OTHM LEVEL 3 CERTIFICATE IN PYTHON

Qualification Number: 610/5280/2

Specification | **February 2025**

TABLE OF CONTENTS

QUALIFICATION OBJECTIVES

The objective of the OTHM Level 3 Certificate in Python provide the learner with the essential skills necessary to write programs in the Python programming language, for a variety of purposes. This certificate is suitable for those who are new to Python, and have little programming knowledge, providing a comprehensive overview of the basics. This might include students, or professionals wishing to advance their technical skills and career.

The qualification provides learners with an understanding of the syntax, structure, and best programming practices in Python, and takes them from writing basic “Hello World” programs in interactive Jupyter notebooks, to more complex programs that make use of a range of data types, functions, classes and control flow, and are spread over multiple files.

As well as teaching learners how to program in Python, this certificate will give them the opportunity to practise and advance a variety of transferable skills, including problem-solving, logical thinking, and a better understanding of the structure and function of programming languages.

Upon completion of the certificate, learners will have the necessary skills to understand and build a range of essential programs, such as text-based games, simple automation scripts, and programs for data analysis and manipulation. The knowledge and skills they will gain as part of this certificate will provide them with a solid foundation that will also allow them to progress on to more advanced concepts and techniques in the Python programming language.

QUALITY, STANDARDS AND RECOGNITIONS

OTHM Qualifications are approved and regulated by Ofqual (Office of Qualifications and Examinations Regulation). Visit the Register of Regulated Qualifications.

OTHM has progression arrangement with several UK universities that acknowledges the ability of learners after studying Level 3-7 qualifications to be considered for advanced entry into corresponding degree year/top up and Master’s/top-up programmes.

REGULATORY INFORMATION

Qualification Title	OTHM Level 3 Certificate in Python
Ofqual Qualification Number	610/5280/2
Regulation Start Date	31/01/2025
Operational Start Date	03/02/2025
Duration	6 Months
Total Credit Value	32 Credits
Total Qualification Time (TQT)	360 Hours
Guided Learning Hours (GLH)	180 Hours
Sector Subject Area (SSA)	6.1 – Digital technology (practitioners)
Overall Grading Type	Pass / Fail
Assessment Methods	Coursework
Language of Assessment	English

EQUIVALENCES

OTHM Level 3 Certificates represent practical knowledge, skills, capabilities and competences that are assessed in academic terms as being equivalent to GCE AS/A Levels.

QUALIFICATION STRUCTURE

The OTHM Level 3 Certificate in Python qualification consists of 4 mandatory units, 32 credits, 160 hours Total Qualification Time (TQT) and the recommended Guided Learning Hours (GLH) for this qualification is a minimum of 320 hours.

All units are mandatory.

Unit Ref No	Unit Title	Level	Credit	GLH	TQT
D/651/4741	Getting Started with Python	3	8	40	80
F/651/4742	Writing Python Programmes	3	8	40	80
H/651/4743	Data Types and Classes	3	8	40	80
J/651/4744	Building and Running Python Applications	3	8	40	80

DEFINITIONS

Total Qualification Time (TQT) is the number of notional hours which represents an estimate of the total amount of time that could be expected to be required in order for a learner to achieve and demonstrate the achievement of the level of attainment necessary for the award of a qualification.

Total Qualification Time is comprised of the following two elements –

a) the number of hours which an awarding organisation has assigned to a qualification for Guided Learning, and

b) an estimate of the number of hours a Learner will reasonably be likely to spend in preparation, study or any other form of participation in education or training, including assessment, which takes place as directed by – but, unlike Guided Learning, not under the Immediate Guidance or Supervision of – a lecturer, supervisor, tutor or other appropriate provider of education or training.

(Ofqual 15/5775 September 2015)

Guided Learning Hours (GLH) are defined as the hours that a teacher, lecturer or other member of staff is available to provide immediate teaching support or supervision to a learner working towards a qualification.

Credit value is defined as being the number of credits that may be awarded to a learner for the successful achievement of the learning outcomes of a unit. One credit is equal to 10 hours of TQT.

ENTRY REQUIREMENTS

These qualifications are designed for learners who are typically aged 16 and above.

The entry profile for learners is likely to include at least one of the following:

- Relevant Level 2 Diploma qualification or equivalent qualification.
- GCSEs in 5 subjects or equivalent qualifications. Must include Mathematics and Physics at grade C/5 or above.
- Mature learners (over 21) with relevant management experience (learners must check with the delivery centre regarding this experience prior to registering for the programme)

English requirements: If a learner is not from a majority English-speaking country, they must provide evidence of English language competency. For more information visit the English Language Expectations page on the OTHM website.

