

**INTERNSHIP**

**REPORT**

A

report

submitted

for

the

partial

fulfillment

for

the

award

of

the

degree

of

BACHELOR

OF

DATA SCSCIENCE

SCIENCE

SUBMITTED

BY

**SRIYA.U**

30823U48008

UNDER

THE

SUPERVISION

OF

Miss.

NANDINI K (PYTHON DEVELOPER)

TECHPOWER SOLUTIONS

CHENNAI

JUNE-2025

DEPARTMENT

OF

COMPUTER

SCIENCE

AND

APPLICATIONS

D.

K.M

COLLEGE

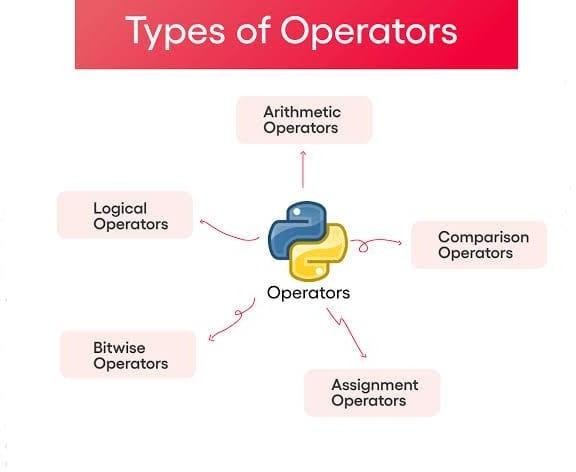
FOR

WOMEN(AUTONOMOUS)

