CareerFi

Handwritten Notes by Python

Index

1. Introduction to Python

1.1 What is python ?

Python is a general-purpose, dynamic, high-level and interpreted programming language.
It is designed to be simple and easy to learn, making it an ideal choice for beginners. One of the key strengths of Python is its versatility.
Python supports the object-oriented programming approach, allowing developers to create applications with organized and reusable code.

1.2 Features of Python

Readability :- Python's syntax is designed to be clear and reusable, making it easy for both beginners and experienced programmers to understand and write code.

Simplicity :- Python emphasizes simplicity and avoids complex syntax, making it easy for learn and use compared to other programming language.

Dynamic typing: Python is dynamically typed, meaning you don't need to explicitly declare variable types.

Large standard Library: Python provides a vast standard library with ready-to-use modules and functions for various tasks, saving developers time and effort in implementing common functionalities.

Object-Oriented Programming (OOP) :- Python supports the object-oriented programming paradigm, allowing for the creation and manipulation of objects, classes and inheritance.

Cross-Platform Compatibility :- Python is available on multiple platforms, including Windows, macOS, and Linux, making it highly portable and versatile.

Extensive Third-Party Libraries :- Python has a vast ecosystem of third-party libraries and frameworks that expand its capabilities in different domains, such as web development, data analysis, and machine learning.

Interpreted Nature :- Python is an interpreted language, meaning it does not require compilation. This results in a faster development cycle as code can be executed directly without the need for a seprate compilation step.

Integration Capabilities :- Python can easily integrate with other languages like C, C++, and Java, allowing developers to leverage existing codebases and libraries.

1.3 Applications Of Python

Python is widely used in various domains and offers numerous applications due to its flexibility

ease of use. Here are some key areas where python finds application:

Web development :- Python is extensively used in web development Frameworks such as Django and Flask. These Frameworks provides efficient tools and libraries to build dynamic websites and web applications.

Data Analysis and Visualization :- Python's rich ecosystem of libraries, including NumPy, Pandas, and Matplotlib, make it a popular choice for data analysis and visualization. It enables professionals to process, manipulate, and visualize data effectively.

Machine learning and Artificial Intelligence :- Python has become the go-to language for machine learning and AI projects. Libraries like Tensorflow, keras, and scikit-learn provide powerful tools for implementing complex algorithms and training models.

Automation and Scripting: Python's easy-to-read syntax and rapid development cycle make it an ideal choice for automation and scripting tasks. It is commonly used for tasks such as file manipulation, data parsing, and system administration.

1.4 Python Installation

To download and install Python, follow these steps:

For Windows :

1. Visit the official Python website at www.python.org/downloads/
2. Download the python installer that matches your system requirements.
3. On the Python Releases for Window page, select the link for the latest Python 3.x.x release.
4. Scroll down and choose either the "Windows x86-64 Installer" for 64-bit or the "Windows x86 executable installer" for 32-bit.
5. Run the downloaded installer and follow the instructions to install python on your Windows system.

For Linux (specifically Ubuntu)

1. Open the Ubuntu software Center folder on your Linux system.
2. From the All software drop-down list box, select Developer Tools.
3. Locate the entry for Python 3.x.x and double-click on it.
4. Click on the install button to initiate the installation process.
5. Once the installation is complete, close the Ubuntu Software Center folder.

1.5 First Python Program :

Writing your first python program is an exciting step towards learning the language. Here's a simple example to get you started.

5.

```
# Printing Hello World using python
print ("Hello World!")
```

Let's break down the code:

- The print() function is used to display the specified message or value on the console.
- In this case, we pass the string "Hello, World!" as an argument to the print() function. The string is enclosed in double quotes.
- The # symbol indicates a comment in Python. Comments are ignored by the interpreter and are used to provide explainations or notes to the code.

2. Modules, Comment & Pip

* 2.1 Modules in python:

Modules provide a way to organize your code logically. Instead of having all your code in a single file, you can split it into multiple modules based on their purpose. When you want to use the functionality from a module, you can import it into your current program or another module.

This allows you to access and use functions, classes and variables defined within that module, you can avoid writing the same code repeatedly and instead reuse the code defined in the module.

Three main types of modules:

Modules
- Built-in modules
- External Modules
- User-defined Modules

1) Built-in modules:

These are modules that come pre-installed with python. They are part of the standard library and

provide a wide range of functionalities. Built in modules are readily available for use without the need for additional installations.

External Modules:

These are modules that are created by third-party developers and are not part of the standard library.

They extended Python's capabilities by providing additional functionalities for specific purposes. External modules can be downloaded and installed using package managers like pip (python package index).

Popular external modules include numpy for numerical computations.

User Defined Modules:

These modules are created by the python programmers themselves. They allow users to organize their code into seprate files and reuse functionality across multiple programs.

User-defined modules can contain functions, classes, variables, and other code that can be imported and used in other Python scripts or modules.

* 2.2 Comments in Python

Comments in Python are used to provide explanatory notes within the code that are not executed or interpreted by the computer. They are helpful for improving code readability and for leaving reminders or explainations for other developers who might work with the code in the future

In Python, comments are denoted by the hash symbol (#) followed by the comment text. When the Python interpreter encounters a comment, it ignores it and moves on to the next lines of code. Comments can be placed at the end of line or an a line by themselves.
It's important to that comments are meant for human readers and are not executed by the Python interpreter. Therefore, they have no impact on the program's functionality or performance.

Types of Comments :

Types of Comments
- Single-line Comment
- Multi-line Comment

1. Single-line comments

Single-line comments are used to add explainatory notes or comments on a single line of code. They start with a hash symbol (#) and continue untill the end of the line. Anything written after the hash symbol is considered a comment and is ignored by the python interpreter.
Here's an example:

```
# This is a single-line comment
x = 5 # Assigning a value to the variable x
```

2. Multi-line Comments:

Multi-line comments, also known as block comments, allow you to add comments that span multiple lines. Python does not have a built-in syntax specifically for multi-line comments, but you can acheive this by using triple quotes (either single or double) to create a string that is not assigned to any variable. Since it is not used elsewhere in the code, it acts as a comment.
Here's an example:

```
"""This is a multi-line comment.
   It can span across multiple lines
   and is enclosed within triple quotes.
"""
x = 5  # Assigning a value to the variable x.
```

* 2.3 What is a pip?

In simple terms, pip is a package manager for python. It stands for "pip Installs Packages" or "pip install Python".
When working with python, you may need to use external libraries or modules that provide additional functionalities beyond what the standard library offers. These libraries are often developed by the Python community and are available for anyone to use.
Pip makes it easy to install a package, manage and uninstall these external libraries. It helps you

Find and download the libraries you need from the Python Package Index (PyPI), which is a repository of Python packages maintained by the community.
With pip, you can install a package by running a simple command in your terminal or command prompt. It will automatically fetch the package from PyPI and install it on your system, along with any dependencies it requires.

11

3. Variables, Data Types Keywords & Operators

3.1 Variables in Python

In Python, variables are used to store values that can be used later in a program. You can think of variables as containers that hold data. What makes python unique is that you don't need to explicitly declare the type of a variable. You simply assign a value to a variable using the "=" operator.
For example, you can create a variable called "name" and assign it a value like this:

```
name = "Yadnyesh"
```

Here, "name" is the variable name, and "Yadnyesh" is the value assigned to it. Python will automatically determine the type of variable based on the value assigned to it.

Variables in Python can hold different types of data, such as numbers, strings, lists or even more complex objects. You can change the value of a variable at any time by

assigning a new value to it. For instance:

```
age = 25
age = 26 # Updating the value of age variable.
```

Python also allows you to perform operations on variables. For example, you can add, substract, multiply, or divide variables containing numbers. You can even combine variables of different types using operators. For instance:

```
x = 5
y = 3
z = x+y # The value of 'z' will be 8.

greeting = "Hello"
name = "John"
message = greeting + " " + name # The value of 'message' will be "Hello John".
```

Variables provide a way to store and manipulate data in python, making it easier to work with information throughput your program. By giving meaningful names to variables, you can make your code more readable and understandable.

3.1.1 Identifier in Python

In Python, an identifier is a name used to identify a variable-function, class, module, or any other user-defined object. An identifier can be made up of letters (both uppercase and lowercase), digits and underscores (_). However, it must start with a letter or an underscore.

Here are some important rules to keep in mind when working with identifiers in python.

1. Valid characters : An identifier can contain letters (a-z, A-Z), digit (0-9) and underscores (_). It cannot contain spaces, special characters like @, #, or $.
2. Case Sensitivity : Python is case-sensitive, meaning upper-case and lowercase letters are considered different. So, "myVar" and "myvar" are treated as two different identifiers.
3. Reserved Words : Python has reserved words, also known as keywords, that have predefined meanings in the language. These words cannot be used as identifiers. Examples of reserved words include "if", "While" and "def".
4. Length : Identifiers can be of any length. However, it is recommended to use meaningful and descriptive name that are not excessively long.
5. Readability : It is good practice to choose descriptive names for identifiers that convey their purpose or meaning. This helps make the code more readable and understandable.

Here are some examples of valid identifiers in python

- my_variable
- count
- total_sum
- PI
- Myclass

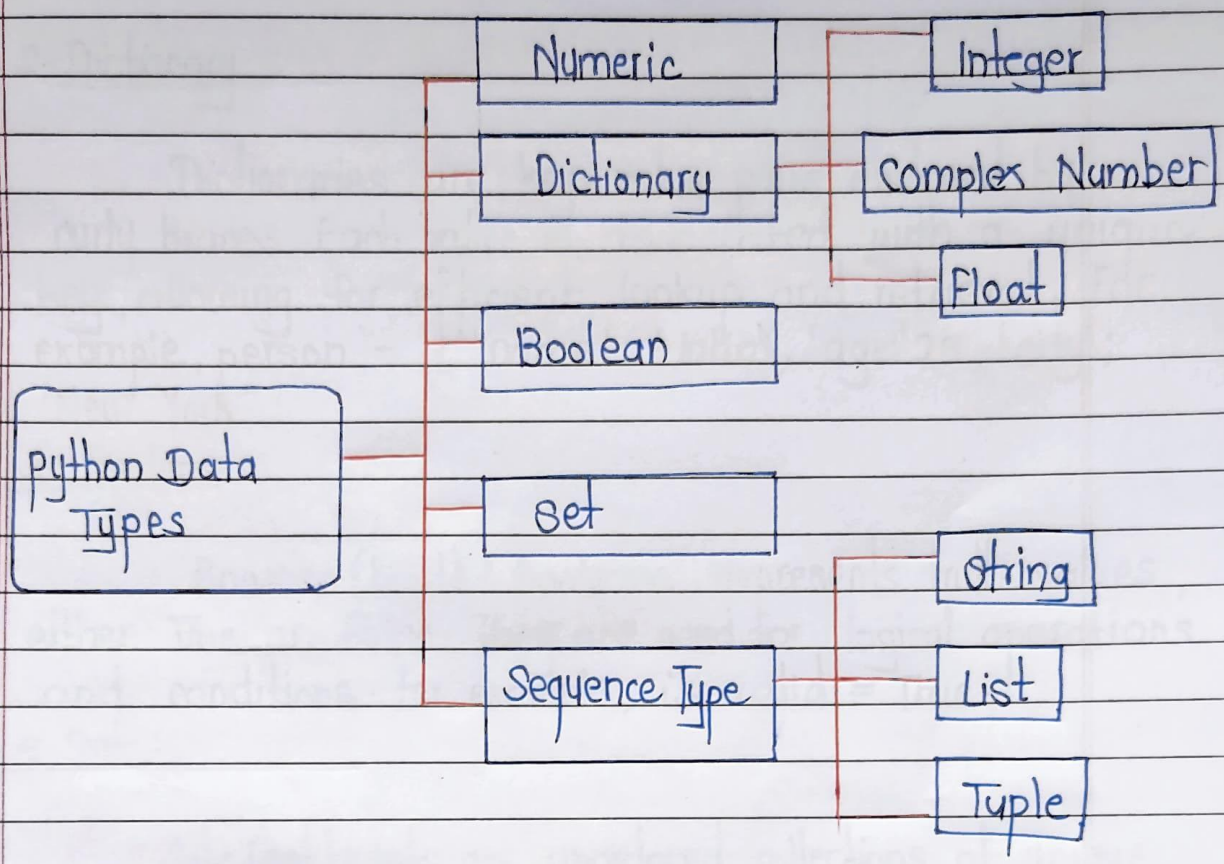
And here are some examples of invalid identifiers:

