Day 1 Complete

May 26, 2023

Session 1: Introduction to Python

Discussions:

1) Python(.py) vs Jupyter Notebook(.ipynb)
2) VS Code (local) vs Google Colab (cloud)
3) Python in Physics

Print Function: Display everything

[ ]:
```
print('Hello World')
print('statement 1','statement 2')
print('first line','second line',sep='\n')
```

```
Hello World
statement 1 statement 2
first line
second line
```

Exercise: The average temperature of New York is 55 degrees Fahrenheit. What is the average temperature in Celsius and Kelvin?

Hints:

$^{\circ}C = \frac{5}{9}(^{\circ}F - 32)$

$K =^{\circ} C + 273.15$

[ ]:
```
# convert farenheit to celcius and kelvin
farenheit = 55
celcius = (farenheit - 32) * 5/9
kelvin = celcius + 273.1
print("The temperature in celcius is: ", celcius)
print("The temperature in kelvin is: ", kelvin)

# round the temperature to 2 decimal places
celcius = round(celcius,2)
kelvin = round(kelvin,2)
print("The temperature in celcius is: ", celcius)
```

```
print("The temperature in kelvin is: ", kelvin)
```

```
The temperature in celcius is:  12.777777777777779
The temperature in kelvin is:  285.8777777777778
The temperature in celcius is:  12.78
The temperature in kelvin is:  285.88
```

Session 2: Basics of Python - Introduction to variables, data types, operators, and basic functions.

[ ]:
```
# Variables
an_integer = 5
a_float = 5.0
a_string = "5"
a_boolean = True
```

[ ]:
```
# operators
# + - * / % **
# > < >= <= == !=
# and or not
```

[ ]:
```
# Example: multiplying force and distance to get work
force = 5 # newtons
distance = 10 # meters
work = force * distance
print("The work done is: ", work, " joules")
```

```
The work done is:  50  joules
```

Let's start off with some high school examples. Assume you know the masses and the velocities of two particles, how can you calculate their momenta?

Hints: $p = mv$

[ ]:
```
# Given masses and velocities
m1 = 6
m2 = 43

v1 = 68
v2 = -32
```

[ ]:
```
# Calculate the momenta
p1 = m1 * v1
p2 = m2 * v2

print("Momentum of the first particle: ", p1)
print("Momentum of the second particle: ", p2)
```

```
Momentum of the first particle:  408
Momentum of the second particle:  -1376
```

What about their kinetic energy?

Hints: $K = \frac{1}{2}mv^2$

[ ]:
```
K1 = 1/2 * m1 * v1**2
K2 = 1/2 * m2 * v2**2

print("Kinetic energy of the first particle: ", K1)
print("Kinetic energy of the second particle: ", K2)
```

```
Kinetic energy of the first particle:  13872.0
Kinetic energy of the second particle:  22016.0
```

Now, let's assume these two particles are the decay product of a stationary particle. Can we verify the conservation of momentum in this case?

[ ]:
```
# Verify momentum conservation
p0 = 0  # The initial momentum is zero because the initial particle is at rest

# conserve_bool = (p0 - p1 - p2 == 0)
conserve_bool = (p0 == p1 + p2)

print("Is momentum conserved in this case?: ", conserve_bool)
```

```
Is momentum conserved in this case?:  False
```

Owh! Does this indicate that momentum conservation is not true?

Or does it indicate that there is an extra particle that we cannot detect?

Could it be dark matter?

Session 3: Control Structures - Introduction to if statements, loops, and logical operators.

Introduction to Lists, Useful List Operators

[ ]:
```
science_law = ['Newton\'s Law', 'Davinci\'s Law', 'Gas Law']
print('the first element is: ',science_law[0])
for law in science_law:
    print(law)
science_law.append('Ohm\'s Law')
print(science_law)
science_law.remove('Gas Law')
print(science_law)
science_law.pop()
print(science_law)
science_law.insert(1,'Charles\' Law')
print(science_law)
```

```
science_law.sort()
print(science_law)
science_law.clear()
print(science_law)
```

```
the first element is:  Newton's Law
Newton's Law
Davinci's Law
Gas Law
["Newton's Law", "Davinci's Law", 'Gas Law', "Ohm's Law"]
["Newton's Law", "Davinci's Law", "Ohm's Law"]
["Newton's Law", "Davinci's Law"]
["Newton's Law", "Charles' Law", "Davinci's Law"]
["Charles' Law", "Davinci's Law", "Newton's Law"]
[]
```

Repeatable Work, Let's let the computer do the work for us! (For Loop)

[ ]:
```
# for loop: for each element in an array, do something
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

[ ]:
```
# print the factors of 48
for i in range(1,49):
    if 48 % i == 0:
        print(i)

# alternative way to print the factors of 48
import math
start = 1
end = math.sqrt(48)+1
end = int(end)
for i in range(1,end):
    if 48 % i == 0:
        print(i, 48/i)
```

```
1
2
3
4
6
8
```

```
12
16
24
48
1 48.0
2 24.0
3 16.0
4 12.0
6 8.0
```

Condition, True or False? (If Statement)

[ ]:
```python
# check if a number is even or odd
for i in range(10):
    if i % 2 == 0:
        print(i, ' is even')
    else:
        print(i, ' is odd')
```

```
0  is even
1  is odd
2  is even
3  is odd
4  is even
5  is odd
6  is even
7  is odd
8  is even
9  is odd
```

Prime number is an interesting set of numbers in mathematics, let's use python to check if a number is a prime number.

[ ]:
```python
# see if a number is prime
number = 7
is_prime = True
for i in range(2, number):
    if number % i == 0:
        is_prime = False
        break
print("The number is prime: ", is_prime)
```

```
The number is prime:  True
```

(Optional) In astronomy, we would want to know which region in EM spectrum a light is situated (blue, radio, infrared etc), given frequency or wavelength. Let's use conditions to find out if the given wavelength is in the visible region.

