Step1: Download python

1. Go to the python website: visit the https://www.python.org
2. Download python 3.13.5: click the "Download python 3.13.5" button.

Step2: run the installer

1. open the installer: Double-click the downloaded file to start the installation.
2. Select Stepup options: click the "Add python 3.13.5 to PATH" checkbox and then click "Customize installation"

Step3: Customize installation.

1. Customize options: Click "Next"
2. Advanced options: Check "install for all users" and ~~leave~~ leave the other default options selected the click "Install"

Step4: Complete installation

1. Installation process: allow the installation to complete
2. Installation complete: once you see the "setup was successful" message click "close"

step 5: Verify installation

1. Open command prompt: Type "cmd" in the windows search & press enter.
2. Check python version: In the command prompt. dype python -version and press Enter. you should see python 3.13.5

Program to demonstrate different datatypes.

- data type represents data inside a variable
- data type specifies types of value a variable can hold

* datatypes:

1) int: Integer Number ex:- 10, 5, 0
2) float: decimal number ex:- 3.4, 2.56
3) str: text data (String) ex:- "Hi"
4) bool: boolean values (true, false)
5) list: ordered mutable collection
6) tuple: ordered, immutable collection
7) dict: key : value pairs
8) set: unordered, unique collection.

numeric datatype: these datatypes deals with number

1) int = int stores integer values
   x = 10

2) float = float stores floating values with decimal point
   x = 3.14

3) Complex: represent complex numbers which have a real and imaginary part
   x = 2 + 3j

* String datatype:

a collection of information enclosed in " ".

name = "Alice"

* Boolean datatype

it represents boolean value True as 1 & False as 0

flag = true

* Sequence type

1 list = represents an orderedred mutable (changeable) sequence of elements

list 1 = [1, 2, 3]

2 duple = similar to list but tuples are immutable

tupl = (1, 2, 3)

3 range :- range sequence of numbers range (5)

0, 1, 2, 3, 4

* duple, set datatype

set : represent an unordered collection of unique element.

- empty set represented by using set()

s = (1, 2, 3)

* Mapping type

1 dict : the unordered collection of key.values
key must be unique and immutable
while values can be of any type.

dict1 = (name: "Ram" age: 30)

- Integer : a = 10
  print ("int", a) # 10
- float b = 10.5
  print ("float", b) # 10.5
- Complex : c = 4i + 3j
  print (c) # 4i + 3j
- String : Str1 = "Hello world"
  print (Str1) # Hello world
- boolean (bool)
  x = True
  print (x) # True

Internal true = 1 . false = 0.

- List

```
num = [1, 2, 3, 4, 5]
print (num)   # 1, 2, 3, 4, 5
num. oppend (69)
print (num)   # 1, 2, 3, 4, 5, 69
```

- Tuple

```
t = (10, 20, 30)
print [t [10]]   # 10
```

- set

```
s = {1, 2, 3, 4, 5}
.print (5)
s. add (14)
print (s)   # 1, 2, 3, 4, 5, 14
```

- dictionary

```
student = {"name": "Ram", "age" = 18}
.print (student) # name = Ram
                  age = 18
```

decision making statements are used to control the flow of program based on creation conditions. python has several decision making statements.

1 if statement

this statement is used to execute a block of code if a certain condition is true

```
ex :- x = 10
      if (x > 5):
        print ("x is greater")
          # x is greater
```

2 if else statement:

this ~~statmen~~ statement is used to execute a block ~~of~~ of code if a certain condition is drue and if the condition is false then else block executed

```
ex :- x = 63
      if (x % 2 == 0):
        print ["Even"]
      else
        print ("odd")

      # odd.
```

3 if elif else statement:

this block is executed if we want to check multiple conditions. When if block's condition becomes false then elif block executed

```
ex:- age = 18
     if (age<=0):
        print ("Invalid")
     elif (age > 18):
        print ("eligible for voting")
     else
        print ("not eligible")
     # eligible for voting
```

4 Nested if statement: Used do check condition within condition

```
num = 15
if (num > 10):
   if (num%2 == 0):
      print ("greater than 10 and even")
   else:
      print ("greater than 10 and odd")
else:
   print ("num is less than 10")

# greater than 10 and odd.
```

looping statements are used to execute a block of code repetedly for a specified number of times. python has two types of looping statement.

