HyperionDev

Curriculum title	Python Programmer
Curriculum code	900221-000-00-00
Module code	[module code]
NQF level	4
Credit(s)	40
Quality assurance functionary	QCTO - Quality Council for Trades and Occupations
Originator	MICT SETA
Qualification type	Skills Programme

Python Programmer

Learner Practical Skills Workbook

Name	
Contact Address	
Telephone (H)	
Telephone (W)	
Cellular	
Email	

Table of contents

Table of contents **2**
Practical Skills Module Specifications **4**
Provider Programmme Accreditation Criteria **5**
Physical Requirements: 5
Human Resource Requirements: 5
Legal Requirements: 5
Exemptions: 5
Purpose of Practical Skills: Module 2 **6**
Practical Skills Module 2: Topic 1 7
Practical task **8**
Practical Skills Module 2: Topic 2 11
Practical task **12**
Step 2: Assign a float value to represent price. 12
Step 3: Use the type() function to check data types. 12
Step 4: Assign a large number using underscores. 13
Step 5: Perform arithmetic operations. 13
Practical Skills Module 2: Topic 3 15
Practical task **16**
Step 1: Assign a float to represent a grade. 16
Step 2: Round the float to the nearest integer. 16
Step 3: Convert a list of strings to floats. 16
Step 4: Compare floats. 16
Step 5: Multiply floats. 17
Step 6: Format a float to two decimal places. 17
Step 7: Create a list of floats. 17
Step 8: Add floats together. 17
Step 9: Convert a float to an integer. 17
Step 10: Use sum() on a list of floats. 17
Step 11: Convert an integer to a float. 18
Practical Skills Module 2: Topic 4 19
Practical task **20**
Step 1: Assign float values to customer balances. 20
Step 2: Set the interest rate. 20
Step 3: Calculate monthly interest. 20
Step 4: Update the account balance. 21
Step 5: Format the result for display. 21
Practical Skills Module 2: Topic 5 23
Practical task **24**
Step 1: Declare a Boolean to represent membership status 24
Step 2: Use True or False to indicate overdue books 24

Step 3: Compare values to validate a membership ID 24
Step 4: Evaluate multiple Boolean conditions 25
Step 5: Create a function that returns a Boolean 25
Practical Skills Module 2: Topic 6 27
Practical task **28**
Step 1: Create a list of student grades. 28
Step 2: Access individual elements by index. 28
Step 3: Use list slicing. 28
Step 4: Add and update elements. 29
Step 5: Delete a specific element. 29
Step 6: Calculate the average of the grades. 29
Step 7: Remove elements using different methods. 29
Step 8: Concatenate two lists. 29
Practical Skills Module 2: Topic 7 31
Practical task **32**
Step 1: Pack and unpack tuples. 32
Step 2: Compare tuples to sort by stock quantity. 32
Step 3: Use tuples as keys in a dictionary. 33
Step 4: Delete tuples from list or dictionary. 33
Step 5: Slice tuples to extract specific details. 33
Practical Skills Module 2: Topic 8 35
Practical task **36**
Step 1: Create a dictionary of student data. 36
Step 2: Access a student's details by ID. 36
Step 3: Add a new student and remove one. 37
Step 4: Check for a student ID before adding/modifying. 37
Step 5: Add a new key-value pair to a student record. 37
Step 6: Update grade and add a subject. 37
Step 7: Merge in new student records. 37
Practical Skills Module 2: Topic 9 39
Practical task **40**
Step 1: Create sets for available book genres. 40
Step 2: Add and remove elements from sets. 40
Step 3: Sort a list of books in a genre. 41
Step 4: Count the number of genres in each set. 41
Step 5: Merge fiction and non-fiction genres. 41
Practical Skills Module 2: Topic 10 43
Practical task **44**
Step 1: Create arrays (lists) for inventory categories. 44
Step 3: Insert a new product. 44
Step 4: Modify an existing product. 44
Step 6: Pop an element from an array. 45
Step 7: Delete an element at a specific position. 45

Step 8: Search and get the index of a value. 45
Step 9: Reverse the array. 45
Step 10: Count how many times a value appears. 45
Step 11: Traverse the array. 45
Share your thoughts **48**

Practical Skills Module Specifications

List of Practical Skill Module Specifications

900221-000-00-PM-02	Troubleshoot computer and network faults	L4	Cr8

You will demonstrate your competence in the knowledge topics and achievement of the assessment criteria through a process of continuous assessment (evaluation of your progress throughout the skills programme). These assessments involve interpreting evidence of your ability to perform specific tasks..

The practical tasks in this workbook must be submitted to the facilitator when you have completed them. They will be added to your portfolio of evidence (PoE), which will be signed by your facilitator as evidence that you have successfully performed these tasks.

Pay close attention to your facilitator's instructions and ensure you complete the activities within the given time.

Take note

This module has a total of 310 marks available. To meet the passing requirements, you will need to achieve a minimum of 233 marks, which represents 75% of the total marks. Achieving this threshold will ensure that you have met the necessary standards for this module.

