CMPT 120
Intro to CS & Programming I

WEEK 12 (Mar. 31-Apr. 4)

— *Jérémie O. Lumbroso* —

Lecture 29:
Practice Questions, Order of Execution, Variable Scopes

http://www.sfu.ca/~jlumbros/Courses/CMPT120/

Debugging exercises

PRACTICE EXERCISE 2

Instructions

- Try to do this in exam condition
  - on paper, and no computer
  - not looking at documents (or only minimally)
- The challenge in debugging code
  - it is no longer about how **you** would do things, but how someone else would do them (or in this case, fail to do them)
  - first step: understand what the person is trying to do
  - second step: run an example or two on paper (or in Python Tutor) to see if you can identify the problem
  - try to do as little as possible to make the code work for those examples

Why is Debugging Important?

- Debugging is where **at least 50%** of a programmer's time is spent (optimistic!!)
- Debugging exercises help you
  - learn how to think rationally/methodically about fixing **your own** bugs
  - learn how to put yourself in the mindset of somebody else
  - avoid the **NIH** (Not Invented Here) syndrome, also known as "*Reinventing the Wheel*"

Final Tip(s)

- Up until now, debugging exercises have comprised of code that was originally submitted on CodeWrite
- It is useful for you to try to remember: "*What were the main problems I encountered?*" (As a matter of fact this a useful question to ask yourself at any time.)
- For instance, for parentheses matching
  - you have to match more than one set of parentheses, so a `True`/`False` flag does not work
  - have a counter for open and closed bracket
  - the counters must be equal at the end
  - the nb of closed brackets **cannot** be larger than the nb of open brackets, at any time

Sidenote

- When you want to split a very, very, very long line in Python, you can break up the line using the **backwards** slash character \
- Then create a new line

```
if s == "some very long strings to compare" and \
   s == "another very long string to compare":
  print "it's one of those strings"
```

- The indentation for the lines after the backslash does not matter.

Counting Vowels 1

```
def count_vowels(phrase):
  numVowels = 0

  for x in range(len(phrase)):
    if phrase[x] == "a" or phrase[x] == "e" or \
       phrase[x] == "i" or phrase[x] == "o" or \
       phrase[x] == "u":
      numVowels = numVowels+1

  return numVowels
```

A Works

B Doesn't work

- Does this function work?
- If not, what do you think is the problem?
- If not, can you give an example of an input for `phrase` for which this function calculates the wrong number of vowels?

Counting Vowels 1

```
def count_vowels(phrase):
  numVowels = 0

  for x in range(len(phrase)):
    if phrase[x] == "a" or phrase[x] == "e" or \
       phrase[x] == "i" or phrase[x] == "o" or \
       phrase[x] == "u":
      numVowels = numVowels+1

  return numVowels
```

A — I know why

B — Clueless (Still?)

- The submission does not work
- Consider the input `"It does not work"`
- What does the function return? What should it returns?
- **Needs to take into account UPPERCASE vowels.**

Counting Vowels 2

```
def count_vowels(phrase):
  count = 0
  if type(phrase) != str:
    return count
  else:
    phrase.lower()
    for i in phrase:
      if i != "a" and i != "e" and i != "i" and i != "o" and i != "u":
        count = count + 0
      else:
        count = count + 1
  return count
```

A Works

B Doesn't work

- Does this function work?
- If not, what do you think is the problem?
- If not, can you give an example of an input for `phrase` for which this function calculates the wrong number of vowels?

Counting Vowels 2

```
def count_vowels(phrase):
  count = 0
  if type(phrase) != str:
    return count
  else:
    phrase.lower()
    for i in phrase:
      if i != "a" and i != "e" and i != "i" and i != "o" and i != "u":
        count = count + 0
      else:
        count = count + 1
  return count
```

A — I know what to change

B — Clueless again

- This function does not work
- The problem is that `phrase.lower()` does not **modify** the variable phrase, it **returns** a value
- The solution is to do: `phrase = phrase.lower()`

Counting Vowels: *Du Bist Dran!*

- The two functions we have seen use a very long line for their `if` statement, and compare every vowel in a different line
- Some people had tried (also in the Rock/Paper/Scissors) something like this, which does **NOT** work:

```
if ch == 'a' or 'e' or 'i' or 'o' or 'u'
```

- Can you write a version of `counting_vowel` that uses a more convenient way of comparing variables (perhaps using lists? perhaps checking if a character is inside a list?)

A Done!

B Not sure?

C Clueless

Counting Vowels 3

```
def count_vowels(phrase):
  k = 0
  for ch in phrase:
    if ch in [a, e, i, o, u, A, E, I, O, U]:
      k = k + 1
  return k
```

A Works

B Doesn't work

- Does this function work?
- If not, what do you think is the problem?
- If not, can you give an example of an input for `phrase` for which this function calculates the wrong number of vowels?

Counting Vowels 3

```
def count_vowels(phrase):
  k = 0
  for ch in phrase:
    if ch in [a, e, i, o, u, A, E, I, O, U]:
      k = k + 1
  return k
```

A — I know what to change

B — Clueless again

- Here the function uses the **in** keyword to test membership in a list (like what we did on Monday)
- The problem is Python will think the letters in the list are variable – but we mean them as strings

Counting Vowels 3

```
def count_vowels(phrase):
  k = 0
  for ch in phrase:
    if ch in ['a', 'e', 'i', "o", 'u', 'A', 'E', 'I', 'O', 'U']:
      k = k + 1
  return k
```

- Here the function uses the **in** keyword to test membership in a list (like what we did on Monday)
- The problem is Python will think the letters in the list are variable – but we mean them as strings
- What we need to do is put quotes around every letter

We resume our exploration of Monday…

ORDER OF EXECUTION

Top-level code

- We call "top level" any code that is not in a function (or later a class, or module)
- The "top level" code is generally code that does not have any indentation in front of it

Order of Execution 1

- What is the order of execution of this block of code?

```
def fun(a, b):                  #1
  c = a + b*2                   #2
  print "inside function"       #3
  return c                      #4

# TOP LEVEL
print "here we start"           #5
val = fun(2, 3)                 #6
print val                       #7
```

- Order of execution: 5, 6, 1, 2, 3, 4, 6b, 7
- (Convention 6b means that we go back to that line for assignment)

Order of Execution 2

```
def fun(a,b):                                  #1
   c = a + b*2                                 #2
   return c                                    #3

# TOP LEVEL
accum = 0                                      #4
for i in [1,2,3]:                              #5
   accum = accum + fun (i,i+1)                 #6
print accum                                    #7
```

- Order of execution: 4, 5, 6, 1, 2, 3, 6b, (5), 6, 1, 2, 3, 6b, (5), 6, 1, 2, 3, 6b, 8

A Done

B Help!

Ordering of Functions

- Can this work? Or not?

```
def funA(a):
  return funB(a+1)

print "here"

def funB(b):
  return b*2

print funA(3)
```

A Works

B Error

- What is the order of execution?
- It is only important that, when we **call** funA, funB is defined.

Structure of a Program

- Many functions
- The top level assembles these functions
- Modular programming (desirable) means
  - we separate repetitive tasks in functions
  - we group the logic in the top-level of the program

Pacing and Understanding

How well did you understand today?

A. Too easy or **too slow**

B. Everything went at a good pace, and I am fine

C. Too fast, but I will catch up on my own

D. I do not like doing exercises in class

E. I am like a cow getting slaughtered – that's how I think of the final; at this point, I would pay you for a guaranteed good grade