- for loop:- used to iterate over a sequence (such as a list, tuple or string) and execute a block of code for each item in a sequence.

- While loop:- used to execute a block of code as long as a certain is true.

examples:

- for loop:-

```
fruits = ["Apple", "Banana", "Orange"]
for F in fruits:
    print(F)

# Apple, Banana, orange
```

- While loop:

```
i = 0
while(i<5):
    print(i)
    i += 1
```

range based for loop

```
for i in range(5)
    print(i)
```

output : 0
1
2
3
4
5

Enumerate loop

```
Fruits = ["apple", "banana", "cherry"]
for i, fruit in enumerate(Fruits):
    print(i, "{i}: {fruit}")
```

output : 0{i}: {Fruit}
1{i}: {Fruit}
2{i}: {Fruit}

Nested loop

```
for i in range(3):
    for j in range(3):
        print(i, j)
```

output = 0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2

This program demonstrates the usage of different looping statements in python including for loops, while loops and nested loops

loop control statements:
python also provides loops control statement that can be used to control the flow of loops.

Break statement:-
This is used to exit a loop permenately

```
For i in range (10)
    if (i == 3):
        break
    print (i)
```

output = 0
1
2

Continue statement:- This is used to skip current iteration of a loop & move on to the next iteration

```
for i in range (10):
    if (i == 4):
        continue
    print (i)
```

1 Concatenation:- Using the + operator we can concatenate two strings

```
Str1 = "Hello"
Str2 = "python
result = Str1 + Str2
# Hello python
```

2 reptication :- Using the operation we can repeat the string

```
Str1 = "Hii"
result = str1 * 3
print (result)
Output : Hii
         Hii
         Hii
```

3 Indexing :- accessing individual characters in a string using indices

```
S = "Cocsit"
print (S[0])
ouput = C
```

4 Slicing :- extracting ~~substring~~ substring from a string using slicing

```
S = "AnushKa"
# print ("Slice of S (1 to 4)")
print (S[1:4])
print (S[:3])
output = nush
         anu
```

5 String method: Using methods like uppercase(), lowercase(), and strip() to manipulate strings

String methods

```
str1 = "Hello world"
print(str1.upper())
print(str2.lower())
print(str1.strip())
```

Output :- HELLO WORLD
hello world
Hello world

6 Split and join :- splitting string into substring and joining substrings into a single string

```
str1 = "apple", "banana, cherry"
Fruits = str1.split(",")
print("split:", fruits)
print(join:", ", ".join(fruits))
```

7 Comparison :- Comparing two strings for equality & inequality

```
str1 = "Hello"
str2 = "World"
print(str1 == str2, "equal")
print(str1 != str2, "unequal")
```

output : unequal

8 searching : searching for substrings in a string using the in operation and find() method

```
print("does 'i' exist in str1? ", 'i' in str1)
print("index of 'o' in str2:", str2.find('o'))

# does 'i' exist in str1
# index of 'o' in str2 : 1
```

Here a program that demonstrate how to declare function and pass argument

- Function with no arguments

```
def greet ():
    print (" Hello")
output = Hello
```

- Function with arguments

```
def greet-name(name):
    print (" Hello", name)
```

Output :- ~~Prat~~ Prerana
Hello Prerana

- Function with multiple arguments

```
def add (a, b)
    return a+b:
```

- function with default argument values

```
def greet. with. default (name= "world"):
    print (" Hello, {name}!")
```

- function with variable number of arguments

```
def : Sum ( numbers):
    return sum(numbers)
```

- function with keyworded arguments.

```
def message (name, message):
    print (f "{name}. {message}")
```

call the function

```
greet()
greet_name("John")
print(add(5,7))
greet_with_default()
greet_with_default("alice")
print sum(1,2,3)
greet_with_msg(name="bob", message
        ="welcome")
```

call the function with keyword argument in any words

```
greet_with_message(message"welcome", name="Prerana
```

this program demonstrate the following aspects of function in python.

