Numerical Methods

Math 3338

Worksheet 0

Installing Python3 and First Program

You are expected to complete this before the first day of class. It's highly recommended to bring your laptop to class, although there will be limited computers available.

1 Directory Structure

Before we do anything, lets setup a nice directory structure. Go to your documents folder and create a folder called "Numerical_Methods" (use the underscore, spaces will prove to be annoying in the future). Inside this folder, make another folder called "Worksheets". Believe it or not, this is where we'll store Worksheets.

Each Worksheet will end up being it's own "Project", which will isolate it from other worksheets. Python always looks within it's own folder for dependencies and having too many objects can cause confusion.

2 Installing Python3

Visit the following link `https://www.anaconda.com/distribution/` and download and install the Python 3.10[1] version.

You now have Python 3.10 installed. Congratulations.

3 Using Python3

One of the programs you installed is called Spyder. Find this program and open it. This will be our primary IDE (Integrated Development Environment), so we should get used to it. In the toolbar (at the top) find the Projects menu and select "New Project...". We want a New Directory, set the project name to wk_00 (or something to signify it's Worksheet 00). Change the location to the Worksheet folder in the Numerical_Methods folder. You'll follow this process for every worksheet (you'll change the name though). Spyder will open a default file "temp.py", **DO NOT USE THIS**, open a new file.

Time to start coding. You should see file with some writing in it. This is a script file, and it's how we'll be doing homework. Save this file and give it a good name (I called mine "first.py", that's not a great name). Notice, by default it saves it in the Project folder. This is good.

Click the bottom of the blank file and type,

```
print("Hello World!")
```

Click the big green arrow at the top of the screen (or press F5 on your keyboard, if you're using a laptop you may need to press Fn+F5). Somewhere in Spyder you should see the "iPython Console" and in that console you should see the phrase "Hello World!". This is the stereotypical first program. Even though it's not really a program.

If this did not work (it gives you an error), you must stop by my office before class. We're not going to have time to get stuff working in class.

We will be using the iPython console quite a bit. Your IDE needs to be able to iPython, this is why we will be using Spyder.

[1] or, you know, whatever the newest version is. 3.11?

4 A Review of Python

CP	2.2.1, 2.4
NMEP	1.2

Table 1: Sections Covered

On every worksheet, I'll include a table like this. "CP" stands for "Computational Physics". This is the required text for the course. "NMEP" stands for "Numerical Methods in Engineering with Python3" this is the optional book for the course. It is not required to purchase but you should *carefully search for the lowest price*.

The remainder of this worksheet is a review of Python. You should go through this even if you are proficient in Python. You may learn a couple of things. If you are nervous with Python, this will help you be at the required level.

4.1 variables

To store a variable use =. Try the following,

```
banana = 6
print(banana)
```

Now the variable *banana* is 6. You should have seen this printed. Python will always guess the *type* of a variable. In this case, banana is an integer. There are many, many types.

```
banana = banana + 7 #banana is now 13
banana += 4 #banana is now 17. This is easier than the previous
banana *= 2 #banana is 34
banana *= 1.0 #banana is now 34.0, indicating a floating point number
```

Try the previous commands. Feel free to include some print statements to see the value of banana at each step. # signifies the beginning of a comment.

4.2 strings

Strings are basically words. To define a single line string use either " or ', a multiline string is three quotes. For example,

```
string1 = "I'm a string!"
string2 = 'Me too'
string3 = """I'm a multiline string
see
I'm
on
another LEVEL"""
```

Try printing these as well. Strings can be joined using the "+" operation.

```
print(string1+ " middle " + string2)
```

The type of a string is *str*. Whenever you encounter something new in Python, it's helpful to know what you can do with it. In the Console window (where you saw the output), try typing

```
help(str)
```

This will list everything you can do with a string with a description of what it does. At the moment this is information overload. Let's take a single example, the function *upper*. The description says it will make a copy of S in uppercase. Let's try,

```
upper_string = string1.upper()
print(upper_string)
```

Notice, since *upper* belongs to the string type, it goes after the variable as ".function()". In programmer terms, upper is a method is the string class.

4.3 tuples

A *tuple* is a static sequences of things. In programmer terms it is an *unmutable* type, it can't be mutated.

```
t = (1,2,"a","hello there")
```

This defines a tuple. Tuples can contain any type of object, although it can be bad practice to put types that are mutable in a tuple (like a *list* below).

You can access elements of a tuple,

```
print(t[0],t[1])
print(t[3])
```

Notices, tuples are indexed starting at 0.

Type "help(tuple)" in the interpreter (without quotes). Notice you can do far less with tuples than you could with strings. Notice there are a bunch of methods that start and end with two _, these are operator methods. This defines things like an ordering (<) and other helpful things.

In particular, lets explore

```
__len__(self,/)
```

This tells us we can use the "len" function on tuples. Try the following,

```
print(len(t))
```

If you guessed this will print the length of t, you are correct.

4.4 lists

A *list* is similar to a tuple, except it's mutable. This makes it really versatile and useful.

```
L = [1,2,"a","hello there"]
```

This defines a list. Try to determine what the following will do,

```
L[0] = 10
L[2] += "b"
L[-1] += ". General Kenobi"
```

If you print L, you'll see the results. Try the same with the tuple t and you'll get an error.

Type "help(list)" in the interpreter, and you'll see a wide variety of things you can do to a list. Throughout the semester we'll use most of these. Let reverse L.

```
L.reverse()
print(L)
```

Notice this function *changed L in place*. This has advantages and disadvantages.

You can also put lists inside of lists. Or lists, inside lists, inside lists.

```
L = [[1,2,3],[4,5,6],[7,8,9]]
```

You should think about what $L[0]$ is and the difference between $L[0]$ and $L[0][0]$.

4.5 dictionaries

A dictionary is a hash table. This probably doesn't make sense. A list is indexed by integers, if you want the 10^{th} element of a list, the computer has to walk through the previous 9 entries to find the 10^{th}. This can be slow. A dictionary just knows where each element is located (it uses an object hash).

Dictionaries have the form {key:value}, where a key is any *hashable* type, which must be unmutable. In particular a list cannot be a key, but strings and tuples can. The values can be any type.

```
d = {1:"value","a":10,(1,2):[7,8]}
```

This defines a dictionary. To call an element from a dictionary, use it's key.

```
print(d[1])
print(d["a"])
print(d[1,2])
```

The last one was special. You should figure out why. To add a new key,

```
d[10] = "new thing"
print(d)
```

Type "help(dict)" to see the dictionary methods.

As you first start using Python, Lists seem like the best data structure. As you get more experienced, you realize dictionaries are the best.