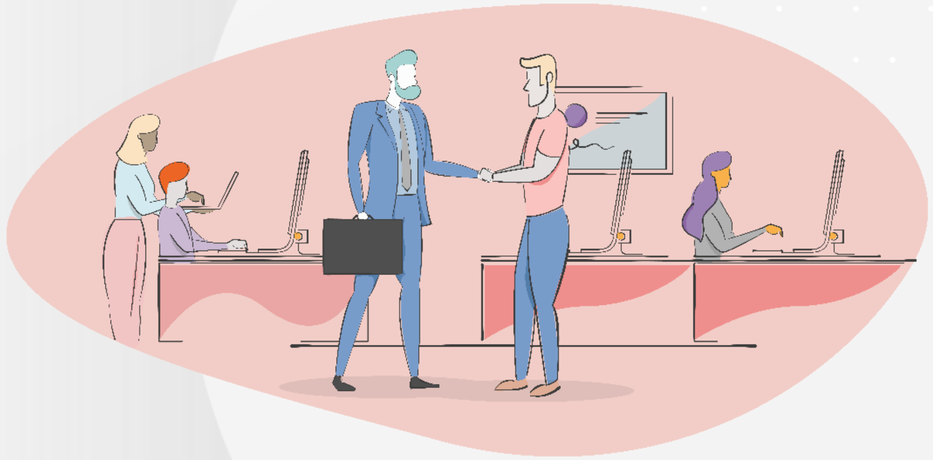
Python for data engineers



L5 Data Engineer Higher Apprenticeship
Module 3 / 12 (**“Programming and Scripting Essentials”**)
Topic 3 / 8

rev. 1 (2024)

Ice breaker: Discussion

A bit of fun to start…

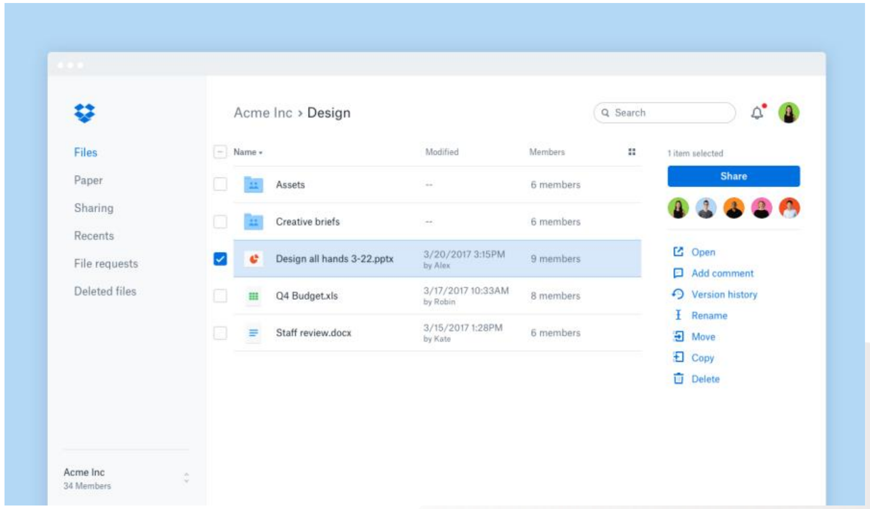
1. If you could have dinner with any historical figure, who would it be and why?
2. If you were given a one-time chance to teleport, where would you go and why?
3. Can you share an instance where Python significantly improved your data engineering workflow?

Submit your responses to the chat or turn on your microphone

The real-world value of Python

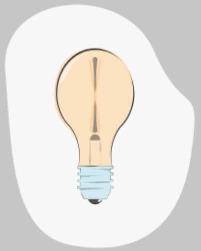
A real-world success story…

- Dropbox is a cloud-based platform that allows users to store their graphics, content, and files
- The company scaled to over 200 million users in just a few years, with Python playing a significant role in this success
- Dropbox uses Python extensively for almost everything on their platform



Name	Modified	Members
Assets	--	6 members
Creative briefs	--	6 members
Design all hands 3-22.pptx	3/20/2017 3:15PM by Alex	9 members
Q4 Budget.xls	3/17/2017 10:33AM by Robin	8 members
Staff review.docx	3/15/2017 1:28PM by Kate	6 members

Dropbox



Guido van Rossum, the creator of Python, even worked with Dropbox to make its Python Stack more reliable, contributing to the performance and reliability of their platform.

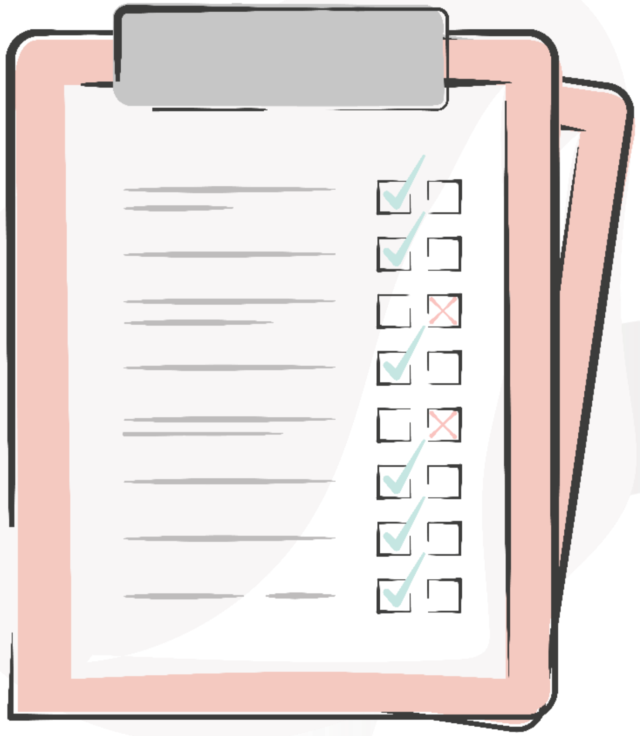
BPP

Webinar agenda

This webinar will cover the following:

- SDLC
- XP – eXtreme programming
- Getting started with Google Colab (practical)
- Testing

