# salesStudent Version

| Section A – Program/Course details | | | |
| --- | --- | --- | --- |
| **Qualification code:** | ICT40120 | **Qualification title:** | Certificate IV in Information Technology (Programming) |
| **Unit code:** | Python Programming  ICTPRG302   ICTPRG435 | **Unit title:** | Python Programming  Apply introductory programming techniques  Write scripts for software applications |
| **Department name:** | BDIT, Computing & Information Technology | **CRN number:** | 54602 |

| Section B – Assessment task details | | | |
| --- | --- | --- | --- |
| **Assessment number:** | 1 | **Semester/Year:** | 2/2023 |
| **Due date:** | Session 9 | **Duration of assessment:** | 9 weeks |
| **Assessment method** | Project/Report/Portfolio | **Assessment task results** | Ungraded result |
| Other: Click here to enter text. |

| Section C – Instructions to students |
| --- |
| **Task instructions:** |
| You are required to write programming code to create three software applications (practical tasks) with documentation which will constitute your portfolio. Instructions for each task is supplied below in the supporting document section.  If a topic / question is incomplete or insufficient, you will be asked to resubmit after receiving feedback. Some class time may be allocated during class to work on this assessment task but most will require work outside of class time If a task or part thereof is incomplete or insufficient, you will be asked to resubmit the program / documentation after receiving feedback. Some class time will be allocated during class to work on the tasks, however, most will require some work outside of class time.  Some performance criteria could be assessed multiple times – see marking criteria below.  The major concepts covered in each practical exercise are listed as follows:  • Practical Task 1 – Python Programming Basics • Practical Task 2 – Text-File Processing (with pseudocode and flowchart) • Practical Task 3 – Lists and Classes (with testing documentation) |

| Section D – Conditions for assessment | |
| --- | --- |
| **Conditions:**  Student to complete and attach Assessment Submission Cover Sheet to the completed Assessment Task. | |
| Each practical task is to be completed individually.  To ensure that you do not fall behind, make sure that each task is completed by the recommended date/time. ALL tasks MUST be completed to a satisfactory standard by the Session 9.   Python 3 programming language and the Thonny and/or and/or Visual Studio Code development environment to be used.   Make sure that you submit a completed and “signed” Assessment Coversheet after completing the final practical task. You can discuss with your assessor if you feel you require special consideration or adjustment for this task.  Students must meet all criteria listed in the marking guide to be satisfactory in this task. Students may resubmit any of the practical exercises / documentation of this assessment task if not successful, within the enrolment period as per Holmesglen conducting assessment procedure.   This is an individual task; however, you are allowed to get information, feedback and ideas from your assessor and peers to help complete any assessment item | |
| **Equipment/resources students must supply:** | **Equipment/resources to be provided by the RTO:** |
| Equipment/resources students must supply: USB or portable hard drive or cloud storage to store documents.  Students must also have access to development environments: Thonny, Visual Code Studio and Python 3  Students attending by remote learning must also have a computer (with a webcam and microphone / headset) and access to the internet. Students must also have downloaded and installed WebEx from: https://www.webex.com/downloads.html. | Computer and Internet Access, Thonny, Visual Code Studio and Python 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| Section E – Marking Sheet - Student Answer Sheet | | | |
| **Student ID:** | 100623374 | **Student name:** | Lucas Beyer |
| **Unit code:** | Python Programming  ICTPRG302   ICTPRG435 | **Unit title:** | Python Programming  Apply introductory programming techniques  Write scripts for software applications |
| **Date:** | 5/12/2023 | | |

