Computational Assignment 1

Assigned Tuesday, 3-17-20., **Due Tuesday, 3-22-20.**

Congratulations on installing the Jupyter Notebook! Welcomne to your first computational assignment!

Beyond using this as a tool to understand physical chemistry, python and notebooks are actually used widely in scientific analysis. Big data analysis especially uses python notebooks.

Introduction to the notebook

If you double click on the text above, you will notice the look suddenly changes. Every section of the notebook, including the introductory text, is a technically a code entry. The text is written in a typesetting language called **Markdown**. To learn more see https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet)

To run a code entry in the notebook, select the section you want to run and type

```
shift+enter
```

If you want to make notes on a notebook, you can press the plus sign in the toolbar above to creat a new entry. Then make sure to switch the menu that in the toolbar from **code** to **Markdown**.

We can also run calculations this way.

In the entry below, I haved typed

```
123+3483
```

Select the entry and type `shift+enter`

```
In [1]: 123+3483
Out[1]: 3606
```

Once you run an entry, the output is displayed on the screen, when applicable

Now try some arithmatic yourself in the blank entry below.

(Don't forget to hit `shift+enter` to run your calculation!)

```
In [1]: 1+5+6
Out[1]: 12
```

Introduction to programming and python

Python is a very powerful and intuitive modern programming language. It is easier to learn than many other languages. Because of the wide availability of libraries such **numpy** and **scipy** (among many others), it is very useful for scientific calculations.

In this section, we will cover some very basic concepts. I assuming that nearly everyone has little or no previous programming experience, which is common for most chemistry and biology students.

We will slowly build up to the skills we need to run complex calculations!

Our first python code: "Hello World!"

The first thing we usually learn how to do is print a simple message to the output. Run the following entry.

```
In [ ]: print("Hello World!")
```

Print is function that takes in text as an argument and outputs that text.

A slightly more complicated example. Run the following entry

In [ ]:
```
# This is a comment in python

# Set a variable
x = 1 + 7

# print the result of the variable
print(x)
```

The lines that begin with "#" are comments. They are not read by the notebook and do not affect the code. They are very useful to make your code human readable

This snippet of code set assigned the result of `1+7` to the variable `x` and then used `print` to output that value.

Loops

One of the benifits of the computer is that it can run a calcualtion many times without having to manually type each line. The way that we do this is to use a **loop**.

In [12]:
```
# This is an example of a loop

# The colon is required on the first line
for i in (1,2,3):
    # This indentation is required for loops
    print ("Hello World, iteration",i)
```

```
Hello World, iteration 1
Hello World, iteration 2
Hello World, iteration 3
```

Explanation

1. The command `for` tells the code that this is a loop
2. The variable `i` is the counting variable. Everytime the loop runs, it will sequentially take on a different value from the list
3. The `(1,2,3)` is list of values.

Sometimes we need to run a loop many times or iterate over a large list of numbers. For this, the `range` command is useful

In [20]:
```
# The command range(a,b) creates a list of numbers from a to b
for i in range(-3,3):
    print ("Hello World, iteration",i)
```

```
Hello World, iteration -3
Hello World, iteration -2
Hello World, iteration -1
Hello World, iteration 0
Hello World, iteration 1
Hello World, iteration 2
```

Note that the `range(a,b)` command makes a list that spans from `a` to `b-1` . In the example above `range(-3,3`) makes a list that goes from -3 to 2

Conditional Statements: IF

Many times we want the computer to do something after analyzing a logical statement. **If this is true, then do that**. These are called conditional statements

In [28]:
```
# Conditional example

a = 100

if (a>0):
    #Like in the loop example, the indentation defines what happens in t
    # block of the if statement
    print("the number is positive")
elif (a<0):
    print("the number is negative")
elif (a==0):
    print("the number is zero")
```

```
the number is positive
```

Now we can try it again with a different value for `a`

In [31]:
```
# Conditional example again

a = -1234

if (a>0):
    print("the number is positive")
elif (a<0):
    print("the number is negative")
elif (a==0):
    print("the number is zero")
```

```
the number is negative
```

Once more time

In [32]:
```
# Conditional example again

a = 0

if (a>0):
    print("the number is positive")
elif (a<0):
    print("the number is negative")
elif (a==0):
    print("the number is zero")
```

```
the number is zero
```

Bringing it all together

These can all be combined together to perform complicated actions. Note the indentation and colons. They matter.

Combined Example

In [5]:
```
# A loop with an if statement

for i in range(-1,2):
    print("Iteration",i)
    if (i==0):
        print("zero!")
```

```
('Iteration', -1)
('Iteration', 0)
zero!
('Iteration', 1)
```

Exercise

Following the examples above, write a code snippet that uses the `range` command to scan from -10 to 10 and print whether the number is positive, negative, or zero.

To turn this in, upload the notebook in Github, this will make a committ that I can see.

In [22]:
```
for i in range (-10,11):
    print("number",i)
    if (i>0):
      print("positive")
    if (i<0):
        print("negative")
    if(i==0):
        print("zero")
```

```
('number', -10)
negative
('number', -9)
negative
('number', -8)
negative
('number', -7)
negative
('number', -6)
negative
('number', -5)
negative
('number', -4)
negative
('number', -3)
negative
('number', -2)
```

```
negative
('number', -1)
negative
('number', 0)
zero
('number', 1)
positive
('number', 2)
positive
('number', 3)
positive
('number', 4)
positive
('number', 5)
positive
('number', 6)
positive
('number', 7)
positive
('number', 8)
positive
('number', 9)
positive
('number', 10)
positive
```

```
In [ ]:
```