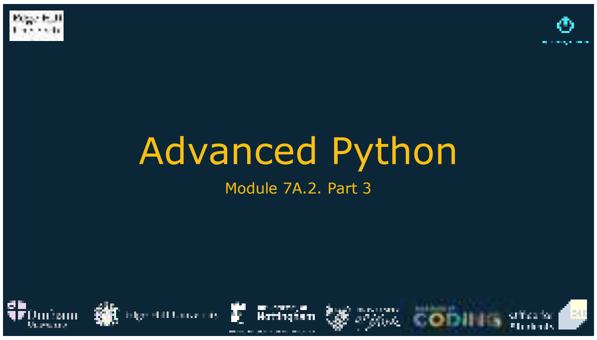
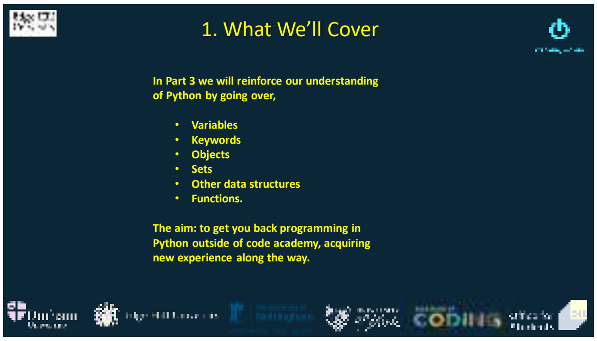


Advanced Python

Module 7A.2. Part 3

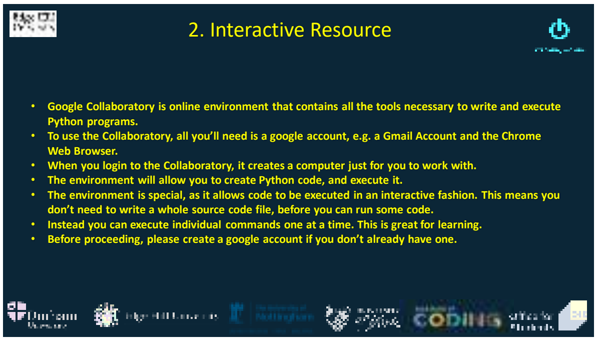


1. What We'll Cover

In Part 3 we will reinforce our understanding of Python by going over,

- Variables
- Keywords
- Objects
- Sets
- Other data structures
- Functions.

The aim: to get you back programming in Python outside of code academy, acquiring new experience along the way.



2. Interactive Resource

- Google Collaboratory is online environment that contains all the tools necessary to write and execute Python programs.
- To use the Collaboratory, all you'll need is a google account, e.g. a Gmail Account and the Chrome Web Browser.
- When you login to the Collaboratory, it creates a computer just for you to work with.
- The environment will allow you to create Python code, and execute it.
- The environment is special, as it allows code to be executed in an interactive fashion. This means you don't need to write a whole source code file, before you can run some code.
- Instead you can execute individual commands one at a time. This is great for learning.
- Before proceeding, please create a google account if you don't already have one.

3. Google Colab

4

4. Google Colab Notebook

Link to the notebook:
https://colab.research.google.com/drive/1JNwsQ6PM7IifWK2fXMEa0PPjw2RjxIMj

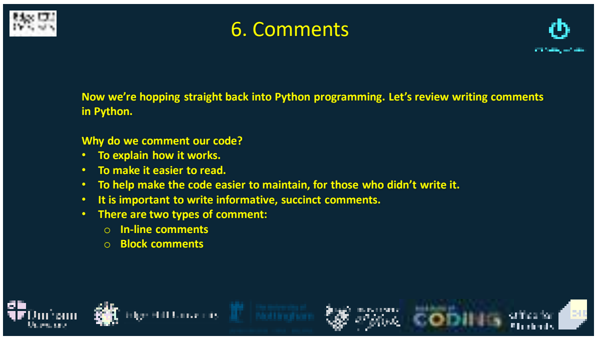
5

5. Setting Standards

- So far we've used code academy to write our code.
- We've not thought about the best way to write code – how to make it clear, concise, and easy to maintain.
- There is a set of standards, some might say guidelines, for writing "good" Python code.
- The standard was created when a Python Enhancement Proposal (PEP) was approved.
- PEP 8 defined some coding standards which I encourage you to adopt.
- The code we write will adhere to this standard – but if you're ever in doubt about coding styles or conventions – check the standard.
- Good programmers try to keep to the standards!

