1. BASICS

a) Loops & if-else statements

November 27, 2021

1 AIM

Before moving on to the linear algebraic concepts, we will first recap some basics in Python programming. This includes loops and if-else statements, which is the focus of this record.

2 Grade calculator

Write a python program to generate the grade of a student based on the percentage obtained.

[12]:
```
# Inputting percentage and validating input
while True:
    try:
        percentage = int(input("Enter student's percentage: "))
        if percentage < 0 or percentage > 100:
            percentage = 1/0
        else:
            break
    except:
        print("Invalid input!")

# Percentage to grade conversion
if percentage > 90:
    grade = "A+"
elif percentage > 80:
    grade = "A"
elif percentage > 75:
    grade = "B+"
elif percentage > 70:
    grade = "B"
elif percentage > 65:
    grade = "C+"
elif percentage > 60:
    grade = "C"
elif percentage > 55:
    grade = "D+"
elif percentage > 50:
    grade = "D"
```

```
elif percentage > 40:
    grade = "E"
else:
    grade = "F"

print("Received grade: " + grade)
```

```
Enter student's percentage: 78
Received grade: B+
```

3 Finding the factors of a number

Write a python program to accept a number from the user and predict whether it is a prime or not. If it is not a prime, then display all the factors of the number.

[8]:
```
# Inputting integer
while True:
    try:
        n = int(input("Enter an integer: "))
        break
    except:
        print("Not an integer!")

# Checking if prime, and finding factors if not
i = 2
isPrime = 1
factors = []
# This uses the minimum exit condition to check if n is prime
while i * i <= n:
    if n % i == 0:
        factors.append(i)
        isPrime = 0
    i = i + 1
# Finding the remaining factors if number is non-prime
if isPrime == 0:
    while i <= n/2:
        if n % i == 0:
            factors.append(i)
        i = i + 1
if isPrime:
    print(n, "is prime.")
else:
    print("The factors of", n, "are", factors)
```

```
Enter an integer: 677
677 is prime.
```

4 Identifying the triangle type

Write a python program to check if a triangle is equilateral, isosceles or scalene.

[11]:
```
print("Program to check if your triangle is equilateral, isosceles or scalene")
print("------------------------")
print("Enter x to input lengths of edges")
print("Enter anything else to input interior angles")
option = input()

if option == "x":
    while True:
        try:
            print("Enter lengths of edges...")
            a = float(input("a = "))
            b = float(input("b = "))
            c = float(input("c = "))
            break
        except:
            print("Non-numeric inputs!")
else:
    while True:
        try:
            print("Enter two interior angles in degrees...")
            a = float(input("a = "))
            b = float(input("b = "))
            c = 180 - a - c
            print("c =", c)
            break
        except:
            print("Non-numeric inputs!")
print("Triangle type:", end = " ")
if a == b and b == c:
    print("Equilateral")
elif a == b or b == c:
    print("Isosceles")
else:
    print("Scalene")
```

```
Program to check if your triangle is equilateral, isosceles or scalene
------------------------
Enter x to input lengths of edges
Enter anything else to input interior angles
r
Enter two interior angles in degrees…
a = 45
b = 87
c = 54.0
```

```
Triangle type: Scalene
```

5 Sum of first n natural numbers

Write a python program to find the sum of squares of first n natural numbers.

[17]:
```
while True:
    try:
        n = int(input("Enter upper limit: "))
        if n < 0:
            n = 1/0
        else:
            break
    except:
        print("Input must be a natural number!")
mySum = 0
for i in range(1, n + 1):
    mySum = mySum + i * i
print("The sum of the squares of the first", n, "natural numbers is", mySum)
```

```
Enter upper limit: 87
The sum of the squares of the first 87 natural numbers is 223300
```

6 CONCLUSION

Loops enable us to go through numerous values within a range or a list, and perform the same operations on each one until a condition is reached. This is seen in finding the factors of a number, or in adding the first n natural numbers, where the same set of operations and conditions are applied to values within a range one by one, until the desired conclusion is reached. We have also used loops for ensuring valid inputs, where the loop only breaks if the input is valid.

If-else conditions are the basic way in which we can check if certain conditions are met for one or more entities. This is seen clearly in grade calculation and identifying the triangle type, where we check if the input value or values fulfil the given conditions.

