Machine Learning

Python Programming Basics & Operators

Control Flow · Conditions · Loops

Class Objective

By the end of this class, students will be able to:

- Understand how programs make decisions
- Use conditional statements (`if`, `elif`, `else`)
- Implement repetition using `for` and `while` loops
- Control loop execution using `break` and `continue`
- Combine conditions and loops to solve real problems

Why Control Flow Matters

Without control flow, programs:

- Run line by line only once
- Cannot make decisions
- Cannot repeat tasks

Control flow allows programs to:

- Choose different paths
- React to data
- Automate repetitive work

This is fundamental to **machine learning pipelines**.

Control Flow in Python

Python provides:

- **Conditional statements** → decision making
- **Loops** → repetition
- **Loop controls** → precise execution control

These structures define **program logic**.

Conditional Statements — Overview

Conditional statements execute code **only if a condition is true**.

Python keywords:

- `if`
- `elif`
- `else`

Conditions evaluate to:

- `True`
- `False`

if Statement

Syntax:

```
if condition:
    # code executes if condition is True
```

Example:

```
age = 20

if age >= 18:
    print("Eligible to vote")
```

if–else Statement

Syntax:

```
if condition:
    # True block
else:
    # False block
```

Example:

```
marks = 40

if marks >= 50:
    print("Pass")
else:
    print("Fail")
```

if–elif–else Statement

Used when multiple conditions exist.

Example:

```
score = 78

if score >= 90:
    print("Excellent")
elif score >= 60:
    print("Good")
elif score >= 50:
    print("Pass")
else:
    print("Fail")
```

Only one block executes.

Conditions & Comparison Operators

Operator	Meaning
`==`	Equal
`!=`	Not equal
`<`	Less than
`>`	Greater than
`<=`	Less than or equal
`>=`	Greater than or equal

Example:

```
x = 5
y = 10
print(x < y)
```

Logical Operators in Conditions

Operator	Meaning
`and`	Both conditions true
`or`	At least one true
`not`	Reverse condition

Example:

```
age = 25
if age >= 18 and age < 60:
    print("Working age group")
```

Common Conditional Mistakes

- Using `=` instead of `==`
- Forgetting colon `:`
- Incorrect indentation
- Overly complex nested conditions

Tip: keep conditions simple and readable.

Loops — Why We Need Them

Loops are used to:

- Repeat tasks
- Process lists and datasets
- Train models over multiple iterations

Python supports:

- `for` loop
- `while` loop

for Loop

Used to iterate over sequences.

Syntax:

```
for variable in sequence:
    # loop body
```

Example:

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

Output:

```
1 2 3 4 5
```

for Loop with List

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

for fruit in fruits:
    print(fruit)
```

Common in:

- Dataset processing
- Feature iteration

while Loop

Repeats as long as condition is true.

Syntax:

```
while condition:
    # loop body
```

Example:

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

Avoiding Infinite Loops

Infinite loop occurs when:

- Condition never becomes false

Bad example:

```
while True:
    print("Running forever")
```

Always:

- Update loop variable
- Ensure exit condition

Loop Control Statements

Python provides:

- `break` → exit loop immediately
- `continue` → skip current iteration

Used for fine-grained control.

break Statement

```
for num in range(1, 10):
    if num == 5:
        break
    print(num)
```

Output:

```
1 2 3 4
```

Stops loop entirely.

continue Statement

```
for num in range(1, 6):
    if num % 2 == 0:
        continue
    print(num)
```

Output:

```
1 3 5
```

Skips only current iteration.

Combining Conditions & Loops

Real-world example:

```
values = [4, -2, 7, 0, 9]

for v in values:
    if v <= 0:
        continue
    print(v)
```

Used frequently in:

- Data cleaning
- Feature filtering

Practical Example — Password Attempts

```
attempts = 0

while attempts < 3:
    password = input("Enter password: ")
    if password == "admin123":
        print("Access granted")
        break
    attempts += 1
```

Demonstrates:

- `while` loop
- condition
- `break`

Hands-On Exercise #1

Write a program to:

- Check if a number is positive, negative, or zero
- Use `if`, `elif`, `else`

Hands-On Exercise #2

Write a loop that:

- Prints numbers from 1 to 20
- Skips multiples of 3
- Stops when number reaches 15

Common Beginner Errors

- Incorrect indentation
- Infinite `while` loops
- Using wrong condition
- Misplaced `break` or `continue`

Debug by:

- Printing intermediate values
- Reading error messages carefully

Knowledge Check

1. Difference between `for` and `while` loop
2. What does `break` do?
3. What happens if indentation is incorrect?
4. Write a condition to check if age is between 18 and 60
5. When would you use `continue`?

Summary

- Control flow enables decision-making
- `if`, `elif`, `else` control execution paths
- `for` and `while` enable repetition
- `break` and `continue` refine loop behavior
- These concepts are foundational for ML programming

Next Class

Functions & Modular Programming

- Defining functions
- Parameters & return values
- Reusability
- Introduction to modules