EMBA Program
MB-511

Data Science for Managerial Decisions (MB 511)
A Short Course in Data Science using Python

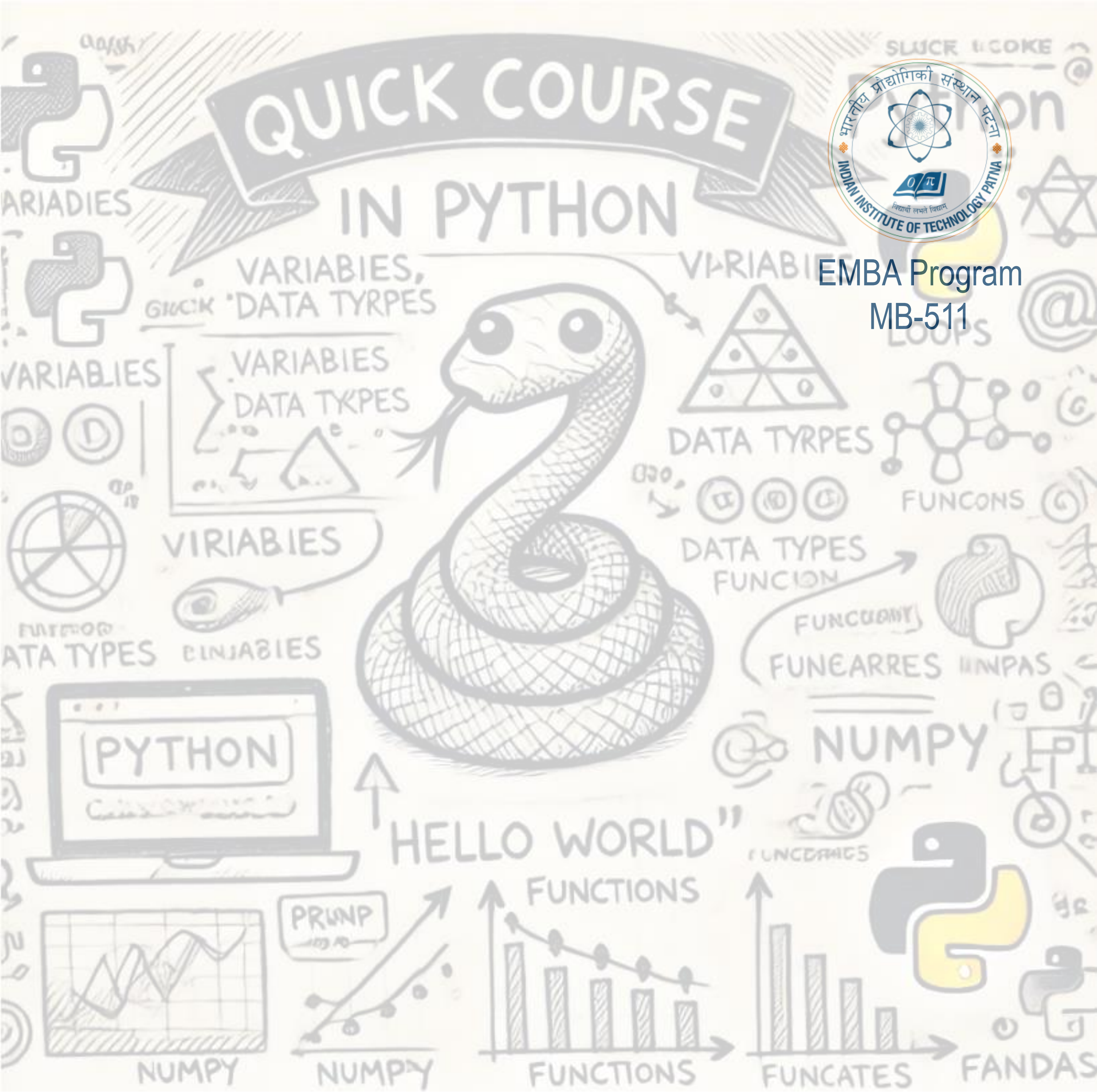
Instructor
Anant Prakash Awasthi

Course

- Getting and setting
- A Quick Introduction
- Industry Products written in Python

- Data Types & Structure
- Loops and Decision Making
- Functions & Libraries
- Error Handling
- Working with Operating System and Dates
- Data Exchange
- Data Management
- Data Visualization
- Feature Engineering
- Quick Introduction to AI
- Model Development and Validation
- Model Deployment

- Case Study 1 – Data Management
- Case Study 2 - Bank Marketing
- Case Study 3 - Student Performance
- Case Study 4 - AI4I 2020 Predictive Maintenance



EMBA Program
MB-511

Getting and setting

Python from Python Website

python

VS Code

Managed Python Distribution

ANACONDA.

