Python Primer Sept 2025

Simon Tomlinson
simon.tomlinson@ed.ac.uk

1

What is a Programming Language?

- A programming language is a set of ‘rules’ for writing instructions that a computer understands
- A collection of instructions written in a file is called the programme source code
- Source code is then converted to an executable file (in some way) and run on the computer to perform a specific task

2

What is Python Programming?

- Python is a particular set of rules, a language, used to write instructions that will be executed on a computer
- Python is a general-purpose programming language and widely used in Edinburgh & world-wide
- There are two main ways we use Python in Edinburgh
  - For standalone programmes
  - For workflows

3

Standalone Programmes

- Individual applications that we run from a machine
- These programmes (or the source for these programmes) are accessible from the machine we run the programme
- These typically take input from the user and generate output
- These applications are available on the local machine to run

- Example
  - In Bioinformatics Algorithms we write a programme that aligns two DNA sequences. This programme takes sequence and alignment parameters as input, and generates an alignment text file as output

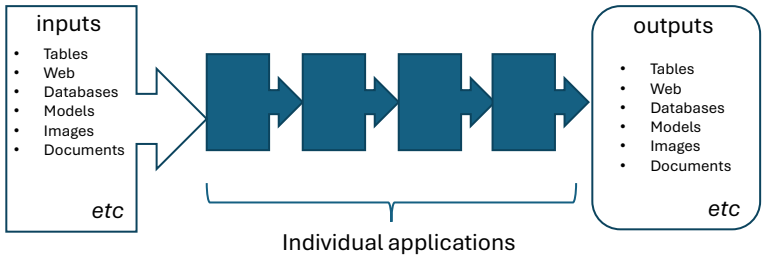
4

Python Workflows

- Typically used to perform complex data processing
- Collections of pre-written software components that are integrated together using Python or another language
- Data flows through the workflow and outputs are generated from that workflow
- Examples
  - Running complex machine learning pipelines (eg Applied Machine Learning)
  - Running systems biology simulations (eg Practical Systems Biology)
  - Data processing and mining of data (Introduction to Python Programming for Data Science)
  - Typically you might load some data, process and clean the data, run specific pre-built programmes to perform analysis and generate outputs
  - Python is used to connect all the different tasks together and manage the flow of data through the pipeline

5

Workflows and Applications



- Workflows typically orchestrate lot of individual applications feeding the output of one component into another component as input
- Individual applications take input and output and provide a step along the way

This is a simplification, but it is a useful way to think about the different ways we use Python within the teaching programmes.

6

Writing Python

- What do we need to get started with Python?
  - A tool to write Python code
  - A way to turn that code into an executable programme and run it on a computer
- Python code is just text (ASCII or Unicode) and so any text editor can be used to write code
- We need to have a Python interpreter installed to run the Python code on the local machine

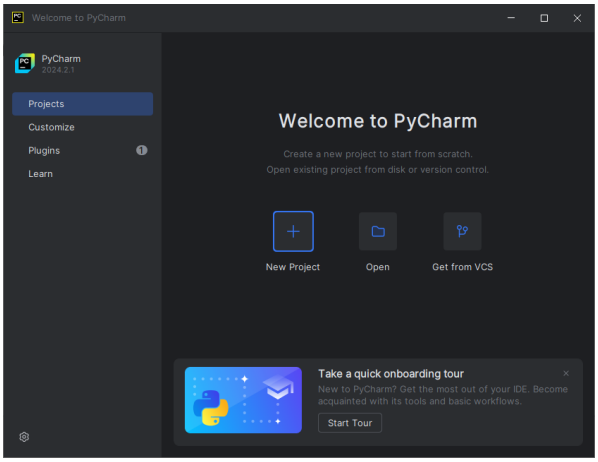
7

Python Integrated Development Environments (IDEs)

