6.149 Checkoff 2

`http://web.mit.edu/6.149/www/materials.html`

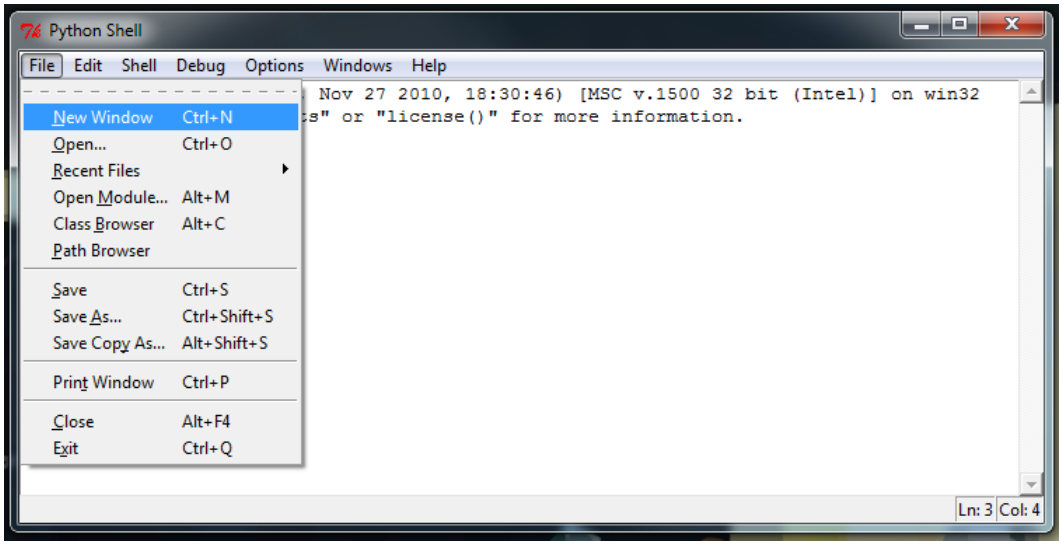
What to complete

Due: Wednesday, January 14, 2015 @ 5 p.m.

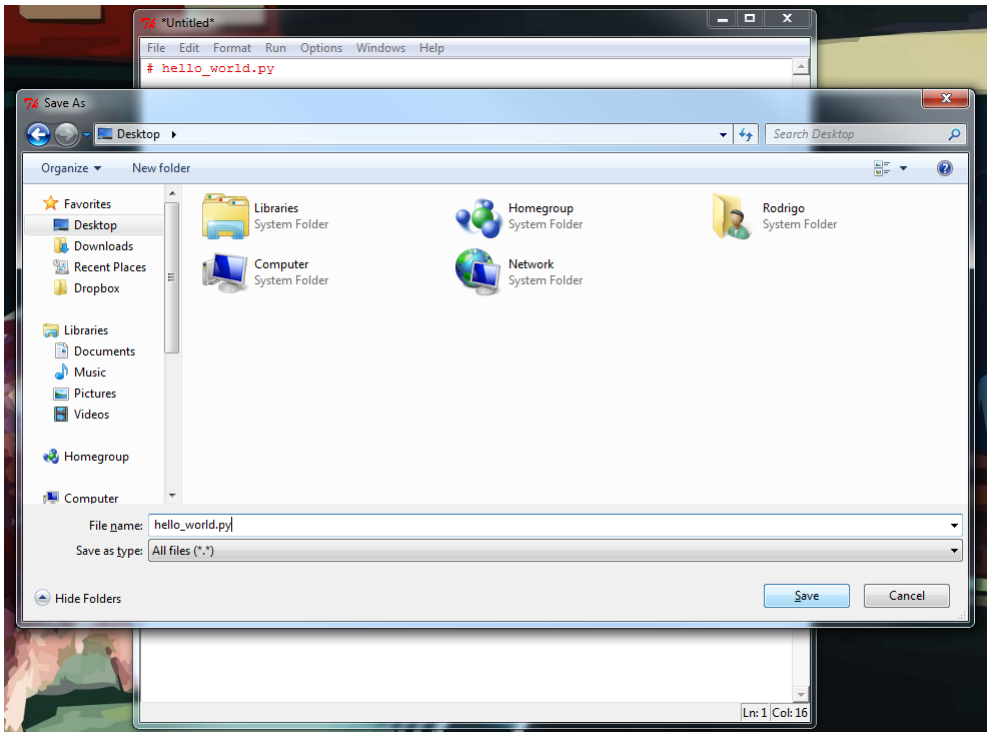
1. `checkoff2_user_input.py`, which will contain your code for 2.1 - User input
2. `checkoff2_rps.py`, which will contain your code for 2.2 - Rock, paper, scissors

Recall: Creating a file and running a program in IDLE

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



2. Save the file with a `.py` extension. Skipping the '.py' portion of the file name will prevent you from seeing syntax highlighting.



