Lab: Day 1

Media Computation

What you'll do:

Changing colors

Negatives

Grayscale

Making sunsets

What you'll learn:

Jython Basics

Calling Functions

Assigning Variables

For loops

Exercise 0: Getting Started

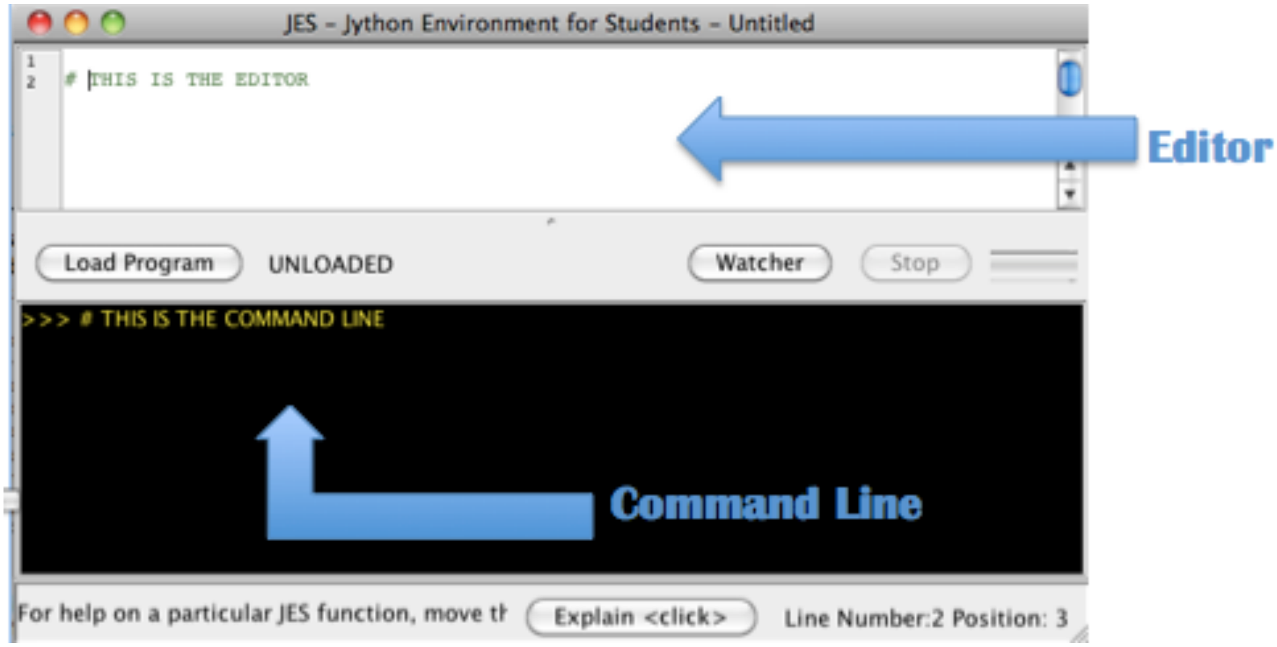
Instructions:

- Go to *Places* – the drop-down menu in the upper left corner
- Go to *Home Folder*
- Go to *JES-4-3-nojava*   *#this is the folder you will be working in*
- Create a **"lab1"** folder.

Big Picture:

We want you to feel comfortable with the interpreter.

Interpreter:


JES – Jython Environment for Students – Untitled
Editor
Load Program
UNLOADED
Watcher
Stop
Command Line
Explain <click>
Line Number:2 Position: 3

```
# THIS IS THE EDITOR
```

```
>>> # THIS IS THE COMMAND LINE
```

For help on a particular JES function, move th

Activity:

- Discuss the difference between the editor and the command line with your partner.

- Try to think of some benefits of each.

- Ask a lab assistant or teacher for verification of your ideas.

Exercise 1: Jython Basics

"To succeed, you will soon learn, as I did, the importance of a solid foundation in the basics**"** -Alan Greenspan

Command-line

Type each of the following expressions into the Jython prompt `>>>`, in the command-line, ending the line with the `Enter` key. Predict the results before you type them!!! Some of these expressions might cause Jython to error.

Jython Expressions:

```
#this is a comment
3
2 + 3
5 + 6 + 7
-16 - -16
3 * 4 + 1
3 * (4 + 1)
from operator import add, mul
3 * 4
mul(3, 4)
mul(3, add(4, 1))
2 ** 3
pow(2, 3)
pow(pow(2, 3), abs(-2))
```

```
#do this column second
from math import sqrt, exp
exp(1)
sqrt(144)
pi
from math import pi
pi
pi * 3
print(pi)
print(4)
print(add(9,1))
print(print(2))
False or True
True and False
```

Assigning Variables

```
x = 3
x
x + 1
x
x = x + 1
x
y = 1
y = y + 2
y
x = y
```

Using the editor is next!