Collaboration and Version Control

git

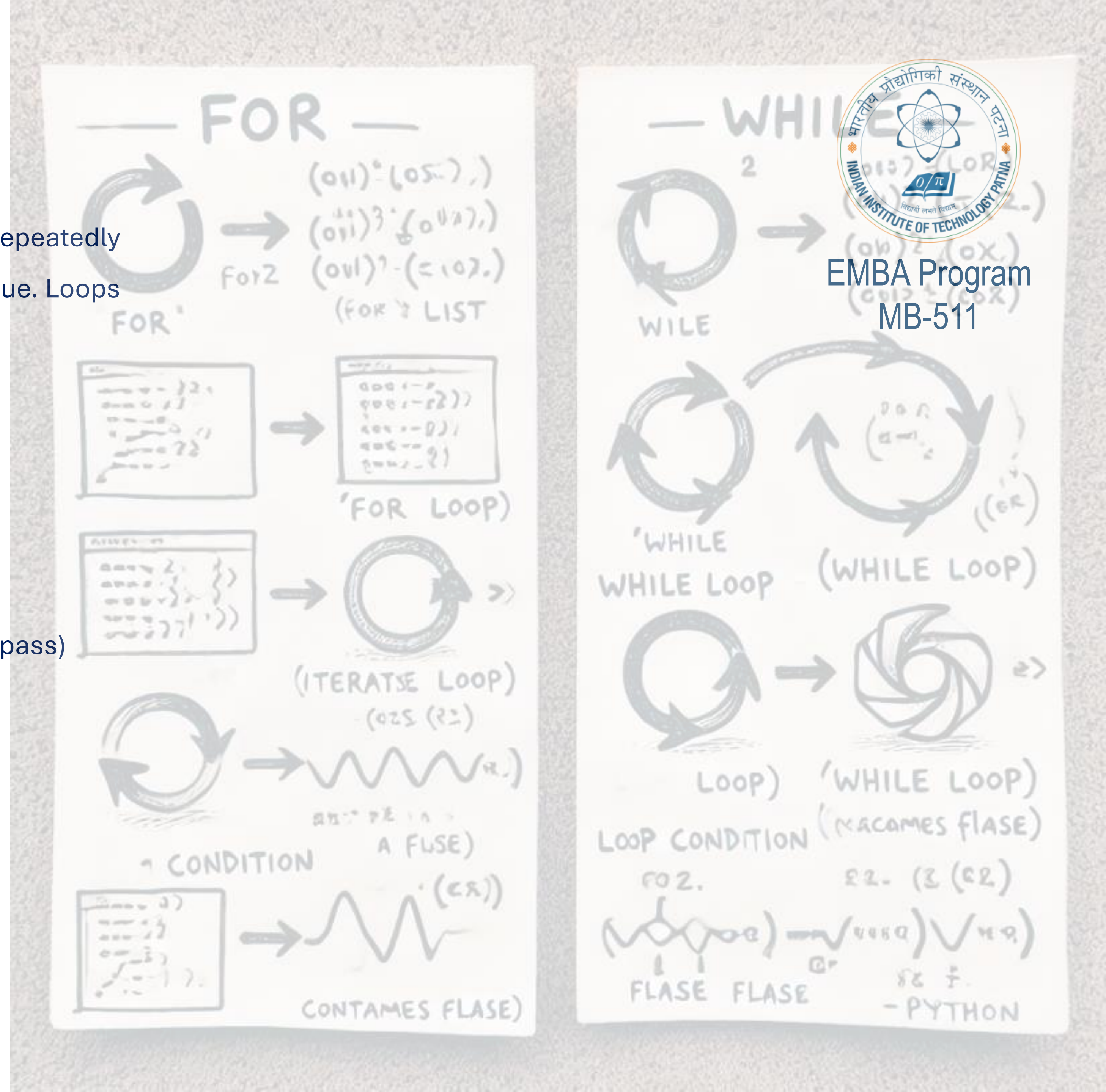
CLI

Introduction to Loops

In programming, loops are constructs that allow you to repeatedly execute a block of code as long as a specified condition is true. Loops help reduce redundancy by automating repetitive tasks.

Key Features:

- Repetition
- Iteration Over Collections
- Conditional Execution
- Flexibility with Control Statements (break, continue, and pass)
- Efficiency
- Nesting

