Name: ……………………………………………….. ( ) Class: ……… Date: …………………..

|  |  |  |
| --- | --- | --- |
| **4.1** | **Program Development** | **Data Validation, Test Cases, Program Errors, Debugging** |

**Data Verification**

Data verification is the process of ensuring that entered data matches the original data exactly, such as asking for a password to be entered twice. It is typically performed by a human, not a computer.

**Data Validation**

Data validation is the process of ensuring that data satisfies a given set of requirements using a computer. In general, it is different from data verification and there are four types of checks that can be performed in data validation:

|  |  |
| --- | --- |
| **Type of Check** | **Common Feature(s)** |
| Length checks | Typically make use of the len() function |
| Range checks | Typically make use of the <, <=, >, >= operators |
| Presence checks | Typically compare data to values such as None or an empty string "" |
| Format checks | Typically make use of the index and slice operators as well as string methods such as str.isalpha() and str.isnumeric() |

In general, if data validation fails and that data was entered by the user, we should always ask for the data to be re-entered.

1. The following is a simple program that asks the user for an integer and outputs the integer multiplied by 2:

|  |  |
| --- | --- |
| 1  2 | **number = int(input("Enter an integer: "))**  **print(number \* 2)** |

The following are two examples of how the program behaves with valid data:

|  |
| --- |
| Enter an integer: **2017**  4034 |
| Enter an integer: **-1965**  -3930 |

However, the program crashes when invalid data is entered:

|  |
| --- |
| Enter an integer: **seventeen**  Traceback (most recent call last):  File "C:/Examples/Q1.py", line 1, in <module>  number = int(input("Enter an integer: "))  ValueError: invalid literal for int() with base 10: 'seventeen' |

Add data validation to the program so that its behaviour with valid data is unchanged while it should print “Data validation failed!” and ask again if invalid data is entered. The input string is valid only if contains at least one numerical digit, its first character is either a minus sign or a numerical digit, and the remaining characters are numerical digits only (no commas or other separators allowed):

|  |
| --- |
|  |

1. Write the input entry and validation code for a program that needs to accept a phone number that has the format “XXXX-XXXX”, where X represents a numerical digit. If the input entered via the keyboard is invalid, your input validation code should keep trying by asking for the input to be entered again. (You do not need to write the remainder of the program.)

|  |
| --- |
|  |

**Test Cases**

Test cases are needed to detect errors and imperfections within our programs. Test cases should cover three types of conditions:

|  |  |
| --- | --- |
| **Type of Conditions** | **Description** |
| Normal conditions | Situations where input data follows what is expected during normal use of the program |
| Boundary conditions | Situations where input data is at the limit of what the program is designed to cope with or where special handling is required |
| Error conditions | Situations where input data would normally be rejected by the program |

Input data under normal conditions is called **normal data**, while input data under error and boundary conditions are called **abnormal data** and **extreme data** respectively.

1. You are asked to write a program that accepts a positive integer and outputs the sum of its digits. Complete the expected output of the following two test cases that cover normal conditions for this problem:

|  |  |
| --- | --- |
| **Test Case 1 for Normal Conditions** | |
| **Expected Input** | 2017 |
| **Expected Output** |  |

|  |  |
| --- | --- |
| **Test Case 2 for Normal Conditions** | |
| **Expected Input** | 1965 |
| **Expected Output** |  |

Design two test cases that cover error conditions for this problem. (You may choose your own expected error messages.)

|  |  |
| --- | --- |
| **Test Case 1 for Error Conditions** | |
| **Expected Input** |  |
| **Expected Output** |  |

|  |  |
| --- | --- |
| **Test Case 2 for Error Conditions** | |
| **Expected Input** |  |
| **Expected Output** |  |

1. You are asked to write a program that accepts a 24-hour time in the format “HH:MM” where HH represents the hours part (00 to 23) and MM represents the minutes part (00 to 59) of the time. The program then outputs the number of minutes left until the next midnight as a sentence. The following are two test cases that cover normal conditions for this problem:

|  |  |
| --- | --- |
| **Test Case 1 for Normal Conditions** | |
| **Expected Input** | 22:00 |
| **Expected Output** | There are 120 minutes left until the next midnight. |

|  |  |
| --- | --- |
| **Test Case 2 for Normal Conditions** | |
| **Expected Input** | 12:34 |
| **Expected Output** | There are 686 minutes left until the next midnight. |