Link to the PEP8 Coding Guidelines:
https://www.python.org/dev/peps/pep-0008/#comments

6

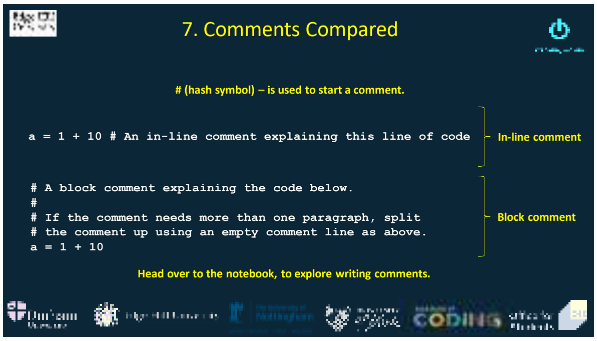


6. Comments

Now we're hopping straight back into Python programming. Let's review writing comments in Python.

Why do we comment our code?

- To explain how it works.
- To make it easier to read.
- To help make the code easier to maintain, for those who didn't write it.
- It is important to write informative, succinct comments.
- There are two types of comment:
  - In-line comments
  - Block comments



7. Comments Compared

# (hash symbol) – is used to start a comment.

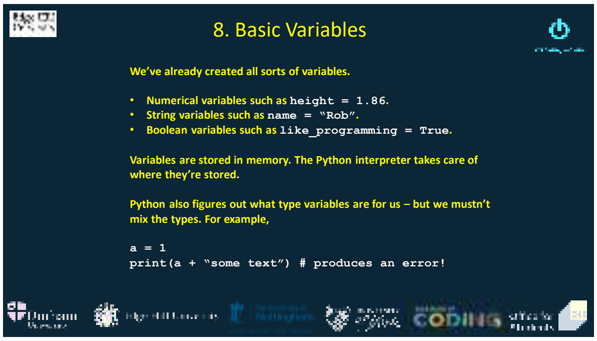
```
a = 1 + 10 # An in-line comment explaining this line of code
```

In-line comment

```
# A block comment explaining the code below.
#
# If the comment needs more than one paragraph, split
# the comment up using an empty comment line as above.
a = 1 + 10
```

Block comment

Head over to the notebook, to explore writing comments.



8. Basic Variables

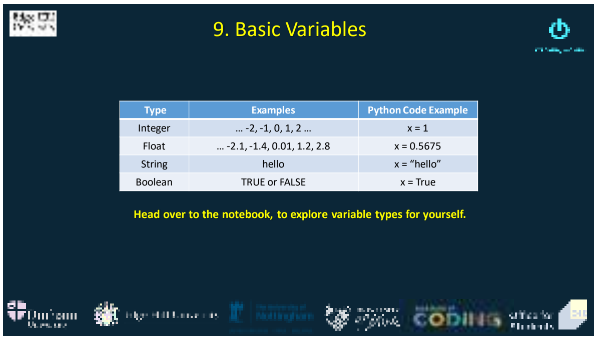
We've already created all sorts of variables.

- Numerical variables such as `height = 1.86`.
- String variables such as `name = “Rob”`.
- Boolean variables such as `like_programming = True`.

Variables are stored in memory. The Python interpreter takes care of where they're stored.

Python also figures out what type variables are for us – but we mustn't mix the types. For example,

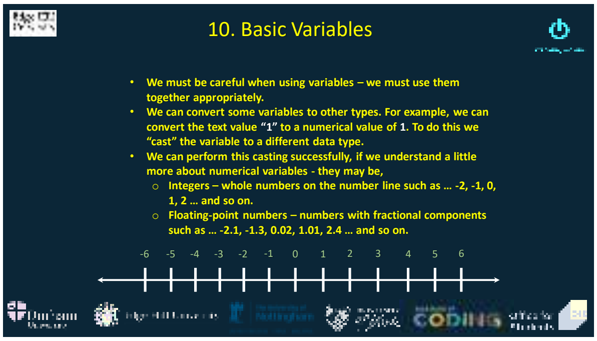
```
a = 1
print(a + “some text”) # produces an error!
```