- IDEs provide software environments that support the writing of Python code
- They highlight code, check for errors, automatically provide hints on what code instructions to use. They help with programme execution
- Different IDEs are used on courses depending upon the particular task being completed
  - For workflows, RStudio server, Jupyter Hub or Visual Studio Code
  - For standalone, PyCharm or Visual Studio Code
- IDEs are installed on local teaching machines, servers or on the University central system for writing code called Noteable.
- You can also install IDEs on your own computer and use these to write code for courses

8

Creating and Running Python for this Course

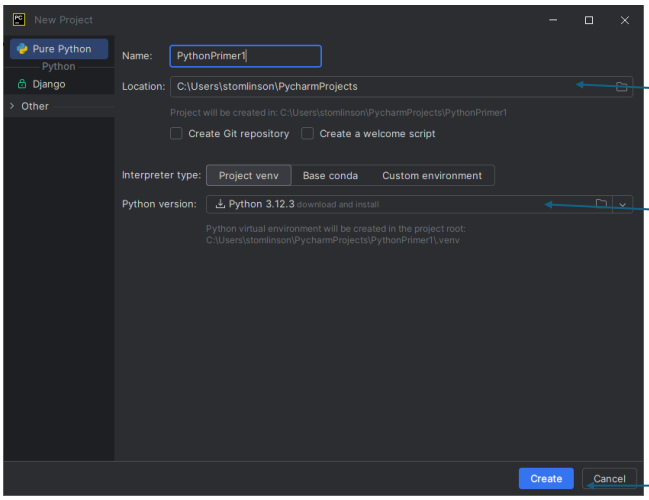


- PyCharm IDE is installed on the desktops
- We will use PyCharm for the class

- We will go through how to use this IDE later in the course- for now please follow the lecture!

9

New Project…



Files stored here on disk in this folder

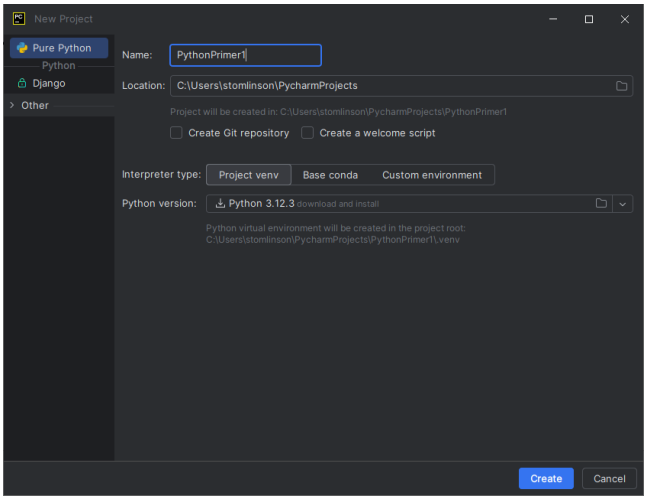
Select most recent Python to use

- You may see slightly different options and different paths
- But just give a name and select the Python version
- Leave other options as default (as shown)

Click Create

10

Features to Note…

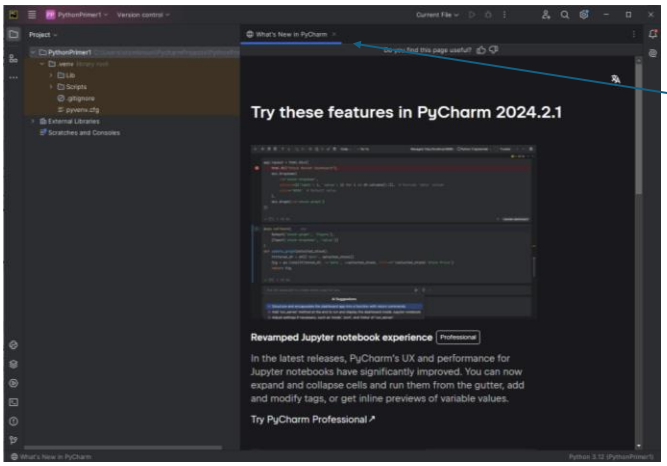


- PyCharm supports Git so you can create an online software archive using Git
- PyCharm will write some test code for you - we don't need this
- Pycharm supports venv and conda
- Pycharm will use a local Python version and download this and set this up

