Sumedh

* Complete Python
  - Mostly Theory

↓ all ppts merged

SRM Institute of Science and Technology, Chennai

Prepared by:

Dr. P. Robert

Assistant Professor

Department of Computing Technologies

SRMIST-KTR

SRM Institute of Science and Technology, Chennai

LEARNING RESOURCES	
S. No	**TEXT BOOKS**
1.	*Zed A Shaw, Learn C the Hard Way: Practical Exercises on the Computational Subjects You Keep Avoiding (Like C), Addison Wesley, 2015*
2.	*W. Kernighan, Dennis M. Ritchie, The C Programming Language, 2nd ed. Prentice Hall, 1996*
3.	*Bharat Kinariwala, Tep Dobry, Programming in C, eBook*
4.	*http://www.c4learn.com/learn-c-programming-language/*

Unit-IV

Python: Introduction to Python - Introduction to Google Colab - Basic Data Types: Integers, Floating Points, Boolean types - Working with String functions - Working with Input, Output functions - Python-Single and Multi line Comments/ Error Handling - Conditional & Looping Statements : If, for, while statements - Working with List structures - Working with Tuples data structures - Working with Sets - Working with Dictionaries - Introduction to Python Libraries - Introduction to Numpy - High Dimensional Arrays

Introduction to Python

- Python is a general-purpose programming language.
- It is high level and object-oriented programming language.
- Python is a very simple programming language so even if you are new to programming, you can learn python without facing any issues.
- Python is used in all domains
  - Web Development
  - Software Development
  - Game Development
  - AI & ML
  - Data Analytics

Python is developed by- Guido van Rossum

Features of Python Programming

- ✔ Python is free and **open-source** programming language.
- ✔ It is high level programming language.
- ✔ It is simple and easy to learn.
- ✔ It is **portable**. That means python programs can be executed on various platforms without altering them.
- ✔ It is object-oriented programming language.
- ✔ It can be **embedded** in or C++ programs.
- ✔ Python programs are ***interpreted***. (No need to compile).
- ✔ It has huge set of **library**.
- ✔ Python has a powerful set of built-in data types and easy to use control statements.

Interactive Mode

The >>> indicates that the python shell is ready to execute and send your commands to the python interpreter. The result is immediately displayed on the python shell.

To run your python statements, just type them and hit the enter key. You will get the results immediately.

For Example, to print the text “Hello World”

```
>>> print(“Hello World”)
Hello World
>>>
```

Pros and Cons of Interactive Mode

The following are the advantages of running your code in interactive mode:

1. Helpful when your script is extremely short and you want immediate results.
2. Faster as you only have to type a command and then press the enter key to get the results.
3. Good for beginners who need to understand Python basics.

Script Mode

If we need to write a long piece of python program, script mode is the right option. In script mode, we have to write a code in a text file then save it with a .py extension which stands for python. Note that we can use any text editor for this, including Sublime, Atom, notepad++, etc.

First.py // save the file name

```
a=10
b=20
if a>b:
    print("a is greater")
else:
    print("b is greater")
```

Output

```
C:\Users\Jose>python abc.py
b is greater
```

Pros and Cons of Script Mode

The following are the advantages of running your code in script mode:

1.It is easy to run large pieces of code.

2.Editing your script is easier in script mode.

3.Good for both beginners and experts.

The following are the disadvantages of using the script mode:

1.Can be tedious when you need to run only a single or a few lines of cod.

2.You must create and save a file before executing your code.

Basic data types

Integers

In Python, integers are zero, positive or negative whole numbers without a fractional part and having unlimited precision, e.g. 0, 100, -10. The followings are valid integer literals in Python.

Example:

```
#integer variables
x = 0
print(x)
x = 100
print(x)
x = -10
print(x)
x = 1234567890
print(x)
x = 5000000000000000000000000000000000000000000000000000
print(x)
```

Basic data types

Integers

Integers can be binary, octal, and hexadecimal values.

Example:

```
b = 0b11011000 # binary
print(b)
216
o = 0o12 # octal
print(o)
10
h = 0x12 # hexadecimal
print(h)
18
```

Basic data types

Float

In Python, floating point numbers (float) are positive and negative real numbers with a fractional part denoted by the decimal symbol . or the scientific notation E or e, e.g. 1234.56, 3.142, -1.55, 0.23.

Example:

```
f = 1.2
print(f) #output: 1.2
print(type(f)) #output: <class 'float'>
f=123_42.222_013 #output: 12342.222013
print(f)
f=2e400
print(f) #output: inf
```

As you can see, a floating point number can be separated by the underscore _. The maximum size of a float is depend on your system. The float beyond its maximum size referred as inf, Inf, INFINITY, or infinity. For example, a float number 2e400 will be considered as infinity for most systems.

Basic data types

Float

Use the float() function to convert string, int to float.

Example:

```
f=float('5.5')
print(f) #output: 5.5

f=float('5')
print(f) #output: 5.0
```

Basic data types

Boolean

The Python Boolean type is one of Python's built-in data types. It's used to represent the truth value of an expression. For example, the expression 1 <= 2 is True , while the expression 0 == 1 is False . Understanding how Python Boolean values behave is important to programming well in Python.

Example:

```
>>> type(False)
<class 'bool'>
>>> type(True)
<class 'bool’>
a=1
b=2
c=a<b
print (c)
Output
True
```

Working with String Functions

String is collection of characters. All string methods return new values. They do not change the original string. Strings in python are surrounded by either single quotation marks, or double quotation marks.

```
>>> print(“Hello”)
‘Hello’
>>> print(‘Hello’)
‘Hello’
>>> dept=“Engineering”
‘Engineering’
```

String Methods

Looping through a String

We can loop through the characters in a string, with a for loop.

