be based on its contents. To check which type a variable has been assigned we can use the **type()** function.

Example 6: Printing out the data type used by a variable called `result`

```
print(type(result))
```

1.4.Strings

In Python the string data type is used to store text, symbols, and numbers. For example, a string could be used to store a twitter comment. Strings in Python are like the VARCHAR and TEXT datatypes used in many Database Management Systems (DBMS).

Example 7: Creating a string

```
comment = "The rain in spain falls mainly on the plane"
```

A string is just an array of characters. We can access them individually using array notation. To find the first character in the string we access the 0th element.

```
print(comment[0])
```

Example 8: We can find the 5th character

```
print(comment[4])
```

Example 9: Extracting the last character by subtraction (or the offset from the end)

```
print(comment[-1])
```

Example 10: Selecting a substring within a string (this selects characters between position 0 and 11)

```
print(comment[0:11])
```

Example 11: Combining strings using the addition operator

```
little = comment[12] + comment[13] + comment[5] + comment[24]
print(little)
```

Example 12: Finding the length of a string using the **len()** function

```
print(len(little))
```

All variables in Python that are instances of the data type string inherit all the methods of the string class. A list of common string methods can be found at the following [link](#). To apply a method on an instance of a variable we use the dot notion.

For instance we can use the `string.lower()` method to convert a string into lowercase

```
"Hello World".lower()
```

Example 13: Converting a string to uppercase using the `string.upper()` method

```
"Hello World".upper()
```

1.5.Lists

Lists in Python represent collections of other items or indeed other lists. Lists can be created using a series of comma-separated values (items) enclosed in square brackets.

Example 14: Creating an empty list

```
items = []
```

Example 15: Creating a list to represent a shopping list

```
items = ["eggs", "bread", "milk"]
```

Example 16: Determining how many items are contained in the list

```
print(len(items))
```

Example 17: Extracting the first item from a list

```
print (items[0])
```

Example 18: Extracting the last item from a list

```
print(items[-1])
```

To add items to a list we use the **append** method.

Example 19: Adding Coffee to the shopping list

```
items.append("coffee")
```

Example 20: Determining the size of the list

```
print(len(items))
```

Items can also be removed from a list by using the **remove** method that is associated with a specific list. NB To remove an item from a list it must already exist in the list – otherwise an error will be raised.

```
items.remove("tea")  
print(items)
```

The append method will always place a new item at the end of the list. To add an item at a specific location we need to use the **insert** method. The insert method accepts two parameters, the position where the item should be inserted and the item itself.

The following code adds the “tea” item back into the list at the 0th location, i.e., at the front.

Example 21: Adding a new item to the list using the insert method

```
items.insert(0,"tea")
```

We can also create lists by splitting apart other variables. The example below splits a string into its individual words.

```
msg = "This is an important message"  
print(msg.split())
```

1.6.Dictionaries

Dictionaries in Python represent a mapping between distinct keys and values. A good analogy is an address book, which maps names (keys) to addresses (values).

Example 22: Creating an empty dictionary

```
address_book = {}
```

Example 23: Creating a dictionary to store the addresses of two people

```
address_book = {"person1": "address1", "person2": "address2"}
```

Example 24: Accessing the value of a specific key

```
print(address_book["person1"])
```

Example 25: Adding a new key and value to a dictionary

```
address_book["person3"] = "address3"
```

Example 26: Viewing all keys in a dictionary as a list

```
address_book.keys()
```

1.7. Comparison operators

Comparison operations can be used to compare two values or variables. In Python there are 6 possible comparison operations, and the result of a comparison is always either True or False.

Syntax	Purpose	Example
==	Equal	a == b
!=	Not Equal	a != b
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b

1.8. Arithmetic operators

Whenever we want to perform mathematical manipulation of a variable, we use arithmetic operators. In the previous example we used the multiply (*) operator and the power (**) operator. Other operators are shown in the table below.

