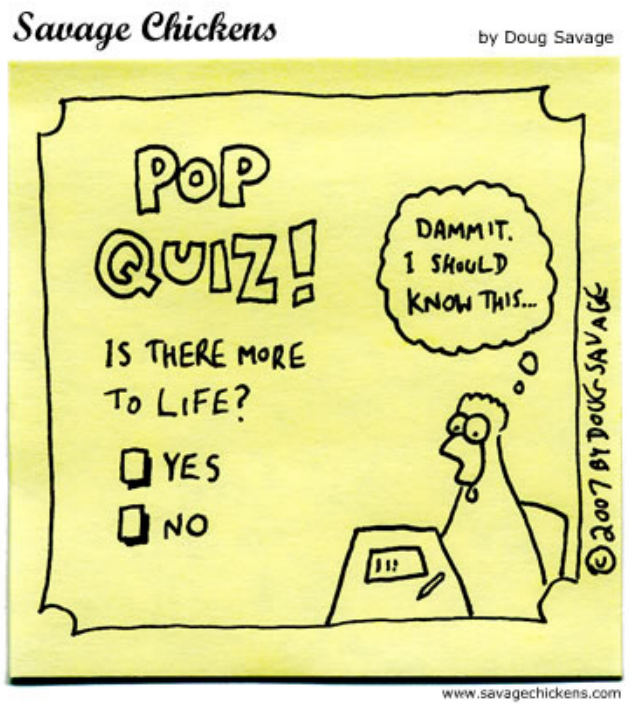
CMPT 120
Intro to CS & Programming I

WEEK 3 (Jan. 20-24)

— *Jérémie O. Lumbroso* —

Lecture 6:
Functions and modules

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

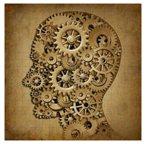


See if you have understood two important notions for this lecture

SMALL POP QUIZ

Pop Quiz on Blocks

Q1. What is the output of this code?

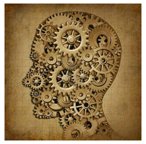


```
sumvar = 0
for i in range(1, 10):
    sumvar = sumvar + i
    print sumvar
```

A	B	C	D
1 3 6 10 15 21 28 36 45	45	Error	Other answer

Pop Quiz on Blocks

Q2. What is the output of this code?

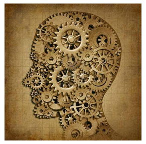


```
sumvar = 0
for i in range(1, 10):
    sumvar = sumvar + i
print sumvar
```

A	B	C	D
1 3 6 10 15 21 28 36 45	45	Error	Other answer

Pop Quiz on Blocks

Q3. What is the output of this code?



```
sumvar = 0
for i in range(1, 10):
sumvar = sumvar + i
print sumvar
```

A

```
1
3
6
10
15
21
28
36
45
```

B

```
45
```

C

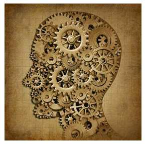
Error

D

Other answer

Pop Quiz on Variables

Q4. What is the value of myvar at the end?



```
myvar = 3
myvar = myvar * 4 + 2
```

- A 28
- B 14
- C 3
- D -2/3
- E I don't know

Making reusable blocks of code

FUNCTIONS

Never Repeat!

- Like in the sandwich example, there are sequences of actions that are useful in different contexts
  - Spread (XXX)
  - Open jar (XXX)
- These sequences can be reused, and can be reused with different things as XXX

Add Consecutive Integers

- *"Add all integers from 1 to 100."*
- Several ways of doing it:
  - Take calculator and 1+2+…+100 = 5050
  - Go in Python interpreter:

```
>>> sumvar = 0
>>> for i in range(1, 101):
...   sumvar = sumvar + i
...
>>> print sumvar
```

- What if you now need sum from 1 to 200?
- Retype everything? Does that seem smart?!