Example

```
for x in “Engineering”:
    print(x)
```

Output

```
E
N
G
I
N
E
E
R
I
N
G
```

String Methods

String Length

To get the length of the string, use the **len()** method.

```
>>> dept="Engineering"
>>> print(len(dept))
```

11

Check String (in keyword)

To check particular pattern or character is present in the string, we can use in keyword.

```
>>> student="I am doing Bio-Technology"
>>> print("Bio" in student)
```

True

String Methods

Check String(not in keyword)

To check pattern or character is present in the string, we can use in keyword.

```
>>> student="I am doing Bio-Technology"
>>> print("Bio" not in student)
False
```

String Slicing

We can extract range of characters by using the slice. Specify the start index and end index, separated by a colon, to return part of the string.

```
>>> student="I am doing Bio-Technology"
>>> print(student[0:5])
I am
```

String Methods

```
>>> student=“I am doing Bio-Technology”
>>> print(student[2:4])
am
```

Slice from the Start

```
>>> print(student[:25])
I am doing Bio-Technology
```

Slice to the End

```
>>> print(student[0:])
I am doing Bio-Technology
```

String Methods

Negative Indexing

```
>>> print(student[-1:])
y
>>> print(student[-2:])
gy
>>> print(student[-10:])
Technology
```

String Methods

Modify String

Upper Case

```
>>> name="Bana Singh"
>>> print(name.upper())
BANA SINGH
```

Lower Case

```
>>>print(name.lower())
bana singh
```

Remove Whitspace

```
>>> name="      Bana Singh   "
>>>print(name.strip()) // Removes whitespace from the beginning or end
Bana Singh
```

String Methods

Modify String

Replace String

```
>>> name=“Bana Singh”
>>> name
'Bana Singh'
>>> print(name.replace("Bana","Param Vir Chakra"))
Param Vir Chakra Singh
```

String Methods

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found

String Methods

Method	Description
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits

String Methods

Method	Description
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case

String Methods

Method	Description
join()	Converts the elements of an iterable into a string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value

String Methods

Method	Description
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string

String Methods

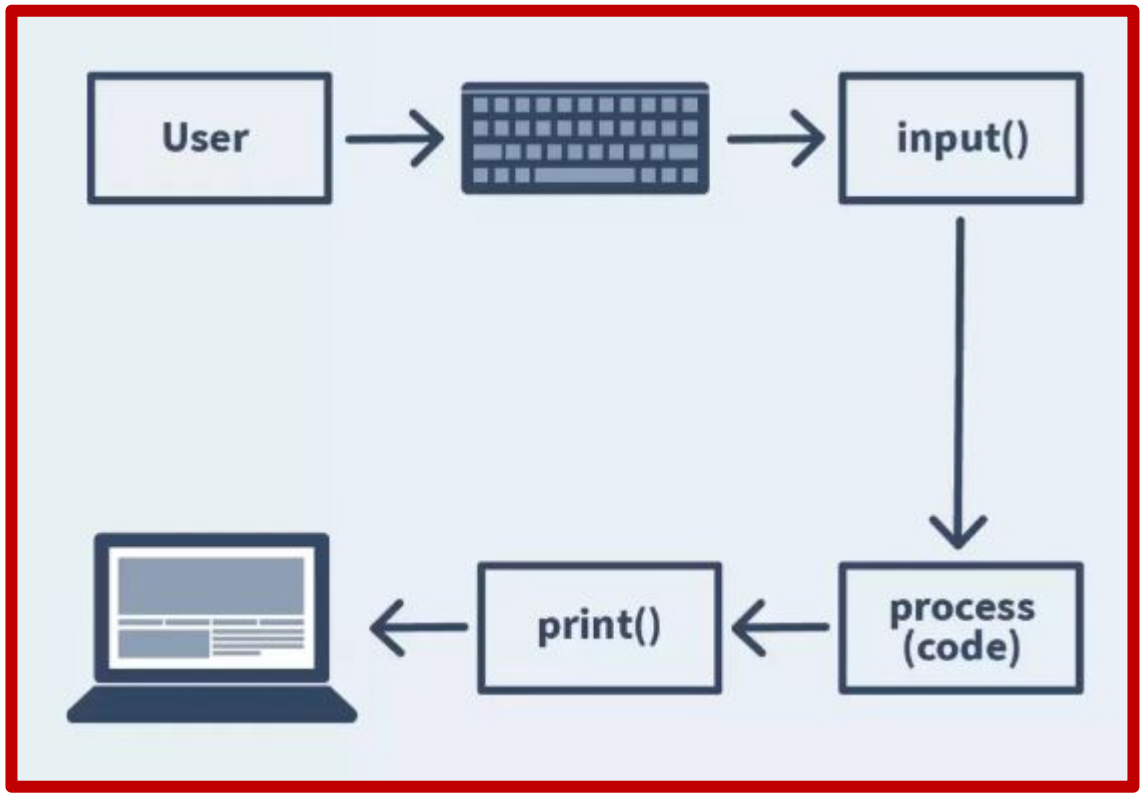
Method	Description
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string

String Methods

Method	Description
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

Working with input and output functions

In Python, we use the print() function to output data to the screen. Sometimes we might want to take input from the user. We can do so by using the input() function. Python takes all the input as a string input by default.



Working with input and output functions

Python Output

We use the widely used print() statement to display some data on the screen.

The syntax for **print()** is as follows:

print (“string to be displayed as output ”)

print (variable)

print (“String to be displayed as output ”, variable)

print (“String1 ”, variable, “String 2”, variable, “String 3” ……)

Example

>>>print (“Welcome to Python Programming”) Welcome to Python Programming

>>>x = 5

>>>y = 6

>>>z = x + y

>>>print (z)

Working with input and output functions

Python Output

We use the widely used print() statement to display some data on the screen.

The syntax for **print()** is as follows:

print (“string to be displayed as output ”)

print (variable)

print (“String to be displayed as output ”, variable)

print (“String1 ”, variable, “String 2”, variable, “String 3” ……)

Example

>>>print (“Welcome to Python Programming”) Welcome to Python Programming

>>>x = 5

>>>y = 6

>>>z = x + y

>>>print (z)

11

Working with input and output functions

Python Input

In Python, input( ) function is used to accept data as input at run time. The syntax for input() function is,

Variable = input (“prompt string”)

Where, prompt string in the syntax is a statement or message to the user, to know what input can be given. If a prompt string is used, it is displayed on the monitor; the user can provide expected data from the input device. The input( ) takes whatever is typed from the keyboard and stores the entered data in the given variable. If prompt string is not given in input( ) no message is displayed on the screen, thus, the user will not know what is to be typed as input.

Example:

```
>>>city=input (“Enter Your City: ”)
Enter Your City: Tirunelveli
>>>print (“I am from “, city)
I am from Tirunelveli
```

Working with input and output functions

Python Input

```
x = int (input("Enter Number 1: "))
y = int (input("Enter Number 2: "))
print ("The sum = ", x+y)
```

Output:

```
Enter Number 1: 34
Enter Number 2: 56
The sum = 90
```

Python- Single and Multi Line Comments

A comment is a piece of code that isn't executed by the compiler or interpreter when the program is executed. Comments can only be read when we have access to the source code. Comments are used to explain the source code and to make the code more readable and understandable.