Syntax	Purpose	Example
+	Addition	$y = a + b$
-	Subtraction	$y = a - b$
*	Multiplication	$y = a * b$
/	Division	$y = a / b$
%	Modulus (remainder)	$y = a \% b$
**	Power	$y = a ** b$

The screenshot below shows how different arithmetic operations can be put together to create a program that calculates a person's body mass index based on their height and weight.

```

"""
Body Mass Index (or BMI)
"""

height_in_m = 1.65
weight_in_kg = 52

bmi = weight_in_kg / height_in_m ** 2
print(bmi)

```

1.9.Program Flow and Control

Control statements are used so that you can regulate how different parts of your run based upon the condition of one or more variables.

The simplest example of a control statement is the **IF statement**.

```

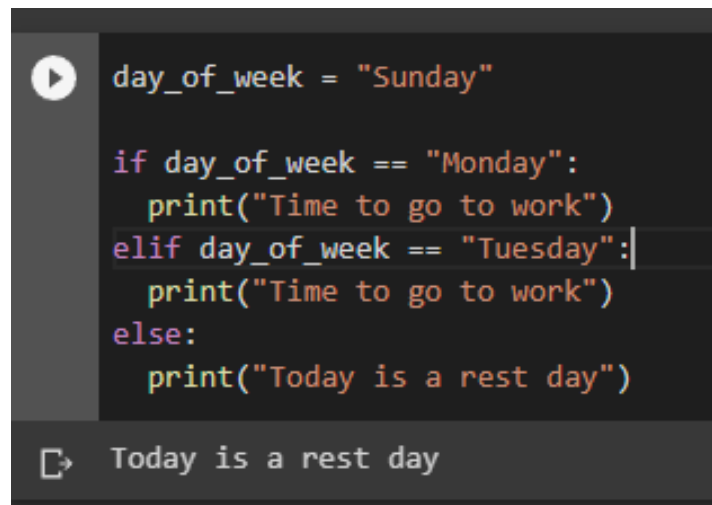
▶ day_of_week = "Sunday"

if day_of_week == "Monday":
    print("Time to go to work")
else:
    print("Today is a rest day")

Today is a rest day

```


If statements can be chained together to check multiple conditions. The first condition to be satisfied will be executed.

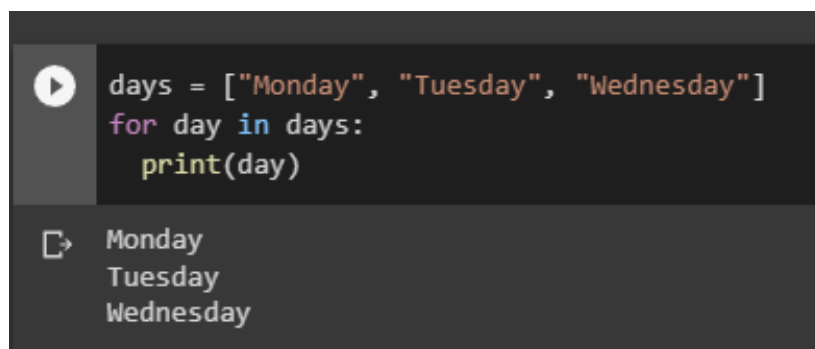


```
day_of_week = "Sunday"

if day_of_week == "Monday":
    print("Time to go to work")
elif day_of_week == "Tuesday":
    print("Time to go to work")
else:
    print("Today is a rest day")
```

Today is a rest day

Loops are used whenever we want a section of code to be run multiple times. A **for** loop is used when we know in advance that the loop will run a finite number of times.



```
days = ["Monday", "Tuesday", "Wednesday"]
for day in days:
    print(day)
```

Monday
Tuesday
Wednesday

In other situations, we may not necessarily know in advance how many times the loop should be run. In this case we might use a **while** loop. In a while loop we specify the conditions required for the loop to continue running. When the condition is no longer true it will stop.

```
import random

balance = 1000.0

while balance > 0:
    print(balance)
    food_costs = (100.0 * random.random())
    balance = balance - food_costs
```

```
1000.0
989.5011159189954
902.6306213166715
894.0332808457473
844.8074241401617
764.8663633703978
751.8375814810196
745.1493904076982
654.3014005461719
563.8176852076558
500.6277696614095
449.84013708305747
391.5515815176297
357.97940410685857
258.9315540703164
165.83672141652613
124.10025778930228
120.48746453946575
69.22443068383657
```

1.10. Logical and Membership Operators

Logical operators (or, and, not) are used in Python programs when we want to connect two or more expressions. For example, we may choose to run a section of code only when multiple sets of conditions are satisfied or when perhaps at least one is satisfied.