PROGRESSION

Successful completion of the OTHM Level 3 Certificate in Python provides learners with the opportunity to access a wide range of academic progression, including the OTHM Level 4 or Extended Level 5 Diploma in Cyber Security.

As this qualification is approved and regulated by Ofqual (Office of the Qualifications and Examinations Regulation), learners are eligible for a range of progression routes. For more information visit the University Progressions page on the OTHM website.

DELIVERY OF OTHM QUALIFICATIONS

OTHM do not specify the mode of delivery for its qualifications, therefore OTHM centres are free to deliver this qualification using any mode of delivery that meets the needs of their learners. However, OTHM centres should consider the learners' complete learning experience when designing the delivery of programmes.

It is important that centres develop an effective delivery method to teaching and learning that supports the progression and stretch of learners.

OTHM Centres must ensure that the chosen mode of delivery does not unlawfully or unfairly discriminate, whether directly or indirectly, and that equality of opportunity is promoted. Where it is reasonable and practicable to do so, it will take steps to address identified inequalities or barriers that may arise.

Guided Learning Hours (GLH) which are listed in each unit gives centres the number of hours of teacher-supervised or direct study time likely to be required to teach that unit.

ASSESSMENT AND VERIFICATION

All units within this qualification are assessed and internally quality assured by the centre and externally verified by OTHM. The qualifications are Criteria referenced, based on the achievement of all the specified learning outcomes.

To achieve a ‘pass’ for a unit, learners must provide evidence to demonstrate that they have fulfilled all the learning outcomes and meet the standards specified by all assessment criteria. Judgement that the learners have successfully fulfilled the assessment criteria is made by the assessor.

Specific assessment guidance and relevant marking criteria for each unit are made available in the Assignment Brief document. These are made available to centres immediately after registration of one or more learners.

The assessor should provide an audit trail showing how the judgement of the learners’ overall achievement has been arrived at.

Assessment Tracking and Recording Learner Progress

It is necessary to track and record learner achievement throughout the delivery period of the Diploma and this should not be left until the end of the course.

This will include requalification gular review of learner work through formative and summative assessment and internal quality assurance at planned intervals during the programme:

- before decisions have been made on any unit
- sampling evidence once one or two of the units or assignments are completed

Tracking learner progress, recording the achievement of each learner per criteria on a unit-by-unit basis ensures:

- the assessment evidence is clearly measured against national standards
- learner progress is accurately tracked
- the assessment process can be reliably verified
- evidence is valid, authentic and reliable for the safety of certification
- identification of which assessments are outstanding
- internal verification is timely
- samples for standards verification and other external audits can be made available as required
- up to date, securely stored assessment records help to minimise the risk of assessment malpractice and potential issues; maintaining the integrity of the qualification.

Tutors/Assessors should provide learners with formative and summative feedback to aid development during their studies.

Formative Assessment

Formative assessment is an integral part of the assessment process, involving both the Tutor/Assessor and the learner about their progress during the course of study.
Formative assessment takes place prior to summative assessment and focuses on helping the learner to reflect on their learning and improve their performance and does not confirm achievement of grades at this stage.

The main function of formative assessment is to provide feedback to enable the learner to make improvements to their work. This feedback should be prompt so it has meaning and context for the learner and time must be given following the feedback for actions to be complete. Feedback on formative assessment must be constructive and provide clear guidance and actions for improvement.

All records should be available for auditing purposes, as we may choose to check records of formative assessment as part of our ongoing quality assurance.

Summative Assessment

Summative assessment is used to evaluate learner competence and progression at the end of a unit or component. Summative assessment should take place when the assessor deems that the learner is at a stage where competence can be demonstrated.

Learners should be made aware that summative assessment outcomes are subject to confirmation by the Internal Verifier and External Quality Assurer (EQA) and thus is provisional and can be overridden. Assessors should annotate on the learner work where the evidence supports their decisions against the assessment criteria. Learners will need to be familiar with the assessment and grading criteria so that they can understand the quality of what is required.

Evidence of both formative and summative assessment MUST be made available at the time of external quality assurance – EQA.

RECOGNITION OF PRIOR LEARNING AND ACHIEVEMENT

Recognition of Prior Learning (RPL) is a method of assessment that considers whether learners can demonstrate that they can meet the assessment requirements for a unit through knowledge, understanding or skills they already possess and do not need to develop through a course of learning.

