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

Practical Skills Module Specifications

List of Practical Skill Module Specifications

900221-000-00-PM-01	Install computer software and hardware	L4	Cr3
900221-000-00-PM-02	Troubleshoot computer and network faults	L4	Cr8
900221-000-00-PM-03	Maintain computer and network security	L4	Cr6
900221-000-00-PM-04	Provide support to end Users	L4	Cr8
900221-000-00-PM-05	Getting Started with REST API and GUI	L4	Cr4
900221-000-00-PM-06	Use Cases with Python	L4	Cr11

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.

During the skills programme, you will complete various procedures and tasks that will be assessed to confirm your competence. This workbook includes formative assessment activities (which help you to assess your own learning and identify your strong and weak areas) and summative assessments (which assess your learning and competencies gained at the end of particular learning areas).

The formative assessment activities can be completed in groups, pairs, or on your own.

The **summative assessments** must always be **completed on your own**.

The activities and assessments 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.

Provider Programme 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 6

Use Cases with Python, NQF 4

The focus of the learning in this Module is on providing the learner with an opportunity to complete use cases using Python programming language

The learner will be required to:

- PM-06-PS01: Python exercises may be conducted at the end of the skills programme or may be integrated into learning at the end of the applicable module/section
- PM-06-PS02: Python exercises may be conducted at the end of the skills programme or may be integrated into learning at the end of the applicable module/section
- PM-06-PS03: Use a known framework (of own preference) with functions to build a game
- PM-06-PS04: Use a known framework (of own preference) with functions to assemble a complete solution applying all the below aspects given contextualised context within a specific sector

Practical Skills Module 6: Topic 1

Topic Code	PM-06-PS01
Topic	Python exercises may be conducted at the end of the skills programme or may be integrated into learning at the end of the applicable module/section

Scope of Practical Skill

Given an applicable instruction and access to a learning platform, connected to framework/ environment of own choice, the learner must be able to:

- PA0101 Use DML commands with Python
- PA0102 Understand Python Data Types
- PA0103 Use Python's Predefined String Functions
- PA0104 Master Python key concepts
- PA0105 Manipulate strings and data
- PA0106 Create and execute custom functions
- PA0107 Create, sort and modify Python lists
- PA0108 Deal with error handling in Python
- PA0109 Convert decimals

Applied Knowledge

- AK0101 Concept, definition and functions of Python programming principles

Internal Assessment Criteria

- IAC0101 Expected results are achieved

Resources:

Knowledge	Information from Python Programmer Curriculum
National Curriculum Framework	900221-000-00-PM-06

Practical task

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

Scenario: Customer Management
You will create a Customer Management System that includes database operations. Steps 1 and 8 focus on working with a SQLite database, creating, manipulating, and handling errors when accessing customer data. The other steps cover general Python tasks like data validation, string manipulation, functions, and list operations, which can be completed independently of the database.

Note: To keep your code organized and easy to test, implement each step as a separate function within the same file. You can use a main function to call these step functions in sequence.

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

Step 1: Use DML (Data Manipulation Language) commands with Python

- Create a SQLite database and a table for storing customer information (name, balance, email).
- Add customer records for the following individuals:

Name	Balance	Email
Alice Smith	R 120.50	Alice@example.com
Bob Johnston	R 75.30	Bob@example.com

- Implement the following operations:
  - **Create:** Insert new customer records into the database.
  - **Read:** Retrieve and display customer information from the database.
  - **Update:** Modify an existing customer's information (e.g., change the balance).

    - **Delete:** Remove a customer record from the database.

Step 2: Understand Python Data Types

- Validate customer data types (strings, floats, etc.):
    - Declare sample variables (e.g. `name = "Alice Smith"`, `balance = 120.50`).
    - Use a conditional to check if the name is a string and if the balance is a float.

  Tip: You can use the `isinstance()` function to validate the data types, which will return `True` if the value matches the specified type.

Step 3: Use Python's Predefined String Functions

- Declare variables to hold the raw customer name and email (e.g., `"  alice smith  "`, `"ALICE@EXAMPLE.COM"`).
- Format this data for consistency using Python's built-in string methods:
    - Use `.strip()` to remove any leading or trailing whitespace.
    - Use `.title()` to ensure proper capitalization of names (e.g., `"alice smith"` → `"Alice Smith"`).
    - Use `.lower()` to standardize email addresses (e.g., `"Alice@Example.Com"` → `"alice@example.com"`).