Single Line Comment in Python

Single line comments are those comments which are written without giving a line break or newline in python.

Example:

```
#This is a single line comment in python
import numpy as np
```

Multi Line Comment in Python

As the name specifies, a multi line comment expands up to multiple lines. But python does not have syntax for multi line comments. We can implement multi line comments in python using single line comments or triple quoted python strings.

Example:

```
"""This is a multiline comment in python which
    expands to many lines"
```

Error Handling in Python

An exception is a type of error that occurs during the execution of a program.

However, Python throws an exception while running a program, which must be handled to prevent your application from crashing.

Python has many built-in exceptions that are raised when your program encounters an error (something in the program goes wrong).

When these exceptions occur, the Python interpreter stops the current process and passes it to the calling process until it is handled. If not handled, the program will crash.

Error Handling in Python

For example, let us consider a program where we have a function A that calls function B, which in turn calls function C. If an exception occurs in function C but is not handled in C, the exception passes to B and then to A.

If never handled, an error message is displayed and our program comes to a sudden unexpected halt.

Error Handling in Python

Why should we use exceptions?

1. Exception handling mechanism allows you to separate error-handling code from normal code.
2. It clarifies the code and enhanced readability.
3. It is a convenient method for handling error messages.
4. In python, we can raise an exception in the program by using the raise exception method.

Error Handling in Python

Read

Important Python Errors

Error Type	Description
ArithmeticError	ArithmeticError act as a base class for all arithmetic exceptions. It is raised for errors in arithmetic operations.
ImportError	ImportError is raised when you are trying to import a module which does not present. This kind of exception occurs if you have made typing mistake in the module name or the module which is not present in the standard path.
IndexError	An IndexError is raised when you try to refer a sequence which is out of range.
KeyError	When a specific key is not found in a dictionary, a KeyError exception is raised.
NameError	A NameError is raised when a name is referred to in code which never exists in the local or global namespace.

Error Handling in Python

Important Python Errors

Error Type	Description
ValueError	Value error is raised when a function or built-in operation receives an argument which may be of correct type but does not have suitable value.
EOFerror	This kind of error raises when one of the built-in functions (input() or raw_input()) reaches an EOF condition without reading any data.
ZeroDivisonError	This type of error raised when division or module by zero takes place for all numeric types.
IOError-	This kind of error raised when an input/output operation fails.
syntaxError	SyntaxErrors raised when there is an error in Python syntax.
IndentationError	This error raised when indentation is not properly defined

Error Handling in Python

Example:

```
if age>=18:
        print("Eligible to Vote")
```

Error:

```
Traceback (most recent call last):
  File "C:/Users/Jose/Desktop/demo2.py", line 1, in <module>
    if age>=18:
NameError: name 'age' is not defined
```

Error Handling in Python

Example: (Division by Zero Error)

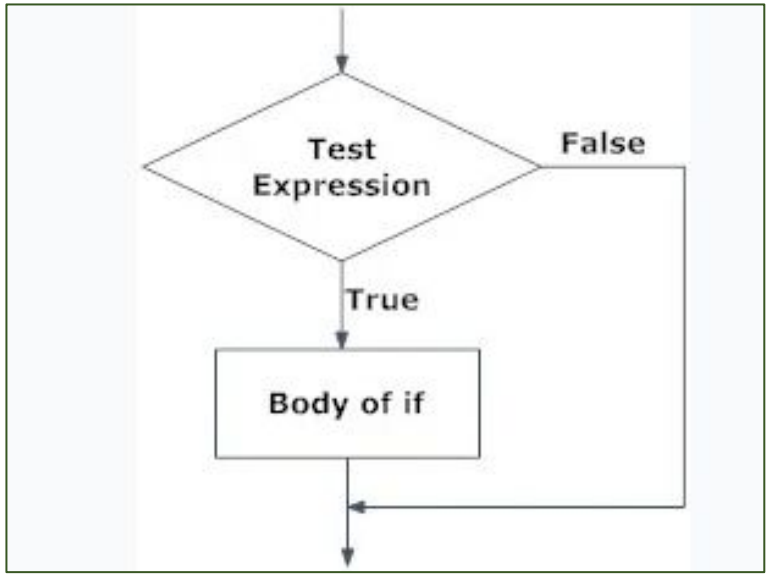
```
import sys
a=20
try:
    a/0
except ArithmeticError as e:
    print (e)
    #print (sys.exc_type)
    print ('This is an example of catching ArithmeticError')
```

Output:

```
division by zero
This is an example of catching ArithmeticError
```

Python Conditional and Looping Statements

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b



Python if statement syntax

```
if  test_expression:
        statements(s)
```

Here, the program evaluates the test expression and will execute statement(s) only **if the test expression is True**.

If the test expression is **False**, the statement(s) is not executed.

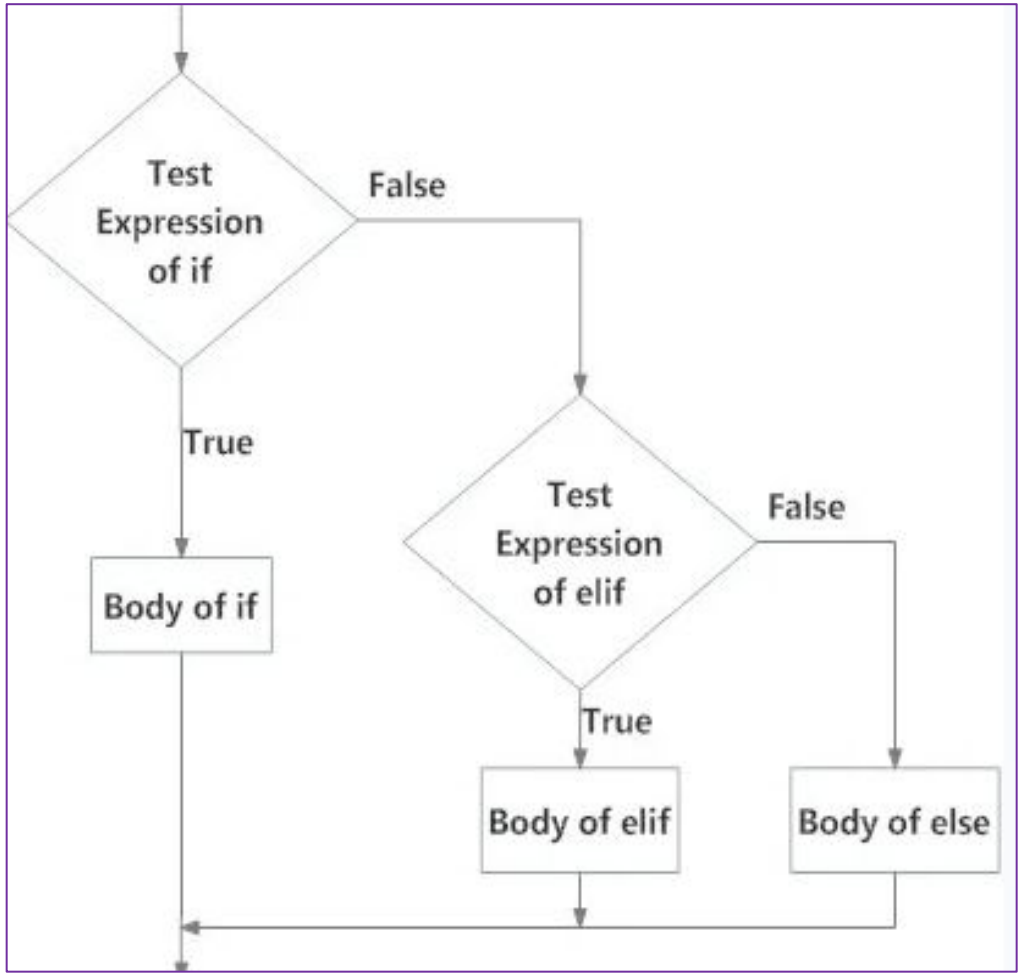
Python Conditional and Looping Statements

