BIO508: Lab Session 3

Announcements

- As homeworks get more complicated, I will get less able to assign partial credit; make sure your functions give the output requested. Testing is an important part of coding.
- Common homework mistakes:
  - Naming functions
    - There is a reason we ask for specific function names in the homework, and it is because the grading script relies on these names to find your functions.
    - Function names are case sensitive (`Mean` is not the same as `mean`).
  - Printing vs. returning in a function
  - Make sure your triple-quoted documentation string is in the correct format!
  - Don't leave `if __name__ == "__main__"` block empty; this produces an error.
  - Returning Booleans vs. strings
  - You can call one function from another (for example, you can call `mean` in `stdev`).
  - `float(iN)/iM` is not the same as `float(iN/iM)`
- Another useful Python resource is Learn Python The Hard Way (`learnpythonthehardway.org/book/`)

The get() function for dictionaries

There was a mistake in my explaination of the `get()` function yesterday: `hashSomeDict.get(strKey, valueDefault)` doesn't have the side effect of adding `strKey` into the dictionary; it simply returns the default value. Try the following in your Python interpreter:

```
hashA = { 'a': 1 }

print hashA.get( 'b', 0 )

print hashA

hashA[ 'b' ] = hashA.get( 'b', 0 )

print hashA
```

Now think: what did the `hashRet[strCodon] = 1 + hashRet.get( strCodon, 0 )` command in yesterday's `codon_count()` function really mean?

Reviewing Some Common Homework Mistakes

Naming functions

In a new text file called `lab03_ex1.py`, type the following *exactly*:

```
#!/usr/bin/env python

def my_func():

    return 5

if __name__ == "__main__":

    print my_func()
    print My_func()
```

Run this script on the command line using `python lab03_ex1.py`.

You should see:

```
sph182-159:Lab 03 emmaschwager$ python lab03_ex1.py
5
Traceback (most recent call last):
  File "lab03_ex1.py", line 10, in <module>
    print My_func()
NameError: name 'My_func' is not defined
```

When Python gives you an error, the first line (`File "lab03_ex1.py", line 10, in <module>`) gives you the line number where the error occurred, the second line gives a copy of that line and the third line tells you something (sometimes cryptic) about what the error might be.

1. In English, what is the error telling you?

2. Fix the script so that it runs without an error.

Printing vs. Returning in a Function

In a new text file called `lab03_ex2.py`, type the following *exactly*:

```
#!/usr/bin/env python

def add_print(a,b):
    print "ADDING",a,"+",b
    print a + b

def add_return(a,b):
    print "ADDING",a,"+",b
    return a+b

if __name__ == "__main__":

    num1 = 3
    num2 = 5

    c = add_print(num1,num2)
    d = add_return(num1,num2)

    print "c = ",c
    print "d = ",d

    if c==(num1+num2):
        print "add_print works!"
    else:
        print "add_print doesn't work."

    if d==(num1+num2):
        print "add_return works!"
    else:
        print "add_print doesn't work."
```

Run this on a terminal using `python lab03_ex2.py`.
You should see:

```
sph182-159:Lab 03 emmaschwager$ python lab03_ex2.py
ADDING 3 + 5
8
ADDING 3 + 5
c =  None
d =  8
add_print doesn't work.
add_return works!
```

1. Which function "works"? Why do you think that is?

2. Does it help to change the values of `num1` and `num2`? Why or why not?

3. Try adding the line `print "extra line"` on the line after the `return` statement in `add_return` so that `add_return` looks like:

   ```
   def add_return(a,b):
       print "ADDING",a,"+",b
       return a+b
       print "extra line"
   ```

   Save and run the script again; do you see `"extra line"` in your output? Why do you think that might be?

Leaving `if __name__=="__main__"` block empty

In a new text file called `lab03_ex3.py`, type the following:

```
#!/usr/bin/env python


if __name__ == "__main__":
```

Run the file using `python lab03_ex3.py`.
You should see:

```
sph182-159:Lab 03 emmaschwager$ python lab03_ex3.py
  File "lab03_ex3.py", line 6

    ^
IndentationError: expected an indented block
```

Note that when you see an indentation error, you usually forgot to indent something following a colon.

1. In English, what does this error mean?

2. Fix the script so that it runs without an error (note that there are an infinite number of ways to do this; you can choose to do it simply or complicatedly.)

Booleans versus Strings

Open a Python interpreter by opening a terminal (command line) and typing `python`. Type the following commands:

```
bTrue = True
bFalse = False
sTrue = "True"
sFalse = "False"
if bTrue: print "Truth!"

if not(bFalse): print "Not falsehood!"

if sTrue: print "Truth!"

if not(sFalse): print "Not falsehood!"
```

You should see:

```
>>> bTrue = True
>>> bFalse = False
>>> sTrue = "True"
>>> sFalse = "False"
>>> if bTrue: print "Truth!"
...
Truth!
>>> if not(bFalse): print "Not falsehood!"
...
Not falsehood!
>>> if sTrue: print "Truth!"
...
Truth!
>>> if not(sFalse): print "Not falsehood!"
...
>>>
```

1. Why do you think `not(sFalse)` doesn't return `True`?

References

Remember that in Python, *unit* types such as integer, Boolean, float and string are stored by value but that *collection* types such as lists and dictionaries are stored by reference. This can lead to some counter-intuitive behavior. We will first explore some of the common bugs that can arise.

Open the Python interpreter by going to the command line and typing `python`. Type the following:

```
aList1 = [1,2,3]
aList2 = aList1
aList1[0] = 12
```

1. What is the value of `aList1`? Of `aList2`? Are they the same or different? Why?

2. Can you think of some ways to circumvent this problem?

Now back in the interpreter, type the following:

```
hDict1 = {'a':13,'b':18}
hDict2 = hDict1
hDict2['c'] = 12
```

1. What do you think is the value of `hDict1`? Double check your answer.

2. Can you think of some ways to circumvent this problem?

Again going back to the interpreter, type the following:

```
aList3 = [12,7,15,2,10]
sorted(aList3)
aList3
aList3.sort()
aList3
```

1. What happens to `aList3` when you use the `sorted()` function?

2. What happens to `aList3` when you use the `.sort()` function?

3. In your own words, how would you describe the difference between these two functions?

You can quit the interpreter by typing `quit()`. This will take you back to the command line.
Some additional reading, just for fun: copy vs. deepcopy in Python `http://stackoverflow.com/questions/3975376/understanding-dict-copy-shallow-or-deep/3975388#3975388`.