1. function declaration :- def function_name(parameters):
2. passing arguments : you ca pass argument to function when calling it.
3. default argument values :- you can specify default values for arguments in case they are not provided
4. a variable number of arguments : you can use the operator to accept a variable number of arguments.
5. keyword arguments : you can arguments using their name & they can be in any order.
6. function type : a function that performs a specific task

function (simple) : a function that performs a specific task.

function with arguments: a function that accepts arguments to customize it's behaviour

~~function with return value: a function that accepts arguments to customize it's b~~

function with return value: a function that returns a value after execution.

function with default argument: a function that has default value for some or all of it's arguments.

function with variable argument: a function that accepts variable number of argument.

collection are data structure that allow us to store and manage group of items

The main built in collection are

list → ordered → mutable, allows duplicates

tuple → ordered → mutable allows duplicates

set → unordered mutable. no. duplicates

dictionary → key value pairs. mutable

list (ordered, mutable allows duplicates)

```
my-list = [10, 20, 30, 20, 40]
print("list:", my-list)
my-list.append(50)
print("updated list", my-list)
```

output = list : [10, 20, 30, 20, 40]
updated list [10, 20, 30, 20, 40, 50]

tuple (ordered, immutable, allow duplicates)

```
my-tuple = (1, 2, 3, 2, 4)
print(my-tuple)
```

output = 1, 2, 3, 2, 4

set (unordered, unique element)

```
my_list = {10, 20, 30, 40}
print("set", my_list)
my_set.add(50)
print("unordered my=set)
```

output: 10, 20, 30, 40, 50

dictionary (key-valuepair)

```
my_dict {
        "name": "Prerana",
        "age": 18,
        "city": "Nilanga"
      }
print("dictionary", my_dict)
my_dict["age"] = 22
print("updated", my_dict)
```

1 Square pattern:

The square pattern is a type of pattern where numbers a characters printed in a sequare shape

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

output :-

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

Explanation:

The program users two nested loops to print the square pattern. the outer loop iterates over the rows of the square and the inner loop iterates over columns.

In each iteration of the inner loop. an asterik (*) is printed followed by a space (end=" ")

after each row is printed, a newline character is printed (print())

2 right angle triangle pattern

The right angle triangle pattern is a type of pattern where numbers or characters are printed in a right angled triangle shape

Here's an example on a light angled triangle
triangle pattern program in python

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

output :-

```
*
* *
* * *
```

The program uses two nested loops to print the right angle pattern the outer loop iterates over the rows of the triangle and the inner loop j in range(i) iterater over the columns.
after each row is printed, a newline characters is printed (print())

3 pyramid pattern.

The pyramid pattern is a type of pattern where numbers or characters are printed in pyramid shape. Here is an example of pyramid pattern program in python.

```
n=5
for i in range(n):
    for j in range(n-i-1):
        print(" ", end=" ")
    for k in range(2*i+1)
        print("*", end=" ")
    print()
```