```
[ ]: # check if a given wavelength is in the visible spectrum
wavelength = 510e-9 # meters
is_visible = False
if wavelength >= 380e-9 and wavelength <= 750e-9:
    is_visible = True
print("The wavelength is in the visible spectrum: ", is_visible)
```

```
The wavelength is in the visible spectrum:  True
```

Challenge: Develop a program to determine the EM spectrum region given frequency.

```
[ ]: input = 88.1 * 10**6
is_frequency = True

# convert to wavelength if frequency is given
if (is_frequency):
    input = 3 * 10 **8 / input

# set conditions based on http://labman.phys.utk.edu/phys222core/modules/m6/
 ↪The%20EM%20spectrum.html
if (input > 1*10**(-3)):
    output = "Radiowave"
elif (input < 1*10**(-3) and input > 25*10**(-6)):
    output = "Microwave"
elif (input < 25*10**(-6) and input > 2.5*10**(-6)):
    output = "Infrared"
elif (input < 2.5*10**(-6) and input > 750*10**(-9)):
    output = "Near Infrared"
elif (input < 750*10**(-9) and input > 400*10**(-9)):
    output = "Visible Light"
elif (input < 400*10**(-9) and input > 1*10**(-9)):
    output = "Ultraviolet"
elif (input < 1*10**(-9) and input > 1*10**(-12)):
    output = "X-rays"
else:
    output = "Gamma rays"

print(output)
```

```
Radiowave
```

With the code, you can easily determine the region of EM spectrum a light is situated in, for the following values of frequency and wavelength:

1) JWST observing frequency: 6.2 GHz
2) Hubble observing wavelength: 0.55 microns
3) Peak frequency of the sun: 6.5 x 10^14 Hz
4) The frequency of Traxx FM in Malaysia: 88.1 MHz

Install Python Library using pip

[ ]:
```
!pip install numpy
!pip install matplotlib
!pip install scipy
!pip install pandas
!pip install wget
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: numpy in
/home/mingkang/.local/lib/python3.10/site-packages (1.24.2)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: matplotlib in
/home/mingkang/.local/lib/python3.10/site-packages (3.7.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/home/mingkang/.local/lib/python3.10/site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in
/home/mingkang/.local/lib/python3.10/site-packages (from matplotlib) (23.1)
Requirement already satisfied: pillow>=6.2.0 in
/home/mingkang/.local/lib/python3.10/site-packages (from matplotlib) (9.5.0)
Requirement already satisfied: python-dateutil>=2.7 in
/home/mingkang/.local/lib/python3.10/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: cycler>=0.10 in
/home/mingkang/.local/lib/python3.10/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/home/mingkang/.local/lib/python3.10/site-packages (from matplotlib) (4.39.3)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/lib/python3/dist-
packages (from matplotlib) (2.4.7)
Requirement already satisfied: numpy>=1.20 in
/home/mingkang/.local/lib/python3.10/site-packages (from matplotlib) (1.24.2)
Requirement already satisfied: contourpy>=1.0.1 in
/home/mingkang/.local/lib/python3.10/site-packages (from matplotlib) (1.0.7)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from
python-dateutil>=2.7->matplotlib) (1.16.0)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: scipy in
/home/mingkang/.local/lib/python3.10/site-packages (1.10.1)
Requirement already satisfied: numpy<1.27.0,>=1.19.5 in
/home/mingkang/.local/lib/python3.10/site-packages (from scipy) (1.24.2)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pandas in
/home/mingkang/.local/lib/python3.10/site-packages (2.0.0)
Requirement already satisfied: pytz>=2020.1 in
/home/mingkang/.local/lib/python3.10/site-packages (from pandas) (2023.3)
Requirement already satisfied: python-dateutil>=2.8.2 in
/home/mingkang/.local/lib/python3.10/site-packages (from pandas) (2.8.2)
Requirement already satisfied: tzdata>=2022.1 in
/home/mingkang/.local/lib/python3.10/site-packages (from pandas) (2023.3)
```

```
Requirement already satisfied: numpy>=1.21.0 in
/home/mingkang/.local/lib/python3.10/site-packages (from pandas) (1.24.2)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from
python-dateutil>=2.8.2->pandas) (1.16.0)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: wget in
/home/mingkang/.local/lib/python3.10/site-packages (3.2)
```

Session 4: NumPy - Introduction to NumPy for scientific computing, including creating arrays, indexing, and basic mathematical operations.

Remember List? Numpy is like List on steroids! Very useful for scientific computing.

[ ]:
```
import numpy as np
# convert normal python list to numpy array
a = [1, 2, 3, 4, 5]
a_numpy = np.array(a)

a = np.linspace(0, 100, 11) # 11 points from 0 to 100 [0, 10, 20, 30, 40, 50,␣
 ↪60, 70, 80, 90, 100]
b = np.arange(0, 100, 5) # from 0 to 100, 5 increments [0, 5, 10, 15, 20, 25,␣
 ↪30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]

print(a)
print(b)

a_random_number = np.random.randint(5) # a random number between 0 and 4 (5␣
 ↪excluded)
print(a_random_number)
ten_random_number = np.random.randint(1, 100, 10)
print(ten_random_number)
a_random_float = np.random.uniform(-5,5)
print(a_random_float)
random_100_numbers = np.random.uniform(-5, 5, 100) # 100 random numbers from -5␣
 ↪to 5
print(random_100_numbers)
```

```
[  0.  10.  20.  30.  40.  50.  60.  70.  80.  90. 100.]
[ 0  5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95]
2
[33 37 76 42 73 10 94 39 71 27]
-0.47741942240698254
[-4.78109002  4.93180175  2.99700754  2.60011686 -1.54138996  2.15573476
  4.38002076  3.27808654  3.70940904 -2.34213571  4.15884679 -2.2002053
 -0.62304978 -2.03522911 -2.52440953 -4.78459562  1.50115494  4.67168363
 -1.18173124  2.71223747 -3.27376589 -1.36356583  4.8212638   4.63905628
 -4.43295627 -0.19315589  1.45159594  4.3529408   0.31876449  0.40033927
```