Step 4: Master Python key concepts

- Use loops, conditionals, and logical operations to manage customer balances:
    - Define a list of customer records, where each customer is represented as a dictionary with keys like `"name"` and `"balance"`.
    - Loop through the list using a `for` loop.
    - Use an `if` statement to check if a customer's balance is below a specific threshold (e.g., R 100.00).
    - Print a message for each customer that meets the condition.

Step 5: Manipulate strings and data

- Extract and display specific information about customers:

    - Define a string with customer name, email, and balance separated by commas.
    - Use the `.split()` string method to separate the values into individual components.
    - Convert the balance from a string to a float.
    - Print each piece of information in a readable format.

Step 6: Create and execute custom functions

- Create a function to calculate a discount for customers based on their balance.
- Define a function called `calculate_discount()` that accepts a customer's balance as a parameter.
- Use a conditional to check if the balance is greater than 100.
    - If it is, return a 10% discount based on the balance.
    - Otherwise, return 0 (no discount).
- Call the function with a sample balance and print the result.

Step 7: Create, sort and modify Python lists

- Create a list of dictionaries, where each dictionary contains a customer's name and balance.
- Use the `sorted()` function to sort the list by the customer's balance.
- Print the sorted list of customers, displaying both their name and balance.

Step 8: Deal with error handling in Python

- Handle errors when retrieving customer data from a database by using try-except blocks.
- For example, check if the query result is empty and raise an error if the customer is not found, then catch and display the error message gracefully.

Step 9: Convert decimals

- Convert balances to display them as integers.
- Format the balance as a currency using an f-string to show two decimal places (e.g., `$120.75`).

  Tip: You can import Python's `math` module to convert a decimal balance (float) to an integer using `math.floor().`

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 Use DML commands with python		
PA0102 Understand Python Data Types		
PA0103 Use Python's Predefined String Functions		
PA0104 Master Python key concepts		
PA0105 Manipulate strings and data		
PA0106 Create and execute custom functions		
PA0107 Create, sort and modify Python lists		
PA0108 Deal with error handling in Python		
PA0109 Convert decimals		
Name of member		
Signature		
Date		
	Total	**/45**

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 6: Topic 2

Topic Code	PM-06-PS02
Topic	Python exercises may be conducted at the end of the skills programme or may be integrated into learning at the end of the applicable module/section

Scope of Practical Skill

Given a range the learner must be able to:

- PA0201 Build a:
    - Alarm clock
    - Calculation of compounded interest
    - A GST (good service tax) calculator (GUI)
    - Language translator
    - Contact book
    - Create a scientific calculator in Python

Applied Knowledge

- AK0201 Concept, definition and functions of Python programming principles

Internal Assessment Criteria

- IAC0201 Expected results are achieved

Resources:

Knowledge	**Information from Python Programmer Curriculum**
National Curriculum Framework	900221-000-00-PM-06

Practical task

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

Scenario: Personal Toolkit
You are tasked with developing a Personal Toolkit in Python. This toolkit consists of a set of utility applications designed to help individuals manage their daily life more efficiently. Each tool serves a specific purpose, ranging from time management and financial planning to communication and calculations. Your job is to create each of these tools separately, all organized in a single folder. Each step below reflects a key function in your toolkit:

Create a folder named **python_projects** and complete the following:

Step 1: Build an alarm clock

- Create a Python file called **alarm_clock.py.**
- Build a basic alarm clock that plays a sound at a specified time:
  - Asks the user to enter an alarm time in `HH:MM:SS` format.
  - Continuously checks the current time until it matches the alarm time.
  - Play a sound and print “Alarm ringing!” when the time matches.
  - You can use the `sleep()` function to pause the program to check the time every second.
- *Tip*: You can import the `datetime`, `time`, and `winsound` (for Windows) libraries to create this program. If you're using macOS or Linux, you may use an alternative like `playsound`.

Step 2: Build a compounded interest calculator

- Create a Python file called **compounded_interest.py.**
- Build a program to calculate compounded interest based on user input:

- Use the compound interest formula: A = P * (1 + r/n)^(n*t)
- Prompt the user to enter:
  - Principal amount (P)
  - Annual interest rate (as a percentage)(r)
  - Time in years (t)
  - Number of times compounded per year (n)

- Convert the annual interest rate from a percentage to a decimal before using it in calculations.
- Calculate and print out the total total amount (A) and the interest earned (A - P).

