CMPT 120

Topic: Python’s building blocks
- More Statements
Step 5 – Testing
- Test Cases and Errors
Last Lecture

- We learnt the following Python statements, i.e., Python’s building blocks by solving a problem using the 4 steps of the software development process
  - Categories of Statements
    1. Assignment statement
    2. Input statement
    3. Output statement
    4. Operational statements
  - + conversion built-in functions
Last Lecture – We Practised!

Software Development Process

1. Does my Python program execute?
2. Does it solve the problem? No!
3. And create test cases.

4. Implementation (Coding)
   - Python program
     - Example: `Average_of_5_Midterms.py`

5. Testing
   - Using Python IDLE Program Editor

Compute the average of 5 mid term exams.
Learning outcomes

At the end of this course, a student is expected to:

• Describe and **apply** (i.e., use in s/w dev.) fundamental concepts and terminology of Python:
  • Expressions
  • Operators
  • Statements

• Create (design) simple algorithms:
  • **Solve problems by designing** simple algorithms, e.g., basic calculations

• **Test** small size programs in Python:
  • Test boundary case
  • Describe and apply techniques to debug programs:
    • Recognize and categorize errors as syntax, runtime, semantic
Today’s Menu

• Continue learning Python statements (Python’s building blocks)
  • Categories of Statements
    4. Operational statements
      • Mathematical/arithmetic operators
        • Order of operations
        • Augmented assignment operators
      • Function terminology
  as we practice Step 5 of the software development process -> Testing and Debugging
  • 3 kinds of errors
1. Does my Python program execute?
   • Yes? -> Youpi! 😊
   • No? -> If it does not execute because there is an error (a bug), then we need to debug it
     • Debugging: identifying and correcting errors
     • Once we have debugged our program, we can move on to Question 2
   • What could go wrong?
Error #1 - Syntax error

- Occurs when our Python code does not follow the Python syntax rules

- Our textbook says that …
  - if there is a syntax error anywhere in our program, Python Interpreter displays an error message and quits, and we will not be able to run the program

- Example:
Test your knowledge of Syntax Error

- Looking at the software development process and its 4 steps ...
  - At which step would a syntax error be detected?
  - In which step(s) would we fix a syntax error?
  - How do we fix a syntax error?
BTW, where does the word “bug” come from?

A bit of history: Grace Hopper and UNIVAC

Source: https://upload.wikimedia.org/wikipedia/commons/3/37/Grace_Hopper_and_UNIVAC.jpg
From a bug found in UNIVAC

• From Grace Hopper’s notes:

Source: https://mostinterestingfacts.files.wordpress.com/2009/04/bug.jpg
Step 5. Testing and Debugging

2. Does it solve the problem?
   • We answer this question by testing our Python program using test cases
Test Case

• **Test case** contains
  
  1. **Test data**: specific value
  2. **Expected result**: must be computed beforehand (before we start our testing)
     • Before feeding the test data into our program, we must know what the expected result will be otherwise we will be unable to ascertain whether our program works or not
Example of test cases for our `Average_of_5_Midterms.py`

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Part 1 Test Data</th>
<th>Part 2 Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>all 5 midterms = 8.0</td>
<td>8.00</td>
</tr>
<tr>
<td>2</td>
<td>mt1 = mt2 = mt3 = mt4 = mt5 =</td>
<td></td>
</tr>
</tbody>
</table>

Can you think of another test case?
Observations about the creation of test cases
Step 5. Testing and Debugging

• We test our Python program using 1 test case at a time
  • We feed the test data of the test case into our executing program
  • We compare its actual results (perhaps displayed on the computer monitor screen) with the expected results of the test case
    • If actual results == expected results -> Youpi! 😊
    • If they are not, then we need to debug
    • The error could be in …
      • Our program
      • Our expected results
    • Once we debugged, we test our program again with all the test cases we have used so far

• What could now go wrong?
Error #2 - Runtime error

- Our textbook defines a runtime error as follows:
  - The second type of error is a runtime error, so called because the error does not appear until after the program has started running (i.e., at runtime)

- These errors are also be called exceptions because they usually indicate that something exceptional (and bad) has happened
Runtime error - Example

It may be the case that the data assigned to our variables are such that an error occurs

• Example:
  
  ```python
  sum = float(input("Please enter the sum: "))
  # Let’s imagine the user enters 2635
  numOfStds = float(input("Please enter number of students: "))
  # Let’s imagine the user enters 0
  average = sum / numOfStds
  ```

• Result:
  
  Traceback (most recent call last):
      File "<pyshell#2>"", line 1, in <module>
        sum / numOfStds
  ZeroDivisionError: integer division or modulo by zero
Fixing Syntax and Runtime Errors

• Fixing syntax and runtime errors can be tricky because it requires us to understand Python Interpreter’s error messages
  • **Trick:** become familiar with Python Interpreter’s error messages (i.e., acquire experience by using Python as much as possible)
Test your knowledge of Runtime Error

- Looking at the software development process and its 4 steps ...
  - At which step would a runtime error be detected?
  - In which step(s) would we fix a runtime error?
  - How do we fix a runtime error?
Error #3 - Semantic error