3. Start every program with a few lines of comments, listing your name, your kerberos username, and a brief description of what the file does. Recall that a comment line begins with a '`#`' (pound) symbol.

4. To run your program, chose **Run Module** from the **Run** menu. If you have not saved your work, you will be prompted to do so.

2.1 User input

First, create the file `checkoff2_user_input.py` to save your code for this problem. At the top of the file, write your name, kerberos username, and a short description of the program in a comment. The goal of this exercise is to ask the user for his/her first name, last name, and date of birth; after retrieving this information, print everything to the Shell as part of a sentence. Recall that you can gather input from the user with the function `raw_input('text')`, as taught in lecture.

Note: Python 2.7 has two built-in functions for getting user input. The first, `raw_input`, automatically typecasts the user's input to a string. The second, `input`, evaluates what the user types as Python code. If the user types a number, like 3.0, the input will be saved as the float 3.0. (`raw_input` would save the value as the string "3.0".) While `input`'s functionality may initially seem convenient, a user inputting an unexpected type could cause errors when you try to call a function with the value. Additionally, a user could type potentially malicious code, such as deleting files from your computer; for these reasons, `input` has been deprecated in Python 3. Please use `raw_input` for all work in this class.

Your output should look like this:

```
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 can pass multiple arguments (or inputs) to the print function by separating them by a comma. just need to separate the arguments with a comma. You can use this syntax for any two types of objects. Calling print will multiple arguments in this manner adds a space between the arguments. For example, the lines:

```
color_1 = 'red'
color_2 = 'blue'
color_3 = 'yellow'
print color_1, color_2, color_2
```

prints

```
October 20 1977
```

Note that we want to include a comma between the date and the year in the output of our program. To add text between variables, you can concatenate, or add, other strings to the variables you pass as input to the print function:

```
print color_1, color_2 + ' and', color_3
```

prints

```
red blue and yellow
```

Note that each term is evaluated before being passed as an input to the print function. The entire expression

```
color_2 + ' and'
```

is a single term. The `+` sign concatenates two strings (in this case, `color_2` and `' and'`), but cannot be used to combine an integer and a string, for example. Adding a number to a string isn't a well-defined operation, so Python throws an error whenever it encounters attempts to do so.

Note that while

```
color_2 + ' and'
```

has a space before the word 'and,' we don't type a space afterward. Remember that passing multiple terms to the print function automatically adds a space after each term. Python adds the space without any additional instruction from the coder.

2.2 Rock, Paper, Scissors

The goal of this exercise is to practice using conditional statements (if, elif, else). You will write a small program that will determine the winner of a rock-paper-scissors game, given two inputs that will represent the choices of two players. Your program will print this result.

1. First, create a "truth table" representing the outcome of the game for all possible combinations of inputs from the two players. The table will help you ensure that you aren't missing any possible branches of your code. Recall that rock beats scissors, paper beats rock, and scissors beats paper.

Player 1	Player 2	Winner
Rock	Rock	Tie
Rock	Scissors	Player 1

2. Create a new file `checkoff2_rps.py`. At the top of the file, write your name, kerberos username, and a short description of the program in a comment. The program's interaction should appear as in the following example:

```
Player 1's move? rock
Player 2's move? scissors
Player 1 wins.
```

The only valid inputs are rock, paper, and scissors. If the user types anything else, your program should print, 'You entered an invalid input. You can only play rock, paper, or scissors.' Use the truth table to make sure you have all necessary if statements.

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 (sky_color == 'red' and
   time_of_day == 'morning'):
   print 'Sailors take warning.'
```

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 sky_color == 'red' and\
   time_of_day == 'night':
   print 'Sailors' delight.'
```

Before writing your code, paste the following code into a file and run it. What do you expect the output to be? Is it different than what you expected? Feel free to ask questions on Piazza if you don't see what's happening here compared to the code above.

```
greater_than_two = 3
less_than_two = 1
if less_than_two and greater_than_two > 2:
    print "Both values are greater than 2."
```

3. Test your code! At the end of checkoff2_rps.py, write a comment detailing three distinct test cases containing the input for Player 1, input for Player 2, what you expected the outcome to be, and what the outcome was when you tested the code. For example, one test case may look something like:

```
# Test Case 1
# Player 1's move: rock
# Player 2's move: scissors
# Expected outcome: Player 1 wins.
# Actual outcome: Player 1 wins.
```

Of course, if your actual outcome does not match your expected outcome, you should go back through your code and see what went wrong!