VELLORE-632001

|  |  |  |
| --- | --- | --- |
| **CERTIFICATE**  This is to certify that the internship report is a bonafide work carried out by SRIYA.U  **(30823U48008)** in partial fulfillment for the award of **BACHELOR OF COMPUTER**  **SCIENCE AND APPLICATIONS,** of the **TECHPOWER SOLUTIONS CHENNAI.** This internship was attended through offline mode and submitted during May 2025-june 2025.  **Department Internship Coordinator Head of the Department**  **Principal Controller of Examination** | | |
| **DECLARATION**  I **SRIYA.U(30823U48008**) hereby declare that the Internship Report Submitted to the D.K.M College for Women(Autonomous), Vellore. In partial fulfillment of the requirements for the award of the Bachelor of Computer Applications is a record of original and independent project work prepared by me after completion of one month work at TECHPOWER SOLUTIONS, CHENNAI.  I also confirm that, the report is only for my academic requirement not for any other purpose.  PLACE :VELLORE  DATE :  SIGNATURE OF THE CANDIDATE  [SRIYA.U] | | |
| **ACKNOWLEDGEMENT**  I sincerely owe my gratitude to all the person who helped and guided me in completing this internship.  I am thankful to Dr.R.Banumathy m.a..,m.phil..,ph.d..,principal,D.K.M College for women(Autonomous)-Vellore, for giving the necessary environment to acquire knowledge and skill.And also I am thankful to respected secretary D.Maninathan b.e.., for the encouragement and for having permitted me to do this internship.  I take this opportunity to express my profound gratitude to Mrs.G.Sangeethalakshmi M.Sc(IT)..,M.Phil..,Assistant Professor and Head,Department of computer applications,  D.k.m college for women(autonomous),for the continuous encouragement and inspiration.  I take this opportunity to express my profound gratitude to Mrs.B.Arulmozhi M.Sc..,B.Ed..,M.phil..,SET.,NET., Assistant Professor and Head, Department of Computer Science, D.K.M College for Women(Autonomous), Vellore for her consistent guidance and enthusiasm for the successful completion of internship.  I am thankful to External Guide Ms. Nandini, Trainer, TECH POWER SOLUTIONS,chennai for her guidance, monitoring, continuous encouragement throughout the progress of this Internship.  I extend my gratitude to Internship co-ordinator Mrs. S.Shanthi M.C.A.., M.Phil.., SET.., Assistant Professor, Department of Computer Science and Applications, D.K.M College for Women(Autonomous) Vellore, for providing an opportunity to do the Internship, and thank her for supporting me at internship.  My sincere thanks to LIBRARIANS, other FACULTIES & FRIENDS, who deliberately provided their consistent support for completing my internship. Then I thank them all for their continuous support.  Lastly, I thank almighty, my parents and friends for their constant encouragement without which this internship would not be possible.  INDEX   |  |  |  | | --- | --- | --- | | S.NO | CONTENT | PAGE NO | | 1. | **Introduction:**   * 1. Basics of programming language   2. Python language history   3. Python applications and features |  | | 2. | **Installation of software:**   * 1. How to install PyCharm IDE community Edition 2024.1.1 |  | | 3. | **Basics:**   * 1. Keywords   2. Variables   3. Commands   4. Operators |  | | 4. | **Making statements:**   * 1. Decision making statements   2. Control statements |  | | 5. | **Standard data types:**   * 1. Numbers   2. String   3. List   4. Tuples   5. Dictionary |  | | 6. | **Python functions:**   * 1. Defining a function   2. Calling a function |  | | 7. | **Importing module:**   * 1. Import statements   2. from-import statement |  | | 8. | **Python oops concept:**   * 1. Object   2. Class   3. Method |  | | | |
| |  |  |  |  | | --- | --- | --- | --- | |  | 9. | **Python oops concept:**  9.1 Inheritance  9.2 Polymorphism |  | |  | 10. | **Python oops concept:**  10.1 Data Abstraction  10.2 Encapsulation |  | |  | 11. | **Filehandling:**  11.1 Opening a file  11.2 Closing a file  11.3 Reading a file  11.4 Writing a file |  | | . |  | Exception Handling |  | |  | 13. | ARITIFICAL INTELLIGENCE  13.1 Machine Learning  13.2 Deep learning. |  | |  | 14. | **Project:** |  | |  | 15. | **Conclusion** |  | |
| **WEEKLY OVERVIEW OF INTERNSHIP ACTIVITIES**   |  |  |  | | --- | --- | --- | | DATE | DAY | NAME OF THE TOPIC/MODULE COMPLETED | | 28/05/2025 | Wednesday | Introduction to python and installation of python | | 29/05/2025 | Thursday | Python syntax and variables | | 30/05/2025 | Friday | Python datatypes type casting and input function | | 31/05/2025 | Saturday | Python operators and control flow statements | | 02/05/2025 | Monday | Python data structures and loops in python | | 03/05/2025 | Tuesday | List ,Tuple,Set,Dictionary and Functions in Python | | 04/05/2025 | Wednesday | Lambda functions and Recursion | | 05/05/2025 | Thursday | File handling and object-oriented programming in Python | | 06/05/2025 | Friday | Inheritance, Polymorhism,Encapsulation | | 09/05/2025 | Monday | Python libraries overview | | 10/05/2025 | Tuesday | Introduction to Artificial Intelligence and its evolution | | 11/05/2025 | Wednesday | Types of Artificial intelligence | | 12/05/2025 | Thursday | Machine learning and its algorithm | | 13/05/2025 | Friday | Deep learning and Natural Language processing CNN, RNN | | 14/05/2025 | Saturday | Real life AI applications | | | |
| **INTRODUCTION**  Python is a high level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It users English Keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.   **Python is Interpreted**:   * Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. * This is similar to PEARL and PHP.  **Python is Interactive**: * You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.    **Python is object-oriented**:   Python supports object-oriented style or technique of programming that encapsulates code within objects.   **Python is a Beginner’s Language**:   Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games. |
| **HISTORY**   * Python was developed by Guido van Rossum in the last eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the   Netherlands.   * Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol68, SmallTalk, Unix shell, and other scripting languages. * At the time when he began implementing Python, Guido van Rossum was also reading the published scripts from "Monty Python's Flying Circus" (a BBC comedy series from the seventies, in the unlikely case you didn't know). It occurred to him that he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python. * Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress. * Python 1.0 wat released on 20 February, 1991. * Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle detecting garbage collector and support for Unicode. With this release the development process was changed and became more transparent and community-backed. * Python 3.0 (which early in its development was commonly referred to as Python 3000 or py3k), a major, backwards- incompatible release, was released on 3 December 2008 after a long period of testing. Many of its major features have been back ported to the backwards-compatible Python 2.6.x and 2.7.x version series. * In January 2017 Google announced work on a Python 2.7 to go trans compiler, which The Register speculated was in response to Python 2.7's planned end-of-life.   APPLICATIONS:   * + - 1. Console based application       2. Audio (or) Video based applications       3. 3D CAD applications       4. Web applications       5. Enterprise applications       6. Applications for images   2 | | |
| **FEATURES:**  Python's features include:   * **Easy-to-learn**: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly. * **Easy-to-read**: Python code is more clearly defined and visible to the eyes. * **Easy-to-maintain**: Python's source code is fairly easy-to- maintain. * **A broad standard library**: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh. * **Interactive Mode**: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code**.** * **Portable**: Python can run on a wide variety of hardware platforms and has the same interface on all platforms. * **Extendable**: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient. * **Databases**: Python provides interfaces to all major commercial databases. * **GUI Programming**: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of UNIX. * **Scalable**: Python provides a better structure and support for large programs than shell scripting. | | |
| **INSTALLATION OF PYTHON**  **How to install Python PyCharm on Windows?**  Python is a programming language that lets you work quickly and integrate systems  more efficiently. We need to have an interpreter to interpret and run our programs. There are certain online interpreters like GFG-IDE, IDEONE Code Pad, etc. Running Python codes on an offline interpreter is much more compatible than using an online IDE. In this article, we will learn How to install Python PyCharm on Windows.  PyCharm is one of the most popular Python-IDE developed by JetBrains and used  for performing scripting in Python language. PyCharm provides some very useful features like Code completion and inspection, Debugging process, support for various programming frameworks such as Flask and Django, Package Management, etc.  PyCharm provides various tools for productive development in Python.  **How to install Python IDE?**   Before, starting with the process of Installing Python IDE PyCharm in Windows, one must ensure that Python is installed on their system. To check if the system is equipped with Python, go to the Command line( window button + R).  **How to Install PyCharm?**  **Here is a process to download PyCharm :**  Step-1: Go to website official website of JetBrains https://[www.jetbrains.com/pycharm/download/](http://www.jetbrains.com/pycharm/download/) and click on the “DOWNLOAD” link of the Community section.  Step-2: After clicking on Download Click on Next  Step-3: After Click on Next , You need to choose the destination folder according to your choice. | | |
