Lab 2: Python Modes and Programming

In Wed lecture (the one where we posted a video due to the weather), we learned about three ways to run Python: interactive mode, script mode, and notebook "mode" (people outside of 301 won't use that vocabulary for notebooks, hence the quotes). In this lecture, you'll practice those three modes. You'll also get practice with operators, modular arithmetic, and Boolean logic.

To get started, please create a `lab2` directory inside your previously created course directory (named `cs301` or similar). Then, open a terminal and use `cd` to navigate to `lab2` (you may need to review the steps in the first lab document to remember how to find the pathname of the `lab2` directory).

The Three Modes

Interactive Mode

Run `python --version` in the terminal. You might see something like this:

```
Python 3.7.2
```

As long as it some form of 3.X.X, you're fine. If not, try running `python3 --version` instead. If you need to run the latter, please use `python3` whenever we say `python` in the directions this semester.

From the `lab2` directory, do the following. Run `pwd` in the terminal (this is a shell command in both bash and PowerShell). Then type `python` and hit ENTER.

You should see something roughly like this (details will vary):

```
Python 3.7.2 (v3.7.2:9a3ffc0492, Dec 24 2018, 02:44:43)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Those `>>>` symbols are a Python prompt. This means you can type Python code, but your shell commands will not work until you exit Python again. To see this, try typing this Python code: `print("hi")`. Then hitting ENTER. The message `hi` should be printed.

Now try typing `pwd` again and hitting ENTER. This should give you the following error, because `pwd` is only valid in the shell (not Python):

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  NameError: name 'pwd' is not defined
```

Now let's try exiting Python and going back to the shell prompt. On Mac, you can do this with `CONTROL-D` (hold down the control key, then press the D key at the same time). On Windows, you can use `CONTROL-Z ENTER` (hold down the control key, then press the Z key at the same time; release both, then press the enter key).

Now that you've exited, try running both `pwd` and `print("hi")` again. This time, the former should work and that latter should fail (because we're in the shell, and the former is a shell command whereas the latter is Python code).

Re-enter interactive mode and try running a few Python expressions:

- `10 / 4`
- `10 // 4`
- `10 % 4`
- `not False`
- `not not True`
- `not not not True`
- `2 ** 1000000` (warning, this is a REALLY big number)
- `1/0` (should fail)
- `"ha" * 100`
- `"ha\n" * 100`
- `print("ha\n" * 100)`
- `print("ha\n\n" * 100)`

Script Mode

From shell mode (make sure you've exited Python from before!), type `idle laugh.py`. This will normally open up a file named `laugh.py` in the idle editor, but in this case it will create a new empty file named `laugh.py` since one doesn't already exist. Paste the following into the editor:

```
print("ha " * 10 + "!")
```

From the run menu, click `Run Module` (saving your file if necessary); or, you can accomplish the same by hitting `F5` on your keyboard.

Now close the shell window (where your program just ran and printed "ha ha ..."), and remove the print in your file, so it looks like this:

```
"ha " * 10 + "!"
```

Run run your code again (e.g., with `F5`). Notice how it doesn't show any of your output? In interactive mode, prints usually aren't necessary, but they are in script mode.

Add back the print, then close any idle windows that are open. Now that you're back in the shell, run the following:

`python laugh.py`

Let's try to create a second program:

1. run `idle circle.py`
2. paste `print((8 / 2) ** 2 * 3.14)`
3. run the program (either in idle with `F5`, or by exiting idle and running `python circle.py` in the shell)

The program computes the area of a circle. *Can you figure out what the radius of that circle is by reading the code?*

Notebook Mode

You have already seen how to create a new notebook and turn it in. Now, you'll get practice downloading and running existing notebooks. For this part, download the `bool.ipynb`, `ops.ipynb`, and `mod.ipynb` files to your `lab2` folder. IMPORTANT: use the GitHub download procedure we've shown you involving right-clicking on the `Raw` button.

In the terminal (which should still have `lab2` as the current working directory), run `jupyter notebook`. You should see something like the following:

You can now click on any of the three notebooks you've downloaded to view the contents. The first thing you'll want to do after opening it is click `Kernel` > `Restart & Run All` so you can see the outputs. The exercises you should do with each notebook are described below.

Boolean Logic

Open and run the `bool.ipynb` notebook. There will be six cells, with outputs `True`, `False`, `False`, `True`, `False`, and `True`. Take moment to review each cell and think about why the output is either True or False. Review the lecture slides or ask your TA if any of these are surprising to you.

There are different ways to get `True` or `False` in Python. One way is to directly type those words explicitly, as we already have.

The more common way is to use comparison operators to make claims that are either True or False. For example `1+1 == 2` is True, whereas `-2 > 1` is False.

Your job is to edit the notebook and replace each True or False with a Python expression (of your own choosing) that has the same truth value. For example, the last expression says `True or False`, so I might edit that to instead be `1+1 == 2 or -2 > 1`. You should get creative, but try to use each of the following comparison examples at least once: `==`, `!=`, `<`, `<=`, `>`, and `>=`.

After each change, rerun the cell and make sure that your modification didn't change the result. For example, that last cell was originally `True`, and I'll verify it is still so after I change the code to `1+1 == 2 or -2 > 1` and rerun it.

Order of Operations

Open and run the `ops.ipynb` notebook. There will be six cells, each containing a simple Python expression involving two operators. Python has operator-precedence rules to determine which of the two operators to execute first.

Of course, the Python programmer (you!) can add parentheses to change the order in which the operators are executed, thereby changing the result. Your job is to add parentheses in each case to get desired result. For example, in the first cell, `3 ** 4 - 1` evaluates to 80, but you should change it to be `3 ** (4 - 1)`, thereby getting 27.

Modular Arithmetic

What time does a clock show one hour after twelve o'clock? Not 13 o'clock (that doesn't exist) -- instead, it wraps back around to 1 o'clock. This is a weird kind of arithmetic, where adding doesn't always make a number larger.

This alternative arithmetic is called **modular arithmetic**, and we can use the modulo operator (`%`) in Python to perform modular addition. However, there's a twist with respect to the clock: in CS, we count from 0, so if we were to have a **CS clock**, it would go from 0 o'clock to 11 o'clock (instead of from 1 o'clock to 12 o'clock).

Open and run the `mod.ipynb` notebook. You'll see many cells of the form `X % 12`. This computation is answering the question: *if we start at 0 o'clock and wait* `X` *hours, what time is it?* The `% 12` part means time wraps around at 12 o'clock, meaning that there is no 12 o'clock, just 0 o'clock again (remember we have a CS clock that goes from 0 to 11 o'clock).

Try writing some expressions to answer the following questions regarding a CS clock:

- what time will it be 6 hours after 9 o'clock? (run `(9 + 6) % 12` in a cell to find the answer, which is 3 o'clock)
- what time was 2 hours before 1 o'clock? (run `(1 - 2) % 12` in a cell to find the answer, which is 11 o'clock)
- what time was 4 hours before 2 o'clock?
- what will it be 12 hours after 6 o'clock?
- what will it be 13 hours after 6 o'clock?
- what will it be 24 hours after 6 o'clock?
- what will it be 25 hours after 6 o'clock?