BIO508: Lab Session 2

Announcements

- TA's Office Hours: Thursday 3:30 to 4:30 in Kresge 205
- Questions about HW grades should be directed to me.
- From last lab: the `ftype` command sequence for Windows machines is unnecessary
- Homework 2 is due at 11:55pm on Monday
- If you need an extension, be sure to let us know ***early***. Extensions will only be granted:
  - For a good reason. "I started my HW too late" is not a good reason.
  - Until 3:30pm the Thursday following the due date, since I will go over common mistakes in lab.
- Everyone should have an account on the website at this point.
- Some useful Python resources:
  - `http://docs.python.org/2/tutorial/`
  - Appendix 4 in Haddock and Dunn (p. 479-492)
  - Google!!

Submitting a Python script

1. When you submit your homework, submit only one text file named `problems<2 digit problem set number>.py`. For example, this assignment will be called `problems02.py`.
2. The very start of the file should be the "shebang": `#!/usr/bin/env python`.
3. The file should start with a triple-quoted documentation string containing (in order):
   - Your name
   - The file name (eg. `problems02.py`)
   - Your course number (BIO 508 or BIST 297) followed by a space and then the date
   - How many hours you spend on the assignment (excluding extra credit and optional components)
   - A blank line
   - A paragraph of what you felt the overall goals of the assignment were and what you gained from it
   - A blank line
   - A paragraph describing your thoughts on the assignment, major problems you had, and any comments/suggestions
4. Only function definitions should be outside the `__name__ == "__main__"` block. You define your functions outside that block but *run* them inside the block.
5. ***MAKE SURE IT RUNS!!!*** Please keep your TA happy by testing that your finished program executes properly: run `python <problem set file>` on the command line. If you get an error, make sure to fix it before your submit your program.

Python types and keywords

Python types

There are many built in *types* in Python which we will use in this class (and beyond!). Table **??** lists those we've gone over so far.

Type	Description	Example	Notes
String	A collection of letters, numbers, punctuation, etc. surrounded by double quotes, `""`	`"This is a string"` `"A \" in a string"`	• You can use either single (') or double (") quotes in Python • You can use \ as an escape sequence to insert oddball characters. The most common uses are `\'`, `\"`, `\\`, `\t` and `\n`.
Boolean	Logical constructs of truth and falsehood	`True` `False`	• `False` and `None` are false • Any number (floating or integer) that represents zero is false • Any string or collection with no length or elements is false (e.g., `""`, `[]`, {}) • `True` and anything that isn't false is true.

List	A collection of stuff, enclosed in square brackets and with comma-separated elements	`aListOfInts = [1,2,3]`
		`["a string",0,2.2]`
		`[-12,strAString,["a","b"]]`
			• To reference an element of a list, use the name of the list with square brackets (indices start at 0). E.g., `aListOfInts[0]` ⇒ `1`.
			• Using negative numbers for indices counts backwards from the end of a list. E.g., `aListOfInts[-1]` ⇒ `3` and `aListOfInts[-2]` ⇒ `2`.

Dictionary	A collection of key-and-value pairs, comma separated, enclosed in curly brackets	`hashDictionary =` `{"am":[1,2], "pm":12}`	• To reference an element of a dictionary, use square brackets and the key name. E.g., `hashDictionary["am"]` ⇒ `[1,2]` • To find out if a value is really a key, use the `.get()` function. E.g., `hashDictionary.get("am")` ⇒ `[1,2]` but `hashDictionary.get("myvalue")` ⇒ `None`
Numeric	A number: either floating point or integer.	`iX = 1` `dX = 1.0`	• Can be an element of a list, or a key/value in a dictionary. • ***CAUTION:*** Although on paper `1` and `1.0` mean the same thing, in Python they sometimes behave differently.
None	Nothing at all.	`None`	• Serves as a place holder • Is `False` • Is not of any *type*

Table 1: The Python types we will use in this class.

Python keywords

Key words are words with special meanings. Don't use these words as variable names! Table **??** shows some of the common keywords you will need in this class.