We return to some of these features later…

11

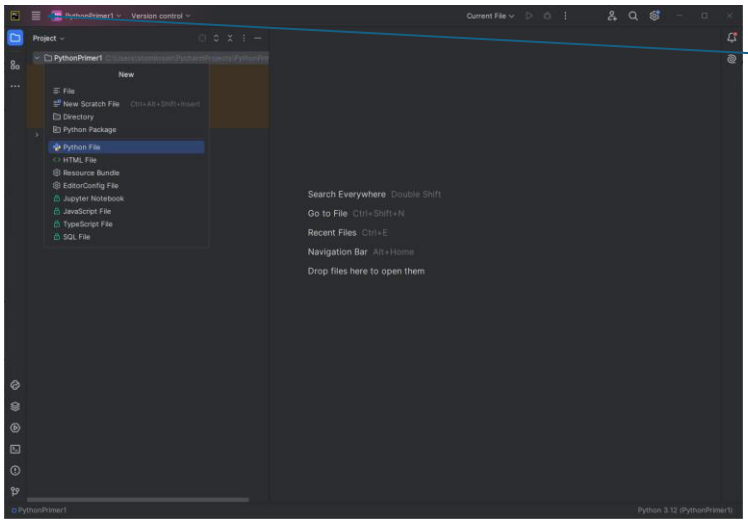
If you get this page



Close this by clicking "X"