9. Basic Variables

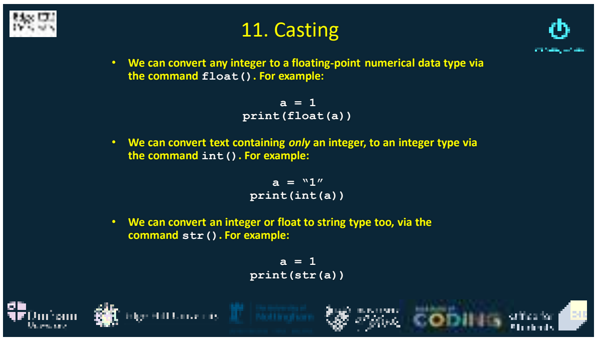
Type	Examples	Python Code Example
Integer	… -2, -1, 0, 1, 2 …	x = 1
Float	… -2.1, -1.4, 0.01, 1.2, 2.8	x = 0.5675
String	hello	x = “hello”
Boolean	TRUE or FALSE	x = True

Head over to the notebook, to explore variable types for yourself.



10. Basic Variables

- **We must be careful when using variables – we must use them together appropriately.**
- **We can convert some variables to other types. For example, we can convert the text value “1” to a numerical value of 1. To do this we “cast” the variable to a different data type.**
- **We can perform this casting successfully, if we understand a little more about numerical variables - they may be,**
  - **Integers – whole numbers on the number line such as … -2, -1, 0, 1, 2 … and so on.**
  - **Floating-point numbers – numbers with fractional components such as … -2.1, -1.3, 0.02, 1.01, 2.4 … and so on.**



11. Casting

- **We can convert any integer to a floating-point numerical data type via the command `float()`. For example:**

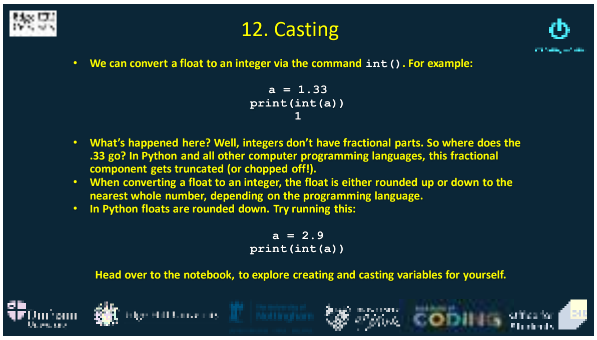
```
a = 1
print(float(a))
```

- **We can convert text containing *only* an integer, to an integer type via the command `int()`. For example:**

```
a = “1”
print(int(a))
```

- **We can convert an integer or float to string type too, via the command `str()`. For example:**

```
a = 1
print(str(a))
```



12. Casting

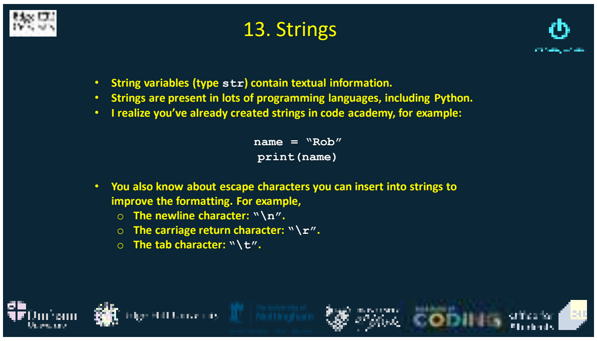
- We can convert a float to an integer via the command `int()`. For example:

```
a = 1.33
print(int(a))
1
```

- What’s happened here? Well, integers don’t have fractional parts. So where does the .33 go? In Python and all other computer programming languages, this fractional component gets truncated (or chopped off!).
- When converting a float to an integer, the float is either rounded up or down to the nearest whole number, depending on the programming language.
- In Python floats are rounded down. Try running this:

```
a = 2.9
print(int(a))
```

Head over to the notebook, to explore creating and casting variables for yourself.

