6.189 Homework 1

Readings

How To Think Like A Computer Scientist: Monday - chapters 1 (all), 2 (all) & 4.12.
Tuesday - Chapter 4.1, 4.2, 4.4-4.7, 6.1 & 6.2.

What to turn in

Turn in a printout of your code exercises, stapled to your answers to the written exercises, at 2 PM on Wednesday, January 5th.

Exercise 1.0 – Installing Python

Follow the instructions on installing Python and IDLE on your own computer on the **Materials** page of the course website, in the **Handouts** section. **Be sure to install Python 2.6.x**. Ask an LA for help if you run into any trouble. Before continuing, play around with the Python shell a bit and explore how you can use it as a calculator.

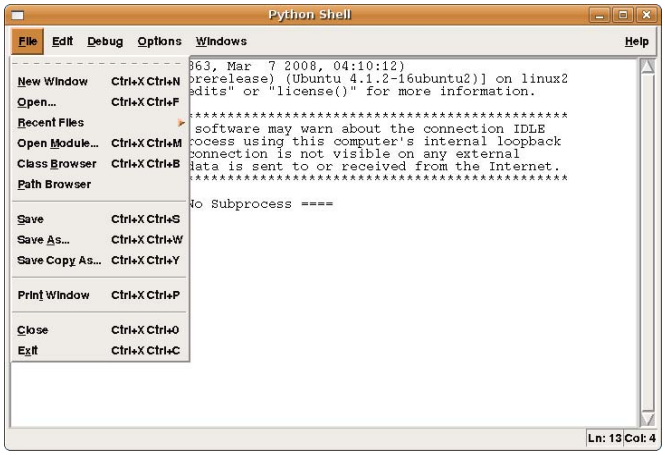
Exercise 1.1 – Hello, world!

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

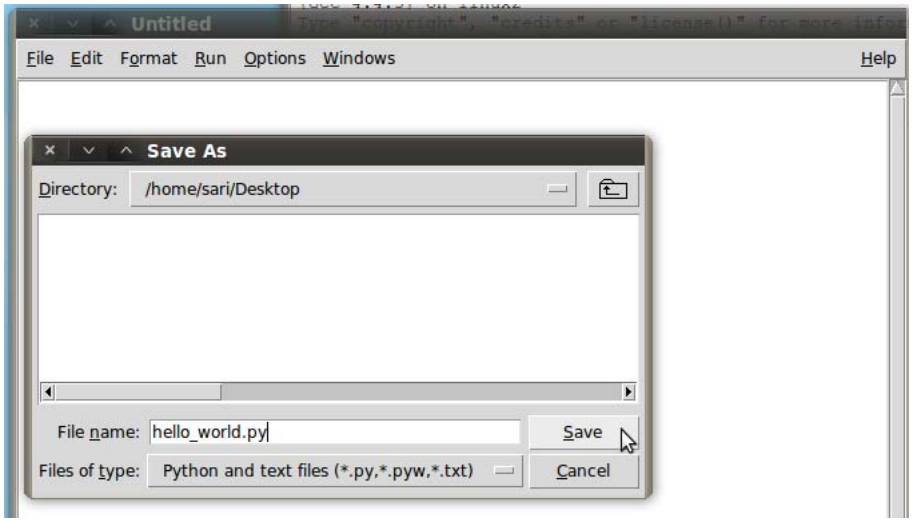
`print` x: Prints the value of the expression x, followed by a new line.

Create a new program called `hello_world.py`. You will use this file to write your very first 'Hello, world!' program, as well as your answers for the rest of the exercises for today. How to create a program file:

1. Open a new window by choosing **New Window** from the **File** menu.



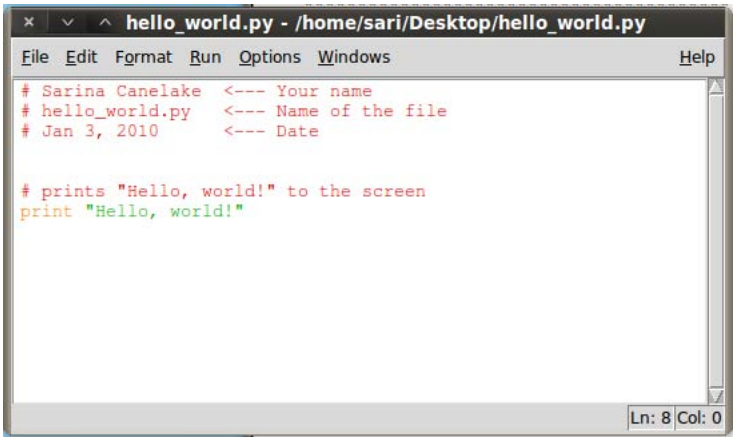
2. Save the file as `hello_world.py`. Do NOT skip the '.py' portion of the file name - otherwise, you will lose out on syntax highlighting!