1. BASICS

b) Plotting basics

November 27, 2021

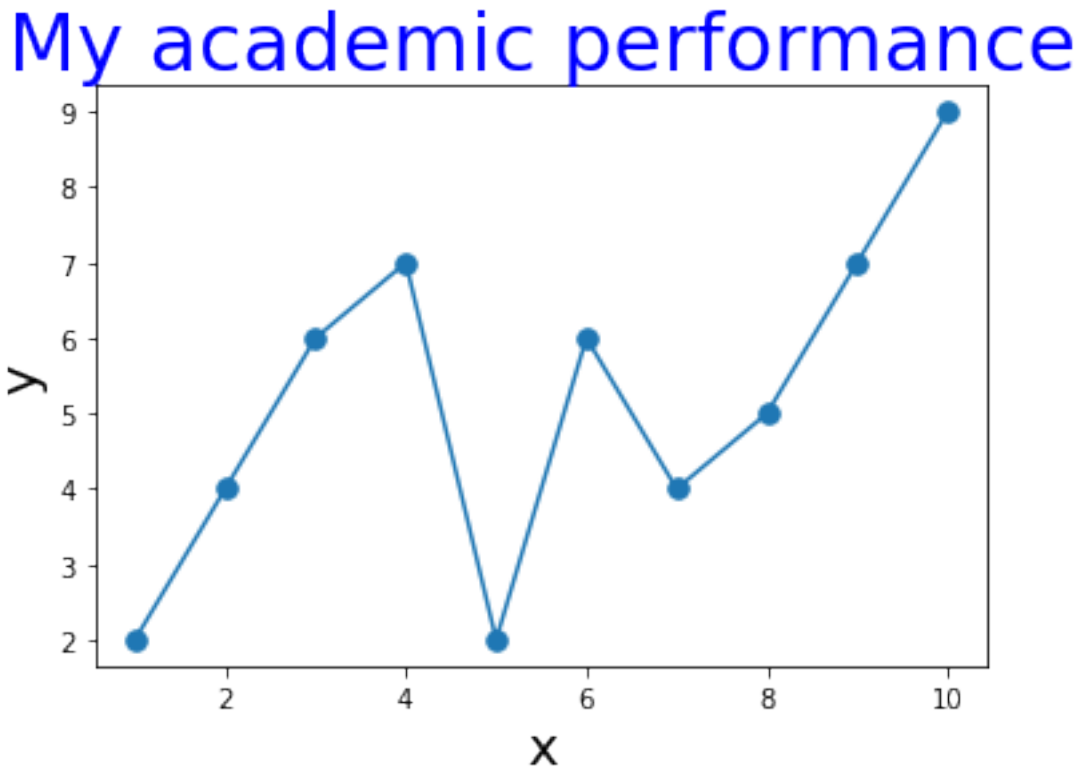
1 AIM

Plotting is one of the most essential tools in visualising data and results of our computations. Here, I will go through the basic forms of plotting, i.e. simple line graphs, scatterplots, bar graphs and histograms.

2 Simple line graph

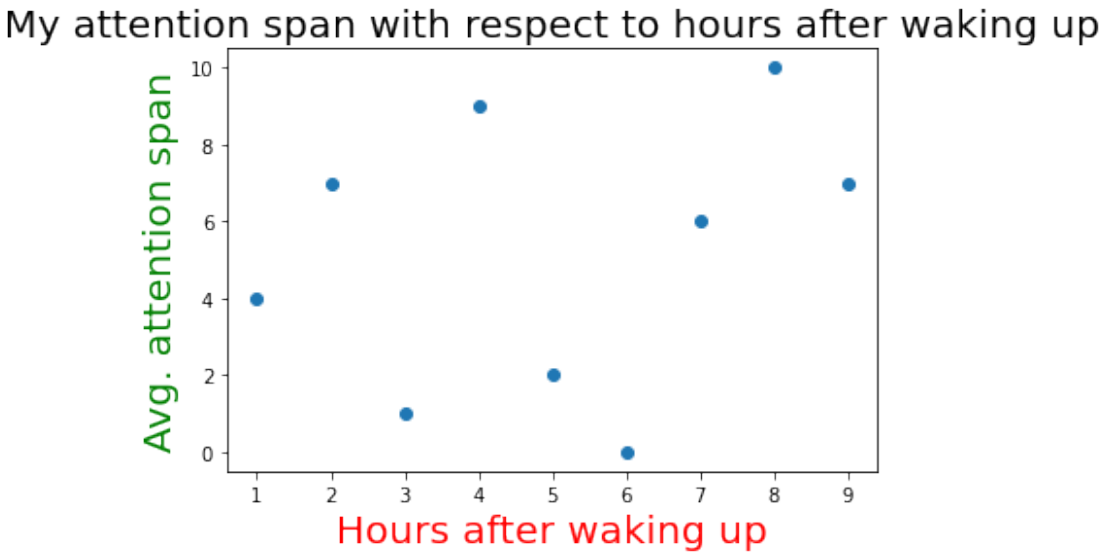
[51]:
```
from matplotlib.pyplot import scatter, plot, hist, xlabel, ylabel, title
from numpy import random
```

