

# BST281

## Lab Session 6

### **Announcement:**

- Problems2 was posted. Deadline is March 10<sup>th</sup> (this Friday).
- Keep updated with class progress (Lecture/Lab/Canvas announcement).
- Reminder: March 1, 6, 8 are for journal club presentations.
- Today's lab is modified from **Lab 4** material, focus on
  1. read and write files
  2. debugging.

# Debugging

## Debugging?

Debugging is the process of figuring out what is going wrong with your code. Today we will talk about *print statement debugging*. It is a great start for beginners, there are some other more specialized ways or functions but we won't see it in this lab. Debugging using the print statement, it's a great way to develop intuition about what your code is doing and where you should look to fix any problem.

For debugging you require to read your code and read your error messages. Try to narrow down the general area where your errors are occurring and where the bug may be.

Tips for debugging:

- If you are coding a formula, try to look at intermediate calculations. Make your own test case, a simple calculation that you can do using a calculator or excel, then test your code.
- If you are reading a file and you want modify it or get certain data in some specific form. Add the print statment in intermediate steps to get a sense what your code is actually doing ( what colums are you reading, are the numbers floats or strings?, etc...).
- When using if, if/else, elif statment you can test your code by adding print statement after each case. Make test cases to see that each case runs when it supposed to.

# Doctest

<https://docs.python.org/2/library/doctest.html>

A way to check your answer:

- `python -m doctest problems02.py`

Example online:

If you run `example.py` directly from the command line, `doctest` works its magic:

```
$ python example.py
$
```

There's no output! That's normal, and it means all the examples worked. Pass `-v` to the script, and `doctest` prints a detailed log of what it's trying, and prints a summary at the end:

```
$ python example.py -v
Trying:
    factorial(5)
Expecting:
    120
ok
Trying:
    [factorial(n) for n in range(6)]
Expecting:
    [1, 1, 2, 6, 24, 120]
ok
Trying:
    [factorial(long(n)) for n in range(6)]
Expecting:
    [1, 1, 2, 6, 24, 120]
ok
```

# Common errors

## Syntax Errors

Fix the syntax errors for the following examples:

```
[>>> a=3+4 5
File "<stdin>", line 1
    a=3+4 5
        ^
SyntaxError: invalid syntax
[>>> print (hello word)
File "<stdin>", line 1
    print (hello word)
          ^
SyntaxError: invalid syntax
[>>> print "hello word"
File "<stdin>", line 1
    print "hello word"
          ^
SyntaxError: Missing parentheses in call to 'print'

[>>> hello=5
[>>> print(hello)
5
```

```
def mean():
    return sum(aList)/len(aList)
>>> mean()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    mean()
  File "<stdin>", line 2, in mean
    return sum(aList)/len(aList)
NameError: global name 'aList' is not defined
```

# Useful Unix commands for working with Files

Command	Usage	Purpose	Examples
cat	cat <file-name>	Displays all the contents of <file-name> in the terminal.	cat problems02.py
less	less <file-name>	Displays navigable contents of <file-name> in the terminal; you can scroll using ↑ or ↓. Close the file by typing q.	less problems02.py
head	head <file-name>	Displays the first 10 lines of <file-name>	head problems02.py
tail	tail <file-name>	Displays the last 10 lines of <file-name>	tail problems02.py
pipe( )	<cmd1>   <cmd2>	Redirects the output from <cmd1> to the input of <cmd2>	cat problems02.py   less
output redirect (>)	<cmd> > <output-file>	Redirects the output from <cmd> to the file <output-file>. <i>This overwrites &lt;output-file&gt;</i>	head problems02.py > problems02.short.py
output redirect and append (>>)	<cmd> >> <output-file>	Appends the output from <cmd> to the end of file <output-file>	tail problems02.py >> problems02.short.py
cut	cut -f <col numbers> [-d <delimiter>] <file-name>	Prints the <col numbers> columns of <file name>. If the delimiter is not a tab, use -d to set the delimiter.	cut -f 1-3,5 HMP_trunc.txt cut -f 1-3,5 -d , HMP_trunc.csv
wc	wc -l <file-name>	Prints the number of lines in <file-name>	wc -l HMP_trunc.txt

# Unix commands

## For Mac:

- less, head, tail
- |, > and >>
- *cut -f*
- cat
- *wc -l*
- *grep, grep -e*
- *man*
- *ls, lsl, lst, ls -tr*
- chmod

## For Windows:

- Get-Content (alias: gc) is your usual option for reading a text file.
- `gc log.txt | select -first 10 # head`
- `gc log.txt | select -last 10 # tail`
- `gc log.txt | more` # or less if you have it installed
- `type #cat`
- `find #grep`
- `help #man`
- `dir #ls`
- `attrib#chmod`

# Exercise: Unix commands

Download the file `HMP_trunc.txt` from the course website and use it to answer the following questions/do the following things:

1. Use `less`, `head` and `tail` to look at the file and get a feel for how large it is and what it contains.
2. How many lines are there?