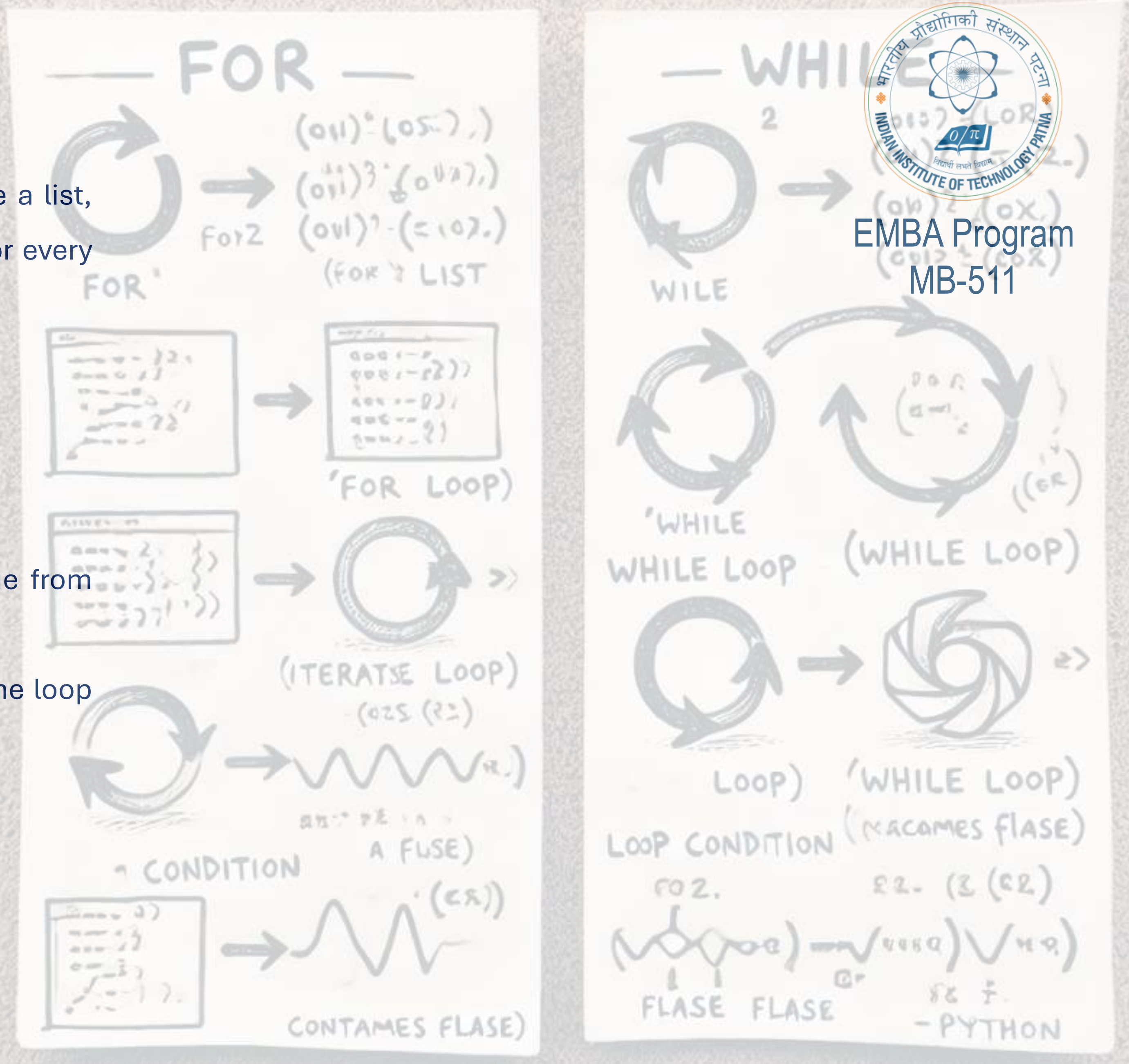


for Loop

A for loop in Python is used for iterating over a sequence (like a list, tuple, dictionary, set, or string). It repeats the block of code for every element in the sequence.

```
for variable in sequence:
    # code to execute
```

- variable: A temporary variable that holds the current value from the sequence during each iteration.
- sequence: The iterable object (like a list, tuple, etc.) that the loop will iterate over.