Define a Function

```
# sumRange returns the sum of first+(first+1)+...+(last-1)+last.
# Hypotheses: first and last are integers, first <= last.

def sumRange(first, last):
  sumvar = 0
  for i in range(first, last+1):
    sumvar = sumvar + i
  return sumvar
```

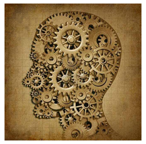
- A function is defined using the keyword `def`
- The syntax is
  - `def <function name>(<parameters>):`
  - `   <block of the function>`
- Can have any number of parameters (including none)
- The keyword `return <value>` means that the function will return that value as a result

What Does Returning a Result Mean?

```
# sumRange returns the sum of first+(first+1)+...+(last-1)+last.
# Hypotheses: first and last are integers, first <= last.

def sumRange(first, last):
  sumvar = 0
  for i in range(first, last+1):
    sumvar = sumvar + i
  return sumvar
```

- Above is the definition of the function `sumRange`:
  - it takes two parameters: `first` and `last`
  - it returns a value
- When we want to use the function, we can make a **call** to the function: `sumRange(1, 10)`
  - we type the name of the function
  - and between parentheses, we replace the name of the parameters with the values that we would want them to take
- The result we will obtain is what the function **returns**

Do It Yourself



```
# sumRange returns the sum of first+(first+1)+...+(last-1)+last.
# Hypotheses: first and last are integers, first <= last.

def sumRange(first, last):
  sumvar = 0
  for i in range(first, last+1):
    sumvar = sumvar + i
  return sumvar
```

- Type the definition of that function in the Python shell or IDLE
- Then make the following **calls** to the function
  - `print sumRange(1, 10)`
  - `print sumRange(1, 10) + sumRange(11, 20)`
  - `print sumRange(1, 20)`
- Python works in the following way: when you make a call to a function, it runs the function then replaces the call by the value calculated by the function; the calls above are equivalent to
  - `print 55`
  - `print 55 + 155`
  - `print 210`

(Actually…)

Carl Friedrich Gauss,

German mathematician in 18th century,

Found a formula for the sum of consecutive integers that doesn't involve having to do a loop

```
# sumRange2 returns the sum of first+(first+1)+...+(last-1)+last.
# Hypotheses: first and last are integers, first <= last.

def sumRange2(first, last):
  numterms = last - first + 1
  return (first + last)*numterms/2
```

Functions Without Return Values

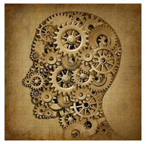
- Functions do not necessarily return a value
- Some functions just "do something"
  - print something on the terminal
  - draw something on the screen
  - save data to a file
- In such a case, we can call the function a **void function** or a **procedure** or a **subroutine**

```
# greet says hello to a person.

def greet(person):
  print "Hello there " + person + "!"
  print "How do you like programming in Python?"
```

Calling a Procedure as a Function

```
# greet says hello to a person.

def greet(person):
  print "Hello there " + person + "!"
  print "How do you like programming in Python?"
```

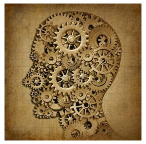
- `greet` is a void function/procedure
- It does not return a value
- What happens if you type
  - `print greet("Simon")`
  - `greet("David") + greet("John")`



A Function We Have Already Seen

- We have already seen one function
  - `range(a, b)`
- This function **returns** a list of integers
  - `[a, a+1, ..., b-1]`
- We have used this return value together with a `for` loop to be able to iterate over a range of integers
- (We will see about lists later on)

What Does This Function Do?



When I type:

```
print surpriseFunction(13, 47, 5)/2
```

what do I get?

```
def surpriseFunction(a, b, c):
  if (a <= b and b <= c) or (a >= b and b >= c):
    return b
  elif (b <= a and a <= c) or (b >= a and a >= c):
    return a
  else:
    return c
```

- **A** 6
- **B** 13
- **C** 25
- **D** 5
- **E** ERROR

How Is This Code Run?

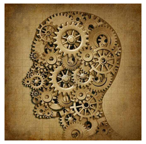
```
mid_value = surpriseFunction(13, 47, 5)/2
```

- The expression on the right of the variable assignment must be evaluated before the variable can be assigned
  - It evaluates the expression `surpriseFunction(13, 47, 5)/2`
- Sub-expressions on either side of the division operator must be evaluated
  - Evaluate `surpriseFunction(13, 47, 5)/2`
  - Now the expression is put on hold until the function can be calculated
- The function `surpriseFunction` is called
- The parameters that are given in the calling code (13, 47, 5) are assigned to the local variables given in the argument list (a, b, c)
  - a = 13, b = 47 and c = 5
- The function ends with return a, so a = 13 is returned by the function
- The calling code gets the return value, 13, and the expression is now 13/2
- The integer 6 is assigned to the variable `mid_value`

Advantages of Functions

- As we said, functions make sense when you are writing code that might be reusable
  - Not necessarily this time around but maybe next time
- Also
  - Easier to build and debug
  - Makes the program easier to read
  - Prevents duplicating your code
- You should **never** copy-paste code
  - What happens if you made a mistake in that code? you have to correct EVERY copy-pasted version
  - What happens when you want to update it?

