**Python Notes**

**1. Introduction to Python**

* **What is Python?**
  + High-level, interpreted, general-purpose programming language.
  + Supports multiple programming paradigms: procedural, object-oriented, and functional programming.
* **Features of Python:**
  + Simple and easy-to-read syntax.
  + Dynamically typed.
  + Large standard library.
  + Cross-platform (works on multiple OS).
  + Supports integration with other languages.

**2. Variables and Data Types**

* **Variables:**
  + Used to store data values.
  + Dynamic typing: No need to declare the type of variable explicitly python itself can maintain it.
* **Data Types:**

Datatypes are used to specify that which the variable can have which type.

* + **Numeric Types**: int, float, complex
  + **Sequence Types**: list, tuple, range
  + **Text Type**: str
  + **Set Types**: set, frozenset
  + **Mapping Type**: dict
  + **Boolean Type**: bool
  + **Binary Types**: bytes, bytearray, memoryview

**1.Numeric Types**: Numeric types are used to represent numbers in Python.

# Numeric Types

x = 10 # Integer  
y = 3.14 # Float  
z = 2 + 3j # complex  
  
# Hera i am Printing the variables  
print("Integer:", x)  
print("Float:", y)  
print("Complex:", z)

**2.Sequence data types :**

**list:** A mutable, ordered collection of elements. Lists are enclosed in square brackets [].

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

**tuple**: An immutable, ordered collection of elements. Tuples are enclosed in parentheses ().

numbers = (2,4,3)  
print(numbers)

**range**: Represents a sequence of numbers, commonly used in for-loops. It is immutable.

numbers = range(1, 5)

print(numbers)

**3.Text Type**: A string is a sequence of characters enclosed in either single quotes ' or double quotes ".

str = “mani”

print(str)

4. **Set Types :**

**set**: A mutable collection that does not allow duplicates. Sets are created using curly braces {} or the set() function.

unique\_numbers = {1, 2, 3, 4, 5}

**frozenset**: An immutable version of a set. Once a frozenset is created, its elements cannot be modified.

immutable\_set = frozenset([1, 2, 3, 4])

5. **Mapping Type:**

**dict**: A dictionary is an unordered collection of key-value pairs. Keys are unique, and values can be of any data type. Dictionaries are created using curly braces {} with key-value pairs separated by colons.

person = {"name": "John", "age": 30}

6. **Boolean Type:**

**bool**: Boolean type can hold one of two values: True or False.

str = “mani”

print(str)

is\_active = True

is\_not = False

print(is\_active)  
print(is\_not)

7. **Binary Types**:

Binary types are used to represent binary data in Python.

**bytes**: Immutable sequence of bytes. Used to represent raw binary data.

str = b”Mani”

print(str)

**bytearray**: Mutable sequence of bytes. Like bytes, but allows modification.

arrs = bytearray([65, 66, 67])

print(arrs)

**Strings :** String is a collections of characters in python. characters enclosed within single quotes (') or double quotes ("). Strings are immutable, meaning once they are created, their values cannot be changed.

str1 = 'Hello, World!'

str2 = "Python is awesome!"

**String methods in python :**

upper(): Converts all characters in the string to uppercase.

text = "hello mani"  
print(text.upper())

**lower**(): Converts all characters in the string to lowercase.  
print(text.lower())

**capitalize**(): Capitalizes the first character of the string and makes the rest lowercase.  
print(text.capitalize())

**title**(): Capitalizes the first character of each word in the string.  
print(text.title())

**swapcase**(): Swaps the case of each character in the string.  
print(text.swapcase())

**find**(): Returns the lowest index where a substring is found, or -1 if not found.  
print(text.find("World"))

**replace**(): Replaces a substring with another substring.  
print(text.replace("World", "Python"))

**count**(): Returns the number of occurrences of a substring in the string.  
text = "Hello, World! Hello!"  
print(text.count("Hello"))

**startswith**(): Returns True if the string starts with the specified prefix.  
print(text.startswith("Hello"))

**len():** Return the length of the string.

**3. Operators**

**In Python, operators are symbols used to perform operations on variables and values. Python supports several types of operators:**

**1. Arithmetic Operators**

These operators perform basic mathematical operations like addition, subtraction, multiplication, etc.