```
-1.99602189  0.69466045 -2.657893   -0.63787519  2.36459296  3.64399929
 3.81995785 -2.30367418  0.66914045 -2.09377482  1.16035342  0.59572543
 1.54104932 -0.75159361 -1.90672703 -4.79428569  1.87821834 -2.66418852
-3.32605523  2.03101473 -4.32583234  1.32181539  1.34673128 -4.42235105
 2.28098745 -2.06584453  3.75732872  0.21755955  2.5766968   2.59637401
-2.33693908  3.58273366  3.9912669  -2.72234793 -4.76463747 -2.28780829
 2.16623437  3.10418677  4.05801597 -2.24839909 -2.35968257 -0.62176054
-3.44066962 -4.39086251 -0.5385285  -1.49192806  3.96253444 -2.58378581
 4.53321425 -3.31522887 -2.4175854   3.73170377  2.25716533 -1.76652692
 1.37296474  4.1026596  -3.00039707  2.14851929 -3.72700278  0.54936705
 4.14471641 -4.08232158 -0.49288915  1.19487458 -1.44662577  1.99392461
-1.38979063  4.79008918  2.62040729 -3.4921723 ]
```

Selecting elemetns from a numpy array

[ ]:
```
period = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# take first 5
print(period[:5])

# take last 3
print(period[-3:])

# take element 3 to 6
print(period[3:7])

# take element 4
print(period[3])
```

```
[1 2 3 4 5]
[ 8  9 10]
[4 5 6 7]
4
```

Numpy Operators: +, -, *, /, **, %, //, np.sin(), np.cos(), np.tan(), np.log(), np.exp(), np.sqrt(), np.pi, np.e

[ ]:
```
# numpy array operations
kilogram = np.array([1, 2, 3, 4, 5])
print(kilogram)
gram = kilogram * 1000
print(gram)

# example: converting concentration to pH value
concentration = np.array([1e-3, 1e-4, 1e-5, 1e-6])
print(concentration)
pH = -np.log10(concentration)
print(pH)
```

```
[1 2 3 4 5]
[1000 2000 3000 4000 5000]
[1.e-03 1.e-04 1.e-05 1.e-06]
[3. 4. 5. 6.]
```

Numpy - Operation between two array

[ ]:
```
mass = np.array([1, 2, 3, 4, 5])
acceleration = np.array([4, 3, 2, 6, 3])

force = mass * acceleration
print(force)
```

```
[ 4  6  6 24 15]
```

Exercise: UM election

[ ]:
```
# numpy array 1 to 24
candidates = np.arange(1, 25, 1)
print(candidates)
# shuffle the array
np.random.shuffle(candidates)
print(candidates)
# take the first 10
winners = candidates[:10]
print(winners)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]
[ 7 20 21 18  1 11 14 10  5  2 23  8 17 24 13 19  4 22  9 15  3 12 16  6]
[ 7 20 21 18  1 11 14 10  5  2]
```

Session 4.5: Scipy - Useful to get you some famous (and not so famous) physical constants.

[ ]:
```
from scipy.constants import c, h, k, hbar, e
print('speed of light: ', c)
print('planck constant: ', h)
print('boltzmann constant: ',k)
print('reduced planck constant: ',hbar)
print('electron charge',e)
```

```
speed of light:  299792458.0
planck constant:  6.62607015e-34
boltzmann constant:  1.380649e-23
reduced planck constant:  1.0545718176461565e-34
electron charge 1.602176634e-19
```

Session 5: Plotting - Introduction to Matplotlib for plotting data, including basic line plots and scatter plots.

Discussions: 1) Matplot library 2) Types of Plots 3) Basic needs for a plot (x and y)

Exercise: Let's Plot a sin graph

Task 1: Prepare x axis data (radian)

[ ]:
```
from scipy.constants import pi