Word(s)/symbol	Syntax	Meaning	Example

`pass`	`pass`	Null statement; fills in empty code blocks	`if False:`
			`  pass`
			`else:`
			`  print( True )`

`del`	`del i`
	`del list[j]`
	`del hash[k]`
		Removes an element from a list or dictionary; or undefines a variable.	`ai = [1,2,3]`
			`del ai[1]`
			`del ai`

`if`
`elif`
`else`
	`if test1:`
	`  body1`
	`elif test2:`
	`  body2`
	`else:`
	`  body3`
		If *test1* evaluates to `True`, then *body1* is executed. Otherwise, if *test2* evaluates to `True`, then *body2* is executed. If neither test evaluates to `True`, then *body3* is executed. You can have zero or more `elif` statements, and zero or one `else` statements.	`if i < 10:`
			`  print("tall")`
			`elif i < 20:`
			`  print("grande")`
			`else:`
			`  print("venti")`

`while`	`while test:`
	`  body`
		Evaluate *test*. If `True`, evaluate *body* and repeat.	`i = 0`
			`while i < 10:`
			`  print(i)`
			`  i = i+1`

`for/in`	`for var in list:` `  body`	For each element in *list*, set *var* to that value and evaluate *body*	`for i in [1,3,7]:` `  print( i )`

`continue`	`continue`	Terminate the current loop iteration and start the next one.	`for i in [1,3,7,11]:`
			`  if i < 10:`
			`  continue`
			`  else:`
			`  print( i )`

`break`	`break`	Terminate the current loop completely.	`for i in [11,7,3,1]:`
			`  if i < 10:`
			`  break`
			`  else:`
			`  print( i )`

Table 2: Some of the important keywords in Python

Python variable names

Restrictions:

- Begin with a letter
- Can have letters, numbers and underscores
- Case sensitive

Curtis' conventions:

- CamelCase for long words
- Typically start variable name with lowercase letters
- Start with some letters indicating the variable type (string, float, integer, etc.)

Practicing with Operators and Functions

The operators and functions that we went over in class are listed on the last page (table **??** and table **??**, respectively). Here are some (simple) exercises for you to try in order to familiarize yourself with what some of these actually do! If you have extra time, by all means experiment with other commands.

1. Download the file `lab02_practice.py` from the course website (`http://huttenhower.sph.harvard.edu/bio508`)
2. Open it in the Python editor of your choice (e.g., jEdit).
3. At the same time, open up a Python interpreter (this can be on your command line by typing `python` or it might come with your Python editor.)
4. Run the commands in the file by copying and pasting them into the Python interpreter.
   - ***NOTE:*** Some of the commands will give you errors. These are intentional.
5. Diagnose and fix (in the document) any errors that appear when you do this.
6. Add a docstring to the top of the document with your name and today's date
7. In the interpreter (you shouldn't need to add any text to your document), answer the following questions/do the following things:
   (a) How many elements are in `aiX`?
   (b) What is the value of the last element of `aiX`? Remove this last element.
   (c) What are the values of the first three elements of `aiX`? Make the second element 100.
   (d) Generate a variable `iXSum` which is the sum of all elements of `aiX`.
   (e) Print the value of `iXSum` to the screen, but only if it's greater than 0; if it's less than 0, display a message but don't quit the program.
   (f) Create a new variable, `aList`, which is composed of all elements in either `aiX` or `astrY`.
   (g) Create a new variable, `strMyString`, where the `12` in `strNewString` is replaced with your favorite number.
   (h) Create a new variable, `aiKeys`, which consists of the keys of `hZ`.
   (i) Create a new variable, `aiSortedKeys`, which consists of the sorted keys of `hZ`.

Making a Python Script

Now that we've gotten a bit of a feel for the operators and functions, let's try to make a script out of this code! But first, a few tips for good programming practice:

- Save any changes before your run; if you don't, python will run the old version because that's the only one that exists. If you think you fixed an error and it's still popping up, double check that you saved the script.
- Use the interpreter! The interpreter is your friend; use it to try out how a bit of code will work *before* you stick it in your script. This can save you a lot of headache in the long run.

Now on to the real thing:

1. Open a new file in the Python editor of your choice, and save it as `lab02_script.py`.
2. Put the "shebang" as the first line of your file
3. Make the next four lines a docstring:

   ```
   """
   <your name>
   <today's date>
   """
   ```

4. Make the `__name__ == "__main__"` block by typing `if __name__ == "__main__":`
5. Within this block, create the variable `strMessage = "Hello, World!"` and print the message.
6. Go to the command line and run your program (`python <path-to-lab02_script.py>`). You should see the output "Hello, World!"
7. Now that we see your script is working, we can make things a bit more realistic. Go back to your editor, and within the `__name__ == "__main__"` block, create the variable `strName` which stores your name as a string.
8. Make a function called `funcGreet` (outside the `__name__ == "__main__"` block) which takes as input a string and prints `"Hello, <string>!"`.
9. Add `funcGreet(strName)` to the `__name__ == "__main__"` block.
10. Run your script again; this time you should see two outputs!
11. Now, create another function called `funcDivSum` that takes as input two lists and returns the ratio of their sums. Since division by `0` is not good, your function should raise an exception if the sum of either is zero.
12. In your `__name__ == "__main__"` block, make two lists, `aList1` and `aList2`. The sum of one of them should be 0. Create the variable `dRatio` which is the result of calling the function with those two lists. Make sure to print `dRatio`!
13. Now run your script again; you should see your error message.
14. Lastly, go back to the script and make sure that neither `aList1` nor `aList2` have a sum of 0. Save and run: you should see three outputs.

Practicing Python and Performance Evaluation

For some more Python practice, we will continue with the `iPython` mini-lab from class. If you would like, you can also work on your homework.

Operator (operation)	Syntax	Meaning	Example
`.` (dot)	*target.method*	Runs the *method* of *target*	`hashD.get("key")`
`#` (comment)	`# This is a comment`	Makes the interpreter ignore the remainder of the line after `#`	`1 + 2 # This should output 3`
`=` (assignment)	*variable* `=` *instruction*	Stores the return value of *instruction* in *variable*	`ia = 12` `ai = [1,3,7]`
`()` (parentheses)	(*grouping*)	Group the operations in *grouping* together.	`(12 + 13)*3`

`in` (inclusion)	*variable* `in` *collection*	Return `True` if *variable* is a memeber of *collection* and `False` otherwise.	`a = 12`
			`if a in [1,2,3]:`
			`  print( a )`

Arithmetic operations:
`+` (addition)
`-` (subtraction)
`*` (multiplication)
`/` (division)
	
	`x + y`
	`x - y`
	`x * y`
	`x / y`
		Returns the numeric value of performing the given arithmetic operation on `x` and `y`. Note that if `x` and `y` are both integers, the return value will be an *integer* as well.	
			`3 + 4`
			`3 - 4`
			`3*4`
			`3/4`
			`3/4.0`

Assignment operations:
`+=` (addition)
`-=` (subtraction)
`*=` (multiplication)
`/=` (division)
	
	`x += y`
	`x -= y`
	`x *= y`
	`x /= y`
		Equivalent to:
		`x = x+ y`
		`x = x - y`
		`x = x * y`
		`x = x/y`
		Assigns to `x` the arithmetic operations of `x` and `y`. Recall that arithmetic operations return integers if both inputs are integer.
			`x = 3`
			`x += 4`
			`x -= 4 x *=5`
			`x /= 4`
			`x /= 2.0`

`**` (power)	`x ** y`	Returns x^y.	`3**2`
`%` (modulo)	`x % y`	Returns the remainder of dividing `x` by `y`	`10 % 3`

Equivalence operators:
`==` (equal)
`!=` (not equal)
	
	`x == y`
	`x != y`
		Returns a Boolean value indicating whether `x` and `y` are equal (`==`) or not equal (`!=`).	`x = 12`
			`x ==12`
			`x != 13`

Numerical Comparisons:
`>` (greater than)
`<` (less than)
`>=` (greather than or equal to)
`<=` (less than or equal to)
	
	`x > y`
	`x < y`
	`x >= y`
	`x <= y`
		Returns a Boolean value indicating whether the comparison of the two numbers is `True`. Note that `x > y` and `x < y` return `False` if `x == y`.	
			`3 > 3`
			`3 < 4`
			`2 >= 3`
			`3 <= 3`

`not` (inversion)	`not` *test*	Returns the opposite Boolean value of that returned by *test*.	`not 4==3`

`and`	*test1* `and` *test2*	Returns `True` if *both test1* and *test2* return `True`. You can use one or more `and`s in one statement.	`x = 2`
			`y = 3`
			`z = [1,2,4,5]`
			`if x == y and x in z:`
			`  print( x )`

`or`	*test1* `or` *test2*	Returns `True` if either *test1* or *test2* return `True`. You can use one or more `or`s in one statement.	`x = 2`
			`y = 3`
			`z = [1,2,4,5]`
			`if x in z or x==y:`
			`  print(y)`

`+` (concatenation)	`x + y` where `x` and `y` are the same type (strings or lists)	Returns a string or list consisting of the combined letters or elements of `x` and `y`.	`"Hello, " + "world!"` `[1,3] + [2,5]`

`*` (repetition)	`x*y`
	where `x` is a string or list and `y` is an integer
		Returns the string or list repeated `y` times.	`"Hello!  "*3`
			`[1,2]*0`
			`[2,7,6]*2`

`:` (slice)	`x[y:z]`
	where `x` is a string or list and `y` and `z` are integers
		Returns the substring or sublist of `x` beginning at index `y` up to but not including index `z`.	`ai = [2,7,6]`
			`strA = "Hello!"`
			`ai[1:3]`
			`strA[2:4]`

Table 3: Some useful/common Python operators.