Step 3: Build a GST (good service tax) calculator (GUI)

- Create a Python file called **gst_calculator.py**.
- Use PyQt5 to calculate GST and create a GUI with:
  - An input for Original Price
  - An input for GST Rate (%)
  - A Calculate button
  - A label showing the GST Amount and Total Price
- Formula to calculate GST:
  - GST Amount = (Original Price × GST Rate) / 100
  - Total Price = Original Price + GST Amount
- Handle invalid inputs with an error message.
- *Tip*: You can import the following classes from the `PyQt5.QtWidgets` module to build the interface: `QApplication`, `QWidget`, `QLabel`, `QLineEdit`, `QPushButton`, `QVBoxLayout`, `QMessageBox`

Step 4: Build a language translator

- Create a Python file called **language_translator.py.**
- Create a basic language translator using the `googletrans` library.
  - The program should:
    - Take user input text.
    - Take a target language code (e.g., `'fr'` for French, `'es'` for Spanish).
    - Translate the input into the specified language.

            - Print the translated result.
- Run your program and show the output in the terminal.
- *Tip*: To avoid errors related to asynchronous calls, install a specific stable version of googletrans that uses synchronous calls: `‘pip install googletrans==4.0.0-rc1’`.

Step 5: Build a contact book

- Create a Python file called **contact_book.py.**
- The program should store, retrieve, and display contact information.
- Use a dictionary to store contacts, where each key is a contact name and the value is another dictionary holding phone number and email.
- Implement a function called `add_contact` that:
    - Prompts the user to enter a name, phone number, and email.
    - Adds this information to the dictionary using the name as the key and a nested dictionary as the value.
    - Prints a confirmation message once the contact is added.
- Implement a function called `view_contacts()` that:
    - Loops through the dictionary.
    - Prints all stored contacts in a clear, readable format, including name, phone number, and email.
- Create a menu-driven interface in a loop that:
    - Displays the following options to the user:
        - Add Contact
        - View Contacts
        - Exit
    - Prompts the user to enter their choice.
    - Calls the appropriate function based on the user’s choice.

Step 6: Build a scientific calculator

- Create a Python file called **scientific_calculator.py.**
- The console-based scientific calculator should perform basic and scientific operations such as:
    - Addition, Subtraction, Multiplication, Division
    - Square Root

    - Power (Exponentiation)
    - Trigonometric Functions: Sine, Cosine, Tangent (input in degrees)
- Use the `math` module to perform advanced calculations.
- Validate user input (e.g., prevent division by zero).
- Ensure that after each operation, the menu is shown again until the user chooses to exit.
- *Tip*: Use the `math` module for advanced calculations such as:
    - `math.sqrt(x)` for square roots
    - `math.pow(x, y)` for exponentiation
    - `math.sin()`, `math.cos()`, `math.tan()` for trigonometric functions
    - `math.radians(x)` to convert degrees to radians

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 Build a:		
• Alarm clock		
• Calculation of compounded interest		
• A GST (good service tax) calculator (GUI)		
• Language translator		
• Contact book		
• Create a scientific calculator in python		
Name of member		
Signature		
Date		
	Total	**/30**

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 6: Topic 3

Topic Code	PM-06-PS03
Topic	Use a known framework (of own preference) with functions to build a game

Scope of Practical Skill

Given an applicable instruction and access to a learning platform, connected to framework/ environment of own choice, the learner must be able to:

- PA0301 Build a basic game using Python (compulsory)
    - Rock, Paper, Scissors Game
    - Number guessing game

Applied Knowledge

- AK0301 Concept, definition and functions of Python programming principles

Internal Assessment Criteria

- IAC0301 Expected results are achieved and the basic game is operable

Resources:

Knowledge	**Information from Python Programmer Curriculum**
National Curriculum Framework	900221-000-00-PM-06

Practical task

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

Scenario: Python Game Development You've been tasked with building a set of games to sharpen your coding skills and demonstrate your understanding of Python fundamentals, libraries, and game logic. Your goal is to implement two games, each in a separate Python file, organized in one folder. These games will vary in complexity and focus on different aspects of Python programming, including randomization and using a framework of your choice.

