Notes written by: Yasir Baig

Introduction To Python.

- What is Python? (Interpreter language):

Python is a programming language. It is used to give instructions to a computer so it can do things like solving problems, building websites, analyzing data, making games or creating AI systems.

- Why Python is important?
  - It is easy to read and write
  - It is Beginner-Friendly
  - Powerful and flexible.

- What is Python history?

Python was created by Guido Van Rossum in 1989. It was officially released in 1991. He wanted to make language that is easy and fun to use.

- Where is Python used?

It is now used in many fields. Web development, data science, Machine learning Automation Game development, cybersecurity etc.

- Why Python is important?

easy to learn, used by big companies, many libraries, helps in modern field like AI, data science, Automation, Very huge community.

- What Kind of Python is there?

Python 3, CPython, Jython, PyPy, MicroPython.

Print Statement In Python.

- The Print() function prints the given object to the standard output device. (screen) or to the text stream file.

Print ("Mirza Yasir Abdullah Baig")

It Prints name or anything in terminal.

∴ The extension for Python file is .py.

Print ( ) → Paratheses.

In Python Print are shown in line by line format.

Default output
- Yasir
- 24
- Bachelor

in this way

You can use single statement or ' '

Print ('Yasir')

If you want to show in 1 line

Print ("Yasir")
Print ("Education", end = " ")
Print (24)

You can print any data type using print

print ("India," "Pakistan", "America")

or

Print ("Yasir", 24, 3.12, True)

Sep = ' ' / Automatically Provide space. You can use anything i.e (/, -, ->, %)

Comments In Python

- In computer programming, comments are hints that we use to make our code more understandable.
- Comments are completely ignored by the Interpreter.

- Types of comments
  - → Single line comments
  - → Multi line comments

```
Print ( "Yasir" ) ## This is comment.
## Print ( "Baig")
```

These are single line comments.

```
"""
Print ( """ This is a large
        Paragraph """)
"""
```

(D)oc - Strings are also called double line comments.

Escape Sequences.

Escape characters/sequences are illegal chars for Python & never get printed as part of output. When backslash is used in python programming, it allows the program to escape next characters.

- \' - Single quotation ("My name is 'Yasir'.")
- \" Double quotation ("My name is \"Yasir\".")
- \\ Backslash
- \n New Line
- \t Tab (use for space)
- \b Backspace.

Variables In Python

- Python variables are simply containers for storing data values. Unlike other values languages such as Java, Python has no commands for declaring a variable, so you create one the moment you first assign a value to it.

```
x = 5   # x is variables.
Print (x)
name_Yasir = "Yasir"
Print (name_Yasir)
x = 2,   y = 3   z = x+y
Prin(z)
```

Rules for Variables.

a-z, A-Z valid

@bc → not valid

--- → valid

ab_c → valid

*123 → not valid

abc123 → valid

ab.cd → not valid

myVar → valid

Snake case first_name

Method to Print variable

```
age = 24
Print ("my age is", age")
```

F-String

```
name = "Yasir"
Print(f"my name is {name}")
```

Data types In Python

- Data Types specify what kind of value a variables holds. They also determine what operations (mathematical, rational, logical) can be Performed on that value.

Main Data Types In Python

→ Numerical Type → Int, float, complex

* Integer (whole number)

```
num1 = 3
Print (num1)
```

* Float (decimal numbers)

```
num2 = 3.14
Print (num2)
```

* Complex number (real + imaginary)

```
num3 = 2 + 3j
Print (num3)
```

* Boolean → True / False

```
is_active = True
Print (is_active)
```

* String → Text endosed in quotes

```
message = "Yasir"
Print (message)
```

* List → Ordered, changeable, allows duplicates

```
fruits = ["apple", "banana", "orange"]
Print (fruits)
```

* Tuple → ordered unchangeable (immutable)

```
coordinates = (10, 20)
Print (coordinates)
```

* Set → Unordered, no duplicates

```
unique-num = {1, 2, 3, 4, 5}
Print {unique-num}
```

* Dic-num = { "name": "Yasir }

```
Print (Dic-num)
```

Type Conversion / Type Casting

Type Conversion (or type casting) means changing the "data type" of a value.

- Python Provide built-in-functions like 'int()', 'float()', 'str()', 'list()', for type casting.
- It's useful when you need to perform operations that requires values to be in specific type.