Provider Programmme Accreditation Criteria

Physical Requirements:

- Valid licensed software and application, including OS
- Internet connection and hardware availability
- Examples and information specified in the scope statement and all the case studies, scenarios and access to hardware and software implied in the scope statements of the modules
- Remote learners: Provider must provide business IT simulation system (e.g. invoice processing)

Human Resource Requirements:

- Qualification of lecturer (SME):
    - NQF 5 industry recognised qualification with 1 year relevant experience
- Assessors and moderators: accredited by the MICT SETA

Legal Requirements:

- Legal (product) licences to use the software for learning and training
- OHS compliance certificate
- Ethical clearance (where necessary)

Exemptions:

- None, but the module can be achieved through RPL

Purpose of Practical Skills: Module 2

Programming Basics for Beginners, NQF 4

The focus of the learning in this module is on providing the learner with an opportunity to apply built-in datatypes appropriately when programming with Python.

The learner will be required to:

- PM-02-PS01: Create, format, modify and delete strings in Python
- PM-02-PS02: Use numbers in Python
- PM-02-PS03: Create a float in Python
- PM-02-PS04: Use double data types in Python
- PM-02-PS05: Declare a Boolean – true or false
- PM-02-PS06: Use Python lists to store multiple items in a single variable
- PM-02-PS07: Use Python tuple to store multiple items in a single variable
- PM-02-PS08: Use dictionaries in Python to store data values
- PM-02-PS09: Use Python sets to store multiple items in a single variable
- PM-02-PS10: Create Arrays in Python

Practical Skills Module 2: Topic 1

Topic Code	PM-02-PS01:
Topic	Create, format, modify and delete strings in Python

Scope of Practical Skill

Given an applicable instruction and access to a learning platform, the learner must be able to:

- PA0101 Create, format, modify and delete strings in Python
- PA0102 Create strings in Python with single quotes, double quotes, or triple quotes

Applied Knowledge

- AK0101 Concept, definition and functions of Python programming functionalities

Internal Assessment Criteria

- IAC0101 Expected results with Python strings are achieved

Resources:

Knowledge	Information from Python Programmer Curriculum
National Curriculum Framework	900221-000-00-PM-02

Practical task

Follow the facilitator's instructions to complete the following activities:

Scenario: Building a Simple "Course Enrolment System"

In this scenario, you are creating a Python program to manage course enrolment for students. The program will involve manipulating strings to store and display student names, course details, and other related information.

Step 1: Create, format, modify and delete strings in Python.

Create a file called `course_enrolment.py`. Inside the file, create two string variables:

- One to store a student's **full name** (a name of your choice, e.g Alice)
- One to store a **course name** (a course name of your choice, e.g Introduction to Python)

Combine the two variables into a **personalised welcome message** using a formatting method of your choice. The output should greet the student and refer to the course.

- *Example output: "Welcome, Alice! You have successfully enrolled in the Introduction to Python course."*

Now, update the course name to include a **course code** at the beginning. Use a format like `CS101: <existing course name>`. Replace the original course name with this updated version.

Reuse the welcome message to reflect the new course name and print it again.

Finally, delete the variable that stores the student's name using the `del` Python keyword.

Try to print it after deletion to confirm that it's been removed.

Step 2: Create strings in Python with single quotes, double quotes, or triple quotes.

In this step, practise using all three types of quotation marks for different purposes.

- Create a new string using **single quotes** to represent another **course code** (e.g. something like SE102).
- Create another string using **double quotes** to represent a **brief course description**. Make sure it explains what the course is about in one sentence.
- Now create a **multi-line string** using **triple quotes**. This could be a welcome message or a short disclaimer with two or more lines of text.
    - *Example idea: Mention that the course material is confidential or describe the type of content the course will cover.*
- Print each of these strings to check they work as intended.

Your facilitator will complete the following evaluation checklist:

Check that the following is accomplished:

Item	Checked (Yes=5 No=0)	Comment: where did you find the evidence?
PA0101 Create, format, modify and delete strings in Python		
PA0102 Create strings in Python with single quotes, double quotes, or triple quotes		
Name of facilitator		
Signature		
Date		
Total		**/10**

Take note

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Practical Skills Module 2: Topic 2

Topic Code	PM-02-PS02:
Topic	Use numbers in Python

Scope of Practical Skill

Given an applicable instruction and access to a learning platform, the learner must be able to:

- PA0201 Create an integer, through assigning the integer value in a variable
- PA0202 Create a number with the float data type, through assigning the value into a variable
- PA0203 Check the data type of a variable or data using python's built-in `type()` function
- PA0204 Create large numbers in python using the underscore (_)
- PA0205 Add, subtract, multiply, divide

Applied Knowledge

- AK0201 Concept, definition and functions of Python programming functionalities

Internal Assessment Criteria

- IAC0201 Expected results with numbers in Python are achieved

Resources:

Knowledge	Information from Python Programmer Curriculum
National Curriculum Framework	900221-000-00-PM-02

Practical task

Follow the facilitator's instructions to complete the following activities:

Scenario: Building a Simple "Shopping Cart Calculator"
In this scenario, you are building a program to manage and calculate prices in a shopping cart. The steps involve working with integers, floats, and arithmetic operations to compute the total price, tax, and discounts. This program will simulate a basic price calculation system used in online stores.