Exercise 1: (cont)

Command-line

Type each of the following expressions into the Jython prompt `>>>`, in the command-line, ending the line with the `Enter` key. Predict the results before you type them!!! Some of these expressions might cause Jython to error.

Strings

```
“kit-kat bar”
“I am” + “adding strings”
var = “I am a saved string”
var[0]
var[10]
var[100]
```

Lists

```
[1, 2, 3, 4]
[“I”, “can”, “contain”, “anything”]
[“different”, 1, 2, “data”, 8]
[1, 2] + [3, 4]
var = [1, 2, 3, 4]
var[0]
var[4]
var[5]
```

Exercise 2: Defining your own functions

Recall the structure of defining a function:

```
def <name>(<arguments names>):
   return <expression>
```

Command-Line

Exercise 2(a): At the python prompt >>>, type the following:

```
>>> def square(n):
...    return n*n
...
>>>
```

Be sure to indent the `return` statement correctly. Then, call the function `cube` with some numerical argument.

Editor

Exercise 2(b): Now we will use the code editor to write a function. Rewrite the following into the editor.

```
def doStuff(x, y):
  return y + x * x
```

Now load the program into the Jython interpreter and test your function, by calling `doStuff` with two numerical arguments at the prompt.

Exercise 2(c): So you decide that you want `doStuff` to actually square y and add x (the opposite of what it is doing now). Edit the function in the editor, and test your function again to make sure it's doing the right thing.

Exercise 3: PICTURES!

Exercise 3(a): Collect the photos

Instructions:

- Download pictures you want to work with and save them to your “lab1” folder.
  - Make sure all of your pictures are of medium size or smaller! **< 800x800 pixels**
  - Make sure all of your pictures are .jpg
  - When you Google images, you can filter by size on the left column

Exercise 3(b): Manipulating Pictures

Experiment:

- Try to remember what we went over in lecture in the first demo!
- Make a photo appear by typing into the editor and loading your program

* You can look at what built-in functions we have by going to the “JES functions” drop-down menu

In the editor:

- make a file using your filepath* and assign that to a variable called `myFile`
- make a picture out of the file and assign that to a variable called `myPic`
- show that picture

Load your program so that JES shows your picture!

** Your filepath should be equivalent to “lab1/filename.jpg”*

Exercise 3(c): `pickAndShow`

- Define a function, pickAndShow—that does all of the above in one swoop—in your editor. It should take no arguments.
- Here’s an example in pseudocode:

```
def pickAndShow()
  - Has the user pick a file and assigns that to a variable
  - Turns that file into a picture and assigns that to variable
  - Shows that picture
```

- Once you have defined the function, call it inside of your editor
- Now save and load the program *# you should be able to choose a picture for it to show*

Handy built-in functions that you *might* find helpful in 1(b) and 1(c):

- pickAFile() – when called, allows the user to choose a file
- makePicture(someFile) – when called with some file, returns the picture form of the file
- show(somePicture) – displays the picture it is called with

Exercise 3(d): `decreaseRed`

Big Picture:

We want you to call the function `decreaseRed`, provided below, on a picture and display the result.

Note: We aren't having you write the function for yourself yet! We are just trying to get you comfortable with calling such functions!

Here is the code, type it into your editor:

```
def decreaseRed(picture):
      for p in getPixels(picture):
         value=getRed(p)
         setRed(p,value*0.5)
```

Call this function on a picture with red in it, and show the result!

Things to keep in mind:

1. We need a picture to call this function on, so be sure to make one.
2. You want to write this all out inside of your editor.

Experiment:

What if you decrease red again and again and again…? Try it!

Exercise 3.5: Extras

Only do these if you have extra time. Or you can help other people. Or both!

Recall the structure of defining a function:

```
def <name>(<arguments names>):
    return <expression>
```

Attempt to write any of the following functions in python for practice

1. **distance:** given two coordinates, which can be a list of two numbers (i.e. [1, 2]), attempts to find the distance between these two points.

```
def distance(point1, point2):
  your code here

>>> distance([1,0] , [1,3])
3
```

2. **Leap year:** given a year, determine if that year is a leap year.

Rules:

- If a year is divisible by 400 then it is a leap year.
- If a year is divisible by 100 and not by 400, then it is **not** a leap year.
- If a year is divisible by 4 then it is a leap year
- Otherwise, it is **not** a leap year.

```
def leapyear?(year):
  your code here

>>> leapyear?(1987)
3
```