Python

Sheryians Coding School

To truly understand this book by heart and make the most out of every concept , watch the full video on our official Youtube channel: Sheryians AI

This book is designed to work hand-in-hand with the visual explanations and examples shown in the video.

made with ❤️ in the Sheryians Coding School

Installation 01

Downloading Python

- For downloading python go on any browser go to python.org and download python for your operating system.
- Now install it and you will get the python Virtual machine which will convert the code to byte code.

Downloading IDE

- IDE stands for Integrated Development Environment.
- You can write and run your code using python virtual machine.
- But the Conventional way is to use a IDE there are many IDE's like VS code, Pycharm, Jupyter.
- But we will use VS code.

Setting up and creating first program

- For setting up we have to go to VS code and in extensions we have to install python and code runner.
- Lets move forward and create our first program and that will not be hello world.

Comments and Variables 02

Comments in python

- Comments are something that are ignored by the python interpreter.
- We have to use # for writing a comment in python.
- Multiline comments are not available in python but we can achieve it by using Doc String “”” Multiline ”””

Variables in python

- In python Variables are used as a storage to store things in python (we will see later what we have to store).
- You can write anything as a variable name.
  Eg. Name = “Akarsh”
  Age = 12

Don’t use these

- You can not use numbers at variable start.
- You can not use spaces in variables.
- You should not use special characters in variables.

Naming Conventions

- You can write variables in python using 3 ways.
  - Camel case - sheryiansSchool
  - Pascal case - SheryiansSchool
  - Snake case - sheryians_school

Data Types in python

What are Data Types

- Data types are the things we store in Variables and it defines what data type variables are.
- Python has built-in data types for different kinds of data.

Numbers

Integer - All the numbers excluding decimal places and fraction.

Float - All the decimal numbers and fraction values are Float.

Complex - Numbers with real and imaginary parts are complex.

Strings

Strings - This is used to store anything in python, literally anything that are available on your keyboard.
You have to use quotes to store anything and it will be considered as string. You can use double Quotes (“”) or single quotes (‘’) to store both works same.

Boolean

Boolean - Theres nothing much to say this is the data type which will and always give the result of True and False.

Strings and type conversion 04

How Strings work

- You know what strings are but you must also know string take more space than other data types like int, float etc.
- This happens because String stores every character with their own Unicode.
- Unicode is a universal character encoding standard that assigns a unique number (code point) to every character, regardless of language.
- Like “A” unicode is 65 and “😀” this emoji unicode is 128522, you can check them by using ord() function in python and convert them back using chr() function.

String Indexing

- You must have thought there are so many characters in a string but can you access everyone.
- Yes thats possible using indexing. Indexing starts from 0 and goes till the number of characters you have.
  - eg - a = “Hello” print(a[0]) ==> output - “H”
- There is negative indexing as well and it starts from -1, but the starting position is from the back of the string.
  - eg - a = “Hello” print(a[-1]) ==> output - “o”

String Slicing

- You know how to access characters in string. But there are slicing option as well.
- Slicing means cutting out a slice from string and this is also done using index values.
  - eg - a = “hello” a[1:4:1] ==> output “ell”
- So here we have start , stop and steps position and keep a note if we use stop at 4 it will slice till 3 only.

Type conversion

- For understanding type conversion you have to look at these 4 things.

int() float() str() bool()

- There are more functions like this but these are 4 main function, looking at these functions you can guess these are used to convert one data type to another.
  - eg a = 12
    a = str(a)
    print(a) ==> “12” (a will be converted to string)

Type conversion types

- There are 2 types of conversion Implicit and Explicit.

Implicit

- In this python automatically converts data from one data type to another.
- You have already seen the example before.
  - eg - a = 12
    print(a/2)
    output - 6.0
- Clearly we had the data type as int but after dividing python automatically converted the data type to float.

Explicit

- In this we as a user use in build functions to convert one data type to another.
- You have seen the example at previous page.

int()	-	Integer
float()	-	Float
complex()	-	Complex
str()	-	String
list()	-	List
tuple()	-	Tuple
set()	-	Set
dict()	-	Dictionary
bool()	-	Boolean

- There are some explicit conversions that you might not understand but further we will understand.

Type conversion concepts

- Some important concepts of type conversions are you cannot convert a character to a int() that basic watch the video for more set of information.
- bool() converter turns everything to True and False but which thing will be converted to true and which false. Lets see.
- There are truthy values and Falsy values, and there are only 7 falsy values that means only 7 things will be converted to false rest True.

  0
  0.0
  False
  “”
  []
  {}
  ()

- All these values are falsy remaining will be converted to True.

Input and output 05

Output

- You probably know till now, how to provide the output of the code you have written and that is with print() function.
- There is no other functions to provide the result on the terminal we just have to use print() function.
- Now when providing the output we can use variables in print statement using a formatted string as shown in the below example.

```
name = input("Enter your name: ")
print(f"Hello, {name}! Welcome to Python programming.")
```

Input

- But the main question is how to ask user for some information.
- For example there is a user and you want to ask the age of that user, how can you do so, it's easy using input().
- Now the default data type of input is always string reason is simple you can store anything in string.
- You have to manually convert the data type of input statements.

Questions

- **Important** watch full video for clear understanding.
- Accept numbers from a user.
- Accept age from the user and print it.

Operators 05

What are operators

- Operators are symbols that perform operations on variables and values. Python has several types of operators for different tasks like arithmetic, comparison, logical operations, and more.
- Lets see every operators one by one.