12

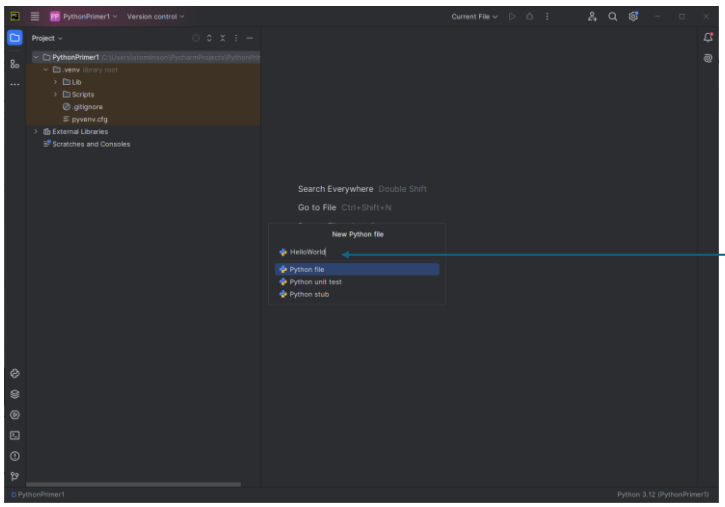
Creating A Python File



- Click here to display the Menu
- Select File New…
- Select Python File

13

Give the file a name…

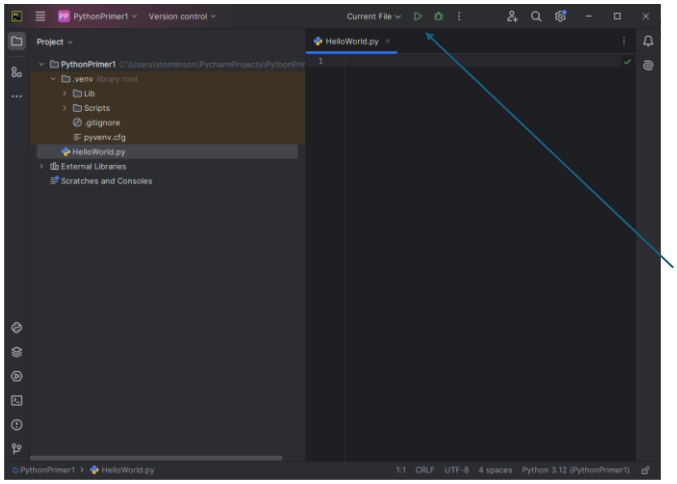


- Type filename “HelloWorld”
- Then press `<Enter>`

Note
`<Enter>` means press the Enter key
`<Tab>` means press the Tab key etc

14

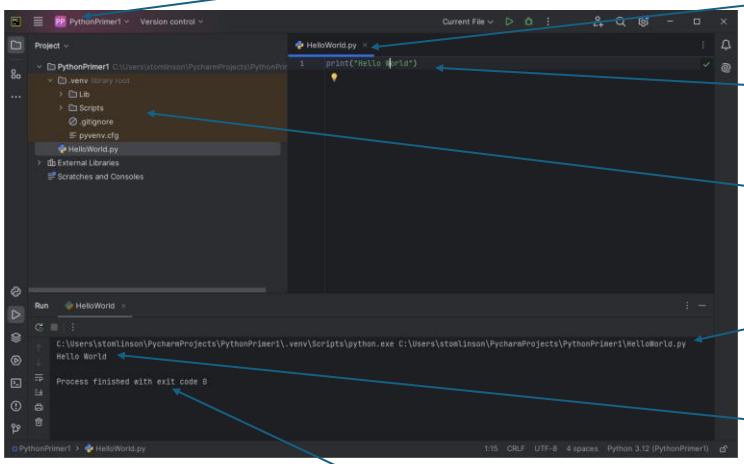
Almost there...



- We can now enter some Python code
- Traditionally the first application should print “Hello World”
- Enter the Python code into `HelloWorld.py`
  ```
  print(“Hello World”)
  ```
- Click the green arrow on the top bar to run the programme

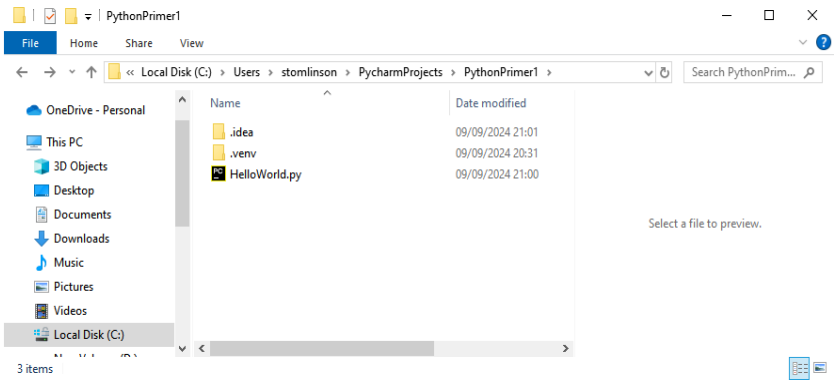
15

Success !



16

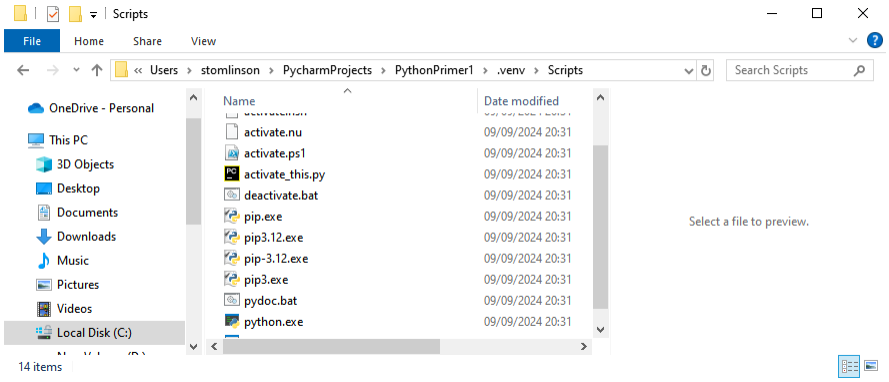
What is stored on Disk?