- 123abc (starts with a digit)
- my-variable (contains a hypen)
- if (a reserved word)
- my var (contains a space)

Understanding and following these rules for identifiers in Python is important to ensure clarity, reddability and proper functioning of your code.

3.2 Data Types in Python

Data types in Python refer to the different kinds of values that can be assigned to variables. They determine the nature of the data and the operations that can be performed on them. Python provides several built-in-data types, including: numeric, dictionary, Boolean, set, sequence types and numeric has Integer, Complex Number, Float. and Sequence Type has String, List and Tuple.



1. Numeric:

Python supports different numerical data Types, including integers (whole numbers), Floating-point numbers (decimal numbers), and complex numbers (numbers with real and imaginary parts)

a. Integers (int):- Integers represent whole numbers without any Franctional part. For example, age = 25.

b. Floating-point Numbers (Float): Floating-point numbers represent numbers with decimal point or Fractions. For example, pi = 3.14.

c. Complex Numbers (Complex): Complex numbers have a real and imaginary part. For ex: $z = 2+3j$ where j is suffix with it.

2. Dictionary :

Dictionaries are key-value pairs enclosed by curly braces. Each value is associated with a unique key, allowing for efficient lookup and retrieval, For example, person = {'name': 'John', 'age': 25, 'city': 'New York'

3. Boolean :

Boolean (bool) : Booleans represents truth values, either True or False. They are used for logical operations and conditions. For example, is_valid = True.

4. Set :

Sets (set) : sets are unordered collections of unique elements enclosed in curly braces. They are useful for mathematical operations such as union, intersection, and difference For example, fruits = {'apple', 'banana', 'orange'}

5. Sequence Type :

Sequences represent a collection of elements and include data types like strings, lists, and tuples. Strings are used to store textual data, while lists and tuples are used to store ordered collections of items.

a. Strings (str) : Strings represent sequences of character enclosed within single or double quotes. For example: name = 'John'.

b. List (list) : Lists are ordered sequences of element enclosed in square brackets. Each element

be of any data type. For example numbers = [1,2,3,4]

c. Tuples (tuple) : Tuple are similar to lists but are immutable, meaning their elements, cannot be changed once defined. They are enclosed in '()' paranthesis.

3.3 Keywords in Python

Keywords in python are special words that are having specific meanings and purposes within the python language. They are reserved and cannot be used as variable names or identifiers.
keywords play a crucial role in defining the structure and behaviour of python programmers.
Keywords are like building block that allow us to create conditional statements, loops, functions, classes, handle errors and perform other important operations.

* List of all the keywords in python :

False await else import pass
None break except in raise
True class finally is return
and continue for lambda try
as def from nonlocal while
assert del global not with
async elif if or yeild

3.4 Operators in Python :

Operators in Python are symbols or special

characters that are used to perform specific operations on variables and values. Python provides various types of operators to manipulate and work with different data types. Here are some important categories of operators in python:

- Arithmetic Operators
- Comparison Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identify Operators

* Arithmetic Operators:

Arithmetic Operators in python are used to perform mathematical calculations on numeric values. The basic arithmetic operators include:

- Addition (+): Adds two operands together. For example, if we have a=10 and b=10, then a+b equals 20.
- Subtraction (-): Subtracts the second operand from the first operand. If the first operand is smaller than the second operand, the result will be negative. For example: if we have a = 20 and b = 5, then a-b equals 15.
- Division (/): Divides the first operand by the other operand and returns the quotient. For example, if we have a=20 and b=10, then a/b equals 2.0.
- Multiplication (*): Multiplies one operand by the other

For example, if we have a = 20 and b = 4, then a*b equals 80

- Modulus (%) : Returns the remainder after dividing the first operand by the second operand. For example, if we have a = 20 and b = 10 then a%b equals 0.
- Exponentiation (**) or power : Raises the first operand to the power of second operand. For example, if we have a = 2 and b = 3, then a**b equals 8.
- Floor division (//) : provides the floor value of the quotient obtained by diving the two operands, it returns the largest integer that is less than or equal to the result. For example, if we have a = 20 and b = 3 then a//b equals 6.

* Comparison Operators:

Comparison operators in python are used to compare two values and return a Boolean value (True or False).

- Equal to (==) : checks if two operands are equal.
- Not equal to (!=) : check if two operands are not equal.
- Greater than (>) : checks if the left operand is greater than right operand.
- Less than (<) : check if the left operand is less than right operand.
- Greater than or equal to (>=) : checks if the left operand is greater than or equals to the right operand.
- Less than or equal to (<=) : checks if the left operand is less than or equals to the right operand.

* Assignment Operators :

Assignment Operators are used to assign values to variables. They include :

- Equal to (=) : Assign the value on the right to the variable on the left.
- Compound assignment operators (+=, -=, *=, /=) : perform the specified arithmetic operations and assign the result to variable.

* Logical operators :

Logical operators in python are used to perform logical operations on Boolean values.

- Logical AND (and) : Returns true if both operands are true, otherwise False.
- Logical OR (or) : Returns true if ~~both are~~ one of them is true, otherwise False.
- Logical NOT (not) : Returns the opposite boolean value of the operand.

* Bitwise Operators :

Bitwise operators perform operations on individual bits of binary numbers.

- Bitwise AND (&) : Performs a bitwise AND operations on the binary representations of the operands.

- Bitwise OR (|) : Performs a bitwise OR operation on the binary representations of the operands.
- Bitwise XOR (^) : Performs a bitwise exclusive OR operation on the binary representation of the operands.
- Bitwise complement (~) : Inverts the bits of the operands.
- Left shift (<<) : shifts the bit of the left operand to the left by the number of positions specified by the right operand.
- Right shif (>>) : shift the bit of the left operand to the right by the number of positions specified by the right operand.

* Membership Operators :

Membership operators are used to test whether a value is a member of a sequence.

- In : Returns true if both operands refer to the same object (sequence)
- Not In : Returns true if both operands do not refers to the same object (sequence)

* Identity operators :

Identity Operators are used to compare the identity of two objects.

- Is : Returns True if both operands refer to the same object.
- Is not : Returns True if both operands do not refers to the same object.

4. List, Dictionary, set Tuples and Type Conversion

4.1 List:

In Python, a list is a versatile data structure that allows you to store and organize multiple items in a single variable. Lists are defined by enclosing a sequence of elements within square brackets [] and seprating them with commas.

- Lists in python are ordered collections of elements where each element is identified by its position or index.
- Lists are mutable, which means you can modify, add or remove elements after the list is created.

List declaration:

In python, you can declare a list by enclosing a sequence of elements within square brackets ([]). Here are a few examples:

```
numbers = [1,2,3,4,5]
Fruits  = ["apple", "banana", "orange", "grape"]
```

In this example, we have two lists. The numbers list contains integers, specifically the numbers 1,2,3,4 &

5.

4.2 Dictionary:

A dictionary in python is a collection of key-value pairs. It allows you to store and retrieve values based on unique keys. Dictionaries are mutable, meaning you can modify them, and they provide fast access to values. They are useful for tasks like data mapping and storing structured information. key must be unique and python provides built-in methods to work with it.

Dictionary Declaration:

```
# Author: Codewithcurious.com
student = {"name": "John Doe", "age": 20, "major":
           "computer Science"}
```

4.3 set:

In python, a set is a collection of unique elements. It is used when you want to store a group of items without any duplicates.
Sets are flexible and allow you to add or remove elements. They don't have a specific order, so you can't access elements by their position.
Sets also support mathematical operations like combining sets or finding common elements.

4.4 Tuple

A tuple in python is an immutable ordered collection of elements enclosed in paranthesis (). It is used to store related data that should not be modified and supports indexing and unpacking for easy access to its elements. Tuples are memory-efficient and commonly used when data integrity and immutability are desired.

Tuple declaration

```
#Author : Codewithcurious.com
Student = ("John Doe", 20, "computer Science")
```

In this example, we have a tuple named student that contains three elements : the student's name ("John Doe"), age (20), and field of study ("computer science").

4.5 Type Convetsion

In python, type casting, also known as type conversion, refers to the process of changing the data type of a variable or value to a different type.

Python provides several built-in functions for type casting, allowing you to convert values between different data types. Here are some common type casting functions:

1. int() : Converts a value to an integer data type.
2. Float() : Converts a value to an float data type

3. str() : Converts a value to an string data type.
4. list() : Converts a value to an list data type
5. tuple() : Convets a value to a tuple data type.
6. bool() : converts a value to a boolean data type.

Type casting is very useful when you need to perform operations or comparisons with values of different data type.

Example:

```
#Author : CodewithCurious.com
# Integer to String
num = 10
num_str = str(num)
print ("Number as a String :", num_str)
# String to Integer
num_str = "20"
num = int (num_str)
print ("Number as a integer :", num)
# Float to Integer
num_float = 3.14
num_int = int (num_float)
print ("Float as an integer :", num_int)
```

Output :

```
#Author : CodeWithCurious.Com
Number as a string : 10
Number as an integer : 20
Float as an integer : 3
```

5. Flow Control

Flow control in programming refers to the ability to control the order in which statements and instructions are executed. It allows you to make decisions and repeat actions based on certain conditions. In python, Flow control is achieved using conditional statements (if, elif, else) and loops (for, while).

Conditional statements allows you to execute different blocks of code based on certain conditions, while loops enable you to repeat a block of code untill a specific condition is met.