[55]:
```
x = range(1, 11)
y = [2, 4, 6, 7, 2, 6, 4, 5, 7, 9]
plot(x, y, marker = ".", markersize = 15)
xlabel("x", size = 20)
ylabel("y", size = 20)
title("My academic performance", size = 30, color = "blue")
None
```



3 Scatterplot

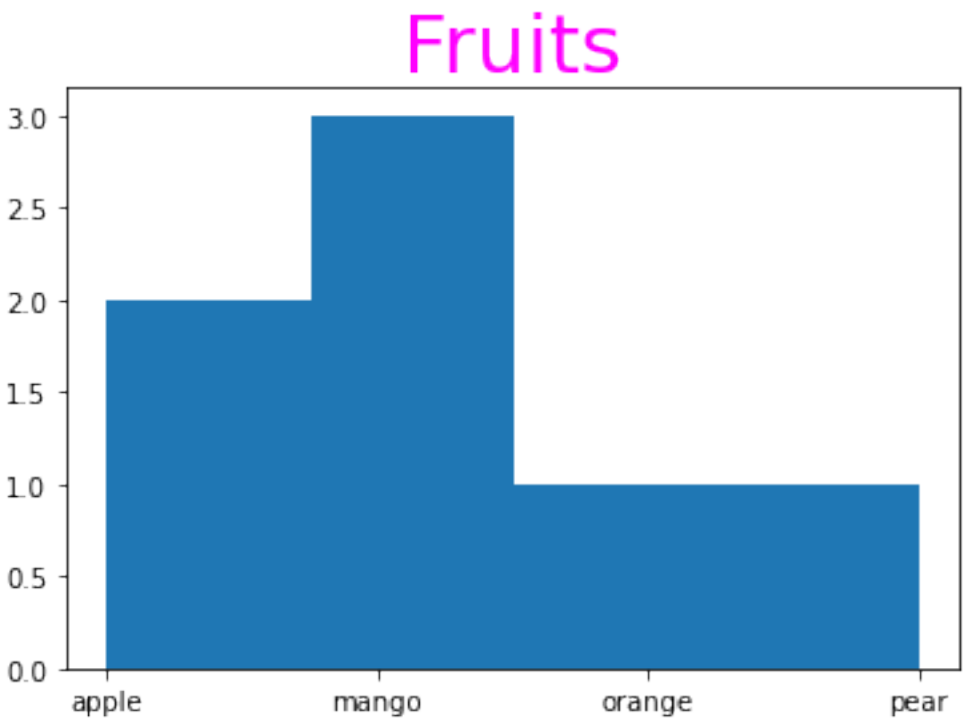
[68]:
```
x = range(1, 10)
y = [4, 7, 1, 9, 2, 0, 6, 10, 7]
scatter(x, y)
xlabel("Hours after waking up", size = 20, color = "red")
ylabel("Avg. attention span", size = 20, color = "green")
title("My attention span with respect to hours after waking up", size = "20")
None
```



4 Bar graph

While this is technically a histogram, its usage here is closer to that of a bar graph...

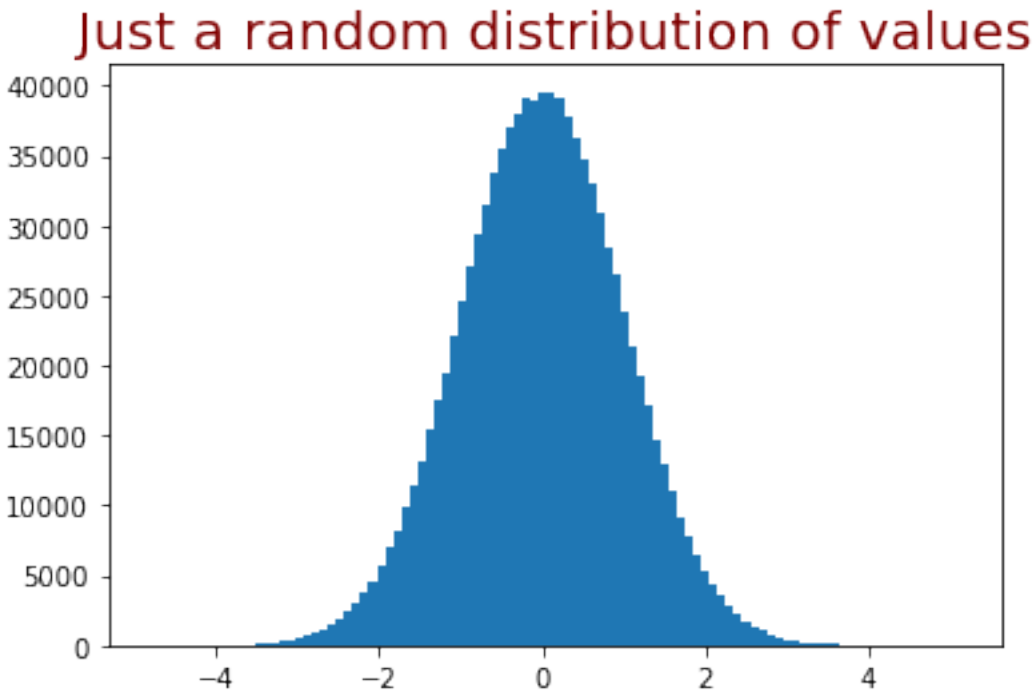
[69]:
```
# Histogram gives you the frequency of each value in the array
values = ["apple", "apple", "mango", "orange", "mango", "mango", "pear"]
hist(values, bins = 4)
# "bins" is the option for the number of different values you want to allow for
# Note that values are attached to frequencies
title("Fruits", size = "30", color = "magenta")
None
```



