In-Class Work: Python — Decisions, Repetitions, Lists & Arrays, and Functions

15. Decisions

Read §2.3.1 and as you read try the commands on page 39

```
x=int(input("Enter a whole number no greater than ten: "))
if x > 10:
    print("You entered a number greater than ten.")
    print("Let me fix that for you.")
    x=10
print("Your number is",x)
```

and the commands at the bottom of page 41

```
x=int(input("Enter a whole number no greater than ten: "))
if x>10:
    print("your number is great than ten.")
elif x>9:
    print("Your number is OK, but you're cutting it close.")
else:
    print("Your number is fine. Move along.")
```

Continue reading §2.3.2 and try the commands

```
x=int(input("Enter a whole number no greater than ten: "))
while x>10:
    print("This is greater than ten. Please try again.")
    x=int(input("Enter a whole number no greater than ten: "))
print("Your number is",x)
```

Continue reading §2.3.3 (The third line needs the correction "is not met" to instead "is met".) but stop reading on page 45 before you get to the commands `f1=1` and following lines. Put the book aside and try yourself to write a program which determines the Fibonacci numbers. When you succeeded, continue reading on page 45.

Try the commands on top of the page 46

```
f1,f2=1,1
while f1<=1000:
    print(f1)
    f1,f2=f2,f1+f2
```

and try to understand how exactly this program works. If you google "Fibonacci Nature" you get some beautiful examples and explanations for the occurance of fibonacci sequence in nature.

16. Repetitions Sample Program

To practice how repetition commands work, copy into your working directory the sample program

```
~kvollmay/share.dir/pythonsamples.dir/sample_repetitions.py
```

Look at this program and before running this program, lookup online (or remind yourself) the used commands. Still before running this program, predict what exactly the program will print on the screen when you run the program. After your prediction, run the program and compare with your prediction.

17. Factorial

Write a program which reads in N and prints out $N!$. (Use a repetition command.)

18. Lists & Arrays Intro: Containers

For examples how to use arrays copy into your working directory the sample program

```
~kvollmay/share.dir/pythonsamples.dir/sample_arrays.py
```

To learn more about lists and arrays, scan this sample file and then read in Newman as described in the following steps.

Lists

Read §2.4.1 and as practice try for page 48 the following commands:

```
r=[1,1,2,3,8]
print(r)
```

and also

```
x=1.0
y=1.5
z=-2.2
r=[x,y,z]
print(r)
y=2.0
print(r)
r=[x,y,z]
print(r)
```

On page 49 and all following pages of this section, replace the `math` package with the `numpy` package the same way as in last class. For example `sqrt(3.0)` is instead `np.sqrt(3.0)`
In additon try the commands

```
a=[3,2.5,4,"hi",-3]
print(a)
print(a[0],a[1],a[3])
a[1]=7.3
print(a)
```

and for page 51 try

```
import numpy as np
r=[1.0,1.5,2.2]
logr=list(map(np.log,r))
print(logr)
```

and for pages 51 and 52 test

```
r=[1.0,1.5,2.2,9.1]
print(r)
r.append(2.8)
print(r)
r.pop()
print(r)
r.pop(1)
print(r)
print(r[2])
```

Arrays

Read §2.4.2 pages 53 and 54. We will also use numpy, but import differently so the commands on page 54 are replaced by

```
import numpy as np
a=np.zeros(4,float)
print(a)
```

and try also

```
b=np.zeros([2,3],int)
print(b)
```

for pages 55 and 56 try

```
d=np.array([[1,2,3],[4,5,6]])
print(d)
d[1,2]=30
print(d)
```

Continue reading, so read §2.4.3. Notice that `loadtxt` allows us to read data from a file, so we do not have to type in by hand each value. Get a feel for how it works, by using in your linux terminal `gedit` (or any other editor you like) to make the file named `values.txt` with the content as the four last lines on page 57. Then you can use a python script which includes the line `import numpy as np` to test the python commands

```
e=sp.loadtxt("values.txt",float)
print(e)
```

In case this made you curious about how to also write into a file of a specified name, you may have a look at `~kvollmay/share.dir/pythonsamples.dir/sample_inout.py`

19. Traffic Model (application of §2.4 & §2.5)

Copy the file

`~kvollmay/share.dir/inclass2021.dir/road.dat`

into your working directory. The file contains 100 integers describing a road of 100 lattice sites at a certain time. -1 means an empty site and a number ≥ 0 means a car with the corresponding speed in mi/h. (**Hint:** use from `~kvollmay/share.dir/pythonsamples.dir/sample\_arrays.py` the command `sp.loadtxt`)

19a. Read in this file, so that you have an array with your values. Check with a print command that you succeeded.

19b. How many cars N are on this road?

19c. What is the average velocity

$$\langle v \rangle = \frac{1}{N} \sum_{i=1}^{N} v_i$$

where v_i is the velocity of car i and there are N cars.

19d. What is the largest velocity ?

20. Fun with Matrix

20a. Write a program that sets an array to be as follows:

```
0 0 0 0 0
0 1 1 1 0
0 0 1 0 0
0 0 0 0 0
0 0 0 0 0
```

Check your program with a print command.

20b. Now change the printing so that there are no square brackets being printed.

20c. Define the length of the array, i.e. 5, as variable you set at the beginning of your program. This will allow us later to change the size of our lattice easily.

20d. (Tool for Fun) You can look at this array graphically by adding to your python script the following command at the beginning

```
import pylab as pl
```

and then after you have set your array the following two lines (assuming that your array is called `lat`)

```
pl.imshow(lat,interpolation='nearest')

pl.show()
```

For more information Newman describes this tool in §3.3.

21. User-Defined Functions

21a. Start reading in Newman's book §2.6 and as practice for the understanding of page 76 and the first half of page 77, test the commands

```
def factorial(n):
  f = 1.0
  for k in range(1,n+1):
    f *= k
  return f

print(factorial(10))
```

21b. Next add to your program that n is read from input and then $n!$ is printed.

21c. Next let's combine this user-defined function with the repetition tools from last class. Write a program which prints out n and $n!$ for $n = 1, 4, 7, 10, 13, \ldots, 22$.

21d. Continue reading on page 77, and include in your program the user-defined function `distance` . If you want to use `scipy` make sure to include in your program
`import scipy as sp`
and now add to your program (or write a small new python script) with the lines

```
def distance  (r,theta,z):
  x = r*np.cos(theta)
  y = r*np.sin(theta)
  d= np.sqrt(x**2+y**2+z**2)
  print("x ",x," y= ",y," d=",d)
  return d
```

and test your program for the case of $r = 2.0, \theta = 0.5\pi, z = 1.0$ and check whether you get the expected value.

21e. Next write a function which reads in as integer
a total number of seconds and returns the time interval in the form of hours, minutes, and remaining seconds. Test your program with a few examples.

21f. Continue reading §2.6 and then include in a program the definition of `factors(n)` as defined on page 81. Test this function by reading in an integer and printing the factorlist.

22. Good Programming Style

Read carefully §2.7. I could not agree more with each of Newman's rules.

Flow Chart:(if time) start working on the flow chart for your main project