RPL policies and procedures have been developed over time, which has led to the use of a number of terms to describe the process. Among the most common are:

- Accreditation of Prior Learning (APL)
- Accreditation of Prior Experiential Learning (APEL)
- Accreditation of Prior Achievement (APA)
- Accreditation of Prior Learning and Achievement (APLA)

All evidence must be evaluated with reference to the stipulated learning outcomes and assessment criteria against the respective unit(s). The assessor must be satisfied that the evidence produced by the learner meets the assessment standard established by the learning outcome and its related assessment criteria at that particular level.

Most often RPL will be used for units. It is not acceptable to claim for an entire qualification through RPL. Where evidence is assessed to be only sufficient to cover one or more learning outcomes, or to partly meet the need of a learning outcome, then additional assessment methods should be used to generate sufficient evidence to be able to award the learning outcome(s) for the whole unit. This may include a combination of units where applicable.

EQUALITY AND DIVERSITY

OTHM provides equality and diversity training to staff and consultants. This makes clear that staff and consultants must comply with the requirements of the Equality Act 2010, and all other related equality and diversity legislation, in relation to our qualifications.

We develop and revise our qualifications to avoid, where possible, any feature that might disadvantage learners because of their age, disability, gender, pregnancy or maternity, race, religion or belief, and sexual orientation.

If a specific qualification requires a feature that might disadvantage a particular group (e.g. a legal requirement regarding health and safety in the workplace), we will clarify this explicitly in the qualification specification.

UNIT SPECIFICATIONS

GETTING STARTED WITH PYTHON

Unit Reference Number	D/651/4741
Unit Title	Getting Started with Python
Unit Level	3
Number of Credits	8
Total Qualification Time (TQT)	80
Guided Learning Hours (GLH)	40
Mandatory / Optional	Mandatory
Sector Subject Area (SSA)	6.1 – Digital technology (practitioners)
Unit Grading Type	Pass / Fail

Unit Aims

In this unit, learners will get started with installing Python on their computers. By the end of the unit, learners will understand Python fundamentals and be able to use them to write basic programs in Jupyter notebooks, use Python's built-in data types and structures to create variables, apply control flow and functions to solve problems, and handle files and exceptions in Python. Learners will gain an understanding of Python syntax and how it differs from other languages. They will also understand the applications of Python and its strengths for performing different tasks. The Python documentation and PEP style guide will be introduced as a reference point for best coding practices and looking up new information.

Learning Outcomes, Assessment Criteria and Indicative Content

Learning Outcome – The learner will:	Assessment Criteria – The learner can:	Indicative Content
1. Understand what Python is and what it is used for.	1.1 Describe Python and its historical origin. 1.2 Explain the differences between Python and other common programming languages. 1.3 Analyse how Python can be useful for a variety of programming tasks.	**Python – History and Development** ● What is Python? Definition as an interpreted programming language. ● History of Python: Guido van Rossum's invention of Python. Python 2 vs. Python 3. ● Python as an interpreted language, vs. compiled languages such as C++ and Java. Python compiles source code to bytecode,

		enabling platform-independent execution and eliminating the need for re-compilation every time a program is run. ● Advantages and disadvantages of interpreted vs. compiled programming languages from perspectives including performance, simplicity, debugging difficulty, and runtime errors. **Applications of Python** ● Python for scientific data processing and analysis of numerical data. ● Python for developing web applications and performing web scraping. ● Python for searching and processing textual data. ● Python for artificial intelligence (particularly deep learning) applications e.g. Tensorflow and Pytorch. ● Comparison with other programming languages. Discuss Python's strengths and weaknesses in each application context. What makes the Python language suitable? *In each case, the use of Python in the specified context could be discussed and learners could be invited to think about what features Python might need in order to make it a suitable language for the given application.*
2. Be able to run basic Python programs.	2.1 Describe basic Python syntax and structure. 2.2 Explain how Python code is executed. 2.3 Analyse the differences between dynamic and static typing and explain the advantages and disadvantages of each. 2.4 Design and run basic Python programs.	**Python “Hello World”** ● Download and install Python via the official website or the command line. ● Installing and using pip – the Python package manager. ● Installing Jupyter notebook dependencies using pip, i.e. the ‘jupyterlab’ or ‘notebook’ packages.