* + (Addition): Adds two operands.
* - (Subtraction): Subtracts the second operand from the first.
* \* (Multiplication): Multiplies two operands.
* / (Division): Divides the first operand by the second.
* // (Floor Division): Divides and returns the largest integer smaller than or equal to the result.
* % (Modulus): Returns the remainder when the first operand is divided by the second.
* \*\* (Exponentiation): Raises the first operand to the power of the second.
* # 1. Addition (+)  
  a = 5  
  b = 3  
  result = a + b  
  print(result) # Output: 8  
    
  # 2. Subtraction (-)  
  a = 5  
  b = 3  
  # result = a - b  
  print(result) # 5-3 Output: 2  
    
  # 3, Multiplication (\*)  
  a = 5  
  b = 3  
  # result = a \* b  
  print(result) # Output: 15  
    
  # 4. Division (/)  
  a = 5  
  b = 2  
  result = a / b  
  print(result) # Output: 2.5  
    
  # 5. Floor Division (//)  
  a = 5  
  b = 2  
  result = a // b  
  print(result) # Output: 2  
    
  # 6. Modulus (%)  
  a = 5  
  b = 2  
  result = a % b  
  print(result) # Output: 1

# 7. Exponentiation (\*\*)  
a = 2  
b = 3  
# result = a \*\* b  
print(result) # Output: 8

**2. Comparison (Relational) Operators**

These operators compare two values and return a boolean result (True or False).

* == (Equal to): Returns True if the values of two operands are equal.
* != (Not equal to): Returns True if the values of two operands are not equal.
* > (Greater than): Returns True if the first operand is greater than the second.
* < (Less than): Returns True if the first operand is less than the second.
* >= (Greater than or equal to): Returns True if the first operand is greater than or equal to the second.
* <= (Less than or equal to): Returns True if the first operand is less than or equal to the second.

# 1. Equal to (==)  
a = 5  
b = 5  
result = a == b  
print(result) # Output: True  
  
# 2. Not equal to (!=)  
a = 5  
b = 3  
result = a != b  
print(result) # Output: True  
  
# 3. Greater than (>)  
a = 5  
b = 3  
result = a > b  
print(result) # Output: True  
  
# 4. Less than (<)  
a = 5  
b = 7  
result = a < b  
print(result) # Output: True  
  
# 5. Greater than or equal to (>=)  
a = 5  
b = 5  
result = a >= b  
print(result) # Output: True  
  
# 6. Less than or equal to (<=)  
a = 5  
b = 7  
result = a <= b  
print(result)

**3. Logical Operators**

These operators are used to combine conditional statements.

* and: Returns True if both operands are True.
* or: Returns True if at least one operand is True.
* not: Reverses the logical state (i.e., returns True if the operand is False, and vice versa).

# 1. AND+  
a = True  
b = False  
result = a and b  
print(result) # Output: False  
  
# 2. OR  
a = True  
b = False  
result = a or b  
print(result) # Output: True  
  
# 3.NOT  
a = True  
result = not a  
print(result) # Output: False

**4. Assignment Operators**

These operators are used to assign values to variables.

