6.189: Introduction to Programming in Python
Session 1

Course Syllabus

1. Administrivia. Variables and Types.
2. Functions. Basic Recursion.
3. Control Flow: Branching and Repetition.
4. Introduction to Objects: Strings and Lists.
5. **Project 1:** Structuring Larger Programs.

6. Python Modules. Debugging Programs.
7. Introduction to Data Structures: Dictionaries.
8. Functions as a Type, Anonymous Functions and List Comprehensions
9. **Project 2:** Working in a Team.
10. Quiz. Wrap-up.

Administrivia

1. **Grading.** Well, it's a pass/fail course – attend the classes, do the homework. Attendance is important: email us in advance if you have to miss a class.

2. **Optional Assignment(s).** There's an optional assignment that you can work on when you have free time, e.g. if you finish the class lab period early. This assignment is *completely optional* – it was designed to give you a better intuition for programming and to help in later course 6 studies.

3. **Textbook.** This class uses readings from the online textbook *How to Think Like a Computer Scientist* – its always nice to have two perspectives on concepts. You can find this textbook at the following link:

http://www.openbookproject.net/thinkcs/python/english2e/index.xhtml

Notes/Homework

1. **Install Python.** You can get the latest version at *www.python.org*.

- Linux and OS X users: You should already have Python installed – to test this, run the command *python* in console mode. Note that the first line tells you the version number: you may want to upgrade your installation if you have a version earlier than **2.3**.

- Windows users: Grab the windows installer from the Downloads section. After installation, you can run the *IDLE (Python GUI)* command from the Start menu.

2. **Reading.** Read 1.1-1.2, 1.8-1.9, and chapter 2 from the textbook.

3. **Writing Programs.** Recall that a program is just a set of instructions for the computer to execute. Let's start with a basic command:

print *x*: Prints the value of the expression *x*, followed by a newline.

Program Text:

```
print "Hello World!"
print "How are you?"
```

Output:

```
Hello World!
How are you?
```

(The compiler needs those quotation marks to identify strings – we'll talk about this later.) Don't worry, we'll be adding many more interesting commands later! For now, though, we'll use this to have you become familiar with the administrative details of writing a program.

Write a program that, when run, prints out a tic-tac-toe board.

- Linux and OS X users: To write your program, just create a text file with the contents of the program text (e.g. the program text above.) To run your program (pretend its named *first_print.py*), use the command *python first_print.py* at the shell.

- Windows users: The best way to program in Windows is using the IDLE GUI (from the shortcut above.) To create a new program, run the command *File->New Window (CTRL+N)* – this will open up a blank editing window. In general, save your program regularly; after saving, you can hit F5 to run your program and see the output.

Make sure to write your program using the computer at least once! The purpose of this question is to make sure you know how to write programs using your computing environment; many students in introductory courses experience trouble with assignments not because they have trouble with the material but because of some weird environment quirk.

You can write the answer to the question in the box below (I know its hard to show spaces while writing – just do your best)

Program Text:

Expected Output:

```
 | |
-----
 | |
-----
 | |
```

4. **Interpreter Mode.** Python has a write-as-you-go mode that's useful when testing small snippets of code. You can access this by running the command *python* at the shell (for OS X and Linux) or by starting the IDLE GUI (for Windows) – you should see a >>> prompt. Try typing a print command and watch what happens.

You'll find that its much more convenient to solve a lot of the homework problems using the interpreter. When you want to write an actual, self-contained program (e.g. writing the game tic-tac-toe), follow the instructions in section 3 above.

5. **Variables.** Put simply, variables are containers for storing information. For example:

Program Text:

```
a = “Hello World!”
print a
```

Output:

```
Hello World!
```

The = sign is an assignment operator which says: assign the **value** “Hello World!” to the variable *a*.

Program Text:

```
a = “Hello World!”
a = “and goodbye..”
print a
```

Output:

```
and goodbye..
```

Taking this second example, the value of *a* after executing the first line above is “*Hello World!*”, and after the second line its “*and goodbye..*” (which is what gets printed)

In Python, variables are designed to hold specific **types** of information. For example, after the first command above is executed, the variable *a* is associated with the **string** type. There are several types of information that can be stored:

- **Boolean**. Variables of this type can be either *True* or *False*.
- **Integer.** An integer is a number without a fractional part, e.g. *-4*, *5*, *0*, *-3*.
- **Float.** Any rational number, e.g. *3.432*. We won't worry about floats for today.
- **String.** Any sequence of characters. We'll get more in-depth with strings later in the week.

Python (and many other languages) are zealous about type information. The string "*5*" and integer *5* are completely different entities to Python, despite their similar appearance. You'll see the importance of this in the next section.

Write a program that stores the value 5 in a variable *a* and prints out the value of *a*, then stores the value 7 in *a* and prints out the value of *a* (4 lines.)

Program Text:

```

```

Expected Output:

```
5
7
```

6. **Operators.** Python has the ability to be used as a cheap, 5-dollar calculator. In particular, it supports basic mathematical operators: +, -, *, /.

Program Text:

```
a = 5 + 7
print a
```

Output:

```
12
```

Variables can be used too.

Program Text:

```
a = 5
b = a + 7
print b
```

Output:

```
12
```

Expressions can get fairly complex.

Program Text:

```
a = (3+4+21) / 7
b = (9*4) / (2+1) - 6
print (a*b)-(a+b)
```

Output:

```
14
```

These operators work on numbers. Type information is important – the following expressions would result in an *error*.

"Hello" / 123   "Hi" + 5   "5" + 7

The last one is especially important! Note that Python just sees that 5 as a string – it has no concept of it possibly being a number.

Some of the operators that Python includes are

- **Addition, Subtraction, Multiplication**. *a+b*, *a-b*, and *a*b* respectively.
- **Integer Division**. *a/b.* Note that when division is performed with two integers, only the quotient is returned (the remainder is ignored.) Try typing *print 13/6* into the interpreter
- **Exponentiation** (a^b). a ** b.

Operators are evaluated using the standard order of operations. You can use **parentheses** to force certain operators to be evaluated first.

Let's also introduce one string operation.

- **Concatenation**. *a+b.* Combines two strings into one. *"Hel" + "lo"* would yield *"Hello"*

Another example of type coming into play! When Python sees a + b, it checks to see what type a and b are. If they are both strings then it concatenates the two; if they are both integers it adds them; if one is a string and the other is an integer, it returns an error.

Write the output of the following lines of code (if an error would result, write error):

print 13 + 6	Output: ____
*print 2 ** 3*	Output: ____
*print 2 * (1 + 3)*	Output: ____
print 8 / 9	Output: ____
print “13” + “6”	Output: ____
print “13” + 6	Output: ____

7. **Review.** Here are the important concepts covered today.

- What is a program? How does one write them.
- Using the interpreter mode.
- Variables: containers for storing information. Variables are **typed** – each variable only holds certain types of data (one might be able to store strings, another might be able to store numbers, etc.)
- Operators: +, -, *, /, **, and parentheses.