Class Project 01

It is *very important* that you actually type the given code in by hand *exactly as it is written*. There are two reasons for this:

1. Attention to detail is a very important difference between people who are good at their job and people who are merely mediocre.
2. Sometimes the point of an exercise will be to fix something which has intentionally been written as broken.

Organization

These projects will be broken into **Exercises**, which will individually be broken into **Tasks** and those into **Subtasks**, if that level of structure is necessary. Code blocks and commands to be typed on the computer `will look like this`.

> Anything in a box like this is a side remark. Consider it like a footnote.
>
> > They can be nested.
> >
> > > I won't use this too often, but it's nice for quotations.

Exercise 1: Setup

These aren't really related to learning to code, they're just getting your "software stack" ready.

Task 1.1: Text Editor

Any text editor will do. There are a number of good cross-platform editors. Unfortunately, most of them are IDEs: *integrated development environments*. The most popular among programmers is Visual Studio Code; for those of us who don't like operating with Microsoft breathing down our necks, there is an alternative version of VS Code which avoids all the yucky Microsoft branding and telemetry: VSCodium. The upside is that these are beautiful text editors. The downside is that it's all too easy to add plugins which oversimplify the process of writing code. We will avoid the use of any plugins which complete code, debug our problems for us, or generally make things too easy.

Operating System	Options
Windows	VS Code, Notepad++
MacOS	VS Code, TextEdit
Linux	VS Code, gedit

> *Advanced*: On the off chance that you're using Linux via the command line, you probably don't need any help with this. I suggest using `screen` and your favorite choice of `vim`, `emacs`, or `nano`.

Henceforth this program will be called your *editor*.

Task 1.2: Find the *Terminal* on your computer

- Windows: the Terminal in Windows is called *PowerShell*
- MacOS: It's called *Terminal*, and it's in *Applications/Utilities* or something similar.
- Linux: If you are using Linux and can't find the terminal, you probably ought not be using Linux. There are many different terminal programs, and every Linux distribution comes with at least one — it's the basic multi-user login console.

Whichever one of these options you're using, I will refer to all of them as your *terminal* and the interface that it provides you access to I will call the *command line*. Commands that will be typed at the command line will look like this:

```
$ touch myface
```

The `$` is called the *prompt*. It might look different, often including the path to the current working directory (or at least the name of the current working directory).

Subtask 1.2.1

Test to see if `python3` works in your terminal.

```
$ python3
```

The response, when Python is correctly installed, should look something like this:

```
Python 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

If that shows up, type `quit()` and press Enter/Return/↵.

Task 1.3: Get Python

To get Python for the system you are using, head to https://python.org and download *Python 3*, in the latest version number; as of this writing, it is Python 3.11. For most purposes, the version number after 3 won't matter.

Task 1.4: Working Environment

You're going to want to know how to navigate in the terminal. You should look up on the internet (using DuckDuckGo, Brave Search, or some other privacy-focused search engine) the commands used in your particular operating system for navigating the command line. At the minimum, you'll need to know how to create a new directory (also called a folder) and how to change directories.

Wherever you keep your files (usually a directory called `Documents` or something) create a new directory called `aam-python`. Inside that you're going to create another directory for every daily activity.

Problem 2: Your First Program — Strings and Comments

Task 2.1: Make a new file

Create a directory in your `aam-python` directory called `project-01`. Open your editor and ***type*** the following code:

```
# Name: Frodo Baggins
print("Hello world!")
print("Hello yourself")
print('Type this exactly the way that I have typed it')
print("Don't change anything, even if you feel 'smart.'")
print("""Sometimes things don't do what you would expect, and\n
\tthat's okay""")
```

Change the line which includes `Frodo Baggins` so that it says your name instead.

Save that as a file named `prj01_02.py` in your `aam-python/project-01` directory.

Task 2.2: Run your new file as a Python script

Open your terminal and change directories to your `aam-python/project-01` directory. At the command line, type

```
$ python3 prj01_02.py
```

Expected Output (What Should I See?)

After running this code through the Python interpreter (that's what happens when you call `python3` with a filename argument), you should see the following output:

```
Hello world!
Hello yourself
Type this exactly the way that I have typed it
Don't change anything, even if you feel 'smart.'
Sometimes things don't do what you would expect, and

