STTP
on
“Machine Learning for Non-coders”

Introduction to Basic Python

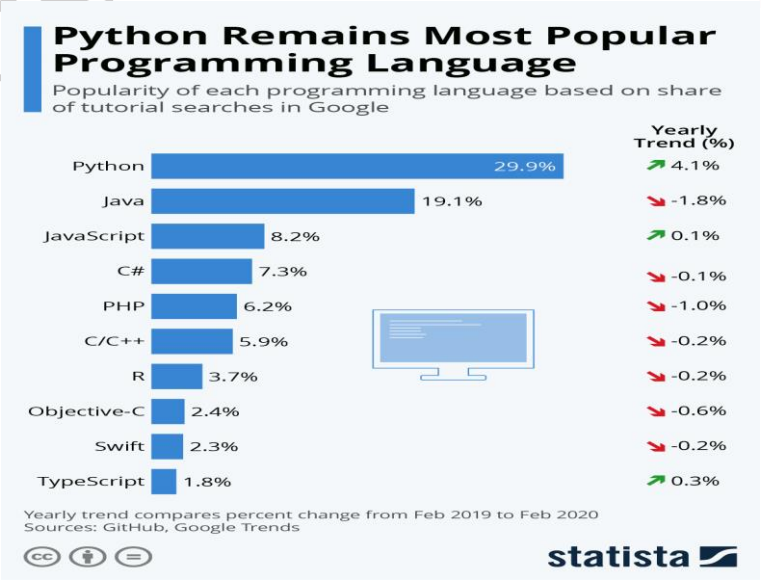
Dr. Ankit Vijayvargiya

RAMAN Lab

Department of Electrical Engineering, MNIT, Jaipur

Brief Introduction:

- Python is a programming language created in 1991 by Guido van Rossum.
- General-purpose language as well as scripting language.
- Interpreted
- Increasingly popular

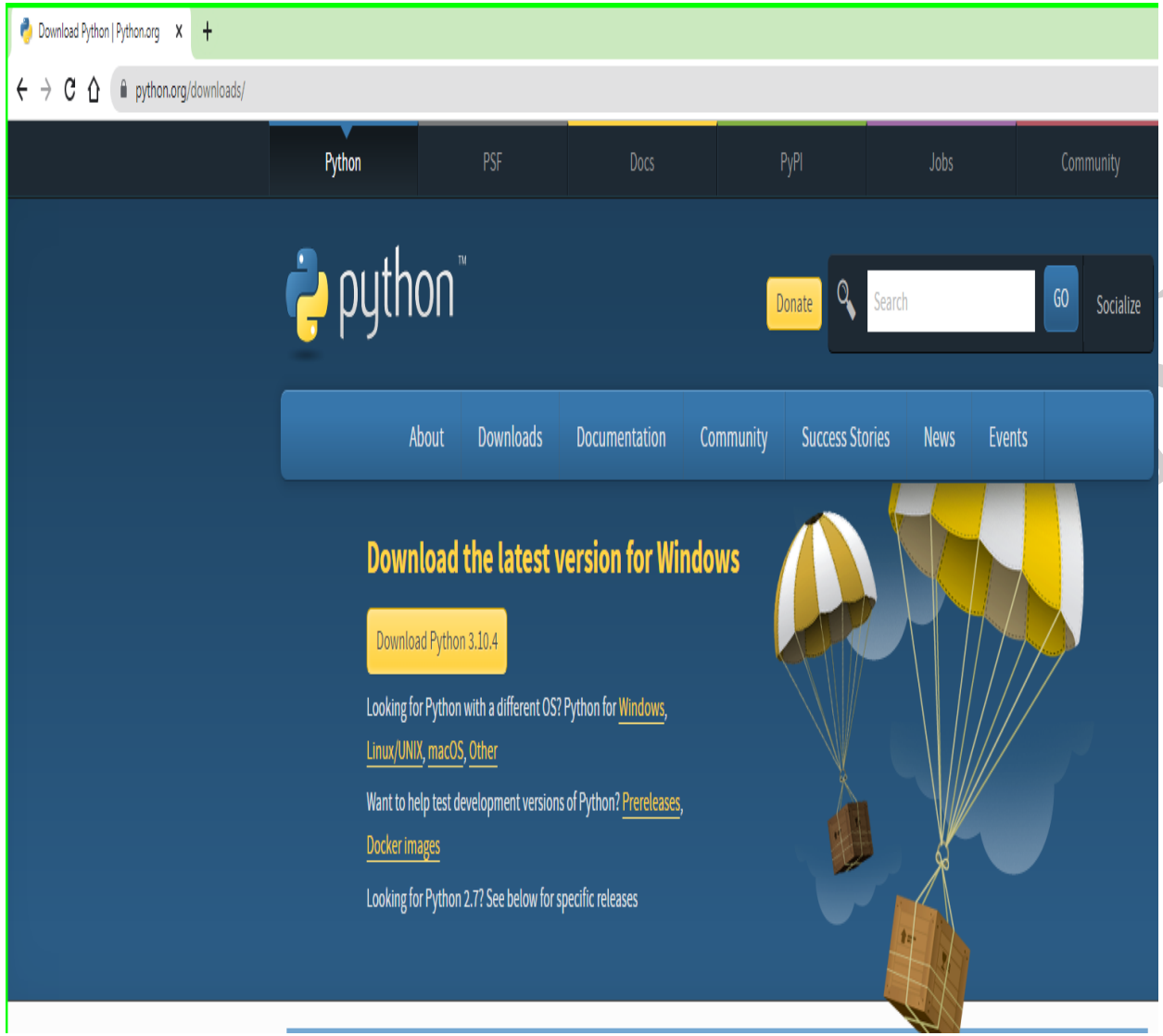


Source: statista.com

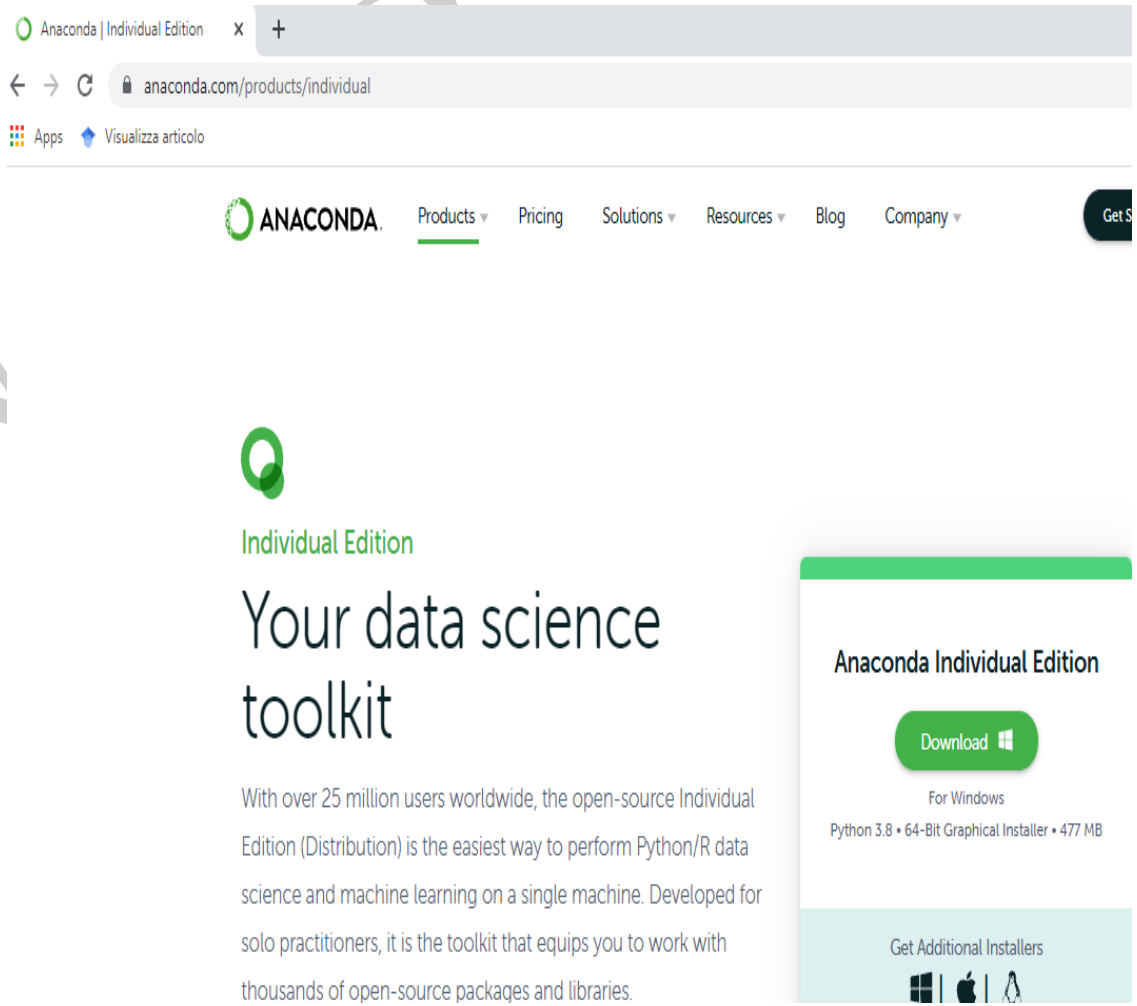
Why Python ????

- Python is Open Source Language.
- It works on a different platform (Windows, Mac, Linux, Raspberry Pi, etc.)
- Python has a syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python has a simple syntax similar to the English language.