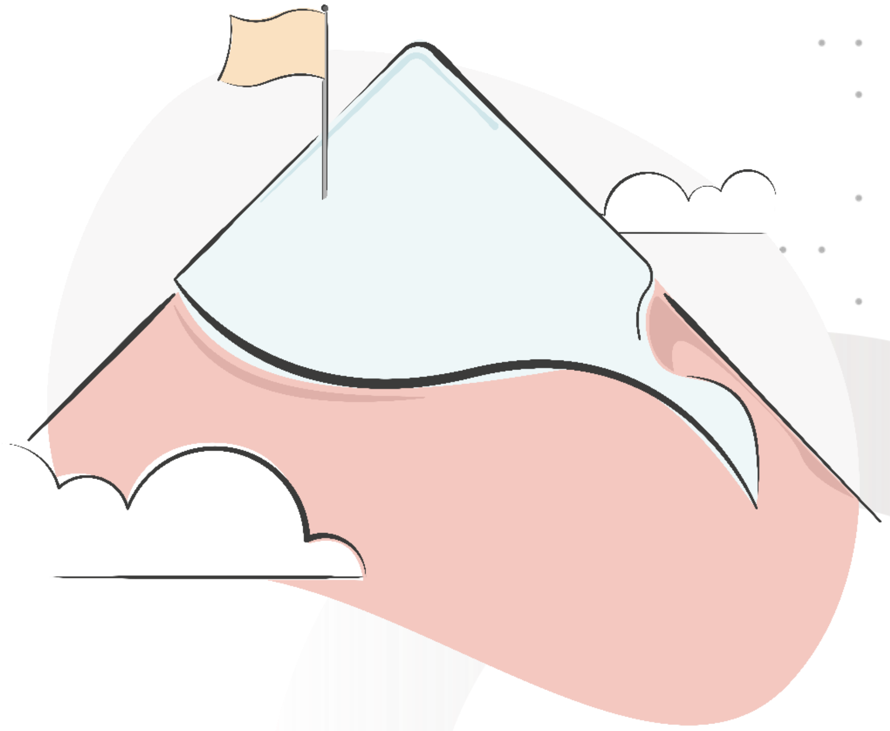
Webinar length: 3 hours



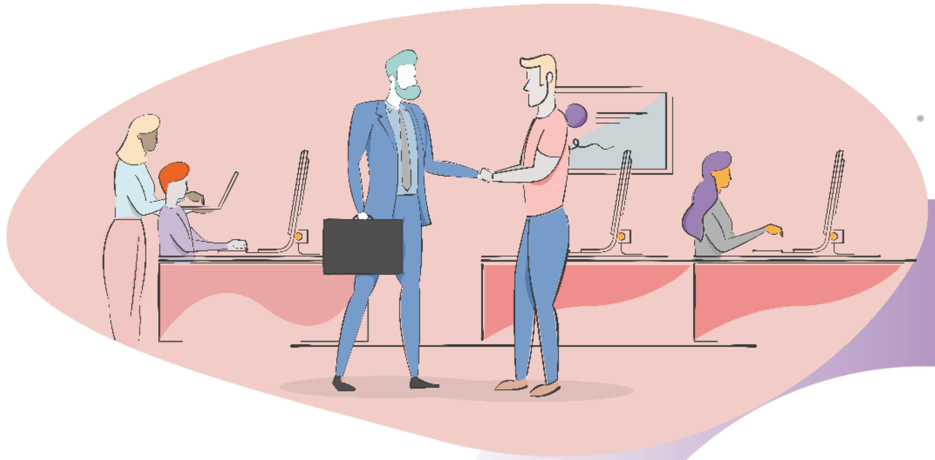
Learning objectives

This webinar supports completion of the following outcomes:

- Identify the main approaches to software development
- Apply variables and common operators in Python
- Explain software testing and test-driven development