* =: Assigns the value of the right operand to the left operand.
* +=: Adds the right operand to the left operand and assigns the result to the left operand.
* -=: Subtracts the right operand from the left operand and assigns the result to the left operand.
* \*=: Multiplies the left operand by the right operand and assigns the result to the left operand.
* /=: Divides the left operand by the right operand and assigns the result to the left operand.
* //=: Floor divides the left operand by the right operand and assigns the result to the left operand.
* %=: Takes the modulus of the left operand by the right operand and assigns the result to the left operand.
* \*\*=: Raises the left operand to the power of the right operand and assigns the result to the left operand.
* # 1. Assign (=)  
  x = 5  
  print(x) # Output: 5  
    
  # 2. Add and Assign (+=)  
  x = 5  
  x += 3  
  print(x) # Output: 8  
    
  # 3. Subtract and Assign (-=)  
  x = 5  
  x -= 3  
  print(x) # Output: 2  
    
  # 4. Multiply and Assign (\*=)  
  x = 5  
  x \*= 3  
  print(x) # Output: 15  
    
  # 5. Divide and Assign (/=)  
  x = 10  
  x /= 2  
  print(x) # Output: 5.0  
    
  # 6. Modulus and Assign (%=)  
  x = 10  
  x %= 3  
  print(x) # Output: 1

**5. Bitwise Operators**

Bitwise operators are used to perform operations on binary numbers.

* & (AND): Performs a bitwise AND.
* | (OR): Performs a bitwise OR.
* ^ (XOR): Performs a bitwise XOR (exclusive OR).
* ~ (NOT): Performs a bitwise NOT (inverts the bits).
* << (Left shift): Shifts the bits to the left by the specified number of positions.
* >> (Right shift): Shifts the bits to the right by the specified number of positions.

# 1. AND (&)  
a = 5 # In binary: 0101  
b = 3 # In binary: 0011  
  
result = a & b # Performs bitwise AND  
print(result) # Output: 1 (In binary: 0001)  
  
# 2. OR (|)  
a = 5 # In binary: 0101  
b = 3 # In binary: 0011  
  
result = a | b # Performs bitwise OR  
print(result) # Output: 7 (In binary: 0111)  
  
# 3. XOR (^)  
a = 5 # In binary: 0101  
b = 3 # In binary: 0011  
  
result = a ^ b # Performs bitwise XOR  
print(result) # Output: 6 (In binary: 0110)  
  
# 5. Left Shift (<<)  
a = 5 # In binary: 0101  
  
result = a << 1 # Shifts bits of a to the left by 1  
print(result) # Output: 10 (In binary: 1010)  
  
# 6. Right Shift (>>)  
a = 5 # In binary: 0101  
  
result = a >> 2 # Shifts bits of a to the right by 1  
print(result) # Output: 2 (In binary: 0010)

**6. Membership Operators**

These operators test whether a value is in a sequence (like a list, tuple, string, etc.).

* in: Returns True if the value is found in the sequence.
* not in: Returns True if the value is not found in the sequence.

# Membership operators  
a = [1, 2, 3, 4, 5]  
print(3 in a) # True  
print(6 not in a) # True

**7. Identity Operators**

These operators compare the memory locations of two objects.

* is: Returns True if both operands refer to the same object in memory.
* is not: Returns True if both operands refer to different objects.

# 1. is  
a = [1, 2, 3]  
b = a  
  
result = a is b  
print(result) # Output: True  
  
a = [1, 2, 3]  
b = [1, 2, 3]  
  
result = a is b  
print(result) # Output: False  
  
# 2. is not  
a = [1, 2, 3]  
b = [1, 2, 3]  
  
result = a is not b  
print(result) # Output: True  
  
a = [1, 2, 3]  
b = a  
  
result = a is not b  
print(result) # Output: False

**3. Control Flow in Python**

**if, elif, and else statements:**

**if :** this statement is executes when condition is true.

**elif** : it is also called as “else if” it will come after when if condition is not true. We can write more then one else if conditions based on requirement.

**else** : Executes a block of code when all previous conditions are False.

age = 20  
  
if age >= 18:

print(“You are an adult.")

elif age > 13:  
 print("You are a teenager.")  
else:  
 print("You are a child.")

**1. For Loop with break and continue**

The for loop is used to iterate over a sequence (like a list, tuple, string, or range).