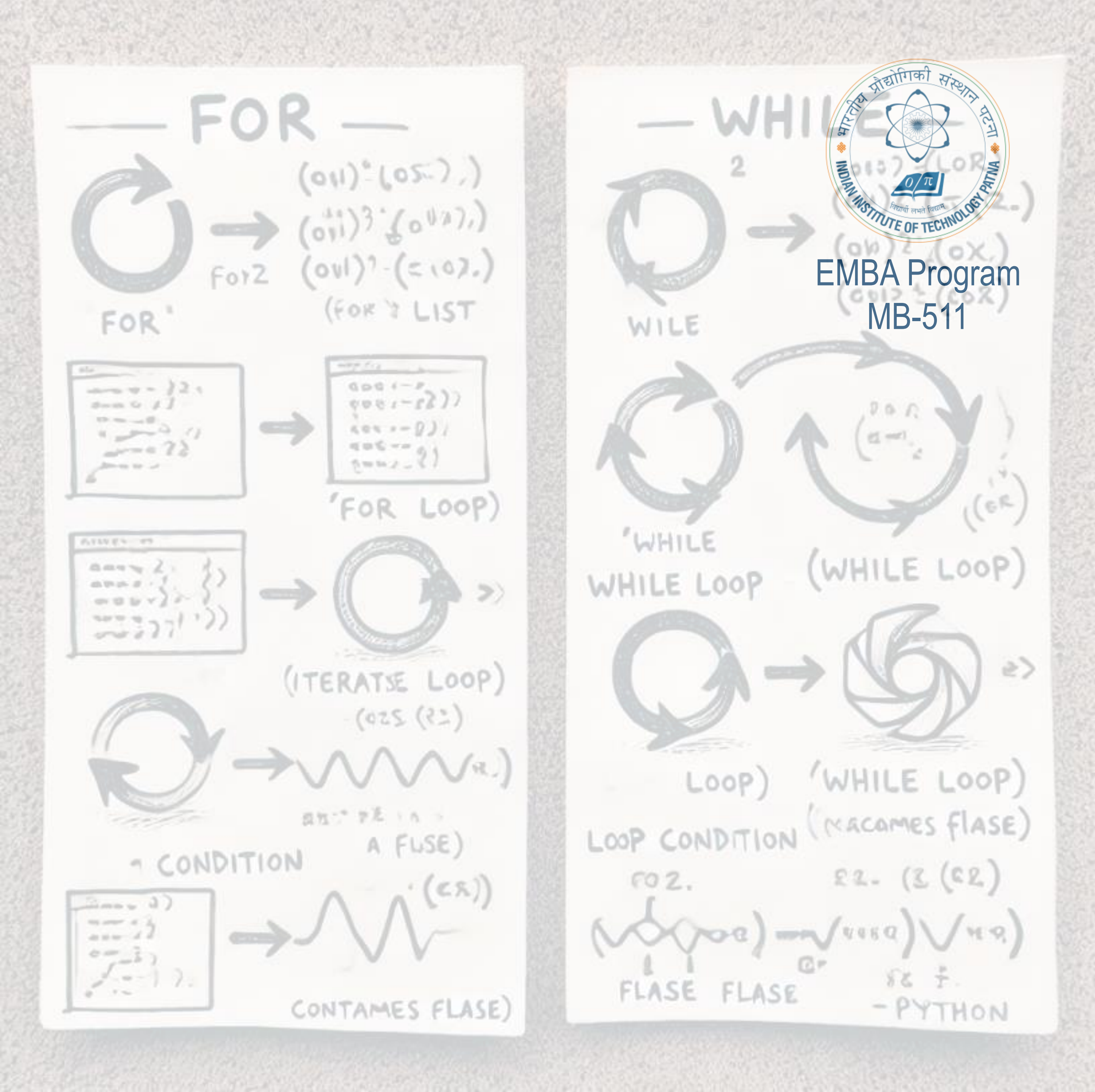
EMBA Program
MB-511

while Loop

A while loop in Python continues executing the code as long as the condition is true. It is more flexible than the for loop since the number of iterations is not fixed.

```
while condition:
    # code to execute
```

- condition: An expression that evaluates to either True or False. As long as it is True, the loop keeps running. If False, the loop stops.