Arithmetic operators

- Arithmetic operators perform mathematical operations like addition, subtraction, multiplication, division, etc.
- There are 7 types of arithmetic operator.
  - addition - +
  - subtraction - -
  - multiplication - *
  - division - /
  - Floor division - //
  - modulus - %
  - Exponentiation - **
- Eg - a = 12
  b = 8
  print(a+b)
  Output - 20
- See the video for proper explanation.

Assignment operator

- Assignment operators are used to assign values to variables. Python also provides compound assignment operators that perform operations like addition, subtraction, multiplication, etc.
- A basic assignment operator is simple =.

Compound assignment operator

- Compound assignment operator combines arithmetic operations with assignment.
- But first you have to understand how things work when we reassign variables in python and also reassigning variables with addition, subtraction etc.
- To understand watch the video carefully.
- Using compound assignment operators the reassigning works better.
  - += - Add and assign
  - -= - Subtract and assign
  - *= - Multiply and assign
  - /= - Divide and assign
  - //= - Floor divide and assign
  - %= - modulus and assign
  - **= - Exponentiation and assign

Comparison operator

- Comparison operators, also called relational operators, are used to compare two values.
- Comparison operators will always provide Boolean result that is True and False.
- comparison operators are as follows
  - == - Equal to
  - != - Not Equal to
  - > - Greater than
  - < - Less than
  - >= - Greater than or equal to
  - <= - Less than or equal to

- Comparison operators will work with numbers but you can use them with strings as well.
- Strings will be comparing the Ascii values of string.

Logical operators

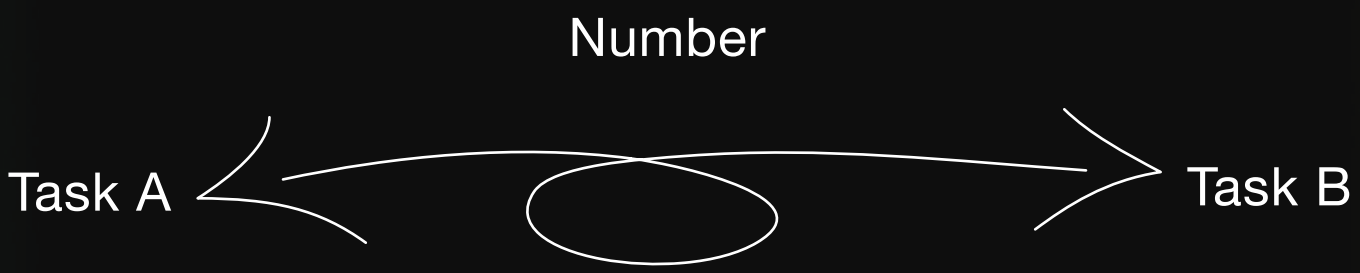
- Logical operators in Python are used to combine multiple conditions and return a Boolean result (True or False).
- There are 3 types of logical operator
  - and - Return True if both condition are True
  - or - Return True if at least one condition is True.
  - not - Reverse the boolean value.
- **important** watch the full video for better understanding.

Trivial Questions

- Answer True and False
- Print(126 > 130)
- print((456 == 456) != (235 == 236))
- print(12 < 10 or 45 == 56 or 69 > 70 or 15 != 13)
- print(True and bool(0))

Conditional statements

- Conditional statements in Python allow decision-making by executing different blocks of code based on conditions.
- Decision making can be understood with an example
  eg - you will receive a number from the user if the number is greater than 10 you will do task A and lower than 10 you will do task B.



- Here the situation is simple Number will decide which task will be executed.
- That means now we will control the flow of our program based on some conditions thats why these statements are also known as control flow statement.

Types of conditional statements

- Generally there are 3 types of variation in conditional statements.
- For syntax you have to watch the video for better understanding.

  - if - Executes if the condition is True
  - if-else - Executes if True, another if False
  - if-elif-else - Checks multiple condition in sequence.

```
if condition:
    # Code to execute if condition is True
```

```
if condition:
    # Code if condition is True
else:
    # Code if condition is False
```

```
if condition1:
    # Code if condition1 is True
elif condition2:
    # Code if condition2 is True
else:
    # Code if all conditions are False
```

Some Questions on Conditional

- The Great thing is you can use logical operators as well.

Q1. Accept two numbers and print the greatest between them.

Q2. Accept the gender from the user as char and print the respective greeting message

Ex - Good Morning Sir (on the basis of gender)

Q3. Accept an integer and check whether it is an even number or odd.

Q4. Accept name and age from the user. Check if the user is a valid voter or not.

Ex- “hello shery you are a valid voter”

Q5. Accept a year and check if it a leap year or not (google to find out what is a leap year)

If- elif ladder

- You cna also create if elif ladder using multiple conditions of elif.
- For understanding solve this question
- take the input of temperature in celsius.
  - Below 0°C → "Freezing Cold ❄️"
  - 0°C to 10°C → "Very Cold 🥶"
  - 10°C to 20°C → "Cold 🧥"
  - 20°C to 30°C → "Pleasant ☁️"
  - 30°C to 40°C → "Hot 🔥"
  - Above 40°C → "Very Hot 🏜️"

Loops in python

- Loops in Python allow us to execute a block of code multiple times without rewriting it.
- Ok lets do one thing go to your VS code and print “hello world” 100 times.
- Manually printing will take 100 code lines to print it. but using loops we need only 2 lines to print 100 times, thats the power of loops.

Types of loops

- There are 2 types of loops in python. For and While loop.

- For understanding both types of loop we well see a great example- you have a bucket filled with water and an empty bucket with a mug.

- In scenario 1 you have to transfer 4 mugs of water from 1 bucket to another.
- In scenario 2 you have to transfer all the water from 1st bucket to another via mug.

