



Module 33: Objects, Variables and Data Types

🛠️ Setting Up Python

▶ Option 1: Practice Online with Jupyter (No Installation Needed)

- Use Python directly in your browser.
- Great for quick **practice, learning, and experiments.**

🌐 Website: <https://jupyter.org/try-jupyter/lab/>

Features:

- No installation or sign-in required
- Supports markdown + code
- Works on PC, tablet, or mobile browser
- Temporary sessions (data not saved permanently)

▶ Option 2: Install Python Locally

1. Download from: <https://www.python.org/downloads>
2. Run installer (tick "Add Python to PATH")
3. Use with any code editor (IDLE, VS Code, PyCharm, etc.)

◆ 1. Objects and Variables

- In Python, **everything is an object**
- A **variable** is a name that refers to an object (like a label)

x = 10

name = "Ajay"

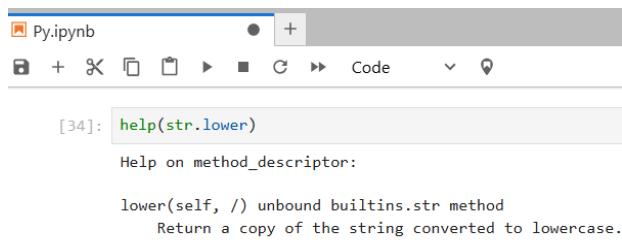
🔧 Methods

- Methods are actions associated with objects. Example:

The screenshot shows a Jupyter Notebook interface. The menu bar includes File, Edit, View, Run, Kernel, Tabs, Settings, and Help. The main area displays two code cells. Cell [35] contains the assignment statements x = 10 and name = "Ajay". Cell [36] contains the method call name.lower(). The output of cell [36] is 'ajay'.

```
File Edit View Run Kernel Tabs Settings Help
Py.ipynb + 
[35]: x = 10
       name = "Ajay"
[36]: name.lower()
[36]: 'ajay'
[ ]:
```

📘 Use `help()` to explore object capabilities:



A screenshot of a Jupyter Notebook interface. The title bar says "Py.ipynb". The toolbar includes icons for file operations and code execution. A code cell at index [34] contains the command `help(str.lower)`. The output shows the documentation for the `lower` method of the `str` class, which converts a string to lowercase.

```
[34]: help(str.lower)
Help on method_descriptor:

lower(self, /) unbound builtins.str method
    Return a copy of the string converted to lowercase.
```

✓ Good Coding Practices

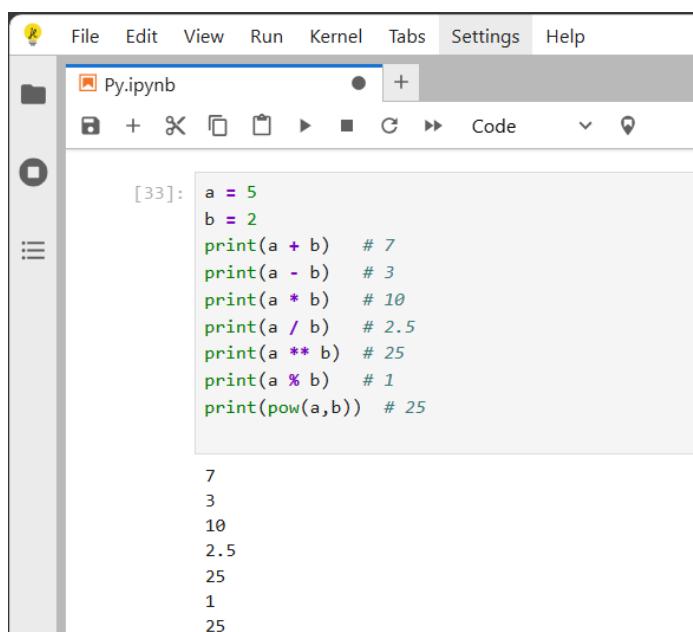
- Use **meaningful variable names**
- Use **snake_case** for variables (e.g. `student_name`)
- Always maintain **proper indentation** (4 spaces)
- Use comments (`#`) to explain steps