Example:

```
Num=3
If Num>0:
    print("Number positive")
```

Python if statement syntax

```
if test_expression:
    Body of if
elif test_expression:
    Body of elif
else:
    Body of else
```



Python Conditional and Looping Statements

The elif is short for else if. It allows us to check for multiple expressions. If the condition for if is False, it checks the condition of the next elif block and so on. If all the conditions are False, the body of else is executed. Only one block among the several if...elif...else blocks is executed according to the condition.

Example:

```
num = 3.4
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

Python Conditional and Looping Statements

Python nested if statement syntax

```
if boolean_expression:
    if boolean_expression:
        statement(s)
    else:
        statement(s)
else:
    statement(s)
```

Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. They can get confusing, so they must be avoided unless necessary.

Python Conditional and Looping Statements

Python nested if statement syntax

Example:

```
x = 30
y = 10
if x >= y:
    print("x is greater than or equals to y")
    if x == y:
        print("x is equals to y")
    else:
        print("x is greater than y")
else:
    print("x is less than y")
```

x is greater than or equals to y
x is greater than y

Python Conditional and Looping Statements

Python Loops

- **while loop**
- **for loop**

Loops are used to perform same set of statements again and again**.**

While loop in Python

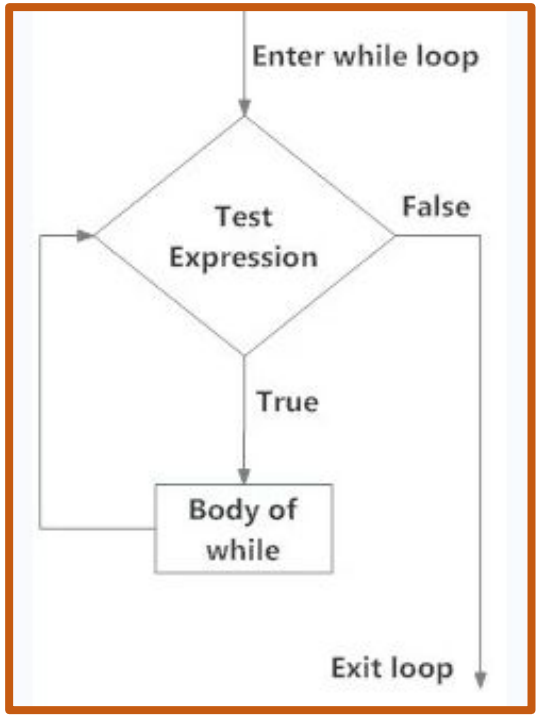
The block of statements in the while loop is executed as long as the test condition is true.

Syntax of while loop

```
while test_expression:
    Body of while
```

Python Conditional and Looping Statements

In the while loop, test expression is checked first. The body of the loop is entered only if the test_expression evaluates to True. After one iteration, the test expression is checked again. This process continues until the test_expression evaluates to False.



Python Conditional and Looping Statements

Example:

```
n = 10
sum = 0
i = 1
while i <= n:
    sum = sum + i
    i = i+1    # update counter
print("The sum is", sum)
```

Output:

```
The sum is 55
```

Python Conditional and Looping Statements

Example: (Find a list of all even numbers from 1 to 25 and calculate the sum of even numbers)

```
n = 25
sum = 0

print("Even numbers are")
i = 1

while i <= n:

    if i%2==0:

        print(i)
        sum=sum+i

    i = i+1    # update counter

print("The sum is", sum)
```

Output

```
Even numbers are
2
4
6
8
10
12
14
16
18
20
22
24
The sum is 156
```

Python Conditional and Looping Statements

for loop in Python

The python for loop is used to iterate over a sequence (list, tuple, String). Iterating over a sequence is called traversing.

Syntax of for loop

```
for val in sequence:
    loop body
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration. Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Python Conditional and Looping Statements

Example: (Find a list of all even numbers from 1 to 25 and calculate the sum of even numbers)

```
n = 25
sum = 0

print("Even numbers are")

for i in range(n):
    if i%2==0:
        print(i)
        sum=sum+i
    i = i+1    # update counter

print("The sum is", sum)
```

Output

```
Even numbers are
2
4
6
8
10
12
14
16
18
20
22
24
The sum is 156
```

Python Conditional and Looping Statements

Example: (Prime no or not)

```
n=int(input(""))
i=2
flag=0
while i<=n/2:
    if n%i==0:
        flag=1
        break
    i=i+1
if flag==0:
    print("prime  ",n)
Else:
   print(“Not a prime ”, n)
```

Python Conditional and Looping Statements

Example: (Prime no or not)

```
n=int(input(""))
i=2
flag=0
while i<=n/2:
    if n%i==0:
        flag=1
        break
    i=i+1
if flag==0:
    print("prime  ",n)
Else:
   print(“Not a prime ”, n)
```

Python Conditional and Looping Statements

Example: (Palindrome)

```
rev=0
n=int(input("Enter any no"))
temp=n
while n>0:
    r=n%10
    rev=rev*10+r
    n=int(n/10)
  #  print(n)
if rev==temp:
    print("Palindrome")
else:
    print("No")
```

Python Conditional and Looping Statements

Example: (Reverse the number)

```
rev=0
n=int(input("Enter any no"))
temp=n
while n>0:
    r=n%10
    rev=rev*10+r
    n=int(n/10)
  #  print(n)
print(rev)
```

Working with List Structures

Lists

List is used to store multiple elements separated by comma and enclosed within the square brackets[ ]. List elements are ordered (it means that elements are added at the end of list), changeable (we can add or remove elements from the list), and allow duplicate values. List elements are indexed , the first index is 0.