Installation of Python

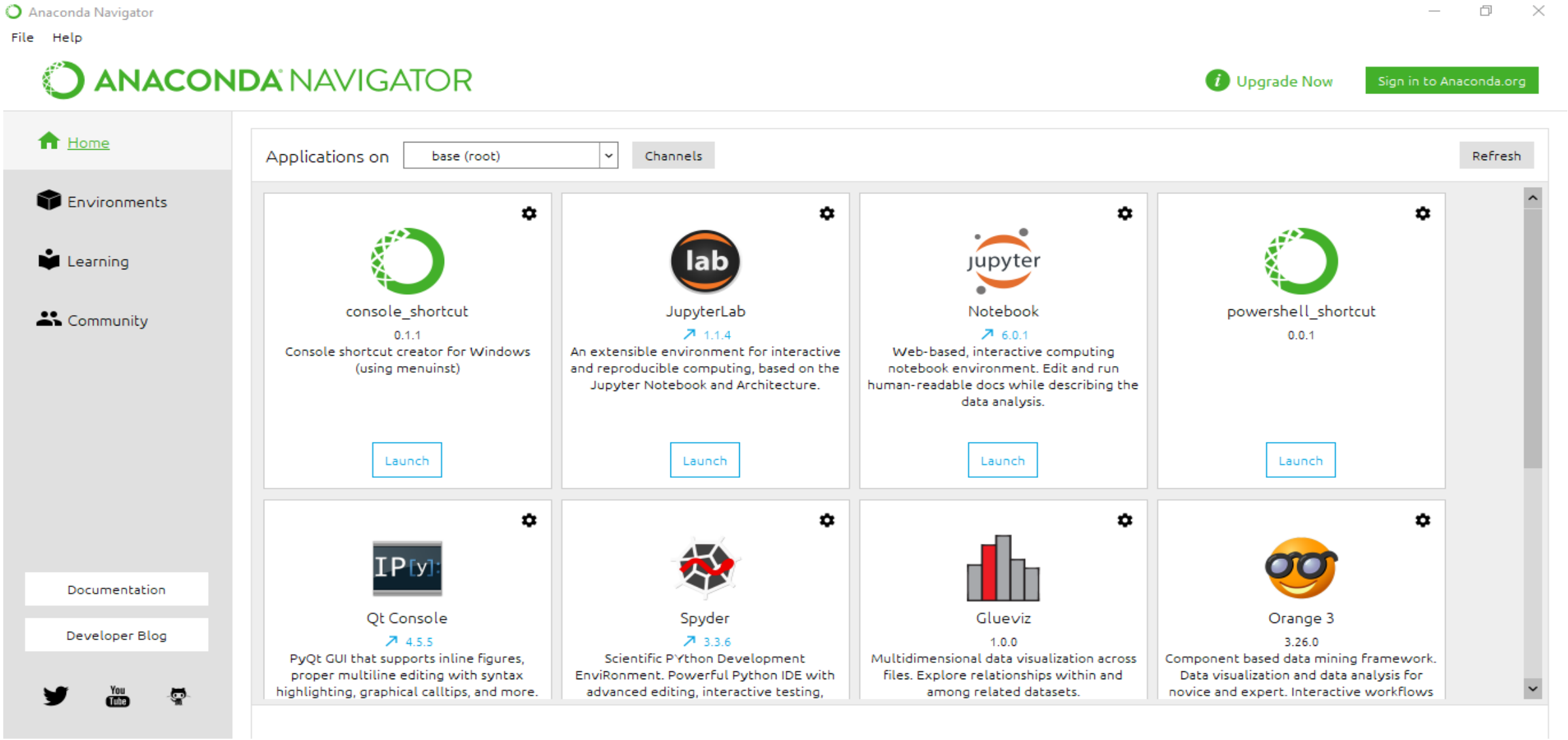


Source: python.org



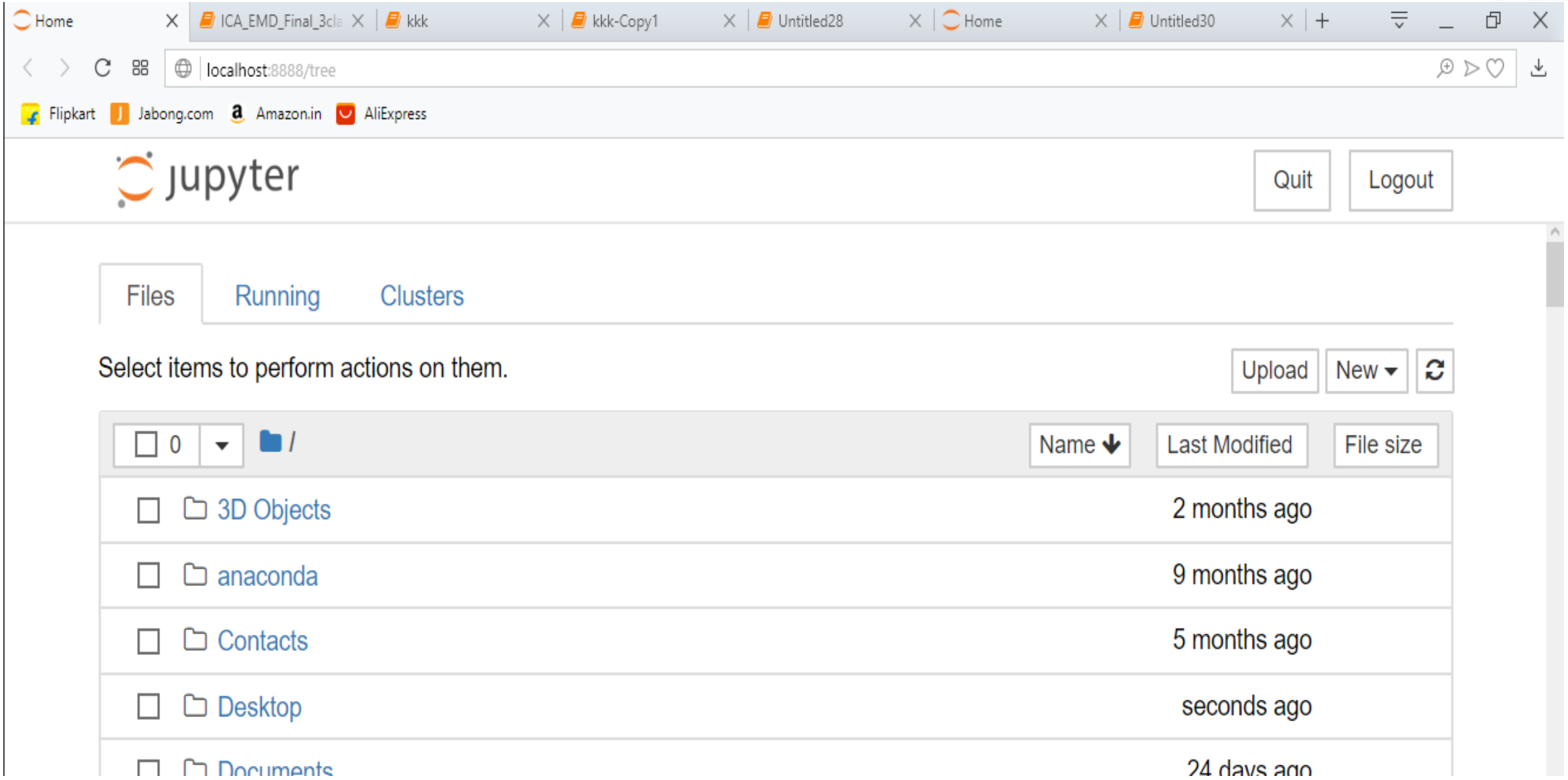
Source: anaconda.com

Python Platform: Anaconda

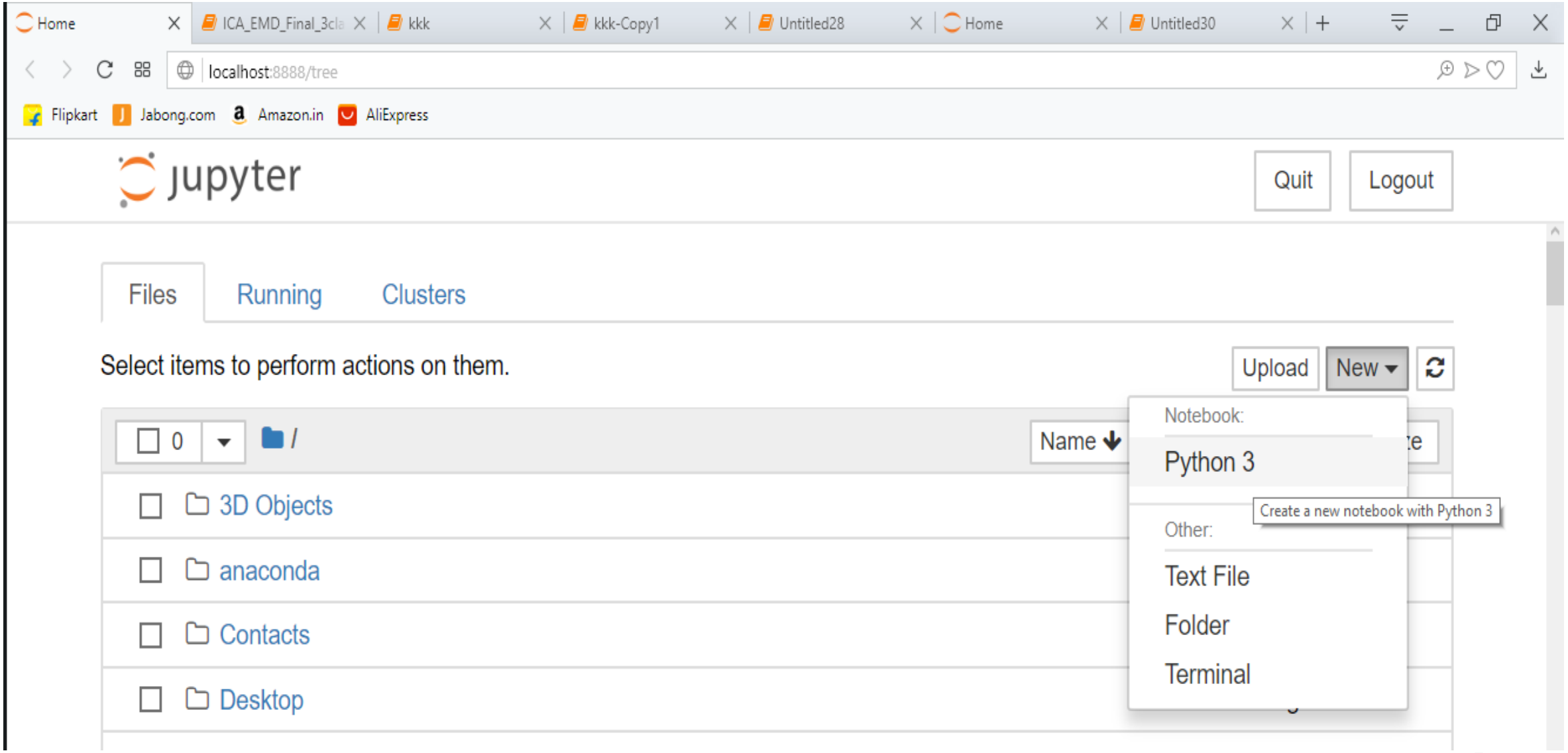


Source: anaconda.com

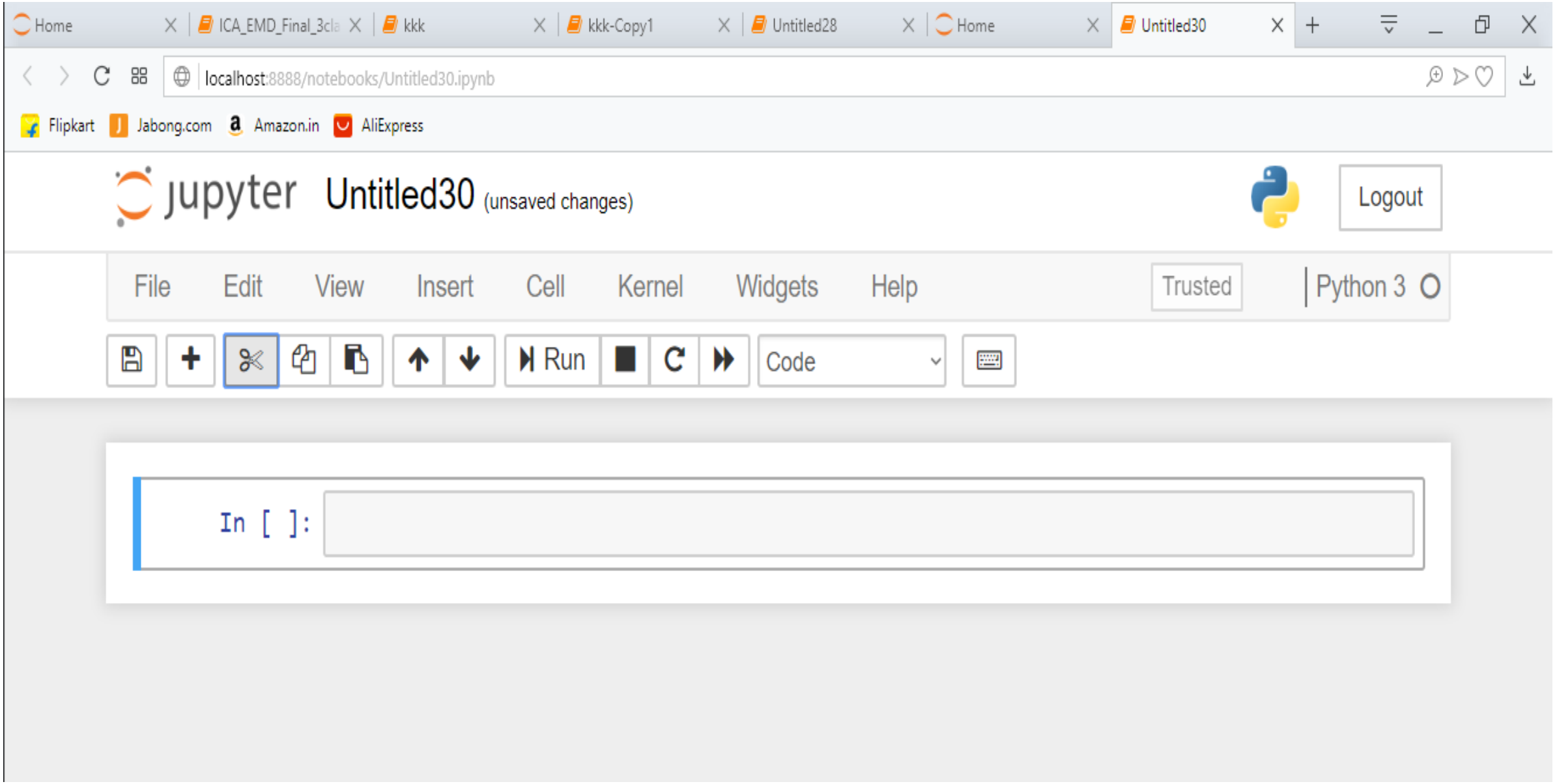
How to write a program in jupyter ???