| Step-4: Choose options of installation according to you choice.  **Getting Started with PyCharm:**  Once the Installation is over, PyCharm can be searched and started from the Start Menu.  Follow the steps given below to do the same:   * Searching from Start Menu: * Getting done with License Agreement: * Setting UI Theme: * Downloading Plugins: * Get Started with PyCharm:   **BASICS OF PYTHON KEYWORDS:**   The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.   |  |  |  |  | | --- | --- | --- | --- | | elif | Assert | Finally for | Or | | Else | Except | Lambda | Yield | | And | In | While | Del | | As | Return | Continue | Global | | If | Break | Not | Pass | | Import | Is | With | For | | Print | Try | Def | not | | Raise | Class | From | Exec | | | |
| **VARIABLES:**  All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.  The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python-Global variables & Local variables.  **Global vs. Local variables:**  Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.  This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions.  When you call a function, the variables declared inside it are brought into scope.  Following is a simple example :  Total= 0; # This is global variable. #  Function definition is here def sum arg1, arg2):  # Add both the parameters and return them." total arg1 + arg2; # Here total is local variable. print "Inside the function local total:”, total return total; sum( 10, 20);  print "Outside the function global total:", total Result: Inside the function local total: 30 Outside the function global total: 0  **COMMENTS**  There are two types of comments in python:  **line comments:** start with # symbol Aline starting with a # is treated as a comment Ex: # | | |

````````````````````



To

find

sum

of

two

numbers.

**Multi**

**line**

**comments:**

when

we

want

to

make

several

lines

as

comment,

then

writing

#

symbol

in

the

beginning

of

every

line.

#

This

is

my

first

program

#

It

adds

two

numbers

**OPERATORS**

Operators

refer

to

special

symbols

that

perform

operations

on

values

and

variables.

**Arithmetic**

**operators:**

Some

basic

arithmetic

operators

are

,

+

-

,

\*,

/,

%,

\*\*,

and

//.

You

can

apply

these

operators

on

numbers

as

well

as

variables

to

perform

corresponding

operations.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Operator** | **Description** | **Example** | | +Addition | Adds values on either side of the operator. | a + b =30 | | -Subtraction | Subtracts right hand operand from left hand operand. | a - b = -10 | | \*Multiplication | Multiplies values on either side of the operator. | a \* b = 300 | | /Division | Divides left hand operand by right hand operand. | b/a = 3 | | %Modulus | Divides left hand operand by right hand operand and returns remainder. | b/a = 0 | | \*\*Exponent | Performs exponential (power)  calculation on operators. | a\*\*b = 10 to the power  20 | | //Floor Division | The division of operands where the result is the quotient in which the digits after the decimal point are removed. | 9//2 = 4 and  9.0//2.0 = 4.0 |   **Program:**  a=21 b=10 print "Addition is", a +b print  "Subtraction is", a-b print "Multiplication is", a \*b print "Division is", a/b print "Modulus is", a% b a=2 b=3  print "Power value is", a \*\*b a=10  8 |
| b=4  print "Floor Division is ", a // b  **Output:**  Addition is 31 Subtraction is II  Multiplication is 210  Division is 2  Modulus is 1 Power value is 8 Floor Division is 2  **Comparison (or) Relational operators:**  These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.   |  |  |  | | --- | --- | --- | | **Operator** | **Description** | **Example** | | = = | If the values of two operands are equal, then the condition becomes true. | (a == b) is not true. | | != | If values of two operands are not equal, then condition becomestrue. | (a!=b) is true. | | <> | If values of two operands are not equal, then condition becomes true. | (a<>b) is true. This is similar to !=operator. | | > | If the value of left operand is greater than the value of right operand, then condition becomes true. | (a>b) is not true. | | < | If the value of left operand is less than the value of right operand, then condition becomes true. | (a<b) is true. | | >= | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. | (a>=b) is not true. | |
| |  |  |  | | --- | --- | --- | | <= | If the value of left operand is less than or equal to the value of right operand, then condition becomes true. | (a<=b) is true. |   **Example:**  a=20 b=30 if a<b:  print "b is big" elif a > b: print "a is big" else:  print "Both are equal"  **Output:**  b is big.  **Example:**  a=82 b=27 a+=b  print a a=25 b=12 a-=b print a a=24 b=4 a\*=b print a  10 |

a=4

b=6

a\*\*=b

print

a

**Output:**

109

13

96

4096

**Logical**

**operators:**

**Operator**

**Description**

**Example**

And

Logical

AND

If

both

the

operands

are

true

then

condition

becomes

true.

(

a

and

b)

is

true

Or

Logical

OR

If

any

of

the

two

operands

are

non-

zero

then

condition

becomes

true.

(

a

or

b)

is

true

not

Logical

NOT

Used

to

reverse

the

logical

state

of

its

operand.

Not

a

(

and

b)

is

false

**Example:**

a=20

b=10.

C=30

if

a>=b

and

a>=c:

print

"a

is

big"

elif

b>=a

and

b

>=

c:

print

"b

is

big"

else:

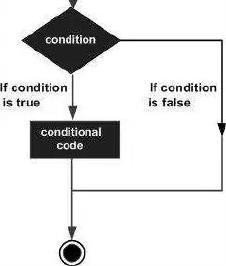
print

"c

is

big"

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **output:** c is big.  **Membership operators:**  Python's membership operators test for membership in a sequence, such as strings, lists, or tuples.   |  |  |  | | --- | --- | --- | | **Operator** | **Description** | **Example** | | In | Evaluates to true if it finds a variable in the specified sequence and false otherwise. | x in y, here in results in a I if x is a member of sequence  y. | | Not in | Evaluates to true if it does not finds avariable in the specified sequence and false otherwise. | x not in y, here not in results in a l if x is not a member of sequence y. |   **Example:**  a=3 list= [1, 2, 3, 4, 5]; if (a in list):  print "available" else:  print " not available"  **Output:**  Available  **Identity operators:**  Identity operators compare the memory locations of two objects. |
| |  |  |  | | --- | --- | --- | | **Operator** | **Description** | **Example** | | Is | Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. | x is y, here is results in 1 if id(x) equals id(y). | | Is not | Evaluates to false if the variables on either side of the operator point to the same object and true otherwise. | x is not y, here is not results in 1 if id(x) is not equal to id(y). |   **Example:**  a=20 b=20 if (a is b):  print "Line 1- a and b have same identity" else:  print "Line 1 - a and b do not have same identity" if (id(a)==id(b)): print "Line 2-a and b have same identity" else:  print "Line 2- a and b do not have same identity"  **Output:**  print "Line 1 - a and b have same identity" print "Line 2-a and b have same identity" | |



**MAKING**

**STATEMENTS**

**Decision**

**making**

**statements:**

Decision

making

is

anticipation

of

conditions

occurring

while

execution

of

the

program

and

specifying

actions

taken

according

to

the

conditions.

Decision

structures

evaluate

multiple

expressions

which

produce

True

or

False

as

outcome.

You

need

to

determine

which

action

to

take

and

which

statements

to

execute

if

outcome

is

True

or

False

otherwise.

Following

is

the

general

form

of

a

typical

decision

making

structure

found

in

most

of

the

programming

languages:

Python

programming

language

assumes

any

non-zero

and

non-null

values

as

True,

and

if

it

is

either

zero

or

null,

then

it

is

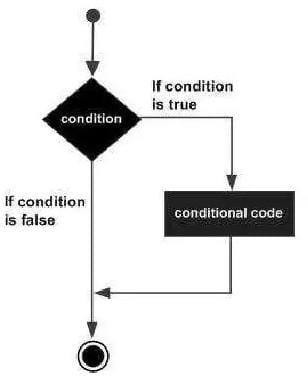
assumed

as

False

value.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Statement** | **Description** | | If statements | if statement consists of a Boolean expression followed by one or more statements. | | If…else statements | if statement can be followed by an optional else statement, which executes when the Boolean expression is FALSE. | | Nested if statements | You can use one if or else if statement inside another if or else if statement(s). |   **The if statement:**  It is similar to that of other languages. The if statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison. |



**Syntax:**

If

condition

First,

the

condition

is

tested.

If

the

condition

is

True,

then

the

statements

given

after

colon

(:)

are

executed.

We

can

write

one

or

more

statements

after

colon

(:).

**Example:**

a=10

b=15

if

a

<

b

:

print

"B

is

big"

print

"B

value

is",

b

**Output:**

B

is

big

B

value

is

15

**The**

**if**

**...**

**else**

**statement:**

An

else

statement

can

be

combined

with

an

if

statement.

An

else

statement

contains

the

block

of

code

that

executes

if

the

conditional

expression

in

the

if

statement

resolves

to

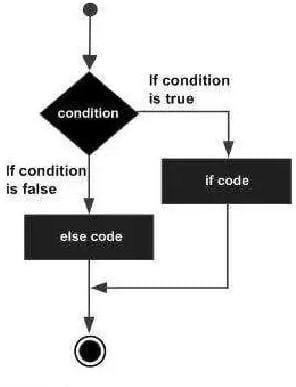
0

or

a

FALSE

value.



The

else

statement

is

an

optional

statement

and

there

could

be

at

most

only

one

else

statement

following

if.

**Syntax:**

If

condition:

statement(s)

else:

statement(s)

**Example:**

a=48

b=34

if

a<b:

print

"B

is

big"

print

"B

value

is",b

else:

print

"A

is

big"

print

"A

value is",

a

print

"END"

**Output:**

A

is

big

A

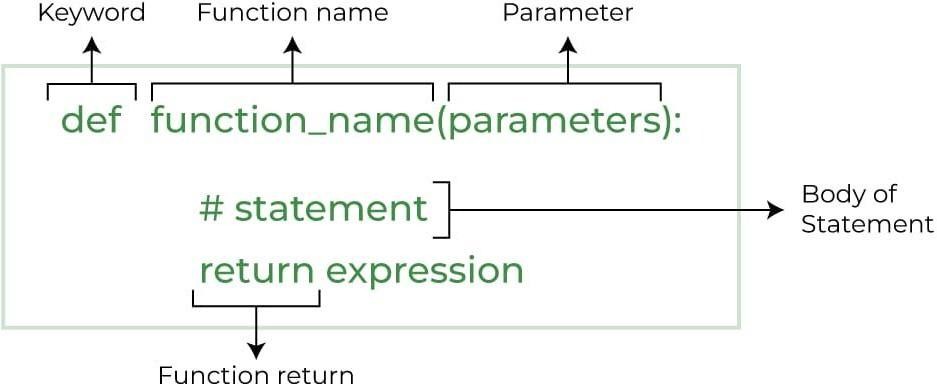
value

is

48

END

|  |
| --- |
| **Q) Write a program for checking whether the given number is even or not.?**  **Program:**  a=input("Enter a value: ") if a%2==0:  print "a is EVEN number" else:  print "a is NOT EVEN Number"  **Output-1:**  Enter a value: 56 This is EVEN Number **Output-2:** Enter a value:27  This is NOT EVEN Number  **The Elif Statement:**  The elif statement allows you to check multiple expressions for True and execute a block of code as soon as one of the conditions evaluates to True.  Similar to the else, the elif statement is optional. However, unlike else, for which there can be at most one statement, there can be an arbitrary number of elif statements following an if.  **Syntax:**  if condition 1: statement(s)  elif condition2: statement(s) else:  statement(s)  **Example:** a=20 b=10 |