Function	Syntax	Meaning	Example

`len`	`len( list )`
	`len( hash )`
		Gives the length of *one* list or hash table which is the argument.	`hashD = {'a':12,'b':19}`
			`ai = [12,19,13]`
			`len(hashDictionary)`
			`len(ai)`

`range`	`range(end)`
	`range(begin, end)`
	`range(begin, end, step)`
		With one argument, return a list from `0` to *`end`* `- 1`. With two arguments, return a list from *`begin`* to *`end`*`-1`. With three arguments, increment each list element by *`step`*.	`range(12)`
			`range(-1,3)`
			`range(3,13,2)`

`raise`	`raise Exception(` *`message`* `)`	Terminate program execution and print *`message`*	`raise Exception( "the dead" )`
Type-Independent Functions			
`str`	`str(x)`	Converts `x` to a string.	`str(5)` `str([1,2])`
`int`	`int(x)`	Converts `x` to an integer	`int(2.3)` `int("4")`
`float`	`float(x)`	Converts `x` to a floating point value	`3/float(4)` `float("5.67")`
`cmp`	`cmp(x,y)`	Returns `-1` if `x < y`, `0` if `x==y` and `1` if `x > y`.	`cmp(3.5,4)` `cmp("abc","123")`
Numeric Functions			
`abs`	`abs(x)`	Returns the absolute value of numeric `x`	`abs(-3)`
`round`	`round(x)`	Returns the floating point representation of the closest integer of numeric `x`	`round(5.7)` `round(-4.2)`
`min` `max`	`min(list)` `max(list)`	Returns the minimum or maximum of a list of numbers.	`min([2,1,4,-1])` `max([5,-12])`
`sum`	`sum(list)`	Returns the sum of a list of numbers.	`sum([2,-1,3])`
String Functions			
`print`	`print(string)`	Displays *`string`* on the screen, followed by a newline.	`print("Hello, world!")`
`.strip`	*`string`*`.strip()`	Returns a new copy of *`string`* with the white space removed from beginning and end.	`"\tString\t\n".strip()` `"I am a String".strip()`
`.find`	*`string`*`.find(`*`strFind`*`)`	Returns index of the first occurrence of *`strFind`* in *`string`*, or `-1` if *`strFind`* is not in *`string`*. Note that this is case sensitive!	`"A string".find("A")` `"A string".find("12")`
`.replace`	*`string`*`.replace(`*`strFind, strReplace`*`)`	Returns a copy of *`string`* in which all instances of *`strFind`* have been replaced by *`strReplace`*.	`"ID\tValue\t\n".replace("\t",",")`
List (array) Functions			

`.append`	*`list`*`.append(`*`elementToAdd`*`)`	Adds *`elementToAdd`* at the end of *`list`*.	`a = ["a","b"]`
			`a.append("c")`
			`print(a)`

`.pop`	*`list`*`.pop()`	Removes the last (highest index) element from *`list`*.	`a = ["a","b","c"]`
			`a.pop()`
			`print(a)`

`.reverse`	*`list`*`.reverse()`	Reverses the list *`list`*. This *does not* return a new copy; it modifies *`list`* directly.	`a = [3,2,1]`
			`a.reverse()`
			`print(a)`

`sorted`	`sorted(`*`list`*`)`	Returns a sorted copy of the list *`list`*.	`sorted([2,3,1])` `sorted(["d","b","c"])`
Dictionary (hash) Functions			
`.keys`	*`hash`*`.keys()`	Returns a list of all the keys for dictionary *`hash`*.	`hashA = {"a":12,"c":19}` `hashA.keys()`
`.values`	*`hash`*`.values()`	Returns a list of all the values for dictionary *`hash`*.	`hashA = {"a":12,"c":19}` `hashA.values()`

Table 4: Some useful/common Python functions.