Homework

• Posted to Class Website on Wednesdays
• Due on Wednesdays at 5pm
• Submit via email to me: rachel36@uw.edu
  – Name as: Lastname_Firstname_PS#_whatever.ext
    eg: Bradshaw_Rachel_PS1_answers.docx
    Bradshaw_Rachel_PS1_code.py

• Written Exercises
  – MS Word, OpenOffice, or txt format

• Code/result sets
  – Text format ONLY! (extension can be .py, .txt, etc)
Saving your work

• If using a lab computer:
  – Bring in a memory stick to save your scripts, datasets, etc
  – Get a dropbox account
  – Email yourself the files

• Or, bring your own laptop
Computer Setup

Python

![Python Logo](http://www.python.org/download/releases/2.7.3/)

Command Shell

Mac:

1. Applications
2. Utilities
3. Terminal

PC:


Text Editor

Mac: Text Wrangler
PC: Notepad++, Text Pad

![Text Editor Screenshot](www.notepad-plus-plus.org)

![Text Editor Screenshot](www.barebones.com/products/TextWrangler/download.html)

![Text Editor Screenshot](www.textpad.com)
Two ways to run python

Via Interpreter
Enter command
See result
Enter next command
See next result...

Saved Scripts
Write your program
Run it
Change it a bit
Run it again

Use the interpreter to test syntax, or to try commands that you’re not sure will work when you run your program. You don’t write programs in the interpreter.
Running the python interpreter

- Start a terminal session
- Type “python”
- This should start the Python interpreter
The interpreter

• Try printing various things (in your spare time)
  – Leave off the quotation marks
  – Print numbers, letters, and combinations
  – Print two things, with a comma between them
  – Enter a mathematical formula
  – Leave off the word “print”
Your first program

• In your terminal, Ctrl-D out of the python interpreter.
• Type "pwd" to find your present working directory.
• Open TextWrangler.
• Create a file containing one line:
  
  ```python
  print "hello, world!"
  ```
• Be sure that you end the line with enter.
• Save the file as "hello.py" in your present working directory.
• In your terminal, type "python hello.py"

```
> python hello.py
hello, world!
```

(This tells the computer “use python to run the program hello.py”. Yes, the result is somewhat anticlimatic.)
From yesterday: variables

- A variable is like a box with a name.
- You can put a value in the box.
- Values have types, (e.g. int, float, str, list, dict)
From yesterday: variables

- A **variable** is like a box with a name.
- You can put a value in the box
- Values have types, (e.g. int, float, str, list, dict)

- For example, we can assign the name “pi” to the object 3.14159, as follows:
  
  ```python
  pi = 3.14159
  ```

- You can refer to variables by name later on to do things with the values they hold
  
  ```python
  pisq = pi * pi
  ```
The `import` command

• Many python functions are available only via “packages” that must be imported (other functions are always available – called “built-in”)

```python
>>> print log(10)
Traceback (most recent call last):
  File <pyshell#0>, line 1, in <module>
NameError: name 'log' is not defined
```
The `import` command

- Many Python functions are available only via “packages” that must be imported (other functions are always available – called “built-in”)

```python
>>> print log(10)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
NameError: name ‘log’ is not defined

>>> import math
>>> print math.log(10)
2.30258509299

>>> print log(10)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
NameError: name ‘log’ is not defined
```
The command line

• To get information into a program, we can use the command line.
• The command line is the text you enter after the word “python” when you run a program.
• The zeroth argument is the name of the program file.
• Arguments larger than zero are subsequent elements of the command line, separated by spaces.

> python my-program.py 1 abc D E

`zeroth argument` `first argument` `second argument`
Reading command line arguments

• We need the sys package in order to read command line arguments. We can write a program like this:

```python
import sys
print sys.argv
print sys.argv[0]
print sys.argv[1]
```

Which will produce this when run:

```
> python my-program.py 1 abc D E
my-program.py
1
```
Class problem #1

• Write a program called “print-two-args.py” that reads the first two command line arguments after the program name, stores their values as variables, and then prints them on the same line with a colon between.

> python print-two-args.py hello world
hello : world

Hint- to print multiple things on one line, separate them by commas:

>>> print 7, “pickles”
7 pickles
import sys
arg1 = sys.argv[1]
arg2 = sys.argv[2]
print arg1, "::", arg2
Class problem #2

• Write a program called “add-two-args.py” that reads the first two command line arguments after the program name, stores their values as variables, and then prints their sum.

```python
> python add-two-args.py 1 2
3.0
```

Hint: to read an arguments as a decimal number, use the syntax:
```python
foo = float(sys.argv[1])
```

Or for an integer number:
```python
bar = int(sys.argv[1])
```

The technical name for this is “casting” - the argument starts as a string object and is cast to a float or int object (two kinds of Number objects in Python).

Command line arguments always start as String objects
import sys
arg1 = float(sys.argv[1])
arg2 = float(sys.argv[2])
print arg1 + arg2

Notice that this expression gets evaluated first, then printed
Challenge problems

• Write a program called “circle-area.py” that reads the first command line argument as the radius of a circle and prints the area of the circle.

> python circle-area.py 15.7
774.371173183

• Do the same thing but read a second argument as the unit type and include the units in your output.

> python circle-area2.py 3.721 cm
43.4979923683 square cm
import sys
radius = float(sys.argv[1])
print 3.1415 * radius * radius

(or slightly better)

import sys
import math
radius = float(sys.argv[1])
print math.pi * radius * radius

the math package contains most simple math constants and functions that are not built in

the math constant pi

import sys
import math
radius = float(sys.argv[1])
units = sys.argv[2]
print math.pi * radius * radius, "square", units

a literal string
Reading and follow-up

• Chapter 1 of *Python for Software Design* by Downey (PDF linked via course site)

• Practice writing, saving, running python scripts on your computer.