Intuition of loops

- In first scenario you know the number of mugs to transfer from one bucket to another.
  - Here you know how many numbers of iteration you have to go through, like you have to transfer only 4 mugs.
  - So when you know the number of iterations you will use a FOR loop.

- In the second scenario, you don't know how many mugs you need to transfer, but you do know the condition that determines when to stop.
  - So when you don’t know how many iteration you have to use but you know a condition that determines when to stop you will use WHILE loop.

Range function

- Before understanding for loops you should know how range function works.
- The range() function is used to generate a sequence of numbers, which is commonly used in loops.
- Syntax of range function is simple range(start, stop, steps).
- you have 3 points from where you want to start, till where you want to stop and how many steps you want.
- If you don't mention start point the default value will be 0 .
  if you don't mention the steps the default steps will be 1.
  you have to mention the stop point otherwise the range function will not work.

Loops for numbers

- For using loops with numbers you need the range function.
- Best way to understand is going with an example
  - You have to print numbers from 1 - 5.
    we will solve this question using for loop and range function.

```
for i in range(1, 6):
    print(i)
```

- So this is how you use a for loop. watch the full video for clear understanding of syntax.

Loops for strings

- Loops for strings work slightly differently. You can iterate through a string in two ways:
  a. Using index values.
  b. Iterating directly over the string.

- Iterating using the index values, Now we know that index values are numbers and for numbers we again have to use range function.

```
a = "Nature"

for i in range(len(a)):
    print(a[i])
```

- Here you iterated over the string using the index values for better understanding watch the video.

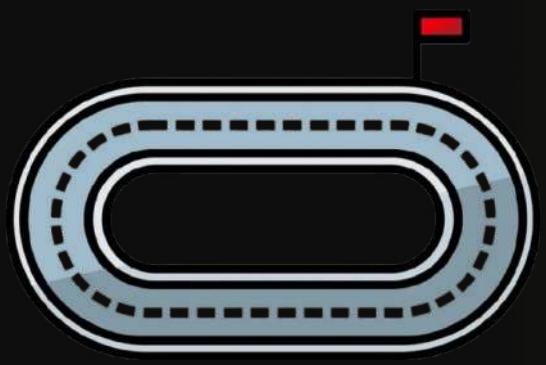
- Second way is simpler we can directly iterate but using this method will give you the direct access to the characters instead of index values.

```
a = "Nature"

for char in a:
    print(char)
```

Break continue else

- Things written above are very important for loops.
- Lets say you have this race track and you have to complete 20 laps but when you were completing the 16th laps and it started raining now you cannot complete the rounds.



- The above example simulate the example of your break statement.
- The break statement in Python is used to exit a loop prematurely when a certain condition is met. Once break is encountered, the loop stops immediately, and control moves to the next statement after the loop.
- The Continue statement skips one of the iteration and rest of the iterations will run for understanding lets say you will not run the 16th lap but all other lap will be there.
- You have seen the else statement used with if, but it can also be used with loops. When else is used with a loop, it only executes if the loop completes without encountering a break statement. If break is executed, the else block will not run.

For Loop questions

- Accept an integer and Print hello world n times.
- Print natural number up to n.
- Reverse for loop. Print n to 1.
- Take a number as input and print its table.
- Sum up to n terms.
- Factorial of a number.
- Print the sum of all even & odd numbers in a range separately.
- Print all the factors of a number.
- Accept a number and check if it a perfect number or not.
  A number whose sum of factors is equal to the number itself

  Ex - 6 = 1, 2, 3 = 6
- Check wether the number is prime or not.

- Reverse a string without using in build functions.
- Check string is Pallindrome or not.
- Count all letters, digits, and special symbols from a given string
  Given: str1 = "P@#yn26at^&i5ve"
  Expected Outcome:
  Total counts of chars, digits, and symbols
  Chars = 8
  Digits = 3
  Symbol = 4

While loop

- You have previously taken the information of loops and you also know conditional statements it is going to be easy for you to understand this now.
- The while loop repeats a block of code as long as a condition is True. It is useful when the number of iterations is unknown before execution.

```
while condition:
    # Code to execute
```

- So there is not much that you have to understand about while loop it also have break, continue and else.
- Now you just have to find out which loop will be used on what questions.

While loop questions

- Separate each digit of a number and print it on the new line.
- Accept a number and print its reverse.
- Accept a number and check if it is a pallindromic number (If number and its reverse are equal)
- Create a random number guessing game with python.

What are functions

- Functions in Python group reusable code into a block that can be executed by calling the function name. This helps avoid repetition and makes programs modular and readable.

- There are many in-build functions in python like print(), input() len() etc.
- But you can create your own function and they are called as user defined functions. To make your own function you have to use def keyword and then name the function. After this you have to call the function using name() and paranthesis.

```
def greet():
    print("Hello, welcome to Python!")

greet()  # Calling the function
```

Functions parameters and arguments

- First thing I want to talk about is parameters, parameters are variables listed inside the function definition.
  For making the function we have to accept inside the parenthesis of the function.

```
def greet(name):  # 'name' is a parameter
    print(f"Hello, {name}!")
```

- Arguments are the Values passed to a function when it is called.
- For example you can say you have created the parameters that are working like variables then we can pass the values to our variables using arguments.

```
def greet(name):  # 'name' is a parameter
    print(f"Hello, {name}!")

greet("Alice")  # "Alice" is an argument
```

- As you can see name is the parameter and Alice is the argument that we passed to name. And you can pass N number of parameters and arguments but they must be same like if you have taken 3 parameters you have to provide 3 arguments otherwise there will be error.
- And another thing is first parameter, will always capture first argument and so on. These arguments are called positional argument.

