Introduction to Scientific Computing with Python L1

May 11, 2020

Dr. Pietro Giampa

TRIUMF

2020 Winter Term

Lecture 1/3

0.1 Outline

```
1) Why Do We Need Programing in Physics?
2) Basic Variables and Operations
3) Building Loops (for, while)
4) Set Conditional Statements (if, elif, else)
5) Contructing Functions
6) Exercises
```

1 Why Do We Need Programing in Physics?

Programming is used to perform operations that would otherwise too cumbersome to do by hand:

```
* Multi-step calculations
* Data-streaming
* Data-handling
* Data-visualization
* Simulations
```

Python is lately gaining more and more traction in physics because is a great "high level" programming language. Most of the "annoying" things from other languages like C++ or java are automated or smoothed out in Python. Moreover, Python has a very large range of available libraries for all sorts of actions and functionality (machine learning techniques, data-visualization libraries, multiple simulation packages, etc.).

Examples of Python Applications at TRIUMF:

```
* Beam monitoring software
* Different experiments data analysis
* Theoretical calculations
* Optical simulations
```

2 Basic Variables and Operations

One of the advantages of Python is that the list of variables types is reduced to a minimum to help users. Fundamentally there are three types of variables:

```
* NUMBERS (10, 23.89, 0x100, 3.1j)
* STRINGS ('Hello World','Type Here','I am a string example')
* BOOLEAN (true, false)
```

Strings and Boolean are a simple set, while with numbers you have multiple options:

```
* INTEGER - signed integer - [0, 1, 2, 3 .... etc]
* LONG - long integer, octal and hexadecimal - [51924361L, 0x19323L .... etc]
* FLOAT - floating point real values - [0.32, 100.2, 54.67899 .... etc]
* COMPLEX - standard complex numbers - [7.12j, 0.876j, 23j ... etc]
```

Let's run some examples. As mentioned before, the vast range of Python libraries is what makes Python GREAT. In today's lecture, we are introducing a library called NumPy, which is a compilation of mathematical functions and arrays (or vectors) operations (addition, subtraction, multiplication, sqrt, pow, ... etc). To import libraries you need to tell Python to fetch a given library and give it a tag name.

Example: import 'library-name' as 'tag-name'

NB: Please note that all the libraries that we are going to use in this course are already installed in Jupyter, however, if you need a library that is not already set you will need to install that library first.

[1]:
```
### Import Needed Libraries ###
import numpy as np
```

2.0.1 Example 1.2.1

You are on a plane from Vancouver to Ottawa and you know the avg cruising speed (1024.0 [km/h]) and the length of the travel (3.8 [h]) and you want to estimate the distance you travelled.

PS: Air Canada removed distance travelled from their interactive Map (very annoying).

[2]:
```
############################################################
## Example 1.2.1                                          ##
## Introduction to Scinetific Programming with Python     ##
##                                                        ##
## Your Name, Institution Name, Year                      ##
## Pietro Giampa, TRIUMF, 2019                            ##
############################################################

### Import Needed Libraries ###
import numpy as np

### Define Variables ###
```

```
avg_speed = 1024.0 # [km/Hour]
travel_time = 3.8 # [Hour]

# Direct operation with individual variables
# Treat variables as single numbers
distance_travel = avg_speed*travel_time

### Print Results ###
print('First Method')
print(distance_travel, '[km]')

# Operation via numpy
# Treats variables as objects not just numbers (Great for Arrays)
distance_travel_v2 = np.multiply(avg_speed,travel_time)

### Print Results ###
print('Second Method')
print(distance_travel_v2, '[km]')
```

```
First Method
3891.2 [km]
Second Method
3891.2 [km]
```

2.0.2 Example 1.2.2

You are running an experiment and you have to monitor a given observable, let's say the level of the LN2 dewar you are using for cooling your experiment. You want to write a small program that lets you input an LN2 level in % and convert that number into litres (100% = 255 L, assume linearity).

[3]:
```
############################################################
## Example 1.2.2                                          ##
## Introduction to Scinetific Programming with Python ##
##                                                        ##
## Pietro Giampa, TRIUMF, 2019                            ##
############################################################

### Prompt Keyboard Input ###
Level = input('Enter Current LN2 Level [%]')

### Convert from String to Float
Level = float(Level)

### From percentage to fraction
Level = Level/100.
```

```
### Convert % to Liters ####
Liters_Converter = 255.
Level_L = Level*Liters_Converter

### Print Results ###
print('LN2 Level -',Level_L,'[Liters]')
```

```
Enter Current LN2 Level [%]32.4
LN2 Level - 82.62 [Liters]
```

3 Building Loops

Loops are a fundamental block in programming. While we are gonna focus on Python, the basic concept of loops is the same for basically all programming languages (C++, Java .... etc).

