



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

◆ Assignment Operators

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

◆ Comparison Operators

| Operator | Description | Example | Result |
|----------|------------------|------------------------|-------------------------|
| == | Equal to | <code>x == 5</code> | <code>True/False</code> |
| != | Not equal to | <code>x != 3</code> | <code>True/False</code> |
| > | Greater than | <code>x > 3</code> | <code>True/False</code> |
| < | Less than | <code>x < 3</code> | <code>True/False</code> |
| >= | Greater or equal | <code>x >= 3</code> | <code>True/False</code> |
| <= | Less or equal | <code>x <= 3</code> | <code>True/False</code> |

◆ Logical Operators

| Operator | Description | Example | Result |
|------------------|-----------------------|-------------------------------------|------------|
| <code>and</code> | True if both are True | <code>x > 2 and x < 10</code> | True/False |
| <code>or</code> | True if one is True | <code>x < 2 or x > 10</code> | True/False |
| <code>not</code> | Reverse the result | <code>not(x > 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`.

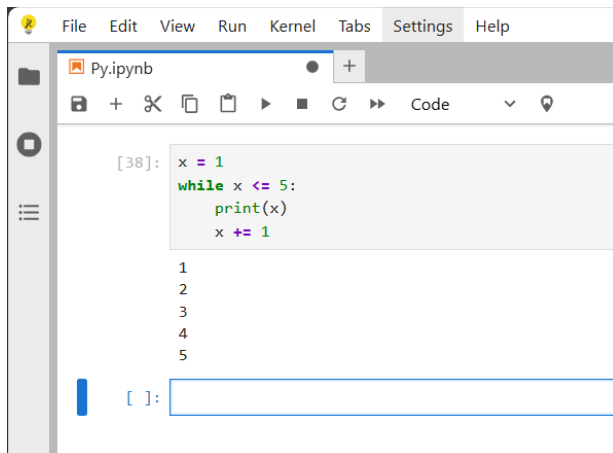
◆ For Loop

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

```
[37]: for i in range(5):  
      print(i)  
  
0  
1  
2  
3  
4  
  
[ ]: |
```

◆ While Loop

Repeats as long as the condition is `True`.



A screenshot of a Jupyter Notebook interface. The top menu bar includes File, Edit, View, Run, Kernel, Tabs, Settings, and Help. The notebook is titled 'Py.ipynb'. The code cell [38] contains a while loop that prints numbers 1 through 5. The output shows the numbers 1, 2, 3, 4, and 5 on separate lines. Below the code cell is an empty input prompt []:

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

```
1
2
3
4
5
```

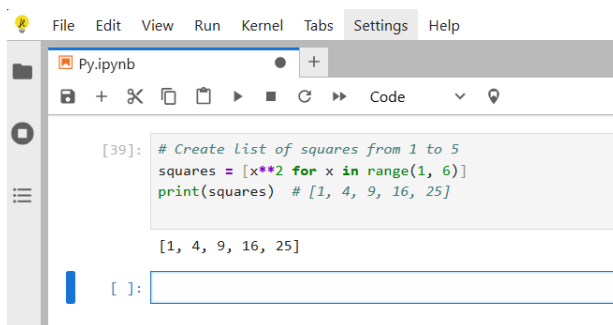
```
[ ]:
```

◆ 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.



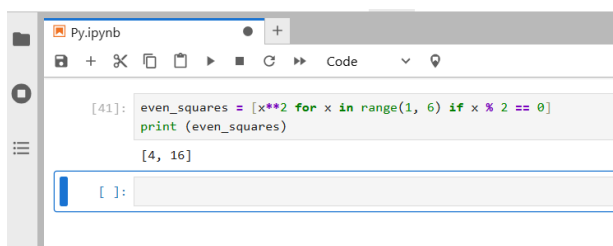
A screenshot of a Jupyter Notebook interface. The top menu bar includes File, Edit, View, Run, Kernel, Tabs, Settings, and Help. The notebook is titled 'Py.ipynb'. The code cell [39] contains a list comprehension that creates a list of squares from 1 to 5. The output shows the list [1, 4, 9, 16, 25]. Below the code cell is an empty input prompt []:

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

```
[1, 4, 9, 16, 25]
```

```
[ ]:
```

✅ You can also use conditions:



A screenshot of a Jupyter Notebook interface. The top menu bar includes File, Edit, View, Run, Kernel, Tabs, Settings, and Help. The notebook is titled 'Py.ipynb'. The code cell [41] contains a list comprehension that creates a list of even squares from 1 to 5. The output shows the list [4, 16]. Below the code cell is an empty input prompt []:

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

```
[4, 16]
```

```
[ ]:
```

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.