	2.5 Use appropriate methods to install Python, pip, and Jupyter. 2.6 Use Jupyter notebooks and the command line to run Python code.	● Using the command line to access the Python interpreter. Python “Hello World” i.e. printing a basic message using the “print” function. ● Receiving input from the user using “input”. **Python Basic Syntax and Structure** ● Whitespace and its importance in Python. Spacing consistency, tabs vs. spaces. Comparison to other programming languages where whitespace is not of the same significance (e.g. C++) – discussion of the benefits and disadvantages of this syntax, e.g. from a readability perspective. ● Variables and dynamic (duck) typing (runtime vs. compile time). Dynamic typing allows for change of variable types at runtime. Local and global variable scope in Python. Use of brackets and indentation to define scope. *Note – pip and Python packages can be introduced here, but there is no need to go into too much detail since package management and the installation of packages will be covered as part of Learning Objective 4.*
3. Understand a range of basic data types in Python.	3.1 Describe the major built-in data types in Python. 3.2 Explain the difference between an int and a float. 3.3 Analyse the difference between equality of value and reference to the same object. 3.4 Use logical operators to perform comparative binary operations in Python. 3.5 Use mathematical operators to perform calculations, considering order of precedence.	**Python Data Types** ● Integers, floats, strings, bools. ● The None type, which represents a null or undefined value. ● Introduction to iterables and the range function. ● Using ‘type()’ and ‘id()’ to identify objects and their types. Comparing object equality: using ‘id’ to check for object equality and using the == operator to check for equality of value.

	3.6 Use functions to convert between basic data types.	**Manipulating Data Types** ● Basic mathematical operations in Python: addition, subtraction, multiplication, division, exponentiation. Order of precedence of operations. Using brackets to alter the order of precedence. ● Logical operators. Using operators to compare and evaluate the truth of mathematical statements using bools and integers: and, or, not, equals (==), not equals (!=). Python's use of short-circuit evaluation of arguments. ● Data type conversion: using the str, float, and int functions.
4. Be able to use the documentation and style guides to program more effectively.	4.1 Describe the Zen of Python. 4.2 Explain how to use the Python documentation to look up variables, functions and classes. 4.3 Use the PEP style guide to understand best stylistic practices.	**Programming with Style** ● Using the PEP style guide to understand best practices for styling and organising your code. ● The Zen of Python. **Python Documentation** ● Overview of Python's official documentation. ● Using the documentation to understand function inputs and outputs. ● Using the documentation to understand class attributes and methods. *N.B. The documentation is introduced here. Learners will be introduced to the idea of object-oriented programming and using functions in the Writing Programs in Python unit, for which it will be a useful reference.*

Assessment

To achieve a 'pass' for this unit, learners must provide evidence to demonstrate that they have fulfilled all the learning outcomes and meet the standards specified by all assessment criteria.

Learning Outcomes to be met	Assessment Criteria to be covered	Assessment type	Word count (approx. length)
All	All ACs under	Coursework	2000 words

Indicative Reading List

Gaddis T. (2020) *Starting Out with Python (3rd edition)* ISBN 978-0133582734

Matthes E. (2023) *Python Crash Course (3rd edition)* ISBN 978-1718502703

Shaw Z. (2024) *Learn Python the Hard Way: A Deceptively Simple Introduction to the Terrifyingly Beautiful World of Computers and Data Science* ISBN 978-0138270575

The PEP Editors. (2000) *PEP 0 – Index of Python Enhancement Proposals (PEPs)* https://peps.python.org/pep-0000/

Tim Peters. (2004) *PEP 20 - The Zen of Python* https://peps.python.org/pep-0020/

Additional Resources

Google Colab (Online Jupyter notebooks) https://colab.research.google.com/

Python 3 Documentation https://docs.python.org/3/

WRITING PYTHON PROGRAMMES

Unit Reference Number	F/651/4742
Unit Title	Writing Python Programs
Unit Level	3
Number of Credits	8
Total Qualification Time (TQT)	80
Guided Learning Hours (GLH)	40
Mandatory / Optional	Mandatory
Sector Subject Area (SSA)	6.1 – Digital technology (practitioners)
Unit Grading Type	Pass / Fail

Unit Aims

In this unit, learners will be introduced to techniques for automating code and controlling program flow. They will learn about functions and their importance, how to perform repeated execution using 'for' loops and 'while' loops, and how to conditionally execute code using if statements. They will also be introduced to imports and will learn how to import external code, which can be used to extend and enhance their own programs. They will be taught how to interface with files, using the open(…) function. Finally, they will learn about exceptions and how 'try, except, else, finally' clauses can be used to handle unexpected events and runtime errors.

