Introduction to: Computers & Programming: Post-Midterm 1 Review

Adam Meyers
New York University
Summary

• Some Procedural Matters
• Review of Concepts from Test
• Review of Part 1
• Review of Part 2
• Lab Tomorrow
# Grading Curve

<table>
<thead>
<tr>
<th>Letter Grade</th>
<th>Raw Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>93 and Above</td>
</tr>
<tr>
<td>A-</td>
<td>88-92</td>
</tr>
<tr>
<td>B+</td>
<td>84-87</td>
</tr>
<tr>
<td>B</td>
<td>79-83</td>
</tr>
<tr>
<td>B-</td>
<td>75-78</td>
</tr>
<tr>
<td>C+</td>
<td>70-74</td>
</tr>
<tr>
<td>C</td>
<td>65-69</td>
</tr>
<tr>
<td>D</td>
<td>40-64</td>
</tr>
<tr>
<td>F</td>
<td>39 and Below</td>
</tr>
</tbody>
</table>
Raw Score vs Letter Score

• Only the Letter Score counts for purposes of the final grade, e.g., there is no difference between a 71 and a 73 if both are part of the same range.

• It is only worth haggling over your grade if:
  – the change in score will effect your letter grade, e.g.,
    • raw score is 91 (an A-) and the discrepancy is 2 points
    • discrepancy is worth 5-10 points
    • Etc.

• Of course, understanding everything you got wrong is important regardless of the grade.
Grading Considerations

• If Midterm 2 is significantly better than Midterm 1, Midterm 2 will count more towards the final grade than Midterm 1.
  – If it is a real outlier, I will not count it at all, e.g.,
    • If (1\text{st} Midterm == D) and (2\text{nd} Midterm == A) and (Final == A) and (Homework Grade == A):
      Final Grade = A

• In other words improvement over the course of the class can have a big influence on final grade. I have not figured out exactly how to quantify this, but it will be a major factor.

• **Main purpose of final grade:** An indicator of state of knowledge at the end of the class.

• **Administrative purposed:**
  – C is the minimum grade in this class if you want to take the JAVA programming class.
  – A or A- average is recommended if you want to be a CS major
Only 2/3 questions on Section 2

• Timing:
  – Easier to do 2 well within the time limit

• Which Ones Should I Count?
  – If you don't tell me, I have to pick somehow
Variable Assignment & Input Statements

- Input statement is a function
  - It takes one parameter, a message to a user
  - It returns one value, a string entered by a user
  - You can assign this value to a variable

**Correct:** Variable1 = input('Give me a Number")
  - ## Variable1 is set to whatever the user inputs

**Incorrect:** input('Give me a Number')
  - ## The input function is used, but the output returned by the statement is essentially thrown away

- It is usually not useful to redefine parameters with input statements, e.g.,
  - def tweeze_eyebrows(left_eyebrow,right_eyebrow):
    - left_eyebrow = input('Enter Left Eyebrow')
    - right_eyebrow = input('Enter Right Eyebrow')
    - etc. …..
if and elif: Understand Dependencies

- If answer1:
  
  ```python
  print(event1)
  ask(question2)
  if answer2:
    print(event2)
  elif answer3:
    print(event3)
  elif answer4:
    print(event4)
  ```

- If answer1 is True and answer2 is False, event3 will not be printed
- If answer1 is false, question2 will not be asked
Accumulator Variables

- def intersperse (string1,string2):
  
  big_string = " ## initialize big_string as empty
  
  if len(string1)==len(string2):
    
    for index in range(len(string1)):
      
      big_string=big_string+string1[index]+string2[index]
      
      print(big_string)
      
      return(big_string)
      
  def triangle_number(number):
  
    triangle_output = 0
    
    for num in range(1,number+1):
      
      triangle_output = triangle_output+num
      
      print(num,triangle_output)
      
    return(triangle_output)
While loops with Accumulators and Sentinel Variables

def ninety_nine_bottles():
    more = True
    num = 99
    while more:
        print(num,'bottles of beer on the wall')
        print(num, 'bottles of beer...if one of those bottles should happen to fall')
        num = num-1
        if num == 0:
            more = False
        else:
            answer = input('More?')
            answer = answer.lower()
            if answer == 'no':
                more = False
Other Ways to exit Loops

• return – exits function, will exit loop also
  def blah():
    while True:
      if …:
        return(157)

• Break – only returns from inner-most loop
  def blah_blah():
    for number in range(5000):
      if number >= 57:
        break
    print(number)
Using Non-Booleans as Booleans

• Most non-booleans can be equivalent to True
• The following exceptions which can be equivalent to False
  – The empty string ("")
  – The number zero (0)
  – range(0)
  – None (what a void function like print returns)
  – And a small number of others objects, all of which play a similar role in their object-type
Boolean Expressions containing Non-Booleans

- Non-Boolean behave with connectives and control structures as per previous slide
- False or 'No' → evaluates as 'No'
- if False or 'No':
  
  ```
  print('hi')  ## 'hi' will print
  ```
- False and '' → evaluates as ''
- if False and '':
  
  ```
  print('hi')  ## hi will not print
  ```
How This Relates To Test Examples

• **Answer1 == 'Yes' or 'Y'**
  – Always evaluates as True
  – Equivalent to (Answer1 == 'Yes') or 'Y'
    • (Answer1 == 'Yes') evaluates as True or False
    • 'Y' is treated as True no matter what
    • The statement is always True
  – Not Equivalent to: (Answer1 == 'Yes') or (Answer1 == 'Y')

• **Answer1 == 'Yes' and 'Y'**
  – Equivalent to (Answer1 == 'Yes') and 'Y'
    • True whenever (Answer1 == 'Yes') is True
    • Not Equivalent to (Answer1 == 'Yes') and (Answer1 == 'Y')
    • Not Equivalent to Answer1 in ['Yes','Y']
print is a function

string (str) is a datatype

• Strings are concatenated via the + operator
• print takes any number of arguments separated by commas
  – print(57,'abc',my_turtle)
  – Each item (somehow) is converted into a string before printing
• print takes two optional keyword parameters
  – sep and end
Common Error on Test

• Using print's keyword arguments on strings
  
  - 'abc'+'efg',sep='*
    
    • Intention: 'abc'+'*'+'efg'
  
  - 'abc'*5,sep='*',end='&'
    
    • Intention: ('abc'*4)+'*+'&
  
• Parameters can only be arguments of functions
  
  - print('abc','efg',sep='*',end='\n**\n')
**while** and **if** are different

- Only **if** is followed by **else** or **elif**
  - **If** …:
    - XXX
  - **elif** …:
    - XXX
  - **else**:
    - XXX

- **while** begins a loop and is not connected to elifs or else
  - while expression_containing_variable:
    - XXX (including possible change of variable that ends loop)
break and return

• **break**
  – ends a loop, statements following break at the same level of indent do not execute
  – not covered in class because it is possible to exit a loop without **break**

• **return**
  – returns a value from a function
  – The function is exited immediately after **return**
  – Statements following **return** don't execute
The Tests in PDF form
Answers as PY files

• 007 test:
  – http://cs.nyu.edu/courses/fall16/CSCI-UA.0002-007/midterm1_version2.pdf
  – http://cs.nyu.edu/courses/fall16/CSCI-UA.0002-007/midterm1_version2.py

• 011 test:
  – http://cs.nyu.edu/courses/fall16/CSCI-UA.0002-007/midterm1_version1.pdf
  – http://cs.nyu.edu/courses/fall16/CSCI-UA.0002-007/midterm1_version1.py