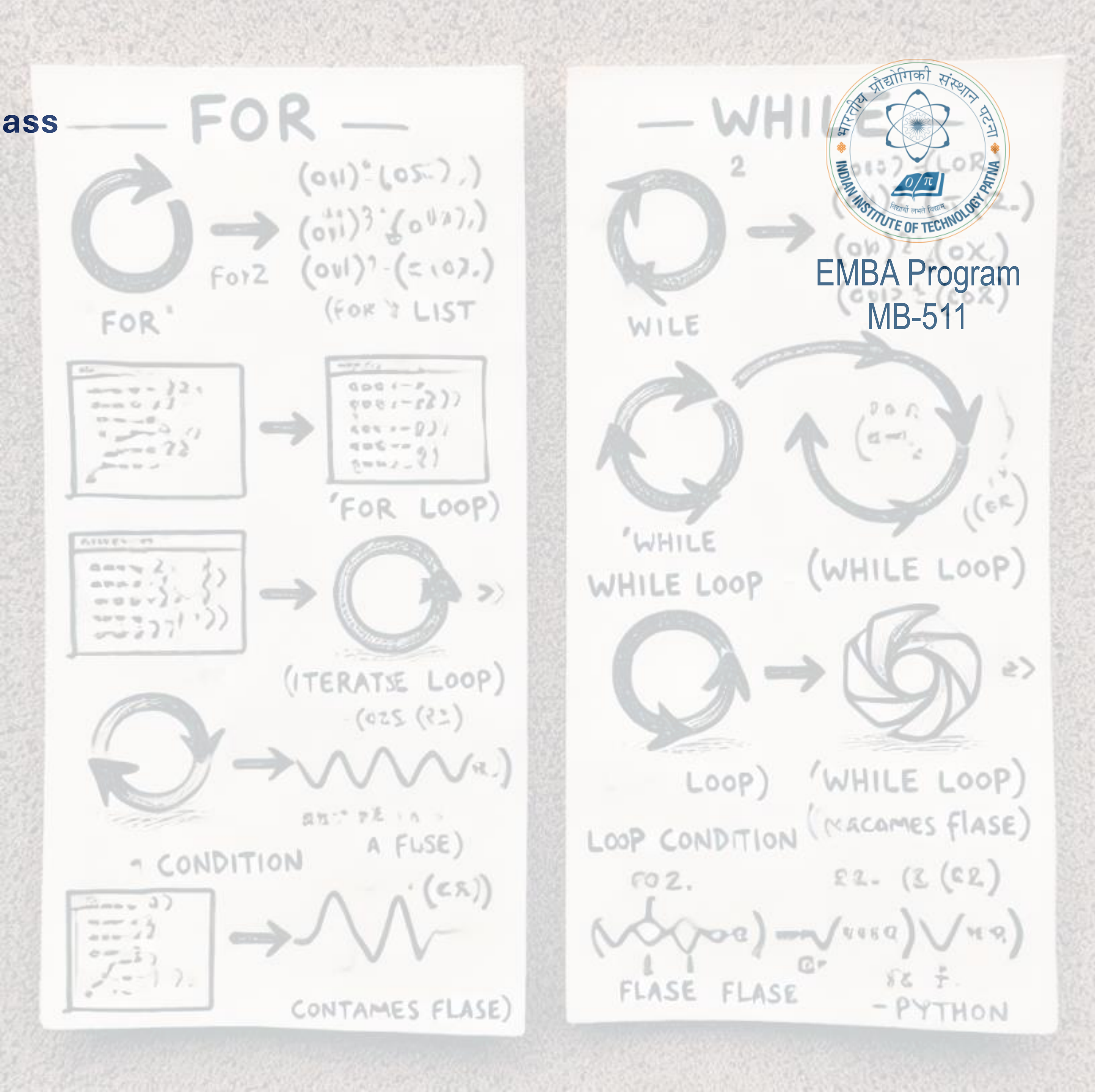
Controlling Loops with break, continue, and pass (break)

The break statement immediately terminates a loop and skips the rest of the code. This is useful when you want to exit the loop early based on some condition.

```
for i in range(10):
   if i == 5:
      break  # exits the loop when i equals 5
   print(i)
```

Output:

```
0
1
2
3
4
```



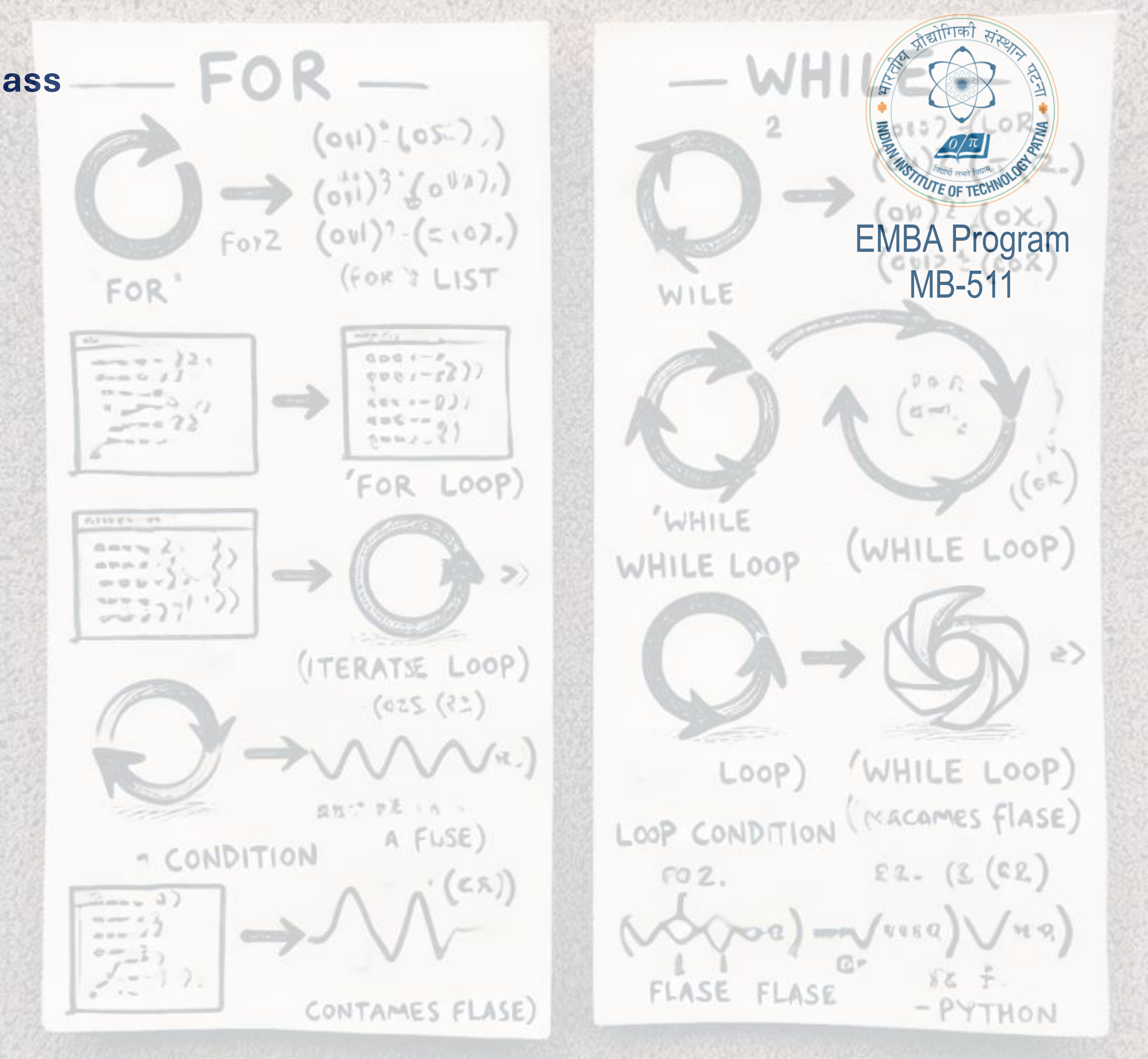
Controlling Loops with break, continue, and pass (continue)

