Python basics III: functions and commenting code

Functions

functions are one of the cornerstones of programming, basically they are small reusable pieces of code.

```
# function definition, this tells python
what will happen WHEN you call the function.
def function_name(arguments):
   function statements ....

#calling a function
function_name(arguments)
```

Functions examples

```
def times(x, y): # x and y are both arguments to the function times
    return x * y # the value that is returned when the function is
called

print(times(2, 4))
#output
#8

x = times(3, 4) # save the result from the function
print(x)
#output
#12

#functions can have no arguments
def two():
    return 2

print(two())
#output
#2
```

More examples

```
print(two()*two())
#output
#4

#functions can have optional arguments
def times(x, y=2): # y is 2 unless specified
    return x * y

print(times(2)) # equivalent to times(2,2)
#output
#4

print(times(2,3)) # overrides the default value of y=2
#output
#6
```

Self documenting code

the easiest thing to do increase readability is called self documenting code. Writing code that has variable and function names that are indicators of what they store and what they do

```
my_list = [1, 3, 5, 7]
# vs
odd_numbers = [1, 3, 5, 7]
```

```
my_dict = {
    "New York" : "New York City",
    "Nebraska" : "Lincoln",
}
# vs
capitals_by_state_name = {
    "New York" : "New York City",
    "Nebraska" : "Lincoln",
}
```

```
def a():
    print "hello world"

# vs
def print_hello_world():
    print "hello world"
```

Good commenting style

We will be following the google style guide for this class

https://github.com/google/styleguide/blob/gh-pages/pyguide.md#38-comments-and-docstrings

avoid describing code, it is not useful, see example below.

Good commenting style

```
#BAD
# returns True if large_num is greater than 5 <- BAD
if large_num > 5:
    return True

#GOOD
# calculate y position using linear line relationship from x position,
slope and offset
y = m*x + b
```

Doc strings

Documentation strings are built-in help information for a function, that can be accessed by `.__doc__` for each function

```
def add(x, y):
    """performs simple additions between two integers

    Args:
        x: the first integer to be added
        y: the second integer to be added

    Returns:
        returns the integer x + y
    """

    return x + y
```