How to write a program in jupyter ???



How to write a program in jupyter ???



Python vs Other Language :

- On average Python code is smaller than JAVA/C++ codes by 3.5 times.
- Python is interpreted while C/C++, etc. are compiled.
  - **Compiler** : spends a lot of time analysing and processing the program, faster program execution
  - **Interpreter** : relatively little time is spent analysing and processing the program, slower program execution

Python First Program: Print “Hello World”

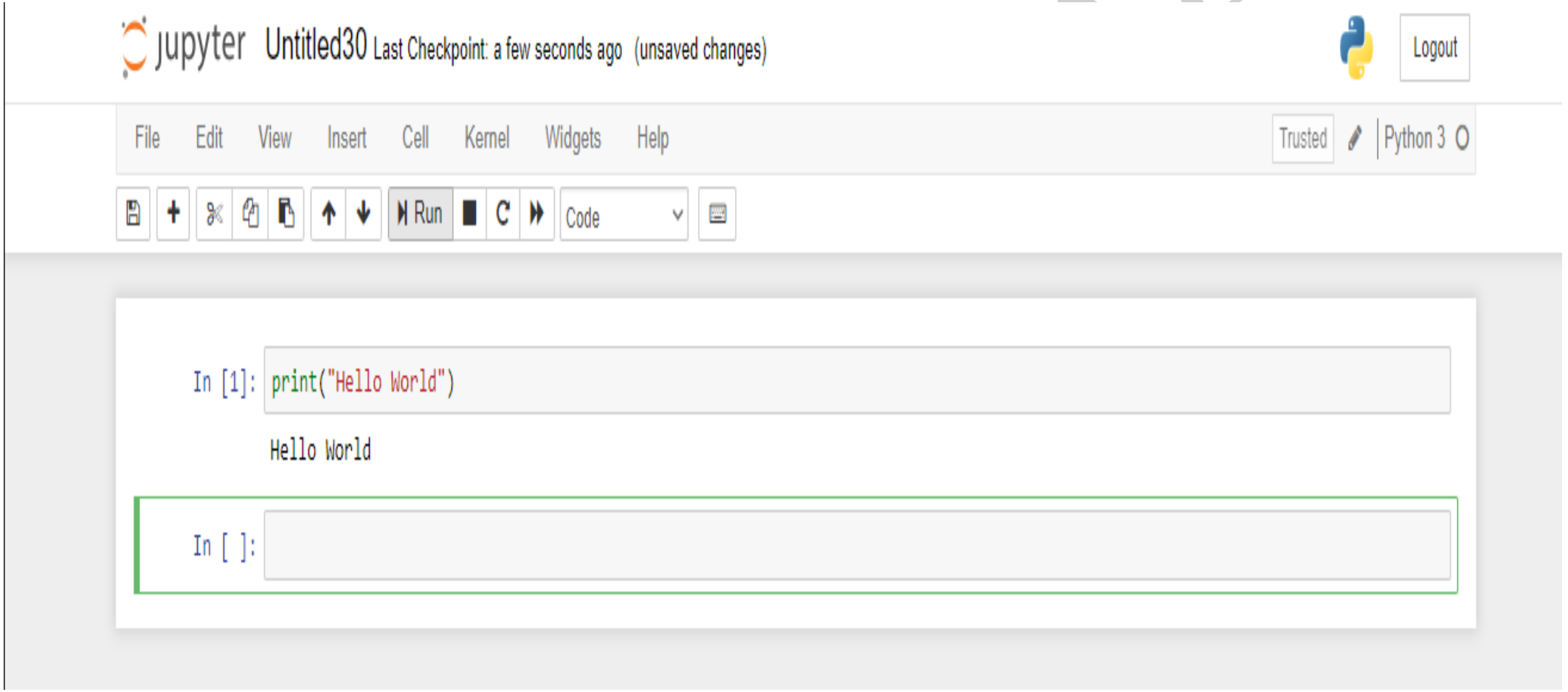
C Code

```
#include <stdio.h>
int main()
{
    printf("Hello World");
    return 0;
}
```

C++ code:

```
#include <iostream>
int main( )
{
        cout << "Hello World";
        return 0;
}
```

Python First Program: Print “Hello World”



Print in Python:

1. Print('write any message')
   or
   Print("write any message")

```
In [1]: print("Hello World")

        Hello World
```

2. Print(Item1 , Item2)

```
In [5]: a=2
        b=3
        print(a,b)

        2 3
```

Print in Python:

3. print('The value of x is {} and y is {}'.format(5,10))

```
In [7]: print('The value of x is {} and y is {}'.format(5,10))

        The value of x is 5 and y is 10
```

4. print("%s %s" %('hello', 'world'))

```
In [8]: print("%s %s" %('hello', 'world'))

        hello world
```

Exercise

Write a program(WAP) to print your name two times.

Exercise

WAP to print the following string :

"Twinkle, twinkle, little star, How I wonder what you are! Up above the world so high, Like a diamond in the sky. Twinkle, twinkle, little star, How I wonder what you are"

Exercise

WAP to print the following string in the given format :

"Twinkle, twinkle, little star, How I wonder what you are! Up above the world so high, Like a diamond in the sky. Twinkle, twinkle, little star, How I wonder what you are"

```
Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky.
Twinkle, twinkle, little star,
How I wonder what you are
```

Exercise

WAP to print the following string in the given format :

"Twinkle, twinkle, little star, How I wonder what you are! Up above the world so high, Like a diamond in the sky. Twinkle, twinkle, little star, How I wonder what you are"

```
Twinkle, twinkle, little star,
        How I wonder what you are!
                Up above the world so high,
                Like a diamond in the sky.
Twinkle, twinkle, little star,
        How I wonder what you are
```

Comments in Python:

- For single line comment hash (#) symbol is used.

```
In [30]: #write a program to print Hello World
         print("Hello World")
```

```
Hello World
```

- For multi-line comments use triple quotes, either ''' or """.

```
In [34]: """write
         a program
         to print
         Hello World"""

         print("Hello World")
```

```
Hello World
```

Python Indentation:

- Python uses indentation instead of braces to determine the scope of expressions.
- All lines must be indented the same amount to be part of the scope (or indented more if part of an inner scope).
- This forces the programmer to use proper indentation since the indentation is part of the program.

```
In [35]: print("My name is")
             print("XYZ")

  File "<ipython-input-35-4f1c46e5ac8a>", line 2
    print("XYZ")
    ^
IndentationError: unexpected indent
```

Variables in Python:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- Reserved words such as break, for, continue, etc. can't be used as variable's names.
- In Python, multiple assignments can be made in a single statement as follows:

```
In [2]: x = 4 # x is of type int
        x = "SHYAM" # x is now of type str
```

```
In [6]: a, b, c = 5, 3.2, "Hello"
        print(a,b,c)

        5 3.2 Hello
```

User Input in Python:

```
In [*]: print("whats your Name")
        name= input(">")
```

```
whats your Name
> Shyam
```

```
In [*]: print("What year were you born?")
        year = int(input(">"))
```

```
In [*]: print("Hi {}, you are {} year old ".format(name,year))
```

User Input in Python:

```
In [1]: print("whats your Name")
        name= input(">")

        whats your Name
        >Shyam
```

```
In [*]: print("What year were you born?")
        year = int(input(">"))

        What year were you born?

        > 1947
```

```
In [*]: print("Hi {}, you are {} year old ".format(name,year))
```

User Input in Python:

```
In [1]: print("whats your Name")
        name= input(">")

        whats your Name
        >Shyam
```

```
In [2]: print("What year were you born?")
        year = int(input(">"))

        What year were you born?
        >1947
```

```
In [3]: print("Hi {}, you are {} year old ".format(name,year))

        Hi Shyam, you are 1947 year old
```

Exercise

Write a Python program that accepts the radius of a circle from the user and computes the area.

Exercise

Write a Python program that accepts the user's first and last name and prints them in reverse order.

Control Flow in Python:

- The if else statement is used in python for decision making.

```
if test expression:
        body of if
else:
        body of else
```

- To test more than one condition following syntax is used-

```
if test expression:
        body of if
elif test expression:
        body of elif
else:
        body of else
```

Control Flow in Python:

```
print("Enter the Value of X")
X=int(input(">"))

if X >= 5:
    print("The value is greater than 5")
else:
    print("The value is Less than 5")
```

```
Enter the Value of X
>6
The value is greater than 5
```

```
print("Enter the Value of X")
X=int(input(">"))

if X > 5:
    print("The value is greater than 5")
elif X<5:
    print("The value is Less than 5")
else:
    print("The value is equal to 5")
```

```
Enter the Value of X
>3
The value is Less than 5
```

Python Basic Operators:

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Bitwise operators
- Membership Operators
- Identity Operators

Arithmetic Operators:

Arithmetic operators are used with numerc values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Arithmetic Operators:

```
a=int(input("Input a"))
b=int(input("Input b"))
c=a+b
print(c)
```

```
Input a-2
Input b2
0
```

```
a=int(input("Input a"))
b=int(input("Input b"))
c=a-b
print(c)
```

```
Input a7
Input b5
2
```

```
a=int(input("Input a"))
b=int(input("Input b"))
c=a*b
print(c)
```

```
Input a2
Input b3
6
```

Arithmetic Operators:

```
a=int(input("Input a"))
b=int(input("Input b"))
c=a/b
print(c)
```

```
Input a5
Input b2
2.5
```

```
a=int(input("Input a"))
b=int(input("Input b"))
c=a%b
print(c)
```

```
Input a5
Input b2
1
```

```
a=int(input("Input a"))
b=int(input("Input b"))
c=a//b
print(c)
```

```
Input a5
Input b2
2
```

Comparison Operators:

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Comparison Operators:

```
a=int(input("Input a"))
b=int(input("Input b"))
if a==b:
    print(a+b)
else:
    print(a-b)
```

```
Input a5
Input b3
2
```

```
a=int(input("Input a"))
b=int(input("Input b"))
if a>b:
    print(a+b)
else:
    print(a-b)
```

```
Input a3
Input b2
5
```

```
a=int(input("Input a"))
b=int(input("Input b"))
if a!=b:
    print(a+b)
else:
    print(a-b)
```

```
Input a5
Input b3
8
```

```
a=int(input("Input a"))
b=int(input("Input b"))
if a<b:
    print(a+b)
else:
    print(a-b)
```

```
Input a3
Input b2
1
```

Comparison Operators:

```
a=int(input("Input a"))
b=int(input("Input b"))
if a>=b:
    print(a+b)
else:
    print(a-b)
```

```
Input a3
Input b3
6
```

```
a=int(input("Input a"))
b=int(input("Input b"))
if a<=b:
    print(a+b)
else:
    print(a-b)
```

```
Input a3
Input b2
1
```

Logical Operators:

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Logical Operators:

```
a=int(input("Input a"))
b=int(input("Input b"))
if a>5 and b<5:
    print(a+b)
else:
    print(a-b)
```

```
Input a9
Input b6
3
```

```
a=int(input("Input a"))
b=int(input("Input b"))
if a>5 or b<5:
    print(a+b)
else:
    print(a-b)
```

```
Input a9
Input b6
15
```

```
a=int(input("Input a"))
b=int(input("Input b"))
if not(a>5 or b<5):
    print(a+b)
else:
    print(a-b)
```

```
Input a9
Input b6
3
```

Bitwise Operators:

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
~	Bitwise NOT
^	Bitwise XOR
>>	Bitwise right shift
<<	Bitwise left shift

Membership Operators:

- in and not in are the membership operators in Python. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

```
In [20]: x = 'Hello world'
         print('h' in x)

         False

In [19]: print('H' in x)

         True

In [21]: print('K' not in x)

         True
```

Identity Operators:

- is and is not are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory.

```
In [22]: x = ["apple", "banana"]
         y = ["apple", "banana"]
         z = x
         print(x is z)

         True

In [23]: print(x is y)

         False
```

Basic Datatypes:

- Integers (default for numbers)

  Z=5

- Floats

  Z=5.1

- Strings : strings are enclosed within either single quotes or (‘ ‘) or double quotes (“ “).

  Z = ‘python’

  Z = “python”

Python List:

List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.

```
# empty list
        mylist = []
# list of integers
        mylist = [1, 2, 3]
# list with mixed datatypes
        mylist = [1, "Hello", 3.4]
# nested list
        mylist = ["mouse", [8, 4, 6], ['a']]
```

Accessing List Element:

```
mylist = [1, 2 , 3, 4]
```

```
print(mylist[0])
```

```
1
```

```
mylist = [1, "Hello", 3.4]
```

```
print(mylist[1])
```

```
Hello
```

```
print(mylist[1][1])
```

```
e
```

```
mylist = ["mouse", [8, 4, 6], ['a']]
```

```
print(mylist[1][2])
```

```
6
```

Slicing Operator for Python List:

```
mylist=['a','b','c','d','e','f','g','h']
```

```
# print element 3rd to 5th
print(mylist[2:5])
```

```
['c', 'd', 'e']
```

```
# print element begining to 4th
print(mylist[:4])
```

```
['a', 'b', 'c', 'd']
```

```
# print element 4th to end
print(mylist[3:])
```

```
['d', 'e', 'f', 'g', 'h']
```

```
print(mylist[:-4])
```

```
['a', 'b', 'c', 'd']
```

```
print(mylist[-4:])
```

```
['e', 'f', 'g', 'h']
```

Add Element to Python List:

1. append:

```
In [27]: x= [1,'Shyam',25]

In [28]: print(x)
         [1, 'Shyam', 25]

In [29]: x.append("Male")

In [30]: print(x)
         [1, 'Shyam', 25, 'Male']
```

```
In [1]: x= [1,'Shyam',25]

In [2]: print(x)
        [1, 'Shyam', 25]

In [3]: y=['Male', 25000]

In [4]: print(y)
        ['Male', 25000]

In [5]: x.append(y)
        print(x)
        [1, 'Shyam', 25, ['Male', 25000]]
```

Add Element to Python List:

2. insert:

```
In [1]: x= [1,'Shyam',25]

In [2]: print(x)
        [1, 'Shyam', 25]

In [3]: y=['Male', 25000]

In [4]: print(y)
        ['Male', 25000]

In [5]: x.insert(3,y)

In [6]: print(x)
        [1, 'Shyam', 25, ['Male', 25000]]
```

3: extend

```
In [1]: x= [1,'Shyam',25]

In [2]: print(x)
        [1, 'Shyam', 25]

In [3]: y=['Male', 25000]

In [4]: print(y)
        ['Male', 25000]

In [5]: x.extend(y)

In [6]: print(x)
        [1, 'Shyam', 25, 'Male', 25000]
```

Add Element to Python List:

4. + operator:

```
In [1]: x= [1,'Shyam',25]

In [2]: print(x)
        [1, 'Shyam', 25]

In [3]: y=['Male', 25000]

In [4]: print(y)
        ['Male', 25000]

In [5]: print(x+y)
        [1, 'Shyam', 25, 'Male', 25000]
```

```
In [3]: x= [1,'Shyam',25]

In [5]: print(x + ['hi']*3)
        [1, 'Shyam', 25, 'hi', 'hi', 'hi']
```

Delete Element from Python List:

```
In [9]: mylist=['a','b','c','d','e','f','g','h']

In [10]: #del one value
         del mylist[2]

In [11]: print(mylist)

['a', 'b', 'd', 'e', 'f', 'g', 'h']
```

```
In [9]: mylist=['a','b','c','d','e','f','g','h']

In [12]: #del multiple value
         del mylist[2:4]

In [13]: print(mylist)

['a', 'b', 'f', 'g', 'h']
```

```
In [9]: mylist=['a','b','c','d','e','f','g','h']

In [14]: #del entire list
         del mylist

In [15]: print(mylist)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-15-aca29853c9f0> in <module>
----> 1 print(mylist)

NameError: name 'mylist' is not defined
```

Python Tuple:

- A tuple is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas in a list, elements can be changed.
- Advantages of Tuple over List
  - Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.

Creating a Tuple:

```
# Empty Tuple
my_tuple=()
print(my_tuple)
```

```
()
```

```
# Tuple having integer
my_tuple=(1,2,3)
print(my_tuple)
```

```
(1, 2, 3)
```

```
# Tuple with mixed datatypes
my_tuple= (1, "Hello" , 3.5)
print(my_tuple)
```

```
(1, 'Hello', 3.5)
```

```
# Nested Tuple
my_tuple= ("Hello" , [1,2,3,4] , (1,2,3))
print(my_tuple)
```

```
('Hello', [1, 2, 3, 4], (1, 2, 3))
```

Assessing Element in a Tuple:

```
: my_tuple=('a','b','c','d','e','f','g','h')

: print(my_tuple[0])
```

```
a
```

```
: print(my_tuple[5])
```

```
f
```

```
: print(my_tuple[8])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-29-82cd0f8a39f9> in <module>
----> 1 print(my_tuple[8])

IndexError: tuple index out of range
```

```
: print(my_tuple[1:3])
```

```
('b', 'c')
```

Changing a Tuple:

- Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples

```
my_tuple1=(1,2,3)
my_tuple1[0]="hello"
print(my_tuple1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-37-cc49132ecbe0> in <module>
      1 my_tuple1=(1,2,3)
----> 2 my_tuple1[0]="hello"
      3 print(my_tuple1)

TypeError: 'tuple' object does not support item assignment
```

```
my_list1=[1,2,3]
my_list1[0]="hello"
print(my_list1)
```

```
['hello', 2, 3]
```

Built-in function with Tuple and list:

Creating List and Tuple:

```
my_tuple1=(1,2,3,4,5,6)
print(my_tuple1)
```

```
(1, 2, 3, 4, 5, 6)
```

```
my_list1=['a','b','c','d']
print(my_list1)
```

```
['a', 'b', 'c', 'd']
```

Maximum Value (max):

```
print(max(my_tuple1))
print(max(my_list1))
```

```
6
d
```

Length (len):

```
print(len(my_tuple1))
print(len(my_list1))
```

```
6
4
```

Minimum Value (min):

```
print(min(my_tuple1))
print(min(my_list1))
```

```
1
a
```

Built in function with Tuple and list:

Sorting (sorted):

```
my_tuple1=(1,8,3,11,17,6)
my_list1=['a','d','g','s','p','m']

print(sorted(my_tuple1))
print(sorted(my_list1))
```

```
[1, 3, 6, 8, 11, 17]
['a', 'd', 'g', 'm', 'p', 's']
```

Summation (sum):

```
print(sum(my_tuple1))
```

```
46
```

```
print(sum(my_list1))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-51-e36b49e13c83> in <module>
----> 1 print(sum(my_list1))

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Python Dictionaries:

- Dictionary is an unordered collection of key-value pairs. It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data.

Creating a Dictionaries:

```
# Empty Dictionary
my_dict={}
```

```
# Dictionary with Integer Keys
my_dict={1:"Apple", 2:"Ball"}
print(my_dict)
```

```
{1: 'Apple', 2: 'Ball'}
```

```
# Dictionary with Mixed Keys
my_dict={"Name":"Shyam", "Age":25}
print(my_dict)
```

```
{'Name': 'Shyam', 'Age': 25}
```

Access Element from a Dictionary:

```
# Dictionary with Mixed Keys
my_dict={"Name":"Shyam", "Age":25}
print(my_dict)
```

```
{'Name': 'Shyam', 'Age': 25}
```

```
print(my_dict["Name"])
```

```
Shyam
```

Change /Add and Delete in a Dictionaries:

```
In [16]: my_dict["Age"]=28
         print(my_dict)
```

```
{'Name': 'Shyam', 'Age': 28}
```

```
In [17]: my_dict["Gender"]="Male"
         print(my_dict)
```

```
{'Name': 'Shyam', 'Age': 28, 'Gender': 'Male'}
```

```
In [18]: del my_dict["Age"]
         print(my_dict)
```

```
{'Name': 'Shyam', 'Gender': 'Male'}
```

Python Nested Dictionary:

```
In [1]: People= {1:{"Name":"Shyam", "Age":25, "Sex": "Male"},
                 2:{"Name":"Ram"  , "Age":15, "Sex": "Male"}}
```

```
In [2]: print(People)
```

```
{1: {'Name': 'Shyam', 'Age': 25, 'Sex': 'Male'}, 2: {'Name': 'Ram', 'Age': 15, 'Sex': 'Male'}}
```

```
In [3]: print(People[1])
```

```
{'Name': 'Shyam', 'Age': 25, 'Sex': 'Male'}
```

```
In [4]: print(People[1]["Age"])
```

```
25
```

Python Nested Dictionary:

```
In [5]: # Add Item in Dictionaries
People["Salary"]=2000
print(People)
```

```
{1: {'Name': 'Shyam', 'Age': 25, 'Sex': 'Male'}, 2: {'Name': 'Ram', 'Age': 15, 'Sex': 'Male'}, 'Salary': 2000}
```

```
In [6]: People[1]["Salary"]=2000
People[2]["Salary"]=1000
print(People)
```

```
{1: {'Name': 'Shyam', 'Age': 25, 'Sex': 'Male', 'Salary': 2000}, 2: {'Name': 'Ram', 'Age': 15, 'Sex': 'Male', 'Salary': 1000},
'Salary': 2000}
```

```
In [7]: del People[1]["Salary"]
print(People)
```

```
{1: {'Name': 'Shyam', 'Age': 25, 'Sex': 'Male'}, 2: {'Name': 'Ram', 'Age': 15, 'Sex': 'Male', 'Salary': 1000}, 'Salary': 2000}
```

While Loop:

- while loop is used to execute a block of statements repeatedly until a given a condition is satisfied.
- when the condition becomes false, the line immediately after the loop in program is executed.

Syntax :

```
while expression:
    Body of while
```

:

```
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
```

```
Hello Geek
Hello Geek
Hello Geek
```

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).

