CMPT 120
Intro to CS & Programming I

WEEK 12 (Mar. 31-Apr. 4)

— *Jérémie O. Lumbroso* —

Lecture 30:
Practice Questions, Order of Execution, Variable Scopes

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

Running code

PRACTICE EXERCISE 3

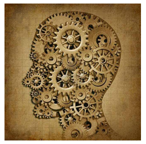
Instructions

- Try to do this in exam condition
  - on paper, and no computer
  - not looking at documents (or only minimally)
- The challenge in running code is…
  - it is **SUPER BORING**
  - what you need is to be **methodical**: pretend you are Python Tutor, and **do every single step** of running the code, even if it seems pointless
  - <u>first step:</u> keep track of the values of variables and how they change for **EVERY LINE OF CODE**
  - <u>second step:</u> if there are parameters to choose (like for the min/max/median exercise), then try all combinations that make sense

Why is Running Code Important?

- First: it's faster than **walking** code
- No seriously: you might ask, "*Since Python can run any code I give it, and Python Tutor can even do it step-by-step, why do **I** need to know how to do it?*"
- Fair question.
- Knowing how to run code
  - is an important part of knowing how to debug code (because you need to look at code, and understand what it is doing)
  - is useful because if you ever find yourself writing programs over the long term, then knowing how to run code in your head (+ making good use of comments) is a good way to pick back up where you left off
  - but for us — you (loving class) and I (loving professor) — it is mostly a tool to check if you understand what is going on
- Sometimes, the code is nonsense, but often is does something

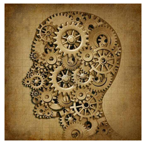
First Piece of Code



```
def funA(a, b):
  r = 0
  for i in range(a):
    r = r + b
  return r

def funB(a, b):
  r = 0
  for i in range(b):
    for j in range(a):
      r = r + 1
  return r

varOne = funA(10, 3)
varTwo = funB(15, 2)

print varOne - varTwo
```

What is printed at the end?
(iClicker numerical vote)

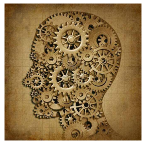
Second Piece of Code



What is printed at the end?
(iClicker numerical vote)

```
def funC(a, b):
    r = b
    while r >= 0:
        r = r - a
    return r + a

print funC(3, 13)            # Q1
print funC(2, 20)            # Q2
print funC(4, 12)            # Q3
```

Third Piece of Code



What is printed at the end?
(iClicker numerical vote)

```
def funD(a, b, c):
  if a > b:
    b = a
  if a > c:
    c = a
  return a + b + c

print funD(1, 3, 13)          # Q1
print funD(4, 2, 10)          # Q2
print funD(10,3, 2)           # Q3
```

We resume our exploration of Monday…

ORDER OF EXECUTION

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
```

- <u>Order of execution:</u> 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
```

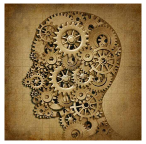
- Order of execution: 4, 5, 6, 1, 2, 3, 6b, (5), 6, 1, 2, 3, 6b, (5), 6, 1, 2, 3, 6b, 8

Ordering of Functions

- Only important that, when we **call** funA, funB is defined

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

print "here"

def funB(b):
  return b*2

print funA(3)
```

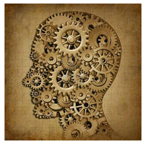
Variable Scope



- Can this work? Or not? Vote with iClicker

```
def funA():
  print k

def funB():
  print k + 1

def funC():
  k = k+1
  print k

k = 10
funA()                           # A) Works B) Does not work
funB()                           # A) Works B) Does not work
print "Value of k", k            # Numeric: value of k
funC()                           # A) Works B) Does not work
print "Value of k", k            # Numeric: value of k
```

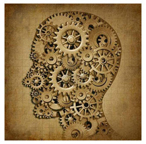
Variable Scope 2



- Can this work? Or not? Vote with iClicker

```
def funD(a):
  g = a + 10

def funE(a):
  h = a + 10
  return h

a = 5
funD(10)
print g                          # A) Works B) Does not work
                                 # Numeric: value of g
funE(10)
print h                          # A) Works B) Does not work
                                 # Numeric: value of h


q = funE(10)
print q                          # A) Works B) Does not work
                                 # Numeric: value of q
```

Variable Scope 3



- Can this work? Or not? Vote with iClicker

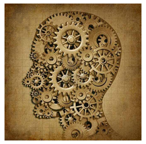
```
def funF(a):
  global g
  g = a + 10

funF(10)
print g                  # A) Works B) Does not work
                         # Numeric: value of g
g = 25
funF(10)
print g                  # A) Works B) Does not work
                         # Numeric: value of h
```

Global Variables, and Functions

- When you define a variable in the top-level, it can **read** anywhere (including inside functions): these are **global** variables
- Global variables cannot be **modified** inside functions without using the global keyword
- Besides global variables, the only way to extract the value of a variable from inside a function is to use the **return** keyword

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