**Project/Report/Portfolio task**

| **Criteria for assessment** | | **Satisfactory** | | **Comment** |
| --- | --- | --- | --- | --- |
| **Yes** | **No** |
| **The following has been submitted for assessment:** | | | | |
| Practical Task 1 – Python Programming Basics | |  |  |  |
| Practical Task 2 – Text-File Processing (with pseudocode, flowchart) | |  |  |  |
| Practical Task 3 – Lists and Classes (with testing documentation) | |  |  |  |
| **Marking criteria for each product document/s supplied:** | | | | |
| 1. | Each program applies basic language syntax rules and best practices. |  |  |  |
| 2. | Appropriate selection of language data types, operators and expressions to create clear and concise code. |  |  |  |
| 3. | Appropriate use of sequence, selection and repetition constructs and syntax used where required in each program. |  |  |  |
| 4. | Modules (classes and functions) used correctly. |  |  |  |
| 5. | Develop an algorithm that is an exact and sufficient description of the solutions. |  |  |  |
| 6. | Develop an algorithm that takes into account all possible situations including the program ending. |  |  |  |
| 7. | Arrays and array of objects, including algorithms to process arrays implemented. |  |  |  |
| 8. | Reading and writing data to a text-file implemented. |  |  |  |
| 9. | Coding standards applied to each program |  |  |  |
| 10. | Internal documentation via comments applied |  |  |  |
| 11. | Simple tests created to confirm that the code meets the design specifications. Test results documented. |  |  |  |
| 12. | Reviewed client requirements and sign off. |  |  |  |

| Section F – Feedback to Student | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Has the student successfully completed this assessment task?** | | | | | **Yes** | **No** |
|  |  |
| **Additional Assessor comments (as appropriate):** | | | | | | |
|  | | | | | | |
| **Resubmission allowed:** | **Yes** | **No** | **Resubmission due date:** |  | | |
| **Assessor name:** |  | | | | | |
| **Assessor signature:** |  | | | | | |
| **Date:** |  | | | | | |

**Supporting document**

# Practical Task 1 - Python Programming Basics

|  |  |  |  |
| --- | --- | --- | --- |
| **Unit code**: | Python Programming  ICTPRG302   ICTPRG435 | **Unit title:** | Python Programming  Apply introductory programming techniques  Write scripts for software applications |
| **Student ID:** | 100623374 | **Student name:** | Lucas Beyer |
| **Date of submission:** | 5/12/2023 | **Student signature**: | Lucas Beyer |

***Topics covered:*** *Basic Python syntax, variables (local and global), modularisation, functions and parameter passing and return values.*

**Steps:**

1. Create a folder called **PracticalTask1.** Inside this folder, create a Python file called “**StudentRegistration.py**”. The aim of this program is to get student details from the user of the program and then display information on the screen.  
     
   This program must be modular. That is, different functions should be created to carry out the different tasks involved. The functions you need to create yourself are shown in the structure chart and described below.

Diagram

Description automatically generated

Figure : Structure Charts representing Practical Task 1

1. The first task of this program to display a main heading. Create a function called printHeadings() to display the headings as shown below.

A picture containing knife

Description automatically generated

Figure : Output of printHeadings() function

1. The second task of this program is to obtain information about each student including their ID, name, the course in which they are enrolled, as well as their course fee. To do this, create a function called inputStudentDetails().

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure : Using a simpleDialog to obtain user input

1. After each student’s information is entered, display their details on the screen.  
     
   Note: As each fee is input a running total of all the fees is accumulated. The course fee value for each student will be **sent back** to the calling function where it will be added to a **local** variable called totalFees.
2. After inputting the details for the **three** students (*call this method three times for three different students*), the totalFees value will be passed to a third function called outputTotalFee(), which will display total fee for all three students.

Text, letter

Description automatically generated

Figure : Final output of program

1. *Do some research as to how to format the fees.*

**Note: Coding standards:**

*All your code must adhere to industry coding standards. The presentation of your code must at the very least show the following:*

* Variables and function names should start in **lowercase** letter. *eg.* ***p****ayRate,* ***c****alcFee();*
* Constant names should be declared in all **uppercase** lettering. *eg. TAX\_RATE*
* Code presentation should follow coding standards including proper indentation style and other formatting features such as appropriate blank lines, etc…
* Internally document your code by writing appropriate comments throughout, including a comment at the top of any file with your name, date, name of the file and a short description of the code.