- Checking / Changing the data type using type()

```
x = 4
Print (x, type (x)   # output 4, int
```

- Type Conversion examples

```
x = int( "100")
y = int ( "200")
z = x + y
Print (z)    # output 300
```

Summary of Type Casting functions in Py

int() - Converts to int (whole number)
str() - converts to str
Float() - Converts to decimal number
List() - converts to list
dict() - converts to dictionary (valid format)
bool() - convert to True or False.

Input Func In Python

The input( ) Function is used to take "user Input" whatever the user types is stored as a "string" by default.

- Simple example

```
x = input("Enter Your name: ")
Print("You entered:", x)
```

Do a Simple Example.

```
name = input("Enter name:")
gender = input("Enter gender:")
age = int(input("Enter age: "))
```

Display using F-Strings.

```
Print(f"My name is {name}, my age is
    {age}, and my gender is {gender}")
```

Note:

- All inputs are stored as string by default.
- use int(), float(), & bool() to convert them if needed.
- Example = age = int(input("Enter number"))

Operators In Python

- What are Operators?

Operators are used to perform operations on variables and values.

- Different Type of Operators.
  - → Arithematic operators
  - → Assignment operators
  - → Comparison operators
  - → Logical operators
  - → Identity operators
  - → Membership operators
  - → Bitwise operators.

- Arithematic Operators.

Arithematic operators are used with numeric values to perform mathematical operations.

→ Addition (+)

```
a = 10 + 5
Print (a)   # output 15
```

→ Substraction (-)

```
a = 10 - 5
Print (a)   # output 5
```

→ Multiplication (*)

```
a = 10 *5
Print (a)   # output 50
```

→ Division (/)

```
a = 10/5
Print (a)   # output 2
```

→ Modulus (%)

→ Modulus (%)

* Modulus gives the remainder of division

```
a = 10 %3
Print (a)      * output 1
b = 5 %10
Print (b)      * output 5
c = 9%10
Print (c)      * output 9
```

→ Exponential (**)

used to raise a number to the power of another.

```
a = 5 **3
Print(a)       * output 125
```

→ Floor Division (//)

It gives the result In whole numbers (Integer Parts only)

```
a = 10 // 4
Print (a)      * output 2
a = -10 // 4
Print (a)      * output -3
```

• Assignment Operators

They are nothing, we already learn about them. There main purpose is to assign value like

a = 2 ⇒ a is assigned value (2)

Let's do some examples

```
x = 5
Print(x)      # output 5
x += 5
Print(x)      # output 10
x = 5
x -= 2
Print(x)      # output 3
x = 5
x *= 3
Print(x)      # 15
x = 5
x /= 2
Print(x)      # 2.5
x = 5
x %= 3
Print(x)      # output 2
x = 5
x //=2
Print(x)      # output
x=5
x** =2
Print(x)      # output
```

- Here is a table for Arthematic Operators & for assignment operators :- These Tables will build more understady

Arithmetic Table

Operator	Name	Example
+	Addition	x+y
-	Substraction	x-y
*	Multiplication	x*y
/	Division	x/y
%	Modulus	x%y
**	Exponentiation	x**y
//	floor Division	x//y

Assignment Table

Operators	Example	Same As
=	x=5	x=5
+=	x+=3	x=x+3
-=	x-=3	x=x-3
*=	x*=3	x=x*3
/=	x/=3	x=x/3
%=	x%=3	x=x%3
//=	x//=3	x=x//3
**=	x**=3	x=x**3

- **Comparison Operators**

Comparison operators are used to compare two values. The Result is always True / False.

>, <, >=, <=, ==, !=

These are some comparison operators we will understand with some examples & Tables.

Comparison Operator

Operator	Name	Example
>	Greater than	x > y
<	less than	x < y
>=	Greater than or equal to	x >= y
<=	less than or equal to	x <= y
==	Equal	x == y
!=	Not equal	x != y

-) > Greater than
  Print (10 > 5) True
-) < less than
  Print(4 < 9) True
-) ~~Print~~ Less Greater than or equal to
  Print (6 <= 3) False
-) Greater than or equal to
  Print (7 >= 7) True
-) equal to (==)
  Print (5 == 5) True
-) Not equal to (!=)
  Print (8 != 10) True.

These are some examples for comparsion.

- Logical Operators.

logical operators are used to combine multiple conditional statements
∴ Same we will discuss some examples and a table for better understanding

Logical Operators

Operator	Description	Example
and	Return true if both true	x<10 and y<15
or	return true if one is true	x<15 or x<24
not	Reserve the result.	not(x<15 and x<120)

x = 10

y = 20

•) and – Returns True only if bothe are True.
Print (x < 15 and y < 25) * True.

