26/05/2025

Control structures:

* Control structures: The blocks that analyze variables and choose directions in which to go based on given parameters.

→ Binary Operators:
① works with two operands.
② Operand can be a+b → wher a, b are two operands and 3 is a variable.

→ Relational operators:
① type of binary operator
② compares values on either side and returns a boolean value

OOPS

→ Aims to implement real-world entities like inheritence, polymorphism encapsulation etc. The main concept of OOPs is to bind the data and functions that work on that together as a single unit.

Main concepts of OOPs
① class
② objects
③ polymorphism
④ encapsulation
⑤ Inheritance

* Defining a class
  - length and breadth as attributes
  - init() constructor of class
  - self parameter - refers to newly created instance

Ex: class rectangle:
    def __init__(self):
        self.length = 10
        self.breadth = 5

I/p: rect = retangle()
rect.breadth

o/p: 5

* Parametrised constructer:

Defining:

```
class rectangle:
    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth
```

Instantiating:

```
rect = Rectangle(5, 10)   # 5 → length, 10 → breadth
rect.breadth
```

* Class variable: Defining a variable in a class.

Ex:

```
class circle:
    pi = 3.14          # → class variable
    def __init__(self, radius):   # → instance variable
        self.radius = radius
```

→ How to change class variable: ~~class~~ circle.pi = 40

(circle = class name; pi = class variable)

* Adding a method to the class

- calculate_area() → returns product of length and breadth.
- self identifies its association with instance.

```
class Rectangle:
    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth
    def calculate_area(self):
        return self.length * self.breadth
```

i/p: rect = rectangle(10, 5)

i/p: rect.calculate_area()

o/p: 50