Create Python file called **online_store.py** and complete the following:

Step 1: Assign an integer value to represent quantity.

- Assign an integer value to a variable that represents the **quantity of items** in the shopping cart.
- Use a clear and descriptive variable name (e.g., `item_quantity`).

Step 2: Assign a float value to represent price.

- Assign a float value to a variable that represents the **price of a single item**.
- Ensure the value includes decimals to represent cents.

Step 3: Use the `type()` function to check data types.

- Use the `type()` function to check the data type of both variables created in Step 1 and Step 2.
- Print the result of each check.
    - *Expected output:* One result should indicate `<class 'int'>`, the other `<class 'float'>`.

Step 4: Assign a large number using underscores.

- Assign a large number (e.g., total shopping budget) to a variable using underscores for readability.

- Example: `1_000_000` for one million.
- Print the value to confirm it behaves like a regular number.
    - *Expected output:* The printed value should display without underscores (e.g., `1000000`).

Step 5: Perform arithmetic operations.

Use the variables created so far to complete the following:

- **Calculate the total cost** by multiplying the item quantity by the item price.
- **Apply a discount** by subtracting a fixed amount (you decide the value) from the total cost.
- **Calculate the tax amount** by applying a tax percentage (e.g., 15%) to the discounted total.
- **Compute the final amount payable** by adding the tax to the discounted total.
- Print the result of each calculation step with a label so it's easy to understand. *Expected output:* Using the following example inputs:
    - Item quantity: `3`
    - Item price: `249.99`
    - Discount: `50`
    - Tax rate: `0.15` (i.e., 15%)

Your output should look similar to the example below:

```
Total cost before discount: 749.97
Total after discount: 699.97
Tax amount: 104.9955
```

```
Final amount payable: 804.9655
```

Your facilitator will complete the following evaluation checklist:

Check that the following is accomplished:

Item	Checked (Yes=5 No=0)	Comment: where did you find the evidence?
PA0201 Create an integer, through assigning the integer value in a variable		
PA0202 Create a number with the float data type, through assigning the value into a variable		
PA0203 Check the data type of a variable or data using python's built-in type() function		
PA0204 Create large numbers in python using the underscore(_)		
PA0205 Add, subtract, multiply, divide		
Name of facilitator		
Signature		
Date		
	Total	**/25**

Take note

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Practical Skills Module 2: Topic 3

Topic Code	PM-02-PS03:
Topic	Create a float in Python

Scope of Practical Skill

Given an applicable instruction and access to a learning platform, the learner must be able to:

- PA0301 Create a float
- PA0302 Round floats
- PA0303 Convert a list of strings to a float
- PA0304 Compare floats
- PA0305 Multiply floats
- PA0306 Format floats
- PA0307 Create a list of floats
- PA0308 Add floats
- PA0309 Store a float in an integer
- PA0310 Sum a list of floats
- PA0311 Convert an integer to a float

Applied Knowledge

- AK0301 Concept, definition and functions of Python data types

Internal Assessment Criteria

- IAC0301 Expected results with Python floats are achieved

Resources:

Knowledge	Information from Python Programmer Curriculum
National Curriculum Framework	900221-000-00-PM-02

Practical task

Follow the facilitator's instructions to complete the following activities:

Scenario: Building a "Grading System for Students"
In this task, you are building a **Grading System** using Python. The program will calculate, manipulate, and present students' grades. You'll practise working with **floats**, **lists**, **numeric conversions**, and **basic formatting**, as well as using `input()` to collect a student's grade.

Create a Python file called **grading_system.py** and complete the following:

Step 1: Assign a float to represent a grade.

- Use the `input()` function to get a student's **assignment grade** from the user.
- Convert the input into a float and store it in a variable.

Step 2: Round the float to the nearest integer.

- Round the student's grade to the nearest whole number using Python's `round()` function.
- Print the rounded grade with a label.
    - *Example output:* `Rounded grade: 84`

Step 3: Convert a list of strings to floats.

- Create a list of **grades stored as strings** (e.g., `["78.5", "82.0", "90.25"]`).
- Convert each element in the list to a float and store them in a new list.

Step 4: Compare floats.

- Set a **passing threshold** (e.g., `50.0`).
- Use an `if` statement to compare the original input grade to the threshold.
- Print a message indicating whether the student passed or failed.
    - *Example output:* `Passed: True` or `Student did not meet the threshold.`

Step 5: Multiply floats.

- Create a **weight factor** (e.g., `0.4` to represent 40%).
- Multiply the student's grade by the weight factor to get the **weighted score**.
- Print the weighted score.

Step 6: Format a float to two decimal places.