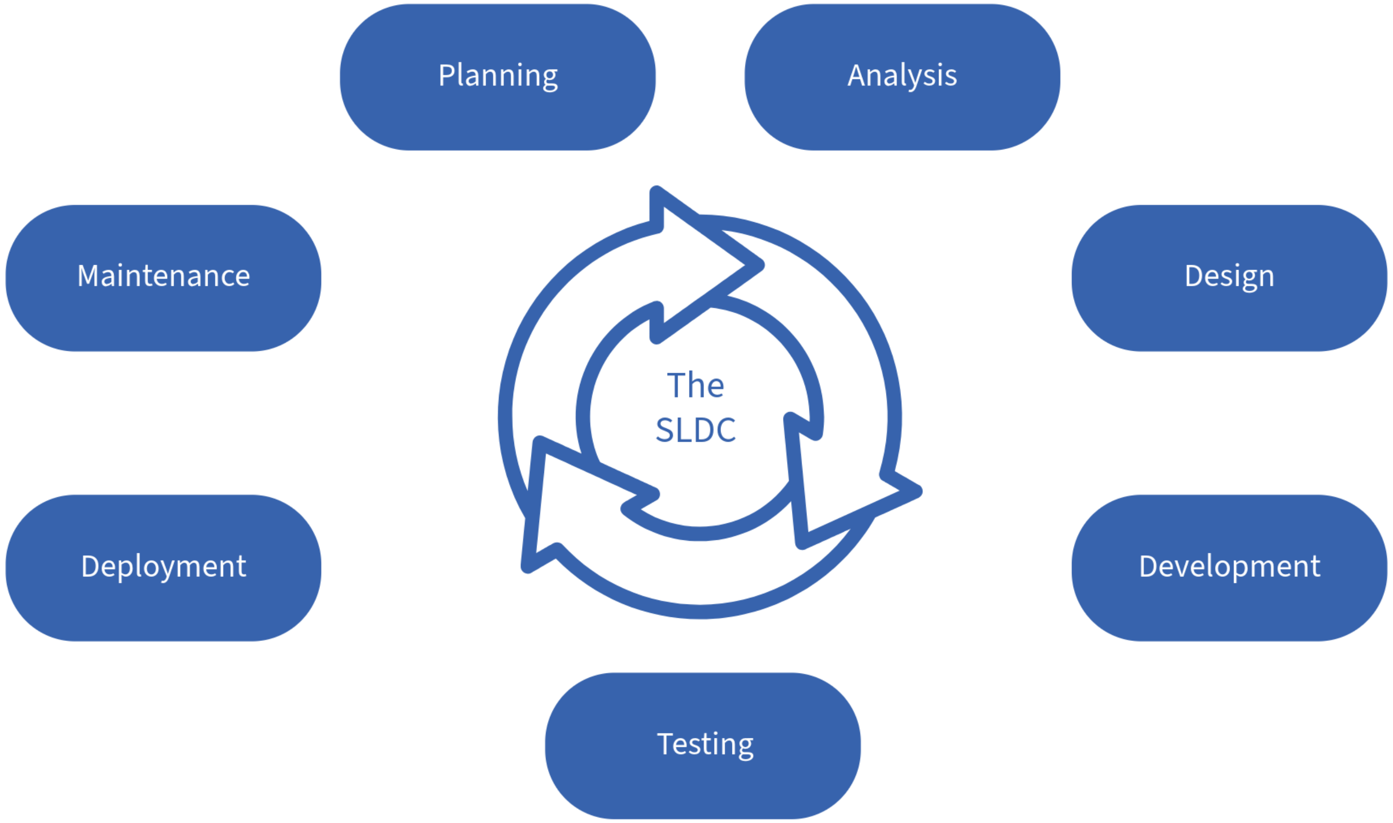


SDLC



The SLDC model

An introduction to the framework…



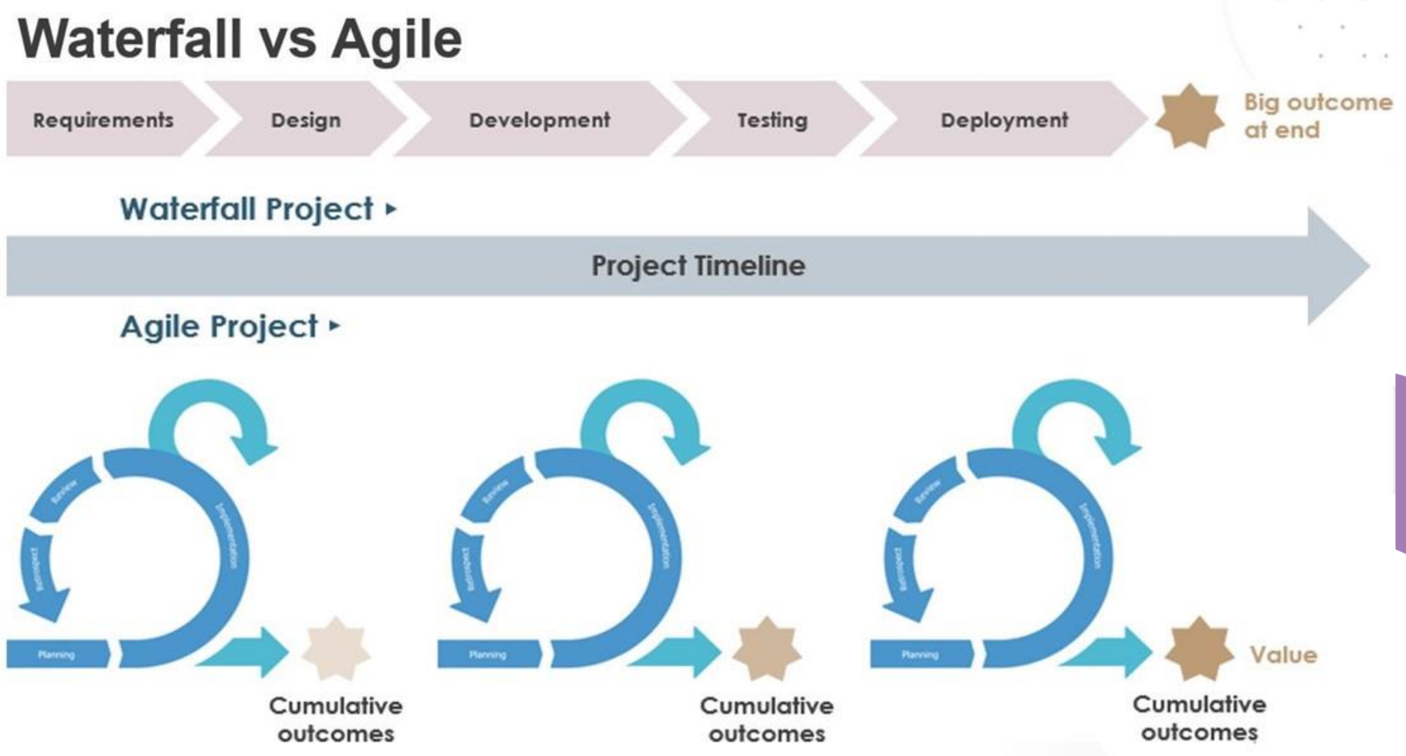
The typical stages of SLDC

SLDC is a framework that describes the activities performed at each stage of a software development project.

Agile SDLCs

The Agile approach…

- Speed up or bypass one or more life cycle phases
- Usually less formal and reduced scope
- Used for time-critical applications
- Used in organisations that employ disciplined methods



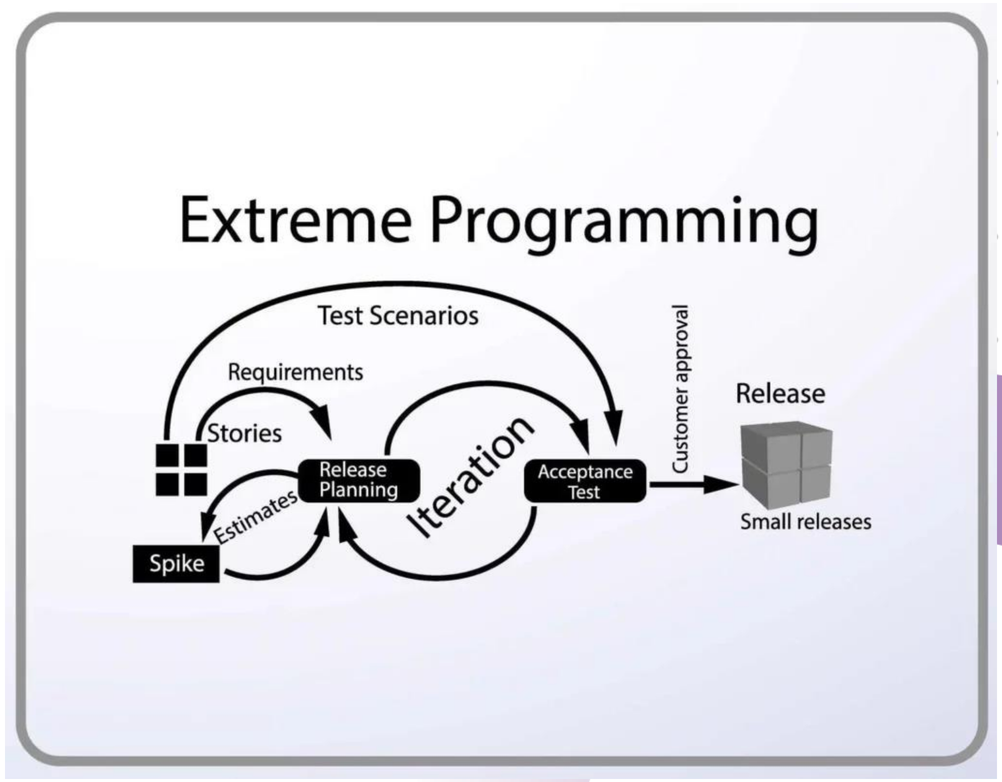
Waterfall vs Agile

Agile development frequently uses **iterative** development or **incremental** delivery (or both).

XP - eXtreme Programming

How does this approach compare?