Learning Outcomes, Assessment Criteria and Indicative Content

Learning Outcome – The learner will:	Assessment Criteria – The learner can:	Indicative Content
1. Be able to write functions in Python.	1.1 Describe how to write functions in Python. 1.2 Explain the need for functions. 1.3 Explain how recursive functions can help us solve problems of particular structure. 1.4 Write recursive functions to solve problems of a special structure. 1.5 Analyse potential problems with using recursive functions.	**Python Functions** ● What is a function? A piece of reusable code that takes zero or more parameters and may return a value. ● Function syntax in Python using 'def'. (def function_name(parameters):). ● Function inputs and outputs: parameters and return values, default arguments vs. keyword arguments, packing and unpacking arguments.

		● The need for and benefit of functions: reusable code to reduce redundancy and improve automation, as well as introducing the notion of hierarchical programs with functions of functions for abstraction. **Recursive Functions** ● Recursive functions as functions that call themselves (self-referencing). ● Recursive cases vs. base case. ● Use cases for recursive functions, e.g. factorial calculation, Fibonacci sequences. ● Exceeding maximum recursion depth (e.g. due to code which hasn't been properly written, due to running recursive functions on inputs that require many more recursions). Result: growing memory usage due to growing stack frames, leading to program failure when maximum recursion depth is exceeded (RecursionError).
2. Understand key concepts in controlling the flow of Python programs.	2.1 Describe how if statements are used to manage conditional execution. 2.2 Describe how 'for' loops can be used to run code multiple times in succession. 2.3 Explain how iterables are used with 'for' loops. 2.4 Explain the function of the 'break' and 'continue' statements in for loops and while loops. 2.5 Analyse the different use cases of 'for' loops and while loops.	**Conditional Execution** ● if statements and structure: 'if', 'elif', 'else'. ● Using logical operators ('and', 'or', 'not') with if statements to combine or negate conditions. ● Inline if statements for concise programming (e.g. x if value == True else y). **Iteration** ● For loops (for i in range(n): func()) and while loops (while value == True: func()). Function of each and key distinctions. Conditional looping in while loops, iterable looping in for loops. Syntax of for loops and while loops. ● Iterables for use in for loops, and the range(…) function for returning a numeric iterator.

		● 'break' and 'continue' statements for interrupting loop execution. ● Use cases of for loops and while loops, emphasising their distinction and purpose. For example, for loops for applying similar operations to multiple files; while loops for repeatedly running file operations until a condition is satisfied.
3. Be able to import and use elements from external modules.	3.1 Describe how packages can be installed using pip. 3.2 Explain how to look up and install new packages using the Python Package Index. 3.3 Use import statements to import modules.	**Importing Modules** ● Extending code functionality using external modules. ● Exploration and understanding of the existence of a range of useful and essential built-in modules: e.g. math, os, sys, shutil, json, datetime, time. ● Examples of useful functionality, such as time.sleep(), os.path.* methods, and sys.argv for command-line arguments. ● Different methods for importing ('import module' vs. 'from module import x' vs. wildcard import 'from module import *'). Reasons for avoiding wildcard imports. *There are many built-in modules in Python that can be useful for different purposes. The modules explored can be tailored to the learners' interests, but it is recommended that certain universally useful modules (math, os, sys, time etc.) should always be covered.* **Installing New Packages** ● Installing new packages using pip. ● Using pip to list installed packages, to upgrade packages, and to uninstall packages. ● Use of the Python Package Index (PyPI) to

		search for and understand the function of Python packages.
4. Be able to read from and write to files.	4.1 Describe the different modes with which a file can be opened. 4.2 Explain how to read from and write to files. 4.3 Analyse the advantages of context management. 4.4 Use 'with' statements to open file objects.	**Reading and Writing to Files** ● open(…) for opening files in a particular mode. file.close() for manually closing the file after performing the necessary operations. ● Basic file modes: Read/Write/Write and create if not exists (r, w, w+). ● Binary read/write mode and use cases (rb, wb etc.). **Context Managers** ● Using open(…) in a 'with' statement to get a context manager. ● Allows for automatic setup and teardown operations. ● Advantages of using 'with' statements for manipulating files include better safety for resource management and avoidance of resource leaks (e.g., automatically closing a file after we have finished performing operations on it).
5. Understand exceptions in Python and how they can be handled.	5.1 Describe how exceptions are handled in Python. 5.2 Describe a range of different types of common exceptions. 5.3 Explain the difference between exceptions and warnings. 5.4 Use 'try, except, else, finally' statements for manual handling of exceptions. 5.5 Analyse how exceptions can help point to problems in code.	**Python Exceptions** ● Definition of an exception in Python, how and why exceptions are raised. Exceptions are events that disrupt normal program execution, often due to errors such as invalid syntax or logic issues. ● Common built-in exceptions: SyntaxError, ZeroDivisionError, IOError, and FileNotFoundError. ● Warnings (unlike exceptions, do not interrupt program flow) for alerting users to non-critical

		potential problems or mistakes that should be amended. Common warnings: e.g. DeprecationWarning, ImportWarning, and RuntimeWarning. ● Using “try, except, else, finally” clauses to manage exceptions, potential use cases for this, and the benefits of doing so (e.g. to handle expected problematic cases and avoid programs terminating).