The continue statement skips the current iteration and moves on to the next iteration of the loop.

```
for i in range(5):
   if i == 3:
      continue  # skips the iteration when i equals 3
   print(i)
```

Output:

```
0
1
2
4
```



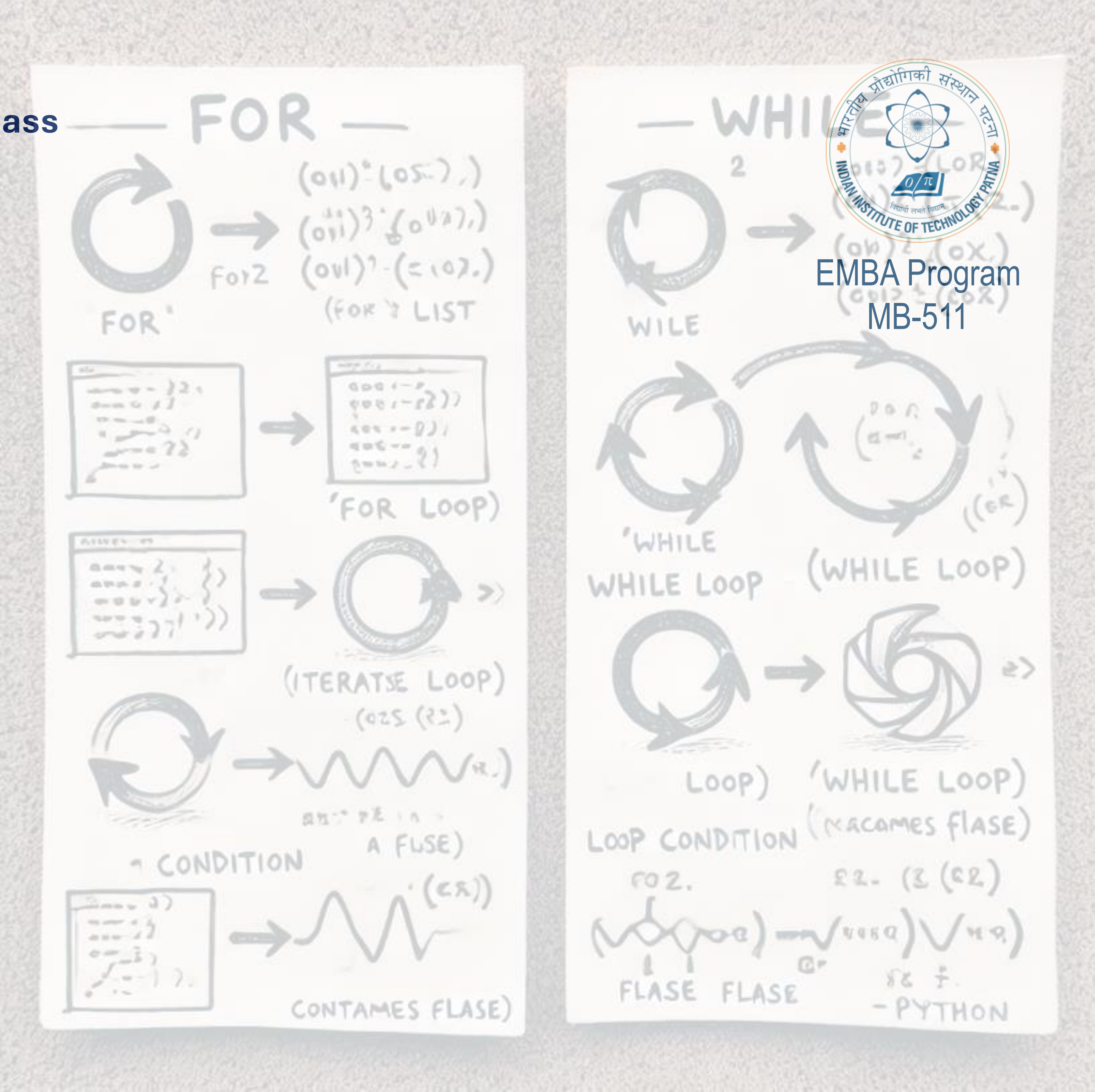
Controlling Loops with break, continue, and pass (pass)

