**How Many Times (Exception Handling)**

**Purpose**

The purpose of this assignment is to reinforce the importance of robust input validation and exception handling in secure software development. The user interaction helps students learn how to create programs that anticipate and handle unexpected input, reducing the likelihood of runtime errors and enhancing software reliability.

**Objectives**

* Develop a script that collects user input and performs numeric operations.
* Practice implementing try-except blocks for error handling.
* Recognize and respond appropriately to invalid input.
* Demonstrate how to gracefully terminate a program when errors occur.
* Reflect secure coding principles in line with the Secure Software Development Life Cycle (SSDLC).

**Learning Outcomes – KA?**

* Use exception handling constructs (try, except, finally) to manage invalid inputs.
* Apply input validation techniques to ensure data integrity.
* Format numeric output using string formatting (e.g., displaying averages to two decimal places).
* Understand the role of exception handling in writing secure, stable, and user-friendly code.
* Demonstrate debugging and problem-solving skills in response to runtime errors.

**Competency Statement**

The assignment titled “How Many Times” is designed to synthesize and assess a comprehensive set of competencies gained from secure scripting and programming instruction. It serves as a critical checkpoint in the course, requiring students to apply theoretical knowledge and practical skills in the context of a realistic programming challenge. Specifically, this assignment functions as an informal code review exercise grounded in the principles of the Secure Software Development Life Cycle (SSDLC), providing students with an opportunity to engage with secure development practices early and meaningfully.

In completing this assignment, students are tasked with developing a Python script that demonstrates the effective use of input validation and exception handling—two essential techniques for building secure and fault-tolerant applications. The goal is not only to meet the functional requirements of the script but also to proactively mitigate common input-related vulnerabilities, such as injection attacks, buffer overflows, or logic errors caused by unexpected input. By addressing both functionality and security, students exhibit their ability to write robust code that aligns with industry practices in secure software development.

Furthermore, this activity emphasizes the importance of code resilience and quality assurance, reinforcing critical thinking and problem-solving skills as students analyze the behavior of their scripts under various input conditions. The assignment ultimately enables students to demonstrate their proficiency in producing maintainable and secure code, thus reflecting a holistic understanding of secure scripting principles and practices.

| Competency Statement (ABCDE Model) | |
| --- | --- |
| Actor | Given students in a secure scripting/programming course |
| Behavior | will analyze, design, and implement a script |
| Condition | that includes input validation and exception handling as part of a “How Many Times” assignment within the context of the Secure Software Development Life Cycle (SSDLC) |
| Degree | to meet defined functional requirements and mitigate common input vulnerabilities |
| Evidence | as demonstrated through a code review checkpoint that verifies the script’s ability to maintain secure and fault-tolerant behavior |

**Tasks**

1. Write a program called how\_many\_times.py, the program shall ask the user to enter two numbers.
2. The program must output the addition and average with two decimal places of the entered numbers.

e.g.,

Enter a whole number: 1

Enter a whole number: 1

Addition: 2

Average: 1.0

Terminating...

1. The program must handle invalid input data properly.

e.g.,

Enter a whole number: 1

Enter a whole number: 1.1

Invalid input. Only whole numbers are allowed

Terminating...

**How Many Times (Loops)**

**Purpose**

The purpose of this assignment is to reinforce the understanding of loops, conditional statements, and input validation. It also introduces how to properly structure a loop that continues until a user-defined condition is met.

**Objectives**

* Implement a loop that repeatedly prompts the user for input.
* Validate user input and reject non-integer values with appropriate error messages.
* Perform basic arithmetic operations (sum and average) on valid inputs.
* Format and display results with proper decimal precision.

**Learning Outcomes**

* Use while loops to continuously prompt users until a condition is met.
* Use try-except blocks to handle invalid input without crashing the program.
* Accumulate numeric input and compute the average.
* Format output using string formatting for precision.
* Enhance user experience with informative error and exit messages.

**Competency Statement**

The assignment “How Many Times” compiles a series of knowledge from secure scripting/programming that allows the student to demonstrate their competency in writing scripts using loops and exception handling to manage user input and perform arithmetic processing. This assignment supports the development of core programming logic, input validation, and formatted output skills. Students apply steps from the Software Development Life Cycle (SDLC), including requirements analysis, implementation, and testing, while designing and developing the solution.

**Tasks**

1. Write a program called how\_many\_times.py. The program shall continually prompt the user to enter whole numbers until the user enters “exit”.
2. The program must output the addition and average with 2 decimal places of the entered numbers.
3. The program must handle invalid input properly.

e.g.,

Enter a whole number ('exit' to quit): 1

Enter a whole number ('exit' to quit): 1

Enter a whole number ('exit' to quit): 1.1

'1.1' is invalid. Only whole numbers are allowed

Enter a whole number ('exit' to quit): a

'a' is invalid. Only whole numbers are allowed

Enter a whole number ('exit' to quit): exit

Addition: 2

Average: 1.0

**How Many Times (Functions)**

**Purpose**

The purpose of this assignment is to develop proficiency in creating modular scripts using functions. Students build a user interactive application that gathers input, validates it, performs calculations, and returns formatted results. The assignment reinforces the importance of separating logic into well defined functions to promote readability, reusability, and maintainability.

