UDACITY

Best Programming Practices

This document will provide some guidance on good programming practices.

- Summary
- Practices You Should Get in the Habit of Doing
    - Develop well-structured programs
    - Comment your code
    - Naming Conventions
    - Indentation
- Useful Links

Summary

Do you think of a programmer as somebody working isolated from the rest of humanity? Hiding behind the screen with little or no communication with other human beings? The programming world does not have to be a lonely and cold place, and Udacity wants to convince you of that. In fact, the programming world is a deeply collaborative one. By standing on the shoulders of giants, you'll find out that you don't always have to reinvent the wheel.

Programming is about collaboration and working together: it involves feedback and communication of ideas. Because a problem generally has many possible solutions, it requires as much reading someone else's code as writing your own. In fact, it is often harder to read code than to write it. And that is the reason why I'd like to introduce to you some good programming practices.

Practices You Should Get in the Habit of Doing

As you journey into a programming world, trying to develop a few simple habits from the start can prove to be a huge investment in your time and skills. Readability is the name of the game: the more readable your code is, the better it is.

Develop well-structured programs

It's naturally easier to read and understand code that has a visible structure associated with it By structure, I am referring to the logical organization of your code's layout, the intuitive, high level flow of your code. Of course, the structure of a program is going to depend entirely on what the program does, what computer language is being used, and who is coding it—there are many good ways to organize your code. Nonetheless, it is something you should always keep in the back on your mind when coding.

Deliberately thinking of your program in a structured way can be immensely beneficial as you develop your code. Try to get into the habit of coding one piece of the program at a time. You code one part, test it and make sure that piece is working before moving on, and continue this process for the rest of your program. Coding one big program in a single try will lead to more debugging time later.

Comment your code
Comments are human-readable annotations that you can include in your code, and the purpose of them is to make it easier for anybody to understand what your program does and why it is done that way. Not surprisingly, adding organized, neat comments to your code will allow others to understand your program better. It is the way all programmers document code and help each other understand themselves better, and you shouldn't be the exception. Remember that programs must be written for people to read, and only incidentally for machine to execute. Take pride in your comments and let people know exactly what you program does when you share it with others.

Try not to go overboard though! Adding too many comments can be distracting and can take away from your code's readability. As you practice and interact with more and more code, you will find a balance and become accustomed to the right amount of comments.

Check out the example below for commented code in Python - can you tell the comments apart from the code?:

```
class comment_example(object):
    '''
    Multiple lines of comments can be included by
enclosing them in
    triple quotes. They are usually used to document a
module, class
    or function.
    '''
    def greatest_counting_machine(self, n):
        """ This is a single line comment that describes 
what the
        method does."""
        i = 0
        total = 0
        while i < n:
            total += 2  # Single, in-line comments are 
denoted by #
            i = i + 1   # Use them wisely to explain what 
needs
                        # explanation.
        return total

count = comment_example()
print count.greatest_counting_machine(12) # This is a 
comment about
                                          # the print 
statement
```

As you can probably discern from the image above, comments can be added by using the number sign (#) or triple quotes ("""). The comments added with a number sign are used for inline annotations, and you will probably see them more often. The comments enclosed by triple quotes are known as docstrings, and are used only for class and method definitions. Don't worry if you don't know what exactly a class or a method is yet, at least now you know how to document them—bravo!

Try to always be as clear as possible with the comments you add. As a general rule of thumb, you should always follow the format that's already being used in whatever code you are working on. Also, sometimes the comments you need to include at the end of a line end up being too long to be easily readable. In that case, you can include right above the respective line of code, as shown below:

```
count = great_class()
print count.greatest_counting_machine(12) # This is a
short comment


# This is a long comment - more than 140 characters or so
- regarding
# the line of code below, so it's included on top of it.
print count.greatest_counting_machine(12)
```

Naming Conventions
Without going into too much detail of what variables or procedures are if you are not familiar, you should know that a programmer gets to name them. And even though I encourage you to come up with unique and simple names, there does exist several naming conventions on how to format them. Not following naming conventions won't result in a program error per se, but it will take away from the readability of your code and that's exactly what we are trying to avoid.

You should always try to be consistent when developing code. If you are going to use a specific naming convention, make sure you stick with it throughout your entire program. It's a good idea to look up what are the naming conventions associated to the programming language you plan to use. The usual naming convention for variables and functions in Python is lower_case_with_underscores.

Indentation
When programming, it is often natural to consider piece of code as being within or contained by another, in which case it can be easily denoted by indenting your code. Having organized, correct indentation in your code can greatly improve its readability.

In the Python programming language, indentation is especially important because incorrectly applying it will produce errors. Check out the examples below - even though you may not know exactly what it's happening, the indentation should give you at least an idea for what is contained within what:

```
class comment_example(object):
    def greatest_counting_machine(self, n):
        i = 0
        total = 0
        while i < n:
            total += 2
            i = i + 1
        return total
    if n <= 0:
        print "Please input a number greater than 0"

count = comment_example()
print count.greatest_counting_machine(12)
```

Can you see the difference with the code below where no indentation is applied? In Python, such code will produce an error and will not execute.

```
class comment_example(object):
def greatest_counting_machine(self, n):
i = 0; total = 0
while i < n:
total += 2
i = i + 1
return total
if n <= 0:
print "Please input a number greater than 0"

count = comment_example()
print count.greatest_counting_machine(12)
```

Useful Links

- PEP 8: the official document for the conventions of Python code.
- Google-styleguide
- Why coding style matters: better code quality.