# 6.S189 Homework 1

`http://web.mit.edu/6.189/www/materials.html`

## What to turn in

Do the warm-up problems for Days 1 & 2 on the online tutor. Complete the problems below on your computer and get a checkoff in lab or office hours for them. Problems 1.1 - 1.4 can be completed after Monday's lecture and will be required for Checkoff 1; problems 2.1 and 2.2 can be completed after Tuesday's lecture and will be required for Checkoff 2. Don't wait until the last minute to get your checkoffs!

## Exercise 1.1 – 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 2.7.X**. Ask a TA 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 1.2 – Hello, world!

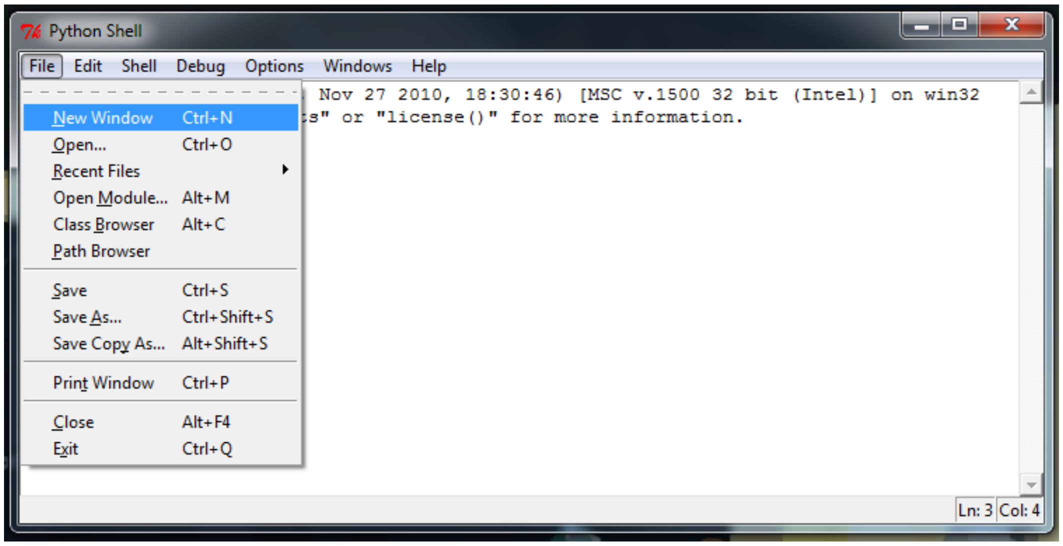
Recall that a program is just a set of instructions for the computer to execute. Let's start with a basic command:

`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.

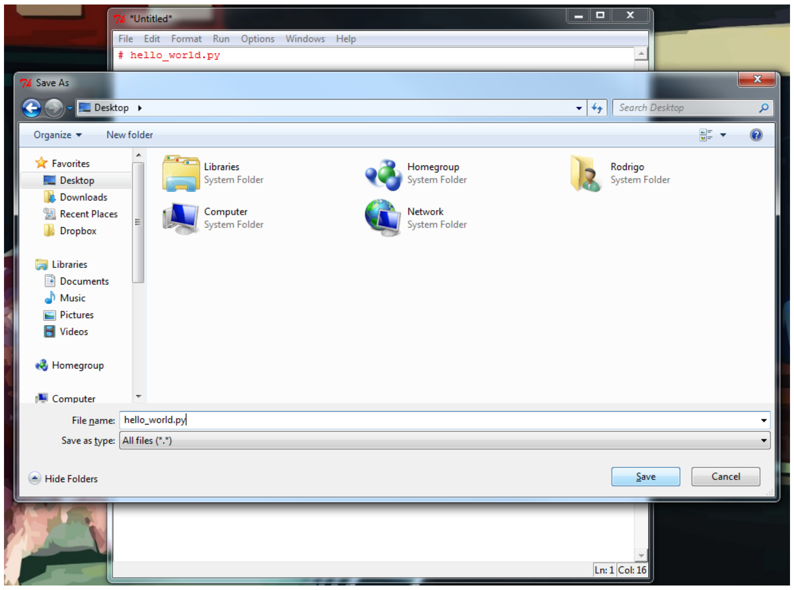
How to create a program file:

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



```
 Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on win32
s" or "license()" for more information.
```

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!