Types of Arguments

- Now there are 3 types of argument that we can pass to parameters. positional argument, default argument, keyword argument, For understanding these we will first see examples.

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

print(add(3, 5))  # 3 is assigned to 'a', 5 to 'b'
```

```python
def introduce(name, age):
    print(f"I am {name} and I am {age} years old.")

introduce(age=25, name="John")
```

```python
def greet(name="Guest"):
    print(f"Hello, {name}!")

greet()       # Uses default value "Guest"
greet("Bob")  # Uses "Bob"
```

- First example shows how positional arguments work.
- Second example shows hoe default argument works here if you don’t pass any value still the function will run.
- Last example shows how keyword argument works using this you can pass values in any order.

In-build data structures

- Data structures are used to store, organize, and manipulate data efficiently. Python provides several built-in data structures.
- And for storing multiple values we will again use variables.
- Now in python we have 4 types of in-build data structure List, Tuple, Dictionary, Set.

Custom data structures

- Now there are some custom data structures as well like Stack, Queue, Linked List, Graph etc.
- And around these data structures there are some algorithms like searching algorithms, sorting algorithms.
- And this is why the study is called data structures and algorithm.

- Lets be clear this python notes are not for the DSA this will cover all the in-build data structures.

List Powers

- Before starting we need to understand some of the terminology.
  - Mutable - Mutability refers to whether an object's value can be changed after creation. And List allows this.
  - Duplicates - we know data structures are used to store multiple values so duplicates means same value occuring multiple time. List allows this.
  - Ordered - List maintains ordered data structure maintains the sequence of elements as they were inserted. This means you can access elements using their position (index).
  - Heterogenous - List have heterogenous nature that means we can have multiple data types inside the list.

List Basics

- First we have to know what is the syntax of list and how, to create a list we have to use square brackets ([]).

```
fruits = ["apple", "banana", "cherry"]
```

```
numbers = [10, 20, 30, 40]
```

- Now list has Indexing and slicing and it is same as string if you forgot how string indexing and slicing works watch the video or revise the notes.
- The changes we saw in string and list is about mutability, we can't change the values of string. but we can of list.

```
# Define a list
numbers = [10, 20, 30]

# Modify the value at index 1 (2nd element)
numbers[1] = 99

# Print the updated list
print(numbers)  # Output: [10, 99, 30]
```

List Traversing and methods

- Now list traversing is also similar to string traversing it can be looped using the index values and directly.
- Now list has some methods that are used to do many, and don't worry if you are not sure what are methods, for now just think they are like function, further we will see it clearly.

- Now see some of the examples of the methods you will get it what they are used for.

```
numbers = [5, 2, 9, 1, 5, 6]  # Initial list

numbers.append(10)  # Adds 10 to the end
numbers.insert(2, 15)  # Inserts 15 at index 2
numbers.extend([20, 25, 30])  # Adds multiple elements at the end
numbers.remove(5)  # Removes the first occurrence of 5
popped_item = numbers.pop(3)  # Removes and stores the element at index 3
index = numbers.index(6)  # Finds the index of 6
count_5 = numbers.count(5)  # Counts occurrences of 5
numbers.sort()  # Sorts the list in ascending order
numbers.reverse()  # Reverses the list order
new_numbers = numbers.copy()  # Creates a copy of the list
numbers.clear()  # Removes all elements from the list
```

- So these are some methods that are used in list .

Some Questions on List

- Print positive and negative elements of an List.
- Mean of List elements.
- Find the greatest element and print its index too.
- Find the second greatest element.
- Check if List is sorted or not.

Tuple Powers

- Before starting we need to understand some of the terminology.
  - Immutable - Tuples are not mutable you cannot change the values of tuple
  - Duplicates - You can have duplicate values in tuple there are no restriction.
  - Ordered - Set are ordered and you can access them through index values.
  - Heterogenous - Set also have heterogenous nature and can have different types of data structure in tuple.

Tuple Traversing and methods

- Tuples are traversed in the same manner as List are traversed.
- But remember tuples are like strings you can't change anything once it's made we can't change them.
- Well the use case is not much in question solving but still you have to understand it.
- Methods of tuple are:

```
t = (5, 2, 9, 1, 5, 6)  # Initial tuple

index = t.index(9)  # Finds the index of first occurrence of 9
count_5 = t.count(5)  # Counts occurrences of 5
```

- Yes there are only 2 methods of tuple one for finding the index and other of counting the occurrences of an element.

Set Powers

- Before starting we need to understand some of the terminology.
  - mutable - Sets are mutable you can change the values of set.
  - Duplicates - You cannot have any duplicate values in set
  - that means every element will be unique.
  - Unordered - Sets are unordered and you cannot access them through index values.
  - Heterogenous - Set is semi-heterogenous it can store some data types like string, numbers, tuples but not everything

How Set stores value in python

- Each value in a set is hashed using a hash function (hash() in Python).
- The hash is used as an index to store the element in memory.
- Since hashing does not maintain order, sets are unordered.
- Only immutable (hashable) objects can be stored in a set (e.g., numbers, strings, tuples). Mutable objects like lists and dictionaries are not allowed.
- See the video for more clear understanding.

Set Traversing

- A set cannot be traversed using the index values cause it is unordered and has no index.
- So many times it will give random values. you can watch the video for complete understanding.

Set methods

- Now set methods are very powerful cause you don't have any indexing you cannot change the values but set is mutable so we use methods for this.
- For adding and removing the elements you can use methods as follows.