*eg*

*# StudentRegistration.py*

*# This program processes student details to register them into courses and then*

*# displays this information onto the screen.*

*# Author: Agnes Hennessy*

*# Date: 20/07/21*

***When complete, zip up your directory and upload to the appropriate Dropbox on Brightspace.***

# Practical Task 2 – Reading and Writing Files (and Pseudocode)

|  |  |  |  |
| --- | --- | --- | --- |
| **Unit code**: | Python Programming  ICTPRG302   ICTPRG435 | **Unit title:** | Python Programming  Apply introductory programming techniques  Write scripts for software applications |
| **Student ID:** |  | **Student name:** |  |
| **Date of submission:** |  | **Student signature**: |  |

***Topics covered:*** *Algorithm design, selection, repetition, and text file processing*

This practical task is divided into two parts. In Part A you are to write pseudocode to represent the logic of your application. In Part B you are to implement your pseudocode. You may need to review your pseudocode after you have implemented your code and make updates and changes to it.

**Part A: Pseudocode and Flowchart**

Design your program by modelling your PayrollReport program using pseudocode. Use your pseudocode to create the programming code. In addition to this, draw a flowchart based on your pseudocode. Save these in a word document in the same folder as your submission.

**Part B: Application 1**

**Steps:**

1. Create a folder called **PracticalTask2.** Inside this folder, create a Python file called “**PayrollReport01.py**”.   
     
   The aim of this program is to read employee details from a text file (supplied and named **employees.txt**). Once each employee is read in, you are to determine if they are a “Manager”, “Sales” person, or in “Administration”.

After you have processed all records, you are to generate an output text file called **PayrollReport.txt** indicating the total salaries for Managers, Sales and Administrators. You are also required to indicate the total number of employees on the payroll, the overall total payroll amount and the average payroll.  
  
This program must be modular. That is, different functions should be created to carry out the different tasks involved.

1. Study the contents of **Employees.txt**. *Do NOT change or edit this file.*

Graphical user interface, text, application

Description automatically generated

Figure 1: Sample records from "**employee.txt**"

Read in and process each record one at a time. Display each record to the screen.

Text

Description automatically generated

rdFigure : Partial display of employees after being read into program

1. Once you have processed all the records, display the total pay of all employees, the number of records processed, the average pay, as well as the total pay for the three types of employee positions (Managers, Sales or Administration)to the screen.
2. Save the total pay of all employees, the number of records processed, the average pay as well as the total pay for the three types of employee positions (Managers, Sales or Administration)to a text file named **PayrollTeport.text**.

**Text

Description automatically generated**

Figure 3: Sample Screen display and contents of “**PayrollReport.txt”**

**All currency values must be rounded to two decimal places. You will need to research this yourself.**

**Part B: Application 2**

1. Inside the folder called **PracticalTask2** create another Python file called “**PayrollReport02.py**”.   
     
   This time, you are going to implement the functionality of the application described above, however you will be using a CSV file as your input file. The aim of this program is to read employee details from a CSV file (supplied and named **employees.csv**). Once each employee is read in, you are to determine if they are a “Manager”, “Sales” person, or in “Administration”.

After you have processed all records, you are to generate an output file called **PayrollReport.csv** indicating the total salaries for Managers, Sales and Administrators. You are also required to indicate the total number of employees on the payroll, the overall total payroll amount and the average payroll.  
  
This program must be modular. That is, different functions should be created to carry out the different tasks involved.

*Apply appropriate industry coding standards to your code, including: Comments, indentation style and naming conventions.*

***When complete zip up your whole directory (including your pseudocode) and upload to   
the appropriate Dropbox on Brightspace.***

# Practical Task 3 – Classes and Lists (and Testing Documentation)

|  |  |  |  |
| --- | --- | --- | --- |
| **Unit code**: | Python Programming  ICTPRG302   ICTPRG435 | **Unit title:** | Python Programming  Apply introductory programming techniques  Write scripts for software applications |
| **Student ID:** |  | **Student name:** |  |
| **Date of submission:** |  | **Student signature**: |  |

***Topics covered:*** *Classes, lists, sorting, searching and testing and recording*

This practical task is divided into two parts. In Part A you are to implement an application that allows you to add, sort , search and display books using a menu. In Part B you are to produce a testing strategy to test and record your application.

**Part A: Application**

**Steps:**

1. Create a folder called **PracticalTask3.** Inside this folder, create a Python class called “**Book.py**”. The Book class has the following attributes: isbn, title, author and price. Create a constructor method as well as a display method.  
     