Create a folder named **python_games** and complete the following:

Step 1: Build a rock, paper, scissors game

- Create a Python file called **rps_game.py.**
- Build a basic Rock, Paper, Scissors game where the player chooses between three options, and the computer randomly selects one.
- Use Python's `random` module to generate the computer's choice.
- Write a function that compares the player's choice and the computer's choice to determine the result (win, lose, or tie).
- Display the result to the player.
- *Tips*:
  - You could create a graphical version of this game using the `Kivy` framework by importing `App` from `kivy.app`, and using `BoxLayout`, `Button`, and `Label` from `kivy.uix` to build the interface.

Step 2: Build a number guessing game

- Create a Python file called **guessing_game.py.**
- Use the `random` module to generate a number between 1 and 100.

- Decide on a fixed number of attempts for the player (e.g., 7 attempts).
- After each guess, display feedback:
  - “Too high!”
  - “Too low!”
  - “Correct!” and end the game if the guess is right.
- If the player runs out of attempts, display “Game Over” and reveal the number.
- Accept only valid integer guesses from the player. Handle invalid input gracefully (e.g., if the input is non-numeric, prompt the user again).
- *Tip*: For a graphical version, you could also use the `Kivy` framework by importing `App` from `kivy.app`, `BoxLayout` from `kivy.uix.boxlayout`, `Button` from `kivy.uix.button`, `Label` from `kivy.uix.label`, and `TextInput` from `kivy.uix.textinput`.

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 Build a basic game using Python (compulsory)		
• Rock, Paper, Scissors Game		
• Number guessing game		
Name of member		
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 6: Topic 4

Topic Code	PM-06-PS04
Topic	Use a known framework (of own preference) with functions to assemble a complete solution applying all the below aspects given contextualised context within a specific sector

Scope of Practical Skill

Given a range the learner must be able to:

- PA0401 Build a GUI application using Python (compulsory) which encompass any of these below:
    - Build a scientific calculator using Python
    - Digital phone book directory

Applied Knowledge

- AK0401 Concept, definition and functions of Python programming principles

Internal Assessment Criteria

- IAC0401 Expected results are achieved and the basic GUI application is operable

Resources:

Knowledge	**Information from Python Programmer Curriculum**
National Curriculum Framework	900221-000-00-PM-06

Practical task

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

Scenario: Python GUI Tools You are assigned to create two Python GUI applications for a desktop tools suite. The first is a scientific calculator capable of performing both basic and advanced mathematical operations, featuring robust input validation and error handling. The second is a digital phone book directory that enables users to add, search, update, and delete contacts.

Create a folder named **gui_projects** and complete the following:

Step 1: Build a GUI-Based Scientific Calculator

- Create a Python file called **scientific_calculator.py.**
- Design a user interface that includes:
  - A display field to show user input and calculation results.
  - Buttons for:
    - Basic arithmetic operations: addition (+), subtraction (−), multiplication (*), and division (/).
    - Advanced operations: exponentiation (^), square root (√) and trigonometric functions (sin, cos, tan).
    - Clear (`C`) button to reset input.
    - Equals (=) button to perform the calculation.
- Implement input validation and error handling to gracefully manage invalid cases such as division by zero and other invalid input formats.
- *Tip*: You can use `PyQt5` widgets and layout classes to build the GUI, including `QApplication` and `QWidget` for the main window, `QLineEdit` for the display field,

`QPushButton` for the calculator buttons, `QGridLayout` to arrange the buttons in a grid, and `QMessageBox` to display error messages.

Step 2: Build a digital phone book directory using Python

- Create a Python file called **phone_book.py.**
- This program should allow users to `add, search, update, delete, and view` contact information stored in a directory.
- Build a graphical user interface that includes:
    - Input fields for contact details (e.g., name and phone number).
    - Buttons for actions such as Add, Search, Update, Delete, and View All Contacts.
    - A display area to show contacts or search results.
- Implement functionality to handle each action, storing contacts in an appropriate data structure (like a list or dictionary).
- Include input validation to prevent errors such as empty fields or invalid phone numbers.
- *Tip*: You can import the following classes from the `PyQt5.QtWidgets` module to build the interface: `QApplication, QWidget, QVBoxLayout, QHBoxLayout, QLabel, QLineEdit, QPushButton, QListWidget, QMessageBox`

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 Build a GUI application using Python (compulsory) which encompass any of these below:		
• Build a scientific calculator using Python		
• Digital phone book directory		
Name of member		
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.

Requirements met **72+/95**		**Total**	**/95**
Requirements not met **Under 72/95**			
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.