Numerical Methods

Math 3338 – Spring 2022

Worksheet 2

Loops, Functions, Reading Files

1 Reading

CP	2.2.4, 2.2.5, 2.3, 2.5
NMEP	1.2, 1.3, 1.4

Table 1: Sections Covered

2 Conditionals

This is just "if" statements. Here are some examples. Try to figure out what will print before you run the code.

```
if 3<4:
    print("Yes")
else:
    print("No")
```

Python uses white space to designate environments. The standard is 4 spaces. Typically, you just press tab and the computer figures it out.

```
if 3==4:
    print("Yes")
elif 3!=3:
    print("No")
else:
    print("huh")
```

Here is non-equality. != is $\neq$.

Checking containment in a list.

```
L = [1,2,3]
if 1 in L:
    print("Hooray")

if 5 in L:
    print("Double Hooray")
elif 4 in L:
    print("Fore")
else:
    print("Dang")
```

3 Loops

3.1 while

These are formatted as follows:

```
while condition:
    do stuff
```

Here is a basic example of building a list,

```
L = []
i = 0
while i<10:
    L.append(i**2)
    i+=1
```

What did this do?

You have to be careful not to accidentally enter an infinite loop. If you do, click the interpreter windows and press Ctrl+C. This should kill the execution.

```
while True:
    pass
```

3.2 for

A for loop runs through an iterator.

```
for variable in iterator:
    do stuff
```

What is an iterator? The easiest version is a list. You can also create custom iterators, which we may do.

Here is the same example as above.

```
L = []
for i in range(10):
    L.append(i**2)
```

The range function returns a *generator*, which is a special iterable. Try this,

```
print(range(10))
```

Figure that out.

There is less danger of falling into an infinite loop here.

```
L = [1,4,-1,2]
for i in L:
    print(i)
```

Showing you can iterate over a list.

You can also *unpack* while iterating,

```
L = [(1,2),(3,4),(5,6)]
for i,j in L:
    print(i*j)
```

3.3 list comprehension

If all we wanted to do was generate the list of squares, there is an easier way,

```
L = [i**2 for i in range(10)]
```

This is a short-hand and it's really useful. You can put if-statements in it too,

```
L = [i**2 for i in range(10) if i!=5]
```

Figure out what that did.

3.4 What should I use when?

It really depends, but my typical order is:

1. List comprehension
2. For loop
3. While loop - Mainly if I want a value to end up within some tolerance

However, as we'll see soon, loops are *slow* so they are best avoid altogether (if possible).

4 Functions

The general way to define a function is,

```
def function_name(inputs):
    do stuff
    return thing
```

For example, here is $f(x) = x^2$,

```
def f(x):
    return x**2
```

You use this by typing,

```
f(3)
```

That should give you 9. Well, it'll probably give you nothing since we didn't print.

It's good practice to have a comment on the second line of the function.

```
def function_name(inputs):
    #Describe function, inputs and outputs
    do stuff
    return thing
```

This is how Python does the "help" stuff.

You can have a "return" statement anywhere in a function, but by convention you should only have one and it should be at the end.

4.1 Optional Arguments

In the range function defined above, the `step` variable defaults to 1. This is an optional argument. Here is another example,

```
def f(x=1,y=3):
    return x**2 + 2*y
```

This is very simple, but let see what we can do with it.

```
f() #1**2+2*3 -> 7
f(2) #2**2+2*3 -> 10
f(2,1) #2**2+2*1 ->6
f(y=2) #1**2+2*2 -> 5
f(y=2,x=3) #3**2+2*2 -> 12
```

You can change the optional variables using *keywords*. If you don't use the keywords, the arguments must be in order. If you use the keywords, the order is irrelevant.

When defining your own functions, optional arguments must be placed after the required arguments.