- Format the weighted score to **two decimal places** for clean presentation.
- Print the formatted score with a label.
    - *Example output:* `Weighted score: 33.60`

Step 7: Create a list of floats.

- Create a list to store at least **three different assignment grades** as float values.
- Print the list to confirm it's correctly formatted.

Step 8: Add floats together.

- Add the grades in the list together to get the **total score**.
- Print the total with a clear label.
    - *Example output:* `Total score: 254.75`

Step 9: Convert a float to an integer.

- Convert the total score into an integer using the `int()` function (this will round down).
- Print the converted value.
    - *Example output:* `Total score rounded down: 254`

Step 10: Use `sum()` on a list of floats.

- Use the built-in `sum()` function to calculate the **sum of all grades** in the list created in Step 7.
- Print the result to confirm accuracy.

Step 11: Convert an integer to a float.

- Assume you have an integer value representing the number of assignments (e.g., 3).
- Convert this integer to a float using `float()`.
- Print the converted value.
    - *Example output:* `Number of assignments as float: 3.0`

Your facilitator will complete the following evaluation checklist:

Check that the following is accomplished:

Item	Checked (Yes=5 No=0)	Comment: where did you find the evidence?
PA0301 Create a float		
PA0302 Round floats		
PA0303 Convert a list of strings to a float		
PA0304 Compare floats		
PA0305 Multiply floats		
PA0306 Format floats		
PA0307 Create a list of floats		
PA0308 Add floats		
PA0309 Store a float in an integer		
PA0310 Sum a list of floats		
PA0311 Convert an integer to a float		

Name of facilitator	
Signature	
Date	

Total	**/55**

Take note

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Practical Skills Module 2: Topic 4

Topic Code	PM-02-PS04:
Topic	Use double data types in Python

Scope of Practical Skill

Given an applicable instruction and access to a learning platform, the learner must be able to:

- PA0401 Use double data types in Python

Applied Knowledge

- AK0401 Concept, definition and functions of Python data types

Internal Assessment Criteria

- IAC0401 Expected results with Python double are achieved

Resources:

Knowledge	Information from Python Programmer Curriculum
National Curriculum Framework	900221-000-00-PM-02

Practical task

Follow the facilitator's instructions to complete the following activities:

Scenario: Banking System - Interest Calculation for Savings Accounts
You are tasked with creating a Python program for a banking system that calculates interest for customers' savings accounts. In this scenario, you use **double-like data types** (represented as float in Python) to handle financial data such as balances, interest rates, and monthly interest.

Create a Python file called **banking_system.py** nd complete the following steps:

Step 1: Assign float values to customer balances.

- Create **three customer account balances**.
- Use float values to represent each balance (e.g., `10000.50`), including cents.
- Store each balance in its own variable with a clear name.

  Tip: These represent the amount each customer has in their savings account.

Step 2: Set the interest rate.

- Assign a float value to a variable to represent the **annual interest rate** (e.g., `3.5` for 3.5%).

  Note: This is an annual rate, but you will calculate monthly interest in the next step.

Step 3: Calculate monthly interest.

- For each customer:
    1. Convert the annual rate to a monthly rate by dividing it by 12.
    2. Calculate the monthly interest using the formula:

```
monthly_interest = account_balance × (monthly_interest_rate / 100)
```

Example: If the balance is `10000.50` and the annual interest rate is `3.5`, the monthly interest rate would be `3.5 / 12 = 0.291666…`

Then: `monthly_interest = 10000.50 × (0.291666 / 100)`

Step 4: Update the account balance.

- Add the calculated monthly interest to the customer's existing account balance.
- Update the original balance variable to reflect the new total.

Step 5: Format the result for display.

- Print each customer's updated account balance.
- Format the output to **two decimal places** for clarity and professionalism.
    - *Example output:*

```
Customer 1's updated balance: R10029.69
Customer 2's updated balance: R8450.13
```

Your facilitator will complete the following evaluation checklist:

Check that the following is accomplished:

Item	Checked (Yes=5 No=0)	Comment: where did you find the evidence?
PA0401 Use double data types in Python		
Name of facilitator		
Signature		
Date		
Total		**/5**

Take note

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Practical Skills Module 2: Topic 5

Topic Code	PM-02-PS05:
Topic	Declare a Boolean – true or false

Scope of Practical Skill

Given an applicable instruction and access to a learning platform, the learner must be able to:

- PA0401 Use double data types in Python

Applied Knowledge

- AK0401 Concept, definition and functions of Python data types

Internal Assessment Criteria

- IAC0401 Expected results with Python double are achieved

Scope of Practical Skill

Given an applicable instruction and access to a learning platform, the learner must be able to:

- PA0501 Declare a Boolean
- PA0502 Define true or false keywords
- PA0503 Compare two values
- PA0504 Evaluate two variables
- PA0505 Create functions that returns a Boolean Value

Applied Knowledge

- AK0501 Concept, definition and functions of Python data types

Internal Assessment Criteria

- IAC0501 Expected results with Python Boolean are achieved

Resources:

Knowledge	Information from Python Programmer Curriculum
National Curriculum Framework	900221-000-00-PM-02

Practical task

Follow the facilitator's instructions to complete the following activities:

Scenario: Library Membership System - Validating Borrower Eligibility
In this scenario, you are tasked with creating a Python program for a library membership system. The program uses Boolean values to determine whether a user is eligible to borrow a book, has overdue books, or meets specific criteria for borrowing. Here's how the steps align with the scenario:

Create a Python file called **borrower_eligibility.py** and complete the following steps:

Step 1: Declare a Boolean to represent membership status

- Create a variable to store whether the user is a **registered library member**.
- Assign either `True` or `False` to the variable based on the user's membership status.
    - *Example idea:* A member should have a value of `True`.