•) or – Returns True if one is True
Print (x > 15 or y > 15) * True

•) not – Reserve the Result
Print (not(x<15 and y<25)) * False

AND Gate – Truth Table

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE – Truth Table

A	B	A or B
0	0	0
0	1	1
0	0	1
1	1	1

Conditional Statements

- What are conditional statements? (If, else, elif) allow us to control the flow of program. They help make decision based on conditions.

- Type of Conditional Statements
  - → If statement
  - → else statement
  - → elif statement
  - → nested If-else

* Example : check person eligible to vote

```
age = int(input("Enter your age: "))
If age >= 18:
    Print ("You can vote")
else:
    Print ("You cant vote")
```

* Example NO 2 " If else, elif.

```
num1 = int(input("Enter Num"))
num2 = int(input("Enter Num"))
If num1 > num2:
    Print ("Num 1 is greater")
elif num1 < num2:
    Print ("Num 2 is greater")
else:
    Print ("Both are equal")
```

: You can add many elif if your Program wants.

Loops In Python

- What are loops?

Looping means representing a block of code multiple times until a particular condition is satisfied.

- Main Type of loops In Python.
  - While loop
  - For Loop

While loop Syntax

```
while condition:
    code
    code
```

It will keep executing the block as long as the condition is True.

```
Print("It prints, when condition become false")
```

Example: (Print hello world 10 Time)

```
i = 1
while i< 10:
    Print ("Hello world")
    i = i + 1
```

Example: (Print even number from 1 to 20)

```
i = 1
while i<= 20:
    if i % 2 == 0:
        Print(i, end = " ")
    i = i+ 1
```

For loop syntax
for variable in range (start, stop, step):

Example:
for i in range (1, 20):
    Print(i)

Example (only odd)
for i in range (1, 20, 2):
    Print (i)

Example: Reverse Value
for i in range (10, 0, -1):
    Print (i)

Diff

For loop	While loop
• Syntax is easy • You know exactly how many time to run loop • Iterating over sequence	• Syntax is difficult • Dont know in advanced how many time to in loop • You want to run until condition is false.

Nested loop Example:
for i in range (1, 4):
    Print ("outer loop i = ", i)
    for j in range (10, 14):
        Print ("inner loop J = ", j)

Control Statements Python

- What are Control statements?

These statements are an essential aspect of any programming language. They are used to manage the flow of execution in a program based on certain conditions.

Python control Statements.

- Break
- Continue
- Pass

Break Statements.

The break statements used to terminate the loop immediately when certain condition is met.

Example:

```
for i in range (1, 11):
    Print (i)
    if i == 5:
        break    # output = 1, 2, 3, 4, 5
```

Continue statements:

It skip the current iteration & move to next one

Example

```
for i in range (1, 11):
    for if i == 5:
        continue    # skip 5
    Print(i)    # 1, 2, 3, 4, 6, 7, 8, 9, 10
```

Pass statement

It does nothing (used as placeholder for future code).

Lists In Python

What are lists?

lists are sequence data types in Python that can store multiple values in a single variable

lists are mutable (modifiable) and can hold elements of different type (list, float, str). They are similar to array but not flexible

* Creating an empty list

```
my_list = [ ]
```

* Creating a list with mixed data types

```
list = [1, 2, 3, "Yasir", True, 0.7]
Print(list)  # [1, 2, 3, "Yasir", True, 0.7]
```

* Accessing Elements From a list

```
x = [45, 21, 41, 2, 8, 1, 6]
Print (x[2])          # 2
Print (len(x))        # 7
Print (x[-1])         # 3
Print (type(x[2]))    # <class 'int'>
```

* Iterating Through a list

1- By Index

```
a = [1, 2, 43, 54, 5, 5, 6, 67, 4]
for i in range (len(a)):
    Print (a[i])
```

2- By value (cleaner)

```
for value in a:
    Print (value)
```

Built In Functions In list

```
a = [1, 2, 4, 55, 6, 6, 6, 6]
Print (len (a))
Print (sum(a))
Print (max(a))
Print (min(a))
```

* list Methods

```
my-list = [10, 20, 30, 40]
```

1- Append - Add items to the end

```
my-list.append (50)
Print (my-list)
```

2- Insert - Insert at position i

```
my-list.insert (2, 25) - then point
```

3- extend (iterable) - add multiple values.

```
my-list.extend ([60, 70]) then point.
```