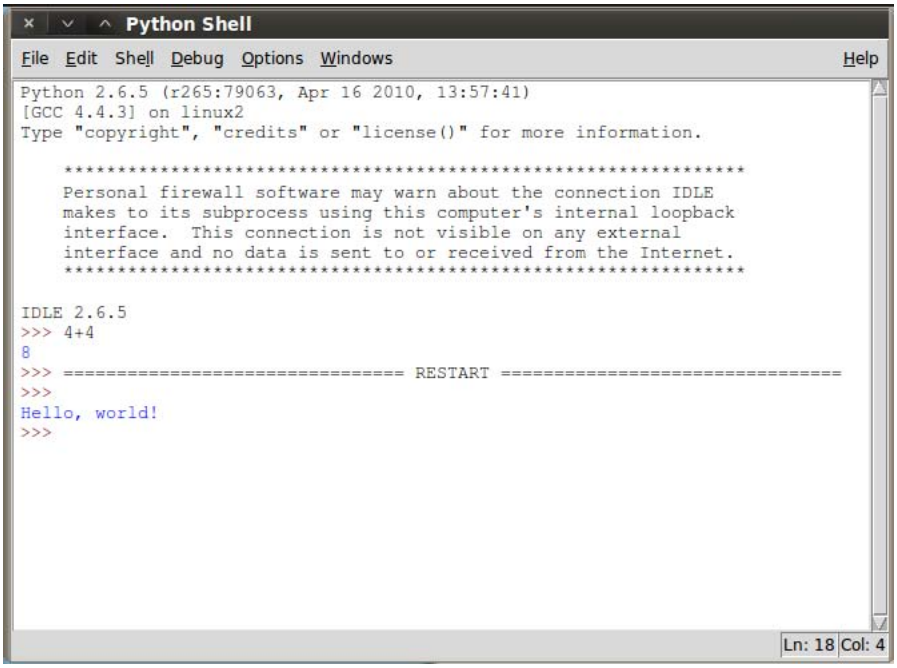
3. Start every program with a bank of comments, with a comment line for your name, your recitation section, the name of your file, and today's date. Recall that a comment line begins with a '`#`' (pound) symbol.

You can now write your very own Hello, world! program. This is the first program that most programmers write in a new programming language. In Python, Hello world! is a very simple program to write. Do this now... it should be only be one line!

When you are done, save your work and run it. Your code should look similar to this:



To run your program, chose **Run Module** from the **Run** menu (or just hit F5 on Windows/Linux, or fn-F5 on a Mac). When you run the code, your shell should look similar to this:



When you run your code, it first prints the line `>>> ===== RESTART =====`, then runs your code underneath that line.

Exercise 1.2 – Printing

From the course website, download the `homework_1.py` template. Remember to put your name and section at the top. If you don't we'll be highly grumpy.

Write a program using `print` that, when run, prints out a tic-tac-toe board. Remember to save your program regularly, to keep from losing your work! The purpose of this exercise is to make sure you understand 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.

Expected output:

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

Exercise 1.3 – Variables

Recall that 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 tells the interpreter to 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!’’. But, after executing the second line, the value of a changes to ‘‘and goodbye...’’. Since we ask the program to print out a only after the second assignment statement, that is the value that gets printed. If you wanted to save the values of both strings, you should change the second variable to another valid variable name, such as b.

Variables are useful because they can cut down on the amount of code you have to write. In homework_1.py, write a program that prints out the tic-tac-toe board from exercise 1.2, but which uses variables to cut down on the amount of typing you have to do. Hint - how many different variables should you need?

Exercise 1.4 – Operators/Order of Operation

Python has the ability to be used as a cheap, 5-dollar calculator. In particular, it supports basic mathematical operators +, -, *, / as well as the power operator (**) and the modulus operator (%).

Program Text:

```
x = 5 + 7
print x
y = x + 10
print y
```

Output:

```
12
22
```

Note that we can use variables in the definition of other variables! Mathematical operators only work on numbers- *ints* or *floats*. Statements such as `'Hi' + 5` or `'5' + 7` will not work.

Part I: Input the following sets of equations, and note the difference between *int* arithmetic and *float* arithmetic. You can do this just in your interpreter (you don't need to turn anything in for this part), but pay attention to the output!