Step 2: Use `True` or `False` to indicate overdue books

- Create another Boolean variable that represents whether the user has **any overdue books**.
- Assign the appropriate `True` or `False` value to reflect that status.

  Tip: If a user has overdue books, set the value to `True`; otherwise, `False`.

Step 3: Compare values to validate a membership ID

- Simulate a valid **membership ID** by assigning a reference value (e.g., `'ABC123'`).

- Create a variable that stores the user's entered ID (you can hardcode or use `input()`).
- Use a comparison operator to check if the user-entered ID matches the valid one.
    - *Expected output:* `Membership ID valid: True` or `False`

Step 4: Evaluate multiple Boolean conditions

- Check whether the user is allowed to borrow a book:
    - The user must **be a member** (`True`)
    - The user must **not have overdue books** (`False`)
- Use a logical operator (`and`, `or`, `not`) to combine the conditions and print the result.
    - *Example output:* `User eligible to borrow: True`

Step 5: Create a function that returns a Boolean

- Write a function named `can_borrow_book()` that accepts:
    - Membership status
    - Overdue book status
    - Membership ID check result
- The function should return `True` if **all conditions** for borrowing are met.
- Otherwise, return `False`.

  Tip: Use an `if` statement or a return statement with logical conditions inside the function.

- Call the function and print the result.
    - *Expected output:* `Final eligibility status: True` or `False`

Your facilitator will complete the following evaluation checklist:

Check that the following is accomplished:

Item	Checked (Yes=5 No=0)	Comment: where did you find the evidence?
PA0501 Declare a Boolean		
PA0502 Define true or false keywords		
PA0503 Compare two values		
PA0504 Evaluate two variables		
PA0505 Create functions that returns a Boolean Value		
Name of facilitator		
Signature		
Date		
	Total	**/25**

Take note

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Practical Skills Module 2: Topic 6

Topic Code	PM-02-PS06:
Topic	Use Python lists to store multiple items in a single variable

Scope of Practical Skill

Given an applicable instruction and access to a learning platform, the learner must be able to:

- PA0601 Create lists
- PA0602 Access list elements
- PA0603 List slicing
- PA0604 Add/Change List Elements
- PA0605 Delete list elements
- PA0606 Find AVERAGE of a List in Python
- PA0607 Remove elements from Python List with [clear, POP, remove, del]
- PA0608 Concatenate variables

Applied Knowledge

- AK0601 Concept, definition and functions of Python data types

Internal Assessment Criteria

- IAC0601 Expected results with Python list are achieved

Resources:

Knowledge	Information from Python Programmer Curriculum
National Curriculum Framework	900221-000-00-PM-02

Practical task

Follow the facilitator's instructions to complete the following activities:

Scenario: Student Grades Management System
You are tasked with developing a Python program for a school that manages student grades. The program will allow the user to perform various operations on a list of grades, such as accessing specific grades, modifying them, removing grades, and calculating the average.

Create a Python file called **student_grades.py** and complete the following steps:

Step 1: Create a list of student grades.

- Create a list that contains at least **five numeric grades** as integers or floats.
- Store the list in a variable with a descriptive name (e.g., `grades_list`).

  Tip: Use values that reflect actual grades (e.g., between 0 and 100).

Step 2: Access individual elements by index.

- Use indexing to **print the first and last grade** from the list.
- Include a label for each printed value so the output is clear.
  - *Example output:*

```
First grade in list: 88
Last grade in list: 76
```

Step 3: Use list slicing.

- Extract and print a **subset of the list** containing the grades of the **top three students**.
- Use slicing notation to get the first three elements.
    - *Expected output:* A sublist with the first three grades.

Step 4: Add and update elements.

- **Add a new grade** to the end of the list using the appropriate list method.
- **Update one of the existing grades** by assigning a new value to a specific index.

  Tip: Choose any index and replace that grade with a new value.

Step 5: Delete a specific element.

- Remove one grade from the list by specifying its index using the `del` keyword.
- Print the updated list to confirm the change.

Step 6: Calculate the average of the grades.

- Use the `sum()` function and the `len()` function to calculate the **average** of the grades.
- Format the average to **two decimal places** and print it.
    - *Example output:* `Average grade: 78.56`

Step 7: Remove elements using different methods.