| c=30  if a>b and a>c: print "a is big"  elif b>=a and b >= c: print  "b is big" else: print "c is big"  **Output:** c is big  **Control statements:**  Control statements are statements which control or change the flow of execution. The following are the control statements available in Python:   * if statement * if... else statement * if... elif...else statement * while loop * for loop * break statement * continue statement   **The if Statement:**  This statement is used to execute one or more statements depending on whether a condition is True or not. The syntax or correct format of if statement is given below:  If condition:  Statements |
| First, the condition is tested. If the condition is True, then the statements given after colon (:) are executed. We can write one or more statements after colon (1). If the condition is False, then the statements mentioned after colon are not executed.  **Example:** num=1 if num --1:  print("one")  **Output:**  One  We can also write a group of statements after colon. The group of statements in Python is called a suite. While writing a group of statements, we should write them all with proper indentation.  Indentation represents the spaces left before the statements.  **Example:**  str='yes'  if str == "yes": print  ("yes")  print ("This is your response")  **Output:**  yes  This is your response.  Observe that every print () function mentioned after colon is starting after 4 spaces only. When we write the statements with the same indentation are belonging to the same group. |
| **The if... else Statement:**  The if else statement executes a group of statements when a condition is True; otherwise, it will execute another group of statements.  **Syntax:**  If condition:  Statements 1 Else: statements2  If the condition is True, then it will execute statements1 and if the condition is False, then it will execute statements2. It is advised to use 4 spaces as indentation before statements and statements2.  In Program 4, we are trying to display whether a given number is even or odd. The logic is simple. If the number is divisible by 2, then it is an even number, otherwise, it is an odd number.  To know whether a number is divisible by 2 or not, we can use the modulus operator (%). This operator gives the remainder of division. If the remainder is 0, then the number is divisible, otherwise not.  **Example:**  X=10  If x % 20:  Print (x, "is even number") else: print (x, is odd number")  **Output:**  10 is even number.  **The if..elif…else statement:**  Sometimes, the programmer has to test multiple conditions and execute statements depending on those conditions. If... elif, else statement is useful in such situations. |
| **Syntax**:  **if condition1:** Statements 1  **elif condition2** Statements2  **elif condition3:**  statements3 else: statements4  When condition 1 is True, the statements will be executed. If condition1 is False, the condition2 is evaluated. When condition2 is True, the statement 2 will be executed When condition2 is False, the condition 3 is tested.  If condition 3 is True, then statements will be executed. When condition3 is False, the statements will be executed. It means statements 4 will be executed only if none of the conditions are True Observe colon (:) after each condition.  **Example:** num=-5 if num==0:  print (num, "is zero")  elif num>0:  print (num, "is positive") else: print (num, "is negative")  **Output:**  -5 is negative.  **The while loop:**  A statement is executed only once from top to bottom. For example, if is a statement that is executed by a Python interpreter only once. But a loop is useful to execute repeatedly. For |
| example, while and for are loops in Python. They are useful to execute a group of statements repeatedly several times.  The while loop is useful to execute a group of statements several times repeatedly depending on whether a condition is True or False. The **syntax** or format of while loop is:  while condition:  statements  Here, statements represent one statement or a suite of statements. Python interpreter first checks the condition. If the condition is True, then it will execute the statements written after colon (:) After executing the statements, it will go back and check the condition again.  If the condition is again found to be True, then it will again execute the statements. Then it will go back to check the condition once again.  In this way, as long as the condition is True, the Python Interpreter executes the statements again and again. Once the condition is found to be False, then it will come out of the while loop.  In the following program, we are using a while loop to display numbers from 1 to 10. Since, we should start from 1, we will store "1' into a variable x. Then we will write a while loop as: while x<=10:  Observe the condition. It means, as long as x value is less than or equal to 10, continue the while loop. Since we want to display 1,2,3,... up to 10, we need this condition.  To display x value, we can use: print(x)  Once this is done, we have to increment x value by 1, by writing x+1.  **Example:**  X=1  While x<=10:  Print(x)  X+=1  Print(“End”) |
| **Output:**  1  2  3  4  5  6  7  8  9  10  End  **The for Loop:**  The for loop is useful to iterate over the elements of a sequence. It means, the for can be used to execute a group of statements repeatedly depending upon the numb elements in the sequence.  The for loop can work with sequences like string, list, tuple, range etc.  The **syntax** of the for loop is given below: for var in sequence:  statements  The first element of the sequence is assigned to the variable written after for and then the statements are executed.  Next, the second element of the sequence is assigned to the variable and then the statements are executed a second time.  In this way, for each element of the sequence, the statements are executed once. So, the for loop is executed as many times as there are a number of elements in the sequence.  24 |
| **Example:** Str = ‘Hello’  For ch in str: Print (ch)  **Output:**  p y  t  h o n  In the above program, the string 'str' contains Hello'. There are 6 characters in the string.  The for loop has a variable 'ch'. First of all, the first character of the string, i.e. H' is stored into ch and the statement, i.e. print (ch) is executed.  As a result, it will display H'. Next, the second character of the string,  i.e. 'e' is stored into ch. Then print (ch)is once again executed and 'e' will be displayed. In the third iteration, the third character1 will be displayed.  In this way the for loop is executed for 6 times and all the 6 characters are displayed in the output.  In the program below, we are using the for loop to display the elements of the string using index'. An index represents the position number of elements in the string or sequence.  For example, str[0] represents the 0th character, i.e. 'H' and str[1] represents the first character, i.e. 'e', and soon.  Hence we can take an index 1' that may change from 0 to n 1 where 'n' represents the total number of elements.  We can use a range(n) object that generates numbers from 0 to n-1. |
| **Example:**  Str = ‘python’ n=len(str) for I in range(n)  print(str[i])  **Output:**  **p y**  **t**  **h o**  **N**  Here, 'n' value will be 6 as the length of the string is 6. We used range in that gives the numbers from 0 to n-1. Hence for loop repeats from 0 to 5and print() function displays str[0], string, str[5]It means H.e.l.l.o will be displayed.  So note that the len() function gives the total number of elements in a sequence like string, list or tuple. The range object, also known as range() function in Python, is useful to provide a sequence of numbers. range(n) gives numbers from 0 to n-1.  For example, if we write range(10), it will return numbers from  0 to 9. We can also mention the starting number, ending number, as range(5, 10).In this case, it will give numbers from 5 to 9. We can also specify the step size.  The step size represents the increment in the value of the variable at each step. For example, range(1, 10, 2)will give numbers from 1 to 9 in steps of 2.  That means, we should add 2 to each number to get the next number. Thus, it will give the numbers: 1,3, 5, 7 and 9. Let's write a program to display numbers from 10 to1 in descending order using the range)object.  In this case, we use range(10, 0, -1). Here, the starting number is 10 and the ending number is one before 0. The step size is -1. |
| It means we should decrement 1 every time. Thus we will get 10, 9, 8, up to 1. Observe that it will not return 0 as the last number. It will stop at 1 that is one number before 0.  **Example:**  For x in range (5,0, -1):  Print (x)  **Output:**  5  4  3  2  1  We will use a for loop to access the elements of a list. As we know, a list is a sequence that contains a group of elements. It is like an array. But the difference is that an array stores only one type of element; whereas, a list can store different types of elements.  **Q.write a program to create a list and retrieve elements from the list using a for loop.** List = [10, 20.5, ‘A’, ‘London’] for element in list:  print(element) **Output:**  10  20.5  A  London.  In the above program, each element of the list is stored into the variable 'element. It means, in the first iteration, element stores 10. In the second iteration, it stores 20.5. |
| In the third iteration, it stores 'A' In the fourth iteration, it stores 'London'. Hence, print(element) displays all the elements of the list.  **Program:**  List=[10,20,30,40,50]  Sum=0  For i in list:  Print (i)  Sum+= i  Print(‘sum= ‘,sum)  **Output:**  10  20  30  40  50  Sum=150.  Observe that there are only two statements in the for loop. They are:  print(i)  sum+=i  **The break Statement**:  The break statement can be used inside a for loop (or) while loop to come out of the loop. When 'break' is executed, the Python interpreter jumps out of the loop to process the next statement in the program.  We have already used break inside for loop in previous program. When the element is found, it would break the for loop and comes out. We can also use break inside a while loop.  Suppose, we want to display numbers from10 to 1 in descending order using a while loop.  For this purpose, we can write a simple while loop as:  28 |