    that's okay
```

Task 2.3: Progress Check!

Answer the following questions:

1. When the *octothorpe* character `#` occurs in a line, it signifies to Python that everything else on the line is a comment and should be ignored. What happens if something is written to the left of `#` on a line?

> Some people have different names for `#`, including *hash*, *pound*, *mesh*, or *number sign*. Relax. It's not important.

2. A sequence of typable characters enclosed in double quotes `"like this"` is a `str`, which is short for *string*. Are there any other ways to enclose a `str`?
3. Even though they take two keystrokes to type, both `\n` and `\t` act as single character. The *backslash* `\` is called the *escape character* and indicates that the character following it should be treated in a special way.
    - How is `"\n"` treated?
    - How is `"\t"` treated?
    - How should `"\\"` be treated?
4. What does the `print(...)` function do?
5. What will happen if you add `print("Lots of people have phone #s")` as the last line of the file?
6. What is wrong with `print('The bird's the word!')`?

Common Problems

1. I got a `SyntaxError: invalid syntax` when I tried `python3 prj01_02.py`.
    - Yes, that's possible. A `SyntaxError` occurs when you type something that doesn't match the rules of Python. Look at what you typed and look at what was specified. Do they match? A good way to check is word-by-word **in reverse order**.
2. I get `can't open file 'prj01_02.py': [Errno 2] No such file or directory`.
    - You need to make sure the current working directory of the terminal is the directory in which you saved your `prj01_02.py` file.
3. Nothing happens when I run `python3 prj01_02.py`.
    - You need to use the `print` function or you won't see output.

Problem 3: Arithmetic

Here are some symbols. Let's give them names, and then during the exercise, you can identify the mathematical operation which goes with each.

Symbol	Name	Symbol	Name
`+`	plus	`<`	less than
`-`	minus	`>`	greater than
`*`	asterisk	`<=`	less than or equal to
`/`	slash	`>=`	greater than or equal to
`%`	percent		

It's good practice to call the symbol by its name when you're programming, even when what you're programming is math, to get used to the symbols being just that — symbols.

> The name of the symbol `*` is *asterisk*, not *Asterix*. Asterix is a cartoon Gaul who has a popular series of comic books in Europe, along with his friend Obelix and a cast of ragtag companions.

Task 3.1 Make another new file

Open a new file in your editor and type the following:

```
# Name: Frodo Baggins
print("Let's count some things.")

print("Bobs", 23 + 40 / 5)
print("Larrys", 3 + 10 * 7 % 6)

print("Here is a strange one:", 4 - 5 + 2 - 6 % 3 + 8 / 10 - 1)

print("Is 5 + 3 < 2 - 5?", 5 + 3 < 2 - 5)
print("Let's check.")
print("5 + 3 =", 5 + 3)
print("2 - 5 =", 2 - 5)

print("Is it greater?",  8 > -3)
print("Is it greater or equal?", 8 >= -3)
print("Is it less or equal?", 8 <= -3)
```

Replace `Frodo Baggins` with your name. Save it as `prj01_03.py` and then in your terminal run the command

```
$ python3 prj01_03.py
```

Task 3.2 Progress Check

Answer the following questions:

1. What arithmetic operators are performed by each of the symbols we described in the table?
2. Explain the order of operations utilized by Python.
3. Is `2 / 3 / 4` the same as `(2 / 3) / 4` or `2 / (3 / 4)` according to Python? Predict it then test your prediction.
4. You can work with Python in *interactive mode* by running

   ```
   $ python3
   ```

   Use interactive mode to discover what the operators `//` and `**` do when applied to integers.

Common Problem: Something weird happened!

This usually comes from miscopying the code. Here are some "tricks" which help you focus on details and notice differences:

1. Write a comment above each line of code explaining to yourself in plain English what the code does.

2. Read your `.py` file... backward. Compare it to the given code.
3. Read your `.py` file out loud, even the symbolic characters, using their names instead of the operator names.

Problem 4: Variables and Names

Programmers like to give memorable names to the values used in computation; in mathematics, there are generally only a few variables floating around in a given problem, but a long enough computer program could utilize hundreds of different variables. In mathematics, a variable is a letter used to represent an unknown quantity; in programming, a *variable* is a name bound to a particular value.

> Some languages allow or require variables to be declared with a particular *type* before the value is assigned. Also, some languages are *strictly typed*, meaning that once a variable has been bound to a particular type of value, it cannot be changed to a different type. My **new favorite language**, Rust, goes even further: the value of a variable cannot even be changed unless the type is specified as *mutable*.

The standard *naming convention* in Python is to use all lower case letters in variable names, and when multiple words are convenient to aid in remembering the purpose of the variable, they should be separated using the underscore character `_`.

Task 4.1: Make another new file!

Open a new file in your editor and type the following:

```
planes = 100
seats_per_plane = 12.0
pilots = 30
passengers = 110
planes_not_flown = planes - pilots
planes_flown = pilots

airpool_capacity = planes_flown * seats_per_plane
average_passengers_per_plane = passengers / planes_flown

print("There are", planes, " small planes in the fleet.")
print("We have", pilots, "pilots today.")
print("There will be", planes_not_flown, "grounded planes today.")
print("We can transport", airpool_capacity - pilots, "passengers today.")
print("We have", passengers, "today.")
print("We will need to put about", average_passengers_per_plane,
"passengers on each plane.")
```

Save that as `prj01_04.py` and run

```
$ python3 prj01_04.py
```

Warning!

Why might I see the following error?