5 Histogram

Visualising a normally distributed random sample of numbers...

[66]:
```
x = range(1, 51)
# Creating a random normal distribution of some number of points
y = random.randn(1000000)
# As the number of points increases, the distribution becomes closer to the␣
 ↪normal curve
hist(y, bins = 100)
title("Just a random distribution of values", size = 20, color = "maroon")
None
```



6 CONCLUSION

Python provides a wide variety of plots and plotting options, and the demonstration here only showcases a limited number of these. Regardless, the above concepts will form the basis of our more advanced plotting assignments.

1. BASICS

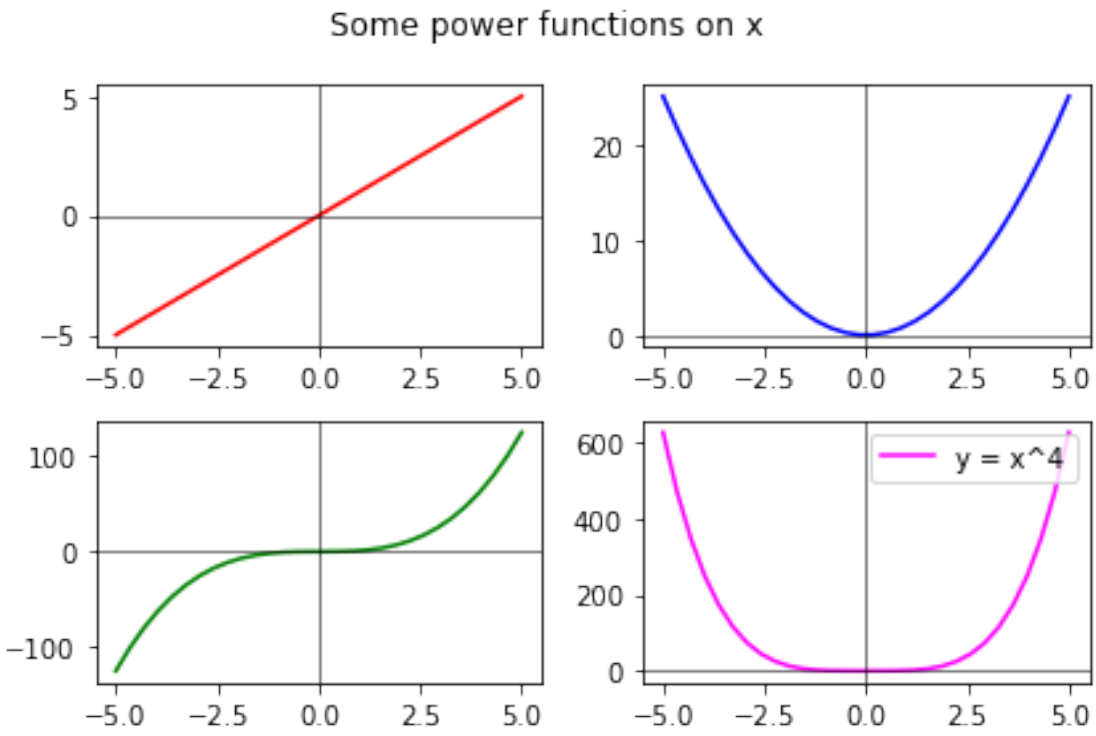
c) Subplots

November 27, 2021

1 AIM

This is a part of plotting recap. Here, we will be plotting the first four power functions i.e. x, x^2, x^3 and x^4. We will plot them as separate graphs, but also as graphs within a single grid i.e. each plot will be a subplot in a grid of plots.