| **Example:** x = 10  while x>=1:  print(‘x=’,x) print(“out of the loop”)  **Output:** x= 10 x= 9 x= 8 x= 7 x= 6 x= 5 x= 4 x= 3 x= 2 x= 1  Out of Loop.  **The continue statement:**  The continue statement is used in a loop to go back to the beginning of the loop. It means, when continue is executed, the next repetition will start.  When continue is executed, the subsequent statements in the loop are not executed.  In the following program, the while loop repeats for 10times from 0 to 9. Every time. ‘x' | |
| value is displayed by the loop.  When x value is greater than 5, continue is executed that makes the Python interpreter go back to the beginning of the loop.  Thus the next statements in the loop are not executed. As a result, the numbers up to 5 are only displayed.  **Program:**  X = 0  While x < 10:  x+=1  If x > 5:  Continue print (‘x= ‘,x)  print (“out of loop”) **Output:** x= 1 x=2 x=3 x=4 x=5 out of Loop.  **DATATYPES IN PYTHON**  Python Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data.  Since everything is an object in Python programming, Python data types are classes and variables are instances (objects) of these classes. The following are the standard or built-in data types in Python:   Numeric:   * Integer * Float * Complex number |
| * Sequence Type:   + Strings   + List   + Tuple * Dictionary   **Numeric Data Types in Python:**  The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number.  These values are defined as Python int, Python float, and Python complex classes in Python.  **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). In Python, there is no limit to how long an integer value can be.  **Float** – This value is represented by the float class. It is a real number with a floating-point representation. It is specified by a decimal point.  **Complex Numbers** – A complex number is represented by a complex class. It is specified as (real part) + (imaginary part) j.  For example – 2+3j In this type() function is used to determine the type of Python data type.  **Example:**  This code demonstrates how to determine the data type of variables in Python using the type() function.  It prints the data types of three variables: a (integer), b (float), and c (complex). The output shows the respective data type Python for each variable.  **Program:**  a = 6  print ("Type of a: ", type(a)) b = 4.0 print ("\nType of b: ", type(b)) c = 2 + 3j print ("\nType of c: ", type(c)) |
| **Output:**  Type of a: <class 'int'>  Type of b: <class 'float'>  Type of c: <class 'complex'>  **Sequence Data Types in Python:**  The sequence Data Type in Python is the ordered collection of similar or different Python data types.  Sequences allow storing of multiple values in an organized and efficient fashion.  There are several sequence data types of Python:   * Python String * Python List * Python Tuple   **String Data Type:**  Strings in Python are arrays of bytes representing Unicode characters.  A string is a collection of one or more characters put in a single quote, double-quote, or triple-quote.  In Python, there is no character data type Python, a character is a string of length one. It is represented by str class.  **Creating a String:**  Strings in Python can be created using single quotes, double quotes, or even triple quotes.  **Example:**  This Python code showcases various string creation methods. It uses single quotes, double quotes, and triple quotes to create strings with different content and includes a multiline string. The code also demonstrates printing the strings and checking their data types.  32 |
| **Program:**  String1 = 'Welcome to the Geeks World' print("String with the use of Single Quotes: ")  print(String1)  String1 = "I'm a Geek" print("\nString with the use of Double Quotes: ")  print(String1)  print(type(String1))  String1 = '''I'm a Geek and I live in a world of "Geeks"''' print("\nString with the use of Triple Quotes: ")  print(String1) print(type(String1))  String1 = '''Geeks  For  Life'''  print("\nCreating a multiline String: ") print(String1) **Output:**  String with the use of Single Quotes:  Welcome to the Geeks World  String with the use of Double Quotes:  I'm a Geek  <class 'str'>  String with the use of Triple Quotes:  I'm a Geek and I live in a world of "Geeks"  <class 'str'> |
| Creating a multiline String:  Geeks  For  Life  **Accessing elements of String:**  In Python programming, individual characters of a String can be accessed by using the method of Indexing.  Negative Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character, and so on.  **Example:**  This Python code demonstrates how to work with a string named ‘String1′. It initializes the string with “ILoveKerala” and prints it.  It then showcases how to access the first character (“G”) using an index of 0 and the last character (“s”) using a negative index of -1.  **Program:**  String1 = "ILoveKerala " print("Initial String:  ") print(String1) print("\nFirst character of String is: ") print(String1[0])  print("\nLast character of String is: ") print(String1[-1]) **Output:**  Initial String:  ILoveKerala  First character of String is:  I  Last character of String is:  a |
| **List Data Type:**  Lists are just like arrays, declared in other languages which is an ordered collection of data.  It is very flexible as the items in a list do not need to be of the same type.  **Creating a List in Python:**  Lists in Python can be created by just placing the sequence inside the square brackets[].  **Example:**  This Python code demonstrates list creation and manipulation. It starts with an empty list and prints it. It creates a list containing a single string element and prints it.  It creates a list with multiple string elements and prints selected elements from the list.  It creates a multi-dimensional list (a list of lists) and prints it. The code showcases various ways to work with lists, including single and multi-dimensional lists.  **Program:** List = []  print("Initial blank List: ") print(List) List = ['MeatAndEat']  print("\nList with the use of String: ") print(List) List = ["Meat", "And", "Eat"]  print("\nList containing multiple values: ") print(List[0]) print(List[2])  List = [['Meat', 'And'], ['Eat']] print("\nMulti-  Dimensional List: ") print(List)  **Output:**  Initial blank List:  35 |
| with the use of String:  ['MeatAndEat']  List containing multiple values:  Meat Eat  Multi-Dimensional List:  [['Meat', 'And'], ['Eat']]  **Python Access List Items:**  In order to access the list items refer to the index number. Use the index operator [ ] to access an item in a list.  In Python, negative sequence indexes represent positions from the end of the array.  Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3].  Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.  **Program:**  List = ["Great", "For", "Great"] print("Accessing element from the list") print(List[0]) print(List[2]) print("Accessing element using negative indexing") print(List[-1]) print(List[-3])  **Output:**  Accessing element from the list:  Great  Great  Accessing element using negative indexing:  Great  Great |
| **Tuple Data Type:**  Just like a list, a tuple is also an ordered collection of Python objects. The only difference between a tuple and a list is that tuples are immutable i.e. tuples cannot be modified after it is created. It is represented by a tuple class.  **Creating a Tuple in Python:**  In Python Data Types, tuples are created by placing a sequence of values separated by a ‘comma’ with or without the use of parentheses for grouping the data sequence.  Tuples can contain any number of elements and of any datatype (like strings, integers, lists, etc.).  Tuples can also be created with a single element, but it is a bit tricky.  Having one element in the parentheses is not sufficient, there must be a trailing ‘comma’ to make it a tuple.  **Example:**  This Python code demonstrates different methods of creating and working with tuples. It starts with an empty tuple and prints it. It creates a tuple containing string elements and prints it. converts a list into a tuple and prints the result. It creates a tuple from a string using the tuple() function. It forms a tuple with nested tuples and displays the result.  **Program:** Tuple1 = ()  print("Initial empty Tuple: ") print(Tuple1) Tuple1 = ('Geeks', 'For')  print("\nTuple with the use of String: ") print(Tuple1) list1 = [1, 2, 4, 5, 6]  print("\nTuple using List: ") print(tuple(list1)) Tuple1 = tuple('Geeks')  print("\nTuple with the use of function: ") print(Tuple1) Tuple1= (0, 1, 2, 3)  Tuple2 = ('python', 'geek')  Tuple3 = (Tuple1, Tuple2) |
| print("\nTuple with nested tuples: ") print(Tuple3)  **:**  Initial empty Tuple:  ()  Tuple with the use of String:  ('Geeks', 'For') Tuple using List:  (1, 2, 4, 5, 6)  Tuple with the use of function:  ('G', 'e', 'e', 'k', 's')  Tuple with nested tuples:  ((0, 1, 2, 3), ('python',  'geek'))  **Access Tuple Items:**  In order to access the tuple items refer to the index number. Use the index operator [ ] to access an item in a tuple. The index must be an integer. Nested tuples are accessed using nested indexing.  The code creates a tuple named ‘tuple1′ with five elements: 1, 2, 3, 4, and 5. Then it prints the first, last, and third last elements of the tuple using indexing.  **Program:**  tuple1 = tuple([1, 2, 3, 4, 5,6]) print("First element of tuple") print(tuple1[0]) |
