Input Validation Selection

Target Course

CS1 (imperative)

Learning Goals

A student shall be able to:

1. Apply principles of secure design and defensive programming techniques when developing software.

IAS Outcomes

The CS2013 Information Assurance and Security outcomes addressed by this module are:

IAS Knowledge Topic	Outcome
Defensive Programming	3. Classify common input validation errors, and write correct input validation code. [Usage] 5. Demonstrate the identification and graceful handling of error conditions. [Usage]
Principles of Secure Design	2. Summarize the principle of fail-safe and deny-by-default. [Familiarity]

Dependencies

- Cover at the same time as selection.
- Optional material assumes knowledge of functions.

Summary

Demonstrate input validation using selection on a simple example.

Estimated Time

[Provide the estimated amount of lecture time to cover this module, using the notion of time as defined in CS2013.]

Materials (Python imperative)

This material demonstrates input validation via selection using the following example problem:

> Write a program that allows the user to enter their weight and height and displays their Body Mass Index (BMI) category.

The questions below can be used to lead an in class discussion. Sample solutions to the problem above are listed at the end of the section.

What input validation is reasonable for the above problem?

- Both height and weight should be numbers
- Height and weight should be reasonable values (e.g. weight between 0 and 1000 and height between 1 and 100). The range check can also help prevent integer overflow/underflow errors.
- Height should not be zero. This would lead to a divide by 0 error in this example.

Should we do validation via the whitelist or blacklist approach?

The two primary approaches to validating input are whitelisting and blacklisting. Blacklisting involves checking if the input contains unacceptable data while whitelisting checks if the input contains acceptable data. Whitelisting is usually preferred since blacklisting relies on programmers anticipating all possible unexpected data [1]. In addition, whitelisting agrees with

the *deny-by-default* design principal, which states that everything not explicitly permitted is by default forbidden.

Do the checks on height and weight listed above adhere to the whitelisting or blacklisting approach?

Whitelisting. Each bullet point above is describing a quality that the field must satisfy.

What should be done with erroneous inputs?

One possibility is to reject the input. An alternative is to attempt to fix an erroneous input; e.g. if "one hundred" is entered as the weight we can convert it to the number 100.

According to the **fail-safe** design principal erroneous inputs should be immediately rejected and then program should stop processing the request. While this approach is less user friendly than the approach of fixing the input, it is meant to resist the type of attack where a malicious user contrives an input which when fixed is actually an unauthorized command. Reference [1] gives the following example, where naive filtering for the specific tag <script> leaves behind valid and malicious code.

```
<scr<script>ipt>alert(document.cookie);</scr<script>ipt>
```

Thus for now, when erroneous inputs are detected our approach will be to inform the user and terminate the program. Later we will learn (using iteration) how to allow the user to re-enter in values until a correct value is entered. This approach avoids having to terminate the entire program and potentially losing work that the user has already put in. In either case the erroneous inputs are not used by the program.

Sample solution to the BMI problem without functions

```
#pre:  User ready to enter weight and height.
#post: Displays the BMI category corresponding to inputs
def main():
    weight = float(input("Enter your weight in pounds: "))

    if weight<=0 or weight>1000:
      print("Weight must be a number in range (0,1000]. Goodbye.")
    else:
      height = float(input("Enter your height in inches: "))
      if height<=0 or height>100:
            print("Height must be a number in range (0, 100]. Goodbye.")
      else:
            bmi=weight / height ** 2 * 703;
            result = None
            if bmi <= 18.5:
                  result = "underweight"
            elif bmi <= 25:
                  result = "normal"
            elif bmi <= 30:
                  result = "overweight"
            else:
                  result = "obese"
```

```
            print("Your BMI category is" , result)
```

Optional Material (assumes functions have been discussed)

How can the use of functions help us with input validation?

Input validation can be a lengthy and somewhat error prone process. Designing functions for some of the common steps can help. Design each function to do a single task (separation of concerns) and design functions so that they are useful in different scenarios. For example

- Design a separate function to obtain different types from a user (integers, decimals, strings)
- Design a function to validate that a given value falls within a certain range
- Design a function to check the length of a given input
- Create a separate function to manage the validation of each input. Then all the checks for one input value occur in a single place.

Sample solution to BMI that uses functions

```
#pre:  User ready to enter a number.
#post: Returns number entered by user
def getNumber(prompt):
    return num = float(input(prompt))
```

```
#pre:  userInput needs to be validated to be inside range  (min, max]
#post: Returns True when userInput is a number and min< userInput<=max.
#      Otherwise, returns False.
def isInputValid(userInput, min, max):
    if userInput > min and userInput <= max:
        return True
    else:
        return False
```

```
#pre:  0 < weight <= 1000, 0< height <=100
#post: Returns category that the BMI corresponds to
def getBMICategory(weight, height):
    bmi=weight/height**2 *703;
    result = None
    if bmi <= 18.5:
        result = "underweight"
    elif bmi <= 25:
        result = "normal"
    elif bmi <= 30:
        result = "overweight"
    else:
        result = "obese"
    return result
```

```
def main():
    weight = getNumber("Enter your weight in pounds: ")
    if not isInputValid(weight, 0, 1000):
        print("Weight must be a number in range (0,1000]. Goodbye.")
    else:
```

```
height = getNumber("Enter your height in inches: ")
if not isInputValid(height, 0, 100):
     print("Height must be a number in range (0, 100]. Goodbye.")
else:
     result = getBMICategory(weight, height)
     print("Your BMI category is" , result)
```

Assessment Methods

Programming Problems

1. Write a program that allows the user to enter one whole number and one floating-point number. Perform the following validation of the two input numbers - the whole number is valid when its value is negative and the floating-point number is valid when its value is greater than or equal to 1.0. When both numbers are valid, perform the computation

   wholeNumber * 2 + floatingPointNumber * 2

   and then display its result. When either number is invalid, display an error message to the user.