Assessment

To achieve a ‘pass’ for this unit, learners must provide evidence to demonstrate that they have fulfilled all the learning outcomes and meet the standards specified by all assessment criteria.

Learning Outcomes to be met	Assessment Criteria to be covered	Assessment type	Word count (approx. length)
LO1	All ACs under LO1	Coursework	1000 words
LO2-LO5	All ACs under LO2-LO5	Coursework	1500 words

Indicative Reading List

Downey A. (2024) *Think Python: How to Think Like a Computer Scientist* ISBN 978-1098155438

Romano F. & Kruger H. (2021) *Learn Python Programming: An in-depth introduction to the fundamentals of Python (3rd edition)* ISBN 978-1801815093

Sweigart A. (2019) *Automate the Boring Stuff with Python (2nd edition)* ISBN 978-1593279929

Additional Resources

Built-In Exceptions (from Python Documentation) https://docs.python.org/3/library/exceptions.html

Python Package Index (PyPI) https://pypi.org/

DATA TYPES AND CLASSES

Unit Reference Number	H/651/4743
Unit Title	Data Types and Classes
Unit Level	3
Number of Credits	8
Total Qualification Time (TQT)	80
Guided Learning Hours (GLH)	40
Mandatory / Optional	Mandatory
Sector Subject Area (SSA)	6.1 – Digital technology (practitioners)
Unit Grading Type	Pass / Fail

Unit Aims

In this unit, collection data types are explored. Learners will gain an understanding of the use cases for different collection data structures, including lists, tuples, dictionaries, and sets. Learners will also be introduced to a range of essential and more advanced techniques for searching, filtering, and manipulating strings. The principles of object-oriented programming in Python will be introduced and learners will be taught how to build and modify classes and their attributes and methods. Instance and class methods and attributes will be distinguished, and learners will gain an appreciation for class inheritance and its mechanics.

Learning Outcomes, Assessment Criteria and Indicative Content

Learning Outcome – The learner will:	Assessment Criteria – The learner can:	Indicative Content
1. Be able to use a wide range of string operations.	1.1 Describe strings as iterables formed from individual characters. 1.2 Use common operations to manipulate strings to remove and add characters. 1.3 Use a variety of techniques to search for data in strings. 1.4 Explain how string formatting techniques can be used to format variables in strings. 1.5 Use regular expressions and string	**Manipulating strings** ● String addition and concatenation. Accessing individual characters in a string by iterating over it, or by using indexing. ● Splitting strings into lists using .split() and use cases (e.g. reading csv data). ● String formatting using .format() and f-strings. Formatting techniques for floats (e.g. rounding to a certain number of decimal places) and

	replacement to perform substitution within strings.	integers (e.g. zero-padding). ● Removing leading and trailing whitespace and characters using .strip(). ● Replacing characters in a string using .replace(). Regular expressions (regex) in Python for searching and replacing in strings (using e.g. re.search, re.match, and re.sub). ● Using file encodings (e.g. UTF-8) for interpreting characters from binary data in files that are being read and written to.
2. Be able to create and use collection data structures in Python.	2.1 Describe a range of Python collections. 2.2 Use operations to build and manipulate collections. 2.3 Analyse the different use cases of lists, tuples, and dictionaries. 2.4 Use methods to create iterables from dictionaries.	**Python Collections** ● Lists (ordered and mutable), tuples (ordered and immutable), dictionaries (key-value pairs), and sets (unordered and unique elements). ● Use cases and comparisons from the perspectives of ordering, mutability, and access methods (key vs. index). **Collection Operations** ● Checking membership using the 'in' keyword. ● Building lists and dictionaries using list/dictionary comprehension. ● Adding, modifying, and removing items from collections. For adding: 'append' for lists, 'add' for sets, and a new key for dictionaries. ● .items(), .keys(), and .values() methods for creating iterables from dictionaries.
3. Understand the principles of classes and objects.	3.1 Use class constructor syntax and methods to define and initialise classes. 3.2 Explain a range of special built-in methods for classes. 3.3 Analyse the difference between class and instance methods and attributes.	**Creating and Modifying Classes and Instances** ● Class constructor method __init__, defining instance variables using reference to 'self'. The __new__ class constructor and why it is rarely modified. ● Class and instance methods. How they are