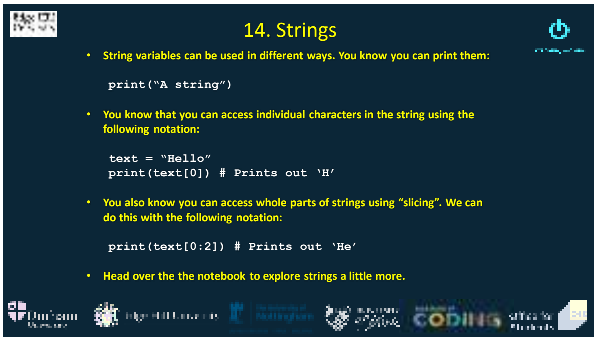


13. Strings

- String variables (type `str`) contain textual information.
- Strings are present in lots of programming languages, including Python.
- I realize you’ve already created strings in code academy, for example:

```
name = “Rob”
print(name)
```

- You also know about escape characters you can insert into strings to improve the formatting. For example,
  - The newline character: “\n”.
  - The carriage return character: “\r”.
  - The tab character: “\t”.



14. Strings

- String variables can be used in different ways. You know you can print them:

```
print(“A string”)
```

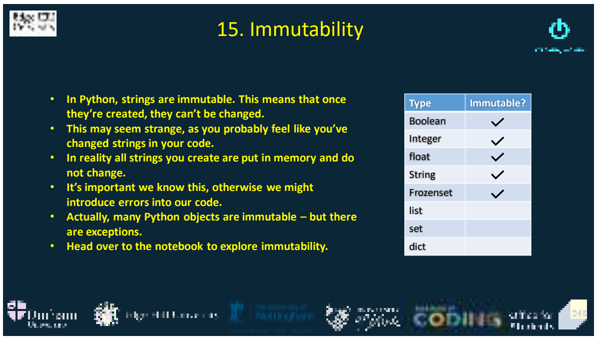
- You know that you can access individual characters in the string using the following notation:

```
text = “Hello”
print(text[0]) # Prints out ‘H’
```

- You also know you can access whole parts of strings using “slicing”. We can do this with the following notation:

```
print(text[0:2]) # Prints out ‘He’
```

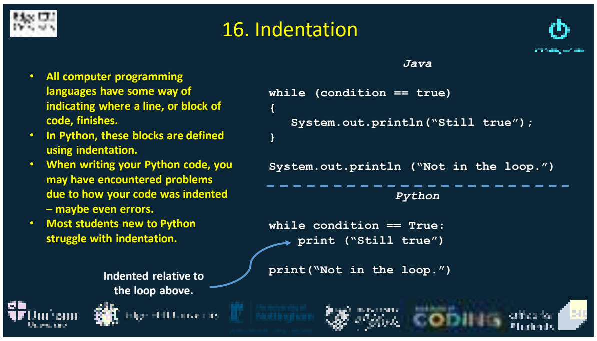
- Head over the the notebook to explore strings a little more.



15. Immutability

- In Python, strings are immutable. This means that once they're created, they can't be changed.
- This may seem strange, as you probably feel like you've changed strings in your code.
- In reality all strings you create are put in memory and do not change.
- It's important we know this, otherwise we might introduce errors into our code.
- Actually, many Python objects are immutable – but there are exceptions.
- Head over to the notebook to explore immutability.

Type	Immutable?
Boolean	✓
Integer	✓
float	✓
String	✓
Frozenset	✓
list	
set	
dict	



16. Indentation

- All computer programming languages have some way of indicating where a line, or block of code, finishes.
- In Python, these blocks are defined using indentation.
- When writing your Python code, you may have encountered problems due to how your code was indented – maybe even errors.
- Most students new to Python struggle with indentation.

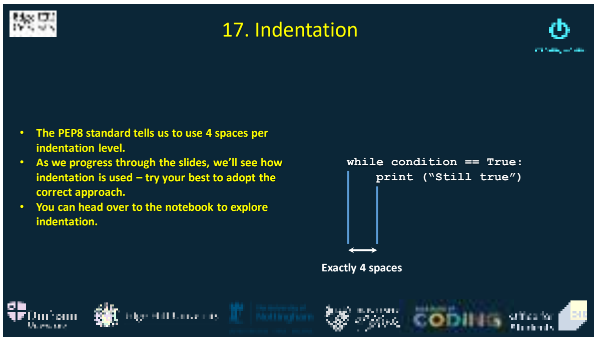
Java

```
while (condition == true)
{
   System.out.println(“Still true”);
}
System.out.println (“Not in the loop.”)
```

Python

```
while condition == True:
    print (“Still true”)

print(“Not in the loop.”)
```

Indented relative to the loop above.