- For small-to-medium-sized teams developing software with vague or rapidly changing requirements
- Coding is the key activity throughout a software project
- Communication among teammates is done with code
- Life cycle and behavior of complex objects defined in test cases – again in code

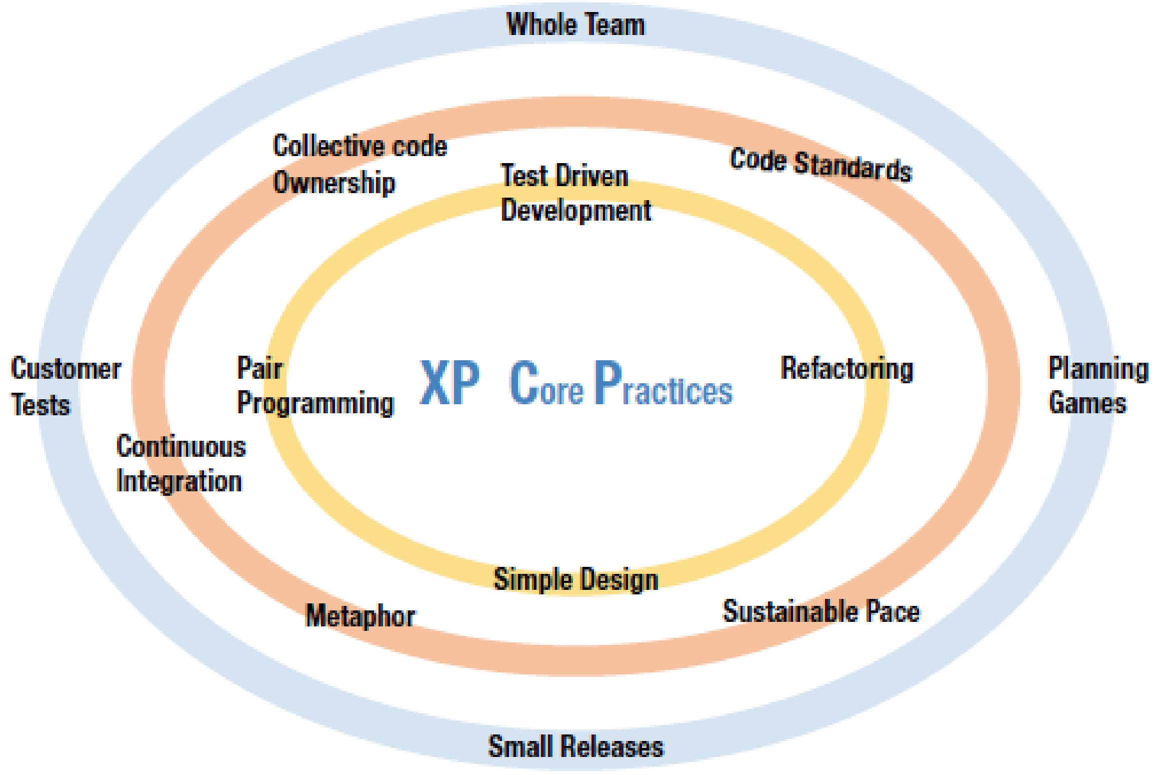


An illustration of the Extreme Programming approach

XP Practices

XP takes good practice to extreme levels

- **Code reviews:** Review code all the time
- **Testing:** Everybody tests all the time
- **Simplicity:** Maintain simplest design for current functionality
- **Design:** Everybody designs daily
- **Architecture:** Everybody refines the architecture
- **Integration Testing:** Build and integrate tests several times a day
- **Short Iterations:** Keep iterations extremely short



The core practices of XP

Knowledge Check Poll

Which of the following statements best describes a key principle of eXtreme Programming (XP) in the context of SDLC?

a) XP advocates for a strict adherence to sequential project phases without overlap

b) In XP, coding is considered a secondary activity that follows after detailed design documentation

c) XP emphasises continuous testing and integration throughout the software development process

d) The Waterfall model is preferred in XP due to its flexibility in handling changing requirements

Submit your responses to the chat!

Knowledge Check Poll

Which of the following statements best describes a key principle of eXtreme Programming (XP) in the context of SDLC?

a) XP advocates for a strict adherence to sequential project phases without overlap

b) In XP, coding is considered a secondary activity that follows after detailed design documentation

c) XP emphasises continuous testing and integration throughout the software development process

d) The Waterfall model is preferred in XP due to its flexibility in handling changing requirements

Feedback: d – Continuous testing and integration are core practices in XP which ensure that changes are tested and integrated frequently to maintain software quality.

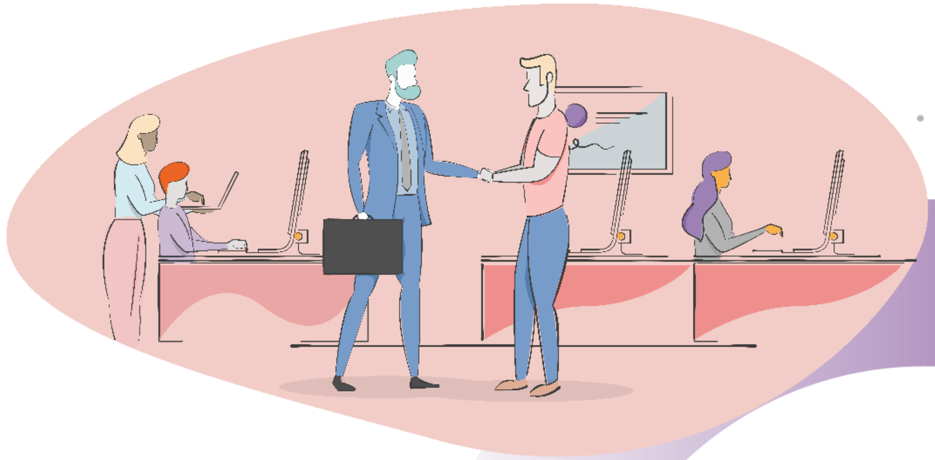
Building Careers Through Education

Everybody Matters. Stronger Together. Student, Learner and Client Centric. Trust and Respect. Embrace Change.

Submit your responses to the chat!

BPP

Python notebooks and Google Colab



Python: getting started

Let's examine our first notebook file together…

Let's examine the first Python notebook file together:

- Get used to writing and executing basic code in Colab
- Discuss different variables and data-types
- Examine how to use expressions and operators