Design three test cases that cover boundary conditions for this problem. (Some of these boundary conditions may also be error conditions. You may choose your own expected error messages for these test cases, if any.)

|  |  |
| --- | --- |
| **Test Case 1 for Boundary Conditions** | |
| **Expected Input** |  |
| **Expected Output** |  |

|  |  |
| --- | --- |
| **Test Case 2 for Boundary Conditions** | |
| **Expected Input** |  |
| **Expected Output** |  |

|  |  |
| --- | --- |
| **Test Case 3 for Boundary Conditions** | |
| **Expected Input** |  |
| **Expected Output** |  |

**Program Errors**

1. Complete the following descriptions for each type of program error:

|  |  |
| --- | --- |
| **Type of Error** | **Description** |
| Syntax errors | * Detected by ……………………………, such as:   + **Compilers**, which store the ………………………………………………… on the computer such that the …………………………………………… and …………………………………………… are no longer needed to run the program, or   + **Interpreters**, which ………………………………………………… the …………………………………………… while the program is running so both the ………………………………… and …………………………………… are still needed to run the program * Caused by …………………………………………………… or the incorrect …………………………………………… of symbols in source code |
| Run-time errors | * Occur while the program is running and may cause the program to ……………………………………… or ……………………………………… * Caused by:   + Incorrect use of ……………………………………………………   + Input data that has not been properly …………………………………   + Conditions outside of the program’s ………………………………… |
| Logic errors | * Detected when the ……………………………………………… of a program does not match the ………………………………………………… * Caused by use of an incorrect or incomplete …………………………………… |

**Debugging**

1. The following is a simple program that asks for a message and attempts to output the number of punctuation marks in the message:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **message = input("Enter a message: ")**  **punctuation\_count = 0**  **for character in message:**  **if not character.isalnum():**  **punctuation\_count += 1**  **print(punctuation\_count)** |

However, this program contains a logic error such that its actual output does not match the expected output for the following message:

|  |  |
| --- | --- |
| **Test Case** | |
| **Expected Input** | Hello, World! |
| **Expected Output** | 2 |
| **Actual Output** | 3 |

Use print statements to detect which character in the message causes the actual output to be different from the expected output.

What character is responsible for the discrepancy? ……………

Modify the program so that this particular logic error no longer occurs. (You do not need to fix the remaining errors in the program, if any.)

|  |
| --- |
|  |

1. There are two variables: x and y that are assigned values as follows.

|  |  |
| --- | --- |
| >>>  >>> | **x = 3**  **y = 5** |

The following is a program that attempts to swap the values of the two variables. However the

results are incorrect.

|  |  |
| --- | --- |
| >>>  >>>  >>>  >>> | **x = y**  **y = x**  **x**  5  **y**  5 |

Use print statements to determine what causes the actual output to be different from the expected output. Explain why this happened.

……………………………………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………………………………

Modify the program such that the values are swapped.

|  |
| --- |
|  |

1. The following is an erroneous program that accepts an integer between 1 and 50, then attempts to output a right-angled triangle with a number of lines equal to the input integer:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **lines = int(input("Input integer between 1 and 50: "))**  **part1 = 0**  **part2 = 0**  **for line in range(lines):**  **part1 += 1**  **part2 += 1 / 8**  **print('\*' \* int(part1 + part2))** |

The program works correctly for the input 5:

|  |
| --- |
| Input integer between 1 and 50: **5**  \*  \*\*  \*\*\*  \*\*\*\*  \*\*\*\*\* |

However, its actual output does not match the expected output for the input 10:

|  |
| --- |
| Input integer between 1 and 50: **10**  \*  \*\*  \*\*\*  \*\*\*\*  \*\*\*\*\*  \*\*\*\*\*\*  \*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\* |

Use print statements to detect what the values of part1 and part2 are when the actual output starts to differ from the expected output for this input.

What is the value of part1 when the output starts to differ (for input 10)? …………………………

What is the value of part2 when the output starts to differ (for input 10)? …………………………

(You do not need to fix the program.)