Python Indentation

In python, indentation plays a crucial role in defining the structure and scope of code blocks. It is used to group statements together and indicate which statement belong to a particular block of code. Python uses indentation instead of traditional braces or brackets of define code blocks.

The standard convention in Python is to use four spaces for indentation. It is recommended to be consistent with indentation throughout your code to ensure readability and maintainability.

Here's an example that demonstrate the use of indentation in python.

```
# Author : CodewithCurious.Com
if condition
    # Code block 1
    Statement_1
    Statement_2
    statement_3
else:
    # codeblock2
    statement_4
    statement_5
```

In above example, the if statement is followed by an indented block of code, which is considered as the if-block.

Decision-Making statements

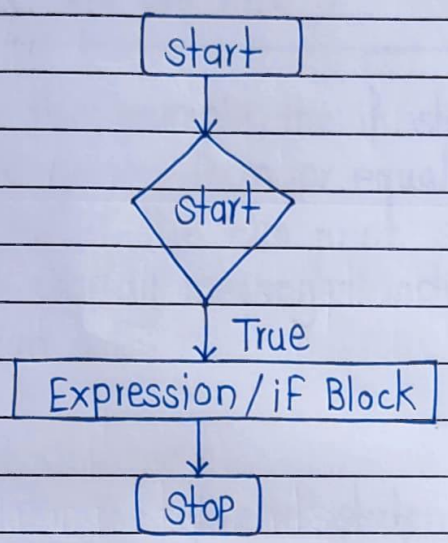
Decision-making is an important concept in programming, allowing us to execute specific blocks of code based on certain conditions. In python, we use different statements for decision making:

- IF statement : It tests a condition and executes a block of code if the condition is true.
- If-else statement : It test a condition and executes a block of code if the condition is true, and another block if the condition is False.
- Nested if statement : It allows us to use an if-else statement inside another if statement, creating multiple levels of conditions and decisions.

5.1 If - statement

The if statement in python is used to perform a specific action or execute a block of code if a condition is true. It allows us to make decisions in our programs based on the evaluation of the condition.

Flowchart :



syntax :

```
if condition
    #code to be executed if the condition is true
```

The condition is an expression that is evaluated to either True or False. If the condition is true, the code block indented below the if statement is executed. If the condition is false, the code block is skipped, and the program moves on the next statement after the if block.

Example:

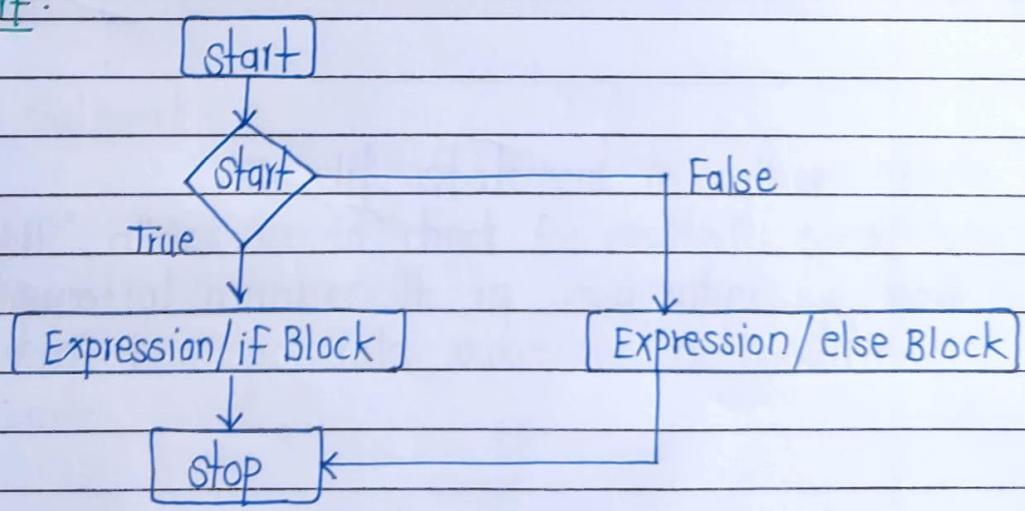
```
age = 25

if age >= 18 :
    print ("You are an adult.")
    print (" You can vote ")
```

In this example, the if statement checks if the variable "age" is greater than or equal to 18. If it is true, the program executes the two print statements within the if block, which display messages indicating that the person is an adult and can vote.

5.2 If-else statement

The if-else statement in python provides a way to perform different actions based on the evaluation of a condition. It allows us to execute one block of code if the condition is true, and another block of code if the condition is false.

Flowchart:



Syntax:

```
if condition
# code to be executed if the condition is true
else:
# code to be executed if the condition is false
```

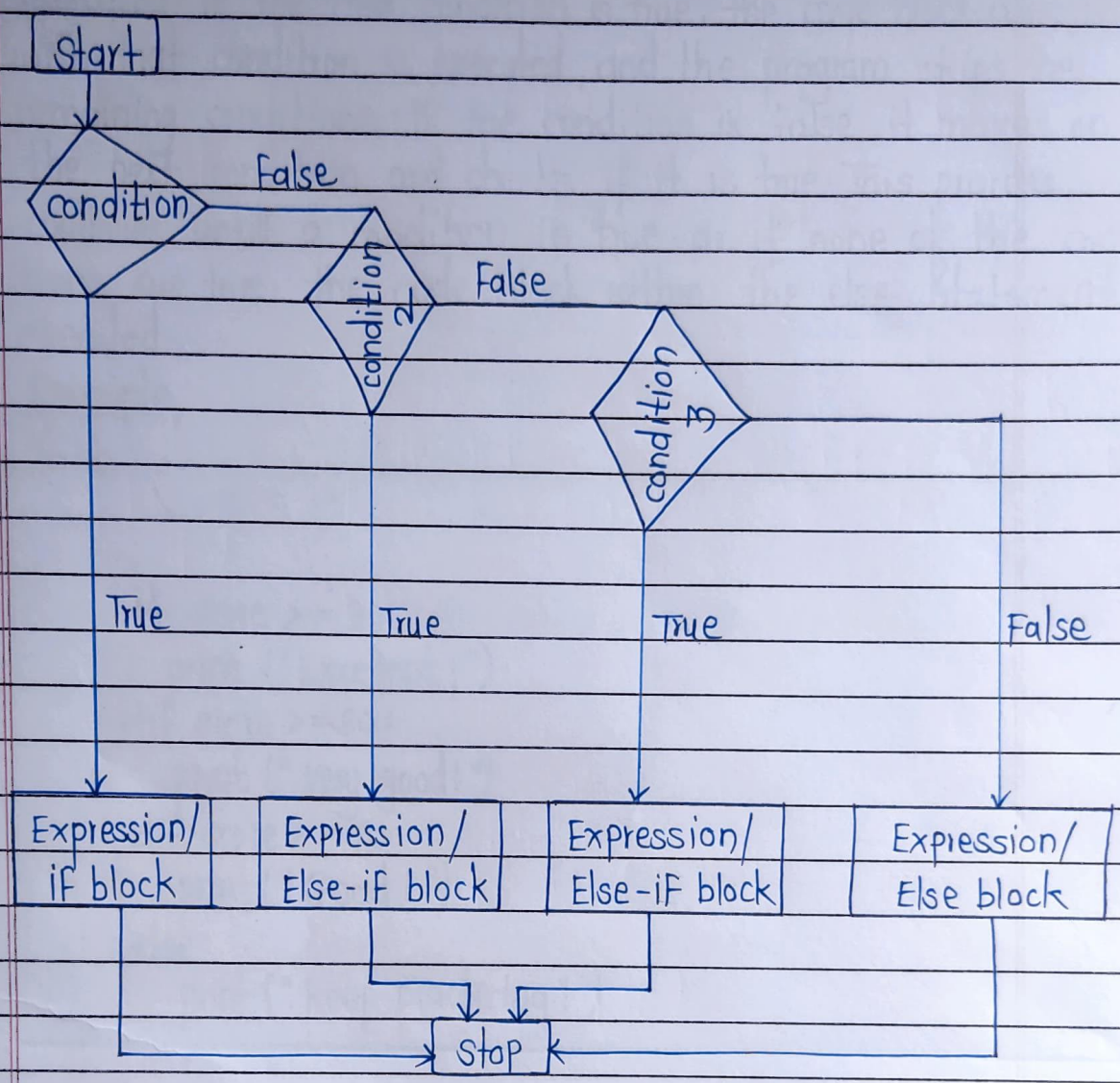
The condition is an expression that is evaluated to either True or False. If the condition is true, the code block indented below the if statement is executed. If the cond^n is false, the code block indented below the else statement is executed.

```
age = 15

if age>= 18:
    print (" You are an adult.")
    print (" You can vote.")
else:
    print (" You are not an adult.")
```

5.3 elif statement:

The elif statement in python, short for "else if", allows us to check for multiple conditions in a sequential manner. It is used when we have more than two possible outcomes or conditions to evaluate.

Flowchart



Syntax:

```
if condition1:
    # code to be executed if condition1 is true
elif condition2:
    # code to be executed if condition1 is false, condition2 is true.
elif condition3:
# code to executed if cond^n1, cond^n2 is false, condition3 is true.
else:
        # code to be executed if all conditions are false.
```

The conditions are evaluated in the order they are specified. If the first condition is true, the code block associated with that condition is executed, and the program skips the remaining conditions. If the condition is false, it moves on to the next condition and checks if it is true. This process continues untill a condition is true. or if none of the conditions are true. the code block within the else statement is executed.

Example:

```
Score = 75

if score >= 90:
    print ("Excellent!")
elif score >=80:
    print (" very good!")
elif score >= 70:
    print (" Good! ")
else:
    print (" keep practicing!")
```

5.4 Nested if statement

The nested if statement in python allows us to have an if-else statement inside another if statement. It enable us to create multiple levels of conditions and decision in our code.

In a nested if statement, the inner if statement is indented under the outer if statement.

The inner if statement is evaluated only if the condition of the outer if statement is true.

Syntax:

```
if condition1:
  #code to be executed if condition1 is true
    if condition2:
  #code to be executed if condition1, condition2 is true
    else:
# code to be executed if condn1 is true but condn2 is false.
 else:
#code to be executed if condition1 is false
```

Example:

```
age=25
income = 50000
if age>=18:
  print ("You are eligible")
 if income >= 30000:
   print (" You qualify for a loan.")
 else:
   print (" You do not qualify for a loan")
else:
 print (" You are not eligible")
```

In this example, the outer if statement checks if the age is greater than or equal to 18. if it is true, the program executes the code block within the outer if statement. Inside this code block, there is an inner if statement that checks if the income is greater than or equal to 30000.

6. Loops

python provides several types of loops to cater to different looping needs. These loops offer different syntax and condition-checking approaches while serving the same purpose of executing statements repeatedly

The available loops in python are:

1. While loop : It represents a statement or group of statement as long as a specified condition is true. The loop body is executed only if the condition is evaluated as true.
2. For loop: This loop is used to iterate over a sequence of elements, such as a list or string. It simplifies the management of the loop variable and executes a code block for each item in the sequence.
3. Nested loops: Python allows us to have loops inside other loops. This concept of nesting loops enables us to iterate through multiple levels of iteration, executing a loop within another loop.