**Objectives**

* Design a Python program using user-defined functions to handle input collection, validation, computation, and output.
* Demonstrate the use of while loops and try-except blocks within function definitions for error handling.
* Convert and process user input through explicit type casting and conditional logic.
* Format and present numerical output (sum and average) with two decimal places.
* Apply the Software Development Life Cycle (SDLC) by planning, implementing, testing, and refining a solution.

**Learning Outcomes**

* Break down a program into logical, well scoped functions.
* Use loops and conditionals effectively within function based designs.
* Handle exceptions using try-except-finally blocks inside functions.
* Calculate the sum and average of valid whole number inputs with correct formatting.
* Structure code in a way that aligns with good software engineering practices, including modularization and testing.

**Competency Statement**

The assignment “How Many Times” compiles a series of knowledge from secure scripting/programming that allows the student to demonstrate competency in writing modular scripts that use functions to structure logic and handle tasks such as input validation, arithmetic processing, and formatted output. Students showcase the ability to design, implement, and test programs in alignment with the Software Development Life Cycle (SDLC), emphasizing clean function design, user interaction, and error handling.

**Tasks**

1. Write a program called how\_many\_times.py. The program shall continually prompt the user to enter whole numbers until the user enters “exit”.
2. Write a function called is\_numeric()that validates if the input is a whole number. The function shall return true if the input is valid, false otherwise.
3. The program shall output the addition and average with 2 decimal places of the entered numbers.
4. The program must handle invalid input properly.

e.g.,

Enter a whole number ('exit' to quit): 1

Enter a whole number ('exit' to quit): 1

Enter a whole number ('exit' to quit): 1.1

'1.1' is invalid. Only whole numbers are allowed

Enter a whole number ('exit' to quit): a

'a' is invalid. Only whole numbers are allowed

Enter a whole number ('exit' to quit): exit

Addition: 2

Average: 1.0

**How Many Times (Lists)**

**Purpose**

The purpose of this assignment is to help students develop skills in managing collections of data using lists. Through user interaction, students will collect, store, and categorize both valid and invalid inputs to use that data to compute results. The assignment reinforces concepts such as input validation, list operations, data organization, and formatted output.

**Objectives**

* Build a script that accepts and processes user input using while loops and input validation.
* Store valid and invalid inputs in separate lists for later display.
* Compute and display the sum and average of the valid numbers entered, formatted to two decimal places.
* Provide a clear and informative output of valid and invalid inputs.
* Follow a structured development process based on the Software Development Life Cycle (SDLC), including planning, implementation, and testing.

**Learning Outcomes**

* Use lists to store and manage collections of user input.
* Differentiate between valid and invalid data using conditional logic and exception handling.
* Apply try-except blocks to manage runtime errors during input processing.
* Calculate and present numerical output such as the sum and average with appropriate formatting.
* Organize and present both raw and processed data in a user friendly format.
* Apply software development best practices, including modular code structure and testing phases aligned with the SDLC.

**Competency Statement**

The assignment “How Many Times” compiles a series of knowledge from secure scripting/programming that allows the student to demonstrate their ability to create list driven scripts that manage and process categorized input data. Students effectively apply control structures, exception handling, and list operations to support user interaction and generate formatted results. This assignment emphasizes problem solving through the use of the Software Development Life Cycle (SDLC), reinforcing the importance of planning, developing, testing, and refining software solutions.

Planning is applied as students outline the script structure in advance, identifying key requirements such as how to validate input, store valid/invalid entries, and format output.

Development is implemented in the core of the code, where students write the script using loops, conditionals, exception handling, and list manipulation to carry out the specified logic.

Testing occurs during the iterative process of running the script with different inputs (valid, invalid, and edge cases) to ensure the logic functions as expected and that errors are handled gracefully.

Refinement happens as students improve the structure and clarity of their code, modularizing sections into functions if necessary, cleaning up output, or improving feedback messages to users based on test results.

**Tasks**

1. Write a program called how\_many\_times.py. The program shall continually prompt the user to enter whole numbers until the user enters “exit”.
2. The program shall output the addition and average with 2 decimal places of the entered numbers. Valid and invalid input must be stored in separate lists. Valid data shall be displayed first, followed by the arithmetical operations, and lastly, the invalid list.
3. The program must handle invalid input properly.

e.g.,

Enter a whole number ('exit' to quit): 1

Enter a whole number ('exit' to quit): 1

Enter a whole number ('exit' to quit): 1.1

'1.1' is invalid. Enter a whole number

Enter a whole number ('exit' to quit): a

'a' is invalid. Enter a whole number

Enter a whole number ('exit' to quit): exit

Valid numbers that were entered: 1, 1

Addition: 2

Average: 1.0

Invalid numbers that were entered: 1.1, a