The pass statement is a placeholder that does nothing. It’s useful when a statement is syntactically required but you don’t want to execute any code at that point.

```
for i in range(5):
   if i == 3:
      pass  # does nothing, just passes
   print(i)
```

Output:

```
0
1
2
3
4
```



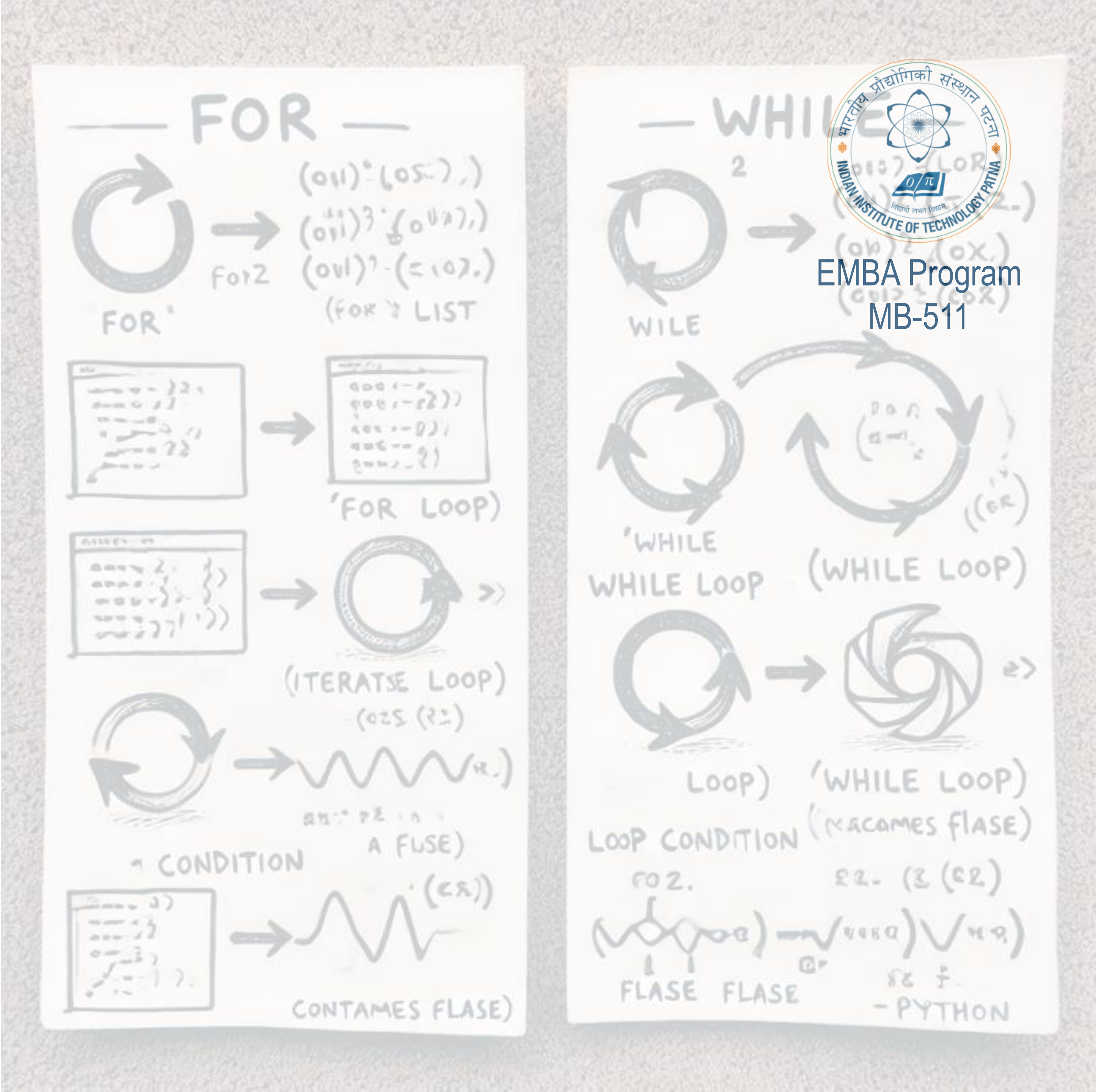
Nested Loops – for loop

Python allows loops to be nested, meaning one loop can be placed inside another. This is useful when working with multi-dimensional data structures like lists of lists.