**2. While loop :** The while loop repeats as long as the condition is True.

**Break** : it will break loop based on the condition.

**Continue** : it will skip iteration based on the condition.

# Membership operators  
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit)

for i in range(10):

if(i == 5):

break

print(i)

for i in range(10,20,2):

if(i == 16):

continue

print(i)

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

**4. Functions**

Functions allow you to group code into reusable blocks. They can take inputs (parameters) and return outputs.

1. **User-Defined Functions**

def add(a,b):  
 a = a+b  
 return a  
print(add(2,3))

def multi(a,b):  
 print(a\*b)  
multi(2,4)  
  
def nam():  
 print("mani")  
nam()

In Python, functions can be categorized based on their definition, arguments, and return behavior.

2. **Built-in Functions**

These are functions that are provided by Python and are always available to use. We no need to define these we can directly use it.

**print**(): Prints output to the console.

**len**(): Returns the length of an object.

**type**(): Returns the type of an object.

**max**(): Returns the maximum of the provided values.

# built in funtions  
# print()  
print("mani")  
  
# len()  
name = "manibabu"  
length\_name = len(name)  
print(length\_name)  
  
#/type  
a = 4  
print(type(a))  
  
# max()  
list = [2,4,2,9,5,2,5,8]  
print(max(list))

3. **Functions with Default Arguments**

These are functions where some parameters have default values

def greet(name="Guest"):  
 print(name)  
  
greet("Alice")  
greet()

4. **Functions with Arbitrary Arguments**

These functions allow you to pass a variable number of arguments

def add\_numbers(\*args):  
 return sum(args)  
# Functions with Arbitrary Arguments  
print(add\_numbers(1, 2, 3))  
print(add\_numbers(10, 20, 30, 40))

# Lambda funtions  
add = lambda x, y: x + y  
print(add(3, 5))

5. **Lambda Functions**

Lambda functions are small, anonymous functions defined using the lambda keyword.

# Lambda funtions  
add = lambda x, y: x + y  
print(add(3, 5))

**Keywords in python**

**1. False :**

* Represents the boolean value False, which is used in conditional statements to indicate that a condition is false.

**2. True :**

* Represents the boolean value True, used in conditional statements to indicate that a condition is true.

**3. None :**

* Represents the absence of a value or a null value.

4. **and :**

* A logical operator used to combine two conditions. The result is True only if both conditions are true.

**5. as :**

* Used to create an alias when importing a module or handling exceptions

**6. assert :**

* Used for debugging purposes to check if a condition is true. If not, it raises an AssertionError.

**7. async :**

* Used to define asynchronous functions. It marks a function as asynchronous and allows the use of await inside it.

**8. await**

* Used to call asynchronous functions. It pauses the execution of the function until the result is available.

**9. break**

* Terminates the program or a loop based on the condition.

**10. class**

* Defines a class, which is a blueprint for creating objects (instances)

**11. continue**

* Skips the current loop iteration and proceeds with the next iteration.

**12. def**

* Used to define a function.

**13. del**

* Deletes a variable, item from a list, or attribute from an object.

**14. elif**

* Stands for "else if" and is used in conditional statements to check multiple conditions.

**15. else**

* It will executer when if or elif condition is not true.

**16. except**

* Used in exception handling to catch exceptions raised in a try block.

**17. finally**

* Used to define a block of code that will execute no matter what, whether an exception is raised or not.

18. **for**

* It is a loop used to iterate the items.

**19. from**

* Used to import specific items from a module

**20. global**

* Declares a variable as global, meaning it is accessible outside of the current function or scope.

**21. if**

* Starts a conditional statement to check if a condition is true

**22. import**

* Used to import modules into the current namespace

**23. in**

* Used to check if a value exists in a sequence

**24. is**

* Used to check if two objects refer to the same memory location

**25. lambda**

* Defines an anonymous function. Used to do simple tasks.