	3.4 Use class inheritance to build child classes. 3.5 Use tests to determine class membership.	defined (first argument is 'cls' rather than 'self'). What their different purposes are. ● Class variables and instance variables. ● Modifying object attributes and deleting objects. ● Special built-in methods for iterables (__iter__ and __next__), arithmetic (__add__, __sub__ etc.), string representation (__repr__), length (__len__), and containers (__getitem__, __setitem__, __contains__). ● Common pitfalls e.g. missing 'self' in instance methods, overwriting built-in methods unintentionally. **Class Inheritance** ● Parent classes and inheritance. Overwriting parent class methods and attributes. ● Multiple inheritance. ● The 'object' class as the base class for all classes in Python (new-style). ● Accessing parent class methods and attributes. The super(…) method. ● Testing class membership using 'isinstance' and 'type', and checking if a class belongs to a parent class.

Assessment

To achieve a 'pass' for this unit, learners must provide evidence to demonstrate that they have fulfilled all the learning outcomes and meet the standards specified by all assessment criteria.

Learning Outcomes to be met	Assessment Criteria to be covered	Assessment type	Word count (approx. length)

All	All ACs under	Coursework	2000 words

Indicative Reading List

Canning J., Broder A., Lafore R. (2022) *Data Structures & Algorithms in Python* ISBN 978-0134855684

Kalb I. (2022) *Object-Oriented Python: Master OOP by Building Games and GUIs* ISBN 978-1718502062

Weisfield M. A. (2013) *The Object-Oriented Thought Process (4th edition)* ISBN 978-0321861276

Wengrow J. (2020) *A Common–Sense Guide to Data Structures and Algorithms (2nd edition)* ISBN 978-1680507225

Zelle J. M. (2024) *Python Programming: An Introduction to Computer Science (4th edition)* ISBN 978-1590282977

Additional Resources

W3 Schools Python Classes and Objects https://www.w3schools.com/python/python_classes.asp

BUILDING AND RUNNING PYTHON APPLICATIONS

Unit Reference Number	J/651/4744
Unit Title	Building and Running Python Applications
Unit Level	3
Number of Credits	8
Total Qualification Time (TQT)	80
Guided Learning Hours (GLH)	40
Mandatory / Optional	Mandatory
Sector Subject Area (SSA)	6.1 – Digital technology (practitioners)
Unit Grading Type	Pass / Fail

Unit Aims

In this final unit, learners will bring together the skills they have acquired previously and will extend their Python programming abilities to enable them to write and organise more complex programs that rely on multiple modules. By the end of the unit, learners will understand Python's role in application development, be able to build modular, maintainable, and efficient Python applications and utilise tools and techniques for debugging, testing, and packaging. To equip them to better handle the growing complexity of the programs, a number of new concepts will be introduced: environment managers for maintaining consistent and repeatable dependencies; the use of Integrated Development Environments for more streamlined development; and finally, enhanced debugging skills including interactive debugging using breakpoints and built-in and external debugging modules.

Learning Outcomes, Assessment Criteria and Indicative Content

Learning Outcome – The learner will:	Assessment Criteria – The learner can:	Indicative Content
1. Understand how to use environment managers to manage Python installations.	1.1 Describe methods for managing Python installations. 1.2 Use Anaconda and Venv to create isolated Python environments. 1.3 Explain the differences between Venv and Anaconda for Python environment management. 1.4 Analyse the benefits of using environment	**Environment Managers** ● Introduction to Anaconda and Venv for Python environment management. ● Venv: Built into Python. Anaconda: comes as its own Python distribution – a cross-platform package and environment manager, including non-Python dependencies. Anaconda includes

	managers.	‘conda’ as an environment and package manager distinct from ‘pip’. ‘Venv’ creates isolated environments in the same Python installation, whereas Anaconda installs separate Python binaries. ● Creating Python environments using environment managers; specifying package version numbers; installing, uninstalling, and modifying packages; understanding the purpose of base environments. **Benefits of Environment Managers** ● Isolating individual installations to mitigate problems with individual environments. ● Enabling the concurrent use of different versions of the same package. ● Reproducible behaviour for programs through specifying exact versions of dependencies.
2. Understand how to use Integrated Development Environments (IDEs) to build larger scale projects.	2.1 Describe different IDEs suited for Python development. 2.2 Explain the key features of IDEs that enhance programming efficiency. 2.3 Use extensions to customise IDEs and integrate additional tools. 2.4 Use IDEs to manage more complex projects. 2.5 Analyse a range of extensions and tools that help to make an IDE more useful.	**Integrated Development Environments (IDEs)** ● Purpose and use of IDEs: improving and enhancing the development experience by providing a platform for the management of complex projects that rely on multiple files. ● Features of IDEs that make them useful for programmers: syntax highlighting/colour coding, automatic syntax correction, line numbering, tools to test and debug code including runtime environments (virtual machines), variable tracing, ability to install extensions and to customise the development environment. ● Popular IDEs for Python development: e.g. Visual Studio Code and Pycharm. JupyterLab as an example of an IDE that learners may have already used.