1. $\frac{5}{2}, \frac{5}{2.0}$, and $\frac{5.0}{2}$ Note that as long as one argument is a float, all of your math will be floating point!
2. $7 * \left(\frac{1}{2}\right)$ and $7 * \left(\frac{1}{2.0}\right)$
3. $5 * *2$, $5.0 * *2$, and $5 * *2.0$
4. $\frac{1}{3.0}$ Note the final digit is rounded. Python does this for non-terminating decimal numbers, as computers cannot store infinite numbers! Take 6.004 to find out more about this...

Part II: In `homework_1.py`, transcribe the following equations into Python (without simplifying!), preserving order of operation with parenthesis as needed. Save each as the value of a variable, and then print the variable.

1. $\frac{3 \times 5}{2 + 3}$
2. $\sqrt{7 + 9} \times 2$
3. $(4 - 7)^3$
4. $\sqrt[4]{-19 + 100}$
5. $6 \mod 4$ - If you aren't familiar with modular arithmetic, it is pretty straightforward- the modulus operator, in the expression $x \mod y$, gives the remainder when x is divided by y. Try a couple modular expressions until you get the hang of it.

Part III: In `homework_1.py`, use order of operation mathematics to create two equations that look the same (ie, have the same numbers) but evaluate to different values (due to parenthesization). Save each as the value of a variable, then print the variables.

Exercise 1.5 – User input

Do this exercise in `homework_1.py`. In this exercise, we will ask the user for his/her first and last name, and date of birth, and print them out formatted. Recall that you can get input from the user using the command `raw_input(''text'')`, as shown in lecture.

Note: There are two functions to get user input. The first, `raw_input`, turns whatever the user inputs into a string automatically. The second, `input`, *preserves type*. So, if the user inputs an int, or a float, you will get an int or a float (rather than a string). Be careful though- you still want to use `raw_input` if you want a string back, or otherwise the user will have to put quotes around their answer. Use `raw_input` here - it's good for string processing, like this problem. `input` will come in handy when using user input to compute math - like in Exercise 1.8.

Here is an example of what this program should do:

Output:

```
Enter your first name: Chuck
Enter your last name: Norris
Enter your date of birth:
Month? March
Day? 10
Year? 1940
Chuck Norris was born on March 10, 1940.
```

To print a string and a number in one line, you just need to separate the arguments with a comma (this works for any two types within a print statement). The comma adds a space between the two arguments. For example, the lines:

```
mo = 'October'
day = '20'
year = '1977'
print mo, day, year
```

will have the output

```
October 20 1977
```

OPTIONAL: Now, for something completely different... a discussion on how to print strings, most prettily...

Note that none of the commas are in this output! To do that you want something like this:

`print mo, day+',', year`.

The `+` sign concatenates two strings, but can *only* be used on two strings. Using it on a number and a string will cause an error (because it is ambiguous as to what you want the program to do!)

That's why it's a great idea to use `raw_input` for this problem; if you use `input` you'd have to convert the int to a string. We'll cover this more in-depth on Thursday, when we get to strings, but you may want to play with string concatenation operations now to get everything to look its prettiest.

At this point, we suggest completing written exercises 1.9 - 1.11 to cement your understanding of Monday's topics. Exercises 1.6-1.8 and 1.12-1.15 cover Tuesday's topics.

Exercise 1.6 – New Operators

Open up IDLE and play around with the new operators showed today in class. Make sure that you understand how to use them and what they are used for! The operators `==, !=, <, >, <=, >=` are called *relation operators*. They work on all types, not just numbers, and return a *Boolean* (True/False) value. Remember, if you are using Booleans, to capitalize True and False! Here's an example shell session; try other examples you can think of.

```
>>> 5 >= 7
False
>>> 'abc' != 'def'
True
>>> x = 'abc'
>>> x == 'abc'
True
```

This next example is strange! Try to understand what's going on here, and ask if you're confused.

```
>>> a = True
>>> b = (5 < 7)
>>> a == b
True
```

Next, the operators `+=, -=, *=, /=` change the value of a stored variable in a quicker way. In the following example, we add 6 to a variable in two different ways; note that we get the same result! Try using all of these operators in your interpreter window before moving on.

```
>>> x = 5
>>> x = x + 6
>>> print x
11
>>> y = 5
>>> y += 6
>>> print y
11
```

We strongly suggest you *finish all the written exercises now*, before continuing on with the next code problem.