4- remove - Remove first value

```
pri my-list.remove (25)
```

5 Pop - Removes & return items at index.

```
my.list-pop ()
```

6. Index - Return first index
Print (my_list.index(30))

7. Count (x) - Counts occurances of x
Print (my_list.count(10))

8. Sort() - Sort in asending order
num_list = [5, 2, 9, 1, 7]
num_list.sort()

9. Reverse() Reverse the list
num_list.reverse()
Print (num_list)

10. Copy() Returns a shallow copy
copy_list = num_list.copy()
Print (copy_list)

11. Clear - Empty the lists
copied_list.clear()
Print (copied_list)

List slicings
* What is slicing?
Slicing means extracting a part of the list using syntax.
list [start:end:step]

nums = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

* Basic Slicing

```
Print (nums [0:5])
Print (nums [:4])
Print (nums [5:])
```

* Step Slicing

```
Print( nums [::2])
Print( nums [1::2])
```

* Negative slicing (Reverse)

```
Print (nums [::-1])
Print (nums [-3:])
Print (nums [-5:-2])
```

* Why use Slicing?

for data cleaning, data extraction, reverseing lists, skipping elements, or efficient subsetting

last lines for list

- They are essential for managing & manipulating sequence of data.
- Use method like append, pop to modify the
- use slicing for powerful subsetting & Transformations.

Tuples In Python

* What are tuples in Python?

Tuples are similar to list, but with one key difference. Tuples are immutable once created. they can't be changed. Tuples supports indexing, slicing, iteration & nesting.
They have only 2 methods, count / index.

Creating tuple

```
tuple1 = ()
Print (tuple 1)

tuple2 = (1, 2, 3, 4)
Print (tuple 2)

tuple 3 = (1, 2, 3, "Yasir", True)
Print (tuple3)
```

Single Item tuple - comma is mandatory

```
tuple 4 = (1,)
Print (tuple 4)
```

Creating tuple from string

```
tuple 5 = tuple ("Yasir")
```

Creating Tuples From List

```
tuple 6 = tuple ([1, 2, 3, 4, 5])
Print (tuple6)
```

Accessing Tuple Elements

```
tuple11 = (1, 2, 3, 4, 5, 6)
Print ( tuple11[1])
Print (tuple11[-2])
```

Nested Tuple

```
tuple22 = (1, 2, 3, (4, 5))
Print ( tuple22[-1][0])
```

Tuple Immutability

```
deltuple = (1, 2, 3, 4, 5)
Print ( deltuple)
```

Tuple Operations : tup1 = (1, 2), tup2 = (3, 4)

- Concatenation

```
Print ( tup1 + tup2)
```

- Repetition

```
Print ( tup1 * 2)
Print (tup2 * 3)
```

Looping Through Tuple

```
Print ("Elements in tup1:")
for in in tup1:
    Print( i)
```

Membership Testing

```
Print ( "Is 1 in tup1?", 1 in tup1)
Print ( "Is 1 not in tup1?", 1 not in tup1)
```

- Tuples Built-In- Functions

function_tuple = (1, 2, 2, 4, 54, 65, 6)

Print ("Length:", len(function_tuple)
Print ("Minimum Value:", min len(function_tuple)
Print ("Maximum Value:", max(function_tuple)
Print ("Sorted (ascending):" sorted(function_tuple)
Print ("Sorted (desending): "sorted(function_tuple, reverse = True) "Return a list.

Important Points for

- Tuples are Immutable
- Tuples are only Readable
- Tuple use less values
- Tuples are faster for fixed data.
- Good For protecting data Integrity

Strings In Python.

★ What are strings in Python?
Strings are sequence of uni-code charaters
Example: "Yasir" = 'Y', 'a', 's', 'i', 'r'

. Creating strings
a = "Yasir Baig" - Double Quotes
Print (a)

b = 'Yasir Baig' - Single quotes
Print (b)

c = "It's raining". Double quotes handle single.
Print (c)

d = """ Yasir is an AI developer """
Print (d) - Triple quotes.

e = str ("Mirza Yasir Abdullah Baig")
Print (e) - using str - constructor.