Working with List Structures

Lists

```
>>> Games=[“Cricket”,”Hockey”,”Football”]
>>> Games
['Cricket', 'Hockey', 'Football’]
```

len() Method

To determine the number of elements in the list

```
>>> len(Games)
3
```

type() Method

To find the type of variable used in the program

```
>>> print(type(Games))
<class 'list'>
```

Working with List Structures

Lists

To access elements from the list

```
>>> Games.append("Kabadi")
>>> Games
['Cricket', 'Hockey', 'Football', 'Kabadi']
```

To access the first element

```
>>> Games[1] or print(Games[1])
Hockey
```

Negative Indexing

It means that indexing start from the last item. -1 refers to the last item, -2 refers to the second last item etc.

Working with List Structures

Lists

```
>>> Games[-1] or print(Games[-1])
```

Kabadi

```
>>> Games[-2] or print(Games[-2])
```

Football

Range of Elements

To access range of elements from the list, mention the starting index and ending index. The output will be a new list with specified number of elements.

```
>>> print(Games[1:2])
```

['Hockey']

Working with List Structures

Lists

```
>>> Games
['Cricket', 'Hockey', 'Football', 'Kabadi']
>>> Games[2:]
['Football', 'Kabadi']
```

Remove List (We can specify the element name or index while deleting)

```
>>> Games.remove("Kabadi")
>>> Games
['Cricket', 'Hockey', 'Football']
```

Working with List Structures

Lists

Iterate or Loop through elements

```
>>> Games.append("Golf")
>>> Games.append("Tennis")
>>> Games
['Cricket', 'Hockey', 'Football', 'Golf', 'Tennis']
```

```
>>> for x in Games
...     print(x)
...
Cricket
Hockey
Football
Golf
Tennis
```

Working with List Structures

Lists

Sorting Elements in the List (Ascending or Descending)

```
>>> Games
['Cricket', 'Hockey', 'Football', 'Golf', 'Tennis']
>>> Games.sort()
>>> Games
['Cricket', 'Football', 'Golf', 'Hockey', 'Tennis'] // Ascending
>>> Games.sort(reverse=True)
>>> Games
['Tennis', 'Hockey', 'Golf', 'Football', 'Cricket']
```

Working with List Structures

Lists

Copy List elements to another rlist

```
>>> Games
['Cricket', 'Hockey', 'Football', 'Golf', 'Tennis']
>>>L1= Games.copy()
>>>L1
['Cricket', 'Hockey', 'Football', 'Golf', 'Tennis']
```

Join two different Lists

```
>>> Games
['Cricket', 'Hockey', 'Football', 'Golf', 'Tennis']
>>> L1
['Cricket', 'Hockey', 'Football', 'Golf', 'Tennis']
```

Working with List Structures

Lists

Join two different Lists // Approach-1 (Use concatenation operator)

```
>>> L2=Games+L1
>>> L2
['Tennis', 'Hockey', 'Golf', 'Football', 'Cricket', 'Tennis', 'Hockey', 'Golf', 'Football',
'Cricket']
```

Join two different Lists // Approach-2 (Use append() method)

```
>>> list2
[1, 2, 3]
>>> list1
['x', 'y', 'z']
>>> for x in list2:
...     list1.append(x)
...
>>> print(list1)
['x', 'y', 'z', 1, 2, 3]
```

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Working with Tuples Data Structures

Tuple

Tuple is also a collection data type used to store more elements in a single variable name. It is unchangeable (**Immutable**). In tuples, elements are enclosed in round brackets.

Create a Tuple

```
>>> t1=(1,2,3)   //tuple name is t1
>>> t1
(1, 2, 3)
>>> t2=("a","b","c")  // //tuple name is t2
>>> t2
('a', 'b', 'c')
```

Working with Tuples Data Structures

Tuple

Access Tuple elements // Use index , inside the square brackets

```
>>> t2
('a', 'b', 'c')
>>> print( t2[1])
```

Change Tuple Values

Once tuple is created, we cannot change its values. Because tuples are **immutable**.

But there is another way, **Convert tuple into a list**, **change the list**, and **convert the list back into a tuple**.

Working with Tuples Data Structures

Tuple

```
>>> t2
('a', 'b', 'c')          // Elements in tuple-t2
>>> t3=list(t2)          // tuple is converted into a list
>>> t3
['a', 'b', 'c']

>>> t3[1]="Butterfly"     // change the element in list

>>> t3

['a', 'Butterfly', 'c']

>>> t2=tuple(t3)          // Convert the list into tuple

>>> t2

('a', 'Butterfly', 'c')
```

Working with Tuples Data Structures

Tuple

Loop through a Tuple

```
>>> cricketers=("Rohit","kholi","Dawan","Ashwin","Jadeja")

>>> cricketers

('Rohit', 'kholi', 'Dawan', 'Ashwin', 'Jadeja')

>>> for x in cricketers:
...     print(x)
...
Rohit
kholi
Dawan
Ashwin
Jadeja
```

Working with Tuples Data Structures

Tuple

Join Tuples

```
>>> cricketers=("Rohit","kholi","Dawan","Ashwin","Jadeja")
>>> bowlers= ("Kuldip","Bumrah","Ishanth","Pandya")
>>> cricket= cricketers + bowlers
('Rohit', 'kholi', 'Dawan', 'Ashwin', 'Jadeja', 'Kuldip', 'Bumrah', 'Ishanth',
'Pandya')
```

Working with Sets

Sets are used to store multiple items in a single variable. A set is a collection of unordered, unchangeable, and unindexed.

