Lab 0 - Introduction to Python

If you do not have a strong preference for languages to use in this assignment, we suggest using Python 3 for the Oracle assignment. Python comes pre-installed on all department machines and on Macs. If you would like to install Python in your own computer, grab version 3.X from the Python website.
If you've never used Python before, we recommend reviewing Prof. Philip Klein's Python lab from *Coding the Matrix*. Python's official online documentation also provides an introduction.

Making Python Executable (for Linux and Mac users)

For Oracle, we ask for two executable programs: `topsort.py` and `oracle.py`. To make a file executable, we must provide an interpreter directive (colloquially known as the "shebang" or "hashbang" on Unix machines) and set executable permissions. If you don't have the files `topsort.py` and `oracle.py`, create them and add the following lines to the top of each:

```
#!/usr/bin/python3
print("Hello World!")
```

The `#!` line serves as the interpreter directive, and the Python interpreter will run the file's code when executed. To set executable permissions, run `chmod +x topsort` and `chmod +x oracle`. To learn the details of Unix permissions, we recommend reviewing the relevant parts of this Wikipedia article. To test your programs, run `./topsort.py` and `./oracle.py`. If everything worked, you'll receive a "Hello World!" from each.

Command Line Arguments

Python's `sys` module provides access to command line arguments. To import the module, add `import sys` to your program. The expression `sys.argv` produces a list of strings corresponding to command line arguments. Note that the program's name comprises the first element of this list, so `sys.argv[1]` returns the first user-specified command line argument.

Python Type Hints

Python's typing module provides a way for us to let us remember what the type signatures of our functions and variables are. Some IDEs also use these type hints to give you warnings if you pass in the wrong types to function arguments or return with the wrong type.

You can use these type hints by adding them to function declarations as follows:

```
def my_function(n: int, name: str) -> str:
    ...
```

You can also define more complex types such as lists of numbers, dictionaries, and tuples by importing and using the type constructors provided in the typing module.

```
from typing import List, Tuple
def take_first_two(in_list: List[int]) -> Tuple[int, int]:
    ...
```

The typing module docs also have more information about this, including using type variables and other things, although you won't need those for the Oracle assignment. Adding type hints can be really helpful to keep your function's inputs and outputs straight. Try writing your own function using type hints!

Testing

It's always good to test your programs the best you can! In Python, a good way of doing this is to use an `assert` statement. This works by taking in a boolean expression afterwards, and throwing an error if the boolean expression evaluates to `False`. So, as an example, we could test the function `add1` below using the `assert` statements following.

```
def add1(n: int):
    return n + 1

if __name__ == "__main__":
    assert add1(1) == 2  # test passes
    assert add1(-1) == 0  # test passes
    assert add1(3) == 5  # test fails, throws an error
```

Note that it is good style to wrap your tests (and other top-level code) in an `if __name__ == "__main__"` block so that it only runs if the file is run directly but not if it is imported (such as when we autograde your Oracle assignment).

Try writing a simple function and writing a few tests that pass and a few that fail. See what happens if you import a file with a failing test if you don't have the `if __name__ == "__main__"` block.