EEB 177 Lecture 12

Thursday Feb 16th, 2017

Topics

- functions
- python debugger
- in class challenge!

Schedule Next Week

Guest Lectues by Jonathan Chang on web scraping.

no office hours (I will be out of town)

Hacking Sessions

Tuesday from 6-8:30 in LS 3209

Homework #5

Due Wednesday by 5:00 PM

Projects Deadline #3

Due Friday by 5:00 PM. To complete this assignment:

- Commit with 1 working function due Friday, February 17th

Here is one way to control the number of significant figures

```
def get_at_content(dna, sig_figs):
    length = len(dna)
    a_count = dna.upper().count('A')
    t_count = dna.upper().count('T')
    at_content = (a_count + t_count) / length
    return round(at_content, sig_figs)
```

Encapsulation

OK you have made changes to your function to improve formatting. What you may not have noticed is that you did not need to modify (much) the rest of the program outside the function. This demonstrates the idea of **encapsulation** which really just mean breaking up a big program in to smaller parts that can be managed in a straightforward way.

Keep this idea in mind as you start to consider you final projects!

Keyword arguments

If you define a function in the typical way, the order that you pass the arguments is critical.

```
In [62]: def outOfOrder(number, pet):
   ....:         print "I own {} {}s".format(number, pet)
   ....:
In [63]: outOfOrder(3, "dog")
I own 3 dogs
In [64]: outOfOrder("dog", 3)
I own dog 3s
```

Keyword arguments, wherein the name of the argument followed by an "=" followed by the value, can help avoid these kinds of errors.

```
outOfOrder(number=3, pet="dog")
#same as
outOfOrder(pet="dog", number=3)
```

Default values

It is often useful to pass a default value to a function that can be overridden.

```
def get_at_content(dna, sig_figs=2):
    length = len(dna)
    a_count = dna.upper().count('A')
    t_count = dna.upper().count('T')
    at_content = (a_count + t_count) / length
    return round(at_content, sig_figs)

get_at_content("ATCGTGACTCG")
get_at_content("ATCGTGACTCG", 3)
get_at_content("ATCGTGACTCG", sig_figs=4)
```

Now the value will be written to two decimal places unless otherwise specified.

Testing functions

It is important that your code is behaving the way you expect it will. One way to do this is to invoke functions with arguments that should produce a known value. You can use the **assert** function to test if your code passes this test.

```
assert get_at_content("ATGC") == 0.5 #this should work
assert get_at_content("ATGCNNNNNNNNNN") == 0.5 #what about
```

Modify your get_at_content() program so that you function will pass an assert with "Ns"

Improved code

```
def get_at_content(dna, sig_figs=2):
    dna = dna.replace('N', '')
    length = len(dna)
    a_count = dna.upper().count('A')
    t_count = dna.upper().count('T')
    at_content = (a_count + t_count) / length
    return round(at_content, sig_figs)
```

Now the function passes the assert test.

assert

assert statements are used to make sure your functions are receiving valid input and returning expected outputs

syntax

```
assert Expression[, Arguments]
```

Python evaluates the expression and continues with the program if the result is True. Otherwise it raises an assert error.

```
def KelvinToFahrenheit(Temperature):
   assert (Temperature >= 0),"Colder than absolute zero!"
   return ((Temperature-273)*1.8)+32
```

It is a good idea to include a series of assert tests for functions you creates. You can easily comment the assert statements out as needed.

```
assert get_at_content("A") == 1
assert get_at_content("G") == 0
assert get_at_content("ATGC") == 0.5
assert get_at_content("AGG") == 0.33
assert get_at_content("AGG", 1) == 0.3
assert get_at_content("AGG", 5) == 0.33333
```

You can use assert to check for variable types in your program

```
def chkassert(num):
    assert type(num) == int, "this function takes an int!"

chkassert(1)

chkassert('a')
```

Debugging

the implementation of the python debugger (**pdb**)Ipython provides another way of searching out bugs. Lets examine it. Start with this function.

```
def createabug(x):
    y = x**4
    z = 0.
    y = y/z
    return y

createabug(5)
```

What happens?

Use your text editor to ave this file as *debugme.py* in your classwork directory, Load and execute your function:

```
In [1]: %run debugme.py
...
----> 4     y = y/z
      5     return y
      6

ZeroDivisionError: float division by zero #doesn't work...
```

Now start the debugger in your ipython notebook and run again:

```
In [72]: %pdb
Automatic pdb calling has been turned ON
In [73]: %run debugme.py
...
ZeroDivisionError: float division by zero
> /home/vagrant/scripts/debugme.py(4)createabug()
      3     z = 0.
----> 4     y = y/z
      5     return y

ipdb> # <---- NOTICE THIS!
```

Now we are in the debugger shell!

pdb commands

Within the debugger we can move around in our code and examine variables to see what is happening.

- **n** move to the next line.
- **ENTER** repeat the previous command.
- **s** step into function or procedure (i.e., continue the debugging inside the function, as opposed to simply run it).
- **p x** print variable x.
- **c** continue until next break-point.
- **q** quit
- **l** print the code surrounding the current position.
- **r** continue until the end of the function
- **! expr** evaluate an expression.

try out pdb

```
ipdb> p x
25
ipdb> p y
390625
ipdb> pz
*** NameError: name 'pz' is not defined
ipdb> px
*** NameError: name 'px' is not defined
ipdb> p x
25
ipdb> p y
390625
ipdb> p z
0.0
ipdb> p y/z
*** ZeroDivisionError: ZeroDivisionError('float division b
ipdb> !y/z
*** ZeroDivisionError: float division by zero
ipdb> q
In [74]: pdb
Automatic pdb calling has been turned OFF
```

Automatically launch pdb

If you want to start pdb within a longer script, place this line at the point you want to examine:

```
import pdb;
pdb.set_trace()
```

in class exercise

1. write a function that

in class exercise

Modify the homework you did for chapter 4 exercise 2 by inserting the pdb line above within your **for** loop. Run the pdb within ipython. Make a commit with the pdb line included and upload a link to your repository showing this change.

In Class Challenge