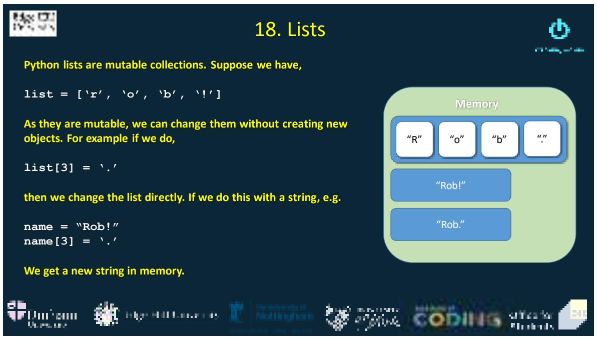


17. Indentation

- The PEP8 standard tells us to use 4 spaces per indentation level.
- As we progress through the slides, we'll see how indentation is used – try your best to adopt the correct approach.
- You can head over to the notebook to explore indentation.

```
while condition == True:
    print (“Still true”)
```

Exactly 4 spaces



18. Lists

Python lists are mutable collections. Suppose we have,

```
list = [‘r’, ‘o’, ‘b’, ‘!’]
```

As they are mutable, we can change them without creating new objects. For example if we do,

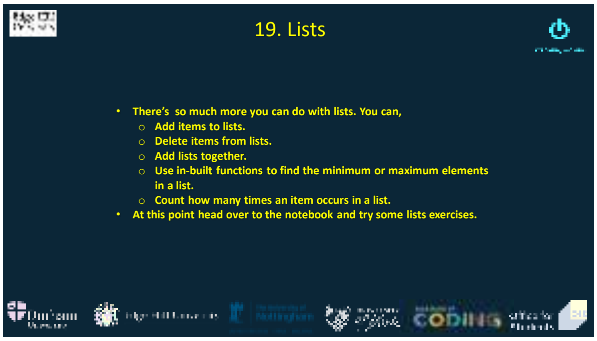
```
list[3] = ‘.’
```

then we change the list directly. If we do this with a string, e.g.

```
name = “Rob!”
name[3] = ‘.’
```

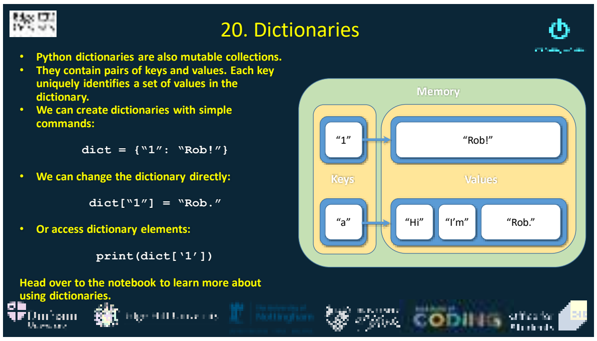
We get a new string in memory.

Memory
“R”
“o”
“b”
“.”
“Rob!”
“Rob.”



19. Lists

- There’s so much more you can do with lists. You can,
  - Add items to lists.
  - Delete items from lists.
  - Add lists together.
  - Use in-built functions to find the minimum or maximum elements in a list.
  - Count how many times an item occurs in a list.
- At this point head over to the notebook and try some lists exercises.



20. Dictionaries

- Python dictionaries are also mutable collections.
- They contain pairs of keys and values. Each key uniquely identifies a set of values in the dictionary.
- We can create dictionaries with simple commands:

```
dict = {“1”: “Rob!”}
```

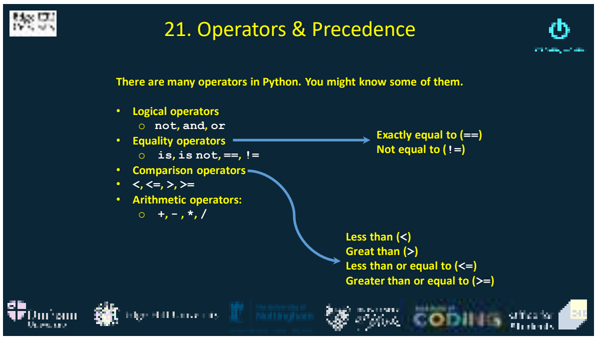
- We can change the dictionary directly:

```
dict[“1”] = “Rob.”
```

- Or access dictionary elements:

```
print(dict[‘1’])
```

Head over to the notebook to learn more about using dictionaries.