```
s = {1, 2, 3}

s.add(4)  # Adds an element to the set
s.remove(2)  # Removes 2 (Raises an error if not found)
s.discard(5)  # Removes 5 (No error if not found)
popped_element = s.pop()  # Removes a random element
s.clear()  # Removes all elements
```

- Now these are some of the basic methods but sets also have some special methods for performing some special operations between 2 sets.

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

union_set = A.union(B)  # {1, 2, 3, 4, 5}
intersection_set = A.intersection(B)  # {3}
difference_set = A.difference(B)  # {1, 2}
symmetric_diff = A.symmetric_difference(B)  # {1, 2, 4, 5}
```

- So these are some other operations of sets that can be performed between 2 sets. And we also have shortcuts for them.

```python
print(A | B)  # Union
print(A & B)  # Intersection
print(A - B)  # Difference
print(A ^ B)  # Symmetric Difference
```

- Ok so we have seen these operations and while watching the video you have seen a ven diagram approach as well there are more methods you are open to try them and see the working.
- But at the end set is not used that much in python lets continue.

Dictionary Powers

- Before starting we need to understand some of the terminology.
  - mutable - Dictionaries are mutable, meaning you can change, add, or remove key-value pairs after creation.
  - Duplicates - Keys must be unique, but you can have duplicates in values.
  - Order - Dictionary follows insertion order.
  - Heterogeneous – A dictionary can store different types of keys and values, like integers, strings, lists, or even another dictionary.

Dictionary syntax and working

- Now we know we have to use key and value pairs to store values in dictionary.
- And the keys in dictionary acts like index values that we use in List.

```
student = {"name": "John", "age": 20}
print(student["name"])  # Output: John
```

- Again telling we can perform CRUD(create, read, update, delete) operations on values but not all on keys cause the keys cannot be changed after creation.

Dictionary traversing

- We can traverse both keys and values in dictionary, but default loop is set on keys and you can access the values because of keys.
- So technically you can traverse on both keys and values at the same time.

```
numbers = {1: 10, 2: 20, 3: 30, 4: 40}

for i in numbers:
    print(i, ":", numbers[i])
```

- Do see the video explanation for better understanding.

Dictionary methods

- There are not many dictionary methods lets see the working of some.
- as you all know we can use help(dict) for getting the information of all the methods available.

Dictionary Questions

- Write a Python script to merge two Python dictionaries.
- Write a Python program to sum all the values in a dictionary.
- Count the frequency of each elements
- Write a Python program to combine two dictionary by adding values for common keys.

Errors

- Errors occur due to mistakes in the code that prevent it from running. These can be syntax errors or logical errors.
  - Syntax error

```
print("Hello World"  # Missing closing parenthesis
```

  - Now this above code will give the error of syntax.
  - Indentation Errors

```
def func():
print("Hello")  # No indentation
```

  - You already know what is indentation and if you don't follow it you will get the error.
  - There is one more tab error when you mix tabs and spaces.
  - These errors cannot be handled. but what can be handled are exceptions.

Exceptions

- Exceptions are unexpected events or errors that occurs during the execution of a program, which disrupts the normal flow of the program.

Exceptions

- Exceptions are unexpected events or errors that occurs during the execution of a program, which disrupts the normal flow of the program.

```
print("Start")
print(10 / 0)   # ❌ Raises ZeroDivisionError
print("End")    # ❌ This line will never run
```

- Now this is a ZeroDIvisionError and can be counted as Exception and because of this exception the next line cannot be executed.
- Like this there are many other exceptions just leave the three errors we saw at start otherwise others are exceptions.
- And the good part is we can handel them lets see how.

Exception Handling

Keyword	Purpose
try	Wrap the block of code that might cause an exception.
except	Handle the exception if it occurs
else	Run code only if no exception occurs
finally	Run code no matter what, whether there's an exception or not
raise	Manually throw an exception

- So these are the keywords that we use and all these keywords has their separate purpose as mentioned.
- To see the code part see the video.

What are files

- You all know what are files any name with an extension is file.
- Now that extension can be .py , .txt , .mp3 etc. and when we want to handle these files we will use file handling.

File handling

- File handling means Creating, Reading, Updating, Deleting(CRUD) operations that we can perform in files.
- Now lets see how to perform these operations in python.
- We have to use open() function to open a file in python.

  - Now there are multiple modes to open the file.

Mode	Description
‘r’	Read (default) – file must exist.
‘w’	Write – creates file or overwrites.
‘a’	Append – adds to end of file.
‘x’	Create – creates a new file, fails if it exists

What are files

- You all know what are files any name with an extension is file.
- Now that extension can be .py , .txt , .mp3 etc. and when we want to handle these files we will use file handling.

File handling

- File handling means Creating, Reading, Updating, Deleting(CRUD) operations that we can perform in files.
- Now lets see how to perform these operations in python.
- We have to use open() function to open a file in python.

    - Now there are multiple modes to open the file.

Mode	Description
‘r’	Read (default) – file must exist.
‘w’	Write – creates file or overwrites.
‘a’	Append – adds to end of file.
‘x’	Create – creates a new file, fails if it exists

Syntax

```
file = open("myfile.txt", "r")
print(file.read())          # Read entire file
# print(file.readline())    # Read one line
# print(file.readlines()) # Read all lines into a list
file.close()
```

- This is the basic syntax through which we can open a text file and the ‘r’ there represents read mode and there are multiple modes like this as mentioned before. see the video for other modes as well.
- Now after working you have to close the file manually but for this we have with keyword.

```
with open("data.txt", "r") as f:
    content = f.read()
    print(content)
```

- Now Lets create a basic file handling project.

What is OOPS

- For understanding oops first lets see what we were doing in python for creating a program of addition we first use imperative approach.

```
a = 12
b = 12
print(a + b)
```

- This approach is simple just use 2 variables and add them one problem with this is you have to make 2 other variables for adding 2 other numbers.
- Next approach is using functions to add 2 numbers this is functional approach.