Exercise 1.7 – Rock, Paper, Scissors

In this exercise, you are going to practice using conditionals (if, elif, else). You will write a small program that will determine the result of a rock, paper, scissors game, given Player 1 and Player 2's choices. Your program will print out the result. Here are the rules of the game:

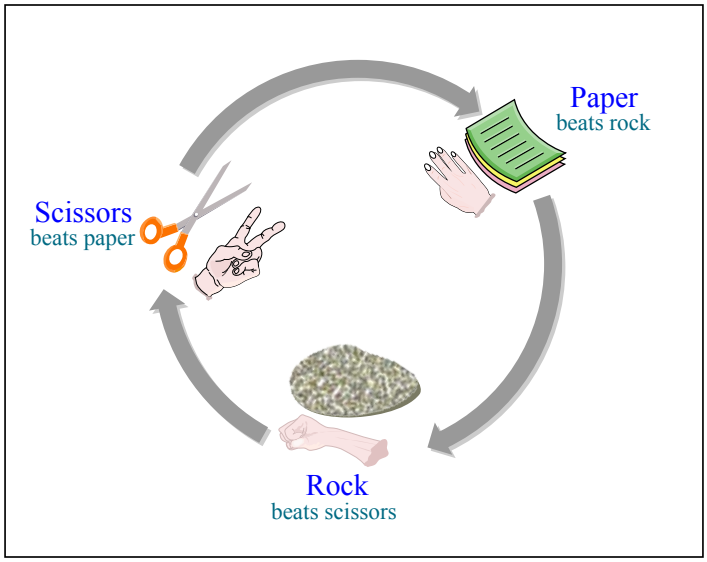


Image by MIT OpenCourseWare.

1. First create a truth table for all the possible choices for player 1 and 2, and the outcome of the game. This will help you figure out how to code the game!

Player 1	Player 2	Result
Rock	Rock	Tie
Rock	Scissors	Player 1

2. Create a new file `rps.py` that will generate the outcome of the rock, scissors, paper game. The program should ask the user for input and display the answer as follows:

```
Player 1? rock
Player 2? scissors
Player 1 wins.
```

The only valid inputs are rock, paper, and scissors. If the user enters anything else, your program should output "This is not a valid object selection". Use the truth table you created to help with creating the conditions for your if statement(s). Read on to the next page before starting...

Note If you have a long condition in your if statement, and you want to split it into multiple lines, you can either enclose the entire expression in parenthesis, e.g.

```
if (player1 == 'rock' and
   player2 == 'scissors'):
   print 'Player 1 wins.'
```

Or, you can use the backslash symbol to indicate to Python that the next line is still part of the previous line of code, e.g.

```
if player1 == 'rock' and\
   player2 == 'scissors':
   print 'Player 1 wins.'
```

Use whichever form you feel comfortable using. When you are done coding *and testing!*, print a copy of the file and turn it in. Make sure your name and section number is in the comment section of your program.

Exercise 1.8 – For & While Loops

Create a new file called `loops.py` and use it for all parts of this exercise. Remember the difference between `input` and `raw_input`? If not, look at Exercise 1.5 again.

Be sure to test your code for each part before moving on to the next part.

1. Using a for loop, write a program that prints out the decimal equivalents of $1/2, 1/3, 1/4, \ldots, 1/10$.
2. Write a program using a while loop that asks the user for a number, and prints a countdown from that number to zero. What should your program do if the user inputs a negative number? As a programmer, you should always consider "edge conditions" like these when you program! (Another way to put it- always assume the users of your program will be trying to find a way to break it! If you don't include a condition that catches negative numbers, what will your program do?)
3. Write a program using a for loop that calculates exponentials. Your program should ask the user for a base `base` and an exponent `exp`, and calculate $\texttt{base}^{\texttt{exp}}$.
4. Write a program using a while loop that asks the user to enter a number that is divisible by 2. Give the user a witty message if they enter something that is not divisible by 2- *and make them enter a new number.* Don't let them stop until they enter an even number! Print a congratulatory message when they *finally* get it right.

When you are done, print a copy of the file and turn it in. Make sure your name and section number is in the comment section of your program.

Exercise OPT.1 – Zeller's Algorithm

OPTIONAL!- Some problem sets will have optional exercises at the end. Feel free to work on these problems if you have time at the end of the assignment, but you certainly don't have to do them. However, you will get excellent practice in Python, and we will give you feedback on any optional work you turn in.

Zeller's algorithm computes the day of the week on which a given date will fall (or fell). In this exercise, you will write a program to run Zeller's algorithm on a specific date. You will need to create a new file for this program, `zellers.py`. The program should use the algorithm outlined below to compute the day of the week on which the user's birthday fell in the year you were born and print the result to the screen.

