Style Guidelines and Reference

Common Errors in Style

Commenting

While you are writing your code, it is good style to add comments to it. These comments make the goal of your functions clear, as well as how they achieve this goal. They are useful not only for others trying to read and understand your code, but also for yourself if you take a second look later on and do not remember your train of thought.

It is good style to include a comment in the line above each function you write, describing the purpose of the function and what its required parameters represent. In addition to this, it is good style to add a comment in the line above above or at the end of a line of code that might require clarification or justification of correctness.

Be sure that your comments include useful information, rather than a re-statement of the obvious! See the first example of poor style below for clarification on this.

Examples of poor style:

```
#Input a list and print some of its elements
def my_function1(my_list):
    #for each element in the List
    for e in my_list:
        if (e % 2 == 1):
            print(e)      #print e

def my_function2(my_list):
    if (my_list == []):
        return []
    else:
        return my_function2(my_list[1:]) + [my_list[0]]
```

Examples of corrected style:

```
#Prints each of the odd numbers in the list of integers, my_list
def my_function1(my_list):
    for e in my_list:
        if (e % 2 == 1):   #If e is odd
            print(e)

#Returns a copy of my_list, in reverse order
def my_function2(my_list):
    if (my_list == []): #Base case, nothing left to reverse
        return []
    else:  #Recursive case, reverse the tail of the list
           #  and then add the head to the back.
        return my_function2(my_list[1:]) + [my_list[0]]
```

Inconsistent Spacing

This is very important in Python!.

When indenting your code, it is important to make sure your spacing is consistent across the entire file. This in especially important in Python, because ambiguous indentation can change the meaning of a function or even cause a syntax error!

To avoid this, make sure you are indenting using only one of the following: One tab, four spaces, or two spaces. We recommend you use four spaces or a single tab.

Examples of poor style:

```
def my_function():
  for i in range(10):
      print(i)
    return None

def my_function():
    for i in range(10):
      print (i)
    return None
```

Examples of corrected style:

```
def my_function():
    for i in range(10):
        print (i)
    return None

def my_function():
  for i in range(10):
    print (i)
  return None
```

Poor Function/Variable Names

It is very important to name your functions and variables appropriately when writing code. It makes it very hard for someone else to understand what you have written if the names do not represent what variables or functions actually do. You may even confuse yourself throughout the process of writing your own code!

There are several ways to name functions and/or variables inappropriately, as shown in the example below:

1. Avoid any name that is a Python keyword. In the example, the function's parameter is called '`list`', which is a reserved keyword in Python representing the List type.
2. Avoid names that suggest anything besides what the variable actually represents. In the example, the loop variable is named '`index`', but it will actually store the *value* of each element in the List.
3. Avoid names that are irrelevant to the program, even if they are not necessarily misleading (as in (2)). In the example, the name '`roflcopter`' is used, but this does not provide any useful information about the value it will be storing.
4. Name functions according to their intended purpose. In the example, the function is named '`even_elements`', but it actually returns a List containing all of the *odd* elements in the original List!

Example of poor style:

```
def even_elements(list):
    roflcopter = []
    for index in list:
        if (index % 2 == 1):
            roflcopter.append(index)
    return roflcopter
```

Example of corrected style:

```
def odd_elements(integer_list):
    odds = []
    for num in integer_list:
        if (num % 2 == 1):
            odds.append(num)
    return odds
```

Magic Numbers

A 'Magic Number' refers to a hard-coded number that exists in the code you have written. It is often hard to tell what these numbers represent, so it is good style to create a new variable in order to give them a name (even if the value will not change throughout the program!).

Example of poor style:

```
def bits_to_KB(bits):
    return (bits / 8.0) / 1024.0
```

Example of corrected style:

```
def bits_to_KB(bits):
    bits_per_KB = 1024.0 * 8.0
    return bits / bits_per_KB
```

Long Lines (80+ Characters)

It is hard to read through a program when too many ideas are all smashed together onto one single line of code. For this reason, it is poor style to have lines of code exceeding 80 characters in length.

Example of poor style:

```
def print_information(L):
    print(L[0], " is the first element in this List.  The last element is ", L[-1])
```

Examples of corrected style:

```
def print_information(L):
    print("The first element in this List is: ", L[0])
    print("The last element in this List is: ", L[-1])

def print_information(L):
    print(L[0], end = " ")
    print("is the first element in this List.", end = "  ")
    print("The last element is", end = " ")
    print(L[-1])
```

Redundancy

There are several different forms of redundancy, all of which are poor style in coding.

First, we have conditions used in '`if/else`' statements which are already implied by the logical negation of the first condition, and are thus redundant. Although both functions are correct, one has better style than the other:

Poor style

```
def between(high, low, n):
    if (n >= low) and (n <= high):
        return True
    if (n < low) or (n > high):
        return False
```

Correct style

```
def between(high, low, n):
    if (n >= low) and (n <= high):
        return True
    else:
        return False
```

Second, we have lines of code that do not do any useful work towards the goal of a function. Omitting these redundant lines will have no effect on the overall calculation, so it is bad style to include them:

Poor style

```
def multiply(x, y):
    sum = x + y
    product = x * y
    return product
```

Correct style

```
def multiply(x, y):
    product = x * y
    return product
```

Lastly, we have unnecessary duplication of code, which can often be fixed with a helper function:

Poor style

```
#Some mathematical function
#(for the sake of example)
def do_math(x, y):
    term1 = ((x - 15) * 110)
    term2 = ((y - 15) * 110)
    return (term1*term2 - 15)*110
```

Correct style

```
#A helper function for do_math
def helper(x):
    return (x - 15) * 110

#Some mathematical function
#(for the sake of example)
def do_math(x, y):
    term1 = helper(x)
    term2 = helper(y)
    return helper(term1 * term2)
```