- Pass statement :

In Python, the pass statement is a placeholder statement that does nothing. It is used when a statement is syntactically required but you don't want to perform any action or write any code at that point. It acts as a null operation and helps in maintaining the structure of program.

The pass statement is commonly used as a placeholder for code that will be implemented later or as a placeholder for empty functions or classes. It allows you to write valid code without the need for immediate implementation.

Demonstrate the Use of the Pass Statement:

```
def my_function():
    pass  # placeholder for future code implementation
if x < 10:
    pass # placeholder for conditional code
class Myclass:
    pass # placeholder for class implementation
```

In the above examples, the pass statement is used to indicate that there will be code written in the future for the function, conditional block, or class.
It allows you to run the program without any syntax errors untill you are ready to implement the actual functionality.
The pass statement serves as a useful tool for maintaining code structure, enabling incremental development, and deferring the implementation of certain parts of the program until a later stage.

6.1 While loop:

The while loop in Python repeatedly executes

a block of code as long as a specified condition remains true. It tests the condition before each iteration and continues to execute the code block untill the condition evaluates to false.

Example:

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

Here, the condition is the expression or condition that is evaluated before each iteration. If the condition is true, the code block is executed.

If the condition is false, the loop is excited, and the program continues with the next statement after the loop.

Syntax:

```
while condition:
    #code block
```

6.2 For loop:

The for loop in python is used to iterate over a sequence of elements, such as a list, tuple, string, or range.

It allows you to perform a set of actions for each item in the sequence. The loop variable takes the value of each item in the sequence one by one, untill the sequence is

exhausted.

Syntax:

```
for item in sequence
    # code block
```

Here, the item represents the loop variable that takes the value of each item in the sequence. The code block inside the loop is executed for each item in the sequence.

Example1 : Iterate over a list

```
Fruits = ["apple", "banana", "cherry"]
for fruit in fruits :
    print (fruit)
```

Output:

```
apple
banana
cherry
```

In the above example, the loop iterates over each item in the fruits list, and the loop variable fruit takes the value of each item successively. The print (fruit) statement is executed for each item, resulting in the names of fruits being printed.

Example2: Iterate using a range

```
For i in range (1,5):
    print (i)
```

Output:

```
1
2
3
4
```

In the above example, the range (1,5) function generates a sequence of numbers from 1 to 4 (exclusive). The loop variable i takes the value of each number in the sequence, and the print(i) statement prints each number.

6.3 For loop using range() Function:

The range() function in Python is often used in conjunction with the for loop to create a sequence of numbers that can be iterated over. The range() function generates a sequence of numbers based on the specified start, and step values.

The syntax of the range() function is as follows:

syntax:

```
range (start, stop, step)
```

Here, start specifies the starting value of the sequence (inclusive), stop specifies the ending value of the sequence (exclusive), and step specifies the increment

between each value in the sequence

Example1: Iterate over a range of numbers

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

Output:

```
0
1
2
3
4
```

In the above example, the range(5) function generates a sequence of numbers from 0 to 4. The for loop iterates over each number in the sequence, and the loop variable i takes the value of each number successively. The print(i) statement is executed for each number, resulting in the numbers being executed.

Example2: Iterate with a custom range and step

```
for i in range (1,10,2):
    print (i)
```

output:

```
1
3
5
7
9
```

In this example, the range (1, 10, 2) function generates a sequence of numbers from 1 to 9 with a step size of 2. The loop variable i takes the value of each number in the sequence, and the print(i) statements prints each number.

6.4 Nested For loop :

A nested for loop in python is a loop that is placed inside another for loop, It allows you to iterate over multiple sequences or perform repetitive actions within a nested structure. The inner for loop is executed for each iteration of the outer for loop, resulting in a combination of iterations.

Syntax:

```
for variable_outer in sequence_outer:
  #outer loop code block
    for variable_inner in sequence_inner:
    # Inner loop code block
```

Here, Variable_outer represents the loop variable for the outer loop, and sequence_outer is the sequence over which the outer loop iterates.

Example:

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

Output:

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 9 12 16 20
5 10 15 20 25
```

In the above example, the outer for loop iterates over the values 1 to 5. Inside the outer loop, there is an inner loop that also iterates from 1 to 5. For each value of the outer loop, the inner loop multiplies the current value of the outer lop (i) with the values of the inner loop (j), and the result is printed.

7. String

7.1 Python String

In Python, strings are a fundamental data type used to represent collections of characters. They can be defined by enclosing characters or sequences of characters in single quotes, double quotes or triple quotes.
Internally, the computer stores and manipulates characters as combinations of '0' and '1' using ASCII or Unicode encoding. Strings in python are also refered to as collections of Unicode characters.

Python provides flexibility in choosing the type of quotes to create strings, including single quotes (' '), double quotes (" ") or triple quotes (''' ''').

Using strings, we can handle text-based information, manipulate and process textual data, and perform string-specific operations in Python programming.

syntax

```
str = "Hi there!"
```

7.2 Creating string in python:

In python, you can create strings by enclosing characters within single quotes (' '), double quotes (" ") or triple quotes (''' ''').

Example:

```
str1 = 'Hello'    #single quotes
str2 = 'World'    #Double quotes
str3 = """ Python is powerful language."""  #Triple quotes

print (str1)
print (str2)
print (str3)
```

Output:

```
Hello
World
Python is powerful language.
```

In the above example, we create three different strings using single quotes, double quotes and triple quotes. All three ways are valid for creating strings in python.

7.3 strings indexing and splitting:

In python, strings are indexed starting from 0, just like in many other programming languages. For instance, the string 'HELLO' can be indexed as shown below:

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

```
str[0] = "H"
str[1] = "E"
str[2] = "L"
str[3] = "L"
str[4] = "O"
```

In this example, each letter of the string "HELLO" is assigned an id index starting from 0. The letter 'H' is at index 0, 'E' is at index 1, "L" is at index 2 and so on. Understanding string indexing is important beacause it allows you to access and manipulate specific characters within a string.

7.4 Deleting the string

In python, you can delete a string by assigning it the value None or using the del keyword. Here are the two methods to delete a string.

- Assigning None to the string

```
my_string = "Hello, World"
my_string = None
```

In this method, the variable my_string is assigned to value None, effectively deleting string.

- Using the del keyword.

```
my_string = "Hello, world"
del my_string
```

Here, the del keyword is used to delete the string object referred to by the variable my_string. After executing this statement, the variable my_string will no longer exist, and the memory occupied by the string will be freed.

7.5 String Formatting

In python, strings supports various operators that allow you to perform operations on strings. Here are some commonly used string operations.

1. Concatenation (+): The concatenation operator is used to combine two or more strings into a single string.

Example:

```
str1 = "Hello"
str2 = " World"
result = str1 + str2
print(result) #output : HelloWorld
```

2. Repetition (*): The repetition operator is used to repeat a string multiple times.

Example:

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

3. Indexing ([]): The indexing operator allows you to access individual characters of a string by their position (index).

Example:

```
str1 = "Hello"
print (str1[0]) # output: H
print (str1[3]) # output: l
```

4. Slicing ([:]): The slicing operator is used to extract a substring from a string by specifying a range of indices.

Example:

```
str1 = "Hello World"
print (str1[6:11]) #output: World
```

5. Membership (in, not in): The membership operators are used to check if a substring exists in a string.

Example:

```
str1 = "Hello World"
print ("World" in str1)      #output: True
print ("python" not in str1) #output: True
print ("print" in str1)      # output: False
print ("Hello" in str1)      # output: True
```

8. Functions

8.1 Function in Python:

Functions in Python are blocks of reusable code that perform a specific task. They help in organizing and structuring programs by breaking them down into smaller, more managable parts.

A function is defined using the def keyword, followed by the function name and paranthesis. Any required input parameters are specified within the paranthesis.

The body of the function is indented and contains the code that will be executed when the function is called.

Functions can return values using the return statement, which allows the function to pass back a result or information to the caller. If a function does not have a return statement, it will return None by default.

- Types of Function:

- Built-in Functions: These are pre-defined functions that are provided by Python. They are readily available for use without requiring any additional code. Built-in-functions cover a wide range of functionalities. such as print(), len(), range(), input() and type(). These functions are accessible throughout the Python program

- User-defined functions: These functions are created by the

the programmer to perform specific tasks based on the program's requirements. User-defined functions help in breaking down a program into smaller, modular components, making the code more organized and reusable. User-defined functions are called by their names, and arguments can be passed into the function if required.

8.2 Creating a Function

To create a function in python, you can follow these steps:

1. Use the def keyword followed by the function name to define the function. For example, def greet():.
2. Add parentheses after the function name to define any parameters the function may accept. For example, def greet(name):
3. Use a colon (:) to indicate the start of the function's code block.
4. Indent the code block that belongs to the function using spaces or tabs.
5. Write the code that you want the function to execute.
6. Optionally, use the return statement to specify the value that the function should return.
7. Call the function by using its name followed by parantheses.

Example:

```
def greet (name):
    print (" Hello, " +name+ " ! ")

greet ("john") # output : Hello, john
```

8.2 Function Calling

Function calling in Python is the process of executing a defined function by using its name followed by parantheses. When a function is called, the code inside the function's code block is executed.

To call a function, follow these steps :

1. Write the function call by using the function name followed by parantheses. For example, greet().
2. If the function accepts any arguments or parameters, pass them inside the parantheses. For example, greet ("John")

Example :

```
def greet (name):
    print( "Hello, " +name + "!"
greet("John")  #output : " Hello, John!"
```

In this example, the function greet is defined with one parameter name. The function is then called with the argument "John" inside the parantheses. This results in the function being executed and printing the greeting message "Hello, John!"

8.3 Return Statement

The return statement in python is used to specify the value that a function should return. It allows a function to

a function to pass back a result or output to the code that called it.

Here's How the return statement works:

- Within a function, when the return statement is encountered, the function's execution is immediately stopped.
- The specified value or expression following the return keyword is evaluated and becomes the result of the function call.

Example:

```
def add_numbers(num1, num2):
    sum = num1 + num2
    return sum
result = add_numbers(5,3)
print(result)  #output : 8
```

The return statement allows function to be more versatile by providing a way to communicate results or data back to calling code.

8.4 Scope of Variables

The scope of a variable in Python refers to the region of a program where the variable is recognized and can be accessed. It determines the visibility and liftime of a variable within a program.

Python has two main types of variable scope:

1. Global scope: Variables defined outside of any function or

or class have global scope.

2. Local Scope: Variables defined inside a function have local scope. They are only accessible within that specific function. Local variables have a lifetime limited to the duration of the function call. Once the function completes execution, the local variables are destroyed.

Example:

```
global_var = 10 # Global variable

def my_function():
    local_var = 20    # Local variable
    print(global_var)  # Accessing global variable
    print(local_var)   # Accessing local variable

my_function()
print(global_var) # Accessing global variable outside the function
print(local_var) # Error: Local variable is not accessible outside
 the function.
```

In this example, global_var is a global variable that can be accessed from both inside and outside the function local_var is a local variable that is only accessible the my_function function. Trying to access local_var outside the function will result in an error beacause it is not in the scope.

9. File Handling

9.1 Introduction to File Input/Output:

File input/output (I/O) refers to the process of reading data from and writing data to files. In python, file I/O allows you to work with external files stored on your computer's disk. It enables you to read data from files, write data to files, and perform various operations on them.

9.2 Opening and closing Files:

Before performing any operations on a file, you need to open it. Python provides the built-in 'open()' function to open files. It takes two arguments: the file name (or path) and the mode. The mode specifies the purpose of opening the file, such as reading, writing, or appending data. After you finish working with a file, it's essential to close it using the close() method to release the system resources associated with the file.

Here's an example of opening and closing a file for reading:

```
file = open("example.txt","r") # open the file in read mode

file.close() #close the file
```

9.3 Reading and Writing Text Files:

Python provides various methods to read and write data to text files. For reading, the most commonly used method is 'read()', which needs the entire contents of file as a string. You can also use 'readline()' to read one line at a time or 'readlines()' to read all lines and return them as a list.

To write data to a file, you can use the 'write()' method.

example:

```
# Reading from a file
file = open("input.txt", "r")
content = file.read()
print(content)
file.close()
# Writing to a file
file = open("output.txt", "w")
file.write("Hello, World!")
file.close()
```

9.4 Working with Binary Files:

In addition to text files, python also allows you to work with binary files, such as images, audio files, or any other non-text files. Binary files contains raw data that is not human-readable. To read or write binary files, you need to open them in binary mode

specifying 'rb' for reading or 'wb' for writing.

5.5 Exception Handling in File operations

When working with files, there are possibilities of errors, such as file not found, permission denied, or disk's full. Python exception handling mechanism allows you to handle such errors gracefully.

You can use the 'try-except-finally' block to catch and handle exceptions. In the case of file operations it's a good practice to wrap the file operations inside a 'try' block and handle the exceptions in the 'except' block. The 'Finally' block is used to ensure that the file is properly closed, even if an exception occurs.

Example:

```
try:
    File = open("example.txt", "r")
  # Perform operations on the file
except IOError:
    print("An Error Occured.")
Finally:
    file.close() # Ensure the file is closed, even if an
    exception occurs.
```

10. Object Oriented Programming (OOP)

10.1 Introduction to OOP in Python

Object-Oriented Programming (OOP) is a programming paradigm that organizes code into objects, which are the instances of classes. OOP focuses on encapsulating data and behaviour together within objects. It allows for code reusability, modularity, and extensibility.

Python is an object-oriented programming language that supports OOP principles. It provides features like classes, objects, inheritance, polymorphism, encapsulation, and more.

10.2 Classes and objects :-

A class is a blueprint or a template for creating objects. It defines the properties (attributes) and behaviors (methods) that objects of that class will have. An object is an instance of class.

Example:

```
class Car:
    def __init__(self, brand, model):
```

```
        self.brand = brand
        self.model = model

    def drive(self):
        print("The car is driving.")
# creating objects of the car class
car1 = car("Toyota", "carny")
car2 = car("Honda", "Accord")
# Accessing object attributes
print(car1.brand)
# calling object methods
car1.drive()
```

10.3. Constructors and Destructors:

In Python, a constructor is a special method that is automatically called when an object is created. It is used to initialize the object's attributes. The constructor method is named '__init__()'.

A destructor is another special method that is called when an object is about to be destroyed or garbage collected. In python, the destructor method is named '__del__()'.

10.4. Inheritance and Polymorphism:

Inheritance is a mechanism in OOP that allows a class to inherit properties and method from another class. The class that is inherited from is called

superclass or parent class, and the class that inherits is called the subclass or child class. In Python, a subclass can override or extend the functionality of the superclass.

Polymorphism is the ability of an object to take on many forms. It allows objects of different classes to be treated as objects of a common superclass.

10.5 Encapsulation and Data Hiding

Encapsulation is the bundling of data (attributes) and methods (functions) within a class. It provides a way to organize code and data into logical units, promoting code maintainability and reusability. Data hiding is a specific aspect of encapsulation where class attributes are kept private, restricting direct access from outside the class. This promotes information hiding and data integrity.

Example:

```
class BankAccount:
    def __init__(self, acc_no):
    self.__acc_no = acc_no

    def get_acc_no(self):
        return self.__acc_no

account = BankAccount("23456")

print(account.get_acc_no())

print(account.__acc_no)
```

10.6 Method Overriding and Overloading

Method overriding occurs when a subclass provides its own implementation of a method inherited from the superclass. Method overloading involves defining multiple methods with the same name but different parameters within a class. Here's an example:

```
class Shape:
    def area(self):
        print("calculating area")
# child class inherited from shape
class Rectangle(shape):
    def area( self, length, width):
    print(" calculating rectangle area:, length* width)

rectangle = Rectangle()

rectangle.area(4,5)
```

11. Exception Handling

11.1 Introduction to Exception Handling

Exception handling in python allows you to handle and manage runtime errors or exceptional situations that may occur during program execution. Instead of the program abruptly terminating when an a error occurs, exception handling provides a way to catch and handle these errors in controlled manner.

11.2 Exception handling mechanism:

In Python, the exception handling mechanism revolves around the try-except block. The code that might raise an exception is placed within the try block. If an exception occurs, it is caught and handled by the corresponding except block. Multiple except blocks can be used to handle different types of exceptions. There can also be an optional else block, which executes if no exceptions occur, and finally block, which always executes regardless of whether an exception occured or not.

11.3 Handling Multiple Exceptions

Python allows you to handle multiple exceptions using multiple except blocks or a single except block that catches multiple exception types. By specifying

different exception types in seprate except blocks, you can define specific actions to be taken for each type of exception. The order of the except blocks is important beacause the first matching exception block is executed.

11.4 Custom Exceptions

In Python, you can create your own custom exceptions by defining a new class that inherits from the built-in 'Exception' class or its subclasses. Custom Experience exception allows you to define and raise specific types of exceptions that are meaningful for your appln. By creating custom exceptions, you can provide more descriptive error messages and handle error scenarios in more precise way.

11.5 Error Handling strategies :

In Python, there are several error handling strategies you can employ:

Strategies	Description
Logging	• Use a logging framework, such as the logging module in python, to record and track errors.
Graceful degradation	• Implement fallback mechanisms or alternative paths to handle errors and allow Program for execution

Strategies	Description
Retry mechanism	Implement logic to automatically retry failed operations.
Error reporting	Notify users, system administrators, or relevant stakeholders about errors through appropriate channels.
Graceful termination	Graceful termination ensures data integrity and avoids leaving resources in an inconsistent state.

Date :

12. Advanced Data Structures

List Comprehsion:

Lists comprehsions are a consice way to create lists in python. They allow you to generate lists by applying an expression to each item in an iterable (e.g list, range, or string)

List comprehsions have a compact syntax and are often used for filtering, transforming, or generating lists based on existing data.

The basic structure includes an expression followed by a 'for' loop, and you can optionally include an 'if' clause for conditional filtering.

They are a powerful tool for creating lists efficiently and are considered more readable than traditional 'for' loops in many cases.

Example:

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = [x** 2 for x in numbers if x % 2 == 0]
print (squared_numbers) # [ output : [4, 16]]
```

Example:

```
numbers = [1, 2, 3, 4, 5]
even_numbers = [x for x in numbers if x % 2 == 0]
print (even_numbers) # output : [2, 4]
```

Date :

Set comprehsions:

Set comprehsions are similar to list comprehsions but create sets instead of lists.
Sets are collections of unique elements, and set comprehsions automatically remove duplicates.
They are useful for creating sets quickly and efficiently, especially when you want to eliminate duplicate values from an iterable.
sets are unordered collections of unique elements, so set comprehsions automatically remove duplicates.
set comprehsions use curly braces '{ }' or the 'set ()' constructor.

Example:

```
numbers = [1,2,2,3,3,4,5]
unique_numbers = {x for x in numbers}
print (unique_numbers) #output : {1,2,3,4,5}
```

Dictionary Comprehsions:

In python, dictionaries are the data types in which the users can store the data in a pair of key/value.

Example:

```
dict1 = {"x":12, "r":31, "l":1, "V":54}
```

Dictionary comprehsion is the method used for transferring

Date :

one dictionary into another dictionary. Throughout the process of transferring the dictionary into another, the user can also include the data of the original dictionary into the new dictionary, and each data can be trasffered as per need.

python also allows the user to perform dictionary comprehsions. The user can create the dictionary by using the simple expression of the built-in-method.

The dictionary comprehsion is created in the following form:

{ key: 'value' (value for(key, value) in iterable form)}

Example1:

```
#let's assume the user have two lists named key and values
key = ['p','q','r','s','t']
Value = [56,67,43,12,6]
# the following method is used for comprehensiong the
 dictionary
User_Dict = {X:Y for (X,Y) in zip (key,Value)}
print ("user_Dict: ", user_Dict)
```

Output:

User_Dict : { 'p' :56, 'q':67, 'r': 43 , 's':12 , 't': 6}

The user can also create the dictionary from a list by using comprehension.

Date :

Python Stack and Queue (using lists):

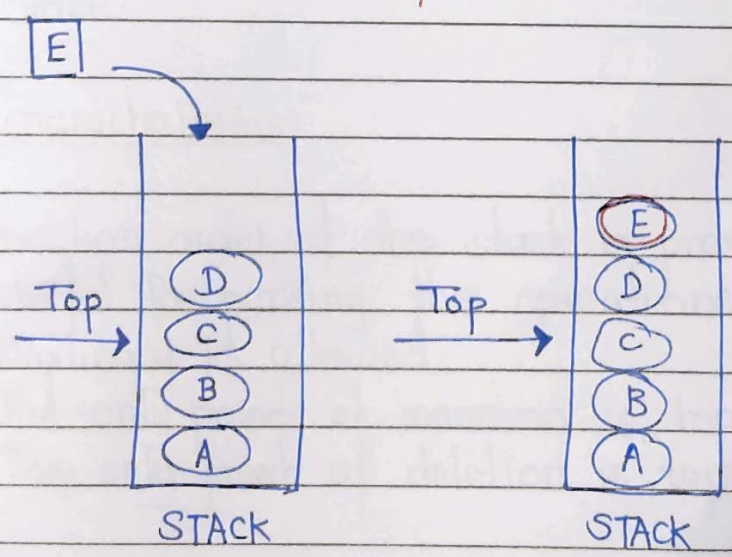
Data structure organizes the storage in computers so that we can easily access the change data. Stacks and Queues are the earliest data structure defined in computer science. A simple python list can act as a queue and stack as well. A queue follows FIFO rule (First In First Out) and used in programming for sorting. It is common for stacks and queues to be implemented with an array or linked list.