Start with the program in Exercise 1.5, but ask for the month as a number between 1-12 where March is 1 and February is 12. If born in Jan or Feb, enter previous year (see the notes below). In the end, print out the name of the user and on what they of the week they were born.

Zeller's algorithm is defined as follows:

Let A, B, C, D denote integer variables that have the following values:

```
A = the month of the year, with March having the value 1, April the
value 2, . . ., December the value 10, and January and February being
counted as months 11 and 12 of the preceding year (in which
case,subtract 1 from C)
B = the day of the month (1, 2, 3, . . . , 30, 31)
C = the year of the century (e.g. C = 89 for the year 1989)
D = the century (e.g. D = 19 for the year 1989)
```

Note: if the month is January or February, then the preceding year is used for computation. This is because there was a period in history when March 1st, not January 1st, was the beginning of the year.

Let W, X, Y, Z, R also denote integer variables. Compute their values in the following order using integer arithmetic:

```
W = (13*A - 1) / 5
X = C / 4
Y = D / 4
Z = W + X + Y + B + C - 2*D
R = the remainder when Z is divided by 7
```

The value of R is the day of the week, where 0 represents Sunday, 1 is Monday, . . ., 6 is Saturday. If the computed value of R is a negative number, add 7 to get a non negative number between 0 and 6 (you don't need to do this in the code). Print out R. You can check to be sure your code is working by looking at `http://www.timeanddate.com/calendar/`.

Run some test cases- try today's date, your birth date, and whatever else interests you!

Feel free to submit your `zellers.py` code if you wish, we'll take a look at it if you do.

Exercise OPT.2 – Secret Messages

OPTIONAL! This exercise is tricky! Be sure to ask the LAs for help if you need them!

The goal of this exercise is to write a cyclic cipher to encrypt messages. This type of cipher was used by Julius Caesar to communicate with his generals. It is very simple to generate but it can actually be easily broken and does not provide the security one would hope for.

The key idea behind the Caesar cipher is to replace each letter by a letter some fixed number of positions down the alphabet. For example, if we want to create a cipher shifting by 3, you will get the following mapping:

```
Plain:   ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher:  DEFGHIJKLMNOPQRSTUVWXYZABC
```

To be able to generate the cipher above, we need to understand a little bit about how text is represented inside the computer. Each character has a numerical value and one of the standard encodings is ASCII (American Standard Code for Information Interchange). It is a mapping between the numerical value and the character graphic. For example, the ASCII value of 'A' is 65 and the ASCII value of 'a' is 97. To convert between the ASCII code and the character value in Python, you can use the following code:

```
letter = 'a'

# converts a letter to ascii code
ascii_code = ord(letter)

# converts ascii code to a letter
letter_res = chr(ascii_code)

print ascii_code, letter_res
```

Start small. Do not try to implement the entire program at once. Break the program into parts as follows:

1. Create a file called `cipher.py`. Start your program by asking the user for a phrase to encode and the shift value. Then begin the structure of your program by entering in this loop (we'll build on it more in a bit):

   ```
   encoded_phrase = ''

   for c in phrase:
      encoded_phrase = encoded_phrase + c
   ```

   What does this loop do? Make sure you understand what the code does *before* moving on!

2. Now modify the program above to replace all the alphabetic characters with '`x`'. For example:

   ```
   Enter sentence to encrypt: Mayday! Mayday!
   Enter shift value: 4
   The encoded phrase is:  Xxxxxx! Xxxxxx!
   ```

   We are going to apply the cipher only to the alphabetic characters and we will ignore the others.

3. Now modify your code, so that it produces the encoded string using the cyclic cipher with the shift value entered by the user. Let's see how one might do a cyclic shift. Let's say we have the sequence:

   ```
   012345
   ```

   If we use a shift value of 4 and just shift all the numbers, the result will be:

   ```
   456789
   ```

   We want the values of the numbers to remain between 0 and 5. To do this we will use the modulus operator. The expression x%y will return a number in the range 0 to y-1 inclusive, e.g. 4%6 = 4, 6%6 = 0, 7%6 =1. Thus the result of the operation will be:

   ```
   450123
   ```

   Hint: Note that the ASCII value of '`A`' is 65 and '`a`' is 97, not 0. So you will have to think how to use the modulus operator to achieve the desired result. Apply the cipher separately to the upper and lower case letters.

   Here is what you program should output:

   ```
   Enter sentence to encrypt: Mayday! Mayday!
   Enter shift value: 4
   The encoded phrase is:  Qechec! Qechec!
   ```

   When you are done, print a copy of the file and turn it in. Make sure your name and section number is in the comment section of your program.

6.189 A Gentle Introduction to Programming

January (IAP) 2011