   *Note: ISBN would not normally be declared as a number. Why?*
2. Create another file called **Catalog.py.** When the program starts, the user should be presented with the following menu system:

**Graphical user interface, text, application, chat or text message

Description automatically generated**

Figure 1: Menu system for Book Catalogue application

The user should only be able to enter options 1 – 5. Create a global list called **catalog**. When the program starts, add a few books to the catalogue.

eg:

* 0553296981, The Diary of a Young Girl, Frank, Anne, 16.50
* 1400082773, Dreams from My Father, Obama, Barrack, 24.99

1. When the user selects Option 1, they should be able to enter new book details (isbn, title, author and price). Instantiate a new Book object and add it to the catalogue. Include some error checking to ensure that all values have been entered.

|  |  |
| --- | --- |
| Graphical user interface, text, application, chat or text message  Description automatically generated | Graphical user interface, text, application, chat or text message  Description automatically generated |

Figure 2: Sample input dialogs

1. When the user selects Option 2, all books in the catalogue should be displayed, **sorted** according to price. If there are no books in the catalogue, an appropriate message should be displayed to the user.

|  |  |
| --- | --- |
| Graphical user interface, text, application, chat or text message  Description automatically generated | Graphical user interface, application  Description automatically generated |

Figure 3: Sample dialogs for input

1. When the user selects **Option 3**, they should be able to **search for a book** in the catalogue by title. If there are no books in the catalogue, an appropriate message should be displayed to the user. If the book is found in the catalogue, it’s details should be displayed in a message box. If the book is not found in the catalogue, an appropriate message should be displayed to the user.  
     
   Note: There may be more than one book with the same / similar title.

|  |  |
| --- | --- |
| Graphical user interface, text, application, chat or text message  Description automatically generated | Graphical user interface, application  Description automatically generated |

Figure 3: Sample dialogs for search functionality

1. When the user selects **Option 4**, they should be asked for an ISBN to enter. If the ISBN is found, then the user then the book can be **deleted AFTER confirmation**. If the book is NOT deleted, give a message to the user. Otherwise, if the book has been deleted, give an alternative message to the user.

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure 4: Sample deletion confirmation dialog

|  |  |
| --- | --- |
| Graphical user interface, text, application  Description automatically generated | Graphical user interface, application  Description automatically generated |

Figure 4: Sample feedback dialogs for deletion

1. When the user selects **Option 5, all books** in the catalogue should be **displayed**. If there are no books in the catalogue, an appropriate message should be displayed to the user.
2. When the user selects **Option 6**, the program should **exit** (after confirmation).

**Part B: Testing and Results**

After you have completed your analysis and design, you can **build** and **test** your application. You are also required to document conduct and document your **testing**.

**Testing**

While you are working on your applications, you will start to observe all the things that can go wrong. The errors in your code may be one of three categories:

* Syntax errors
* Logic errors
* Run-time errors

An application must be vigorously tested to ensure that these errors are fixed before the application is released. As you get more engrossed in your task, you may fail to “see” your errors. Because of this, it is often a practice to have someone else test your application. You have to prepare specific test data to test for each condition.

Example:

If the user has written an if statement to test the age of a person as being greater than 18, what happens if the age of the person is less than, equal to, or greater than 18? Does the application function as per specifications?

*You are to review the test plan you created for your own application. Test and record your results.*

Another factor that may be important is that you follow coding standards as prescribed by the company.

The easiest way to do this is to develop a test plan for each screen.

**Sample testing strategy:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Item | Test Data | Expected Result | Actual Result | Description of test (if applicable) |
| Menu System | 1 | Goes to method to input book details | √ |  |
| 45 | Displays “Invalid input” | √ |  |
| “abc” | Displays “Invalid input” | x | Program crashes – need new code to validate for such values |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Item Input Book Details | Test Data | Expected Result | Actual Result | Description of test (if applicable) |
| Author’s name | 123456 |  |  |  |
| “Smith” |  |  |  |
| *[left blank]* |  |  |  |
|  |  |  |  |

***You must also provide evidence of using a debugger and examine variable contents to detect errors. This may be included in your testing in the form of screen dumps.***

*NOTE: Add your own test data to the tables above and create new tables for the other Book inputs of this program.*

*Apply appropriate coding standards including: comments, indentation style and naming conventions for Classes, variables, constants and methods etc.*

***When complete zip up your whole directory project and upload to the appropriate Dropbox   
including the testing documentation on Brightspace.***

**Part B: Testing and Results**

After you have completed your analysis and design, you can **build** and **test** your application. You are also required to document conduct and document your **testing**.