FOR-LOOP: A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

[4]:
```
for x in 'TRIUMF':
    print(x)
```

```
T
R
I
U
M
F
```

[5]:
```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

WHILE-LOOP: With the while loop we can execute a set of statements as long as a condition is true.

[6]:
```
counter = 0
while counter < 5:
    print(counter)
    counter = counter + 1
```

```
0
1
2
3
4
```

3.0.1 Example 1.3.1

Let's use Poisson statistics as an example. Say you have an experiment with an average event rate of 3.2 events every hour. You want to estimate the probability of getting either 0, 1, 2, 3, 4, and 5 events in the one hour span.

Recall the Poisson distribution:

$P(k, \lambda) = \frac{\lambda^k \cdot e^{-\lambda}}{k!}$

Where k is the expected number of events and λ is the average event rate of the experiment.

[7]:
```
############################################################
## Example 1.3.1                                          ##
## Introduction to Scinetific Programming with Python ##
##                                                        ##
## Pietro Giampa, TRIUMF, 2019                            ##
############################################################

### Define Variables ###
lambda_value = 3.2 # [events/hour]

### Initiate FOR loop ###
for k in range(6):
    Probability = (np.power(lambda_value,k)*np.exp(-lambda_value))/(np.math.
 ↪factorial(k))
    print(k,'-',Probability)
```

```
0 - 0.04076220397836621
1 - 0.1304390527307719
2 - 0.20870248436923505
3 - 0.22261598332718405
4 - 0.17809278666174724
5 - 0.11397938346351824
```

4 Set Conditional Statements

Conditional statements, alongside loops, are the absolute basics of programming. Similar to loops, we will focus on the Python syntax, however, the basic consepts of consitinoal statements holds true for most langueges (C++, Jave .... etc).

Decision making is required when we want to execute a code only if a certain condition is satisfied.

CONDITIONS

```
< - Less than ...
> - Bigger than ...
<= - Less or equal to ...
>= - Bigger or equal to ...
== - Equal to ...
!= - Different than ...
and - and option
or - or option
```

IF STATEMENT: Python makes a decision based on a test expression. if the requirement is passed the subsequent actions are taken, if not the program skips ahead.

[8]:
```
if 3<5:
    print('Requirement Matched')
```

```
Requirement Matched
```

ELSE STATEMENT: The else statement must follows an if statement. The structure works as follows: if the requirement is passed the subsequent actions are taken, if not the program excecutes the actions from the else statement

[9]:
```
if 'test'=='run':
    print('Requirement Matched')
else:
    print('Requirement Not Matched')
```

```
Requirement Not Matched
```

ELIF STATEMENT: Sometimes multiple if-statements in consecutive order are necessary (imagine a desicion tree). In that case, IF is only used for the first itertion, while elif brings the second conditional.

[10]:
```
if 'test'=='run':
    print('First Requirement Matched')
elif 'run'=='run':
    print('Second Requirement Matched')
else:
    print('Requirement Not Matched')
```

```
Second Requirement Matched
```

4.0.1 Example 1.4.1

Generate two random numbers between 0 and 1. Check if both numbers are less than 0.5 multiply them together, if only one of the is less than 0.5 add them together, and if they are both bigger or

equal than 0.5 subtract them. Print the result on the screen (include both random numbers).

[11]:
```
############################################################
## Example 1.4.1                                          ##
## Introduction to Scinetific Programming with Python ##
##                                                        ##
## Pietro Giampa, TRIUMF, 2019                            ##
############################################################

### Define Variables ###
number1 = np.random.rand()
number2 = np.random.rand()
result = 0.0 # initialize result

### Check Conditionals ###
if number1<0.5 and number2<0.5:
    result = np.multiply(number1,number2)
elif number1<0.5 or number2<0.5:
    result = np.add(number1,number2)
else:
    result = np.subtract(number1,number2)