12 34 2. Numbers in Python

🔢 Main Number Types:

Type	Description	Example
Integer	Whole numbers	<code>10</code> , <code>-3</code>
Float	Decimal numbers	<code>3.14</code>
Complex	Real + Imaginary	<code>2 + 3j</code>

⌚ Arithmetic Operations:



A screenshot of a Jupyter Notebook interface. The title bar says "Py.ipynb". The toolbar includes icons for file operations and code execution. A code cell at index [33] contains a script that performs various arithmetic operations using variables `a` and `b`. The output shows the results of addition, subtraction, multiplication, division, exponentiation, modulus, and the power function.

```
[33]: a = 5
b = 2
print(a + b)    # 7
print(a - b)    # 3
print(a * b)    # 10
print(a / b)    # 2.5
print(a ** b)   # 25
print(a % b)    # 1
print(pow(a,b)) # 25
```

```
7
3
10
2.5
25
1
25
```

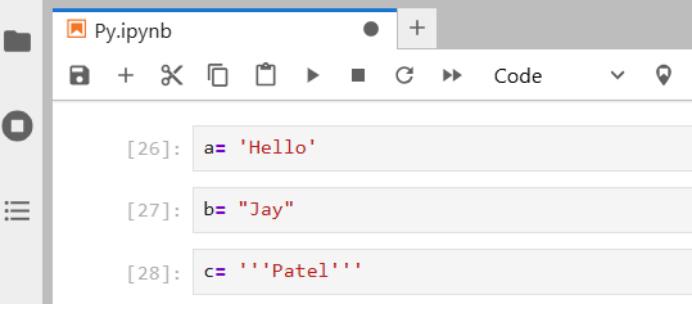
✓ Python follows **PEDMAS** (Parentheses, Exponent, Division/Multiplication, Addition/Subtraction)

3. Strings in Python

- Strings are sequences of characters.

Ways to Declare Strings:

'Single quotes'
"Double quotes"
'''Triple quotes for multiline'''



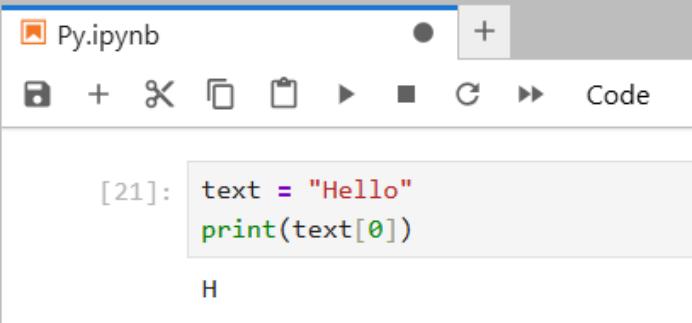
```
[26]: a='Hello'
[27]: b="Jay"
[28]: c=''' Patel'''
```

Special Characters:

Symbol	Purpose
\n	New line
\'	Single quote
\\"	Double quote
\\\	Backslash

4. String Operations

String Indexing



```
[21]: text = "Hello"
       print(text[0])
```

H

String Slicing

```
[23]: text[1:4]
[23]: 'ell'

[24]: text[:3]
[24]: 'Hel'

[25]: text[::-2]
[25]: 'Hlo'

[ ]:
```