While we could write multiple IF..ELSE statements to achieve this, it is often more convenient (and requires less code) to do this using a single IF..ELSE statement using logical operators.

OR – requires one or more conditions to be TRUE

AND – requires all conditions to be TRUE

NOT – negates the logic of the surrounding operators (does the inverse)

```
day_of_week = "Sunday"

if day_of_week == "Saturday" or day_of_week == "Sunday":
    print("Weekend")
else:
    print("Weekday")
```

Weekend

```
day_of_week = "Sunday"

if not day_of_week == "Saturday" and not day_of_week == "Sunday":
    print("Weekday")
else:
    print("Weekend")
```

Weekend

The **membership operator** (`in`) can be used to test for the presence of a value in another variable. The `in` operator is very useful in situations where we might otherwise have to do many comparisons.


```
day_of_week = "Sunday"
weekend_days = ["Saturday", "Sunday"]

if not day_of_week in weekend_days:
    print("Weekday")
else:
    print("Weekend")
```

Weekend

1.11. Adding comments to your code

It's good practice to add comments to your code to explain the purpose of important statements. In Python **triple quotes** are used for comments that span multiple lines. Single line comments can be added by adding a hash (`#`) symbol at the start of any line. Any code written inside comments is **not** executed by Python and is simply ignored.



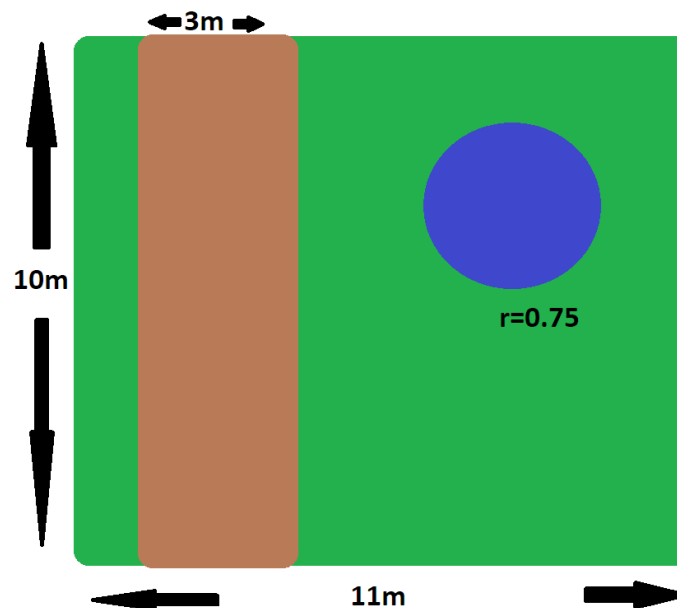
```
# This is a single line comment

"""
This is a a
comment
that
spans
multiple lines
"""
```

Questions

Based on the above theory and examples, attempt the following questions.

- 1) Create a variable that stores your first and last name. Confirm that the data type assigned to this variable is a string and print the result to the console. Modify your program so that instead of printing your name, it prints only your first and last initial.
- 2) A landscape gardener charges £0.75 per square meter of grass that is cut. Develop a Python program to calculate the cost of the job using the garden layout shown below. Any job costing less than £100 incurs a £10 surcharge.



- 3) The following is a list of tweets that were posted on Twitter on a Monday morning. Develop a Python program that prints out each tweet, the length of each tweet and the number of times that the word "coffee" or the hashtag "#coffee" appears in each tweet.

I love #coffee . Buy it #organic , grind your own beans, use unbleached filters. Coffee drinker or not?
Looking for a healthy, simple, and natural alternative for #coffee creamer? Compare to Coffee Mate
#Coffee & sun for a leisurely start today
Drink coffee: Do stupid things faster and with more energy
#Coffee Anyone? Our Coffee Corner knows just how to start your day!

- 4) Create a program that stores tweets that have been posted by different users. Assume that each user can post more than one tweet. For this question you should visit twitter.com to collect a sample of 10 or so tweets.

Your program should print out the name of each user, their tweet count, and the total number of words they have used across all tweets that they have posted.