Stack:

A stack is a data structure that follows the LIFO (Last In First Out) principle. To implement a stack, we need two simple operations:

Push : It adds an element to the top of the stack
pop : It removes an element from the top of the stack.

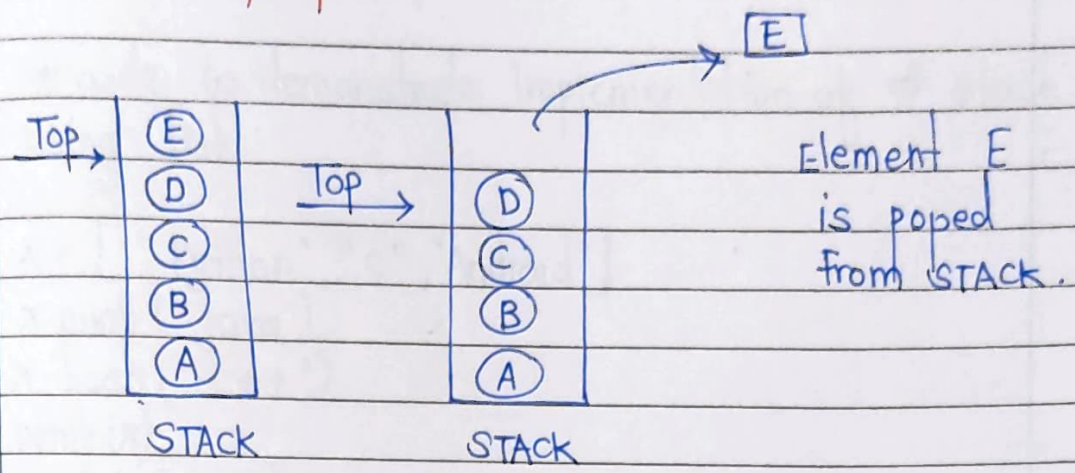
Push operation



Element E is pushed in the stack.

Date :

Pop Operation



Operations:

Adding: It adds the items in the stack and increases the stack size. The addition takes place at the top of the stack.

Deletion: It consists of two conditions, First, if no element is present in the stack, then underflow occurs in the stack, and second, if a stack contains some elements, then the topmost element gets removed. It reduces the stack size.

Traversing: It involves visiting each element of the stack.

characteristics:

1) Insertion order of the stack is preserved.
2) Useful for parsing the operations.
3) Duplicacy is allowed.
4) The only point of insertion is top.
5) The only point of deletion is top.

Date :

Code:

```
#code to demonstrate Implementation of # stack
using list.

x = [ "Python", "c", "Android"]
x.push ("Java")
x.push ("c++")
print (x)
print (x.pop())
print (x)
print (x.pop())
print (x)
```

Output:

```
['python', 'c', 'Android', 'Java', 'c++']
c++
['python', 'c', 'Android', 'Java']
Java
['python', 'c', 'Android']
```

Queue :

A queue follows the First-in-First-Out (FIFO) principle. It is opened from both the ends hence we can easily add elements to the back and can remove elements. The queue has the two ends front and rear. The next element is inserted from the rear end and removed from the front end.

Date :

Operations in python:

Enqueue
Dequeue
Front
Rear

Enqueue: The enqueue is an operation where we add items to the queue. If the queue is full, it is condition of the Queue The time complexity of enqueue is O(1).

Dequeue: The dequeue is an operation where we remove an element from the queue. An element is removed in the same order as it is inserted. If the queue is empty, it is a condition of the Queue Underflow. The time complexity of dequeue is O(1).

Front: An element is inserted in the front end. The time complexity of front is O(1).

Rear: An element is removed from the rear end. The time complexity of rear is O(1).

Date :

13. Functional Programming

Functional programming is a programming paradigm that reads / treats computation as evaluating mathematical functions and avoids changing state and mutable data. In functional programming, functions are first-class citizens, meaning they can be assigned to variable, passed as arguments to other functions, and returned as values from other functions.

In functional programming, functions are first-class citizens, meaning they can be assigned to variables, passed as arguments to other functions, and returned as values from other functions

Functional programming emphasizes immutability, meaning that once a value is assigned, it cannot be changed, and it avoids side affects, which are changes to the state or behaviour that affect the result of a function beyond its return value.

13.1 Map, filter and reduce function :

Map, filter and Reduce are built-in Python functions that can be used for functional programming tasks. With the help of these operations, you may apply a specific function to sequence items using the 'map', filter sequence elements based on a condition using the 'filter', and comulatively aggregate elements using the 'reduce'.

map(): Python's map() method applies a specified function to each item of an iterable (such as a list, tuple,

Date :

or string) and then returns a new iterable containing the results.
The map() syntax is as follows : map(function, iterable)

The first argument passed to the map function is itself a function, and the second argument passed is an iterable (sequence of elements) such as a list, tuple, set, string, etc

Example: usage of the map()

```
data = [1,2,3,4,5]
evens = filter (lambda x: x % 2 == 0, data)
for i in evens:
    print(i, end = " ")
evens = list(Filter (lambda x: x % 2 ==0, data))
print (f" Evens = {evens}")
```

Output:

2.4
Evens = [2,4]

Filter() :

The filter() function in Python filters elements from an iterable based on a given condition, or function and returns a new iterable with the filtered elements.

The syntax for the Filter() is as follows : Filter (function, iterable)

Here, also the first argument passed to the filter function is

Date :

is itself a function, and the second argument passed is an iterable (sequence of elements) such as list, tuple, set, string etc.

Example 1: Usage of the filter():

```
data = [1,2,3,4,5]
evens = Filter(lambda x : x%.2 == 0,data)
for i in evens:
    print(i, ends=" ")
evens = list (filter (lambda x : x%.2 == 0,data))
print(f "Evens = {evens}")
```

Output:

2 4
Evens = [2,4]

reduce() :-

In python, reduce() is a built-in function that applies a given function to the elements of an iterable reducing them to a single value.

The syntax for reduce() is as follows : reduce(function, iterable, [initializer])

The function argument is a function that takes two arguments and returns a single value. The first argument is the accumulated value and the second argument is the current value from the iterable.

Date :

The iterable argument is the sequence of values to be reduced.
The optional initializer argument is used to provide an initial value for the accumulated result. If no initializer is specified, the first element of the iterable is used as the initial value.

Example:

```
from functools import reduce
def add (a,b):
    return a+b
num_list = [1,2,3,4,5,6,7,8,9,10]
sum = reduce (add, num_list)
print (f" Sum of the integers of num_list :{sum}")
sum = reduce (add, num_list, 10)
print (f" Sum of the integers of num_list with initial value
    10 :{sum}")
```

Output:

```
Sum of the integers of num_list : 55
Sum of the integers of num_list with initial value 10 :65
```

13.2 Lambda and Anonymous functions:

Lambda functions, also known as anonymous functions, can be defined inline without requiring a formal function declaration and are short and one-time-use functions. They are helpful for the performance of a single task that only requires one line of code to convey

Date :

```
add = lambda x : x+1
multiply = lambda x : x*2
result1 = add(3)
result2 = add(3)
result3 = multiply(3)
```

Method2 - Using Lambda function:

```
# Using reduce to find the largest of all and printing result
Largest = reduce(lambda x, y : x if x > y else y, num)
print(f"Largest found with method 2 : {largest}")
```

Date :

14. Working with files and Directories

The file handling plays an important role when the data needs to be stored parmanently into the file. A file is a named location on disk to store related information. We can access the stored information (non-volatile) after the program termination.

The file-handling implementation is slightly lengthy or complicated in the other programming language but it is easier and shorter in Python.

In Python, files are treated in two modes as text or binary. The file may be in the text or binary format, and each line of a file is ended with a special character.

Hence, a file operation can be done in the following order:

1) Open a file
2) Read or write - performing operation
3) Close the file

Python Files I/O :

- **Opening a file**

Python provides an open() function that accepts two arguments, file name and access mode in which the file is accessed. The fuction returns a file object which can be used to perform various operations like reading, writing, etc.

Syntax:

file object = open (<file-name>, <access-mode>, <buffering>)

Date :

The files can be accessed using various modes like read, write, or append. The following are the details about access mode to open a file.

Access Mode	Description
1) r	It opens the file to read-only mode. The pointer exists at the beginning. The file is by default open in this mode if no access mode is passed.
2) rb	It opens the file to read-only in binary format. The file pointer exists at the beginning of the file.
3) r+	It opens the file to read and write both. The file pointer exists at the beginning of the file.
4) w	It opens the file to write only. It overwrites the file if previously exist or creates a new one if no file exist with the same name.
5) wb	It opens the file to write only in binary format.
6) w+	It opens the file to write and read both.
7) wb+	It opens the file to write and read both in binary format.
8) a	It opens the file in the append mode.
9) ab	It opens the file in the append mode in binary mode (format)
10) a+	It opens a file to append and read both.
11) ab+	It opens a file to append and read both in binary format.

Date :

```
fileptr = open("file.txt","r")
if fileptr:
    print("file is opened")
```

Output:

```
<class _ 'io.TextIOWrapper'>
file is opened successfully
```

The close method()

Once all the operations are done on the file, we must close it through our python script using the close() method. Any unwritten information gets destroyed once the close() method is called on a file object.

The syntax to use the close() method is given below:

```
fileobject.close()
```

Consider the example:

```
fileptr = open("file.txt","r")
if fileptr:
    print("file is opened successfully")
fileptr.close()
```

Writing the file

To write some text to a file, we need to open the file using the open method with one of the following access modes:

Date :

w : It will overwrite the file if any file exist. The file pointer is at the beginning of the file.

a : It will append the existing file. The file pointer is at the top-end of the file. It creates a new file if no file exists.

Consider the following example:

```
#open the file.txt in append mode. Create a new file if no such file exists.
fileptr = open("file2.txt", "w")
# appending the content to the file
fileptr.write()
# closing the opened the file
fileptr.close()
```

Output:

File2.txt

Read file through for loop :

```
# open the file.txt in read mode. causes an error if no such file exists.
fileptr = open("file2.txt", "r");
# running a for loop
for i in fileptr:
    print(i) # i contains each line of the file
```

Output:
Python is the modern day language.

Date :

14.2 Python OS module

Renaming the file

The Python OS module enables interaction with the operating system. The os module provides the functions that are involved in File processing operations like renaming, deleting, etc. It provides us the rename() method to rename the specified file to a new name. The syntax to use the rename() method is given below:

Syntax:

```
rename (current-name, new-name)
```

The first argument is the current file name and the second argument is the modified name. We can change the file name bypassing these two arguments.