x = np.linspace(0, 6*pi, 100)
print(x)
```

```
[ 0.          0.19039955  0.38079911  0.57119866  0.76159822  0.95199777
  1.14239733  1.33279688  1.52319644  1.71359599  1.90399555  2.0943951
  2.28479466  2.47519421  2.66559377  2.85599332  3.04639288  3.23679243
  3.42719199  3.61759154  3.8079911   3.99839065  4.1887902   4.37918976
  4.56958931  4.75998887  4.95038842  5.14078798  5.33118753  5.52158709
  5.71198664  5.9023862   6.09278575  6.28318531  6.47358486  6.66398442
  6.85438397  7.04478353  7.23518308  7.42558264  7.61598219  7.80638175
  7.9967813   8.18718085  8.37758041  8.56797996  8.75837952  8.94877907
  9.13917863  9.32957818  9.51997774  9.71037729  9.90077685 10.0911764
 10.28157596 10.47197551 10.66237507 10.85277462 11.04317418 11.23357373
 11.42397329 11.61437284 11.8047724  11.99517195 12.1855715  12.37597106
 12.56637061 12.75677017 12.94716972 13.13756928 13.32796883 13.51836839
 13.70876794 13.8991675  14.08956705 14.27996661 14.47036616 14.66076572
 14.85116527 15.04156483 15.23196438 15.42236394 15.61276349 15.80316305
 15.9935626  16.18396215 16.37436171 16.56476126 16.75516082 16.94556037
 17.13595993 17.32635948 17.51675904 17.70715859 17.89755815 18.0879577
 18.27835726 18.46875681 18.65915637 18.84955592]
```

Task 2: Prepare y axis data sin(x)

[ ]:
```
y = np.sin(x)
print(y)
```

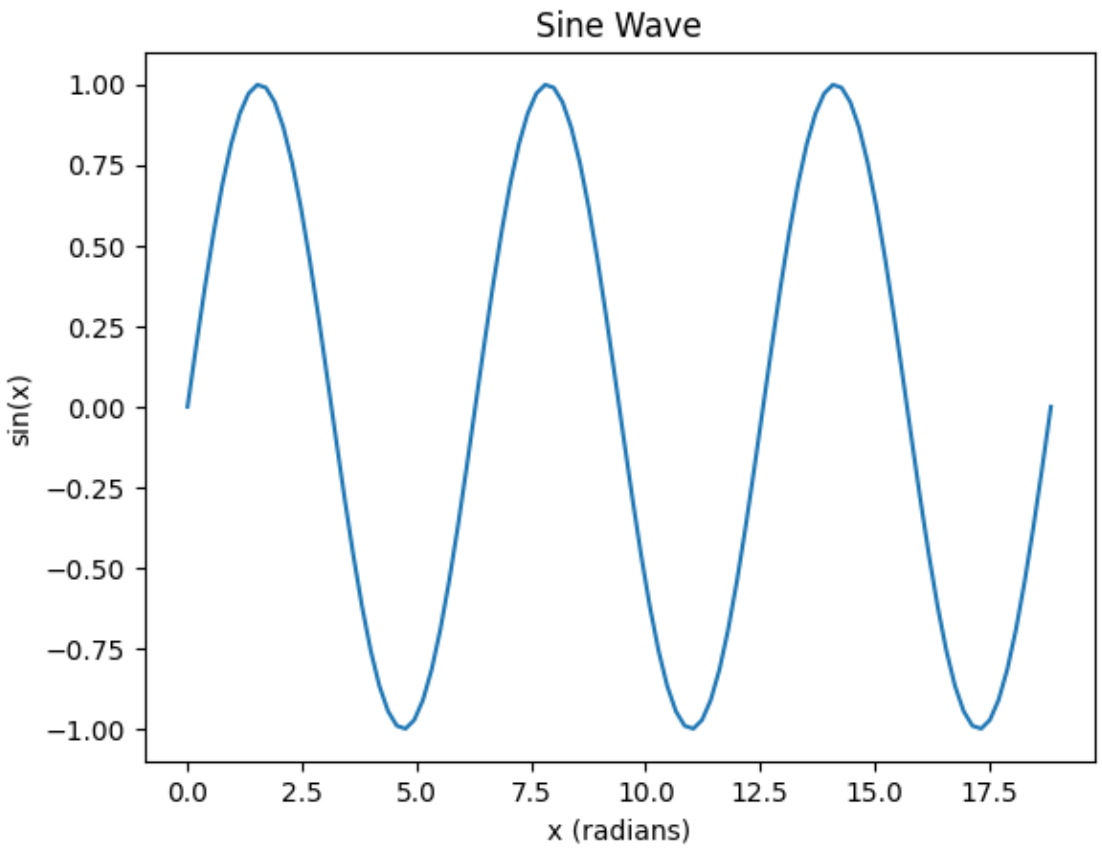
```
[ 0.00000000e+00  1.89251244e-01  3.71662456e-01  5.40640817e-01
  6.90079011e-01  8.14575952e-01  9.09631995e-01  9.71811568e-01
  9.98867339e-01  9.89821442e-01  9.45000819e-01  8.66025404e-01
  7.55749574e-01  6.18158986e-01  4.58226522e-01  2.81732557e-01
  9.50560433e-02 -9.50560433e-02 -2.81732557e-01 -4.58226522e-01
 -6.18158986e-01 -7.55749574e-01 -8.66025404e-01 -9.45000819e-01
 -9.89821442e-01 -9.98867339e-01 -9.71811568e-01 -9.09631995e-01
 -8.14575952e-01 -6.90079011e-01 -5.40640817e-01 -3.71662456e-01
 -1.89251244e-01 -2.44929360e-16  1.89251244e-01  3.71662456e-01
  5.40640817e-01  6.90079011e-01  8.14575952e-01  9.09631995e-01
  9.71811568e-01  9.98867339e-01  9.89821442e-01  9.45000819e-01
  8.66025404e-01  7.55749574e-01  6.18158986e-01  4.58226522e-01
  2.81732557e-01  9.50560433e-02 -9.50560433e-02 -2.81732557e-01
 -4.58226522e-01 -6.18158986e-01 -7.55749574e-01 -8.66025404e-01
 -9.45000819e-01 -9.89821442e-01 -9.98867339e-01 -9.71811568e-01
 -9.09631995e-01 -8.14575952e-01 -6.90079011e-01 -5.40640817e-01
 -3.71662456e-01 -1.89251244e-01 -4.89858720e-16  1.89251244e-01
```

```
 3.71662456e-01  5.40640817e-01  6.90079011e-01  8.14575952e-01
 9.09631995e-01  9.71811568e-01  9.98867339e-01  9.89821442e-01
 9.45000819e-01  8.66025404e-01  7.55749574e-01  6.18158986e-01
 4.58226522e-01  2.81732557e-01  9.50560433e-02 -9.50560433e-02
-2.81732557e-01 -4.58226522e-01 -6.18158986e-01 -7.55749574e-01
-8.66025404e-01 -9.45000819e-01 -9.89821442e-01 -9.98867339e-01
-9.71811568e-01 -9.09631995e-01 -8.14575952e-01 -6.90079011e-01
-5.40640817e-01 -3.71662456e-01 -1.89251244e-01 -7.34788079e-16]
```

Task 3: Use Matplotlib to plot the graph

[ ]:
```
import matplotlib.pyplot as plt
plt.plot(x, y)