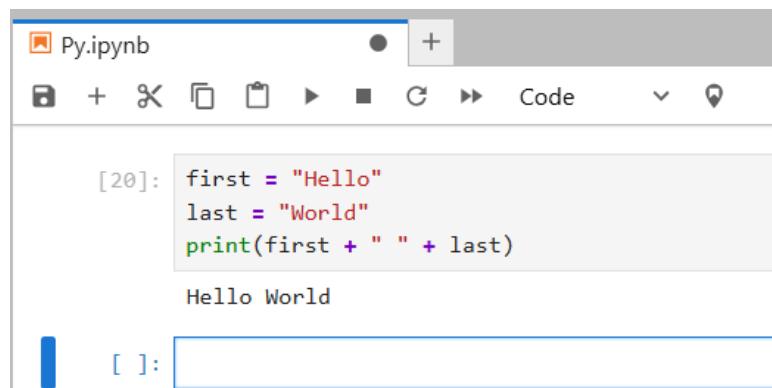
5. String Methods & Properties

Strings are immutable (cannot be changed after creation)

Method	Description	Syntax	Example
<code>upper()</code>	Converts to UPPERCASE	<code>s.upper()</code>	<code>'abc'.upper() → 'ABC'</code>
<code>lower()</code>	Converts to lowercase	<code>s.lower()</code>	<code>'XYZ'.lower() → 'xyz'</code>
<code>find()</code>	Finds index of a char	<code>s.find('x')</code>	<code>'box'.find('x') → 2</code>
<code>replace()</code>	Replace part of string	<code>s.replace(a,b)</code>	<code>'car'.replace('a','u') → 'cu'</code>
<code>split()</code>	Break into list	<code>s.split()</code>	<code>'a b c'.split() → list</code>
<code>islower()</code>	Check all lowercase	<code>s.islower()</code>	<code>'abc'.islower() → True</code>
<code>isupper()</code>	Check all uppercase	<code>s.isupper()</code>	<code>'ABC'.isupper() → True</code>
<code>strip()</code>	Remove outer whitespace	<code>s.strip()</code>	<code>' abc '.strip() → 'abc'</code>

+ 6. String Concatenation & Formatting

Concatenation using `+`

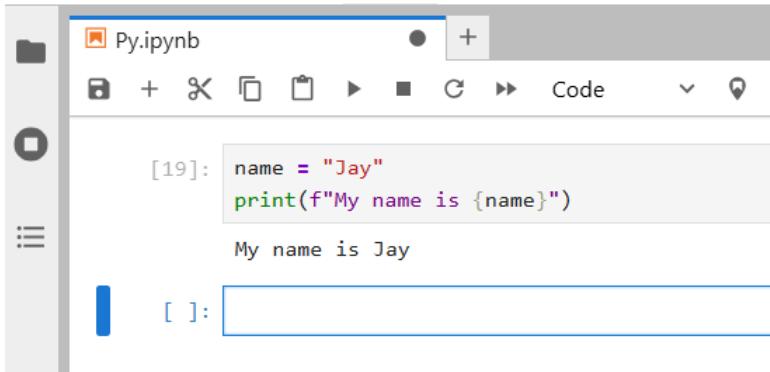


Py.ipynb

```
[20]: first = "Hello"
       last = "World"
       print(first + " " + last)
```

Hello World

Formatting using f-strings



A screenshot of a Jupyter Notebook interface. The title bar says "Py.ipynb". The toolbar includes icons for file operations, cell selection, and code execution. In the code cell, the following Python code is run:

```
[19]: name = "Jay"
print(f"My name is {name}")
```

The output cell shows the result:

```
My name is Jay
```

7. List – Ordered & Mutable Collection

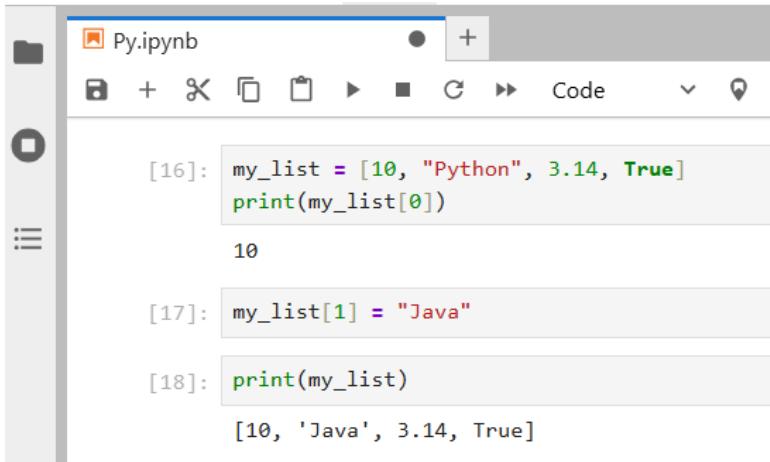
What is a List?