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

print(addition(12, 12))
print(addition(45, 45))
```

- Here the good thing is we can add multiple numbers without using multiple variables.

What is OOPS

- And our next approach is object oriented programming approach.

```
class Addition:
    def __init__(self, a, b):
        print(a + b)

obj = Addition(12, 12)
```

- OOPS (Object-Oriented Programming System) is a programming paradigm based on the concept of "objects", which can contain data (attributes) and code (methods).
- I know it is tough to understand right now but it will be easy after learning there are many concepts that we have to learn like classes, objects , Encapsulation, inheritance, Polymorphism, etc. So lets start.

Classes

- A class is like a blueprint or template for creating objects.
- Think of a class like the blueprint of a house. It defines what the house should have (rooms, windows, etc.) but doesn't build the house. An object is the actual house built using that blueprint.

Syntax of class

- A class is also created with a basic keyword class and a name in front of it.

```
class Car:
    brand = "Toyota"
```

- Creating a class is super simple now lets see what is inside class. There are 2 types of things inside class Attributes and Methods.
  - Attributes - Variables defined inside the class are Attribute.
  - Methods - Functions defined inside a class are Methods.

```
class Animal:
    species = "Dog"  # Attribute

    def make_sound(self):  # Method
        print("Bark!")
```

Accessing attributes and methods

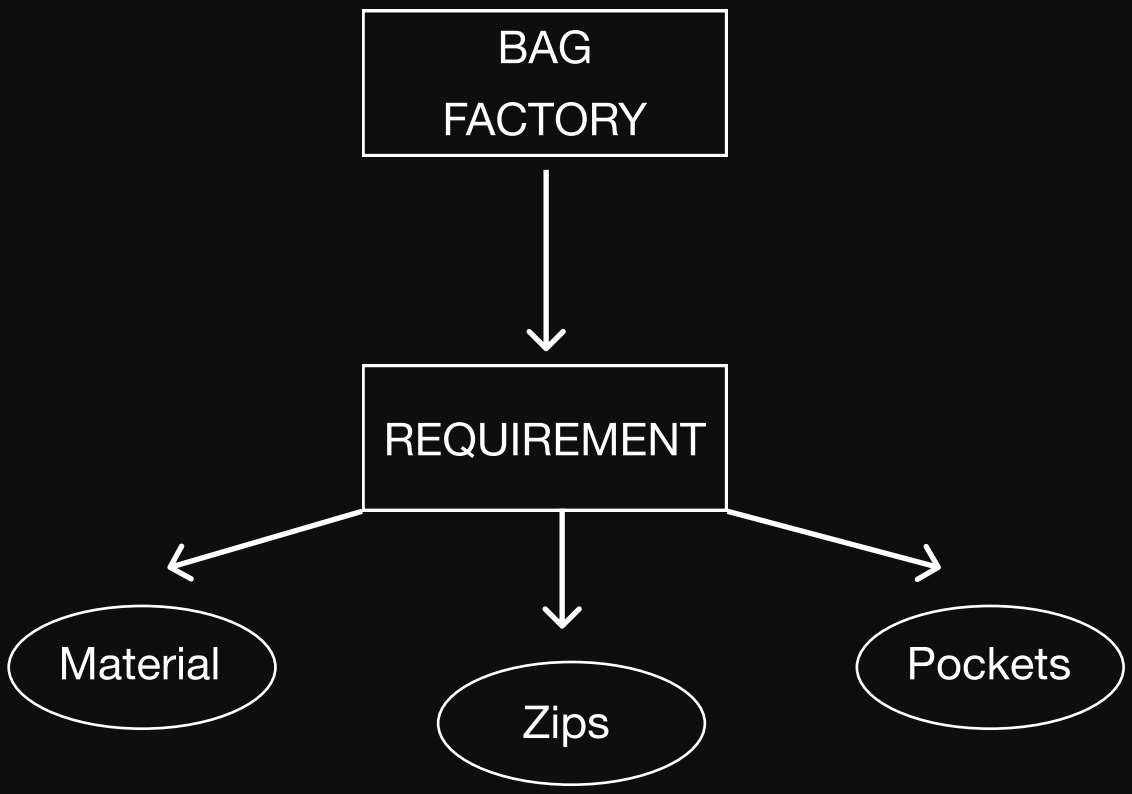
- A class is initialised only one time when we first run the program. and for accessing the attributes and methods we have to first access the class and then attributes and methods.

```
class Animal:
    type = "Cat"  # Attribute

    def sound(self):  # Method
        print("Meow!")

# Directly accessing attribute and method using the class
print(Animal().type)      # Access attribute
Animal().sound()          # Call method
```

Objects



- For understanding objects first look at this example you have a bag factory and that factory requires material of the bag, number of zips you need in that bag and number of pockets you need in your bag.
  - So this is a kind of a blueprint and using this blueprint Reebok, campus and some other companies provided their requirements and created their bags.
  - Thus these companies became objects who created their bags using the blueprint.

Object syntax

- It is very easy to create an object you just have to call the class inside a variable and that variable becomes an object.
- The object has all the powers of a class therefore a class object can access attributes and methods of a class.

```
class Fruit:
    name = "Apple"

# Creating an object
f = Fruit()

# Accessing the attribute
print(f.name)
```

What is constructor

- You saw last example where we wanted material, zips and pockets from the user to create an object.
- If we talk about a function we can ask the user using parameters, but in class we can't have parameters for that we use constructor.
- A constructor is a method that runs automatically when we call a class and this constructor function will target the objects location.