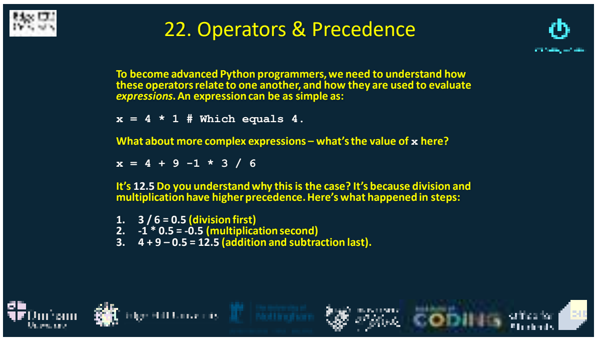
21. Operators & Precedence

There are many operators in Python. You might know some of them.

- Logical operators
  - `not, and, or`
- Equality operators
  - `is, is not, ==, !=`
- Comparison operators
- `<, <=, >, >=`
- Arithmetic operators:
  - `+, - , *, /`

Exactly equal to (==)
Not equal to (!=)

Less than (<)
Great than (>)
Less than or equal to (<=)
Greater than or equal to (>=)



22. Operators & Precedence

To become advanced Python programmers, we need to understand how these operators relate to one another, and how they are used to evaluate *expressions*. An expression can be as simple as:

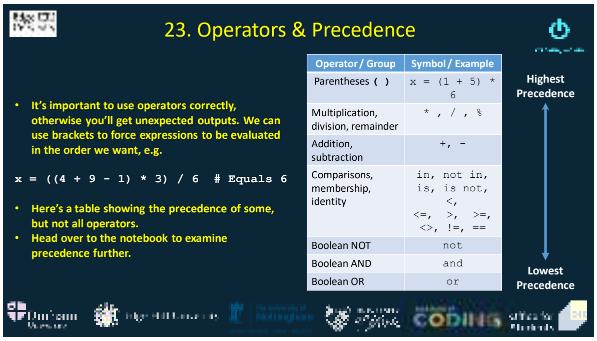
```
x = 4 * 1 # Which equals 4.
```

What about more complex expressions – what's the value of `x` here?

```
x = 4 + 9 -1 * 3 / 6
```

It's 12.5 Do you understand why this is the case? It's because division and multiplication have higher precedence. Here's what happened in steps:

1. 3 / 6 = 0.5 (division first)
2. -1 * 0.5 = -0.5 (multiplication second)
3. 4 + 9 – 0.5 = 12.5 (addition and subtraction last).



23. Operators & Precedence

- It's important to use operators correctly, otherwise you'll get unexpected outputs. We can use brackets to force expressions to be evaluated in the order we want, e.g.

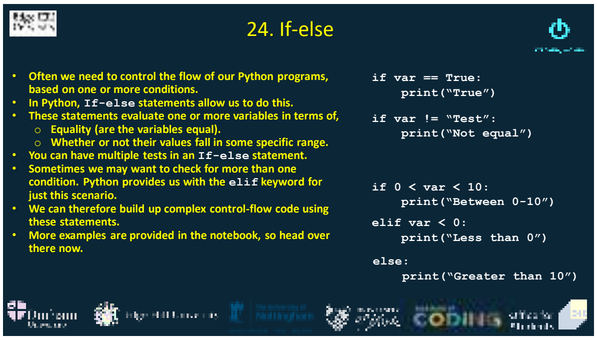
```
x = ((4 + 9 - 1) * 3) / 6  # Equals 6
```

- Here's a table showing the precedence of some, but not all operators.
- Head over to the notebook to examine precedence further.