Example 1:

```
import os
#rname file2.txt to file3.txt
os.rename ("file2.txt", "file3.txt")
```

Removing the file

The os module provides the remove() method which is used to remove the specified file. The syntax to use the remove() method is given below:

Date :

remove (file-name)

Example:

```
import os;
# deleting the file named file3.txt
os.remove("file3.txt")
```

Creating the new directory

The mkdir() method is used to create the directories in the current working directory. The syntax to create the new directory is given below:

mkdir(directory name)

Example

```
import os
# creating a new directory with the name new
os.mkdir("new")
```

The getcwd() method:

This method returns the ~~wo~~ current working directory. The syntax to use the getcwd() method is given below:

Syntax:

os.getcwd()

Date :

Example

```
import os
os.getcwd()
```

changing the current working directory:

The chdir() method is used to change the current working directory to a specified directory.

The syntax to use the chdir() method is given below.
syntax:

```
chdir("new-directory")
```

Deleting directory

The rmdir() method is used to delete the specific directory
The syntax to use the rmdir() method is given below:

Syntax:

```
os.rmdir(directory name)
```

Example:

```
import os
#removing the new directory
os.rmdir("directory_name")
```

It will remove the specified directory.

Date :

14.3 Working with the CSV and JSON files

The CSV files stands for a comma-seprated values file. It is a type of plain text file where the information is organized in the tabular form. It can contain only the actual text data. The textual data don't need to be seprated by the commas. There are also many seprator characters such as tab (\t), colon (:), and semi-colon (;), which can be used as a seprator.

The CSV module work is to handle the CSV files to read/write and get data from specified columns. There are different types of CSV functions, which are as follows:

csv.field-size-limit: It returns the current maximum field size allowed by the parser.

csv.get-dialect - Returns the dialect associated with a name.

csv.list-dialects - Returns the names of all registered dialects

csv.reader - Read the data from a CSV file

csv.writer - Write the data to a CSV file

We can also write any new and existing CSV files in python by using the csv.writer() module. It is similar to the csv.reader() module and also has two methods. i.e writer function or the Dict Writer class.

It presents two functions i.e writerow() and writerows(). The writerow() function only write one row and the writerows() function write more than one row

Date :

15. Regular Expressions

15.1 Introduction to regular Expression

A regular expression is a set of characters with highly specialized syntax that we can use to find or match other characters or groups of characters. In short, regular expressions, or Regex, are widely used in the UNIX world.

Import the re Module

```
# importing re module
import re
```

The re-module in Python gives full support for regular expressions of pearl style. The re module raises the re.error exception whenever an error occurs while implementing or using a regular expression

Metacharacters or Special Characters

- . Dot - It matches any characters except the newline character
- ^ Caret - It is used to match the pattern from the start of the string (starts with)
- $ Dollar - It matches the end of the string before the new line character (Ends with)
- * Asterisk - It matches zero or more occurances of a pattern.
- + plus - It is used when we want a pattern to match at least one
- ? Question mark - It matches zero or one ocurrence of a pattern.

Date :

[] Bracket - It defines the set of characters
| Pipe - It matches any two defined patterns.

Special Sequences:

Special sequences consists of '\' followed by a character listed below. Each character has a different meaning.

character	Meaning
\d	It matches any digit and is equivalent to [0-9]
\D	It matches any non digit character is equivalent to [^0-9]
\s	It matches any white space character and is equivalent to [\t\n\r\f\v]
\S	It matches any character except the white space character and is equivalent to [^\t\n\r\f\v]
\w	It matches any alphanumeric character and is equivalent to [a-zA-Z0-9]
\W	It matches any characters except the alphanumeric character and is equivalent to [^a-zA-Z0-9]
\A	It matches the defined pattern at the start of the string.
\B	It is opposite of \b
\b	r"\bxt" - It matches the pattern at the beginning of a word in a string
\Z	It returns a match object when the pattern is at the end of the string

RegEx functions:

Date :

compile - It is used to turn a regular pattern into an object of a regular expression that may be used in a number of ways for matching patterns in a string.

search - It is used to find the first occurence of a regrex pattern in a given string.

match - It starts matching the pattern at the beginning of the string.

fullmatch - It is used to match the whole string with regex pattern.

split - It is used to split the pattern based on the regex.

finditer - It returns an iterator that yields match objects

sub - It returns a string after substituting the first occurence of the pattern by the replacement.

escape - It is used to escape special characters in a pattern

purge - It is used to clear the regex expression cache.

Date :

16. Web Development with Python

16.1 Introduction to Web development

Web development involves creating websites and web applications that can be accessed over the internet. These applications can range from simple static websites to complex dynamic web applications that offer various functionalities. Python is a popular programming language for web development due to its simplicity, readability, and a wide range of web Frameworks and libraries available for developers.

Components of Web development :

Front-end Development :

Frontend development focuses on creating the user interface and user experience of a website or web application. This involves designing and developing the visual elements that users interact with in their web browsers. Key techonologies and concepts in fronted development include:

HTML (Hypertext Markup language) : Used for creating the structure and content of web pages.
CSS (Cascading Style sheet) : Used for styling and layout of web pages.
Javascript : A programming language used for adding interactivity and dynamic behaviour to web pages.
Frontend Frameworks : Libraries and frameworks like react Angular and Vue.Js are often used to streamline front-end development.

Date :

Backend development :

Backend development focuses on the server-side logic and data processing of a web application. Python is commonly used for backend development, and there are several web development frameworks available for this purpose, including

Flask: A lightweight and minimalistic framework for building small to medium-sized web applications

Django: A high-level framework that provides a comprehsive set of tools for building complex web applications.

FastAPI : A modern, fast and asynchronous web framework for building APIs.

Databases :

Most web applications require data storage and storage and retrievel. Python has libraries and frameworks like SQLAlchemy and Django's ORM (Object - Relational Mapping) for interacting with databases, including SQL and NOSQL databases

HTTP and APIs :

The Hypertext Transfer Protocol (HTTP) is the foundation of data communication on the web. Web developers need to understand how HTTP works, as well as how to create and consume APIs (Application Programming Interfaces) to exchange data between the frontend and backend of web application.

Date :

Web servers

Python web applications are hosted on web services. Popular web servers for hosting python applications include Gunicorn, uWSGI and ASGI servers like Daphne for asynchronous applications.

Steps in web development:

1. Project Setup:

Choose a web framework (e.g Flask, Django) and set up your development environment by installing python and necessary packages.

2. Design and Layout:

Create the visual design and layout of your web application frontend using HTML and CSS.

3. Backend development:

Write the server-side logic for your web application using your chosen python framework. Define routes, handle HTTP requests and interact with databases.

4. Database Integration:

If your application requires data storage, design the database schema and integrate it with your backend code.

Date :

Testing:

Test your web application throughly to ensure it functions correctly and is free of bugs.

Deployment:

Deploy your web application to a web server or cloud hosting platform, making it accessible to users on the internet.

16.2 Building Simple Web applications with Flask or Django (Basic concepts)

Building simple web applications with Flask or Django involves understanding the basic concepts of these Python web frameworks and how to use them to create a functional web application.

Flask:

Flask is a lightweight and micro web framework for Python. You det It is minimalistic and provides the essential tools for building web applications. Flask is known for its simplicity and Flexibility, making it a good choice for small to medium-sized projects.

Basic Concepts:

Routing:

Flask uses routing to map URLs to specific python functions

Date :

You define routes using decorators such as '@app.route("/")', to specify which function should handle a particular URL. For example:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def home():
    return "Hello, World!"
if __name__ == "__main__":
    app.run
```

Views:

In Flask, views are Python functions that handle requests and return responses. These functions are associated with specific routes. In the above example, the 'home()' function is a view that returns "Hello, World!" when the root URL ("/") is accessed

Templates:

Flask allows you to use templates to seprate the HTML code from Python code. You can use Jinja2, a templating engine, to render dynamic content in HTML templates. This sepration of concerns makes it easier to maintain your code. Example:

```
from flask import Flask, render_template
app = Flask(__name__)
```

Date :

```
@app.route("/")
def home():
    return render_template("index.html", message="Hello, World!")
if __name__ = "__main__"
  app.run()
```

static files:

Flask can serve static files (e.g CSS, Javascript, images) using the 'static' folder. This helps in organizing and delivering frontend assets efficiently.

Django :

Django is high-level, full-stack web framework for python. It follows the "batteries-included" philosophy, providing many built-in-features for common web development tasks like authentication, database interaction, and form handling.

Basic Concepts:

Project and App structure:

In Django, a project is the entire web application, while an app is a modular component within the project. Django projects can contain multiple apps. The project structure is created for you start a new Django project using the 'django-admin'.

Date :

Models:

Django's models define the structure of your database tables. Models are python classes that inherit from 'django.db.models.Model'.

Templates

```
<h1> Hello , {{ user.username }} ! <h1>
```

URL Routing:

Django's uses a URL dispatcher to map URLs to view functions. You define URL patterns in the projects 'urls.py' file.

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.home, name='home'),
]
```

Date :

17. Data Analysis and Visualization

Data analysis is the process of inspecting, cleaning transforming and modeling data to discover useful information, draw conclusions, and support decision-making. It's a crucial step in understanding patterns, trends and insights within datasets. Python offers several libraries and tools that are commonly used for data analysis.

Data collection:

Data analysis begins with the collection of relevant data. This data can come from various sources, such as databases, spreadsheets, APIs or web scrapping. Python can be used to automate data retrieval tasks and store data in suitable formats, such as CSV or databases.

Data cleaning:

Raw data often contains inconsistencies, missing values, and errors. Data cleaning involves tasks like:
Handling missing data by filling in values or removing rows/columns.
- Correcting data entry errors
- Standardizing data formats
- Python libraries like pandas are widely used for data cleaning due to their flexibility and data manipulation capabilities.

Data Exploration:

Data exploration involves examining the datasets

characteristics and understanding its structure. key techniques include:

- Descriptive statistics to summarize data (e.g mean, median, variance)
- Data visualization to create plots and charts for initial insights.
- Exploratory Data Analysis (EDA) to investigate relationships and patterns within the data.
- Libraries like Matplotlib and Seaborn are essential for creating visualizations, while Pandas aids in generating summary statistics

Data Transformation :

- Data may need to be transformed to make it suitable for analysis. Transformations can include:
- Feature engineering to create new variables or extract meaningful information from existing ones.
- Normalization or scaling of data to ensure consistent units and ranges.
- Encoding categorical variables into numerical formats
- Python libraries like Scikit-learn and Pandas provide tools for data transformation.

Data Modeling :

- Once the data is cleared and prepared, modeling techniques can be applied to uncover patterns or make predictions Common tasks include:
- Regression analysis to understand relationships between variables.

Date :

- classification for categorizing data into classes
- clustering to group similar data points
- Time series analysis for temporal data.
- Python offers extensive libraries for machine learning and statistical modeling, including Scikit-learn, stats models and TensorFlow.

Data Interpretation:

- After building models, it's important to interpret the results and draw meaningful conclusions. This can involve
- Analyzing model coefficients or feature importances.
- Assessing model performance through metrics like accuracy, precision, and recall.
- Visualizing model predictions and comparing them to actual data
- Jupyter Notebooks, which integrate code, visualizations and explainatory text, are a popular choice for documenting and sharing data analysis.