```
: # Iterating over a list
print("List Iteration")
l = ["geeks", "for", "geeks"]
for i in l:
    print(i)
```

```
List Iteration
geeks
for
geeks
```

```
: # Iterating over a tuple (immutable)
print("\nTuple Iteration")
t = ("geeks", "for", "geeks")
for i in t:
    print(i)
```

```
Tuple Iteration
geeks
for
geeks
```

For Loop:

```
for i in range(0,5):
    for j in range(0,i+1):
        print("*",end=" ")
    print("\n")
```

```
*

* *

* * *

* * * *

* * * * *
```

break:

- The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.
- If break statement is inside a nested loop, break will terminate the inner most loop.

```
for i in range(0,5):
    print("enter the value of x")
    x=int(input(">"))
    if (x>9):
        break
print("The End")
```

```
enter the value of x
>10
The End
```

Continue:

- The continue statement of is used to skip the rest of the code inside the loop for the current iteration only.
- Loop does not terminate but continues on with next iteration.

```
: for i in range(0,5):
      print("enter the value of x")
      x=int(input(">"))
      if (x>9):
          continue
      print("The value is less then 9")
print("The End")
```

```
enter the value of x
>3
The value is less then 9
enter the value of x
>10
enter the value of x
>
```

Functions:

- In python, function is a group of related statements that perform a specific task.

  Syntax:

  def functionname(parameters):

  statements

- Keyword def marks the start of function header.
- Parameters through which we pass value to a function. They are optional.
- A colon (:) to mark the function of header.
- An optional return

```
: def square(data):
      y=data*data
      return(y)

  print("enter the value of x")
  x=int(input(">"))
  z=square(x)
  print(z)
```

```
enter the value of x
>3
9
```

Python Libraries:

Python library is a collection of functions and methods that allows you to perform many actions without writing your code.

Many popular Python toolboxes/libraries:

- Pandas
- NumPy
- SciPy
- SciKit-Learn
- Keras

Visualization libraries

- Matplotlib
- Seaborn

and many more …

How to install Python Libraries??

- Pip is a package-management system used to install and manage software packages written in Python.

```
!pip install pip
!pip install pip –upgrade
```

- Install python libraries using pip.

```
!pip install pandas
!pip install numpy
```

How to Load Python Libraries??

In [ ]:

```
#Import Python Libraries
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import seaborn as sns
```

Pandas library

- **Pa**nel **Da**ta **S**ystem
- Pandas is a popular Python library for **data analysis**. It is not directly related to Machine Learning.
- **Dataset** must be prepared before training. In this case, Pandas developed specifically for **data extraction and preparation.**
- It provides high-level data structures and wide variety tools for data analysis. It provides many inbuilt methods for grouping, combining and filtering data.

How to Read data using pandas??

- Read csv file

```
df = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/Salaries.csv")
```

- Read Excel file

```
df = pd.read_excel("http://rcs.bu.edu/examples/python/data_analysis/Salaries.xlsx")
```

Exploring Data frames:

```
In [6]: import pandas as pd
```

```
In [9]: pima= pd.read_excel(r'C:\Users\Ankit\Desktop\iris.xlsx')
```

```
In [8]: pima.head()
```

Out[8]:

	Id	Sepal Length	Sepal Width	Petal Length	Petal Width	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Numpy library

- It is for large **multi-dimensional array** and **matrix processing**.
- It is very useful for fundamental scientific computations in Machine Learning.

```
import numpy as np

# Creating a rank 1 Array
arr = np.array([1, 2, 3])
print("Array with Rank 1: \n",arr)

# Creating a rank 2 Array
arr = np.array([[1, 2, 3],
                [4, 5, 6]])
print("Array with Rank 2: \n", arr)

# Creating an array from tuple
arr = np.array((1, 3, 2))
print("\nArray created using "
      "passed tuple:\n", arr)
```

```
Array with Rank 1:
 [1 2 3]
Array with Rank 2:
 [[1 2 3]
 [4 5 6]]
```

Numpy library

```
In [8]: import numpy as np
```

```
In [10]: mean=np.mean(X)
         print(mean)
```

```
5.843333333333335
```

```
In [12]: maximum=np.max(X)
         print(maximum)
```

```
7.9
```

```
In [13]: minimum=np.min(X)
         print(minimum)
```

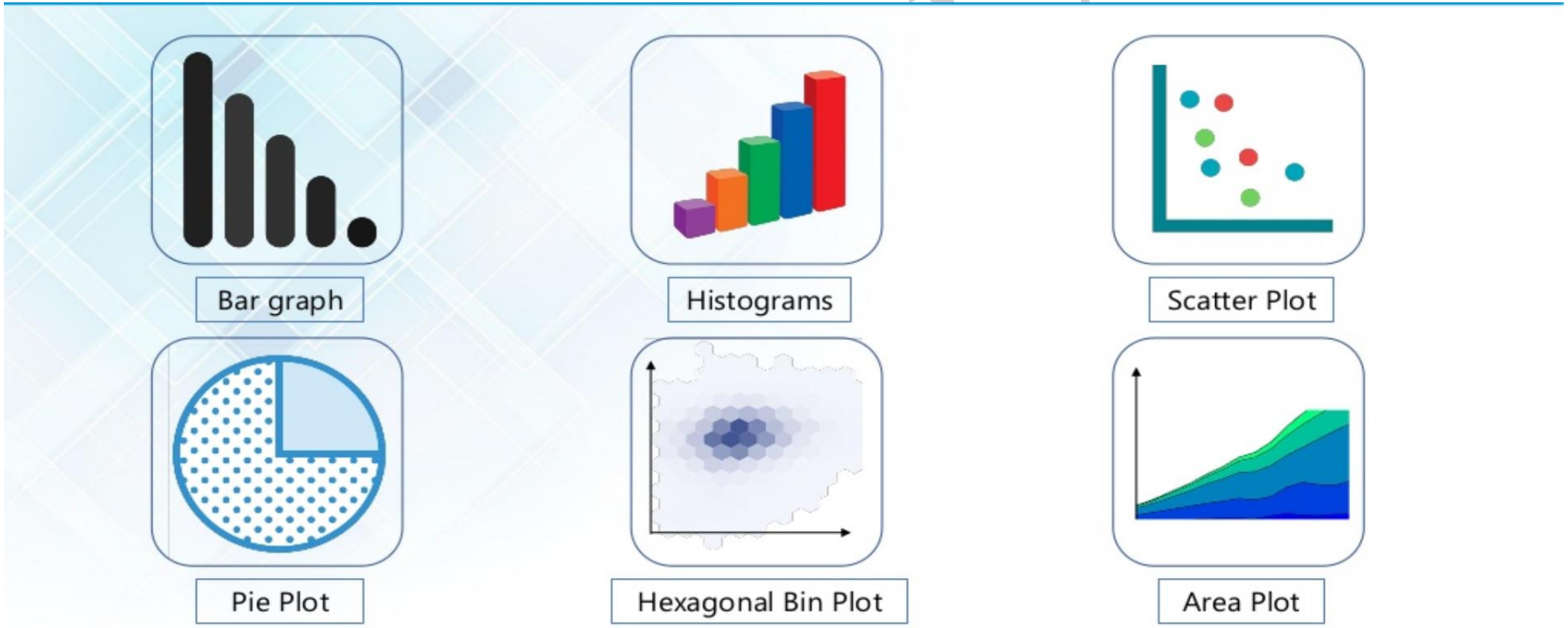
```
4.3
```

Matplotlib:

- Matplotlib is a very popular Python library for **data visualization**.
- Like Pandas, it is not directly related to Machine Learning.
- It particularly comes in handy when a programmer wants to visualize the patterns in the data.
- It is a 2D plotting library used for creating 2D graphs and plots.
- A module named pyplot makes it easy for programmers for plotting as it provides features to control line styles, font properties, formatting axes, etc.
- It provides various kinds of graphs and plots for data visualization, viz., histogram, error charts, bar chats, etc,

Matplotlib:

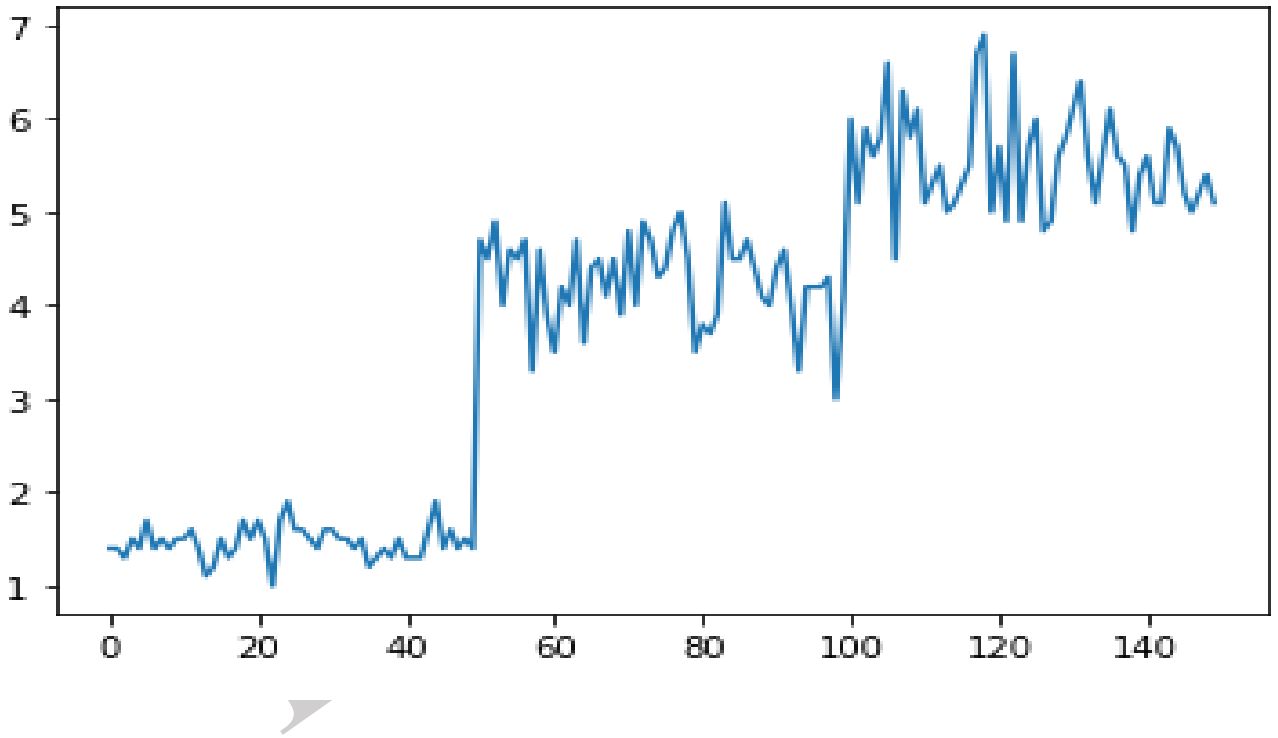
- It provides various kinds of graphs and plots for data visualization, viz., histogram, error charts, bar chats, etc,



Line Plot