Note: Sets are written with curly brackets.

```
//Create a set
Fruits = {"apple", "banana", "cherry"}
print(Fruits)
//Duplicates not allowed
Fruits = {"apple", "banana", "cherry", "apple"}
print(Fruits)
```

Output:

```
{'banana', 'cherry', 'apple'}
```

```
//Length of a set
Fruits = {"apple", "banana", "cherry"}
print(len(Fruits))
```

Output:

```
3
```

Working with Dictionaries

Dictionary

It is used to store data values in **key:value pairs**. Dictionary is a **collection** which is **ordered**, changeable (mutable) and do not allow duplicate values. Dictionary uses curly brackets.

```
>>> shop={"name":"Easybuy","type":"Textiles","year":2006, "year":2006}

>>> print(shop)

{'name': 'Easybuy', 'type': 'Textiles', 'year': 2006}
```

Working with Dictionaries

Dictionary

Accessing Elements

```
>>> print(shop)
{'name': 'Easybuy', 'type': 'Textiles', 'year': 2006}
>>> shop["year"]
2006
```

Change Dictionary Elements // Approach-1

```
>>> shop["year"]=2012
>>> shop
{'name': 'Easybuy', 'type': 'Textiles', 'year': 2012}
```

Working with Dictionaries

Dictionary

Change Dictionary Elements //Approach-2

```
>>> shop.update({“year”:2018})
>>> shop
{'name': 'Easybuy', 'type': 'Textiles', 'year': 2018}
```

Add elements into Dictionary

```
>>> shop[“rate”:”Good”]
>>> shop
{'name': 'Easybuy', 'type': 'Textiles', 'year': 2018, 'rate': 'Good'}
```

Working with Dictionaries

Dictionary

Remove elements from Dictionary

```
>>> shop.pop("rate")
'Good'
>>> shop
{'name': 'Easybuy', 'type': 'Textiles', 'year': 2018}
```

Loop through Dictionary

```
>>> for x in shop:
...     print(x+ "="+str(shop[x]))
...
name=Easybuy
type=Textiles
year=2018
```

Working with Dictionaries

Dictionary

Copy a Dictionary // Use copy method

```
>>> shop
{'name': 'Easybuy', 'type': 'Textiles', 'year': 2018}
>>> shop1=shop.copy()
>>> shop1
{'name': 'Easybuy', 'type': 'Textiles', 'year': 2018}
```

Working with Dictionaries

Dictionary Methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Introduction to Python Libraries

In the **programming world**, a library is a collection of precompiled codes that can be used later on in a program for some specific well-defined operations.

A Python library is a **collection of related modules**. It contains bundles of code that can be used repeatedly in different programs. It makes Python Programming simpler and convenient for the programmer.

A library is a collection of **utility methods**, classes, and modules that your application code can use to perform specific tasks without writing the functionalities from scratch.

List of Libraries in Python

1. Pandas

- Pandas is a BSD (Berkeley Software Distribution) licensed open-source library.
- This popular library is widely used in the field of data science. They are primarily used for data analysis, manipulation, cleaning, etc.
- Pandas allow for simple data modeling and data analysis operations without the need to switch to another language such as R.

Introduction to Python Libraries

Pandas can do a wide range of tasks, including:

- The data frame can be sliced using Pandas.
- Data frame joining and merging can be done using Pandas.
- Columns from two data frames can be concatenated using Pandas.
- In a data frame, index values can be changed using Pandas.
- In a column, the headers can be changed using Pandas.
- Data conversion into various forms can also be done using Pandas and many more.

Introduction to Python Libraries

2.NumPy

- The name "NumPy" stands for "Numerical Python".
- It is the commonly used library.
- It is a popular machine learning library that supports large matrices and multi-dimensional data.
- It can be used in linear algebra, as a multi-dimensional container for generic data, and as a random number generator, among other things.
- Some of the important functions in NumPy are arcsin(), arccos(), tan(), radians(), etc. NumPy Array is a Python object which defines an N-dimensional array with rows and columns.

Features

1. Numpy is a very interactive and user-friendly library.
2. NumPy simplifies the implementation of difficult mathematical equations.
3. It makes coding and understanding topics a breeze.
4. The NumPy interface can be used to represent images, sound waves, and other binary raw streams as an N-dimensional array of real values for visualization.

Introduction to Python Libraries

3. Keras

- Keras is an open-source neural network library written in Python that allows us to quickly experiment with deep neural networks.
- It is an API designed for humans rather than machines.
- It is the very important library used in deep learning.
- In comparison to TensorFlow or Theano, Keras has a better acceptance rate in the scientific and industrial communities.

Features

1. It runs without a hitch on both the CPU (Central Processing Unit) and GPU (Graphics Processing Unit).
2. Keras supports nearly all neural network models, including fully connected, convolutional, pooling, recurrent, embedding, and so forth. These models can also be merged to create more sophisticated models.
3. Keras' modular design makes it very expressive, adaptable and suited well to cutting-edge research.
4. Keras is a Python-based framework, that makes it simple to debug and explore different models and projects.

Introduction to Python Libraries

4. TensorFlow

- It is the high performance numerical calculation library that is open source.
- It is also employed in deep learning algorithms and machine learning algorithms.
- It was developed by researchers in the Google Brain group of the Google AI organization and is now widely used by researchers in physics, mathematics, and machine learning for complex mathematical computations.

Features

- ✔ Responsive Construct: We can easily visualize every part of the graph with TensorFlow, which is not possible with Numpy or SciKit.
- ✔ It is flexible in its operation related to machine learning models .
- ✔ It is easy to train machine learning models.
- ✔ TensorFlow allows you to train many neural networks and GPUs at the same time.

Introduction to Python Libraries

5. Scikit Learn

- It is a python-based open-source machine learning library.
- This library supports both supervised and unsupervised learning methods.
- Scikit learns most well-known use is for music suggestions in Spotify.
- The library includes popular algorithms as well as the NumPy, Matplotlib, and SciPy packages.

Features

- ✔ Cross-Validation: There are several methods for checking the accuracy of supervised models on unseen data with Scikit Learn for example the **train_test_split** method, **cross_val_score,** etc.
- ✔ Unsupervised learning techniques: There is a wide range of unsupervised learning algorithms available, ranging from clustering, factor analysis, principal component analysis, and unsupervised neural networks.
- ✔ Feature extraction: Extracting features from photos and text is a useful tool (e.g. Bag of words)

Introduction to Python Libraries

5. SciPy

- Scipy is a free, open-source Python library used for scientific computing, data processing, and high-performance computing.
- The name “SciPy” stands for Scientific Python.
- This library is built over an extension of Numpy. It works with Numpy to handle complex computations.