**Testing**

While you are working on your applications, you will start to observe all the things that can go wrong. The errors in your code may be one of three categories:

* Syntax errors
* Logic errors
* Run-time errors

An application must be vigorously tested to ensure that these errors are fixed before the application is released. As you get more engrossed in your task, you may fail to “see” your errors. Because of this, it is often a practice to have someone else test your application. You have to prepare specific test data to test for each condition.

Example:

If the user has written an if statement to test the age of a person as being greater than 18, what happens if the age of the person is less than, equal to, or greater than 18? Does the application function as per specifications?

*You are to review the test plan you created for your own application. Test and record your results.*

Another factor that may be important is that you follow coding standards as prescribed by the company.

The easiest way to do this is to develop a test plan for each screen.

**Sample testing strategy:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Item | Test Data | Expected Result | Actual Result | Description of test (if applicable) |
| Menu System | 1 | Goes to method to input book details | √ |  |
| 45 | Displays “Invalid input” | √ |  |
| “abc” | Displays “Invalid input” | x | Program crashes – need new code to validate for such values |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Item Input Book Details | Test Data | Expected Result | Actual Result | Description of test (if applicable) |
| Author’s name | 123456 |  |  |  |
| “Smith” |  |  |  |
| *[left blank]* |  |  |  |
|  |  |  |  |

***You must also provide evidence of using a debugger and examine variable contents to detect errors. This may be included in your testing in the form of screen dumps.***

*NOTE: Add your own test data to the tables above and create new tables for the other Book inputs of this program.*

*Apply appropriate coding standards including: comments, indentation style and naming conventions for Classes, variables, constants and methods etc.*

***When complete zip up your whole directory project and upload to the appropriate Dropbox   
including the testing documentation on Brightspace.***

# Assessment Submission Cover Sheet (VET)

Student to complete relevant sections and attach this cover sheet to each assessment task for submission.

|  |  |  |  |
| --- | --- | --- | --- |
| Student information | | | To be completed by Student |
| **Student name:** | Lucas Beyer | **Student ID:** | 100623374 |

| Program/Course details | | | |
| --- | --- | --- | --- |
| **Qualification code:** | ICT40120 | **Qualification title:** | Certificate IV in Information Technology (Programming) |
| **Unit code:** | Python Programming  ICTPRG302   ICTPRG435 | **Unit title:** | Python Programming  Apply introductory programming techniques  Write scripts for software applications |
| **Department name:** | BDIT, Computing & Information Technology | **CRN number:** | 54602 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Assessment information | | | To be completed by Student | | | | |
| **Teacher name:** | Hridya Ranjith | | | | | | |
| **Due date:** | Session 9 | **Date submitted:** | | | 5/12 | **Re-submission:** |  |
|  | | | | | | | |
| **Student declaration** | | | |  | | | |
| By submitting this assessment task and signing the below, I acknowledge and agree that:   1. This completed assessment task is my own work. 2. I understand the serious nature of plagiarism and I am aware of the penalties that exist for breaching this. 3. I have kept a copy of this assessment task. 4. The assessor may provide a copy of this assessment task to another member of the Institute for validation and/or benchmarking purposes. | | | | | | | |

|  |  |
| --- | --- |
| **Student signature**  For electronic submissions: By typing your name in the student signature field, you are accepting the above declaration. | **Lucas Beyer** |

**Note:**

**Assessor to attach a photocopy of the completed Marking Guide (Section E) from the Student version of the Assessment Task.**

**Final result of the subject/unit will be entered on Banner by the teacher/assessor once all assessment tasks have been assessed.**