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for row in matrix:
    for item in row:
        print(item, end=" ")
    print()  # for a new line after each row
```

Output:

```
1 2 3
4 5 6
7 8 9
```

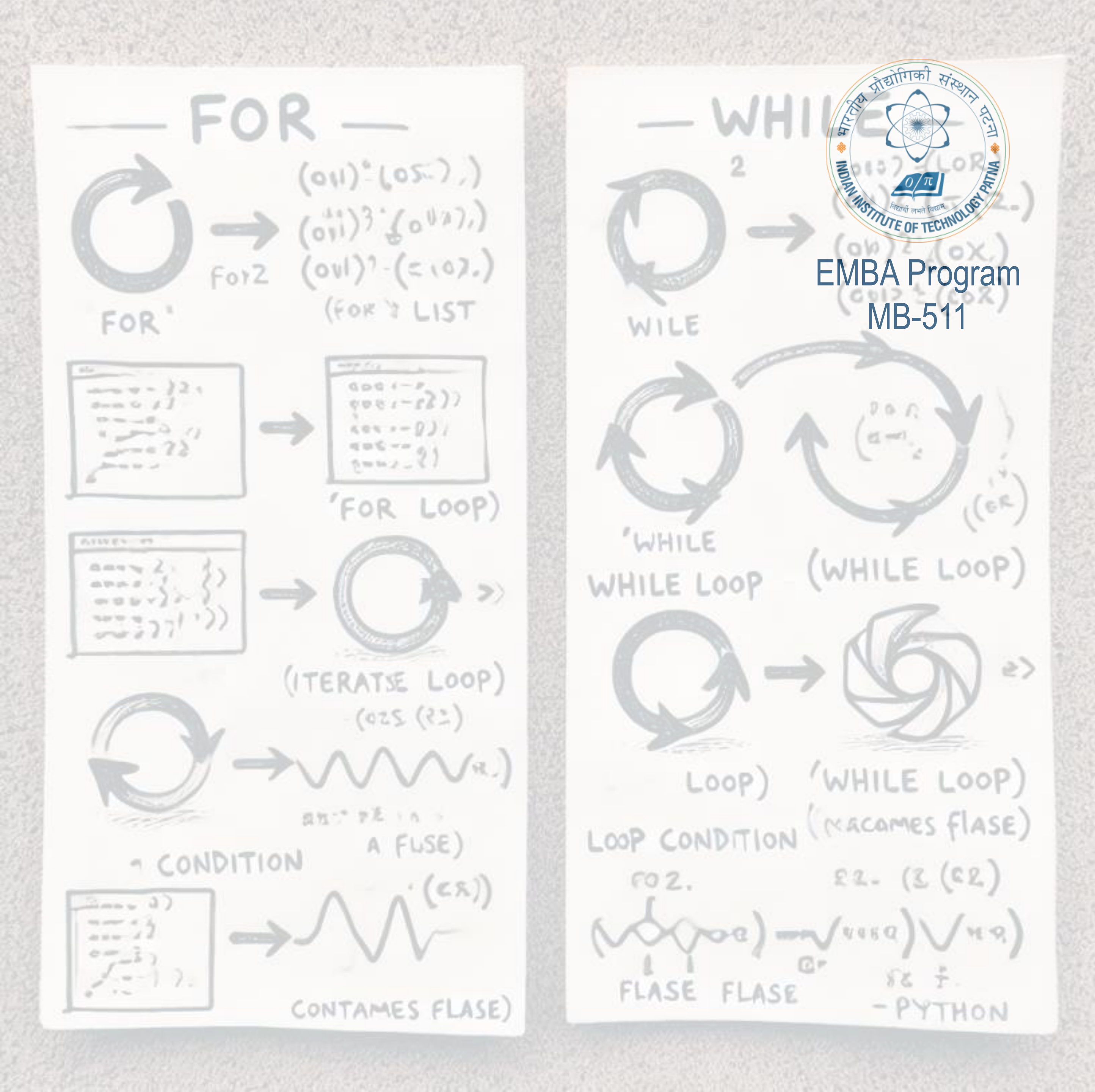


Nested Loops – while loop

```
i = 1
while i <= 3:
  j = 1
  while j <= 3:
    print(f"{i}, {j}")
    j += 1
  i += 1
```

Output:

```
1, 1
1, 2
1, 3
2, 1
2, 2
2, 3
3, 1
3, 2
3, 3
```



Have a question?

Feel Free to Reach out at

- +91-88846-92929 (WhatsApp)
- anant.awasthi@outlook.com (E-Mail)