```
class Student:
    def __init__(self, name):
        self.name = name  # Instance attribute

# Creating an object with a value
s = Student("Riya")

# Accessing the attribute
print(s.name)
```

- To target the objects location we use self keyword.
- For clear understanding watch the video carefully. we will create the bag factory and will create multiple objects where self will target the specific locations of the objects.

Types of Attribute

- Class attribute - A normal variable created inside a class is is a class attribute and thats it.
- Instance attribute - A attribute created using an instance like self.name, self.age etc. It is known as instance attribute.

```
class Car:
    wheels = 4  # Class attribute

    def __init__(self, color):
        self.color = color  # Instance attribute
```

Types of Methods

- Instance Method -An instance method Works with instance (object) of the class. This method can access and modify instance attributes.

```
class MyClass:
    def instance_method(self):
        print("This is an instance method")
```

- Class Method - This method works with the class itself it will not target the instance (object). we have to use @classmethod decorator for creating the class method and it takes cls as their first parameter.

```python
class MyClass:
    @classmethod
    def class_method(cls):
        print("This is a class method")
```

- Static Method - This method doesn't access class or instance directly it also uses a decorator @staticmethod it just acts like a regular function placed inside a class.

```python
class MyClass:
    @staticmethod
    def static_method():
        print("This is a static method")
```

Inheritance

- In general terms Inheritance means property or any possession that comes to an heir.

- But our python neither have an old man or a child then inheritance works where ?
- It works between classes.
- Inheritance allows a class (child class) to inherit properties and behaviors (attributes and methods) from another class (parent class).
- Benefits of using inheritance is :
  - Code reusability
  - Organized structure
  - Easy to maintain and extend

Syntax of Inheritance

- Syntax is very simple just like you take parameters in functions here you will take parameters but those parameters will be classes.

```
class Parent:
    def speak(self):
        print("I can speak!")

class Child(Parent):
    pass
```

- Now the inherited class has all the powers of parent class that means all the methods, attributes can be accessed by the instance of child class as well.

Constructor in Inheritance

- Lets say you have created a parent class with a constructor function inside it and then this class is inherited by another class then the constructor function of parent class will work for the child class as well.

```
class Parent:
    def __init__(self, name):
        self.name = name

class Child(Parent):
    def display(self):
        print(f"My name is {self.name}")
```

- Now lets say you need a new parameter in your child class you have to create a constructor function for your child class but the parameters that can be initialized in the parent class will be initialized using the super() function. Super function will target the parent class.

```python
class Parent:
    def __init__(self, name):
        self.name = name

class Child(Parent):
    def __init__(self, name, age):
        super().__init__(name)
        self.age = age

    def display(self):
        print(f"My name is {self.name}, and I am {self.age} years old.")
```

Types of Inheritance

- Single Inheritance
  - All the inheritance we saw above was single level.
- Multiple Inheritance
  - Multiple Inheritance means there will be 2 parent classes and only 1 child class and the child class will inherit all the attributes and methods of both parents.
  - Note - The constructor function will be inherited of the first class that has been Inherited. This is MRO(Method Resolution Order) followed by python.

```python
class Father:
    def skills(self):
        print("Coding")

class Mother:
    def skills(self):
        print("Cooking")

class Child(Father, Mother):
    def show(self):
        print("I have multiple skills")
```

- Multilevel Inheritance
  - This is a basic case where we will have
    - grandparent class → parent class → child class.
    - The attributes and methods are passed on through all the classes.

```python
class Grandparent:
    def heritage(self):
        print("Heritage from Grandparent")

class Parent(Grandparent):
    pass

class Child(Parent):
    pass
```

Polymorphism

- Polymorphism is a core concept in Object-Oriented Programming (OOP). The word means "many forms" — and in programming, it allows the same interface or method name to behave differently depending on the object or context.

Types of Polymorphism

- Polymorphism can be achieved in python in two ways well if we talk about compile time languages there are 3 ways but python does not support Method overloading.
- Method overloading means having same name methods inside a class but parameters will be different but in python the latest definition will overwrite the previous one.

- Method Overriding
  - This is where a child class overrides a method of the parent class, and Python decides at runtime which method to call, based on the object type.

```
class Animal:
    def sound(self):
        print("Animal makes a sound")

class Dog(Animal):
    def sound(self):
        print("Dog barks")
```

- Duck Typing
  - Python follows the philosophy:
    “If it walks like a duck and quacks like a duck, it must be a duck.”

```
class Duck:
    def talk(self):
        print("Quack!")

class Human:
    def talk(self):
        print("Hello!")
```

  - In the speak() function, we don’t care if it's a Duck or a Human — we only care that the object has a talk() method.

Encapsulation

- Encapsulation means putting data (variables) and code (functions) together in one place — inside a class.
- It also means hiding the internal details of how things work, and only showing what is needed.
- ✅ It keeps data safe from being changed by mistake.
- ✅ It makes your code clean and easy to use.
- ✅ It gives control over what others can access or change.

Access modifiers in python

- Access modifiers means how we give access of our attributes and methods to the object or inherited classes. There are 3 types lets see them one by one.
  - Public Attributes and Methods.
    - Till now every attribute and methods we have created are public means the inherited classes and objects can access them no matter what.

  - protected Attributes and Methods.
    - python protected members are created using a single underscore but it still can be accessed from outside the class so you might wonder whats the point of using them.
    - Python doesn't enforce protected access like other languages (e.g., Java or C++). But it uses a naming convention to tell developers

- Private Attributes and Methods
  - A private variable or method means:
  - ❌ It cannot be accessed from outside the class — only from inside the class where it is defined.
  - In Python, we use two underscores (__) before the name to make it private.