output :

```
    *
   * * *
  * * * * *
```

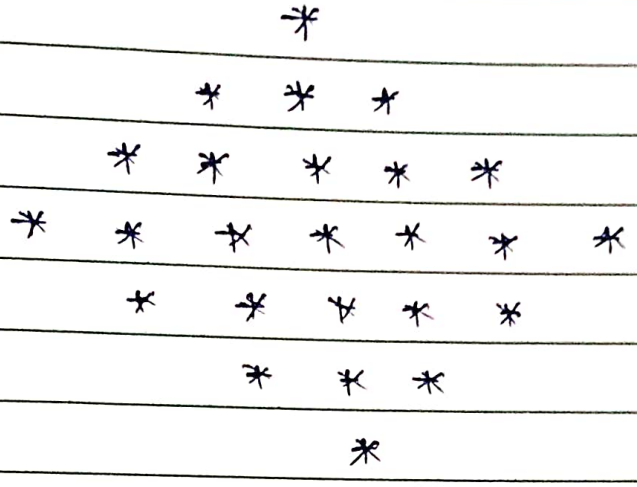
The program user three loops to print the pyramid the outer loop iterater over the rows of the pyramid.
The first inner loop prints spaces to create the indentation needed for the pyramid shape. the number of spaces decreases as the row number increase.

4 Diamond pattern :-
The diamond pattern is a type of pattern where numbers or characters are printed in a diamond shape. Here's is an example of a diamond pattern program in python

```
n = 5
for i in range(n):
    for j in range(n - i - 1):
        print(" ", end = " ")
    for k in range(2 * i + 1):
        print("*", end = " ")
    print()

for i in range(n-2, -1, -1):
    for j in range(n - i - 1):
        print(" ", end = " ")
    for k in range(2 * i + 1):
        print("*", end = " ")
    print()
```

The program will output a diamond pattern (*) with a height of 9 (5 rows for the upper half and 4 rows of the lower half)

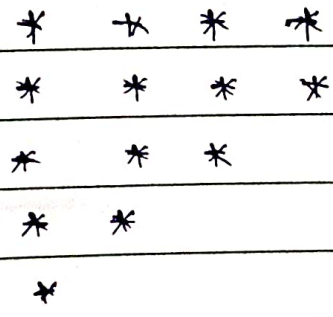


5 Inverted right angle triangle pattern

The inverted right angle triangle pattern is type of pattern where number of characters are printed in an inverted right angle triangle

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

output



The program uses two nested loops to print the inverted right angle triangle pattern. the outer loop iterates over the rows iterates of the triangle in reverse order.

Inheritance is a fundamental concept in object oriented programming that allows one class to inherit the properties and methods of another class.

base class :-

```
class animal:
    def __init__(self,name):
        self.name = name
    def eat(self):
        print("eating")
    def sleep(self):
        print("sleep")
```

derive class :-

```
class dog(animal):
    def __init__(self,name,breed):
        super().__init__(name)
        self.breed = breed
    def bark(self)
        print(f"{self.name} the {self.breed}
              is barking")
```

create instance :-

```
dog = dog("buddy", "golden retriver")
cat = cat("whiskers", "black")
```

call methods :-

```
dog.eat()
dog.sleep()
dog.bark()
cat.eat()
```

This program defines a base class. class animal with methods eat() and sleep(). two derived classes eat() and sleep() two derived class dog and eat inherit from the animal class and add their own specific methods bark() and meow() respectively.

Types of inheritance:-

1. Single inheritance:- a derived class inherited from a single base class.

```
class animal:
    pass
class dog (animal):
    pass
```

2. multiple inheritance:- a derived class inherits from multiple base

```
class animal:
    pass
class mammal:
    pass
class dog (animal, mammal):
    pass
```

3. Multilevel inheritance:- a derived class inherits from a base class that itself inherits from another base class

```
class animal:
    pass
class mammal (animal)
    pass
class dog (mammal):
    Pass
```

4 Hierachical inheritance: multiple derived classes inherit from a single base class

```
class animal
    pass
class dog (animal):
    pass
class cat (animal):
    pass
```

5- Hybrid inheritance: a combination of multiple inheritance types

```
class A:
    def feature_A (self):
        print ("features from A")
```

list operations.

create list :- a list is a collection of items that can be of any data type including strings integers. floats and other lists.

```
my-list = []
# create a list with elements.
my-list = [1, 2, 3, 4, 5]
# Create a list with different data types
my-list = ["apple", 2, 3, 5, True]
# Create a list using the list() function
my-list = list (1, 2, 3, 4, 5)
# create a list using list comprehension
my-list = [x for x in range (1, 6)]
```

characteristic of lists.