# add a title
plt.title("Sine Wave")

# add x and y labels
plt.xlabel("x (radians)")
plt.ylabel("sin(x)")
```

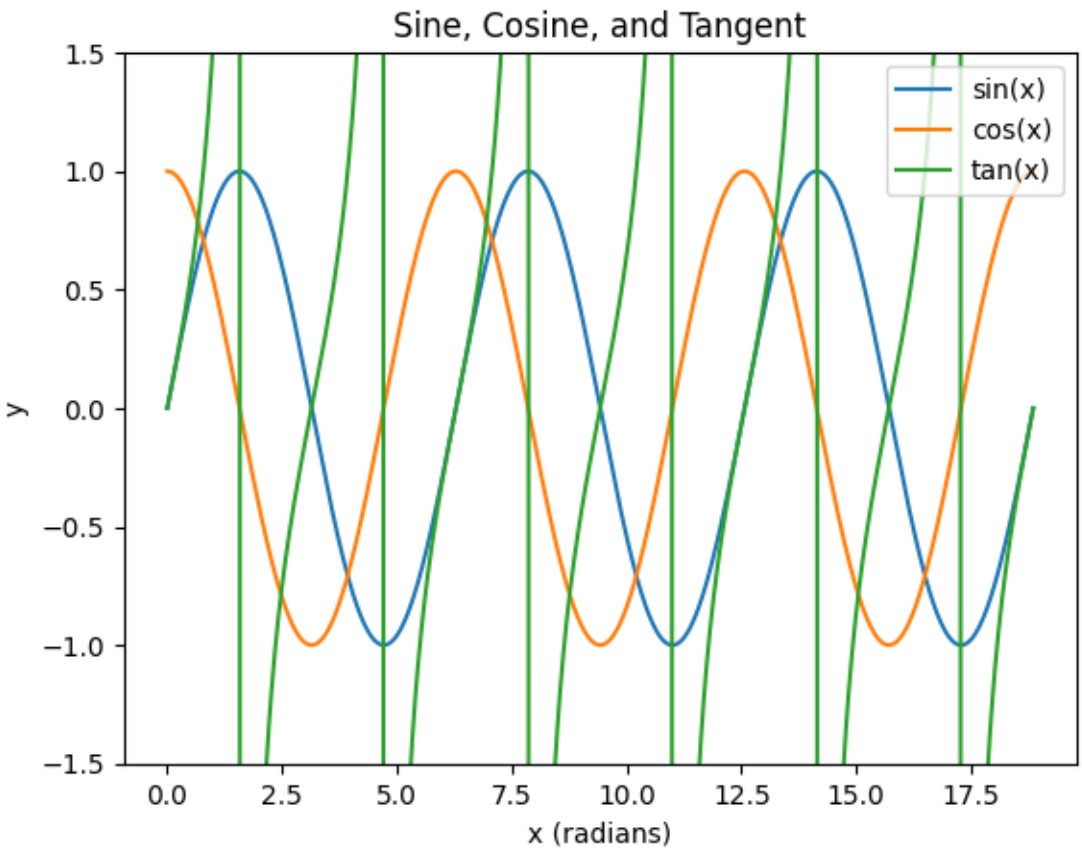
[ ]:
```
Text(0, 0.5, 'sin(x)')
```



Remember that we always got confused between sine, cosine, and tangent? Let's try to plot them all in the same graph.

[ ]:
```
x = np.linspace(0, 6*pi, 1000)

y_sin = np.sin(x)
y_cos = np.cos(x)
y_tan = np.tan(x)

plt.title("Sine, Cosine, and Tangent")

plt.plot(x, y_sin, label="sin(x)")
plt.plot(x, y_cos, label="cos(x)")
plt.plot(x, y_tan, label="tan(x)")
plt.ylim(-1.5, 1.5)

plt.xlabel("x (radians)")
plt.ylabel("y")

plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x7f6da4f2a3b0>
```



Exercise: One of the starting point of astronomy is UV catastrophe. Let's plot the original and planck equations together to see how they differ.

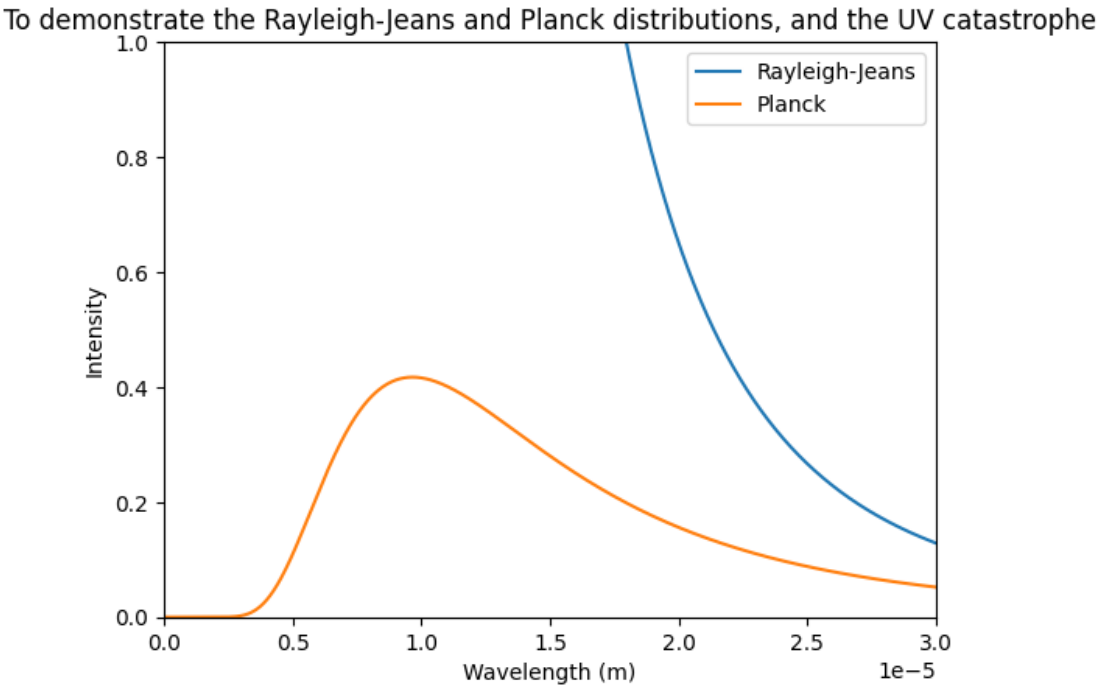
Hints: 1) Get the formulae for both equations 2) set x limit (between 0 and 3e-5) [between 0 and 30000 nanometers] 3) set y limit (between 0 and 1) [between 0 and 1] 4) use T = 300 (Kelvin) for both equations 5) use the constants from scipy

[ ]:
```
# we can make use of constants from scipy
from scipy.constants import c, h, k as k_B, pi
import numpy as np
T = 300 # temperature in Kelvin