# Exercise: Unix commands

Download the file `HMP_trunc.txt` from the course website and use it to answer the following questions/do the following things:

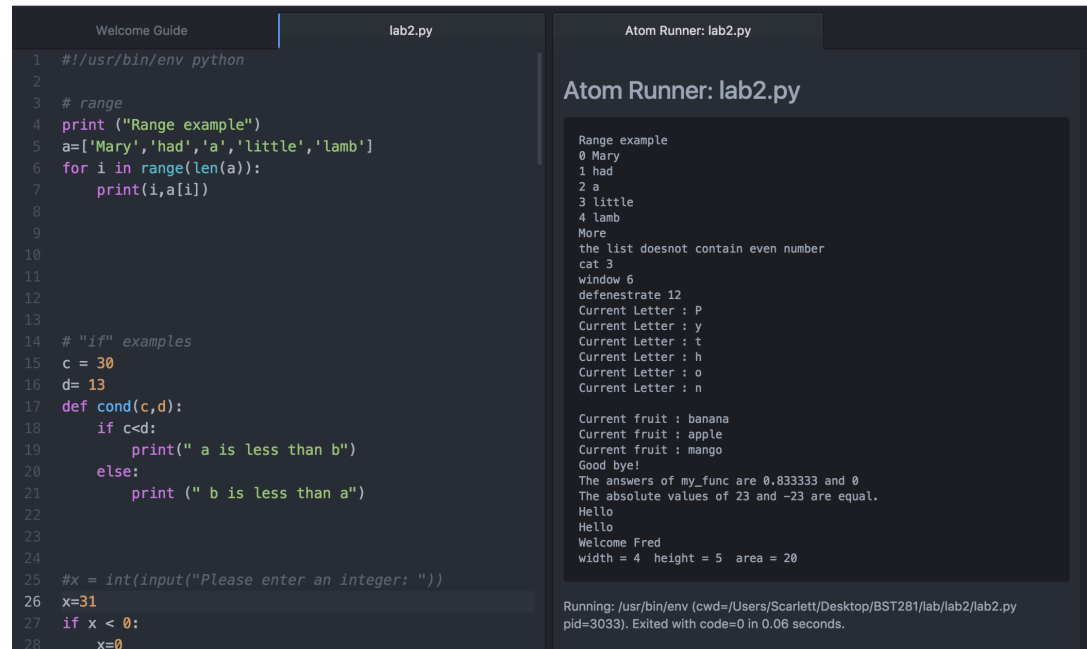
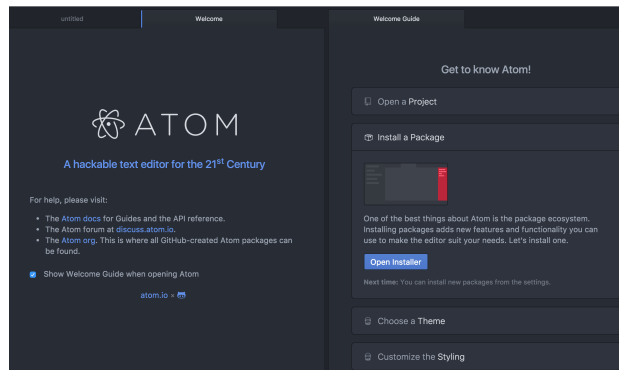
3. What does the first column contain?
4. How many rows correspond to Archaeal abundances? (HINT: those lines start with `k__Archaea`)
5. Extract the names of the Archaea and put them in the file `HMP_archaea_names.txt`



# Run python script in text editor



- Install Atom (text editor)
- Install runner package for Atom
- Ctl+R (mac) or Alt+R (windows)



# Reading and Writing Files within Python

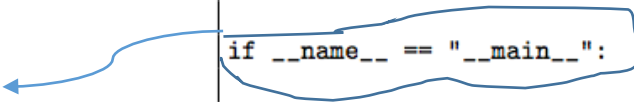
## `sys.argv`

The first task for this is to get comfortable with `sys.argv`. In a new text file called `lab6_ex1.py`, write the following:

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

import sys

if __name__ == "__main__":
    print sys.argv
```



We will ignore  
this for now  
We will talk about  
it in details after  
we cover python  
modules

This will print the arguments that Python sees from `sys.argv`. Now, run this script three different times:

```
1 python lab6_ex1.py
2 python lab6_ex1.py input.txt
3 python lab6_ex1.py input.txt output.txt
```

You should see:

```
[Xues-MacBook-Pro:lab6 Scarlett$ python lab6_ex1.py
['lab6_ex1.py']
[Xues-MacBook-Pro:lab6 Scarlett$ python lab6_ex1.py input.txt
['lab6_ex1.py', 'input.txt']
[Xues-MacBook-Pro:lab6 Scarlett$ python lab6_ex1.py input.txt output.txt
['lab6_ex1.py', 'input.txt', 'output.txt']
```

1. What kind of object is `sys.argv` (integer, dictionary, etc.)?

# Split Function ( from last lab)

## - Some Useful String Functions

We will learn some functions that can be used to deal with strings

- Start by opening the command line and type `python`

`.split()`

Now enter the following into the interpreter:

```
string1="Hello!World"
string1.split("\t")
string1.split("!")
string1.split("e")

string2=",Subj,Sequence1,Sequenece2\n"
string2.split('\t')
string2.split('\n')
string2.split(',')
string2.split('Subj')

string3="Sequence1\tSequence2"
string3.split('\t')
```

```
>>> string1="Hello!World"
>>> string1.split("\t")
['Hello!World']
>>> string1.split("!")
['Hello', 'World']
>>> string1.split("e")
['H', 'llo!World']
>>> string2=",Subj,Sequence1,Sequenece2\n"
>>> string2.split('\t')
[' ,Subj,Sequence1,Sequenece2\n']
>>> string2.split('\n')
[' ,Subj,Sequence1,Sequenece2', '']
>>> string2.split(',')
[' ', 'Subj', 'Sequence1', 'Sequenece2\n']
>>> string2.split('Subj')
[' ', ' ', 'Sequence1,Sequenece2\n']
>>> string3="Sequence1\tSequence2"
>>> string3.split('\t')
['Sequence1', 'Sequence2']
```

1. What does `.split(<string>)` do?

**Ans: It splits a string into a list by removing all instances of from the string.**

2. Is `.split()` a function that modifies the value in place, or that creates a copy and returns it?

**Ans: It creates a copy and returns it.**

# Replace Function

We will learn some functions that can be used to deal with strings

- Start by opening the interpreter (open the command line and type `python`).
- `.split()`

Now enter the following into the interpreter:

```
[>>> string1="Hello!Word"
[>>> string1.replace('H','A')
'Aello!Word'
[>>> string1.replace('H',' ')
' ello!Word'
```

## Append to a list

We will learn how to append elements to an empty list

- Start by opening a new py file in Atom

```
1 aList=[]
2
3 for i in range(10):
4     aList.append(i)
5
6 print (aList)
```

## Using sys.argv and open

In a new text file called `lab6_ex2.py` type the following:

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

import sys

if __name__ == "__main__":

    fInFile = open(sys.argv[1])
    fOutFile = open(sys.argv[2], 'w')
    for strLine in fInFile:
        fOutFile.write(strLine.split('\t')[0])

    fInFile.close()
    fOutFile.close()
```

The input file for this function will be `HMP_trunc.txt` and the output file will be `HMP_ex6.txt`. When you run this correctly, you will not see any output on the screen.

1. What does this function do?
2. What is the correct command to run this file?
3. What does your output file look like? How could you change your code to make it more readable?

# Exercises

1. Write a script called `txt2csv.py` that converts a tab-delimited (txt) file into a comma-separated value (csv) file. Choose an I/O style between:

(a) `python txt2csv.py < input.txt > output.csv`

(b) `python txt2csv.py input.txt output.csv`

Use this script to convert `HMP_trunc.txt` to `HMP_trunc.csv`.

# Exercises (challenging)

2. Write a script called `transpose.py` that takes an input file and outputs its transpose. You can assume that your input file is tab-delimited and that any missing entries are designated by a tab. Choose an I/O style between:

(a) `python transpose.py < input.txt > output.txt`

(b) `python transpose.py input.txt output.txt`

Use this script to convert `HMP_trunc.txt` to `HMP_trunc_transpose.txt`.

# Homework 2 questions?