• Our textbook defines a semantic error as follows:
  • If there is a semantic error in our program, it will run without generating error messages, but it will not produce the expected (correct) result
  • So, in a sense, the meaning of the program is erroneous as it does not do what its intended purpose (its description) said it would

• The bottom line: our Python program will not solve the problem!!!
Semantic error - Example

- Can you think of an example?
Fixing Semantic Errors

• Identifying semantic errors (debugging) can be tricky because it requires us to hand trace the program trying to figure out where it (we) went wrong
  • Trick: work backward
Test your knowledge of Semantic Error

• Looking at the software development process and its 4 steps ...
  • At which step would a semantic error be detected?
  • In which step(s) would we fix a semantic error?
Back to Python statements

• Categories:
  1. Assignment statement
  2. Input statement
     • Conversion function
  3. Output statement
  4. Operational statements
     • Mathematical/arithmetic operator
     • String manipulation operator
      • Next lecture!
  5. Conditional statement
  6. Iterative statement

Some of them are built-in function or method

Today

We’ll see these statements soon!
Order of operations

- Python evaluates arithmetic operators in a predefined order:
  - Rule of precedence -> PEMDRAS

- **P** -> parentheses
- **E** -> exponential
- **M** -> multiplication
- **D** -> division
- **R** -> remainder
- **A** -> addition
- **S** -> subtraction

These 3 operators have the same level of precedence -> when several: *left to right*

These 2 operators have the same level of precedence -> when several: *left to right*
Order of operations

• What happens when we evaluate:

• When an expression is composed of several ...
  • Parentheses
    we first evaluate the expression in the inner most pair of parentheses and we continue evaluating expressions as we work our way to the outer most pair of parentheses
  • Exponential operators
    these operators are evaluated from right to left
  • Multiplication, Division, and/or Remainder operators
    these operators are evaluated from left to right
  • Addition and/or Subtraction operators
    these operators are evaluated from left to right
Let’s Practise!
Revisiting

1. Assignment statement:

- Augmented assignment operators:
  \( +=, -=, *=, /=, //=, **=, %= \)
- **Syntax**: \(<\text{operand1}> < \text{operator} > <\text{operand2}>\)
- How it works:
  - The same way as:
    \(<\text{operand1}> = <\text{operand1}> < \text{operator} > <\text{operand2}>\)
    OR
    \(<\text{operand1}> = <\text{operand2}> < \text{operator} > <\text{operand1}>\)
- Example:
  - \(\text{finalGrade} = \text{finalGrade} + \text{newGradeAssn1}\)
    evaluates to same result as
  - \(\text{finalGrade} += \text{newGradeAssn1}\)
Functions - Terminology related to functions

1. We **call** a function by name
   - Example of a **function call**:
     
     ```
     userName = input("Please, enter your name: ")
     ```

2. The **name of the function** is **input**

3. The expression in parentheses is called the **argument of the function**
   
   ```
   userName = input("Please, enter your name: ")
   ```

4. We say that a function **takes** an argument(s)

5. The **result of the function** is what the function produces
   - In the above example, the **result of the function** is what the user has typed, i.e., her/his name, which is assigned to `userName`
     
     `Anne`

6. We say that a function **returns** a result and the result is called the **returned value**
type( ): another useful function

- **Built-in function**
- **Syntax:** `type(<expression>)`
- **How it works:**
  1. The expression is evaluated
  2. `type(...)` is called with the result of the expression as its argument
  3. `type(...)` returns a value which is the data type of its argument, i.e., the result of the expression

- **Examples:** (from Lab 1)
  - `type(123)`
  - `type("Hello")`
  - `type("123")`
  - `pi = 3.14`
  - `type(pi)`
round() built-in function

- **Built-in function**
- **Syntax:** `round(<expression>[, <ndigits>])`
  - `<expression>` must evaluate to a float
  - `[, <ndigits>]`, if supplied, must evaluate to an int
- **Examples:**
Other useful built-in functions

<table>
<thead>
<tr>
<th>Method name</th>
<th>Use</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolute value</td>
<td>abs(number)</td>
<td>Return the absolute value of a number</td>
</tr>
<tr>
<td>maximum within a list of numbers</td>
<td>max(num1,num2, …)</td>
<td>Returns the maximum value among the numbers in the list</td>
</tr>
<tr>
<td>minimum within a list of numbers</td>
<td>min(num1,num2, …)</td>
<td>Returns the minimum value among the numbers in the list</td>
</tr>
<tr>
<td>round</td>
<td>round(num[,n])</td>
<td>Return the value num rounded to n digits after the decimal point. If n is omitted, it defaults to zero. The result is a floating point number.</td>
</tr>
</tbody>
</table>

Even though we may not talk about all Python built-in functions during our lectures, our task, as a Python software developer, is to become familiar with Python built-in functions:

https://docs.python.org/3.5/library/functions.html
Summary

• Continue learning Python statements (Python’s building blocks)
  • Categories of Statements
    4. Operational statements
      • Mathematical/arithmetic operators
        • Order of operations
        • Augmented assignment operators
      • Function terminology

as we practice Step 5 of the software development process -> Testing and Debugging
• 3 kinds of errors
Next Lecture

- Python Strings