```
plt.plot(data["Petal_length"])
```

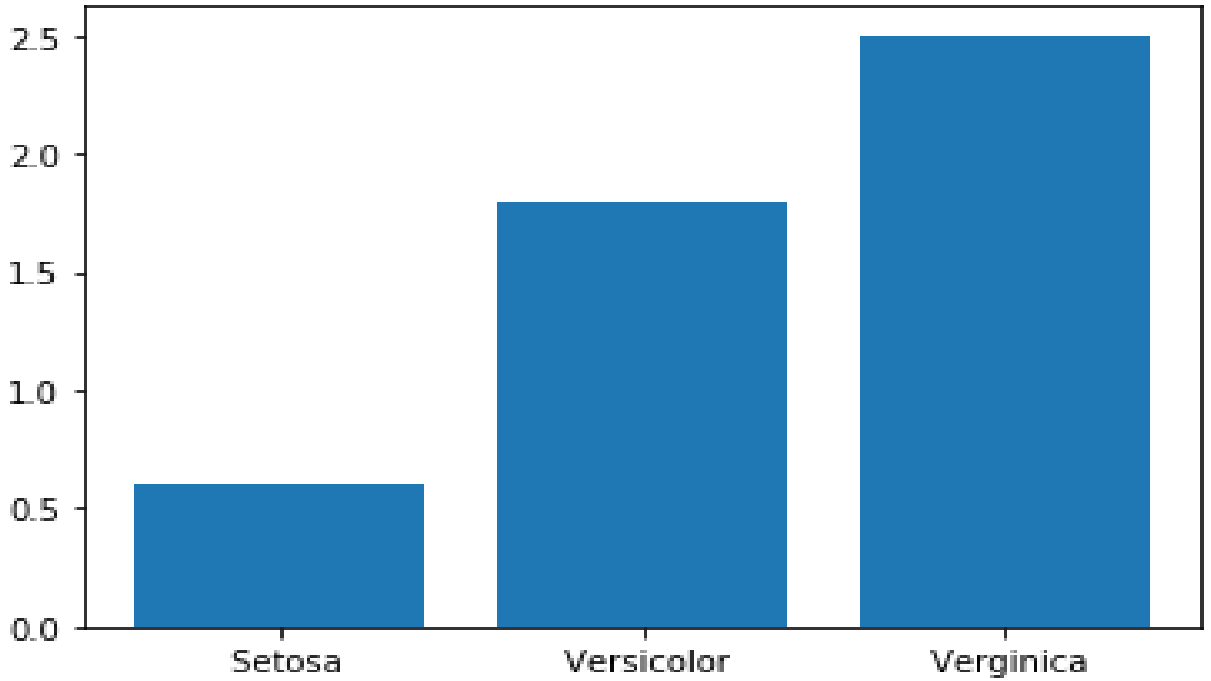
```
[<matplotlib.lines.Line2D at 0x204fd27f748>]
```



Bar Plot

```
plt.bar(data["Species_name"], data["Petal_width"])
```

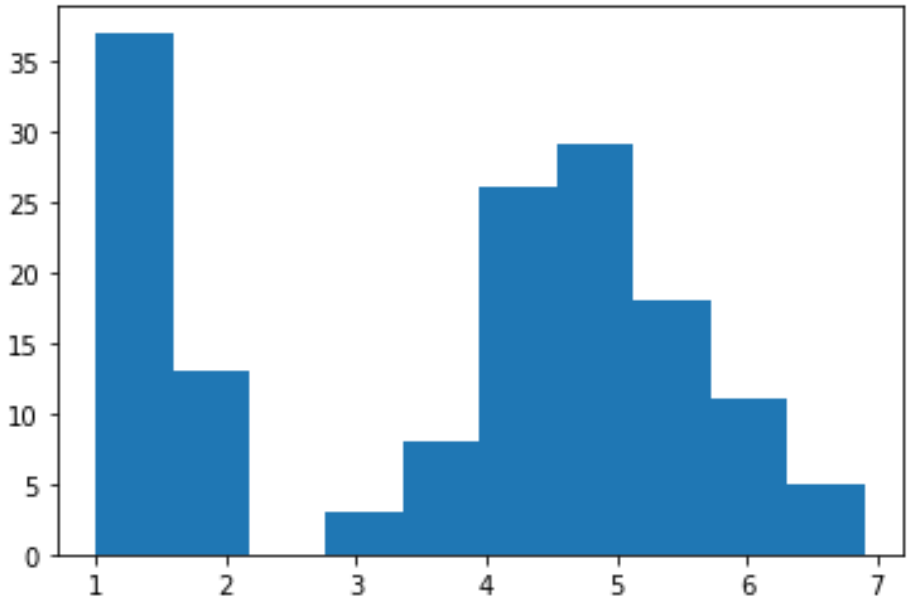
```
<BarContainer object of 150 artists>
```



Histogram Plot

```
plt.hist(data["Petal_length"])
```

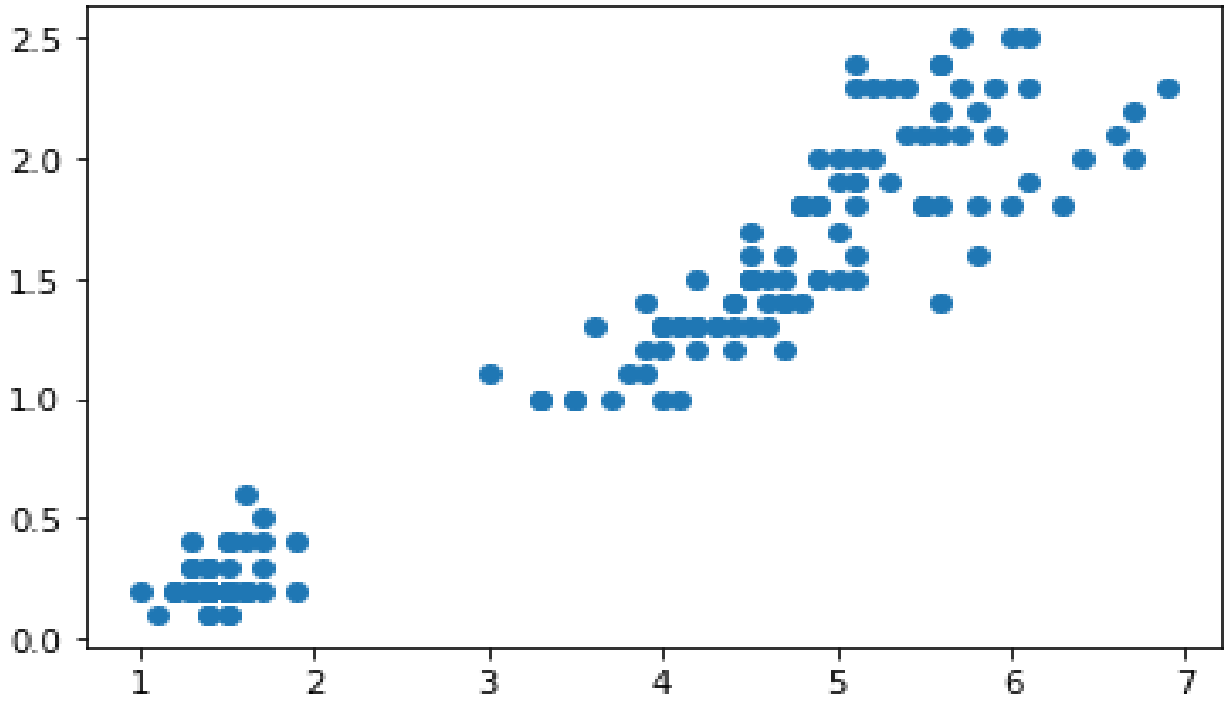
```
(array([37., 13.,  0.,  3.,  8., 26., 29., 18., 11.,  5.]),
 array([1.  , 1.59, 2.18, 2.77, 3.36, 3.95, 4.54, 5.13, 5.72, 6.31, 6.9 ]),
 <a list of 10 Patch objects>)
```



Scatter Plot

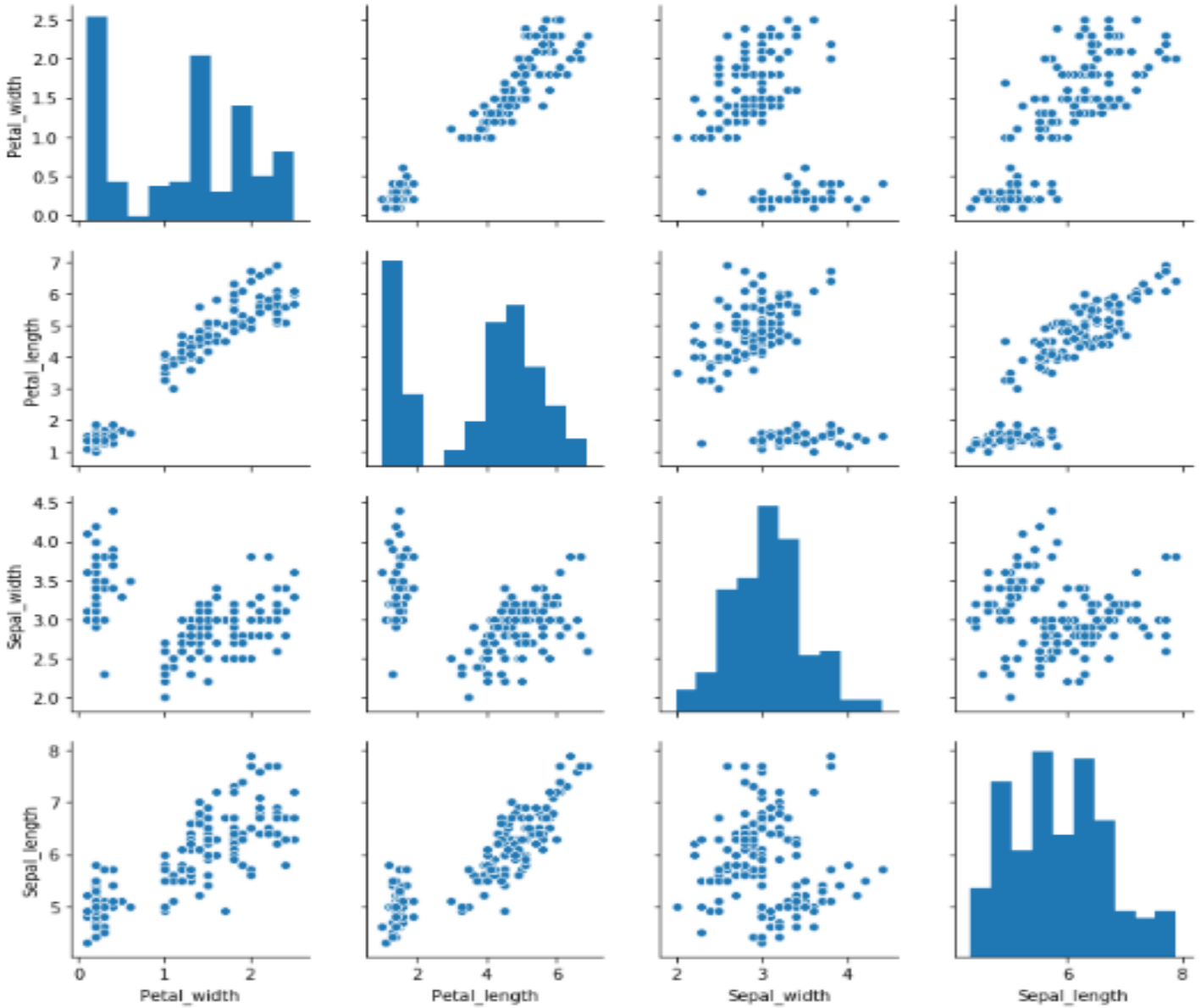
```
plt.scatter(data["Petal_length"], data["Petal_width"])
```