| print("\nLast element of tuple") print(tuple1[-1])  print("\nThird last element of tuple")  print(tuple1[-3])  **Output:**  First element of tuple:  1  Last element of tuple:  6  Third last element of tuple:  3  **Dictionary Data Type in Python:**  A dictionary in Python is an unordered collection of data values, used to store data values like a map, unlike other Python Data Types that hold only a single value as an element.  A Dictionary holds a key: value pair. Key-value is provided in the dictionary to make it more optimized.  Each key-value pair in a Dictionary is separated by a colon : , whereas each key is separated by a ‘comma’.  **Create a Dictionary in Python:**  In Python, a Dictionary can be created by placing a sequence of elements within curly {} braces, separated by ‘comma’.  Values in a dictionary can be of any datatype and can be duplicated, whereas keys can’t be repeated and must be immutable.  The dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing it in curly braces{}. Note – Dictionary keys are case sensitive, the same name but different cases of Key will be treated distinctly |
| **Example:**  This code creates and prints a variety of dictionaries. The first dictionary is empty.  The second dictionary has integer keys and string values.  The third dictionary has mixed keys, with one string key and one integer key.  The fourth dictionary is created using the dict() function, and the fifth dictionary is created using the [(key, value)] syntax  **Program:** Dict = {}  print("Empty Dictionary: ") print(Dict) Dict = {1: 'Goods', 2: 'For', 3: 'Goods'} print("\nDictionary with the use of Integer Keys: ") print(Dict) Dict = {'Name': 'Goods', 1: [1, 2, 3, 4]}  print("\nDictionary with the use of Mixed Keys: ") print(Dict) Dict = dict({1: 'Goods', 2: 'For', 3: 'Goods'}) print("\nDictionary with the use of dict(): ")  print(Dict) Dict = dict([(1, 'Goods'), (2, 'For')]) print("\nDictionary with each item as a pair: ") print(Dict) **Output:**  Empty Dictionary:{}  Dictionary with the use of Integer Keys:  {1: 'Goods', 2: 'For', 3: 'Goods'} Dictionary with the use of Mixed Keys:  {1: [1, 2, 3, 4], 'Name': 'Goods'}  Dictionary with the use of dict():  {1: 'Goods', 2: 'For', 3: 'Goods'} Dictionary with each item as a pair: {1: 'Goods', 2: 'For'} |
| **Accessing Key-value in Dictionary:**  In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets. There is also a method called get() that will also help in accessing the element from a dictionary.  **Example:**  The code in Python is used to access elements in a dictionary. Here’s what it does, It creates a dictionary Dict with keys and values as {1: ‘Geeks’, ‘name’: ‘For’, 3: ‘Geeks’}.  It prints the value of the element with the key ‘name’, which is ‘For’. It prints the value of the element with the key 3, which is ‘Geeks’.  **Program:**  Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'} print("Accessing a element using key:") print(Dict['name'])  print("Accessing a element using get:") print(Dict.get(3))  **Output:**  Accessing a element using key:  For  Accessing a element using get:  Geeks  **PYTHON FUNCTIONS**  Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function.  so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.  **Some Benefits of Using Functions:** |





Increase

Code

Readability



Increase

Code

Reusability

**Python**

**Function**

**Declaration:**

**Syntax:**

**Types**

**of**

**Functions**

**in**

**Python:**



Built-in

library

function:

These

are

Standard

functions

in

Python

that

are

available

to

use.



User-defined

function:

We

can

create

our

own

functions

based

on

our

requirements.

**Creating**

**a**

**Function**

**in**

**Python:**

We

can

define

a

function

in

Python,

using

the

def

keyword.

We

can

add

any

type

of

functionalities

and

properties

to

it

as

we

require.

By

the

following

example,

we

can

understand

how

to

write

a

function

in

Python.

In

this

way

we

can

create

Python

function

definition

by

using

def

keyword.

**Example:**

def

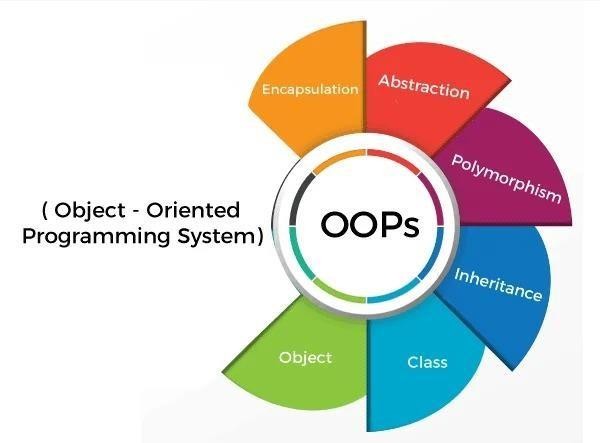
fun():

print("Welcome

to

INDIA")

|  |
| --- |
| **Calling a Function in Python:**  After creating a function in Python we can call it by using the name of the functions Python followed by parenthesis containing parameters of that particular function.  **Example:**  # A simple Python function  def fun(): print("Welcome to INDIA")  **Output:**  Welcome to INDIA  **Python Function with Parameters:**  If you have experience in C/C++ or Java then you must be thinking about the return type of the function and data type of arguments. **Syntax:**  def function\_name(parameter: data\_type) -> return\_type: """Docstring"""  # body of the function return expression **Example:**  def add(num1: int, num2: int) -> int:  """Add two numbers""" num3 = num1 + num2  return num3  num1, num2 = 14, 23 ans =  add(num1, num2)  print(f"The addition of {num1} and {num2} results {ans}.")  **Output:**  and 23 results 37.  43 |
| **Python Function Arguments:**  Arguments are the values passed inside the parenthesis of the function. A function can have any number of arguments separated by a comma(,).  **Example:**  check whether the number passed as an argument to the function is even or odd.  # whether x is even or odd def evenOdd(x): if (x % 2 == 0):  print("even") else:  print("odd") evenOdd(2) evenOdd(3) **Output:** even odd  **Types of Python Function Arguments:**  Python supports various types of arguments that can be passed at the time of the function call.  In Python, we have the following function argument types in Python:   * Default argument * Keyword arguments (named arguments) * Positional arguments * Arbitrary arguments (variable-length arguments \*args and\*\*kwargs)   **Docstring:**  The first string after the function is called the Document string or Docstring in short. This is used to describe the functionality of the function. The use of docstring in functions is optional but it is considered a good practice. | |
| **Syntax:**  print(function\_name.\_doc\_)  **Example:**  Adding Docstring to the function  # A simple Python function to check # whether x is even or odd def evenOdd(x):  """Function to check if the number is even or odd""" if (x % 2 == 0): print("even") else: print("odd") print(evenOdd.\_doc\_)  **Output:**  Function to check if the number is even or odd  **IMPORTINGMODULE**  A python module can be defined as a python program file which contains a python code including python functions, class, or variables. In other words, we can say that our python code file saved with the extension (.py) is treated as the module.  We may have a runnable code inside the python module.  Modules in Python provides us the flexibility to organize the code in a logical way.  **What is Python Module?**   * A Python module is a file containing Python definitions and statements. * A module can define functions, classes, and variables. A module can also include runnable code. * Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized. |
| **Create a Python Module:**  To create a Python module, write the desired code and save that in a file with .py extension.  Let’s understand it better with an example:  **Example:**  Let’s create a simple calc.py in which we define two functions, one add and another subtract.  # A simple module, calc.py def add(x,y):  return (x+y) def subtract(x, y):  return (x-y)  **Import module in Python:**  We can import the functions, and classes defined in a module to another module using the import statement in some other Python source file.  When the interpreter encounters an import statement, it imports the module if the module is present in the search path.  Note: A search path is a list of directories that the interpreter searches for importing a module.  **Syntax:**  import module **Example:**  # importing module calc.py import  calc print(calc.add(20, 6))  **Output:**  26  **Python Import From Module:**  Python’s from statement lets you import specific attributes from a module without importing the module as a whole. |
| **Program:**  # importing sqrt() and factorial from the  # module math from math import sqrt, factorial # if we simply do "import math", then  # math.sqrt(16) and math.factorial() are required. print(sqrt(16)) print(factorial(6))  **Output:**  4.0  720  **Import all Names:**  The \* symbol used with the import statement is used to import all the names from a module to a current namespace.  **Syntax:** from module\_name import \*  **PYTHONOOP’SCONCEPT**  **What is Object-Oriented Programming in Python?**  Python is also an object-oriented language since its beginning.It allows us to develop applications using an object-oriented approach.  We can easily create and use classes and objects.  It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. |



**Class:**

The

class

can

be

defined

as

a

collection

of

objects.

It

is

a

logical

entity

that

has

some

specific

attributes

and

methods.For

example:

if

you

have

an

employee

class,

then

it

should

contain

an

attribute

and

method,

i.e.

an

email

id,

name,age,

salary,

etc.

**Syntax:**

class

ClassName:

statement

<

-

1>

.

.

statement

<

-

N

**Object:**



The

object

is

an

entity

that

has

state

and

behavior.

It

may

be

any

real-world

object

like

the

mouse,

keyboard,

chair,

table,

pen,

etc.



Everything

in

Python

is

an

object,

and

almost

everything

has

attributes

and

|  |
| --- |
| methods.   All functions have a built-in attribute \_doc\_, which returns the docstring defined in the function source code.  **Example:**  class car: def\_init\_  (self,modelname, year):  self.modelname = modelname  self.year = year def display(self):  print(self.modelname,self.year) c1 = car("Toyota", 2016) c1.display()  **Method:**   * The method is a function that is associated with an object. * In Python, a method is not unique to class instances. * Anyobject type can have methods.   **Inheritance:**   * Inheritance is the most important aspect of object-oriented programming. * It specifies that the child object acquires all the properties and behaviors of the parent object. * By using inheritance, we can create a class which uses all the properties and behavior of another class. * The new classis known as a derived class or child class, and the one whose properties are acquired is known as a base class (or) parent class. * It provides the re-usability of the code. |

**Example:**

ClassFirst(object):

def\_init\_(self):

super(first,self),\_init\_()

print(

“

first

”

)

classSecond(object):

def\_init\_(self):

super(second,self),\_init\_()