Ordered: lists maintain the order in which elements were added

Indexed: list elements can be accessed using their index

Mutable:- lists can be modified after creation

dynamic : list can grow or shink - dynamically as elements are added or removed.

2) access list :-

you can access list elements using their index or by iterating over the list. here some ways to access list elements.

accessing elements by index:-

You can access list elements using their index, which is the position of the element in the list indexing starts at 0. so the first element is at index is 0 the second element is at index 1 and so on

```
my_list = [1,2,3,4,5]
print(my_list[0])  # output
```

accessing elements using negative indexing

you can also access list element using negative indexing which starts from the end of the list the last element is at index -1 the second last element is at index -2 and so on

```
my_list = [1,2,3,4,5]
Print(my_list[-1])  # 5
```

accessing element using slicing

you can access a subset of element slicing. slicing" allows you to specify a start & end index and return a new list containing the elements in that range.

```
my_list = [1,2,3,4,5]
print [my_list [1,3])  # [2,3]
```

access elements loops: you can also access list elements using loops such as a for loop or a while loop.

```
my_list [1,2,3,4,5]
for element in my_list :
    print (elements)
```
output : 1,2,3,4,5

```
i=0
while i< len(my_list):
    print (my_list [i])
    i+=1
```
by understanding how to access list elements you can perform a wide range of operations on collection of data in python

4 delete element from list:
You can delete elements froms a list using several methods. Here are some ways to delete elements from a list.

- Using the remove() method.
  The remove() method removes the first occurrence of the specified element from the list

```
my_list = [1,2,3,4,5]
my_list.remove(3)
print (my_list) # 1,2,4,5
```

- Using the pop() method :-
  The pop() method removes and returns the elements at the specified index. if no index is specified. it removes and returns the last element.

Using the del statement:-

The del statement can be used to delete elements from a list by specifying the index of a slice

```
my_list = [1, 2, 3, 4, 5]
del my_list[3]    # [1, 4, 5]
```

Using the clear() method

The clear() method removes all elements from the list

```
my_list = [1, 2, 3, 4, 5]
my_list.clear()
print(my_list) # []
```

by understanding how to delete elements from a list you can perform a wide range of operations on collection of data in a python.

Polymorphism is the ability of an object to take on multiple forms In python. polymorphism can be achieved through method overiding & over loading.

method overiding :- method overloading is when a subclass provides a different implementation of a method that is already defined in the superclass

```
class shape:
    def area(self):
        pass
class square(shape):
    def __init__(self, side):
        self.side = side
    def area(self):
        return self.side**2
class circle(shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * (self.radius**2)
square = square(4)
circle = circle(5)
print(square.area())
print(circle.area()).
```

Method overloading:

python does not support method overloading in the python does classical sense. However you can achiere similer behavior using default argument or variable length argument lists.

```
class calculator:
    def calculate(self, *args):
        if len(args) == 1:
            return args[0] ** 2
        elif len(args) == 2:
            return args[0] + args[1]
        else:
            * raise ValueError("Invalid")
Calculator = calculator()
print(calculator.calculate(5))
print(Calculator.calculate(5, 10))
```

polymorphism with function:

polymorphism can also be achieved with function. you can write a function that can work different types of object.

```
def calculate_area(shape):
    return shap.area()
class square:
    def __init__(self, side):
        self.side = side
    def area(self):
        return self.side**2
class circle:
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * (self.radius**2)
```

```
square = square(4)
circle = circle(5)
print (calculate_area(square)) # 16
print (calculate_area (circle)) # 78.5
```

by using polymorphism you can writ flexible & res reusable code that can work with different types of objects.