```
<matplotlib.collections.PathCollection at 0x204feea09c8>
```



Pairplot using seaborn:

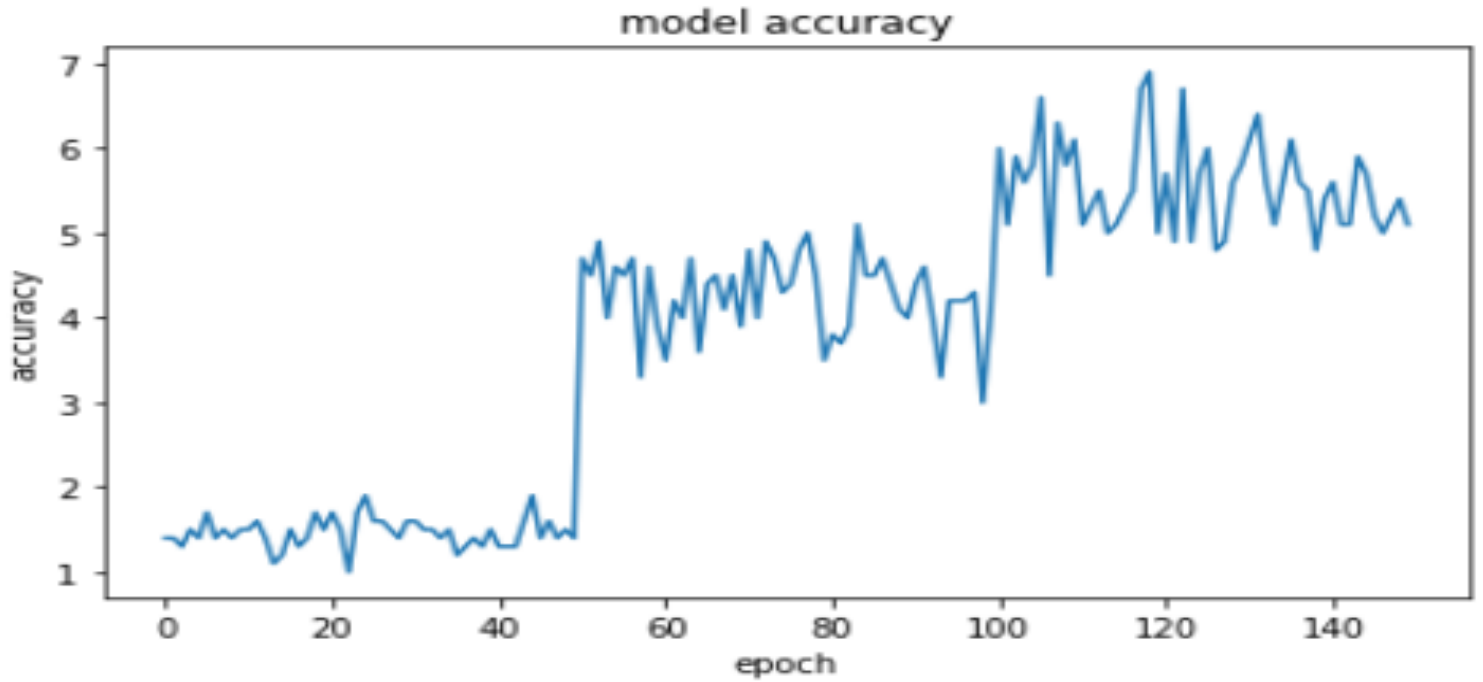
```
In [47]: import seaborn as sns
         del(data["Species_No"])
         sns.pairplot(data)
```



Labels/Size of figure:

```
In [52]: fig = plt.figure(figsize=(7,4))
         plt.plot(data["Petal_length"])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
```

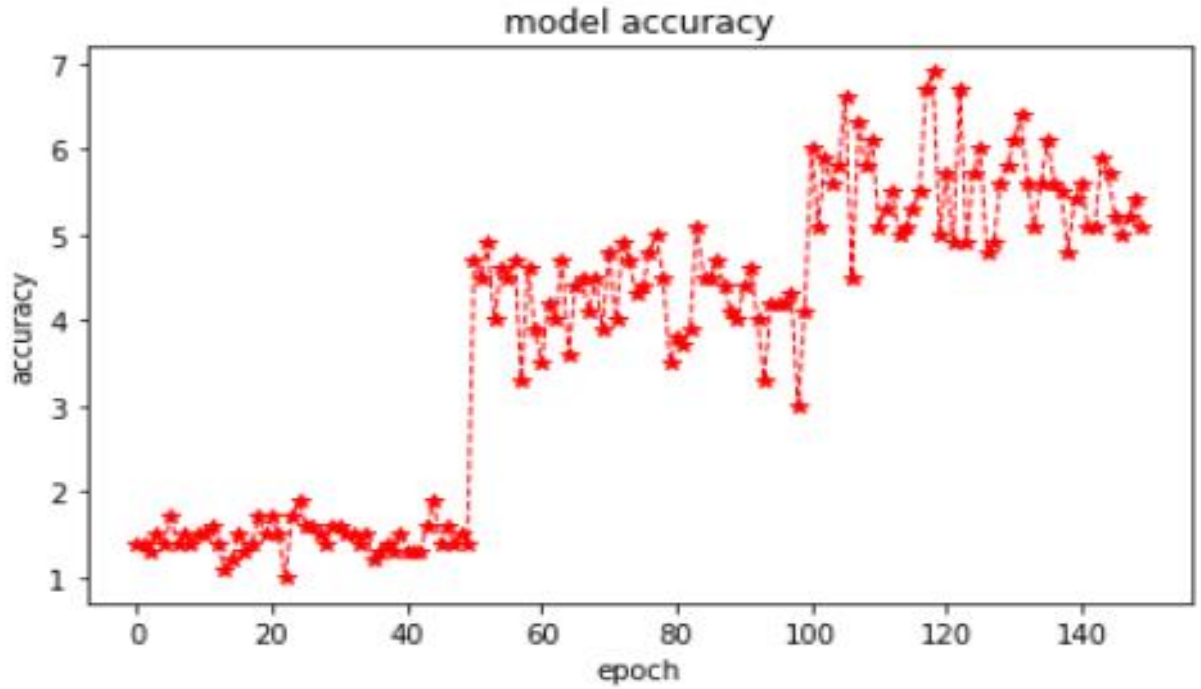
```
Out[52]: Text(0.5, 0, 'epoch')
```



Labels/Size of figure:

```
In [64]: fig = plt.figure(figsize=(7,4))
         plt.plot(data["Petal_length"], color='r',marker='*',linestyle='dashed',linewidth=1.0)
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
```

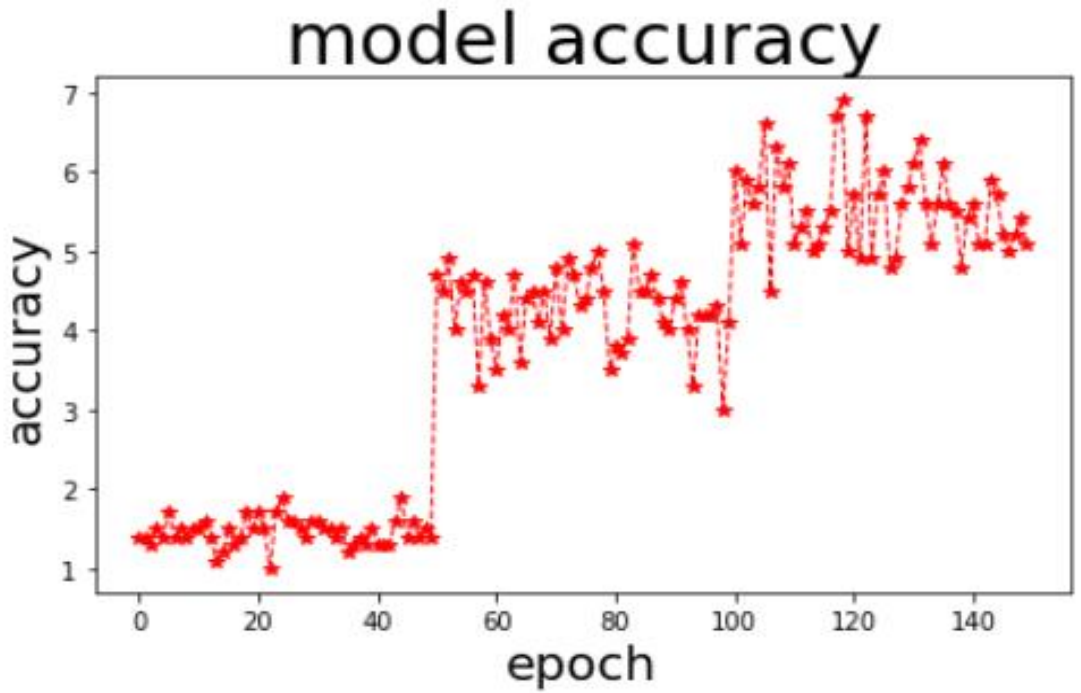
```
Out[64]: Text(0.5, 0, 'epoch')
```



Font Size of labels:

```
In [68]: fig = plt.figure(figsize=(7,4))
         plt.plot(data["Petal_length"], color='r',marker='*',linestyle='dashed',linewidth=1.0)
         plt.title('model accuracy',fontsize=30)
         plt.ylabel('accuracy',fontsize=20)
         plt.xlabel('epoch',fontsize=20)
```

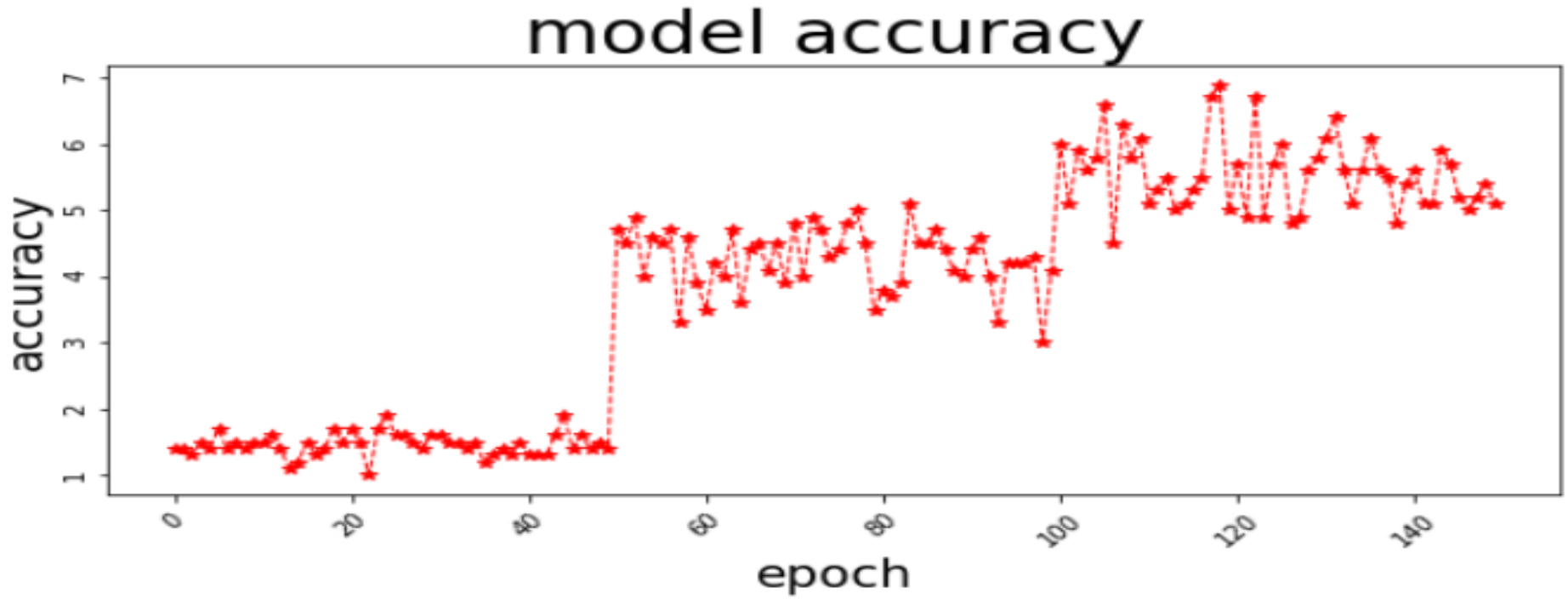
```
Out[68]: Text(0.5, 0, 'epoch')
```



Rotation of axis text:

In [80]:
```
fig = plt.figure(figsize=(10,4))
plt.plot(data["Petal_length"], color='r',marker='*',linestyle='dashed',linewidth=1.0)
plt.title('model accuracy',fontsize=30)
plt.xticks(rotation=45)
plt.yticks(rotation=90)
plt.ylabel('accuracy',fontsize=20)
plt.xlabel('epoch',fontsize=20)
```

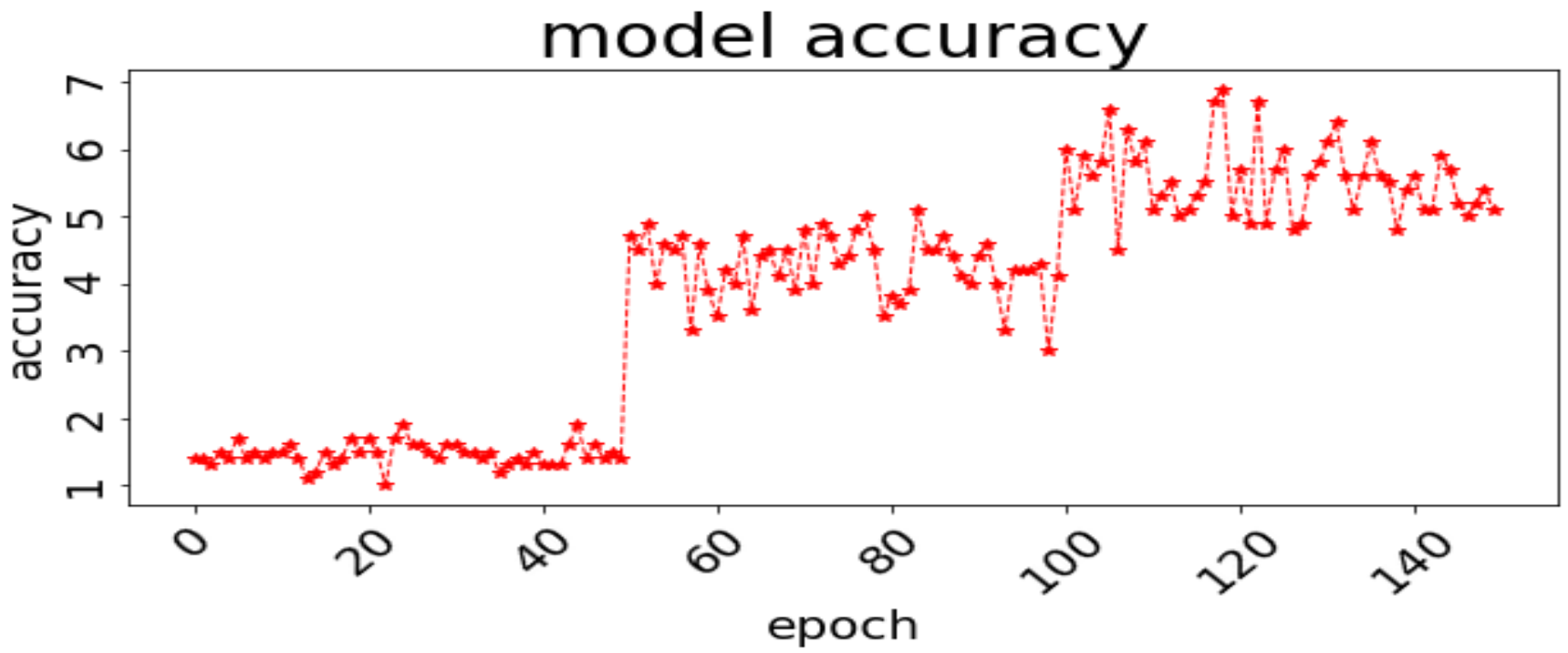
Out[80]:
```
Text(0.5, 0, 'epoch')
```



Size of axis text:

In [82]:
```
fig = plt.figure(figsize=(10,4))
plt.plot(data["Petal_length"], color='r',marker='*',linestyle='dashed',linewidth=1.0)
plt.title('model accuracy',fontsize=30)
plt.xticks(fontsize=20, rotation=45)
plt.yticks(fontsize=20,rotation=90)
plt.ylabel('accuracy',fontsize=20)
plt.xlabel('epoch',fontsize=20)
```

Out[82]: Text(0.5, 0, 'epoch')



THANK YOU

Assignments

Assignment – 1

Print the following pattern

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

Assignment – 1

Print the following pattern

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

```
for i in range(1,6):
    for j in range(1,i+1):
        print (j, end=" ")
    print("\n")
```

```
1

1 2

1 2 3

1 2 3 4

1 2 3 4 5
```

Assignment – 2

WAP to print the first N Fibonacci numbers. N should be user defined.

0, 1, 1, 2, 3, 5, 8, 13, 21,……

Assignment – 2

WAP to print the first N Fibonacci numbers. N should be user defined.

0, 1, 1, 2, 3, 5, 8, 13, 21,……

```
nterms = int(input("Enter the Value of n"))

n1 = 0
n2 = 1
count = 0

if nterms <= 0:
   print("Please enter a positive integer")
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
else:
   print("Fibonacci sequence upto",nterms,":")
   while count < nterms:
       print(n1,end=' , ')
       nth = n1 + n2

       n1 = n2
       n2 = nth
       count += 1
```

```
Enter the Value of n20
Fibonacci sequence upto 20 :
0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , 89 , 144 , 233 , 377 , 610 , 987 , 1597 , 2584 , 4181 ,
```

Assignment – 3

Calculate the Area of triangle, given the sides are 5,6,7

Assignment – 3

Calculate the Area of triangle, given the sides are 5,6,7

Use Heron's formula with sides a, b and c

s=(a+b+c)/2

Area= (s(s−a)(s−b)(s−c))^0.5

Assignment – 3

Calculate the Area of triangle, given the sides are 5,6,7

```
a = 5
b = 6
c = 7


# calculate the semi-perimeter
s = (a + b + c) / 2

# calculate the area
area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
print('The area of the triangle is {}' .format(area))
```

```
The area of the triangle is 14.696938456699069
```

Assignment – 4

Find Armstrong Number From 100 to 2000

Assignment – 4

Find Armstrong Number From 100 to 2000

An Armstrong number of three digits is an integer such that the sum of the cubes of its digits is equal to the number itself.

For example, 371 is an Armstrong number since 3*3*3 + 7*7*7 + 1*1*1= 371.

Assignment – 4

Find Armstrong Number From 100 to 2000

```
lower = 100
upper = 2000

for num in range(lower, upper + 1):

    order = len(str(num))

    sum = 0

    temp = num
    while temp > 0:
        digit = temp % 10
        sum += digit ** order
        temp //= 10

    if num == sum:
        print(num)
```

```
153
370
371
407
1634
```

Assignment – 5

Print all Prime Numbers from 100 to 1000

Assignment – 5

Print all Prime Numbers from 100 to 1000

A prime number is a whole number greater than 1 whose only factors are 1 and itself.

The first few prime numbers are:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29……..

Assignment – 5

Print all Prime Numbers from 100 to 1000

```
lower = 100
upper = 1000

# uncomment the following lines to take input from the user
#lower = int(input("Enter lower range: "))
#upper = int(input("Enter upper range: "))

print("Prime numbers between",lower,"and",upper,"are:")

for num in range(lower,upper + 1):
   # prime numbers are greater than 1
   if num > 1:
       for i in range(2,num):
           if (num % i) == 0:
               break
       else:
           print(num,end=" ")
```

```
Prime numbers between 100 and 1000 are:
101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269
271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461
463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661
673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881 883
887 907 911 919 929 937 941 947 953 967 971 977 983 991 997
```

Assignment – 6

Write a program to find the sum of Natural Number given by the User ?

Natural Numbers : all non negative numbers

0, 1, 2, 3, 4, 5, 6, 7, 8, ……

Assignment – 6

Write a program to find the sum of Natural Number given by the User ?

:

```
num = int(input("Enter a number: "))

if num < 0:
    print("Enter a positive number")
else:
    sum = 0

    while(num > 0):
        sum += num
        num -= 1
    print("The sum is",sum)
```

```
Enter a number: 10
The sum is 55
```

Assignment – 7

Write a program to Accept two int values from user and return their product. If the product is greater than 1000, then return their sum.

Assignment – 7

Write a program to Accept two int values from user and return their product. If the product is greater than 1000, then return their sum.

```
def multiplication_or_sum(num1, num2):
  product = num1 *num2
  if(product < 1000):
    return product
  else:
    return num1 +num2
number1 = int(input("Enter first number "))
number2 = int(input("Enter second number"))
result = multiplication_or_sum(number1, number2)
print("The result is", result)
```

```
Enter first number 12
Enter second number10
The result is 120
```

Assignment – 8

Write a program to check if the input that is given by the user is a Leap Year.

Leap Year: a year, occurring once every four years, which has 366 days including 29 February as an intercalary day.

Assignment – 8

Write a program to check if the input that is given by the user is a Leap Year.

```
# Python program to check if the input year is a leap year or not

# To get year (integer input) from the user
year = int(input("Enter a year: "))

if (year % 4) == 0:
   if (year % 100) == 0:
       if (year % 400) == 0:
           print("{0} is a leap year".format(year))
       else:
           print("{0} is not a leap year".format(year))
   else:
       print("{0} is a leap year".format(year))
else:
   print("{0} is not a leap year".format(year))
```

```
Enter a year: 156
156 is a leap year
```

Assignment – 9

Write a program to find the sum of Natural Number given by the User ?

```
num = int(input("Enter a number: "))

if num < 0:
   print("Enter a positive number")
else:
   sum = 0

   while(num > 0):
       sum += num
       num -= 1
   print("The sum is",sum)
```

```
Enter a number: 10
The sum is 55
```

Assignment – 10

Return the number of times that the string “Emma” appears anywhere in the given string.

Assignment – 10

Return the number of times that the string “Emma” appears anywhere in the given string.

```
def count_jhon(statement):
  count = 0
  for i in range(len(statement)-1):
    count += statement[i:i+4] == 'Emma'
  return count
count = count_jhon("Emma is good developer. Emma is aslo a writer")
print("Emma appeared ", count, "times")
```

```
Emma appeared  2 times
```

Assignment – 10

Apply all the operations of Numpy, Pandas and Matplotlib on the Iris dataset