print(

“

second

”

)

classThird(object):

def\_init\_(self):

super(third,self),\_init\_()

print(

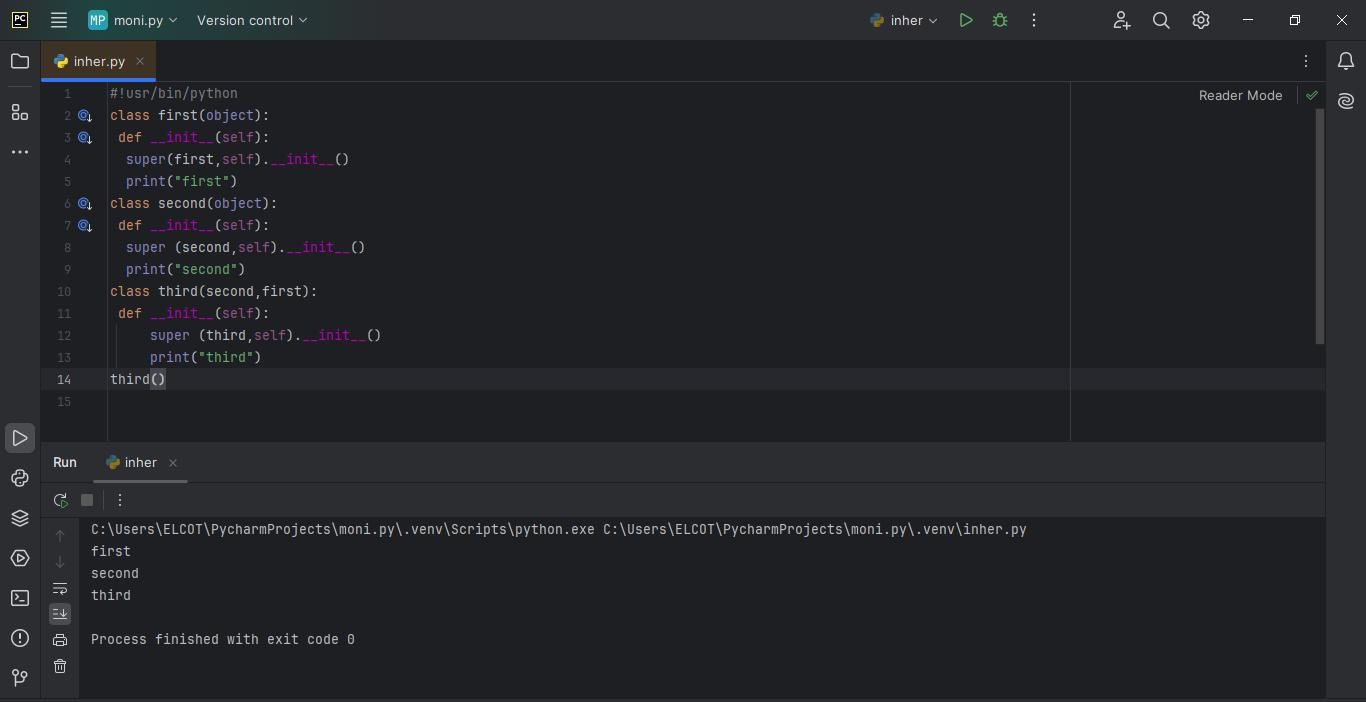
“

third

”

)

Third()



**Polymorphism:**



Polymorphism

contains

two

words

"poly"

and

"morphs".



Poly

means

many,

and

morph

means

shape.



By

polymorphism,

we

understand

that

one

task

can

be

performed

in

different

ways.

**Example:**

ClassBear(object):

def

sound(self):

print(

“

Groarr

”

)

ClassDog(object):

def

sound(self):

print(

“

woof

woof!

”

)

def

makeSound(animalType):

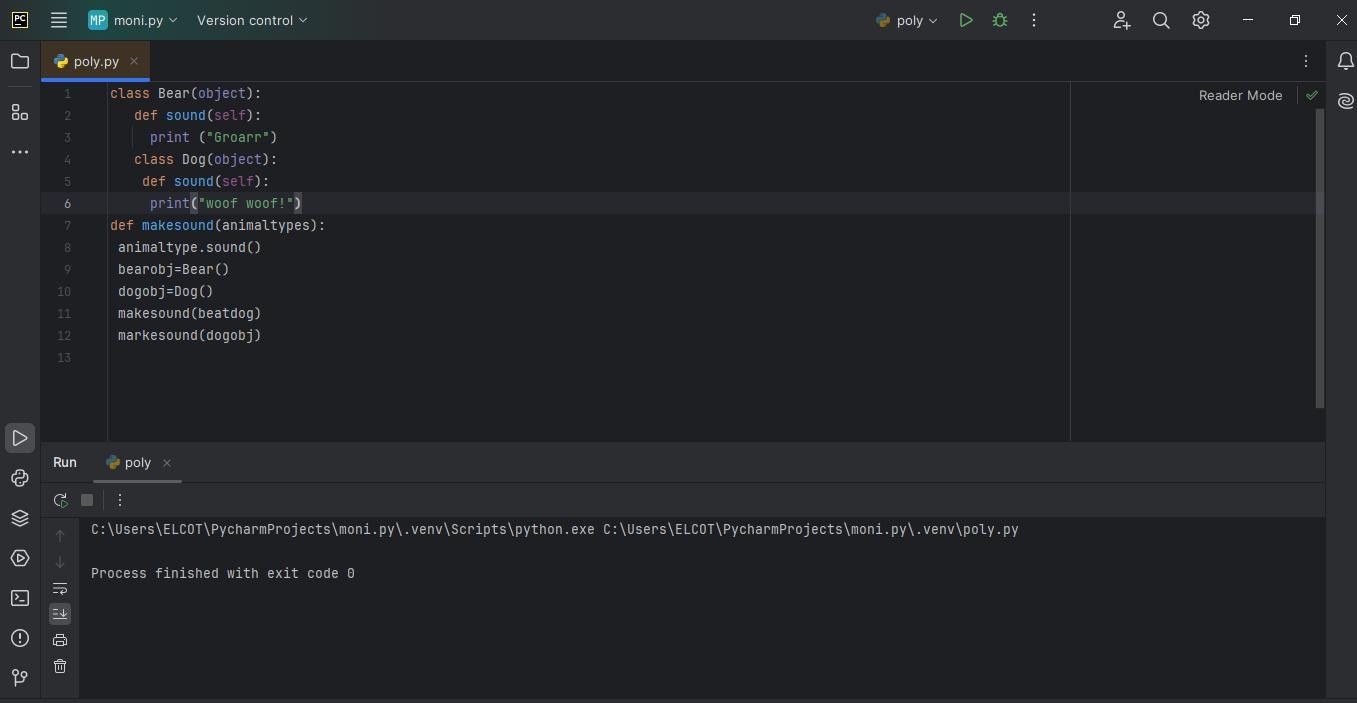
animalType.Sound()

bear()obj=Bear()

dog()obj=Dog()

makeSound(bearobj)

makeSound(Dogobj)



**Encapsulation:**



Encapsulation

is

also

an

essential

aspect

of

object-oriented

programming.



It

is

used

to

restrict

access

to

methods

and

variables.



In

encapsulation,

code

and

data

are

wrapped

together

within

a

single

unit

from

being

modified

by

accident.

**Example:**

class

car:

def\_init\_(self):

self\_updateSoftware()

def

drive(self):

print(

‘

driving

’

)

def\_updatesoftware(self):

print(

‘

updating

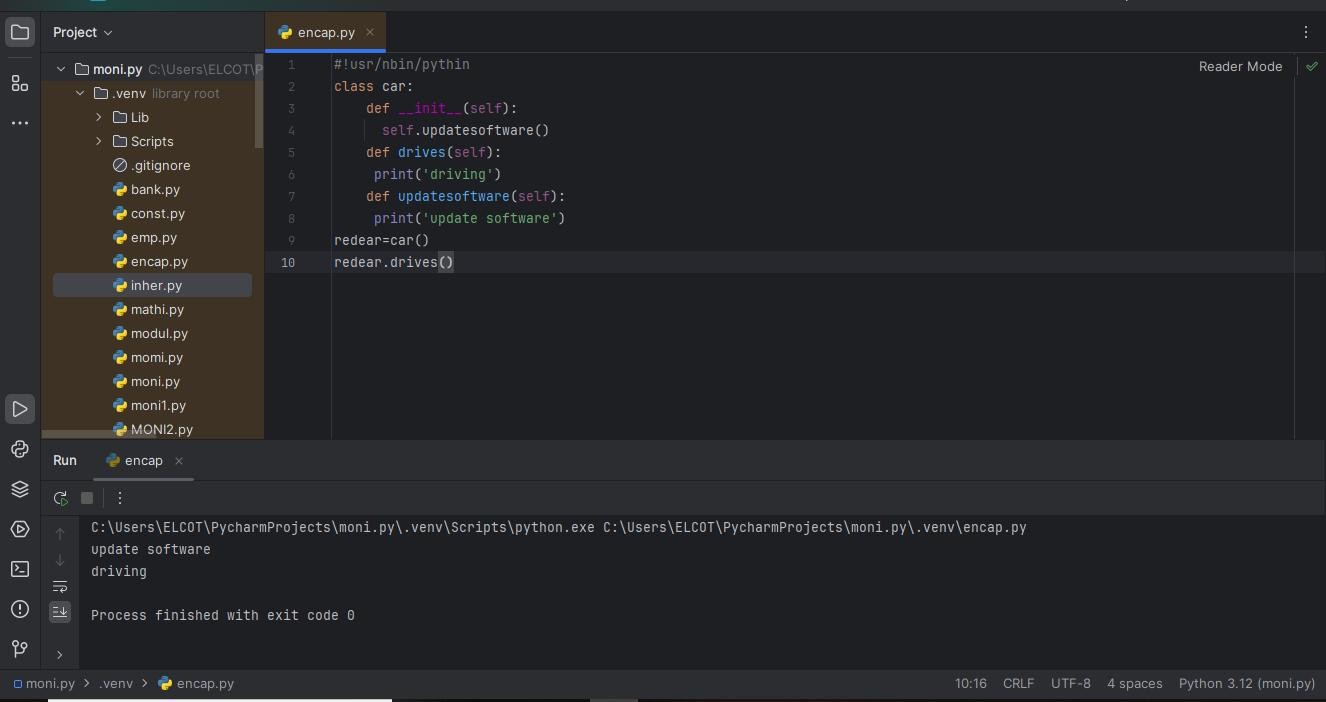
software

’

)

redcar=car()

redcar.drive()



|  |
| --- |
| **Data Abstraction:**   * Data abstraction and encapsulation both are often used as synonyms. * Both are nearly synonyms because data abstraction is achieved through encapsulation. * Abstraction is used to hide internal details and show only functionalities.   **Define a class in Python:**  In python, a class is a defined by using a keyword class like a function definition begins with the keyword def.  **Syntax of a class definition:**  classClassName: <statement1>  <statement-N>   * A class creates a new local namespace to define its all attribute. * These attributes may be data or functions. * There are also some special attributes that begins with double underscore(\_). * When we define a class, a new class object is created with the same class name. * This new class object provides a facility to access the different attributes as well as to instantiate new objects of that class.   **Creating an object in Python:**   * we can create new object instances of the classes. * The procedure to create an object is similar to a function call.   **Example:**  class student:  def\_init\_(self,rollno,name):  self.rollno=rollno self.name=name def displaystudent(self): |
|  |

54

print(

“

rollno:

”

,self.rollno,

”

,name:

”

,self.name)

emp1=student(121,

”

Arjun

”

)

emp2=student(122,

”

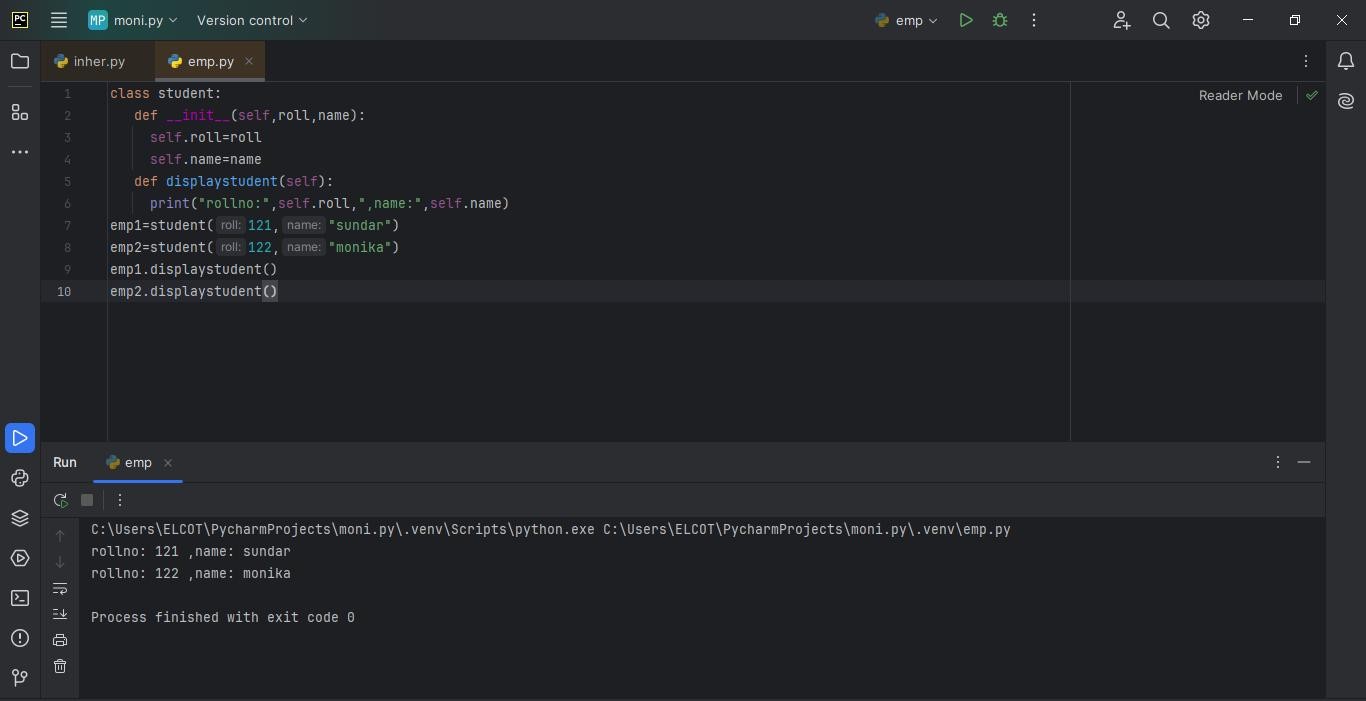
Ajeet

”

)

emp.displaystudent()

emp.displaystudent()



**class**

**complex**

**Number:**

def\_init\_(self,r=0,i=0)

self.real=r

self.img=i

def

getData(self):

print(

“

{0}+{1}

I

”

.format(self.real,self.img))

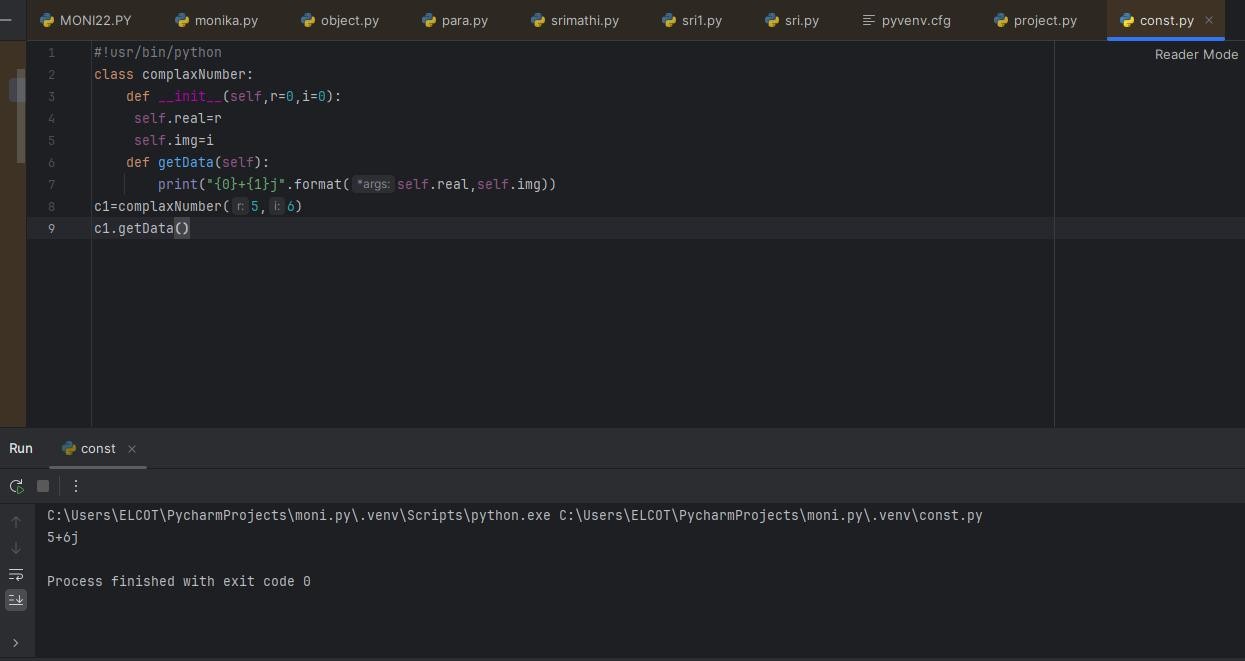
c1=complexNumber(5,6)

c1.getData()

**Output:**

5+6

j



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **FILE HANDLING**  File handling is an essential concept in Python that allows the user to interact with external files. It enables a program to create, read, write, append, and delete files. File handling is useful for data storage, reading large datasets, logging, etc.  **File Handling Modes**   | **Mode** | **Description** | | --- | --- | | 'r' | Read mode – opens the file for reading. File must exist. | | 'w' | Write mode – opens file for writing. Overwrites if file exists. | | 'a' | Append mode – opens file and adds content to the end. | | 'x' | Create mode – creates a new file. Fails if file already exists. | | 'b' | Binary mode – used for binary files like images or audio. | | 't' | Text mode – used for text files (default mode). |   **Advantages of File Handling**   * Permanent storage of data * Easy data sharing * Efficient memory usage * Used for logging, reporting, and data manipulation   Basic File Operations:  **Opening a File**  file = open("sample.txt", "r")  **Reading a File**  content = file.read()  line = file.readline()  lines = file.readlines()  **EXCEPTION HANDLING**  **Error:**  Errors in a program can be categorized into following types:  **1.Compile Time Error:** Errors that occur during the compilation of the program.  **2.Run Time Error:** Errors that occur while the program is running. Some of the example of run time error are:   * Division by Zero * Using of undefined variable * Trying to access a non existing file   **3.Logical Error:** Errors in the program s logic, which do not produce errors but lead to incorrect results. Some of the example of logical error are:   * Giving wrong operator precedence * using wrong variable name for calculation   **4.Syntax Errors:** Mistakes in writing code that prevent the program from running. Some of the example of syntax error are:   * Incorrect Indentation * Misspelled a keyword * leaving out a symbol such as colon (:), parentheses [{()}] or comma (.).   **5.Semantics Errors:** Logic errors that produce unintended results.  **Exception:**   * An exception is a type of error that occurs during program execution. * Python generates an exception whenever an error is encountered. * Exceptions can be handled to prevent the program from crashing.   **Exception handling:**   * Exception handling is the process of dealing with run-time errors. * It involves using 'try...except blocks to catch and handle exceptions.   56 |

**ARTIFICIAL INTELLIGENCE**

Artificial Intelligence (AI) is the branch of computer science that aims to create machines or software that can think, learn, and make decisions like humans. It enables computers to perform tasks that usually require human intelligence.

**Purpose of Using AI:**

* To generate well-structured and easy-to-understand technical explanations.
* To save time while preparing educational content.
* To ensure the documentation is complete, correct, and presentable.

**Benefits of AI:**

* Automation of tasks
* Fast and accurate decision-making
* Reduces human errors
* Can work 24/7 without fatigue

**Based on Capabilities**

This classification explains how **intelligent** or **powerful** the AI system is.

**a) Narrow AI (Weak AI)**

* **Definition**: AI that is trained and designed for a **specific task only**.
* **Example**: Siri, Alexa, Google Maps, Face Recognition.

**b) General AI (Strong AI)**

* **Definition**: AI that has the ability to **think, learn, and perform any task** just like a human.
* **Example**: Still in development, not yet achieved.

**c) Super AI**

* **Definition**: AI that is **more intelligent than humans** and can perform tasks better than humans in all aspects.
* **Example**: Still a concept; may be achieved in the future.

**MACHINE LEARNING:**

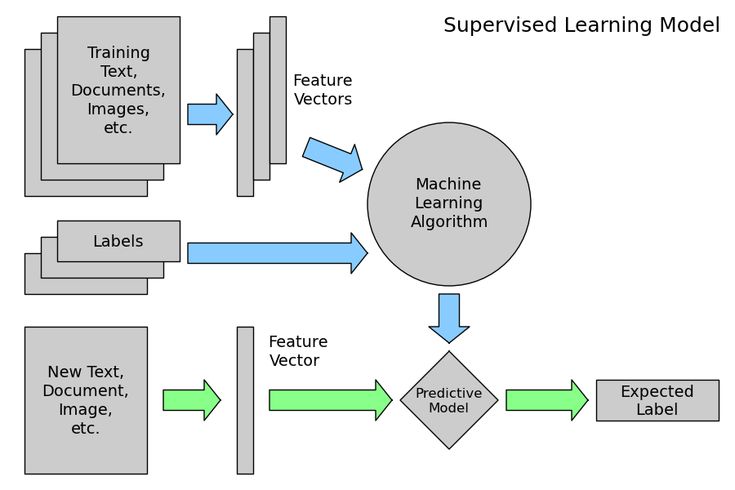
Machine Learning (ML) is a subfield of Artificial Intelligence (AI) that allows computers to learn from data and make decisions or predictions without being explicitly programmed. Instead of writing rules for every task, we give the machine data, and it automatically learns patterns from it. The more data it receives, the better it becomes at making accurate predictions or decisions. ML is widely used in applications such as spam detection, recommendation systems (like Netflix and Amazon), speech recognition, and self-driving cars.

**Types of Machine Learning**

Machine Learning is mainly divided into **three types**:

**1. Supervised Learning**

* The model is trained using **labeled data** (data with correct answers).
* It learns the relationship between input and output.
* **Examples**: Email spam detection, predicting house prices, loan approval.
* **Algorithms**: Linear Regression, Decision Trees, Support Vector Machines.



**2. Unsupervised Learning**

* The model is given **unlabeled data** (no correct output provided).
* It tries to **find patterns or groupings** in the data.
* **Examples**: Customer segmentation, market basket analysis.
* **Algorithms**: K-Means Clustering, Hierarchical Clustering, PCA.

**3. Reinforcement Learning**

* The model learns by **interacting with an environment** and receiving rewards or penalties.
* It is used for tasks where **decision making and learning from mistakes** are important.
* **Examples**: Game playing (like Chess, Go), robotics, self-driving cars.
* **Algorithms**: Q-Learning, Deep Q Networks.

**ALGORITHM**:

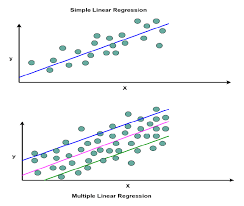
An algorithm is a step-by-step procedure or a set of instructions designed to solve a specific problem or perform a task. It is like a recipe that tells a computer exactly what to do in order to reach a desired result. Algorithms are the foundation of all computer programs and are used in every field of computer science including data processing, calculations, automation, and artificial intelligence.

**Key Characteristics of an Algorithm:**

* Clear and unambiguous steps
* Finite number of steps
* Input and output clearly defined
* Efficient in solving the problem
* Can be implemented using any programming language

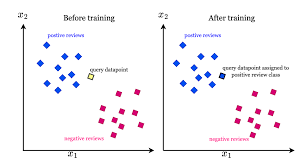
**LINEAR REGRESSION**

**Linear Regression** is a supervised learning algorithm used for **predicting continuous values**. It establishes a **linear relationship** between the input variable (X) and the output variable (Y). The algorithm finds the best-fit straight line (called the regression line) that minimizes the difference between the predicted values and actual values. For example, predicting house prices based on area, number of rooms, etc., is a common use case. It is simple, easy to interpret, and often used as a baseline model in regression tasks.



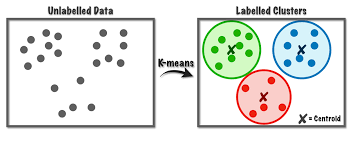
**K-NEAREST NEIGHBORS (KNN)**

**K-Nearest Neighbors (KNN)** is a simple, instance-based supervised learning algorithm used for **classification and regression**. It works by **comparing a new data point** to the ‘K’ most similar data points (neighbors) in the training set and assigning the most common label (in classification) or average value (in regression). KNN does not build a model during training; instead, it memorizes the training data and makes decisions during prediction. It’s widely used for tasks like image classification and handwriting recognition.



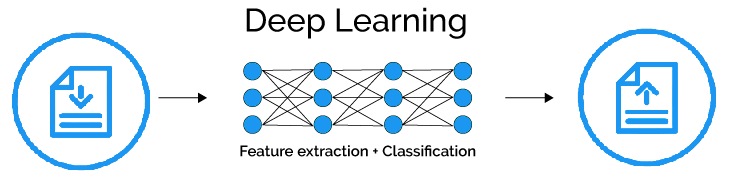