- A **list** is a **sequence** of items.
- It is **ordered** and **mutable**
- Can contain **mixed types**: numbers, strings, lists, etc.

Why Lists Are Useful:

- Dynamic resizing
- Ideal for iteration, collections, queues, and loops
- Nesting possible: lists inside lists

Example:



A screenshot of a Jupyter Notebook interface. The title bar says "Py.ipynb". The toolbar includes icons for file operations, cell selection, and code execution. In the code cell, the following Python code is run:

```
[16]: my_list = [10, "Python", 3.14, True]
print(my_list[0])
```

The output cell shows the result:

```
10
```

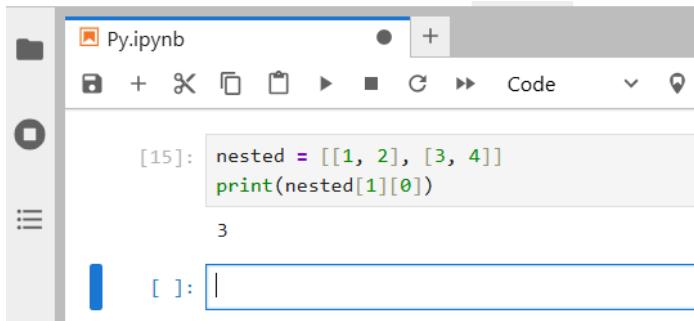
```
[17]: my_list[1] = "Java"
```

```
[18]: print(my_list)
```

The final output cell shows the updated list:

```
[10, 'Java', 3.14, True]
```

Nested List Example:



A screenshot of a Jupyter Notebook interface. The title bar says "Py.ipynb". The code cell at index 15 contains the following Python code:

```
nested = [[1, 2], [3, 4]]
print(nested[1][0])
```

The output cell shows the result "3". Below the code cell, there is an empty input cell starting with "[]:".

List Methods:

Method	Description	Syntax	Example
<code>append()</code>	Add item to end	<code>list.append(x)</code>	<code>l.append(4)</code>
<code>extend()</code>	Add multiple items	<code>list.extend([x,y])</code>	<code>l.extend([5,6])</code>
<code>insert()</code>	Add item at index	<code>list.insert(i,x)</code>	<code>l.insert(1, 99)</code>
<code>remove()</code>	Remove item by value	<code>list.remove(x)</code>	<code>l.remove(3)</code>
<code>pop()</code>	Remove item by index	<code>list.pop()</code>	<code>l.pop()</code>
<code>clear()</code>	Empty the list	<code>list.clear()</code>	
<code>sort()</code>	Sort items (ascending)	<code>list.sort()</code>	
<code>reverse()</code>	Reverse the list	<code>list.reverse()</code>	
<code>index()</code>	Get index of item	<code>list.index(x)</code>	

8. Dictionary – Key-Value Mapping

What is a Dictionary?