**26. nonlocal**

* Used in nested functions to modify a variable in the nearest enclosing scope (excluding global scope).

def outer():

x = 10

def inner():

nonlocal x

x = 20 # Modifies 'x' in the outer function

**27. not**

* A logical operator used to negate a condition. It returns True if the condition is False and vice versa.

**28. or**

* A logical operator used to combine two conditions. The result is True if at least one condition is true.

**29. pass**

* A placeholder statement used when a statement is syntactically required, but you do not want to execute any code.

**30. raise**

* Used to raise exceptions explicitly

**31. return**

* Exits from a function and optionally returns a value

def add(a, b):

return a + b

**32. try**

* Used to start a block of code that may raise an exception, followed by an except block to handle the exception.

**33. while**

* Starts a while loop, which continues as long as a condition is true.

**34. with**

* Used with a context manager (like file handling) to ensure that resources are properly cleaned up when done.

**35. yield**

* Used to produce a value in a generator function, allowing the function to resume execution later.

**Modules and Packages**

**Moduule :**

A module is a single file containing Python code, which can define functions, classes, variables, and runnable code.

For creating a module we need save that file with **module.py** extention.

# mymodule.py

def greet(name):

print(f"Hello, {name}!")

**for import those module into current file**

import mymodule

mymodule.greet("Alice")

**Package in Python**

A **package** is a collection of related modules bundled together in a directory hierarchy.

# here we import entire package

import my\_package.module1

# here we can use functions from the module

my\_package.module1.some\_function()

# for Importing specific functions

from my\_package.module1 import some\_function

**File handling in python**

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

Syntex : open(file\_path, access\_mode)

Example: open(“file\_name.txt”,”r”)

File1 = open(“hello.txt”,”r”)

file\_path = which means there we can place the path of our file.

r = means access mode the open file is which operation is performed.

**The is four different methods or modes for opening a file:**

* **“r” – Read –** Open a file and reading purpose only, we can’t able to do any operation.
* **“a”-**Append – Opens a file for appending, used for adding file not override the before file.
* **“w”-**Write- Open a file for writing, it will override the already existing file.
* **“x” –** create – creates the specified file.

**Reading a file in python:**

After we open a file, we use the read() method to read its contents.

Ex:

file1 = open(“hello.txt”,”r”)

Print(file1.read())

**Output :**

Hello manibabu

How are you..?

**For read only parts of file:**

By default read() method returns the whole text, but we can also specify how many characters we want to return.

**Ex**:

file = open('note.txt', 'r')  
  
content = file.read()  
print(content)  
file.close()

**output :**

Hello, Manibabu!

how are you....

I know you very good boy.

**readline**() : this method is used to read the single line in file.

**Ex :**

file = open('note.txt','r')  
line = file.readline()  
print(line,end=' ')

file.close()

**output : Hello, Manibabu!**

**Writing a file :** This is used write a new file into the existing record file. It will override the existing record to the new record.

**Syntex :**  
# writing a file  
file = open('note.txt', 'w')  
file.write('Hello, Manibabu!\n')  
file.write('how are you....\n')  
file.write('I know you very good boy.\n')  
file.close()

**Appending a file:** This is used to appending a record into the existing file. In this process It will add on new record into the existing record. So before record and new record are exist in the file.

**Syntex :**

file = open('note.txt', 'a')  
file.write('This is add new record not override the existing record.\n')  
file.close()

**Advantages of File Handling in Python:**

1. **Simple and Easy to Use:**
2. Python provides an simple interface for working with files. The built-in open() function and file methods like .read(), write(), and close() make file operations straightforward and easy implement.
3. **Flexible File Operations**
4. **Platform Independent**
5. **Support for Text and Binary Files**
6. **Error Handling**
7. Python provides built-in exception handling using try-except blocks, allowing you to catch and manage file-related errors

**Disadvantages of File Handling in Python:**

* **File Operations Can Be Slow for Large Files**
* **Error-Prone with Improper File Management:**

Which means if we forgot to close or open it in the wrong mode, it can lead to memory leaks, data loss, or corrupt files.