- Demonstrate at least **three of the following methods** to remove grades from the list:
    1. `pop()` — remove by index
    2. `remove()` — remove by value
    3. `clear()` — remove all items
    4. `del` — remove by index or slice
- Print the list after each removal so you can see the effect of each method.

  Tip: You may want to re-initialise the list before each method for testing.

Step 8: Concatenate two lists.

- Create a second list with additional grades.
- Combine the original list with the new one into a single list using list concatenation.
- Print the final combined list.
    - *Example output:* `Combined grades list: [85, 90, 78, 92, 88, 79, 84]`

Your facilitator will complete the following evaluation checklist:

Check that the following is accomplished:

Item	**Checked (Yes=5 No=0)**	**Comment: where did you find the evidence?**
PA0601 Create lists		
PA0602 Access list elements		
PA0603 List slicing		
PA0604 Add/Change List Elements		
PA0605 Delete list elements		
PA0606 Find AVERAGE of a List in Python		
PA0607 Remove elements from Python List with [clear, POP, remove, del]		
PA0608 Concatenate variables		
Name of facilitator		
Signature		
Date		
	Total	**/40**

Take note

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Practical Skills Module 2: Topic 7

Topic Code	PM-02-PS07
Topic	Use Python tuple to store multiple items in a single variable

Scope of Practical Skill

Given an applicable instruction and access to a learning platform, the learner must be able to:

- PA0701 Pack and unpack tuples
- PA0702 Compare tuples
- PA0703 Use tuples as keys in dictionaries
- PA0704 Delete tuples
- PA0705 Slice tuples

Applied Knowledge

- AK0701 Concept, definition and functions of Python data types

Internal Assessment Criteria

- IAC0701 Expected results with Python tuple are achieved

Resources:

Knowledge	Information from Python Programmer Curriculum
National Curriculum Framework	900221-000-00-PM-02

Practical task

Follow the facilitator's instructions to complete the following activities:

Scenario: E-Commerce Inventory Management System
In this scenario, you are tasked with developing a Python program to manage product data for an e-commerce platform. Each product's information, such as its name, category, and stock, is stored as tuples. The program will allow you to perform operations like packing and unpacking tuples, comparing them to sort inventory, using tuples as keys in a dictionary for quick lookups, deleting tuples for discontinued products, and slicing tuples to extract specific data.

Create a Python file called **product_inventory.py** and complete the steps below:

Step 1: Pack and unpack tuples.

- Create a tuple to represent a product using the following details:
  - Product name (string)
  - Category (string)
  - Stock quantity (integer)

    Example structure: `("Bluetooth Speaker", "Electronics", 120)`

- Unpack the tuple into separate variables and print each variable with a label.
  - *Expected output:*

```
Product Name: Bluetooth Speaker
Category: Electronics
Stock: 120
```

Step 2: Compare tuples to sort by stock quantity.

- Create a list containing at least **three product tuples** with the same structure.
- Sort the list of tuples **by the stock quantity** (which is the third item in the tuple).
- Print the sorted list to show the products ordered from lowest to highest stock.

  Tip: You may use a sorting method that sorts based on the third index of each tuple.

Step 3: Use tuples as keys in a dictionary.

- Use each product's **(name, category)** as the key in a dictionary.
- Store the stock quantity as the value for each key.
- Print the dictionary to show product details mapped to their current stock.

  Tip: This simulates quick lookup by product identity.

Step 4: Delete tuples from list or dictionary.

- Choose one product that is now **discontinued**.
- Remove the associated tuple from:
  - The original **list of tuples**
  - The **dictionary** created in Step 3
- Print both updated structures to confirm that the product has been removed.

Step 5: Slice tuples to extract specific details.

- From any product tuple, use **tuple slicing** to extract only the **product name and category**, ignoring the stock quantity.
- Print the sliced result to verify that only the first two elements are included.
  - *Expected output:* A tuple like `("Bluetooth Speaker", "Electronics")`

Your facilitator will complete the following evaluation checklist:**Check that the following is accomplished:**

Item	Checked (Yes=5 No=0)	Comment: where did you find the evidence?
PA0701 Pack and unpack tuples		
PA0702 Compare tuples		
PA0703 Use tuples as keys in dictionaries		
PA0704 Delete tuples		
PA0705 Slice tuples		
Name of facilitator		
Signature		
Date		
Total		**/35**

Take note

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Practical Skills Module 2: Topic 8

Topic Code	PM-02-PS08
Topic	Use dictionaries in Python to store data values

Scope of Practical Skill

Given an applicable instruction and access to a learning platform, the learner must be able to:

- PA0801 Create a dictionary
- PA0802 Access elements
- PA0803 Add and remove elements
- PA0804 Check if a given key already exists in a dictionary
- PA0805 Add Key/Value Pair
- PA0806 Update dictionary
- PA0807 Merge dictionaries

Applied Knowledge

- AK0801 Concept, definition and functions of Python data types

Internal Assessment Criteria

- IAC0801 Expected results with Python dictionaries are achieved

Resources:

Knowledge	Information from Python Programmer Curriculum
National Curriculum Framework	900221-000-00-PM-02

Practical task

Follow the facilitator's instructions to complete the following activities:

Scenario: Student Management System
In this scenario, a school needs a Python program to manage student data. The program will use dictionaries to store information about students, such as their names, grades, and subjects. You will perform various operations like creating dictionaries, accessing and updating student information, adding or removing students, and merging records.