- PythonPrimer1 is just a folder that contains a text file containing the code
- ".venv" folder contains the Python environment

17

Inside the .venv Folder



- This contains programmes including python.exe used to run the code

18

Running Outside the IDE

```
C:\Users\stomlinson\PycharmProjects\PythonPrimer1>.venv\Scripts\python
.exe HelloWorld.py
Hello World
```

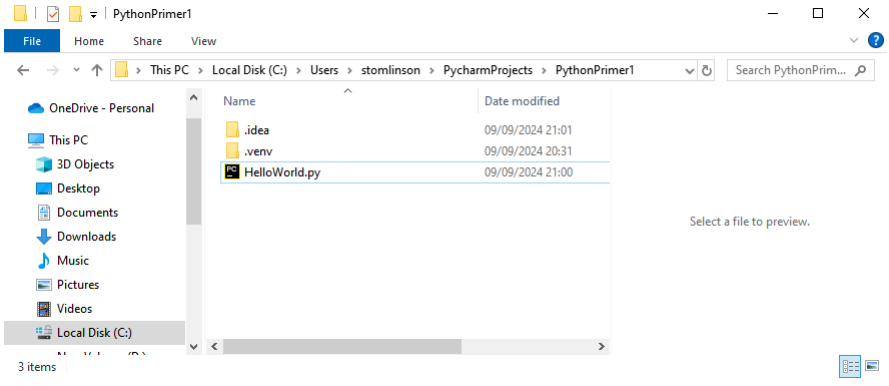
- Your file paths will be different- follow those listed in the IDE
- Here I've ran a Windows command line and entered the path to Python followed by the HelloWorld.py
- This runs the Python programme just like the IDE
- The output is "Hello World" as before

Note

- To run the Windows command line yourself, search for `Cmd` and run the "Command Prompt"
- `C: <Enter>` changes to the `C:` drive
- `cd <path>` changes to a path on the disk eg `cd C:\Users\stomlinson\PycharmProjects`

19

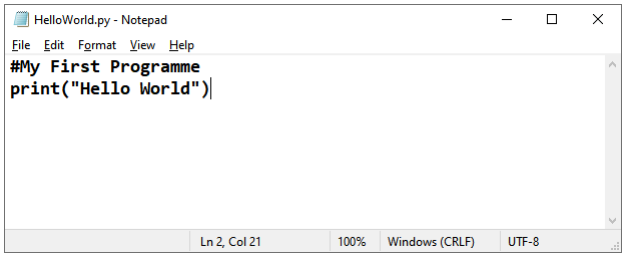
Opening the Python Programme from Windows



- Right click mouse while highlighting "`HelloWorld.py`"
- Select `Notepad.exe` or another text editor