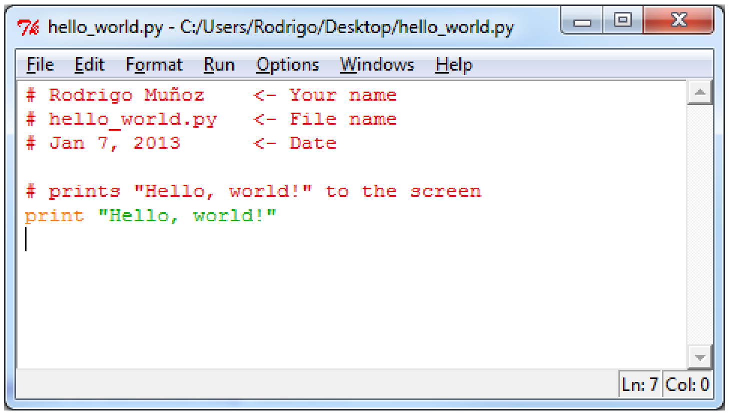
```
# hello_world.py
```

File name: hello_world.py
Save as type: All files (*.*)

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 only be one line!

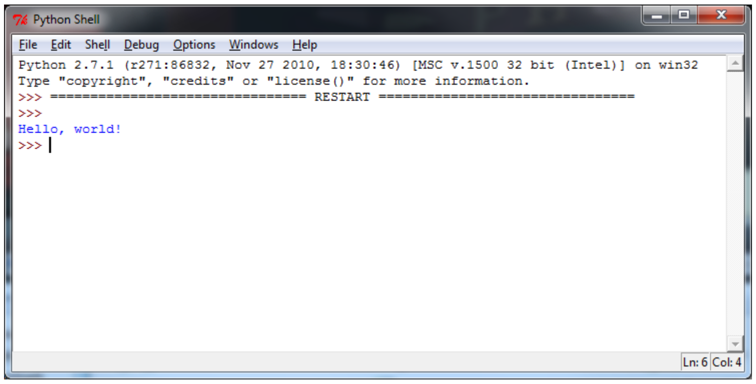
When you are done, save your work and run it. Your code should look similar to this:


hello_world.py - C:/Users/Rodrigo/Desktop/hello_world.py
File Edit Format Run Options Windows Help
Ln: 7 Col: 0

```
# Rodrigo Muñoz     <- Your name
# hello_world.py    <- File name
# Jan 7, 2013       <- Date

# prints "Hello, world!" to the screen
print "Hello, world!"
```

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:



```
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello, world!
>>> 
```

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

## Exercise 1.3 – Printing

From the course website, download the `homework_1.py` template. Remember to put your name and kerberos at the top. If you don't we'll be a bit 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 1.4 – 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 1.3, but which uses variables to cut down on the amount of typing you have to do. Hint - how many different variables should you need?

`--------------------------------------------------------------------------------`

When you are done coding *and testing!*, make sure your name is in the comment section of your programs, then find a staff member to check you off for exercises 1.1 - 1.4.

At this point, we suggest completing the tutor warm-up problems and Day 1 readings to cement your understanding of Monday’s topics. The remaining exercises cover Tuesday’s topics.

## Exercise 2.1 – 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, which we will be needing in later exercises.

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:

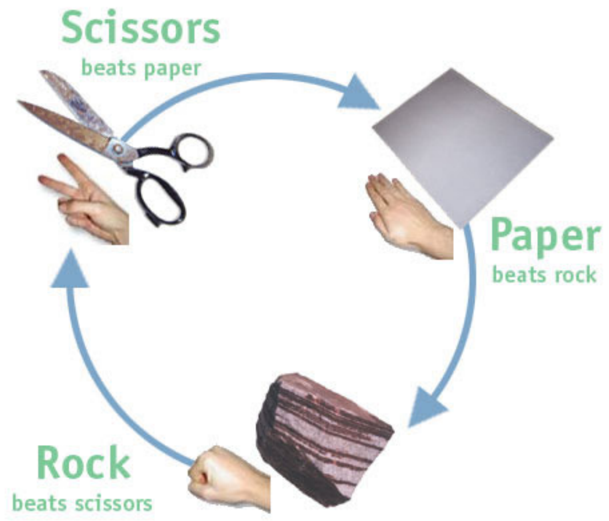
`print mo, day+',', year`.

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 later on, but you may want to play with string concatenation operations now to get everything to look its prettiest.

--------------------------------------------------------------------------------

## Exercise 2.2 – 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:



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, paper, scissors 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!*, make sure your name is in the comment section of your programs, then find a staff member to check you off for exercises 2.1 and 2.2.