Reporting and Visualization:

- Effective communication of findings is essential. Data analysts often create reports and visualizations to convey insights to stakeholders. Python libraries like Matplotlib, seaborn and tools like Jupyter Notebooks facilitate the creation of informative reports.

Automation and Scaling:

- For large datasets or repetitive tasks, automation is

Date :

crucial. Python's scripting capabilities and libraries for data analysis and manipulation enable automation of data processing pipelines and workflows.

17.1 NumPy and Pandas for Data Manipulation

NumPy and Pandas are two fundamental Python libraries commonly used in data analysis and scientific computing. They provide essential tools and data structures for working with numerical data, performing data analysis.

Numpy (Numerical Python):

NumPy is a Python library for numerical and array-based operations. It provides support for multi-dimensional arrays and matrices, along with a vast collection of mathematical functions to operate on these arrays efficiently.

Key features and Use Cases:

Arrays:

NumPy's primary data structure is the 'ndarray' (n-dimensional array). These arrays are homogeneous and can store elements of the same data type, making them efficient for numerical computations.

Element-Wise Operations:

NumPy allows you to perform element-wise operations on arrays, making mathematical computations more straightforward and efficient.

Date :

Array slicing and indexing:

You can access and manipulate specific elements or subsets of arrays using indexing and slicing.

Broadcasting:

NumPy enables operations on arrays with different shapes, automatically aligning dimensions when possible.

Mathematical Functions:

NumPy Provides a wide range of mathematical functions, including basic arithmetic, linear algebra, statistics and more.

Integration with Other libraries:

NumPy serves as a foundational library for many other scientific computing libraries in Python, such as Scipy and scikit-learn.

Example of NumPy Usage:

python

```
import numpy as np
#creating NumPy arrays
a = np.array([1,2,3,4,5])
b = np.array([5,6,7,8,9])
#Element-wise addition
```

Date :

```
result = a+b
# computing the mean of an array
mean = np.mean(a)
```

Array Creation:

```
import numpy as np
# Creating a NumPy array
data = np.array([1,2,3,4,5])
```

Element-wise Operations:

```
# Element-wise addition
result = data+2
```

Array slicing:

```
# slicing an array
sub_array = data[1:4]
```

Broadcasting

```
# Broadcasting : Adding a scalar to an array
data = np.array([1,2,3])
result = data + 2
# result : [3,4,5]
```

All above NumPy excels at numerical computations and array operations.

Date :

Pandas:

Pandas is a powerful Python library for data manipulation and analysis. It introduces two primary data structures:
Series (for one-dimensional data) and Dataframe (for two-dimensional, tabular data)

key Features and Use cases:

DataFrame:

The Pandas Dataframe is a versatile data structure that resembles a table or spreadsheet. It can store data of various types, including numerical categorical, and text data.

```
import pandas as pd
data = {'Name': ['Alice','Bob','charlie'],
        'Age' : [25,30,35]}
df = pd.DataFrame(data)
```

Data cleaning:

Pandas provides functions for handling missing data, duplicate data, and data type conversions, making data cleaning and preparation more manageble.

```
df.dropna()
df.drop_duplicates()
```

Data Selection and filtering:

You can easily select specific rows and columns from a DataFrame based on conditiona, labels or

Date :

positions.

```
filtered_data = df[df['Age'] > 30]
```

Merging and Joining:

Data from different sources can be merged or joined together using SQL- like operations.

```
merged_df = pd.merge(df1, df2, on='common_column')
```

Time Series Analysis:

Pandas has excellent support for time series data, including date and time handling, resampling, and rolling calculations.

Aggregation and Grouping:

Pandas allows you to perform aggregation operations on grouped data.

```
df.groupby('category')['value'].mean()
```

Matplotlib:

Data Visualization with Matplotlib:

Basic Plot Types:

Matplotlib supports various plot types, including line plots, scatter plots, bar charts, histograms and pie charts.

Date :

```
import matplotlib.pyplot as plt

# creating a simple line plot

x = [1,2,3,4,5]
y = [10,12,5,8,9]
plt.plot(x,y)
plt.xlabel('x-axis')
plt.ylabel('Y-axis')
plt.title('simple Line Plot')
plt.show()
```

Customization:

You can customize every aspect of a plot, such as labels, colors, legends, titles, and axis properties.

Subplots:

Matplotlib allow you to create multiple plots within a single figure, which is useful for visualizing multiple datasets or comparisons.

Seaborn:

Data Visualization with Seaborn:

Statistical plots:

Seaborn simplifies the creation of complex statistical plots. For example: 'sns.scatterplot', 'sns.barplot', 'sns.boxplot', and 'sns.pairplot'

Date :

are designed to provide insights and into data distributions and relationships.

```
import seaborn as sns

sns.set(style = 'whitegrid')
tips = sns.load_dataset('tips')
sns.lmplot(x='total_bill', y='tip', data=tips
plt.title('scatter plot with Regression Line')
plt.show()
```

Color Palettes:

Seaborn provides a wide range of color pallettes that make your plots visually appealing.

Integration with Pandas:

Seaborn seamlessly integrates with Pandas Dataframes, making it easy to visualize your data directly from Dataframes.

Date :

18. Machine Learning and AI

18.1 Introduction to Machine Learning

Machine Learning (ML) is a branch of artificial intelligence (AI) that focuses on developing algorithms and models capable of learning from data and making predictions or decisions without being explicitly programmed.

Learning from Data:

ML algorithms learn patterns and relationships from historical data. The more data they have, the better they can generalize and make predictions on new, unseen data.

Types of Machine learning:

1) Reinforcement Learning
2) Supervised Learning
3) Unsupervised Learning

Reinforcement Learning:

In reinforcement learning, an agent learns to make sequences of decisions in an environment to maximize a reward signal. Its commonly used in game playing, robotics, and autonomous systems.

Supervised Learning:

Date :

In Supervised learning, the algorithm is trained on labeled data, where the input data is paired with corresponding target labels. It lears to map inputs to output, making it suitable for tasks like classification and the regression.

Unsupervised learning :

Unsupervised learning deals with unlabeled data It aims to discover hidden patterns or structures in the data often through techniques like clustering or dimensionally reduction.

Applications of Machine learning :

ML has a broad range of applications, including :

- Image and speech recognition
- Natural language processing (e.g chatbots and language translation)
- Recommender systems (e.g movie recommendation on NetFlix)
- Predictive analytics (e.g stock price forecasting)
- Healthcare diagnostics (e.g identifying diseases from medical images)

18.2 Using Libraries like scikit-learn and Tensorflow:

Machine learning libraries provide tools and frameworks to simplify the development of ML models.

Date :

Two widely used libraries are scikit-learn and TensorFlow

Scikit-learn:

This Python library is an excellent choice for traditional machine learning tasks. It offers a vast collection of tools for data preprocessing, feature selection, model selection, and evalulation. Scikit-learn supports various ML algorithms, making it accessible for both beginners and experienced data scientists.

Data Preprocessing:

Scikit-learn allows you to clean, preprocess, and transform your data easily. For example, you can scale features to have zero mean and unit variance using the 'StandardScaler'.

```
from sklearn.preprocessing import standardScaler
scaler = standardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

Machine Learning Models:

It includes a variety of machine learning algorithms like decision trees, support vector machines, and more. For instance, you can create a simple decision tree classifier:

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
```

Date :

```
clf.fit(x_train, y_train)
```

Model Evaluation:

Scikit-learn provides tools to evaluate the performance of your models. You can calculate metrics like accuracy, precision, recall, and F1-score:

```
from sklearn.metrics import accuracy_score
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
```

TensorFlow:

TensorFlow is an open-source machine learning framework developed by Google. It specializes in deep learning and neural networks. TensorFlow offers both high-level APIs for quick model development (e.g keras) and lower-level APIs for more fine grained control.

Neural Networks:

TensorFlow is known for its neural network capabilities. You can create complex neural network architectures like convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

```
import tensorflow as tf

model = tf.keras.Sequential([ tf.keras.layers.Dense
(128, activation = 'relu', input_shape = (784,)),
```

Date :

```
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(10, activation = 'softmax')])
```

Training:

TensorFlow provides tools for taining neural networks using various optimization algorithms, loss functions and custom training loops.

Deployment:

It supports deployment to various platforms, including mobile devices and the web.

```
import tensorflow as tf from tensorflow.keras
import layers, datasets

# Load the CIFAR-10 dataset (x_train, y_train),

(x_test, y_test) = datasets.cifar10.load_data()
x_train, x_test = x_train/255.0, xtest/255.0

model = tf.keras.Sequential([layers.Conv2D(32, (3,3),
        activation ='relu'), layers.Dense(10)])

# compile the model

model.compile(optimizer ='adam', loss = tf.keras.losses
.SparseCategoricalCrosee ntrophy(from_logits = True),

metrics = ['accuracy'])
```

Date :

```
model.fit(x_train, y_train, epochs = 10, Validation_data
= (x_test, y_test))
```

18.3 Simple Machine learning Example:

Prediction Iris Flower Species:

Step1 Data Preparation:

- Dataset : We'll use the famous Iris dataset, which contains measurements of special length, sepal width, petal length and petal width for three species of iris Flowers : setosa, versicolor, and verginica.
- Data cleaning : Ensure there are no missing values or outlie
- Data spliting : Divide the dataset into two parts - a training set and a testing set. Typically, a common split is 80% for training and 20% for testing.

step2 : choose an algorithm:

In this case, we'll use a simple classification algorithm, such as a decision tree classifier.

Step 3: Training the Model.

Feed the training data (features like sepal length, sepal width, petal length, and petal width) and their corresponding labels (iris species) into the decision tree classifier. The algorithm will learn the patterns and relationships between the features and the species during training

Date :

Step 4: Evaluation:

- Use the testing dataset to access the model's performance.
- Calculate metrics like accuracy, precission, recall, and F1-score to understand how tell the model is classifying iris Flowers.

Step 5 Making predictions:

- Once the model is trained and evaluated, it can be used to predict the species of iris flowers when given new medsurements.

Step 6 Fine tuning:

- Adujust hyperparameters of the decision tree, like the maximum depth or minimum samples per leaf, to improve the model's performance.
- Consider tying different algorithms or ensemble methods (e.g Random Forest) to see if they yeild better results.

Here's a simplified python code snippet using scikit-learn to perform this task.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
From sklearn.metrics import accuracy_score

#Load the Iris dataset
```

Date :

```
iris = load_iris()
x = iris.data
y = iris.target

#split the data into training and testing sets.

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size
=0.2, random_state=42)

# Create a Decision Tree classifier.

clf = DecisionTreeclassifier()

# Train the model on the training data.

clf.fit(x_train, y_train)

#Make predictions on the testing data
y_pred = clf.predict (x_test)

#Evaluate the model's accuracy

accuracy = accuracy_score (y_test, y_pred)
print ("Accuracy:", accuracy)
```