• Accessing String charactters (Indexing)
nome = "Yasir"
Print (name)
Positive Indexing
Print (name [0]) # Y
Print (name [1]) # a
Print (name [2]) # s
Print (name [3]) # i
Print (name [4]) # r
Print (name [5]) # Error Index out of range

Negative Indexing

```
Print (name [-1]   # r
Print (name [-2]   # i
Print (name [-3]   # s
Print (name [-4]   # a
Print (name [-5]   # k
```

- Slicing Strings

```
text = " Mirza Yasir Abdullah Baig"

Print ( text [ 0:6]   # "Mirza "
Print ( text [ 2:]    # "rza Yasir Abdullah Baig"
Print ( text [ :5]    # "Mirza"
Print ( text [ : ]    # Full String
Print ( text [ 0: 6: 2 ]  # "Mra"
Print ( text [-6: -1: 2]  # " B"
Print ( text [ : : -1 ]   # Reversed String
Print ( text [ -1 : -7 : -1]  # last 6 char reversed
```

- Editing & Deleting string

```
S = " AI Machine Learning Journey"
Print (S)
```

- Reassigning is allowed

```
S= AI course
Print (S)
```

- Deleting specific character not allowed.
- Entire string deletion is allowed.

```
del s
```

- Strings Operators

- Arithematic

```
Str1 = "Yasir"
Str2 = "Baig"
Print (str1 + " " + Str2) # Concatenation
Print (Str1 * 3) # Repetition.
Print ("*" * 40)
```

Relational Operators

```
Print ("Yasir" == "Baig")
Print ("Abdullah" != "Mirza")
Print ("Islamabad" > "Kasur")
```

- logical Operators.

```
Print ("Yasir" and "Baig") - 'Baig'
Print ("Yasir" or "Baig") - 'Yasir'
Print ("" and "Baig") - falsy
Print ("" or "Baig") - 'Baig'
```

- looping In Strings

```
for char in "Mirza Yasir Abdullah Baig"
    Print (char)
```

- Membership Operators

```
S = "Mirza Yasir Abdullah Baig"
Print ('M' in S) # True
Print ('Z' in not in S) # True
```

- Common Strings Functions.

```
Str_function = "Mirza Yasir Abdullah Baig"
Print ( len (str_function))
Print ( max (str_function))
Print ( min (str_function))
Print ( sorted (str_function))
Print ( sorted (str_function, reverse = True))
```

- String Specific Methods.

```
example = "mirza Yasir abdullah Baig"
Print ( example . capitalize())
Print ( example . title())
Print ( example . upper())
Print ( example . lower())
Print ( example . swapcase())
Print ( example . count("a"))
Print ( example . find("Y"))
Print ( example . startswith ("mirza"))
Print ( example . endswith ("baig")
```

Split & Join => line = "Mirza Yasir Baig"

```
print ( line . split())
Print ( line . split("i"))
Print ( line . split("x"))
```

Join →

```
names = ["Mirza", "Yasir", "Abdullah"]
Print ( "-" . join (names))
```

Replace & Strip (Yours must do)

Dictionary In Python.

- What is dictionary?

A dictionary is the unordered, mutable & indexed collection. It stores data in key & value pairs.

Syntax - { key : value }

Why do we use dictionary?

- Fast data retrieval using keys
- Meaning ful key-value mapping
- Efficient data organization.

- Creating Dictionaries.

```
empty_dic = { }
Print (empty_dic)
Simple_dic = { "nome": "Yasir"}
Print ( Simple_dic)
```

Using dist constructor.

```
user = dict ( name = "Yasir", age = 30)
Print (user)
```

- Accessing Items

```
Print ( simple-dic ["name"]
Print ( simple_dic.get("role"))
Print ( simple_dic.get("salary", 0))
```

- Adding & updating key-value pairs.

```
simple_dict ["experience"] = 2
Simple_dict ["age"] = 25
Print (Simple_dict)
```

- Removing key-value Pairs.

```
del simple_dict ["role"]
Print (simple_dict)

removed = simple_dict.pop("experience")
Print (removed).
```

- Iterating Over dictionary

```
data = {"name": "Yasir", "field": "AI",
        "level": "Advanced"}
# Keys
for key in data:
    Print(key)

Value
for value in data.values():
    Print (value)

Key-value Pairs
for key, value in data.items()
    Print (f"{key} => {value}")
```

- This is all about dictionary
- Dictionaries are very useful in Production level work.
- Dictionaries are important & for Placement the concept is very necessary.

Sets In Python

- What are sets?
- Sts are unordered collections of unique, immutable elements.
They dont allow duplicates.
They are mutable
They don't support Indexing / Slicing.