def rayleigh_jeans(l):
    return (8*pi*k_B*T)/(l**4)
def planck(l):
    return (8*pi*h*c)/(l**5*(np.exp(h*c/(l*k_B*T))-1))

x = np.linspace(1e-9, 3000e-8, 10000)
y_rj = rayleigh_jeans(x)
```

```
y_p = planck(x)

import matplotlib.pyplot as plt
plt.plot(x, y_rj, label="Rayleigh-Jeans")
plt.plot(x, y_p, label="Planck")
plt.ylim(0, 1)
plt.xlim(0,3*10**(-5))
plt.legend()
plt.ylabel("Intensity")
plt.xlabel("Wavelength (m)")
plt.title("To demonstrate the Rayleigh-Jeans and Planck distributions, and the␣
 ↪UV catastrophe")
```

```
/tmp/ipykernel_9410/2424841557.py:9: RuntimeWarning: overflow encountered in exp
  return (8*pi*h*c)/(l**5*(np.exp(h*c/(l*k_B*T))-1))
```

[ ]:
```
Text(0.5, 1.0, 'To demonstrate the Rayleigh-Jeans and Planck distributions, and
the UV catastrophe')
```



Alternative graph, PV = nRT, because the UV catastrophe is too physics.

Day 1 Project: Cosmic Muon Count Rate. Vincent Tee has a tool to detect how many cosmic muons reach us. Let's use the data to plot the count rate of cosmic muons.

Task 1: Run the experiment and store cumulative count for every minute.

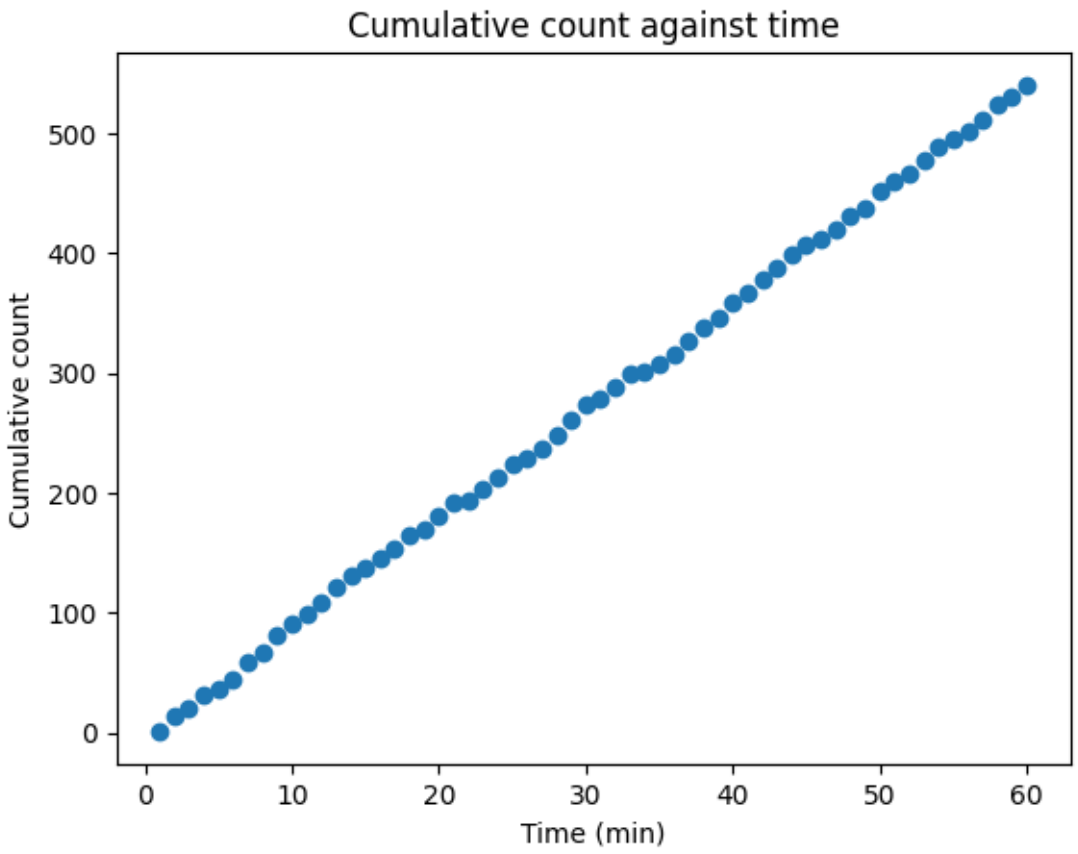
[ ]:
```
# Cumulative count for 60 minutes
cumulative_count = [1, 14, 20, 31, 36, 44, 58, 66, 81, 90, 98, 108, 121, 131,␣
 ↪137, 146, 154, 164, 170, 181, 191, 194, 203, 212, 224, 228, 236, 248, 260,␣
 ↪274, 278, 288, 299, 301, 307, 316, 327, 337, 346, 358, 366, 378, 388, 398,␣
 ↪406, 411, 420, 430, 437, 452, 459, 466, 478, 488, 495, 502, 511, 524, 531,␣
 ↪540]
time = np.linspace(1, 60, 60)
```

Task 2: Plot the cumulative count vs time

[ ]:
```
# calculate count per minute from cumulative count
first_count_per_minute = cumulative_count[0]
print(first_count_per_minute)

for i in range(1, len(cumulative_count)):
    count_per_minute = cumulative_count[i] - cumulative_count[i-1]
    print(count_per_minute)