Features

- ✔ SciPy’s key characteristic is that it was written in NumPy, and its array makes extensive use of NumPy.
- ✔ SciPy uses its specialised submodules to provide all of the efficient numerical algorithms such as optimization, numerical integration, and many others.
- ✔ All functions in SciPy’s submodules are extensively documented. SciPy’s primary data structure is NumPy arrays, and it includes modules for a variety of popular scientific programming applications.

Introduction to Python Libraries

6. PyTorch

- PyTorch is the largest machine learning library that optimizes tensor computations.
- It has rich APIs to perform tensor computations with strong GPU acceleration.
- It also helps to solve application issues related to neural networks.

Features

- ✔ Python and its libraries are supported by PyTorch.
- ✔ Facebook's Deep Learning requirements necessitated the use of this technology.
- ✔ It provides an easy to use API that improves usability and comprehension.
- ✔ Graphs can be set up dynamically and computed dynamically at any point during code execution in PyTorch.
- ✔ In PyTorch, coding is simple and processing is quick.
- ✔ Because CUDA (CUDA is a parallel computing platform and application programming interface that allows software to use certain types of graphics processing unit for general purpose processing – an approach called general-purpose computing on GPUs) is supported, it can be run on GPU machines.

Introduction to Python Libraries

7. Matplotlib

- Matplotlib is a cross-platform, data visualization and graphical plotting library (histograms, scatter plots, bar charts, etc) for Python and its numerical extension NumPy.
- This library is responsible for plotting numerical data. And that's why it is used in data analysis. It is also an open-source library and plots high-defined figures like pie charts, histograms, scatterplots, graphs, etc.

Introduction to Python Libraries

8. PyBrain

- The name “PyBrain” stands for Python Based Reinforcement Learning, Artificial Intelligence, and Neural Networks library. It is an open-source library built for beginners in the field of Machine Learning. It provides fast and easy-to-use algorithms for machine learning tasks.
- It is so flexible and easily understandable and that’s why is really helpful for developers that are new in research fields.

Introduction to Python Libraries

List of other libraries in python are

1. **PyGTK**- Easily create programs with GUI.
2. **Fabric**- command-line tool for streamlining the use of SSH for application deployment or systems administration tasks.
3. **SymPy**- It is an open-source library for symbolic math.
4. **Flask**- A web framework, Flask is built with a small core and many extensions.
5. **Nose**- Nose delivers an alternate test discovery and running process for unittest.
6. **iPython**- iPython Python Library has an architecture that facilitates parallel and distributed computing.
7. **wxPython**- It is a wrapper around wxWidgets for Python.
8. **Pywin32**- This provides useful methods and class for interaction with Windows.
9. **Pillow**- Pillow is a friendly fork of PIL (Python Imaging Library), but is more user-friendly.
10. **PyGame**- PyGame provides an extremely easy interface to the Simple Directmedia Library (SDL) platform-independent graphic, audio, and input libraries.
11. **Scrapy**- If your motive is fast, high-level screen scraping and web crawling, go for Scrapy.

Introduction to NumPy

- NumPy is a python package.
- It stands for Numerical Python.
- It is a library consisting of multidimensional array objects and a collection of routines for processing of array.
- NumPy (**Numerical Python**) is an open source Python library that's used in almost every field of science and engineering.
- The NumPy API is used extensively in Pandas, SciPy, Matplotlib, scikit-learn, scikit-image and most other data science and scientific Python packages.
- The NumPy library contains multidimensional array and matrix data structures.
- NumPy is a Python library used for working with arrays.

Introduction to NumPy

Why use NumPy?

- ✔ In Python we have lists that serve the purpose of arrays, but they are slow to process.
- ✔ NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
- ✔ The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.
- ✔ Arrays are very frequently used in data science, where speed and resources are very important.

Introduction to NumPy

Operations using NumPy

- ✔ Mathematical and logical operations on arrays.
- ✔ Fourier transforms and routines for shape manipulation.
- ✔ Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

Introduction to NumPy

Examples
1D array, 2D array, ndarray, vector, matrix
What are the attributes of an array?

An array is usually a fixed-size container of items of the same type and size. The number of dimensions and items in an array is defined by its shape. The shape of an array is a tuple of non-negative integers that specify the sizes of each dimension.

How to create a basic Array

np.array()

```
import numpy as np
a = np.array([1, 2, 3])
```

Introduction to NumPy

Examples

1D array, 2D array, ndarray, vector, matrix

np.zeros()

```
import numpy as np
a=np.zeros(3)
print(a)
```

Output:

```
[0. 0. 0.]
```

Adding, Removing, and Sorting Elements

```
import numpy as np
a=np.array([1,3,4,2,7,8,12,10])
print("Before Sorting an Elements")
print(a)
print("After Sorting an Elements")
print(np.sort(a))
```

Output:

```
Before Sorting an Elements
[ 1  3  4  2  7  8 12 10]
After Sorting an Elements
[ 1  2  3  4  7  8 10 12]
```

Introduction to NumPy

Examples

Concatenate

```
import numpy as np
a = np.array([1, 2, 3, 4,5])
b = np.array([5, 6, 7, 8])
res=np.concatenate((a, b))
print(res)
```

Output:

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

To know the shape and size of the array

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
print(a.ndim)
```

Output:

```
2
```

Introduction to NumPy

Examples

Convert a 1D array into a 2D array

```
import numpy as np
a = np.array([1, 2, 3, 4, 5, 6])
print(a.shape)
a2 = a[np.newaxis, :]
print(a2.shape)
```

Output:

```
(6,)
(1, 6)
```

Indexing & Slicing

```
import numpy as np
data = np.array([1, 2, 3,4,5,6,7,8])
print(data[0:])
print(data[0:2])
print(data[-3:])
```

Output

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

Introduction to NumPy

Examples

Create an array from existing data

```
import numpy as np
a = np.array([1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
arr1 = a[3:8]
print(arr1)
```

Output:

```
[4 5 6 7 8]
```

Creating Matrices

```
import numpy as np
data = np.array([[1, 2], [3, 4], [5, 6]])
print(data)
```

Output

```
[[1 2]
 [3 4]
 [5 6]]
```

High Dimensional Arrays

Multidimensional Array concept can be explained as a technique of defining and storing the data on a format with more than two dimensions (2D). In Python, Multidimensional Array can be implemented by fitting in a list function inside another list function, which is basically a nesting operation for the list function.

Example:

```
import numpy as n
phd=[[1,2,3],[2,3,1],[3,2,1]]
print(hd)
```

Output:

```
[[1, 2, 3], [2, 3, 1], [3, 2, 1]]
```

```
import numpy as np
c = 4
r = 3
Array = [ [0] * c for i in range(r) ]
print(Array)
```

[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]

High Dimensional Arrays

How to create multi-dimensional arrays using NumPy

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(arr)
```

Output:

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

How to Access and Modify Multi-dimensional Arrays Using NumPy

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(arr[1, 2])
 arr[0, 3] = 20
print(arr)
```

Output

```
7
[[ 1  2  3 20]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

Thank You

NumPy Array Attributes

NumPy - Arrays - Attributes of a NumPy Array

NumPy array (ndarray class) is the most used construct of NumPy in Machine Learning and Deep Learning. Let us look into some important attributes of this NumPy array.