Create a Python file called **student_management.py** and complete the steps below:

Step 1: Create a dictionary of student data.

- Create a dictionary where:
  - Each **key** is a unique student ID (e.g., `"S001"`)
  - Each **value** is another dictionary with:
    - `"name"`: the student's name
    - `"grade"`: the student's grade
    - `"subjects"`: a list of subjects

Example structure:

```
{
    "S001": {"name": "Zara", "grade": "A", "subjects": ["Math", "English"]},
    "S002": {"name": "Liam", "grade": "B", "subjects": ["Science",
"History"]}
}
```

Step 2: Access a student's details by ID.

- Use the student ID to access and print all details for a specific student.
- Use a `print()` statement with labels to clearly show their information.
    - *Example output:*

      ```
      Student Name: Zara, Grade: A, Subjects: ['Math', 'English']
      ```

Step 3: Add a new student and remove one.

- Add a new student record to the dictionary using a new student ID.
- Then remove an existing student using their student ID with the `del` keyword.
- Print the dictionary to confirm the changes.

Step 4: Check for a student ID before adding/modifying.

- Use an `if` statement to check whether a specific student ID **already exists** before:
    - Adding a new student
    - Modifying an existing one
- Print a message indicating whether the student ID was found.
    - *Example output:* `Student ID S003 already exists.` or `Student ID S005 not found.`

Step 5: Add a new key-value pair to a student record.

- Add a new key called `"attendance"` to a selected student's dictionary.
- Set the value as a percentage (e.g., `95.5`) to represent attendance.

Step 6: Update grade and add a subject.

- Change a student's `"grade"` value to a new grade.
- Add a new subject to the `"subjects"` list using an appropriate list method.

  Tip: Append to the existing list of subjects without replacing it.

Step 7: Merge in new student records.

- Create a second dictionary containing **two more students** with the same structure as the original.
- Merge the new dictionary into the original one using the `update()` method.

- Print the final dictionary to show all student records combined.

Your facilitator will complete the following evaluation checklist:

Check that the following is accomplished:

Item	Checked (Yes=5 No=0)	Comment: where did you find the evidence?
PA0801 Create a dictionary		
PA0802 Access elements		
PA0803 Add and remove elements		
PA0804 Check if a given key already exists in a dictionary		
PA0805 Add Key/Value Pair		
PA0806 Update dictionary		
PA0807 Merge dictionaries		
Name of facilitator		
Signature		
Date		
	Total	**/35**

Take note

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Practical Skills Module 2: Topic 9

Topic Code	PM-02-PS09
Topic	Use Python sets to store multiple items in a single variable

Scope of Practical Skill

Given an applicable instruction and access to a learning platform, the learner must be able to:

- PA0901 Create Python sets
- PA0902 Add or remove elements
- PA0903 Sort lists
- PA0904 Determine how many items a set has
- PA0905 Join sets

Applied Knowledge

- AK0901 Concept, definition and functions of Python data types

Internal Assessment Criteria

- IAC0901 Expected results with Python sets are achieved

Resources:

Knowledge	Information from Python Programmer Curriculum
National Curriculum Framework	900221-000-00-PM-02

Practical task

Follow the facilitator's instructions to complete the following activities:

Scenario: Managing a Library Book Collection
In this scenario, a library manager uses Python to manage a collection of book genres available in the library. The program will use sets to track genres, perform operations like adding or removing genres, sorting lists of books within a genre, checking the number of available genres, and merging multiple genre sets when new collections are acquired.

Create a Python file called **genre_manager.py** and complete the following steps:

Step 1: Create sets for available book genres.

- Create two sets:
  - One for **fiction genres** (e.g., `'Fantasy'`, `'Romance'`, `'Historical'`)
  - One for **non-fiction genres** (e.g., `'Biography'`, `'Science'`, `'Self-help'`)
- Store them in clearly named variables.

  Tip: Sets are defined using curly braces `{}` and contain unique, unordered items.

Step 2: Add and remove elements from sets.

- Add the genre `"Thriller"` to the **fiction set** using an appropriate set method.
- Remove the genre `"Science"` from the **non-fiction set**.
- Print both sets to show the updated collections.
  - *Expected output:*

```
Updated fiction genres: {'Fantasy', 'Romance', 'Historical',
```

```
'Thriller'}
Updated non-fiction genres: {'Biography', 'Self-help'}
```

Step 3: Sort a list of books in a genre.

- Create a list of books in the **Fantasy** genre:
  - `"The Hobbit"`, `"Harry Potter"`, `"Game of Thrones"`
- Sort the list alphabetically and print the sorted result.
  - *Expected output:* `Sorted Fantasy Books: ['Game of Thrones', 'Harry Potter', 'The Hobbit']`

    Note: This step uses a list, not a set, because sets cannot be sorted.

