BIO508: Lab Session 3 Key

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?
   That Python can't find a definition for the function `My_func`.
2. Fix the script so that it runs without an error.

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

def my_func():

    return 5

if __name__ == "__main__":

    print my_func()
#    print My_func()
```

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?
   `add_return` works because the return statement allows the value of `a+b` to be assigned to the variable `d`, while printing the result does not allow `c` to have any value assigned.

2. Does it help to change the values of `num1` and `num2`? Why or why not?
   It does not help to change the values because the problem is in the function itself and not with the inputs.

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?
   You don't see `"extra line"` in your output because the function ends with the return statement and doesn't execute any line thereafter.

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?
   It means that Python expected an indent following the definition of the `if __name__=="__main__"` block, but you hadn't indented anything.

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.)

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


   if __name__ == "__main__":

       print "Hi!"
   ```

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`?
   Because `sFalse` is a string and so, by definition, `True`; `not(sFalse)` evaluates to `False`.

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?
   `aList1 = [12,2,3]` and `aList2 = [12,2,3]`, which are the same. They are the same because the `=` operator assigned the *reference* of `aList1` to `aList2`, meaning that any alteration to one alters the other as well.

2. Can you think of some ways to circumvent this problem?
   You could simply use `aList2 = [1,2,3]` instead; or you could use the `deepcopy` function from the `copy` module.

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.
   `hDict1 = {'a':13,'b':18,'c':12}`

2. Can you think of some ways to circumvent this problem?
   You could simply use `hDict2 = {'a':13,'b':18}` instead; or you could use the `deepcopy` function from the `copy` module.

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?
   The function returns a sorted copy of `aList3`, but `aList3` remains unchanged.

2. What happens to `aList3` when you use the `.sort()` function?
   The function changes `aList3` to a sorted version of itself.

3. In your own words, how would you describe the difference between these two functions?
   `sorted()` returns a different list, while `.sorted()` makes a change to the list itself.

You can quit the interpreter by typing `quit()`. This will take you back to the command line.