Operator / Group	Symbol / Example
Parentheses `( )`	`x = (1 + 5) * 6`
Multiplication, division, remainder	`* , / , %`
Addition, subtraction	`+, -`
Comparisons, membership, identity	`in, not in, is, is not, <, <=, >, >=, <>, !=, ==`
Boolean NOT	`not`
Boolean AND	`and`
Boolean OR	`or`

Highest Precedence

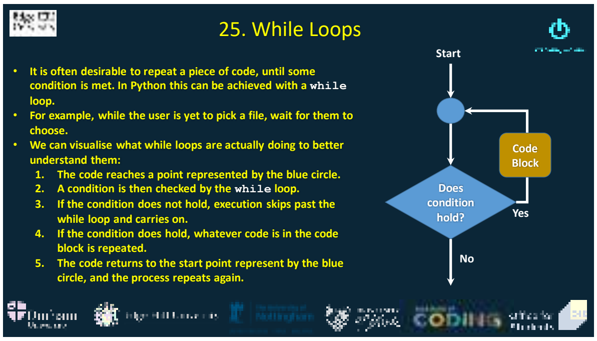
Lowest Precedence



24. If-else

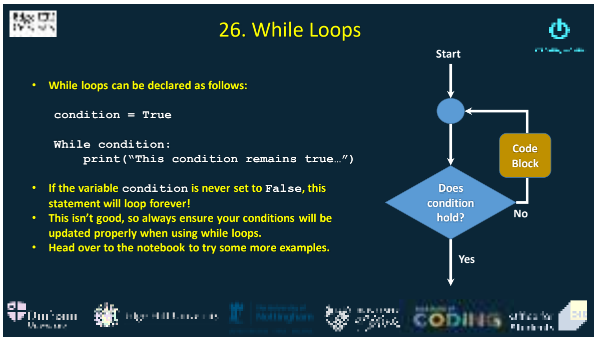
- Often we need to control the flow of our Python programs, based on one or more conditions.
- In Python, `If-else` statements allow us to do this.
- These statements evaluate one or more variables in terms of,
  - Equality (are the variables equal).
  - Whether or not their values fall in some specific range.
- You can have multiple tests in an `If-else` statement.
- Sometimes we may want to check for more than one condition. Python provides us with the `elif` keyword for just this scenario.
- We can therefore build up complex control-flow code using these statements.
- More examples are provided in the notebook, so head over there now.

```
if var == True:
    print(“True”)

if var != “Test”:
    print(“Not equal”)


if 0 < var < 10:
    print(“Between 0-10”)
elif var < 0:
    print(“Less than 0”)
else:
    print(“Greater than 10”)
```



25. While Loops

- It is often desirable to repeat a piece of code, until some condition is met. In Python this can be achieved with a `while` loop.
- For example, while the user is yet to pick a file, wait for them to choose.
- We can visualise what while loops are actually doing to better understand them:
  1. The code reaches a point represented by the blue circle.
  2. A condition is then checked by the `while` loop.
  3. If the condition does not hold, execution skips past the while loop and carries on.
  4. If the condition does hold, whatever code is in the code block is repeated.
  5. The code returns to the start point represent by the blue circle, and the process repeats again.

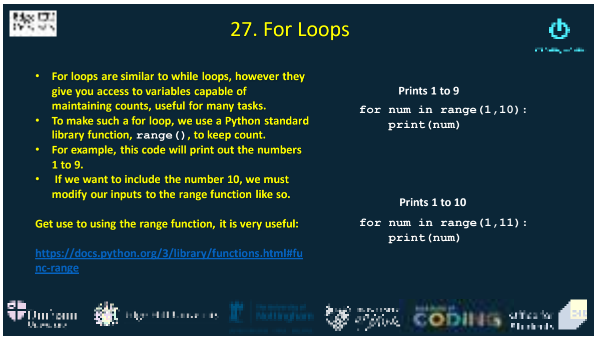


26. While Loops

- While loops can be declared as follows:

```
condition = True

While condition:
    print(“This condition remains true…”)
```

- If the variable `condition` is never set to `False`, this statement will loop forever!
- This isn’t good, so always ensure your conditions will be updated properly when using while loops.
- Head over to the notebook to try some more examples.



27. For Loops

- For loops are similar to while loops, however they give you access to variables capable of maintaining counts, useful for many tasks.
- To make such a for loop, we use a Python standard library function, `range()`, to keep count.
- For example, this code will print out the numbers 1 to 9.
- If we want to include the number 10, we must modify our inputs to the range function like so.