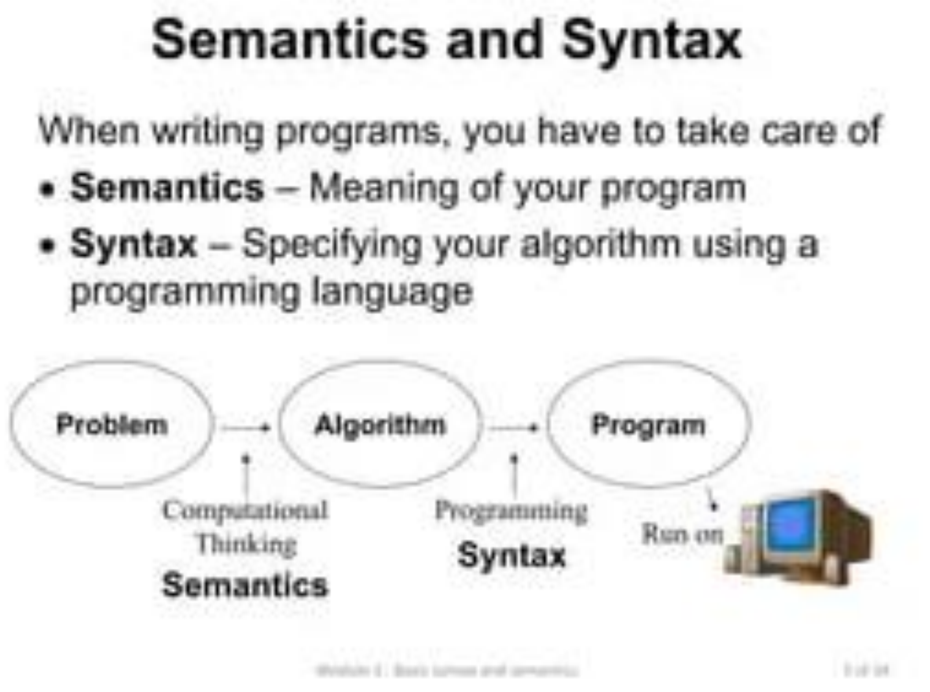
This file can be found at the following link: Python notebook link

Syntax vs semantics

What's the difference?

- **Syntax:** form, grammar
  - ➢ **Correct:** The dog is barking
  - ➢ **Incorrect:** The dog barking

- **Semantics:** meaning, interpretation
  - ➢ **Correct:** The dog barked
  - ➢ **Incorrect:** The dog spoke



Semantics and Syntax

When writing programs, you have to take care of

- **Semantics** – Meaning of your program
- **Syntax** – Specifying your algorithm using a programming language

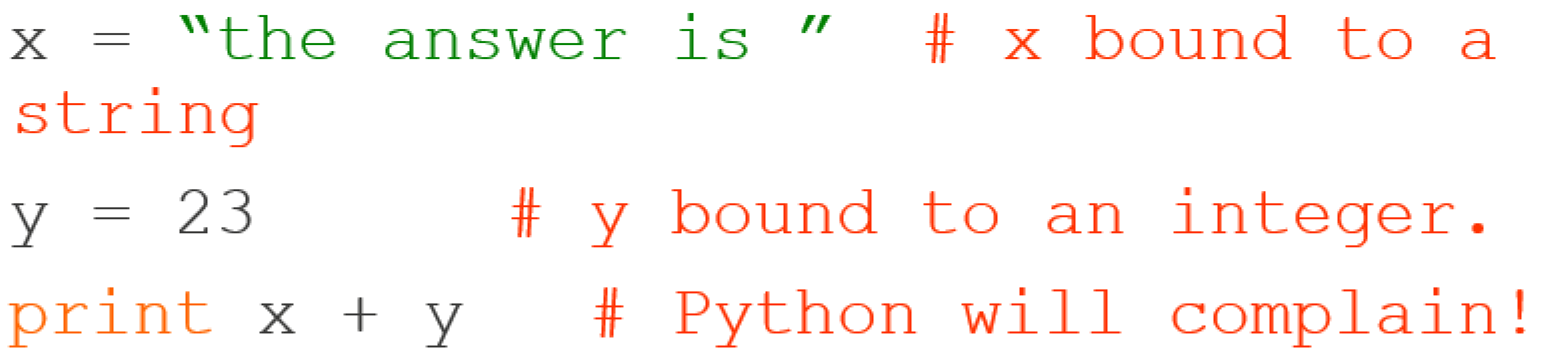
Problem
Algorithm
Program
Computational Thinking
Semantics
Programming
Syntax
Run on

From Problem to Program

Python and types

What’s the difference?

- **Dynamic typing:** Python determines the data types of variable bindings in a program automatically
- **Strong typing:** But Python’s not casual about types, it enforces the types of objects
- For example, you can’t just append an integer to a string, but must first convert it to a string



```
x = “the answer is ”   # x bound to a
string
y = 23          # y bound to an integer.
print x + y     # Python will complain!
```

Converting an Integer to a String

Defining functions

Functions in Python…

Function definition begins with “def.”

Function name and its arguments.

```
def get_final_answer(filename):
    “““Documentation String”””
    line1
    line2
    return total_counter
```

Colon.

The indentation matters…
First line with less
indentation is considered to be
outside of the function definition.

The keyword ‘return’ indicates the
value to be sent back to the caller.

No header file or declaration of types of function or arguments.

Calling a function

What you need to know…

Parameters in Python are Call by Assignment

- Old values for the variables that are parameter names are hidden, and these variables are simply made to refer to the new values

- All assignment in Python, including binding function parameters, uses reference semantics.

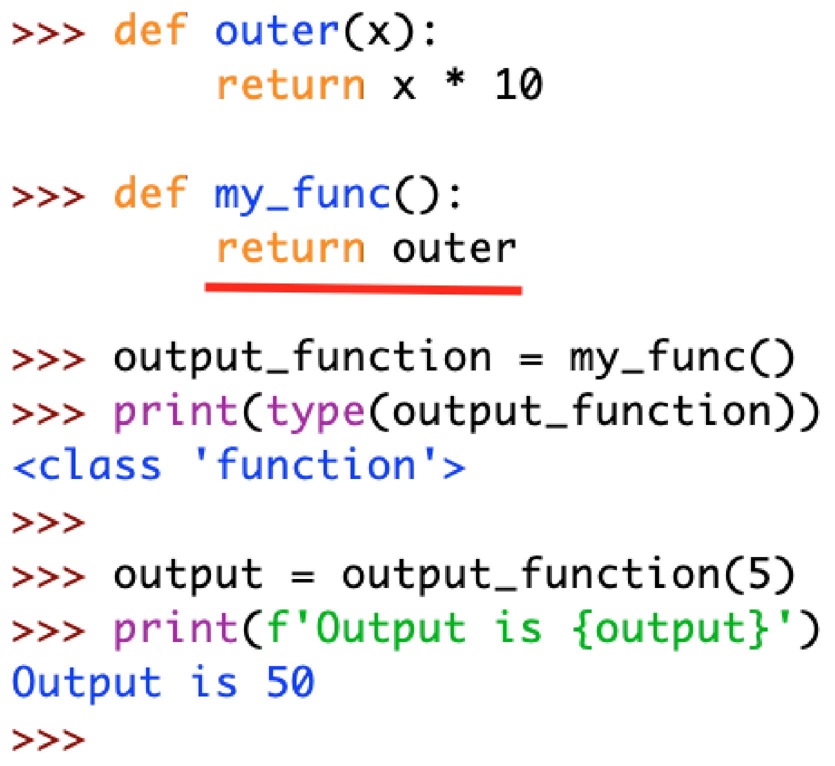
```
>>> def myfun(x, y):
        return x * y
>>> myfun(3, 4)
12
```