[14]:
```
import matplotlib.pyplot as plt
from numpy import linspace
plt.suptitle("Some power functions on x")

x = linspace(-5, 5, 30)

plt.subplot(2, 2, 1)
plt.plot(x, x, color = "red", label = "y = x")
plt.axhline(lw=0.5, color='black')
plt.axvline(lw=0.5, color='black')

plt.subplot(2, 2, 2)
plt.plot(x, x**2, color = "blue", label = "y = x^2")
plt.axhline(lw=0.5, color='black')
plt.axvline(lw=0.5, color='black')

plt.subplot(2, 2, 3)
plt.plot(x, x**3, color = "green", label = "y = x^3")
plt.axhline(lw=0.5, color='black')
plt.axvline(lw=0.5, color='black')

plt.subplot(2, 2, 4)
plt.plot(x, x**4, color = "magenta", label = "y = x^4")
plt.axhline(lw=0.5, color='black')
plt.axvline(lw=0.5, color='black')


plt.legend()

plt.tight_layout()
None
```



2 CONCLUSION

Subplots allow us to view multiple graphs in an organised arrangement.

1. BASICS

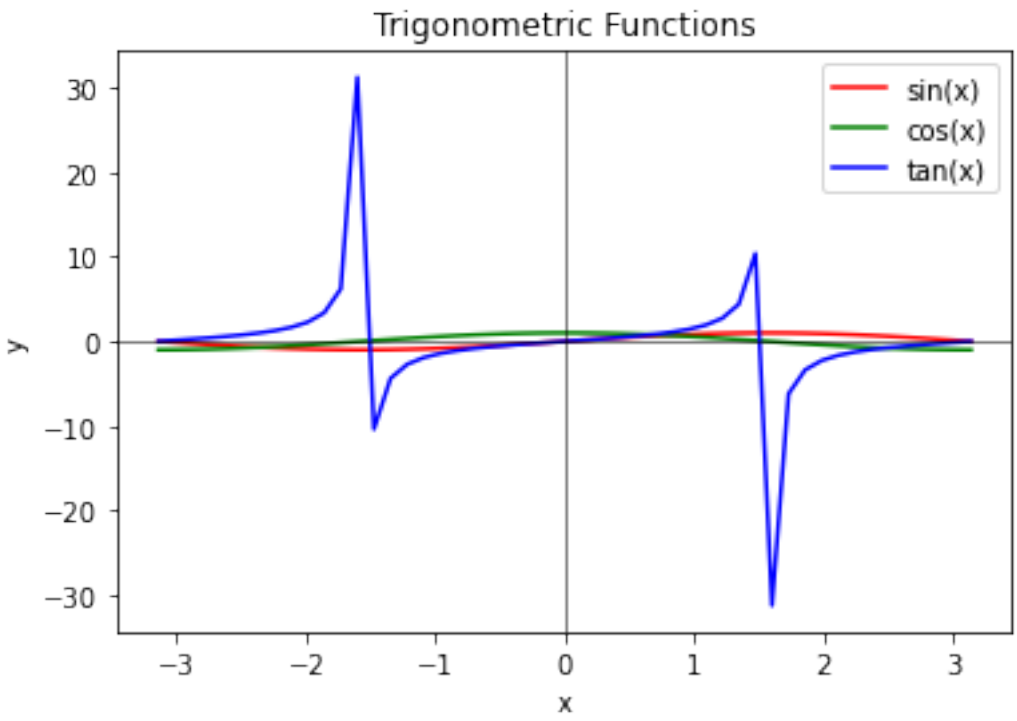
d) Multiple plots in a single graph

November 27, 2021

1 AIM

This is another addition to plotting recap. Here, instead of producing multiple plots as separate graphs, we will produce these plots in a single graph. In this record, we will produce multiple trigonometric functions in a single graph, range and domain.

[29]:
```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-np.pi, np.pi, 50)
plt.plot(x, np.sin(x), color = "red", label = "sin(x)")
plt.plot(x, np.cos(x), color = "green", label = "cos(x)")
plt.plot(x, np.tan(x), color = "blue", label = "tan(x)")

plt.title('Trigonometric Functions')
plt.xlabel('x')
plt.ylabel('y')

plt.axhline(lw = 0.5, color = "black")
plt.axvline(lw = 0.5, color = "black")
plt.legend()
None
```



2 CONCLUSION

Multiple plots in a single graph are an effective way to compare the plots more closely and in a corresponding manner.