Write Your Own Function



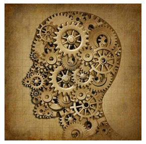
- Define a function that
  - takes two parameters numOne and numTwo
  - checks that numOne and numTwo are positive
    - if either one is not, return 0
    - if they are both positive, return numOne + numTwo
- What name do you give the function?

A I am done and I think I got it

B I am done, and I think I did not get it, or I gave up

Possible Solution

```
def sumTwoInts(numOne, numTwo):
  if numOne <= 0 or numTwo <= 0:
    return 0
  return numOne + numTwo
```

Is this what you had? 

A Yes, that's more or less what I had

B No, I did not find that at all

Things to be Careful About

- Here are some problems you might encounter with functions
  - In your program, does the order in which function appear (are defined) important?
  - Problems of **variable scope**
    - Can variables from outside the function be used in the function? (And should they?)
    - Can variables used inside your function be used outside of the function? (And should they?)
- These are important questions we will see later this week

Python's functions written for you

INTRO TO MODULES

Python Has Modules

- In Python, the notion of **module** is a library that contains lots of functions (among other things) that you can use without having to write them yourself
- Before using a module, you have to **import** it
- Once a module is imported, you can call a function from it by doing
  - `<module>.<function name>(…)`

Example: `math` module

- Python has a **math** module
- It contains all sort of mathematical functions
  - `import math`
  - `math.sqrt(25)`
  - `math.gamma(11)`
  - `math.factorial(10)`

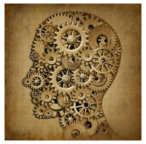
How to Get Help

- Python code can be documented (this is different from being commented)
- The documentation can be accessed from the Python shell by using the `help(...)` command
  - This gives you information on any expression
  - For modules, it tells you what functions they introduce and can be used

Example: `math` module

```
>>> import math
>>> help(math)
Help on module math:

NAME
    math

FILE
    /opt/local/Library/Frameworks/
Python.framework/Versions/2.7/lib/python2.7/
lib-dynload/math.so

MODULE DOCS
    http://docs.python.org/library/math

DESCRIPTION
    This module is always available.  It
provides access to the
    mathematical functions defined by the C
standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in
radians) of x.

 acosh(...)
        acosh(x)

        Return the hyperbolic arc cosine
(measured in radians) of x.

 asin(...)
        asin(x)

        Return the arc sine (measured in
radians) of x.

    asinh(...)
        asinh(x)

        Return the hyperbolic arc sine
(measured in radians) of x.

    atan(...)
        atan(x)

        Return the arc tangent (measured in
radians) of x.


[...]
```

Write Your Own Function



- Open the Python shell
- Type `import random` (this is the randomization module)
- Find out how to use the function `random.randint` by using the help command
- Define a function that
  - draws a random integer between 1 and 100
  - returns True if it is larger or equal to 25, and False if not

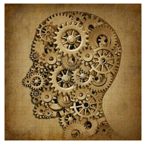
A I am done and I think I got it

B I am done, and I think I did not get it, or I gave up

Possible Solution

```
import random
def randLargerTwentyFive():
  mynum = random.randint(1,100)
  if mynum >= 25:
    return True
  else:
    return False
```

Is this what you had?

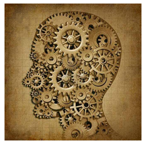


A Yes, that's more or less what I had

B No, I did not find that at all

Pacing and Understanding

How well did you understand today?



- A Too easy, this lecture is way below my abilities
- B Everything went at a good pace, and I am fine
- C Too fast, but I will catch up on my own
- D Too fast, and I need you to slow down
- E I really do not think I can handle this