4.2 Multiple Outputs

Of course you can return more than a single value, just wrap them in a tuple.

```
def f(x):
    return (x,x**2)
```

4.3 Lambda Functions

Every once in a while, you need to use a function exactly once. And it's overkill to write an entire function to do the job. This is the use of *lambda functions*. For example, lets say we want to sort a list of tuples by the second argument, `sorted([(3,2),(4,1),(1,3)])` will sort by the first coordinate. To fix this,

```
sorted([(3,2),(4,1),(1,3)],key = lambda x:x[1])
```

The *key* argument is telling the sort function how to sort the list.

Lambda functions are single use, simple functions. If something is long or complicated, make a real function. These may prove useful as we start integrating, differentiating and other fun stuff.

5 Reading Files

5.1 csv

Here is a function that will load data from a csv file.

```
import csv


def load_csv(file_name)
    #Load a CSV file where the entries are real numbers. Returns a list of lists, [[row]]
    data = []
    with open(file_name,"r") as d:
        csv_file = csv.reader(d)
        for row in csv_file:
            data.append([float(elm) for elm in row])
    return data
```

Go ahead and try to figure this out. Feel free to use this directly.

5.2 text files

We won't be loading many text files in this class, but you'll be writing some (as TeX files). Let's talk about the `open(file,mode= ‘r’,...)`[1].

- “file” input is a path to a file, it is relative to your working directory.
- “mode” is how you'll be using it ‘r’ for read, ‘w’ for write.

Here is an example writing to a file

```
out = "I’m a string\nAnd I’m on a new line"
with open("test.txt","w") as d:
    d.write(out)
```

If you run this you'll have a file called “test.txt”. This can be super useful for generating TeX files and outputting results.

[1]Full documentation: `https://docs.python.org/3/library/functions.html#open`, you should practice reading stuff like this.

Numerical Methods

Math 3338 – Spring 2022

Homework 2 (Due: Thursday, January 20)

Problem 1 (1 pt) Generalize the `fizzbuzz` program from Homework 01. This time we want an optional argument *fizzy* this takes the form of a dictionary and defaults to {`3:"fizz",5:"buzz"`}. In general the input takes form {`number:string`}. The output is similar to fizzbuzz, except it uses the numbers and strings. For example, your function definition should have the form

```
def fizzbuzz(n, fizzy = {3:"fizz",5:"buzz"})
```

Here is some sample output of `fizzbuzz(10,{2:'Dr.  Mitch',3:'Rules'})`

```
Dr. MitchRules
Dr. Mitch
Rules
Dr. Mitch
Dr. MitchRules
Dr. Mitch
Rules
Dr. Mitch
```

Problem 2 (1 pt) Write a function that will solve any quadratic polynomial, $ax^2 + bx + c = 0$. You should have 3 inputs, a, b and c and the output should be a list of tuples $[(x_1, x_2), (y_1, y_2)]$ where $x_1 < y_1$ or $x_2 <= y_2$ if $x_1 = y_1$.

So for example, if the solutions are $3 \pm 2i$, the output is $[(3, -2), (3, 2)]$. If 3 is the only solution, the output is $[(3, 0), (3, 0)]$

The function should be `quadratic(a,b,c)`.

There is a Pickle file on Canvas to test your function. It's a list of tuples `(a,b,c,sol1,sol2)`. It'll be a good idea to use `fuzzy_equal` to check your answers.

Problem 3 (1 pt) Write a function that finds the minimal element of a list.

Do not use the built-in "min" function.

The function should be `my_min(L)`.

There is a pickle file on Canvas. The format is a list of tuples, `[(List,min)]`

Problem 4 (1 pt) Write the following piecewise function,

$$f(x) = \begin{cases} x^2, & x \leq -1 \\ \frac{x}{4} - 2x, & -1 < x < 2 \\ 2^{x-1}, & 2 \leq x \end{cases}$$

Call the function `f(x)`.

There is a pickle file on Canvas. It's a list of tuples, `([x,f(x)])`