20

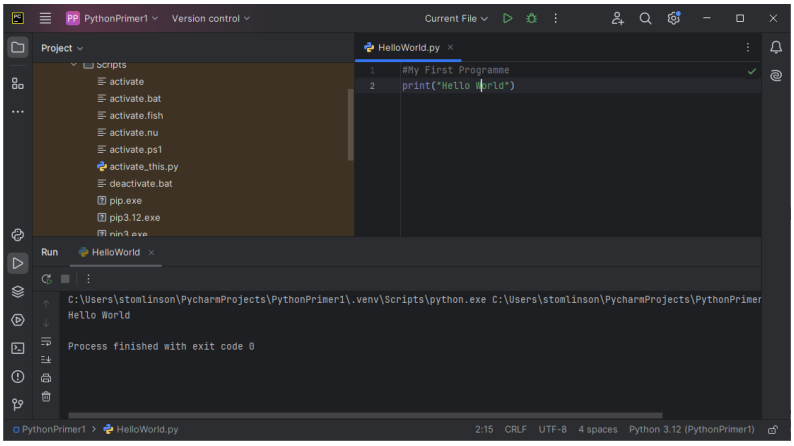
We Can Open (and Edit) the File



- I've edited the file to add a comment and saved the result using Notepad

21

Back in PyCharm



- The editor of PyCharm displays the updated file contents

22

So…

- The take home is that we are using the IDE to create and edit files and then run these files within the IDE
- But really, Projects are Windows folders, Python files are Windows ASCII or Unicode text files.
- When we view and edit code we are using a text editor
- Executing the file just runs python.exe and feeds in the Python code and outputs the result
- The `.env` folder contains the python executable files and the rest of the setup
- When we update setup, we change the contents of .env
- We can have many .env files

- `Understand the logic of how IDEs work and you can work with any IDE. The details of each IDE may be different, but the basic concepts discussed here are mirrored in many different IDEs`

23

Summary

- We've gone through how to load and launch the IDE
- We've looked at how to create a project and a new file
- We've looked at how to write and run a Python programme
- We've considered what happens on the disk when we use the IDE

- *Now use these slides as a guide and create your own HelloWorld application in PyCharm*

24

The Python Language

- It does not matter which IDE you are using, or the task you wish to complete with Python - the core Python language is a common feature!
- This means you can learn Python once and use this knowledge for lots of different purposes
- Better still, once you know one language, it is much easier to understand other programming languages such as R, Bash, Php or even SQL.

25

Consider Our First Programme

```
#My First Programme
print("Hello World")
```

Comment

Statement

- When this simple programme runs, each line is executed in order, top to bottom
- The first line is **comment**, and this is ignored by Python (it is for us humans to read)
- The line of code is a **`statement,`** but some statements extend for more than one line
- The `print` **command** is actually a **function** provided by Python, `print()` takes the "`Hello World`" **string** as a **parameter** and then outputs the result on the screen

Note that a `string` is a **type-** all elements of the same type "behave" in a similar way
Note also that types are entities that exist in a place in memory and (usually have) properties
A property is the value "Hello World" for this string

26

Consider this Programme

```
#My Second  Programme
print("Hello World")

myauthor = "From Simon"
print(myauthor)
```

- "Hello World" is a **string literal-** it is a variable that exists somewhere in Python memory but has no name
- `myauthor` is a **variable**, it exists in memory and has the property here "`From Simon`"
- I could change the value of `myauthor` just by reassigning a value to it
  I could enter `myauthor = "Simon T"` and change the variable to "Simon T"

27

Multiline Comments

```
#######################
#My Second  Programme#
#######################
'''
**********************
*My Second  Programme*
**********************
'''
```

- Use comments to annotate your code!

28

Dynamic Types

- Types are names for different kinds of variables in Python- for example Strings or Integers are types
- But how does Python set the type and how can we tell the type we have?

```
#My Second  Programme
print("Hello World")

myauthor = "From Simon"
print(myauthor)
```

- Python dynamically sets (ie "guesses") the type of a variable- this is designed to make programming easy but it is the source of *many* errors
- The command `id()` reports the memory address of a variable
- The command `type()` reports the type

29

Dynamic Types

- Types are names for different kinds of variables in Python- for example Strings or Integers are types
- But how does Python set the type and how can we tell the type we have?

```
#My Second  Programme
print("Hello World")

myauthor = "From Simon "
ii = id(myauthor)
ty = type(myauthor)

print("myauthor has type %s, ID number address %s" % (ty, ii))
```

Hello World
myauthor has type <class 'str'>, ID number address 2042456649648

- The command `id()` reports the memory address of a variable
- The command `type()` reports the type
- Notice the print command- here it prints out a string representation of the variables called `ii` and `ty`
- Note that `str` is the official type of a string in Python
- 2042456649648 is a memory address…

30

Types Change Dynamically

```
#My Second  Programme
print("Hello World")

myauthor = "From Simon "
myauthor = 12
ii = id(myauthor)
ty = type(myauthor)

print("myauthor has type %s, ID number %s" % (ty, ii))
```