plt.plot(time, cumulative_count, "o")
plt.ylabel("Cumulative count")
plt.xlabel("Time (min)")
plt.title("Cumulative count against time")
plt.show()
```

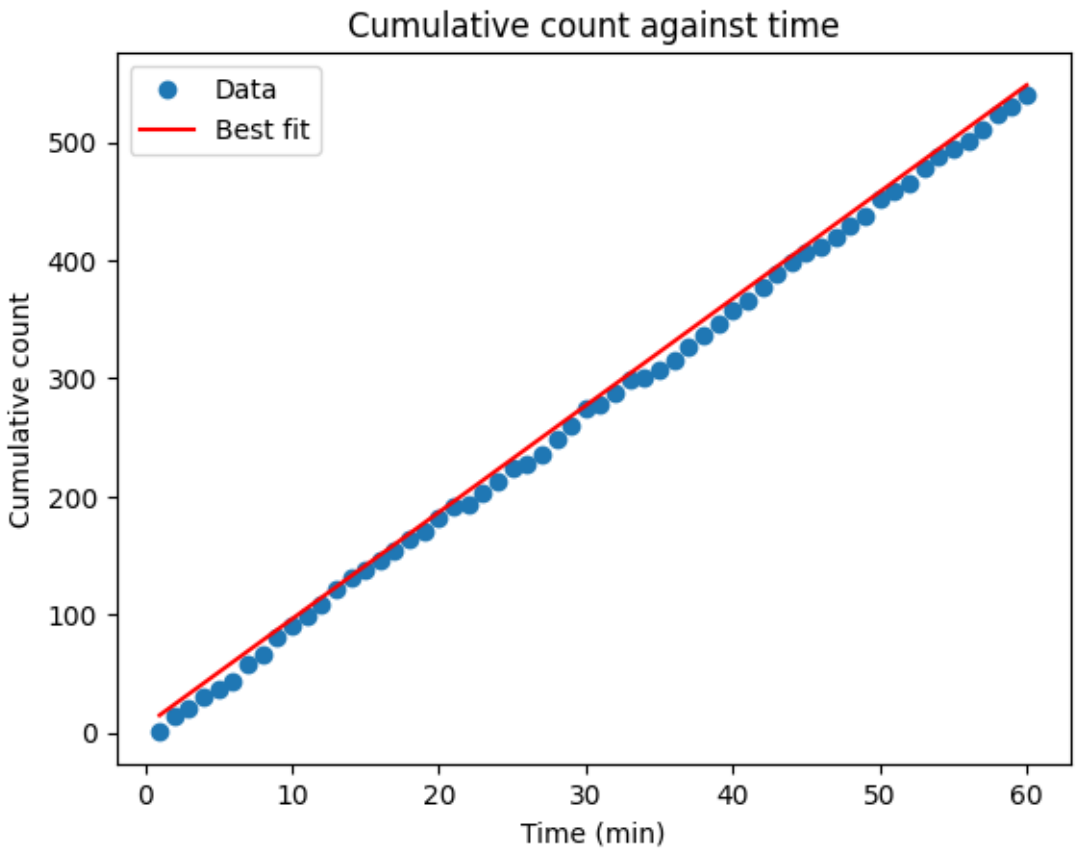
```
1
13
6
11
5
8
14
8
15
9
8
10
13
10
6
9
8
10
6
11
10
3
```

9
9
12
4
8
12
12
14
4
10
11
2
6
9
11
10
9
12
8
12
10
10
8
5
9
10
7
15
7
7
12
10
7
7
9
13
7
9



Now let's try a simple fit to the graph! This is clearly a linear relationship, so we are able to fit it with Y = m X + C

[ ]:
```
# Here is the best fit equation that I obtained by using linear regression
# y = 9.0523 x + 5.524
# Try plotting it and see if it's a good fit!

y = 9.0523 * time + 5.524

plt.plot(time, cumulative_count, "o", label="Data")
plt.plot(time, y, "r", label="Best fit")
plt.legend()
plt.ylabel("Cumulative count")
plt.xlabel("Time (min)")
plt.title("Cumulative count against time")
plt.show()
```

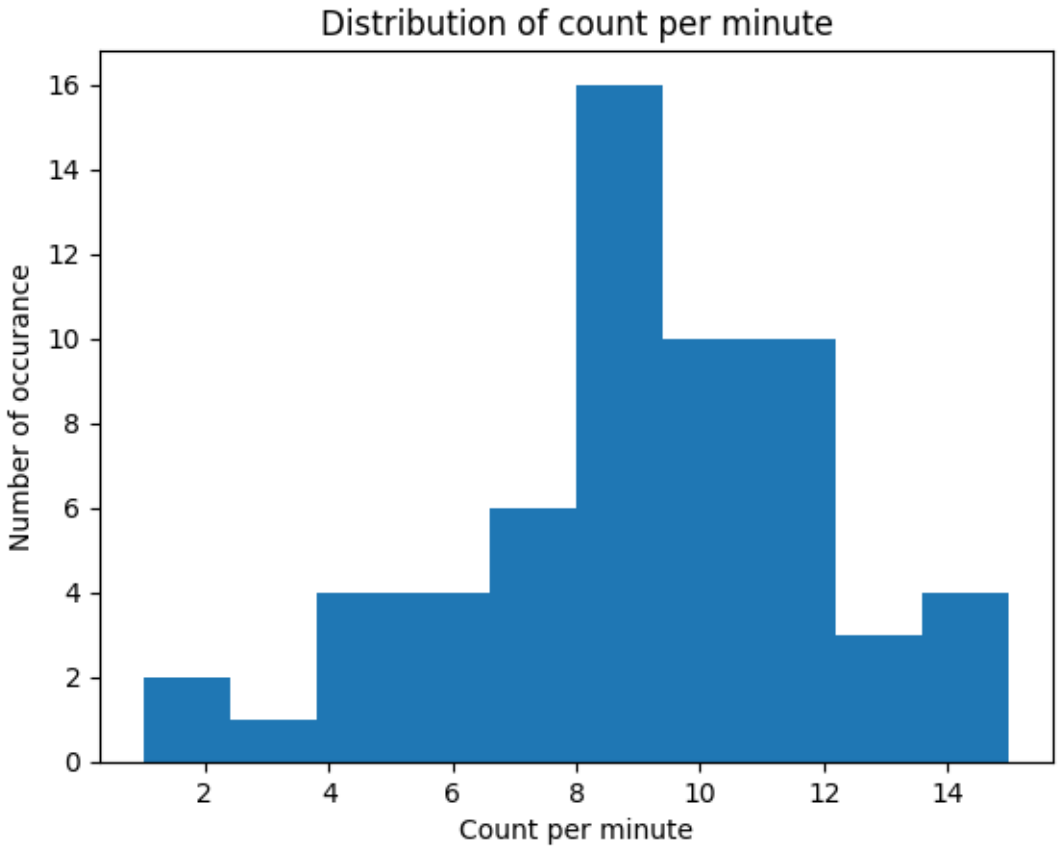


Follow up project: Since muon detection is random process, the quantity "Count per minute" should follow poisson distribution. Let's try to verify this.

[ ]:
```
# Prepare a list for count per minute
count_per_minute_list = [cumulative_count[0]]