Remember? The syntax for a function call

Functions without returns

What you need to know…

- All functions in Python have a return value, even if no *return* line inside the code
- Functions without a *return* return the special value None
  - *None* is a special constant in the language
  - *None* is used like *NULL*, *void*, or *nil* in other languages
  - *None* is also logically equivalent to False
  - The interpreter doesn't print *None*



```
>>> def outer(x):
        return x * 10

>>> def my_func():
        return outer

>>> output_function = my_func()
>>> print(type(output_function))
<class 'function'>
>>>
>>> output = output_function(5)
>>> print(f'Output is {output}')
Output is 50
>>>
```

An example of a Python return statement

True and false

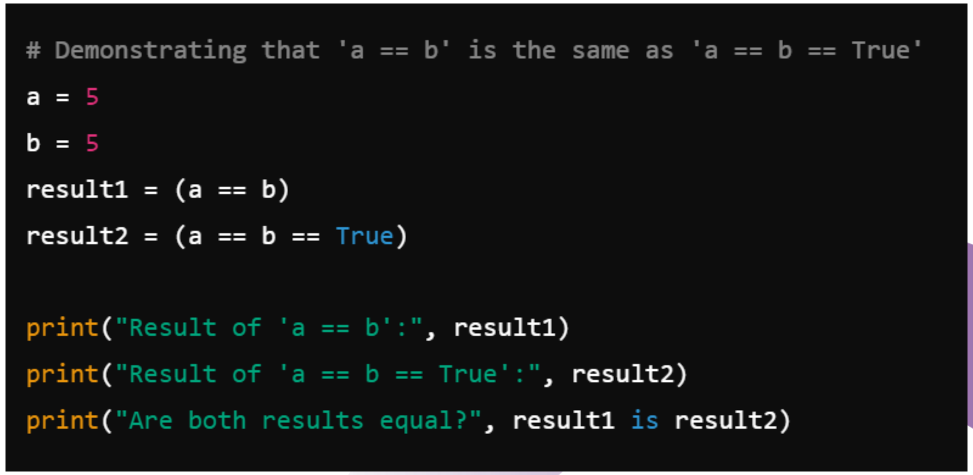
What you need to know…

- *True* and *False* are constants in Python

- Other values equivalent to *True* and *False*:

    - *False*: zero, *None*, empty container or object
    - *True*: non-zero numbers, non-empty objects

- Comparison operators: ==, !=, <, <=, etc.
    - X and Y have same value: X == Y
    - Compare with   X is Y :
        - X and Y are two variables that refer to the identical same object



```python
# Demonstrating that 'a == b' is the same as 'a == b == True'
a = 5
b = 5
result1 = (a == b)
result2 = (a == b == True)

print("Result of 'a == b':", result1)
print("Result of 'a == b == True':", result2)
print("Are both results equal?", result1 is result2)
```

An example of Python true or false

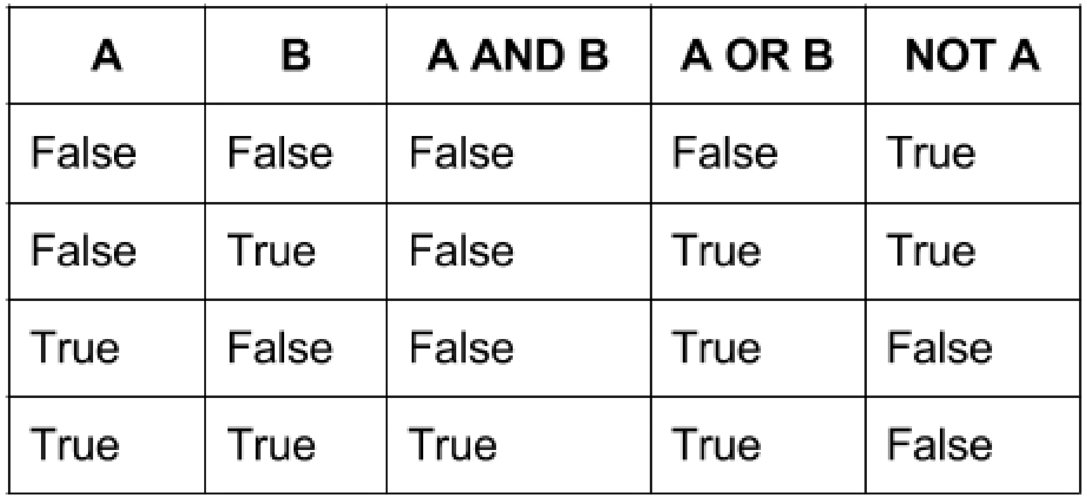
Boolean Logic expressions

You can also combine Boolean expressions…

You can also combine Boolean expressions.

- *True* if a is True and b is True: `a and b`
- *True* if a is True or b is True: `a or b`
- *True* if a is False: `not a`

Use parentheses as needed to disambiguate complex Boolean expressions.



A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Compound expressions in Python

Special properties of and & or

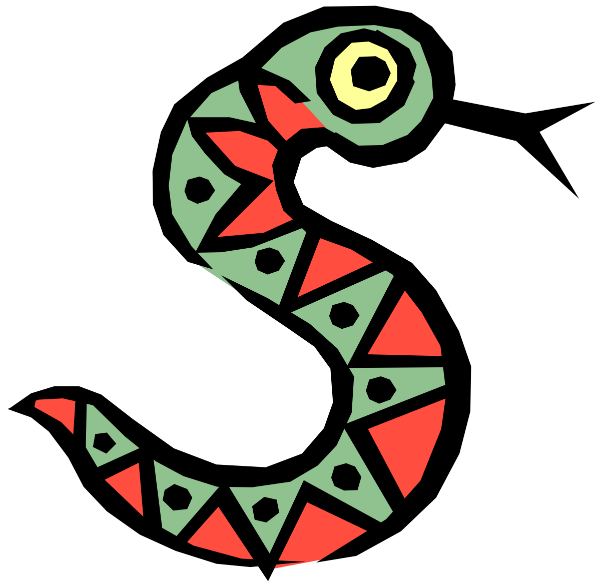
What you need to know…

- X and Y and Z
  - If all are true, returns value of Z
  - Otherwise, returns value of first false sub-expression

- X or Y or Z
  - If all are false, returns value of Z
  - Otherwise, returns value of first true sub-expression

- *And* and *or* use *lazy evaluation*, so no further expressions are evaluated

Control Flow



If statements

What you need to know…

```
if x == 3:
        print “X equals 3.”
elif x == 2:
        print “X equals 2.”
else:
        print “X equals something else.”
print “This is outside the ‘if’.”
```

Note:

Use of indentation for blocks

Colon (:) after boolean expression

Conditional expressions in Python 2.5

What you need to know…

```
x = true_value if condition else false_value
```

Uses lazy evaluation:

- First, `condition` is evaluated
- If *True*, `true_value` is evaluated and returned
- If *False*, `false_value` is evaluated and returned