Get use to using the range function, it is very useful:

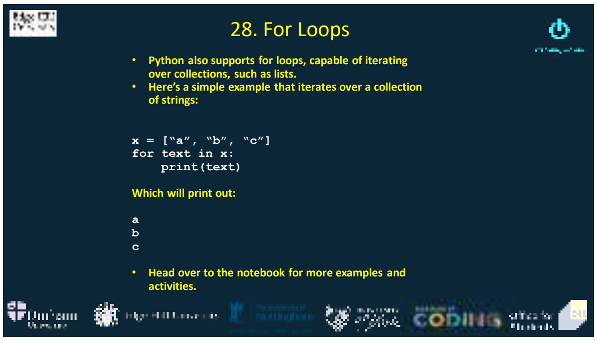
https://docs.python.org/3/library/functions.html#func-range

Prints 1 to 9

```
for num in range(1,10):
    print(num)
```

Prints 1 to 10

```
for num in range(1,11):
    print(num)
```



28. For Loops

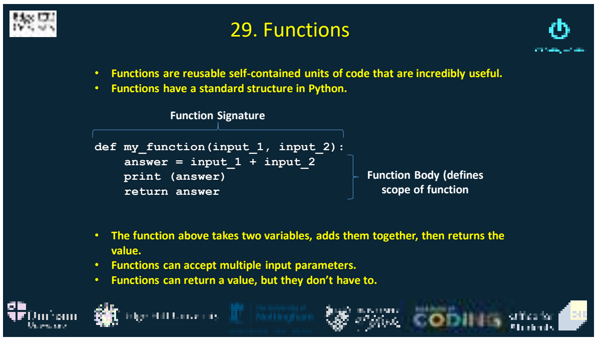
- Python also supports for loops, capable of iterating over collections, such as lists.
- Here’s a simple example that iterates over a collection of strings:

```
x = [“a”, “b”, “c”]
for text in x:
    print(text)
```

Which will print out:

```
a
b
c
```

- Head over to the notebook for more examples and activities.



29. Functions

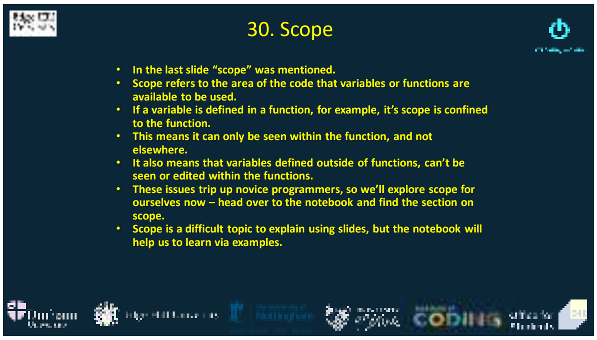
- Functions are reusable self-contained units of code that are incredibly useful.
- Functions have a standard structure in Python.

Function Signature

```
def my_function(input_1, input_2):
     answer = input_1 + input_2
     print (answer)
     return answer
```

Function Body (defines scope of function

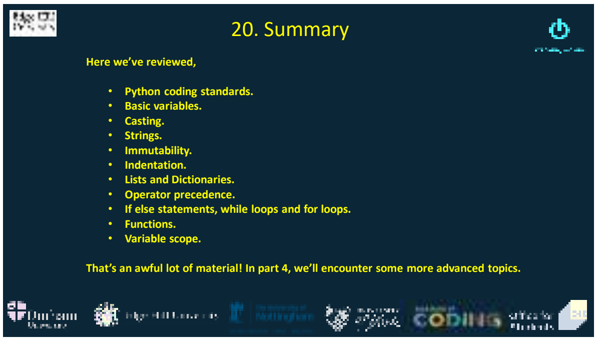
- The function above takes two variables, adds them together, then returns the value.
- Functions can accept multiple input parameters.
- Functions can return a value, but they don’t have to.



30. Scope

- In the last slide “scope” was mentioned.
- Scope refers to the area of the code that variables or functions are available to be used.
- If a variable is defined in a function, for example, it’s scope is confined to the function.
- This means it can only be seen within the function, and not elsewhere.
- It also means that variables defined outside of functions, can’t be seen or edited within the functions.
- These issues trip up novice programmers, so we’ll explore scope for ourselves now – head over to the notebook and find the section on scope.
- Scope is a difficult topic to explain using slides, but the notebook will help us to learn via examples.

31



20. Summary

Here we’ve reviewed,

- Python coding standards.
- Basic variables.
- Casting.
- Strings.
- Immutability.
- Indentation.
- Lists and Dictionaries.
- Operator precedence.
- If else statements, while loops and for loops.
- Functions.
- Variable scope.

That’s an awful lot of material! In part 4, we’ll encounter some more advanced topics.

32