Exception handling is a mechanism in python that allow you to handle runtime errors so that the normal flow the application can be maintained

Try except block

The try except block is used to handle expection the try block contains the code that might raise an exception and the except block contains the code that will handle the exception

```
try:
    x=5/0
except ZeroDivisionError
    print (" cannot divide")
```

Try except else block

The try except else block is used to specify a block of code to execute if no exception were raised in try block

```
try:
  x = 5/2
except ZeroDivisionError:
    print(" cannot divide")
else
    print (" divided")
```

The ~~exce~~ except-finally block:

The try except finally block is used to specify a block of code to execute regardless of whether an exception was raised or not

```
try:
    x = 5/0
except ZeroDivisionError:
    print("cannot divide")
Finally:
    print ("code will executed")
```

raising exceptions: you can raise exception using the raise keyword

```
def divide (x,y):
    if y==0
        raise ZeroDivisionError ("cannot divi
        return x/y
try:
    print divide(5.0)
except: zeroDivisionError " as e:
    print (e)
```

Custom exception: you can create custom exception class.

```
class insufficient balanceError (Exception):
    pass
class bankaccount:
    def __init__(self, balance):
        Self.balance = balance
    def withdraw(self, amount):
        if amount > self.balance:
            raise insufficientbalanceError
                        ("Inffician balance")
        self.balance == amount
try:
    account = bankaccount(100)
    account.withdraw(200)
    except insufficient balance Error as ei
        print(e)
```

by using exception handling you can write more robust and reliable code that can handle unexpected errors and exceptions

lambda function:

lambda functions are small anonymous functions that can be defined inline within a larger expression. they are useful for simple one time use function.

basic lambda Function:- a lambda function can be defined using the lambda function can also take multiple arguments

```
Sum-num = lambda x,y : x+y
print (sum_num(10, 20))
```

using lambda with map filter & reduce

lambda functions are often used with the map(); filter() & reduce() function.

```
num = [1, 2, 3, 4, 5]
```

- using map()

```
squared_num = list(map(lambda x : x** x ,numbers))
print (squared.num) #(1, 4, 9, 16, 25)
```

- using filter()

```
even_num = list (filter (lambda x : x % 2 ==0, numbe
print (even-num)  # [2, 4]
```

using reduce()

```
from functools import reduce
sum_number = reduce(lambda x,y:x+y, numbe
print(sum_number) # 15
```

- real word example

lambda function can be used in real word application such as sorting a list of dictionaries

```
students [
        {"name": "divya", "age": 18},
        {"name": "diksha", "age": 20},
        {"name": "Anshika", "age": 22}
    ]
students.sort(key=lambda x, x["age"])
print(students)
```

String operations

python provides a wide range of string operations and methods that can be used to manipulate and process string

String concatenation :- String concatenation is the process of combining two or more string into a single string

```
str1 = "Hello"
str2 = "World"
print (str1 + str2) # Helloworld
```

String replication

String replication is the process of repeating a string multiple times.

```
str1 = "Hello"
result = "str1 * 3
print (result)  # Hello : Hello : Hello
```

- String indexing

String indexing is the process of accessing individual characters in string

```
str = "Hello"
print (str[0]) # H
```

String slicing :- String slicing is the process of extracting a subset of character from a string

```
str = "Hello world"
print (str[0:5])  # Hello
```

string methods :- python provides a wide range of methods that can be used manipulate and process strings

```
str = "Hello world"
```

upper() method

```
print (str.upper())  # HELLO WORLD
```

lower() method

```
print (str.lower())  # hello world
```

strip() method

```
print (str.strip())
```

split() method

```
str = "apple, banana, orange"
print (str.split(", "))
```

Join() method

```
fruits = ["apple", "banana", "cherry"]
print (", " join (fruits))
```

replace() method

```
Str = "Hello world"
print (Str.replace ("Hello", "earth"))
```

Find() method

```
print (Str.find ("Hello")
```

by using string operations and methods you can manipulate and process string in a wide range of way