Standard use:

```
        x = (true_value if condition else false_value)
```

For Loops

What you need to know…

- A for-loop can step through each of the items in a collection type, or any other type of object which is “iterable”

```
for <item> in <collection>:
    <statements>
```

- If <collection> is a list or a tuple, then the loop steps through each element of the sequence
- If <collection> is a string, then the loop steps through each character of the string

```
for someChar in “Hello World”:
    print someChar
```

For Loops

What you need to know…

```
for <item> in <collection>:
    <statements>
```

- <item> can be more than a single variable name
- When the <collection> elements are themselves sequences, then <item> can match the structure of the elements.
- This multiple assignment can make it easier to access the individual parts of each element

```
for (x,y) in [(a,1),(b,2),(c,3),(d,4)]:
    print x
```

For Loops and Dictionaries

What you need to know…

```
>>> ages = { "Sam" : 4, "Mary" : 3, "Bill" : 2 }
>>> ages
{'Bill': 2, 'Mary': 3, 'Sam': 4}
>>> for name in ages.keys():
            print name, ages[name]
Bill 2
Mary 3
Sam 4
>>>
```

For loops & the range() function

What you need to know…

Since a variable often ranges over some sequence of numbers, the *range()* function returns a list of numbers from 0 up to but not including the number we pass to it.

- range(5) returns [0,1,2,3,4]

So we could say:

```
for x in range(5):
    print x
```

(There are more complex forms of *range()* that provide richer functionality…)

For loops vs. list comprehensions

What you need to know…

- Python’s list comprehensions provide a natural idiom that usually requires a for-loop in other programming languages
- As a result, Python code uses many fewer for-loops
- Nevertheless, it’s important to learn about for-loops

Take care! The keywords for and in are also used in the syntax of list comprehensions, but this is a totally different construction.

List comprehensions

What you need to know…

- The syntax of a *list comprehension* is somewhat tricky

```
[x-10 for x in grades if x>0]
```

- Syntax suggests that of a *for*-loop, an *in* operation, or an *if* statement
- All three of these keywords (‘*for*’, ‘*in*’, and ‘*if*’) are also used in the syntax of forms of list comprehensions

[expression for name in list]

List comprehensions

What you need to know…

`[ expression for name in list ]`

- Where expression is some calculation or operation acting upon the variable name.
- For each member of the list, the list comprehension
  1. sets name equal to that member,
  2. calculates a new value using expression,
- It then collects these new values into a list which is the return value of the list comprehension.

```
>>> li = [3, 6, 2, 7]
>>> [elem*2 for elem in li]
[6, 12, 4, 14]
```

while Loops

What you need to know…

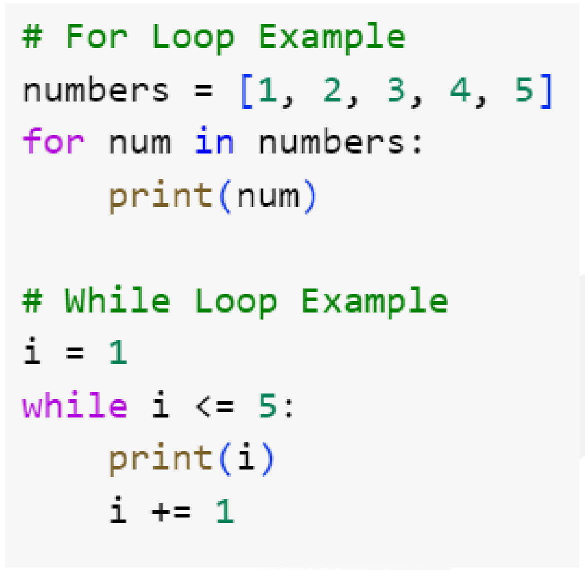
```
>>> x = 3

>>> while x < 5:
                        print x, "still in the loop"
                        x = x + 1

3 still in the loop
4 still in the loop

>>> x = 6
>>> while x < 5:
                        print x, "still in the loop"

>>>
```



```
# For Loop Example
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    print(num)

# While Loop Example
i = 1
while i <= 5:
    print(i)
    i += 1
```

For Loop and While Loop Example

assert

What you need to know…

An *assert* statement will check to make sure that something is true during the course of a program.

If the condition if false, the program stops

(more accurately: the program throws an exception)

```
assert(number_of_players < 5)
```

Google Colab practical

Let’s examine our first notebook file together…

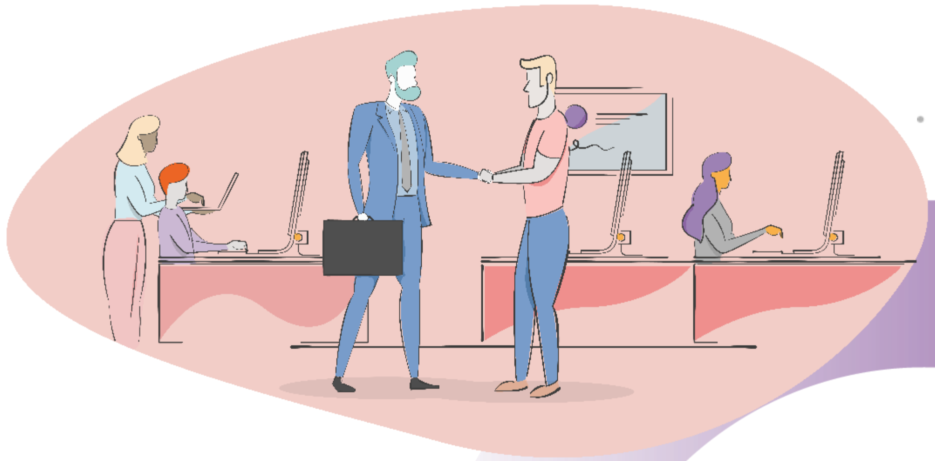
Let’s examine the first Python notebook file together:

- Get used to writing and executing basic code in Colab
- Discuss different variables and data-types
- Examine how to use expressions and operators

This file can be found at the following link: Python notebook link

Google colab

Testing



Testing

What is software testing and why is it performed?

- Absence of bugs and system crashes
- Correspondence between the software and the users’ expectations
- Performance to specified requirements

Defects must be controlled because they lower production speed, increase maintenance costs and can adversely affect business.