### Print Results ###
print(result, number1, number2)
```

```
-0.12373790811193852 0.7662768517599072 0.8900147598718458
```

5 Contructing Functions

Some actions, calculations or conditionals might be used recurrently in a given program, in those cases functioncs are the best way to go.

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

functions can generate actions (like print()) or return values (individual or arrays).

[12]:
```
def linear_function(x,m,c):
    y = m*x + c
    return y
```

5.0.1 Example 1.5.1

Contruct a function to model radioactive decays by following the standard radioactive decay law:

$N(t) = N_0 \cdot e^{-\lambda \cdot t}$

Where $N(t)$ is the number of remaining daugthers at time t, N_0 is the starting daugthers concentration and λ is the half-life.

If you inject 4.6x10^{15} Nuclei of ^{222}Rn in your detector, write a program that determines how many Rn daugthers you will measure after 1 day, 4 days and 9 days.

[13]:
```
############################################################
## Example 1.5.1                                          ##
## Introduction to Scinetific Programming with Python ##
##                                                        ##
## Pietro Giampa, TRIUMF, 2019                            ##
############################################################

### Set Decay Law as Function ###
def DecayLaw(N0, lmbd, t):
    Nt = N0*np.exp(-lmbd*t)
    return Nt

### Define Variables ###
time1 = 1. # [days]
time2 = 4. # [days]
time3 = 9. # [days]
lmbd = 3.8215 # [days]
N0 = 4.6E15 # [nuclei]

### Do Some Math ###
Nt1 = DecayLaw(N0,lmbd,time1)
Nt2 = DecayLaw(N0,lmbd,time2)
Nt3 = DecayLaw(N0,lmbd,time3)

### Print Results ###
print('After',time1,'days:',Nt1)
print('After',time2,'days:',Nt2)
print('After',time3,'days:',Nt3)
```

```
After 1.0 days: 100716695707085.75
After 4.0 days: 1057139696.6123526
After 9.0 days: 5.319252034262702
```

6 Exercises

Problem 1 Write a code capable of determining the force of repulsion (F_r) between two atoms. Estimate F_r for the following cases: * Two Argon Atoms, Separated By: 0.1, 1.0, and 10.0 [nm] * Two Xenon Atoms, Separated By: 0.1, 1.0, and 10.0 [nm] * One Argon and One Xenone Atom, Separated By: 0.1, 1.0, 10.0 [nm]

Your code should accept keyboard-input to receive the three cases. Using a loop and a conditinal, print on the screen the results for when d bigger than 0.5 [nm], for all three cases. Also, still using

loops and conditionals, print out which combination yield the strongest repulsion.

Recall that the Argon atomic number is 18, and the Xenon atomic number is 54. Also, remember that the repulsion force between atoms can be described by Coulumb's law:

$F_r = \frac{1}{4 \cdot \pi \epsilon_0} \cdot \frac{q_1 q_2}{d^2}$

Where F_r is the repulsion force, ϵ_0 is the permittivity of free space ($8.85x10^{-12}[N \cdot m^2/C^2]$), q_1 is first charge, q_2 is the second charge and d is the distance between them.

NOTE: Your code should include at least 2 function and 2 loops (all results must be printed on the screen with clear labling).

Problem 2 For this exercise we are going to work with the Photoelectric effect. As a reminder, the Photoelectric effect is the emission of charge (electrons) when a material is hit by light of a certain wavelength.

$E(\lambda) = \left(\frac{\hbar \cdot c}{\lambda}\right) - \psi$

Where $E(\lambda)$ is the energy of the released electron, $\hbar$ is the Plack constant (4.136×10^{-15} [eVs]), c is the speed of light, λ is the wavelength of the incoming photon, and ψ is the work function of the material.

You have to design an experiment that relies on the production of electrons via the Photoelectric effect, with different materials. Say you have 2 lasers in your lab (properties below) and you want to select the laser that work best with all material. * Write a code that asks you to select a material (via keyboard input) between the following: Si, Al, Au, and Ti. * Write a function that given your selection returns the appropiate work function (see table below for numbers). * Write another function that reproduces the output spectrum from the lasers (Assume it's Gaussian).
* In the main body, use loops and conditional statements to compare lasers. * Print your results on the screen with clear labels.

Work functions table: * Si - 4.60 [eV] * Al - 4.26 [eV] * Au - 5.10 [eV] * Ti - 4.33 [eV]

Lasers Properties: * Laser 1: $\mu = 950$ [nm], $\sigma = 285$ [nm] * Laser 2: $\mu = 750$ [nm], $\sigma = 300$ [nm]

Problem 3 write a code that generates 500 random numbers following the Gaussian distribution from Problem 2 (Laser 1). In your code (via loops and conditionals) estimate how many numbers have been generated within 1σ, 2σ and 3σ. Compare your results with the thoretical expectation.