```
    print("Two or Three")
  case _:
    print("Other number")
```

Output:text

```
Two or Three
```

Loops in Python

Loops allow executing a block of code repeatedly, which is essential for tasks that require repetition like processing items in a list, running certain logic multiple times, or iterating until a condition is met.

Types of Loops

1. for Loop

Used to iterate over a sequence like a list, tuple, string, or range of numbers.

Syntax:

python

```
for variable in sequence:
    # code block to repeat
```

Example:

python

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

2. while Loop

Repeats a block of code as long as a given condition is true.

Syntax:

python

```
while condition:
    # code block to repeat
```

Example:

python

```
i = 1
while i < 5:
    print(i)
    i += 1
```

Loop Control Statements

- break: Terminates the loop immediately, skipping the remaining iterations.
- continue: Skips the current iteration and moves to the next iteration of the loop.
- pass: A placeholder that does nothing; useful when a statement is syntactically required but no action is needed.

Nested Loops

You can use one loop inside another to handle multi-level iteration, such as iterating over a matrix or nested lists.

Example:

python

```
for i in range(1, 4):
    for j in range(1, 4):
        print(f"i={i}, j={j}")
```

Using else with Loops

The else block executes after the loop finishes normally (not terminated by break).

Example:

python

```
for i in range(3):
    print(i)
else:
    print("Loop finished without break")
```