- Creating Sets.

```
S1 = { }  - Wrong way
Print (type(s1))  # dict.
```

correct empty set.

```
S2 = set( )   set( )
Print (type(s2))  # set.
```

* Creating a set with duplicates.

```
S3 = { 1, 2, 3, 3, 4, 4, 5}
Print (S3) - { 1, 2, 3, 4, 5}
```

→ Sets inside sets are not allowed.

- Type conversion

```
list_to_set = set([1, 2, 2, 3])
Print (List_to_set)  # { 1, 2, 3}
```

- Modifying Sets.

```
S = { 1, 2, 3}
s.add(4)
Print(S)
```

```
s.update([5, 6])
Print(s)
s.remove(2)
Print(s)
s.discard(10)
Print(s)
s.pop()
Print(s)
s.clear()
Print(s)
```

- Membership Operators.

```
member = {"Yasir", "abdullah", "Baig"}
Print("Yasir" in member) # True
Print("Ali" not in member) # True
```

- Looping Through Sets.

```
Loop_set = {1, 2, 3, 4, 5}
for item in loop_set:
    Print(item)
```

- Set Methods

```
A = {1, 2, 3}
B = {3, 4, 5}

Print(A.union(B))
Print(A | B)

Print(A.intersection(B))
Print(A & B)
```

Print (A. difference (B))
Print (A - B)

Print (A. symmetric_difference (B))
Print (A ^ B)

- Set Relationships.

set 1 = {1, 2}
Set 2 = {1, 2, 3}
Print (set1. issubset (set2)) # True
Print (set2. issuperset (set1)) # True
Print (set1 isdisjoint ({4, 5})) # True.

- Built-In- Functions

nums = {1, 2, 3, 4, 5}
Print (len (nums))
Print (max (nums))
Print (min (nums))
Print (sum (nums))
Print (sorted (nums))

- Sets are ideal for membership testing, uniqueness filtering, & mathematical set operations.
- Use Frozenset if you need an immutable version of a set.
- Set are unordered; their printed ordered may vary.

Functions In Python

- What are Functions, why use Functions?

Function help:

- organize code into blocks
- Avoid repetition
- Make code modular, reusable, readable
- Perform specific task independently.

Syntax of a Function.

```
def func(Parameters):
    """ docstring """
    #Func Body
    return value
function - call.
```

-
```
def greet():
    Print ("hello, how are you")
greet()
```

Return vs Prints.

- Print - display output to console
- return - send data back to caller for Further use.

Parameters vs Arguments.

- Parameters: Variable Inside function
- Arguments: actual value pass when calling func

Types of Parameters

- Required Paramters.

```
def required_example(name):
    Print(name)
required_example("Yasir")
```

2- Default Parameters

```
def default_example(city = "Lahore")
    Print(city)
default_example()
default_example("Karachi") ** overrides Lahore.
```

3- Named Parameters

```
def student_info(name, age):
    Print(name)
    Print(age)
student_info(age = 24, name = "Yasir")
```

4- Variable_length Parameters

```
def many_args(*args):
    Print(args)
many_args(1, 2, 3, "Python")

def many_kwargs(**kwargs):
    Print(kwargs)
many_kwargs(name = Yasir, age = 24)
```

- Scope In Functions
- local scope . variables declared inside func
- Global scope. variables declared outside all func

File Handling In Python

- What is File handling?

File handling in python refers to the ability to read, write, create, delete, & manipulate files stored on your computer (like .txt, .csv, .json, files)

- Why we use File handling?

→ To store data permanently.
→ To read logs, input, config files
→ To work with datasets.

File Modes.

r	-	read
w	-	write
a	-	append
x	-	create
rt	-	read & write
b	-	binary mode

- Exception handling (try, except)

```
try:
    with open("nofile.txt", "r") as f:
        content = f.read()
except FilenotFoundError:
    Print ("File not Found")
except PermissionError:
    Print ("Permission denied")
else:
    Print ("File read successfully")
finally:
    Print ("operation completed.")
```

Read Methods.

- read()
- readline()
- readlines()

Important Tips.

- Always use 'with open(...)' - auto-close file safely.
- use exception handling to pervent crash.
- File Paths. use relative (./folder/file.txt)
- Always close file if not using 'with'

"The End"

"Thanks For Reading This Resource"

LinkedIn : mirza-yasir-abdullah-Baig.