Python Style Guide
DS2000
Spring 2022

Creating understandable, well-written code is just as important as writing functional code. Any piece of code is read far more than it's written. Computer science is a collaborative field, and it's important to write code that others can read, understand, reuse, and even plug into their own programs. (Even your own code can be difficult to decipher once a few days have gone by!)

The Zen of Python

Above all, remember the foundation upon which Python is built:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Readability counts

If you're ever unsure about a style choice, try to keep your code as simple and understandable as possible -- and you'll be fine!

Code Structure

We'll write *all* our Python programs using the same structure, with the heart of our code inside a main function. Before you write any other code, type `def main():` at the very top and `main()` at the very bottom. Your program's code will go in between.

```
def main():
      # Your code goes here!
      # Your code goes here!

main()
```

Variable Names

Variable names must be short and descriptive. Use lowercase letters, and put in underscores as a way to separate words.

Here are some acceptable variable names:

```
age = 43
birth_year = 1978
first_name = "Laney"
```

And some crummy ones:

```
a = 43
x = 1978
```

```
fn = "Laney"
```

Constants, whose values never change once initialized, should be uppercase:

```
PI = 3.14159
```

Constants are the only permissible global variables. They may be defined at the very top of your program, below your comments but above all your functions. All other variables you use in DS2000 must be local -- i.e., defined within a function.

Strings

In Python, single-quotes and double-quotes are the same. You can use either one, but be consistent throughout an entire program.

Separate literals and variables using a comma. Don't use the % string-formatting operator.

Do this:

```
print("Hello", name)
```

Not this:

```
print("Hello %s" %name)
```

Space between expressions

Put an extra space between operators and variables.

Do this:

```
x = y + 5
```

Not this:

```
x=y + 5
```

Not this:

```
x = y+5
```

Do this:

```
if x == y:
```

Not this:

```
if x==y:
```

Comments

Every program you ever write needs to have comments. Before you write any code, put a block comment at the top of every program with your name, the course, the assignment, the date, and the name of the file.

```
''' Laney Strange
    DS2000
    Homework 1
    January 30, 2022
```

```
    catdog.py
'''
```

Comments explaining your code should appear throughout your program. You'll get the feel of what's too much or too little with experience. You don't need to comment every single time you introduce a new variable (that's too much!), but you also don't want to leave the reader to figure out all your code for themselves (that's too little!). Typically, you'll put a few comments above a block of code that does something interesting.

Comments go above Python statements, not beside them.

Do this:

```
# comment describing my code
python statement
```

Not this:

```
python statement # comment describing my code
```

Put a space between the crosshatch and the comment itself.

Do this:

```
# comment
```

Not this:

```
#comment
```

Code Width

Limit your code to 80 columns or less. In IDLE, the default module window will be 80 columns, so just don't let your lines wrap and you'll be fine.

If you need to continue a line onto the next one, indent the following line so the spacing makes sense:

```
spam = long_function_name(var_one, var_two,
                          var_three, var_four)
```

You can also use Python line continuation, by putting a blackslash (\) where you need the line to carry over:

```
a = 1 + 2 + 3 + 4 + \
    5 + 6 + 7
```

If you have a long string to store or print, use Python's linebreak syntax

```
print("In the loveliest town of all, where the houses were white "
      "and high and the elms trees were green and higher than "
      "the houses, where the front yards were wide and pleasant "
      "and the back yards were bushy and worth finding about.")
```

Vertical Space
Separate groups of logically-tied statements with vertical whitespace. Don't crowd too many lines together, which makes your code hard to follow.

```
list1 = [1, 2, 3, 4]
list2 = [4, 5, 6, 7]

list1 = list2
list1.append(18)
list2.append(17)

print(list1)
print(list2)
```

One Statement Per Line
Don't let multiple statements pile up on the same line.

Do this:

```
if x == y:
      print(“equal!”)
```

Not this:

```
if x == y: print(“equal!”)
```

Functions
When calling Python's built-in functions, leave space before and after the function call, but not within the parentheses:

```
print('Hello!')
num = random.randint(1, 10)
```

When calling any function, add a space after the comma separating your arguments, like this:

```
spam(18, 19, 'hello')
```

Your own functions should be generic -- write a function to add two numbers, rather than a function to add 3 + 4. Function names follow the same style as variable names, and the name of a function describes what it does. Try using verbs for function names.

Here's an example of a function that prints a given argument plus an extra line break (note that the function name is a verb):

```
def print_message(message):
    print(message)
    print("\n")
```

Functions should be concise; keep them under 30 lines of code.

Functions should also accept a limited number of parameters; five of them at the absolute max.

You'll also need to comment every function you write. Function comments should include the parameters and return type, and they should describe the **what** of a function as well. Use Python docstrings to comment your functions, like this:

```
def print_message(message):
    """
      Parameters: message to print, a string
      Returns: nothing
      Does: Print supplied string message plus an extra linebreak
    """
    print(message)
    print("\n")
```