Let us create a Numpy array first, say, array_A.

Pass the above list to array() function of NumPy

array_A = np.array([[3,4,6], [0,8,1]])

Now, let us understand some important attributes of ndarray object using the above-created array array_A.

(1) ndarray.ndim

ndim represents the number of dimensions (axes) of the ndarray.

e.g. for this 2-dimensional array [[3,4,6], [0,8,1]], value of ndim will be 2. This ndarray has two dimensions (axes) - rows (axis=0) and columns (axis=1)

NumPy Array Attributes

(2) ndarray.shape

shape is a tuple of integers representing the size of the ndarray in each dimension.

e.g. for this 2-dimensional array [[3,4,6], [0,8,1]], value of shape will be (2,3) because this ndarray has two dimensions - rows and columns - and the **number of rows is 2** and the **number of columns is 3**

(3) ndarray.size

size is the total number of elements in the ndarray. It is equal to the product of elements of the shape. e.g. for this 2-dimensional array [[3,4,6], [0,8,1]], shape is (2,3), ***size will be product (multiplication) of 2 and 3 i.e. (2*3) = 6. Hence, the size is 6.***

NumPy Array Attributes

(4) ndarray.dtype

dtype tells the data type of the elements of a NumPy array. In NumPy array, all the elements have the same data type.

e.g. for this NumPy array [[3,4,6], [0,8,1]], **dtype will be int64**

(5) ndarray.itemsize

itemsize returns the size (in bytes) of each element of a NumPy array.

e.g. for this NumPy array [[3,4,6], [0,8,1]], itemsize will be 8, because this array consists of integers and size of integer (in bytes) is 8 bytes.

Introduction to Pandas

- Pandas is a Python library used for working with data sets. It is used for data analysis in Python and developed by **Wes McKinney** in 2008.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- [pandas] is derived from the term "panel data.
- Pandas is a valuable open-source library for Python, designed to streamline data science and machine learning tasks. It provides core structures and functions to simplify the process of manipulating and analyzing data.
- Pandas in python is an essential tool for analysts and developers in every field from economics and advertising to neuroscience and NLP.

Introduction to Pandas

Why use Pandas?

- ✔ Pandas allows us to analyze big data and make conclusions based on statistical theories.
- ✔ Pandas can clean messy data sets, and make them readable and relevant.
- ✔ Relevant data is very important in data science.
- ✔ Visualize the data with help from Matplotlib. Plot bars, lines, histograms, bubbles, and more.
- ✔ Store the cleaned, transformed data back into a CSV, other file or database

Introduction to Pandas

Key features of Pandas

- ✔ It has a fast and efficient DataFrame object with the default and customized indexing.
- ✔ Used for reshaping and pivoting of the data sets.
- ✔ Group by data for aggregations and transformations.
- ✔ It is used for data alignment and integration of the missing data.
- ✔ Provide the functionality of Time Series.
- ✔ Process a variety of data sets in different formats like matrix data, tabular heterogeneous, time series.
- ✔ Handle multiple operations of the data sets such as subsetting, slicing, filtering, groupBy, re-ordering, and re-shaping.
- ✔ It integrates with the other libraries such as SciPy, and scikit-learn.
- ✔ Provides fast performance, and If you want to speed it, even more, you can use the **Cython**.

Introduction to Pandas

The key features of Python Pandas

1. Data cleaning
2. Data filtering and selection
3. Data aggregation
4. Data visualization

The benefits of learning Pandas library

1. Efficient data handling
2. Flexibility
3. Easy integration with other libraries
4. Wide adoption and support
5. Readability
6. Handling diverse data sources

The applications of Pandas in Python

1. Data cleaning and preprocessing
2. Data exploration
3. Feature engineering
4. Time series analysis
5. Data science

Introduction to Pandas

What are examples of Pandas operations?

1. Load data from a CSV file
2. Selecting specific columns
3. Filtering rows based on a condition
4. Renaming columns
5. Grouping data by a specific column
6. Merging two DataFrames
7. Creating a line plot with Pandas and Matplotlib

Creating a series object

- Pandas is particularly useful when working with structured data, such as data in CSV or Excel files, SQL databases, or other structured formats.
- The library allows users to read in and manipulate data from these sources, perform calculations and transformations on the data, and generate visualizations or export the results to other formats.
- Two of the most commonly used data structures in Pandas are Series and DataFrames. A Series is a one-dimensional labeled array that can hold any data type. It is similar to a column in a spreadsheet or a dictionary in Python.
- A series can be seen as a one-dimensional array. The data structure can hold any data type, that is includes strings, integers, floats and Python objects.
- Each and every value in a Series is assigned to a label(assigned to an index), labels may be integer values or they may be a name representation.

Creating a series object

Pandas series

You can access the values and index of a Pandas Series using the .values and .index attributes, respectively.

```
import pandas as pd
# create a Series from a list of numbers
s = pd.Series([1, 2, 3, 4, 5])
# print the values and index of the Series
print(s.values)
print(s.index)
```

Output:

```
[1 2 3 4 5]
RangeIndex(start=0, stop=5, step=1)
```

Creating a series object

Pandas DataFrame

Pandas DataFrame is a two-dimensional labeled data structure that can hold multiple Series. It is a core data structure in Pandas, and is used extensively for data manipulation and analysis.

To create a Pandas DataFrame, you can use the pd.DataFrame() function and pass in a dictionary of lists or arrays.

Example

```
import pandas as pd
data = {'name': ['Alice', 'Bob', 'Charlie', 'David'],'age': [25, 30, 35, 40],'gender': ['F', 'M', 'M', 'M']}
df = pd.DataFrame(data)print(df)
```

```
      name  ...  gender
0    Alice  ...       F
1      Bob  ...       M
2  Charlie  ...       M
3    David  ...       M

[4 rows x 3 columns]
```

Thank You