Types of software testing

Some options…

Unit testing

Integration testing

Functional testing

Types of software tests

Regression testing

Performance testing

Usability testing

Security testing

Test planning

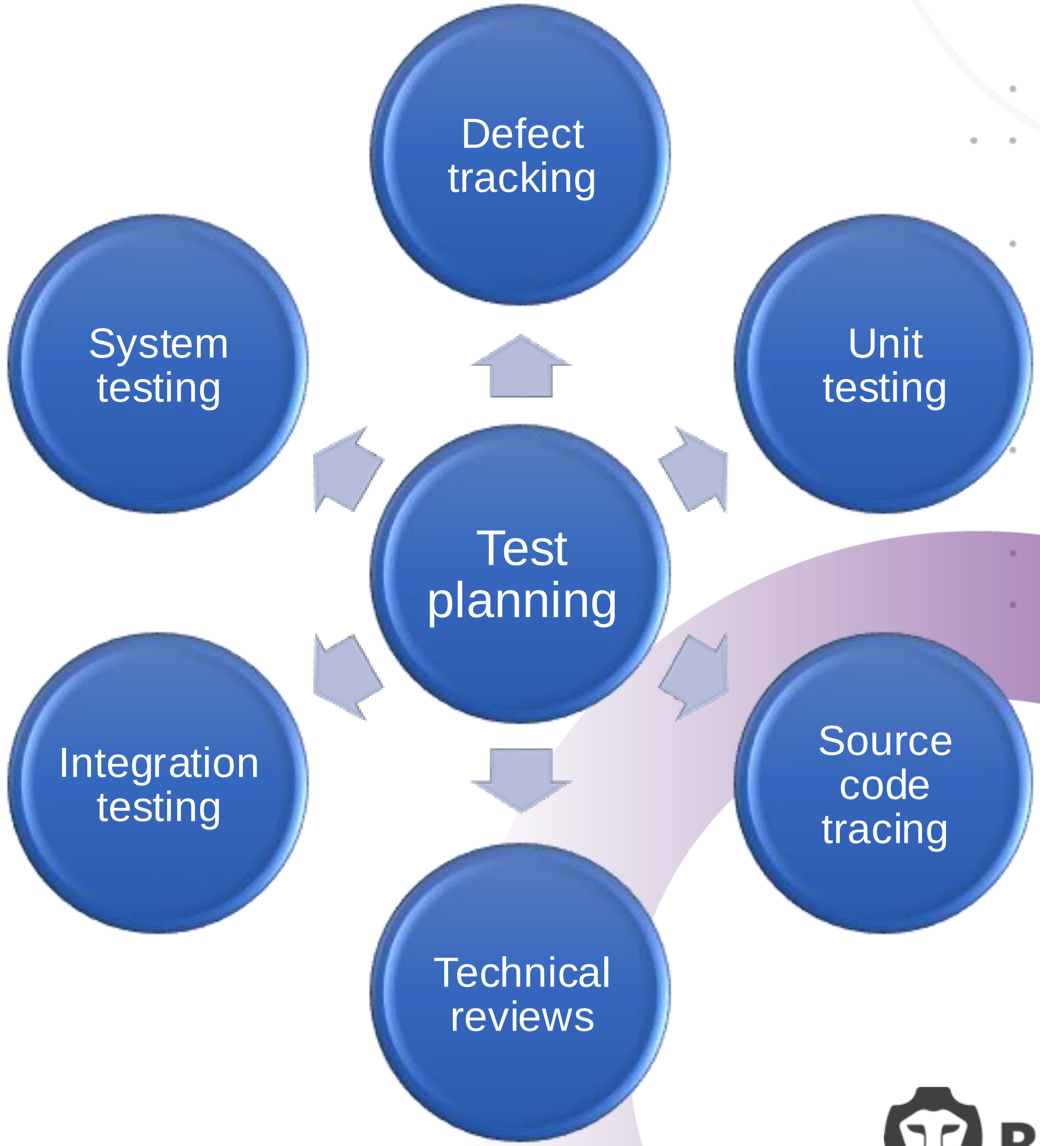
What is it and what does it involve?

1. The plan for quality assurance activities should be in writing

2. Decide if a separate group should perform the quality assurance activities

3. Some elements that should be considered by the plan are:

- defect tracking
- unit testing
- source-code tracking
- technical reviews
- integration testing and system testing



Research task

Group practice

Identify a well-known piece of software eg CRM, Accountancy, Publisher and work out what type of testing you would require to make sure it behaves as required.

Put your answer in the chat

Consider the following questions:

- What can go wrong with this software?
- What should you do about that?

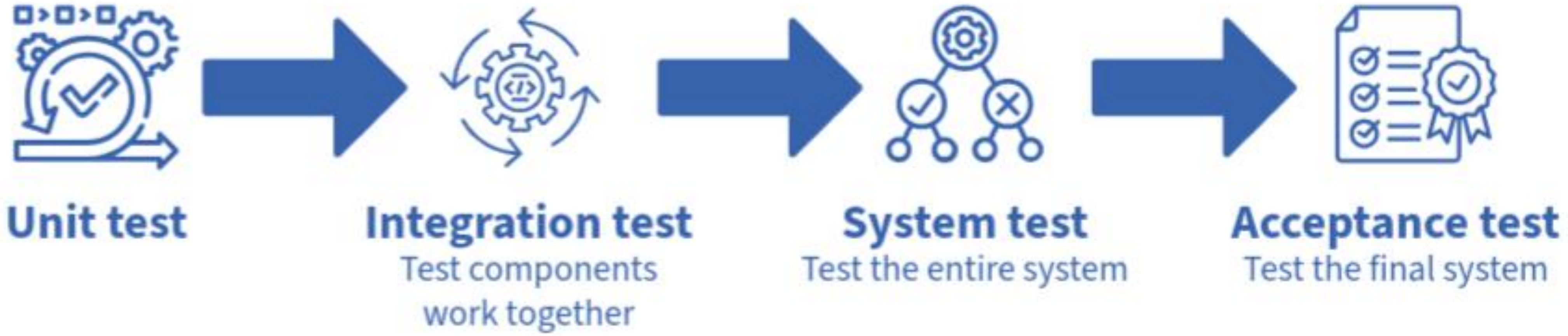
Building Careers Through Education

Everybody Matters. Stronger Together. Student, Learner and Client Centric. Trust and Respect. Embrace Change.

Group practice

BPP

Testing levels



Unit testing

Group discussion

Let’s refer to the Python notebook to understand the basic principles of coded unit testing in python.

Aim to understand how you can use the **unittest** library in Python - to write coded unit tests that test other code!

This file can be found at the following link:

Python notebook link

Building Careers Through Education

Everybody Matters. Stronger Together. Student, Learner and Client Centric. Trust and Respect. Embrace Change.

Group discussion

BPP

Knowledge Check Poll

In Python testing, which of the following statements is true?

a) Unit tests are used to test the system as a whole

b) Test planning is not necessary in Python testing

c) The unittest module in Python is used for constructing and running tests

d) Integration tests are used to test individual components of software

Submit your responses to the chat!

Knowledge Check Poll

In Python testing, which of the following statements is true?

a) Unit tests are used to test the system as a whole

b) Test planning is not necessary in Python testing

c) The unittest module in Python is used for constructing and running tests

d) Integration tests are used to test individual components of software

Feedback: c – The unittest module in Python provides a rich set of tools for constructing and running tests, making it a key part of Python testing.

Submit your responses to the chat!

BPP

Thank you

Do you have any questions, comments, or feedback?