		Using IDEs Effectively ● Opening folders and workspaces. ● Working on multiple projects simultaneously. ● Installing extensions to customise the behaviour of your IDE, e.g. for modified tooltips, for working on files on remote servers. ● Using Large Language Models (e.g. CoPilot) to assist with and accelerate programming (e.g. for code completion, bug identification, and documentation generation).
3. Be able to create and manage custom Python modules and packages.	3.1 Describe the necessary directory structure for packages to be recognised. 3.2 Explain how Python packages are built and installed. 3.3 Use tools to build and install custom Python packages. 3.4 Use tools to upload custom packages to the Python Package Index.	**Creating Custom Modules and Packages** ● The structure of Python packages: the role of __init__.py, nested directories, and naming conventions. ● Writing modules that can be both run independently or used as libraries, and purpose of the 'if __name__==’__main__’ clause. ● Handling command line arguments using e.g. the argparse module. ● Local imports and imports from locations in the system path. ● Building and installing custom Python packages using e.g. 'setuptools'. The mechanics of compiling and building Python sources into wheels and installing them. Use of the setup.py file: guidance on specifying package properties including naming conventions, authorship, and dependencies. ● Uploading custom packages to PyPI using twine. The .pypirc file.

4. Be able to use debugging tools to identify and fix problems in Python programs.	4.1 Describe a range of techniques for debugging Python code. 4.2 Explain how IDE features aid in identifying and resolving problems. 4.3 Use debugging tools to interactively step through and analyse code to identify problems. 4.4 Analyse the importance of unit tests for ensuring expected program functionality.	**Python Debugging** ● Essential debugging techniques: using print statements to display intermediate outputs, using 'try, except' statements to catch exceptions. ● Using static code analysis tools (e.g. pylint, pylance, and IntelliSense) in IDEs to give suggestions and identify syntactic and dependency-related problems with programs. ● Using graphical breakpoints and variable tracing in IDEs to help step through code and isolate issues. ● Using pdb and ipdb to debug Python programs. Importing the debugging modules and setting interactive breakpoints using set_trace(). ipdb's more user-friendly features, such as tab completion. ● Using commands while debugging: use of the 'next' and 'continue' commands to step through the program, the 'list' command to show context, the 'print' command to print variables etc. ● Writing unit tests (e.g. using pytest) for continuous monitoring of program functionality and edge case handling. Importance for continuous integration (CI) workflows.

Assessment

To achieve a 'pass' for this unit, learners must provide evidence to demonstrate that they have fulfilled all the learning outcomes and meet the standards specified by all assessment criteria.

Learning Outcomes to be met	Assessment Criteria to be covered	Assessment type	Word count (approx. length)
All	All ACs under LO1-LO4	Coursework	2000 words

Indicative Reading List

Agans D. (2006) *Debugging: The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems* ISBN 978-0814474570

Beazley D. (2021) *Python Distilled* ISBN 978-0134173276

Lutz M. (2011) *Programming Python (4th edition)* ISBN 978-0596158101

Slatkin B. (2020) *Effective Python: 90 Specific Ways to Write Better Python* ISBN 978-0134853987

Additional Resources

Getting started with conda (Anaconda user guide) https://docs.conda.io/projects/conda/en/latest/user-guide/getting-started.html

Getting Started with Python in VS Code https://code.visualstudio.com/docs/python/python-tutorial

pdb – The Python Debugger (Documentation) https://docs.python.org/3/library/pdb.html

Pycharm Quick Start Guide https://www.jetbrains.com/help/pycharm/quick-start-guide.html

Python Packaging User Guide https://packaging.python.org/en/latest/

venv – Creation of virtual environments (Documentation) https://docs.python.org/3/library/venv.html

IMPORTANT NOTE

Whilst we make every effort to keep the information contained in programme specification up to date, some changes to procedures, regulations, fees matter, timetables, etc may occur during the course of your studies. You should, therefore, recognise that this booklet serves only as a useful guide to your learning experience.

For updated information please visit our website www.othm.org.uk.