Step 4: Count the number of genres in each set.

- Use the `len()` function to determine how many genres are in:
  - The fiction set
  - The non-fiction set
- Print the results with appropriate labels.
  - *Example output:*

    ```
    Number of fiction genres: 4
    Number of non-fiction genres: 2
    ```

Step 5: Merge fiction and non-fiction genres.

- Combine the two sets to form a complete set of **all genres available** in the library.
- Use a set operation (e.g., `union()`) to merge them into a new set.
- Print the final combined set.
  - *Expected output:*

    ```
    All available genres: {'Fantasy', 'Thriller', 'Biography', 'Romance', 'Historical', 'Self-help'}
    ```

Your facilitator will complete the following evaluation checklist:

Check that the following is accomplished:

Item	Checked (Yes=5 No=0)	Comment: where did you find the evidence?
PA0901 Create Python sets		
PA0902 Add or remove elements		
PA0903 Sort lists		
PA0904 Determine how many items a set has		
PA0905 Join sets		
Name of facilitator		
Signature		
Date		
Total		**/25**

Take note

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Practical Skills Module 2: Topic 10

Topic Code	PM-02-PS010
Topic	Create Arrays in Python

Scope of Practical Skill

Given an applicable instruction and access to a learning platform, the learner must be able to:

- PA1001 Create arrays
- PA1002 Import arrays
- PA1003 Access array elements
- PA1004 Insert elements
- PA1005 Modify elements
- PA1006 Pop an element from array
- PA1007 Delete elements
- PA1008 Search and get the index of a value in an array
- PA1009 Reverse an array
- PA1010 Count the occurrence of a value in array
- PA1011 Traverse an array

Applied Knowledge

- AK1001 Concept, definition and functions of Python programming functionalities

Internal Assessment Criteria

- IAC1001 Expected results with Python arrays are achieved

Resources:

Knowledge	Information from Python Programmer Curriculum
National Curriculum Framework	900221-000-00-PM-02

Practical task

Follow the facilitator's instructions to complete the following activities:

Scenario: Inventory Management for an Electronics Store
An electronics store manager uses Python to manage an inventory of products. The inventory is stored in arrays, where each array represents a category of products (e.g., laptops, phones, tablets). The manager needs to perform operations like adding new products, modifying existing ones, deleting discontinued items, searching for specific products, and analysing the inventory.

Create a Python file called **inventory_manager.py** and complete the steps below:

Step 1: Create arrays (lists) for inventory categories.

- Create two arrays (lists) to store the provided laptop and phone brands

```
laptops : ["Dell", "HP", "Lenovo", "Apple"]

phones : ["Samsung", "Apple", "OnePlus", "Nokia"]
```

Step 2: Access array elements.

- Access and print the **first element** in the `laptops` list.

Step 3: Insert a new product.

- Insert `"Acer"` into the `laptops` list at **position 2**.
- Print the updated list to confirm the change.

Step 4: Modify an existing product.

- Change the **second item** in the `phones` list (index 1) to `"iPhone"`.
- Print the updated `phones` list.
    - *Expected output:* `Updated phones list: ['Samsung', 'iPhone', 'OnePlus', 'Nokia']`

Step 6: Pop an element from an array.

- Remove the **last item** from the `phones` list using the `pop()` method.
- Print the list after the item is removed.

Step 7: Delete an element at a specific position.

- Remove the item at index `1` from the `phones` list.
- Print the updated list to confirm the deletion.

Step 8: Search and get the index of a value.

- Search for the product `"Lenovo"` in the `laptops` list.
- Find and print its index.
    - *Expected output:* `Index of Lenovo: 2`

Step 9: Reverse the array.

- Reverse the order of elements in the `phones` list.
- Print the list after reversing it.

Step 10: Count how many times a value appears.

- Count how many times `"Apple"` appears in the `phones` list.
- Print the result.
    - *Expected output:* `Apple appears 1 time(s) in phones`

Step 11: Traverse the array.

- Use a `for` loop to **traverse and print** each product in the `laptops` list.
- Label the output for clarity.
    - *Example output:*

```
Laptop: Dell
Laptop: HP
Laptop: Lenovo
Laptop: Apple
```

Your facilitator will complete the following evaluation checklist:

Check that the following is accomplished:

Item	Checked (Yes=5 No=0)	Comment: where did you find the evidence?
PA1001 Create arrays		
PA1002 Import arrays		
PA1003 Access array elements		
PA1004 Insert elements		
PA1005 Modify elements		
PA1006 Pop an element from array		
PA1007 Delete elements		
PA1008 Search and get the index of a value in an array		
PA1009 Reverse an array		
PA1010 Count the occurrence of a value in array		
PA1011 Traverse an array		

Name of facilitator	
Signature	
Date	

Total	**/55**

Take note

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Requirements met		**Total**	**/310**

233 +/310			
Requirements not met Under /310			
Facilitator signature		**Date**	

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.