- Unordered (in < 3.7), now ordered (3.7+)
- Stores **key-value pairs**
- Keys must be unique & immutable

Why Dictionaries are Useful:

- Fast data retrieval
- Real-life mapping (student data, user profiles)
- Can hold any type of value, including lists & other dictionaries

Example:

```
[12]: student = {
    "name": "Ajay",
    "age": 15,
    "marks": [80, 90, 70]
}

print(student["name"])

Ajay

[13]: student["age"] = 16
student["class"] = "10th"

[14]: print (student)

{'name': 'Ajay', 'age': 16, 'marks': [80, 90, 70], 'class': '10th'}
```

🔗 Nested Dictionary:

```
[11]: school = {
    "classA": {"students": 30},
    "classB": {"students": 25}
}

print(school["classB"]["students"])

25
```

📦 Dictionary Methods:

Method	Description	Example
<code>get()</code>	Get value from key	<code>d.get("name")</code>
<code>keys()</code>	Get all keys	<code>d.keys()</code>
<code>values()</code>	Get all values	<code>d.values()</code>
<code>items()</code>	Get key-value pairs	<code>d.items()</code>
<code>update()</code>	Add another dictionary	<code>d.update({'grade':10})</code>
<code>pop()</code>	Remove a key	<code>d.pop("age")</code>

📦 9. Tuples and Sets

◆ Tuples – Immutable, Ordered

- Cannot change once created
- Used for safe, fixed data
- Supports indexing and nesting

A screenshot of 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 main area shows a code cell with the following content:

```
[10]: t = (1, 2, 3)
      print(t[1])
      2
```

The cell output is displayed below the code.

⚠ Note: Use a comma for single-element tuple → (5,)

◆ Sets – Unordered, Unique

- No duplicate values
- Cannot be indexed
- Good for membership tests & uniqueness

A screenshot of 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 main area shows a code cell with the following content:

```
[5]: s = {1, 2, 3, 3}
      print(s)
      {1, 2, 3}

[6]: s.add(4)

[7]: print(s)
      {1, 2, 3, 4}

[8]: s.remove(2)

[9]: print(s)
      {1, 3, 4}
```

The cell output is displayed below the code.

🔄 Set Operations:

A screenshot of 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 main area shows a code cell with the following content:

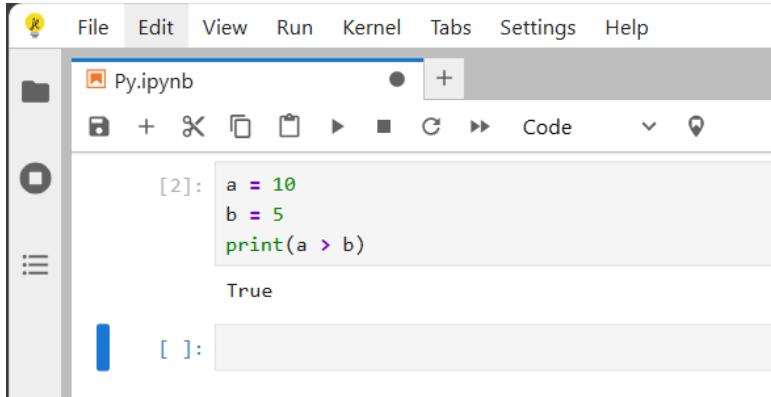
```
[3]: a = {1, 2, 3}
      b = {2, 3, 4}
      print(a | b)
      print(a & b)
      print(a - b)
      {1, 2, 3, 4}
      {2, 3}
      {1}
```

The cell output is displayed below the code.

10. Booleans

Two values: `True` and `False` (capital T/F)

Example:



A screenshot of 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 main area shows a code cell [2]:

```
a = 10
b = 5
print(a > b)
```

The output of the cell is "True". Below the cell is another empty cell []:.

Boolean Operators:

- `and`
- `or`
- `not`

Key Points to Remember

Data Type	Ordered	Mutable	Duplicates	Common Use
List	✓	✓	✓	Sequences, stacks, queues
Tuple	✓	✗	✓	Fixed collections (e.g., coordinates)
Set	✗	✓	✗	Uniqueness, fast membership check
Dict	✓ (3.7+)	✓	✗ (Keys)	Labelled data, lookups

- ◆ Practice with all types
- ◆ Strings are immutable, lists/dictionaries are mutable
- ◆ Use the correct data type based on your goal
- ◆ Strings, lists, dictionaries, tuples, and sets are the **core of Python**