Introduction to computing and data handling 2023

5th class While loop, string operations, input/output 7th Sep

Ohter numpy array functions

Go through the following numpy array functions and see what they do.

- *concatenate,*
- *stack,*
- *split,*
- *insert,*
- *append.*

While loop, tolerance, *break* statement

While is another iterative process. Its basic operation style is,

- evaluate a condition and check if it is true or false,

- do an operation if condition is true, then go back to the beginning of the loop and continue the process,

- exit the loop if condition fails and execute the next line.

```
>>>n=10
>>>while n>0:
>>>          print(n)
>>>          n=n-1
>>>print 'end of loop'
```

You can use the break statement to get out of the body of the iterative process.

Exercise. An expression to calculate square root of a number *a* iteratively starting from a guess value *x* is,

$$y = \frac{x + a/x}{2}$$

Let us assume we want to find the square root of 9.0, and we assume the value is 2.0. The first iteration produces a number close to the real value of √9. But we plan to keep iterating for a better estimate.
When to stop the iteration?
Set a condition and use the break statement to exit from the loop.

```
>>>if y==x:
>>>        break
```

It is always better to use a tolerance, because of round off error, comparing two floats is not a good idea. Remember the exercise in the first class, and see

```
>>>if 0.1+0.1+0.1 == 0.3:
>>>        print('yes')
```

Let us define a tolerance.

```
>>>epsilon=0.00001
>>>if abs(y-x) <epsilon:
>>>        break
```

It is possible to define a relative tolerance *abs(y-x)/x*. We will see the use of both absolute and relative tolerance later.

Basic string operations

Quite a bit of string operations are now even more intuitive as we have already studied numpy array.

Strings are ordered sequences, so are lists, and tuples.

```
>>>s='This is a string'
```

The below functions are easy to understand. See what they do.

```
>>>len(s)
>>>s[0]
>>>s[-1]
```

>>>s[::-1]

Try out the following string methods.

>>>s.upper()

>>>s.find('i')

What does the *in* operator do?

It is a boolean operator returning *True* or *False*.

>>>'s' in s

>>>'t' in s

Exercises

1. Write a program which accepts a name from a user and prints the same name with all letters capitalized.

2. Create a program which defines a list [1,2,3,4,5]. Next, it should accept a number from the user and check if it is inside the list. If in the list it should display "Yes, found it!" or it should display "Sorry, could not find."

3. Write a program that recognizes palindromes (i.e. words that look the same written backwards). For example, is_palindrome("radar") should return True.

4. Read a sentence from user and split them to words and print one word in a line.

5. Write a program to print the following:

```
1
22
333
4444
55555
```

Reading and writing ascii files

Create a file named *inputfile.txt* on the hard disk.

Let the content of the file be '*This is a file*' (without quotes)

To read a file named *inputfile.txt*,

```
>>>f=open('inputfile.txt','r')
>>>print('f')
>>>f.read()
```

You may see a 'newline character' $\backslash n$ which indicates that the file ends at that line.
It is possible to read only a part of the file, by specifying the number of characters you want to read inside the open-close bracket of read function.
What happens if you try to read one more time?
While resolving the above problem, make sure to close the file. It is a good practice to follow.

```
>>>f.close()
```

Use of *readline*
Create a new file named *inputfile_readlines.txt* having three lines as below.

```
This is a file
Having more than one line
It is used to check readline(s)
```

Now open this file as usual.
Try the following three methods and see how they differ.

Method-1

```
>>>f.readline()
```

Method-2

```
>>>f.readlines()
```

Method-3

```
>>>for x in f:
>>>             print(x)
```

Here, the *in* operator takes up an iterative task, looks for each lines in the file and prints it out one by one. This method is normally used to iterate over strings, tuples, and lists.
Try,

```
>>>x='this is a string'
>>>y=(1,2)
>>>z=['a','ab','abc']
>>>for j in x:
 >>>          print(j)
```
Or
```
>>>for j in y:
>>>             print(j**2)
```
Or
```
>>>for content in z:
>>>             print(2*content)
```

Reading/writing data.
For simple data files we will use *numpy*. Later, we will try to work with *pandas* data handling library which is far more powerful and versatile.

Check *np.loadtxt*, *np.genfromtxt*, *np.savetxt*

Exercise. Read the attached file *"blazar_S0716_trim.dat"* using *np.loadtxt* and plot the last 3 columns of data as *x* vs *y* vs *y_error* using *plt.errorbar* function. Label your plots, use easy to read font size and labels. There are different ways to handle this file within *np.loadtxt*. One approach is to use *usecolumns* keyword, another is to use *dtypes*.

Terminate the program in the middle. Use,
```
>>>exit()
```
and
```
>>>import os
>>>os.exit()
```
to see their operation.