```python
class Demo:
    def __init__(self):
        self.name = "Public Member"          # Public
        self._age = 21                       # Protected
        self.__salary = 50000                # Private

    def show(self):
        print("Inside the class:")
        print("Public:", self.name)
        print("Protected:", self._age)
        print("Private:", self.__salary)
```

Abstraction

- Abstraction does not exist in python but we can achieve it using a library we will see what is a library later.
- Abstraction is used to simplifying complex systems by focusing on essential features and hiding unnecessary details.
- It is used to define a common interface for different subclasses.

Abstract classes and methods

- Abstract classes are classes that contains one or more abstract methods.
- A method that is defined but not implemented in the abstract class. subclasses must provide the implementation.

```python
from abc import ABC, abstractmethod

class Animal(ABC):   # Abstract class
    @abstractmethod
    def make_sound(self):   # Abstract method
        pass

class Dog(Animal):
    def make_sound(self):
        print("Dog says Woof!")

class Cat(Animal):
    def make_sound(self):
        print("Cat says Meow!")
```

What are Dunder methods

- Dunder methods are special methods in Python that start and end with double underscores, like __init__, __str__, __add__, etc.
- They automatically get called when you perform certain actions on an object.
- They help you:
  - Customize behavior of your class
  - Make your class objects behave like built-in data types (like strings, lists, etc.)

```
class Person:
    def __init__(self, name):
        self.name = name

p = Person("Ravi")
print(p.name)
```

- Now there are various dunder method see the video for some examples.
- Now lets create a Bank management project using Oops in python.

Decorator

- A decorator is just a function that modifies another function without changing its actual code.
- Imagine you have a cake (your function). A decorator is like putting icing on the cake. It doesn't change the cake itself, but makes it better, prettier, or adds some new flavor!
- For creating a decorator you first have to create a decorator functions and then inside that we will create a wrapper.
- Its tough to understand with text see the video.

```
def my_decorator(func):
    def wrapper():
        print("Something before the function runs.")
        func()
        print("Something after the function runs.")
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")

say_hello()
```

- For making the decorator with Arguments it is tough for this we will move towards our next advance stuff *args , **kwargs.

Args and Kwargs

- They’re special keywords in Python used in function definitions to accept a flexible number of arguments.
- Now you always don’t have to use Args and Kwargs the main thing is * , ** you can use any names in front of them.
- so *args are used for multiple positional arguments, and **kwargs are used for multiple key word arguments.
- And the *args becomes a tuple and **kwargs becomes a dictionary.
- The use case is great
  - You don’t need to know how many inputs you'll get.
  - Helps in building flexible functions, decorators, APIs, and more.

```python
def fun(*args, **kwargs):
    print("Args:", args)
    print("Kwargs:", kwargs)

fun(1, 2, 3, name="Arin", age=21)
```

List, Dictionary and set comphrehension

- All of these Comprehensions are used to create List, Dictionary and set. But you don’t have to use multiple lines of code for loops and If-Else statements.

```
labels = ["Even" if x % 2 == 0 else "Odd" for x in range(5)]
# ['Even', 'Odd', 'Even', 'Odd', 'Even']
```

```
evens = {x: x*x for x in range(10) if x % 2 == 0}
# {0: 0, 2: 4, 4: 16, 6: 36, 8: 64}
```

```
unique_even_squares = {x*x for x in range(10) if x % 2 == 0}
# {0, 4, 16, 36, 64}
```

Lambda functions

- A lambda function is an anonymous, inline function defined using the lambda keyword.
- It's often used for short, simple functions that are used only once or temporarily.
- You can have multiple arguments but there will be only one expression.

- Lets see a basic example:

```
square = lambda x: x**2
print(square(4))  # Output: 16
```

- The argument 4 is passed in x. you can also have multiple arguments and you can also include If - Else expressions

```
check_even = lambda x: "Even" if x % 2 == 0 else "Odd"
print(check_even(7))  # Output: Odd
```

Map filter and zip

- Map is used for applying a function to multiple items.
- Takes a list (or any sequence)
- Applies the same function to every item in that list
- Gives you back a new list (in Python 3, it gives a map object which you can convert to a list)

```
numbers = [1, 2, 3, 4]
doubled = map(lambda x: x * 2, numbers)
print(list(doubled))  # Output: [2, 4, 6, 8]
```

- Use map() when you want to transform every item in a list.
- It doesn't remove or skip items (that's what filter() does).
- You can use it with lambda or normal functions.

- Filter as the name suggest is used to filter out the stuff.
- Takes a list (or other sequence)
- Checks each item using a function (a test)
- Keeps only the items that pass the test (i.e., return True)

```
numbers = [1, 2, 3, 4, 5]
evens = filter(lambda x: x % 2 == 0, numbers)
print(list(evens))  # Output: [2, 4]
```

Modules and packages

- Module is just a single file containing code and we can use this file code in other file.
- A single Python file (.py)
- Contains functions, variables, or classes
- Used to organize and reuse code
- Python comes with lots of ready-to-use modules like:
  - math (for math operations)
  - random (for generating random numbers)
  - datetime (for date and time)

```
import math
print(math.sqrt(16))  # Output: 4.0
```

- A package is a folder that contains one or more modules (Python files). It may also contain sub-packages.
- and you just have to use from and import keywords to use these things. You understood how these things work.
- There are third party packages as well like numpy, pandas, matplotlib etc. and we have to install all of these.
- "And yes, we'll be learning all of these and much more on this channel. I, Akarsh Vyas, as a creator and representative of Sheryians AI, would like to sincerely thank each and every one of you who stayed with us till the end. You are truly precious to us. Much love to all of you!"