



# Module 34: Control Flow and Loops



## 1. Python Operators

Python provides various operators for performing operations on variables and values. Operators are categorized as follows:

---

### ◆ Arithmetic Operators

Operator	Description	Example	Result
<code>+</code>	Addition	<code>5 + 2</code>	7
<code>-</code>	Subtraction	<code>5 - 2</code>	3
<code>*</code>	Multiplication	<code>5 * 2</code>	10
<code>/</code>	Division	<code>5 / 2</code>	2.5
<code>//</code>	Floor Division	<code>5 // 2</code>	2
<code>%</code>	Modulus	<code>5 % 2</code>	1
<code>**</code>	Exponentiation	<code>2 ** 3</code>	8

---

### ◆ Assignment Operators

Operator	Description	Example	Meaning
<code>=</code>	Assign	<code>x = 5</code>	<code>x = 5</code>
<code>+=</code>	Add and assign	<code>x += 2</code>	<code>x = x + 2</code>
<code>-=</code>	Subtract and assign	<code>x -= 2</code>	<code>x = x - 2</code>
<code>*=</code>	Multiply and assign	<code>x *= 2</code>	<code>x = x * 2</code>
<code>/=</code>	Divide and assign	<code>x /= 2</code>	<code>x = x / 2</code>
<code>//=</code>	Floor divide and assign	<code>x //= 2</code>	<code>x = x // 2</code>

---

### ◆ Comparison Operators

Operator	Description	Example	Result
<code>==</code>	Equal to	<code>x == 5</code>	True/False
<code>!=</code>	Not equal to	<code>x != 3</code>	True/False
<code>&gt;</code>	Greater than	<code>x &gt; 3</code>	True/False
<code>&lt;</code>	Less than	<code>x &lt; 3</code>	True/False
<code>&gt;=</code>	Greater or equal	<code>x &gt;= 3</code>	True/False
<code>&lt;=</code>	Less or equal	<code>x &lt;= 3</code>	True/False

---

## ◆ Logical Operators

Operator	Description	Example	Result
<code>and</code>	True if both are True	<code>x &gt; 2 and x &lt; 10</code>	True/False
<code>or</code>	True if one is True	<code>x &lt; 2 or x &gt; 10</code>	True/False
<code>not</code>	Reverse the result	<code>not(x &gt; 3)</code>	True/False

## ◆ Identity Operators

Operator	Description	Example	Result
<code>is</code>	True if same object	<code>x is y</code>	True/False
<code>is not</code>	True if not same obj	<code>x is not y</code>	True/False

## ◆ Membership Operators

Operator	Description	Example	Result
<code>in</code>	Checks if value in sequence	<code>"a" in "apple"</code>	True
<code>not in</code>	Checks if not in sequence	<code>"x" not in "box"</code>	True

## Commenting in Python

- **Single-line comment:** Use `#`

```
# This is a comment
```

- **Multi-line comment:** Use triple quotes (as docstring or block comment)

```
"""
```

```
This is a  
multi-line comment
```

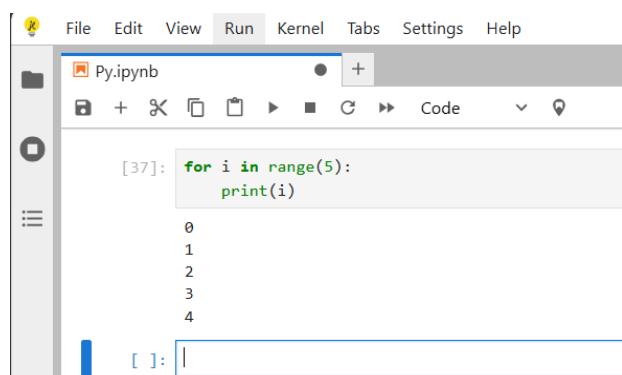
```
"""
```

## 2. Loops in Python

Python supports two types of loops: `for` and `while`.

### ◆ For Loop

Used for iterating over a sequence (list, string, range, etc.)



The screenshot shows a Jupyter Notebook interface. The menu bar includes File, Edit, View, Run, Kernel, Tabs, Settings, and Help. A toolbar below the menu has icons for file operations like new, open, save, and run. The code cell [37] contains the Python code: `for i in range(5): print(i)`. The output cell shows the numbers 0, 1, 2, 3, 4 printed sequentially. The bottom cell input field is empty.

## ◆ While Loop

Repeats as long as the condition is `True`.

A screenshot of a Jupyter Notebook interface. The menu bar includes File, Edit, View, Run, Kernel, Tabs, Settings, and Help. The title bar shows "Py.ipynb". The code cell [38] contains the following Python code:

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

The output cell shows the numbers 1 through 5, each on a new line. The input cell [ ] is empty.

## ◆ Loop Control Statements

Statement	Description
<code>break</code>	Exits the loop immediately
<code>continue</code>	Skips current iteration
<code>pass</code>	Placeholder (does nothing)

## ⌚ 3. List Comprehension

A concise way to create lists in a single line.

A screenshot of a Jupyter Notebook interface. The menu bar includes File, Edit, View, Run, Kernel, Tabs, Settings, and Help. The title bar shows "Py.ipynb". The code cell [39] contains the following Python code:

```
# Create List of squares from 1 to 5
squares = [x**2 for x in range(1, 6)]
print(squares) # [1, 4, 9, 16, 25]
```

The output cell shows the list [1, 4, 9, 16, 25]. The input cell [ ] is empty.

✓ You can also use conditions:

A screenshot of a Jupyter Notebook interface. The menu bar includes File, Edit, View, Run, Kernel, Tabs, Settings, and Help. The title bar shows "Py.ipynb". The code cell [41] contains the following Python code:

```
even_squares = [x**2 for x in range(1, 6) if x % 2 == 0]
print(even_squares)
```

The output cell shows the list [4, 16]. The input cell [ ] is empty.

## 🔍 4. `in` and `not in`

Used to **check membership** in sequences (lists, strings, tuples, etc.)

### ◆ Examples:



```
[42]: 'a' in 'apple'  
[42]: True  
  
[43]: 10 in [10, 20, 30]  
[43]: True  
  
[44]: 'z' not in 'hello'  
[44]: True  
  
[45]: 3 not in [1, 2, 3]  
[45]: False
```

✓ Commonly used in **conditions and loops**.

### 📌 Key Points to Remember

- Operators are essential building blocks for any logic.
- Loops reduce repetition and automate tasks.
- List comprehension makes code **shorter** and **faster**.
- `in` and `not in` are **membership checks**, useful in conditionals.
- Comments are critical for making your code readable.
- Python's simplicity in control flow makes it ideal for beginners and professionals alike.