2. Write a program that asks the user for a year value. The year value must be greater than zero. When the year value is invalid, display an appropriate error message. When the year value is valid, display a message to the user to indicate whether this year value is a leap year. To determine if a year is a leap year, do the following:
   a. For years less than 1582, a year is a leap year if it is divisible by 4.
   b. For years greater than or equal to 1582, a year is a leap year if it is divisible by 400 or if it is divisible by 4 but not 100.

   For example, 1600 and 2000 are leap years; 1896 and 1904 are leap years; but 1700, 1800, and 1900 are not leap years.

3. *Problem assumes iteration has been covered.* Write a program that asks the user for how many numbers they intend to enter. If the value entered is less than zero, display an appropriate error message. If the value entered is valid, use iteration (i.e., looping) to obtain that many numeric values from the user. These numeric values may be any value. After obtaining the correct number of numeric values from the user, compute and display the average of these numbers.

   For example, the user may indicate that they will enter 5 numbers. The user is prompted to enter each number, resulting in -10, 5.5, 100, -1.1, and 13 being entered. Your program would then display 21.48 as the average.

   Note: students would need to check for a divide-by-zero to avoid an arithmetic error.

4. A professor gives a 100-point exam that is graded on the following scale:

90-100: A
80-89: B
70-79: C
60-69: D
<60: F

Have the user enter one number representing the exam score. Ensure that the exam score number is valid (i.e., it is an integer and it is in the range [0,100]). Display an appropriate error message when the exam score is not valid. Otherwise, display the exam score and corresponding exam grade to the user.

5. *Test whether students can identify the need for validation even though it is not explicitly stated in the problem statement by changing the end of the problem description above to be:* Have the user enter one number representing the exam score. Display the exam score and corresponding exam grade to the user.

6. Write Python code that solves the following problem statement.

   Obtain two whole numbers (called w1 and w2), one real number, and a string value that represents the users' name. On a single output line, display the name followed by the real number followed by the two whole numbers. Perform the following computations and display each result on a separate output line.

   Compute the sum of all three numbers.

   Compute the product of all three numbers.

   Compute w1w2.

   Divide the product of all three numbers by the sum of all three numbers.

   Compute w1w2 divided by the real number.

7. Write Python code that solves the following problem statement.

   Obtain two numbers from the user. Perform the following computations and display each result on a separate output line.

   Compute the sum of the two numbers.

   Compute the product of the two numbers.

   Divide the product of the two numbers by the sum of the two numbers.

8. Write Python code that solves the following problem statement.

   Obtain numeric values from the user until the user enters "exit" (without the quotes). Compute and display the sum and average of all negative numbers entered. Compute and display the sum and average of all positive numbers entered. Note that zero is neither negative nor positive.

Multiple Choice

9. Which of the following statements are false?
   a. A Parse Error occurs when you have a syntax error in your code.
   b. A Type Error occurs when the operands of an operator are not the correct data type.
   c. A Value Error occurs when an object of one type is passed as a parameter to a function but the function is expecting an object of another type.
   d. All of the above.
   e. None of the above.

   ANSWER: E. (The statements in a, b and c are all TRUE!)

10. Which of the following is the most appropriate use of a type check when writing input validation code in Python?

a. When the code is expecting the user to enter a numeric value.
b. When the code is expecting the user to enter a negative number.
c. When the code is expecting the user to enter a string value.
d. All of the above.
e. None of the above.

ANSWER: a

11. Which of the following statements are true? (There may be multiple answers to this question!)
    a. Range checking is a common approach for validating input data. It is most often used to ensure that a numeric value is within a range of possible values.
    b. Checking for arithmetic errors is a common approach associated with validating input data. An example of this is to ensure that a divide by zero error will not occur.
    c. When an input validation error is detected by the code, the code should attempt to intrepret the data into something meaningful so that processing may continue. The code should not force the user to re-enter the data.
    d. Writing code that checks the format of data entered by a user is another type of input validation. An example of this is to ensure that a U.S. social security number has exactly nine digits.
    e. Input validation is an important defensive programming strategy as it allows a malicious user to intentionally craft input data that may lead to a security vulnerability.

    ANSWERS: a, b

True/False

12. True or False? A Name Error always means that you have used a variable name before it has a value.

    ANSWER: False. (A Name Error will occur if a word (i.e., a single string token) is entered when a numeric value is expected. Python will try to “evaluate” the token as a variable name.)

References

[1] Survive the deep end: PHP Security. Chapter on Input Validation. Retrieved on August 2015, from http://phpsecurity.readthedocs.org/en/latest/Input-Validation.html.