for i in range(1, len(cumulative_count)):
    count_per_minute = cumulative_count[i] - cumulative_count[i-1]
    count_per_minute_list.append(count_per_minute)

print(count_per_minute_list)

# Plot count per minute
entries, bin_edges, patches = plt.hist(count_per_minute_list, label="Data")
plt.ylabel("Number of occurance")
plt.xlabel("Count per minute")
plt.title("Distribution of count per minute")
plt.show()
```

```
[1, 13, 6, 11, 5, 8, 14, 8, 15, 9, 8, 10, 13, 10, 6, 9, 8, 10, 6, 11, 10, 3, 9,
```

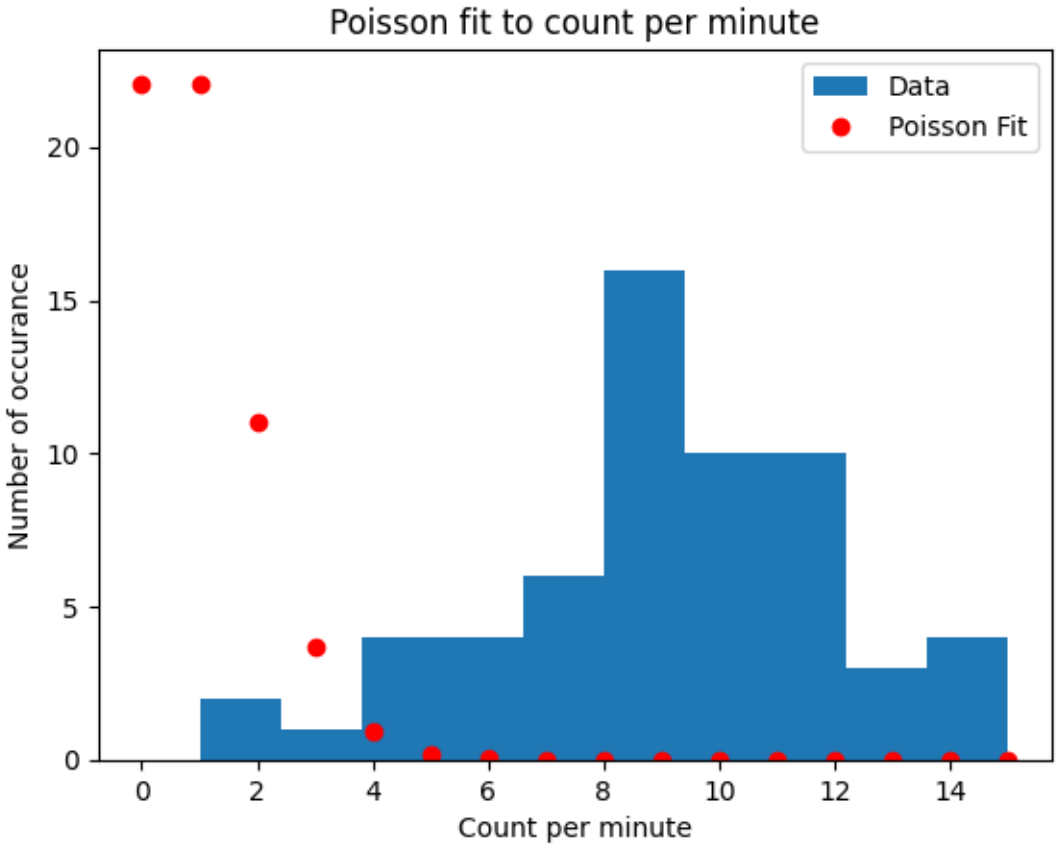
```
9, 12, 4, 8, 12, 12, 14, 4, 10, 11, 2, 6, 9, 11, 10, 9, 12, 8, 12, 10, 10, 8, 5,
9, 10, 7, 15, 7, 7, 12, 10, 7, 7, 9, 13, 7, 9]
```



[ ]:
```python
# Let's try to fit with Poisson distribution!
from scipy.optimize import curve_fit
from scipy.stats import poisson

entries, bin_edges, patches = plt.hist(count_per_minute_list, label="Data")

count = np.linspace(0, 15, 16)
nEntries = 60

def fit_function(k, lamb):
    return poisson.pmf(k, lamb)

# parameters, cov_matrix = curve_fit(fit_function, count, nEntries)
bin_centers = 0.5 * (bin_edges[1:] + bin_edges[:-1])
parameters, cov_matrix = curve_fit(fit_function, bin_centers, entries)
pois = nEntries*fit_function(count, *parameters)
```

```
plt.plot(count, pois, "ro", label="Poisson Fit")
plt.legend()
plt.ylabel("Number of occurance")
plt.xlabel("Count per minute")
plt.title("Poisson fit to count per minute")
plt.show()
```

```
/home/mingkang/.local/lib/python3.10/site-
packages/scipy/optimize/_minpack_py.py:906: OptimizeWarning: Covariance of the
parameters could not be estimated
  warnings.warn('Covariance of the parameters could not be estimated',
```



That's the end of Day 1. See you tomorrow!