# TOPICS:

1. Installation & Environment Setup
2. Data Types

#### 1. INSTALLATION & ENVIRONMENT SETUP

# to check if it is installed or not  
! python --version   
  
# to create virtual env in windows  
# ! python -m venv venv   
  
# to activate virtual env  
# ! venv\Scripts\activate   
  
# create a requirements.txt  
# to install all packages in requirements.txt -> pip install -r requirements.txt

Python 3.12.3

#### 2. Data Types

import pandas as pd  
  
# create a DataFrame  
df = pd.DataFrame({  
 "Category":["Numeric", "Numeric", "Numeric", "Text", "Boolean", "Sequence", "Sequence", "Sequence", "Set Types", "Set Types", "Mapping", "Binary", "Binary", "Binary", "None Type"],  
 "Data Type": ["int", "float", "complex", "str", "bool", "list", "tuple", "range", "set", "frozenset", "dict", "bytes", "bytearray", "memoryview", "NoneType"],  
 "Example": ["x = 10", "y = 10.5", "z = 3 + 4j",   
 's = "Hello"', "flag = True", "[1, 2, 3]",   
 "(1, 2, 3)", "range(5)", "{1, 2, 3}",   
 "frozenset([1,2,3])", '{"key": "value"}',   
 "b'hello'", "bytearray(5)", "memoryview(bytes(5))", "x = None"]  
})  
  
df

Category

Data Type

Example

0

Numeric

int

x = 10

1

Numeric

float

y = 10.5

2

Numeric

complex

z = 3 + 4j

3

Text

str

s = “Hello”

4

Boolean

bool

flag = True

5

Sequence

list

[1, 2, 3]

6

Sequence

tuple

(1, 2, 3)

7

Sequence

range

range(5)

8

Set Types

set

{1, 2, 3}

9

Set Types

frozenset

frozenset([1,2,3])

10

Mapping

dict

{“key”: “value”}

11

Binary

bytes

b’hello’

12

Binary

bytearray

bytearray(5)

13

Binary

memoryview

memoryview(bytes(5))

14

None Type

NoneType

x = None

## NUMERIC DATA TYPES  
# int methods  
x = 10  
x.bit\_length() # 4 # number of bits required to represent the number in binary  
x.to\_bytes(2, byteorder='big') # b'\x00\n' # coverting to bytes  
x.to\_bytes(2, byteorder='little') # b'\n\x00'   
x.from\_bytes(b'\x00\n', byteorder='big') # 10 # coverting from bytes  
x.from\_bytes(b'\n\x00', byteorder='little') # 10   
x.bit\_count() # 2 # number of 1 bits in the binary representation of the number  
  
# float methods  
y = 10.5  
y.as\_integer\_ratio() # (21, 2) # return a tuple of two integers whose ratio is exactly equal to the original float  
y.is\_integer() # False # check if the float is an integer  
y.hex() # '0x1.5000000000000p+3' # hexadecimal representation of the float  
fromhex = float.fromhex('0x1.5000000000000p+3') # 10.5 # convert from hexadecimal representation  
from\_float = float.fromhex(y.hex()) # 10.5  
  
# complex methods  
z = 3 + 4j  
z.real # 3.0 # real part of the complex number  
z.imag # 4.0 # imaginary part of the complex number  
z.conjugate() # (3-4j) # conjugate of the complex number  
z.conjugate().imag # -4.0  
  
# Built-in functions for numeric data types  
abs(-10) # 10 # absolute value  
divmod(10, 3) # (3, 1) # quotient and remainder  
pow(2, 3) # 8 # power  
round(10.5) # 10 # round off  
round(10.5, 0) # 10.0  
round(10.5, 1) # 10.5  
sum([1, 2, 3]) # 6 # sum of elements  
max([1, 2, 3]) # 3 # maximum element  
min([1, 2, 3]) # 1 # minimum element  
  
# math module  
import math  
math.ceil(10.5) # 11 # round up  
math.floor(10.5) # 10 # round down  
math.trunc(10.5) # 10 # truncate  
math.factorial(5) # 120 # factorial  
math.gcd(10, 5) # 5 # greatest common divisor  
math.lcm(10, 5) # 10 # least common multiple  
math.isqrt(10) # 3 # integer square root  
math.sqrt(10) # 3.1622776601683795 # square root  
math.exp(1) # 2.718281828459045 # exponential  
math.log(10) # 2.302585092994046 # natural logarithm  
math.log10(10) # 1.0 # base 10 logarithm  
math.log2(10) # 3.321928094887362 # base 2 logarithm  
math.isfinite(float('inf')) # False # check if the number is finite  
math.isinf(float('inf')) # True # check if the number is infinite  
math.isnan(float('nan')) # True # check if the number is not a number  
math.isclose(10.5, 10.5000000001) # True # check if two numbers are close  
math.isclose(10.5, 10.5000001) # False  
math.isclose(10.5, 10.5000001, rel\_tol=1e-6) # True  
math.isclose(10.5, 10.5000001, rel\_tol=1e-7) # False

# Challenge 1:  
# Write a Python script that:  
  
# Takes a number as input.  
# Determines if it’s an integer or float.  
# If it's an integer, calculate its square root and factorial.  
# If it's a float, convert it to hexadecimal and round it to 2 decimal places.  
# If it's a complex number, return its conjugate and real part.  
  
# Solution:  
num = input("Enter a number: ")  
print(num)  
if type(num) == int:  
 print(f"Square root: {math.sqrt(num)}")  
 print(f"Factorial: {math.factorial(num)}")  
elif type(num) == float:  
 print(f"Hexadecimal: {float.hex(num)}")  
 hex\_num = float.hex(num)  
 print(f"Rounded: {round(float.fromhex(hex\_num), 2)}")  
elif type(num) == complex:  
 print(f"Conjugate: {num.conjugate()}")  
 print(f"Real part: {num.real}")  
else:  
 print(f"Invalid number:{type(num)}")

10.22  
Invalid number:<class 'str'>