**K-MEANS CLUSTERING**

**K-Means Clustering** is an unsupervised learning algorithm used for **grouping data into clusters**. It divides the data into ‘K’ clusters based on their similarity, where each data point belongs to the cluster with the nearest mean. The algorithm starts with K random points as cluster centers and iteratively adjusts them to minimize the distance between points and their assigned cluster center. It is commonly used in customer segmentation, market research, and pattern recognition.



**DEEP LEARNING:**

Deep Learning is a subset of machine learning and artificial intelligence (AI) that mimics the workings of the human brain in processing data and creating patterns for use in decision-making. It involves training artificial neural networks with multiple layers—often referred to as deep neural networks—on large datasets to perform complex tasks such as image recognition, natural language processing, speech recognition, and autonomous driving. Deep learning models automatically extract relevant features from raw input data, reducing the need for manual feature engineering. These models are capable of learning hierarchical representations, where each layer extracts higher-level features from the output of the previous layer.



**Types of Deep Learning**

Deep Learning can be categorized into several types based on the architecture and purpose of the neural network:

**a. Feedforward Neural Networks (FNN)**

These are the simplest type of artificial neural networks where connections between nodes do not form cycles. Data moves in one direction—from input to output.

**b. Convolutional Neural Networks (CNN)**

Primarily used for image and spatial data analysis. They use convolutional layers to detect patterns and features such as edges, shapes, and textures.

**c. Recurrent Neural Networks (RNN)**

Designed for sequential data like time series or natural language. They have loops that allow information to persist and be reused across time steps.

**d. Generative Adversarial Networks (GAN)**

These involve two networks—a generator and a discriminator—that compete to improve performance. GANs are popular for generating realistic images, videos, and other synthetic data.

**e. Autoencoders**

These are unsupervised networks that learn efficient codings of input data. They are often used for dimensionality reduction and anomaly detection.

**Deep Learning Algorithms**

Deep learning algorithms are primarily based on various neural network architectures and learning techniques. Common deep learning algorithms include:

**a. Backpropagation Algorithm**

The fundamental algorithm used to train neural networks. It adjusts weights based on the error rate obtained in the previous epoch.

**b. Stochastic Gradient Descent (SGD)**

A widely-used optimization technique that updates weights incrementally using a subset of the training data.

**c. Adam Optimizer**

An improved version of SGD that combines the advantages of two other extensions of SGD—AdaGrad and RMSProp.

**d. Long Short-Term Memory (LSTM)**

A specialized form of RNN that solves the vanishing gradient problem and is effective for long sequence modeling.

**e. Convolution Operation**

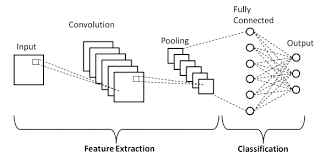
The core of CNNs, it applies filters to input data to detect features.

**4. CNN (Convolutional Neural Network): Definition**

A **Convolutional Neural Network (CNN)** is a deep learning model specifically designed to process data that has a grid-like topology, such as images. CNNs use convolutional layers that apply various filters to the input image to detect and extract features. These filters learn to detect edges, textures, shapes, and even entire objects. A typical CNN architecture includes:

* **Convolutional Layers:** Apply filters to extract features.
* **ReLU Activation Function:** Introduces non-linearity.
* **Pooling Layers:** Reduce the spatial size and computational load.
* **Fully Connected Layers:** Perform classification based on extracted features.

CNNs are extensively used in computer vision tasks like object detection, image classification, and face recognition due to their high accuracy and efficiency.

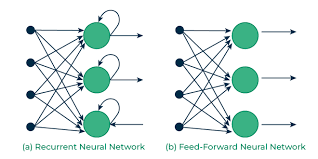


**5. RNN (Recurrent Neural Network): Definition**

A **Recurrent Neural Network (RNN)** is a type of deep learning model designed to handle sequential data. Unlike feedforward networks, RNNs have loops in them, allowing information to persist. This makes them ideal for tasks where context and temporal dynamics are important, such as:

* **Time series forecasting**
* **Language modeling**
* **Speech recognition**

However, traditional RNNs struggle with long-term dependencies due to issues like vanishing gradients. To overcome this, advanced variants such as **Long Short-Term Memory (LSTM)** and **Gated Recurrent Unit (GRU)** have been developed.



**NATURAL LANGUAGE PROCESSING:**

**Natural Language Processing (NLP)** is a branch of Artificial Intelligence (AI) that focuses on enabling computers to understand, interpret, and generate human language in a meaningful way. It bridges the gap between human communication and machine understanding by combining computational linguistics with statistical, machine learning, and deep learning models. NLP powers many real-world applications such as voice assistants (like Siri and Alexa), language translation tools, sentiment analysis, chatbots, email spam detection, and text summarization. It involves several key processes, including tokenization, part-of-speech tagging, named entity recognition, and syntactic parsing. The ultimate goal of NLP is to facilitate seamless interaction between humans and machines, making it possible for systems to comprehend context, emotion, and intent behind spoken or written language. As language is inherently complex and ambiguous, NLP continues to evolve to better grasp nuances, dialects, and multilingual data, playing a crucial role in advancing intelligent automation and human-computer interaction.

**REAL-TIME APPLICATIONS OF ARTIFICIAL INTELLIGENCE (AI)**

Artificial Intelligence (AI) is revolutionizing various industries by enabling machines to simulate human intelligence processes. Real-time applications of AI involve systems that process and respond to inputs instantaneously or within milliseconds, making AI an essential component in critical operations, automation, and smart decision-making across multiple domains.

**1. Autonomous Vehicles**

AI powers the core functionality of self-driving cars. Through real-time processing of data from cameras, LiDAR, radar, and GPS, AI systems make instant decisions such as braking, lane changing, and obstacle avoidance. Advanced deep learning models are used to recognize traffic signs, pedestrians, and road patterns, ensuring safe navigation.

**2. Healthcare Monitoring Systems**

AI is widely used in real-time for patient monitoring in hospitals. Wearable devices and smart sensors track vital signs such as heart rate, oxygen levels, and body temperature. AI algorithms analyze this data to detect abnormalities (e.g., arrhythmias or seizures) and alert healthcare providers immediately, enabling timely medical intervention.

**3. Facial Recognition and Surveillance**

AI-based facial recognition systems are used in airports, public places, and private organizations for identity verification, security, and attendance tracking. These systems analyze facial features in real-time to detect individuals from large databases, enhancing security and automating access control.

**4. Virtual Assistants and Chatbots**

Virtual assistants like **Amazon Alexa**, **Google Assistant**, and **Apple Siri** use AI to understand voice commands and provide real-time responses. Similarly, AI-powered chatbots are used in customer support to handle queries, process requests, and provide instant responses, improving user experience and reducing operational costs.

**5. Fraud Detection in Banking**

AI systems continuously monitor financial transactions in real-time to detect suspicious behavior and fraudulent activities. By analyzing transaction patterns, user behavior, and geographic data, AI algorithms can flag anomalies, initiate security protocols, and alert the user or authorities instantly.

**CONCLUSION:**

Participating in this 15-day internship on Python and Artificial Intelligence (AI) has been a highly enriching and transformative experience. The program commenced with a thorough introduction to Python programming, covering fundamental concepts such as variables, loops, functions, and object-oriented programming. As the internship progressed, I delved into essential Python libraries like NumPy, Pandas, and Matplotlib, which allowed me to perform data analysis and visualization efficiently. These sessions provided hands-on experience and helped me understand how Python serves as a strong foundation for AI and data-driven applications.

As we transitioned into the AI segment, I gained insight into core AI and machine learning principles, including supervised and unsupervised learning, data preprocessing, and model evaluation techniques. Using tools like Scikit-learn and TensorFlow, I was able to apply these concepts practically by building simple machine learning models and analyzing their performance. This internship significantly enhanced my technical and analytical skills, giving me the confidence and motivation to further explore the fields of Python programming and Artificial Intelligence in both academic and professional settings.

Overall, this internship has not only expanded my technical capabilities but also strengthened my problem-solving approach and logical thinking. The structured learning, practical exercises, and mentorship provided throughout the program have been invaluable in shaping my understanding of how Python and AI work together to solve real-world challenges. I am grateful for the opportunity to have been a part of this internship and look forward to applying the knowledge and experience gained in future projects and endeavors.