Hello World
myauthor has type <class 'int'>, ID number
140712878021400

- Here I make one change- I change myauthor from value "`From Simon " to 12`
- OK this does not make much sense- but 12 is an integer and so what is the type of myauthor
- I did not explicitly change the type of `myauthor` but Python decided it is now a type `int (an integer)`

- Python dynamically changed the type

31

Memory Addresses Are Different

```
#My Second  Programme
print("Hello World")

myauthor = "From Simon "
ii = id(myauthor)
ty = type(myauthor)
print("myauthor has type %s, ID number %s" % (ty, ii))

myauthor = 12
ii = id(myauthor)
ty = type(myauthor)
print("myauthor has type %s, ID number %s" % (ty, ii))
```

Hello World
myauthor has type <class 'str'>, ID number
2271876034480
myauthor has type <class 'int'>, ID number
140712878021400

- So the name myauthor changed types in this programme dynamically
- myauthor does not change name
- myauthor does change the memory address it addresses

32

So

- `myauthor` is a variable- but really a variable is a **name** of something
- `myauthor` has a **dynamic type** that determines what can be done with this variable and what it stores
- `myauthor` is the name of a place where a **value** is stored
- `myauthor` is used to **locate a value stored in memory**
- A stored value in memory is called an **object**

Note: All variable are case sensitive names!

33

Python Language Basics

34

Data Types

Python has a number of built-in data types. You can also define your own types!

Commonly used Built-In Types

- Integer defined by the type `int`
- String defined by the type `str`
- Booleans (True/False) by type `bool`
- Float represented by the type `float`

35

Data Types

integer_value: 42 class: <class 'int'>
float_value: 23.23 class: <class 'float'>
bool_value: True class: <class 'bool'>
string_value: 42 class: <class 'str'>

```
#Examples of variables in Python- types are dynamic but set by          g
literal values
integer_value = 42
print("integer_value: ", integer_value, " class: ", type(integer_value))

float_value = 23.23
print("float_value: ", float_value, " class: ", type(float_value))

bool_value = True
print("bool_value: ", bool_value, " class: ", type(bool_value))

string_value = "42"
print("string_value: ", string_value, " class: ", type(string_value))
```

- Note the variable name does not set the type `float_value = True` would be a `bool` not a `float`
- Conventionally variable names are lowercase and cannot begin with a number

36

Operators

A character that represents a particular operation or logic.

`a +b`	- Add a and b together (+ is the operator)
`a - b`	- Substract b from a
`a *b`	- Multiply a *b
`a /b`	- a divided by b
`a % b`	- The modulus (remainder) after dividing a by b
`a = b`	-Assignment
`a == b`	-Equivalence operator checks two values are the same
`a >= b`	-Checks if something is greater than or equal to something else

37

Strings

- Represented by the str data type
- An ordered series of Unicode characters
- Can be created by assigning quoted string literal to a variable
- Quotes can be single or double quotes
- Strings have methods eg `st.capitalize()` change case of string
- String elements can be accessed by index such as `st[0]` or `st[1:7]`
- Python is zero indexed- the first element is `st[0]`
- In PyCharm starting typing a command will produce some suggestions `st.` for example

38

Flow Control Statements

```
if x >y:
    do_something()
elif p>q:
    do_something_else()
else:
    or_do_this()
```

- This is an example of an if statement
- x > y can be replaced by any logical operation (a statement that evaluates to True or False
- The statement can contain more than one elif: blocks of code

This reads, if x is greater than y then do_something() otherwise if p > q then do_something_else(). If none are true run or_do_this()

- do_something(), do_something_else or or_do_this() are functions (see later) but in the if statement we could have multiple lines of code
- This block of code inside statements is called a a **suite** in Python
- The blocks of code (like other blocks of code in Python) are defined by white spaces.

39

White Spaces in Python

```
if x >y:
    do_something()
elif p>q:
    do_something_else()
else:
    or_do_this()
```

- Blocks of code in Python are defined by white spaces
  - conventionally 4 spaces for each block of indenting shown in yellow
  -You can have multiple levels of indentation
- If blocks do not have exactly the same level of indenting code will error
- Simple if statements can be written in one line-try to avoid this as this can lead to errors- use multiple lines for if statements unless they are extremely simple

40

Other Flow Control- While Loop

```
#While loop
while x>y:
    do_something()
else:
    do_something_else()
```

- Here the else: part is optional
- The loop will run while x>y

41

For loop

```
#For loop
for x in y:
    do_something()
else:
    do_something_else()
```

- For each x in y `do_something()` otherwise `do_something_else()`
- y has to be iteratable ie a sequence of characters in a string that can be stepped through one after another

42

Break and Continue

- If a `break` statement is encountered the current running loop is exited at that point
- If a `continue` command is encountered, the running loop will move to the next iteration

- You will see break and continue often used with if statements as this then modifies the flow of execution under particular conditions.

43

Creating and Calling Functions

- Functions are block that can be called within the running programme.
- They can be passed parameters and return values
- When the flow of execution encounters a function call the flow of execution jumps to the function body, runs the commands in the function and then returns to the next line of the code
- Functions can be defined in a single file (eg at the top of a file) or brought into an application as a separate file or a library

44

Function Basic Design

```
def my_function(my_param):
    do_something()
    return avalue
```

- Here I create a function called `my_function` (give functions names that indicate what they do!!)
- This function excepts `my_param`, a single parameter. We can have multiple parameters in practice
- `do_something()` is a call to another function but inside the function could be a block of code
- `return` specifies a value that is returned to the calling programme when the programme completes
- If we do not specify a return – Python will return `None` – a type of null value
- Conventionally python function names are lowercase with "_" underscores separating words

45

Example Function

```
#An example function
def hi_world(takecredit):
    print("Hello World from ",takecredit)
    return "finished"

#We still need to call the function
result = hi_world("Simon")

print("The function returned: ", result)
```

- Note here the function body is defined by indentation
- The function returns "finished" as a variable called result

46

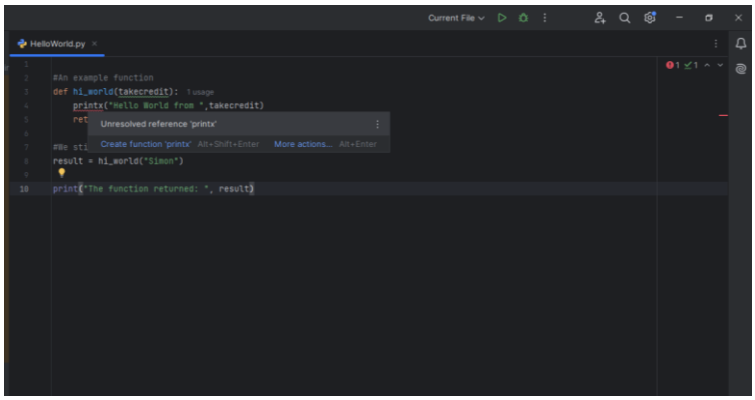
Handling Errors

- Python has very detailed (but complex) error handling methods
- There are different types of error- eg syntax errors in code or errors that you only encounter in running code
- Use the PyCharm IDE to discover errors in your code
- Use simple print statements to find errors in your running code

- As you develop your skills you can then start to use the more advanced features of the IDE to debug your programme or use error handling to captures errors automatically when your programme runs

47

Error Handling in PyCharm



- Here I have a typo (`printx` rather than `print`), the IDE spots this and underlines the code
- If you move the mouse pointer over the area the IDE will give an indication of why this is an error

48

Extra Reading

A good place to learn about Python is the official python tutorial

https://docs.python.org

49