📘 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.)

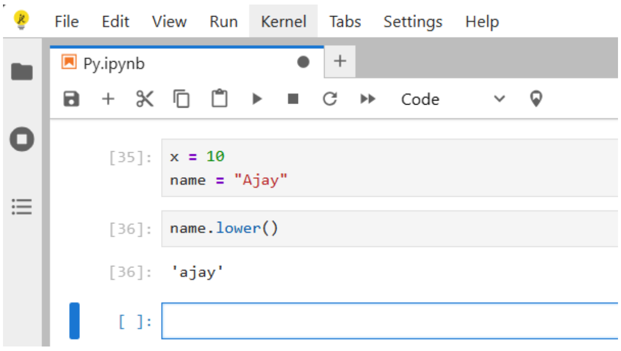
🔷 1. Objects and Variables

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

x = 10
name = "Ajay"

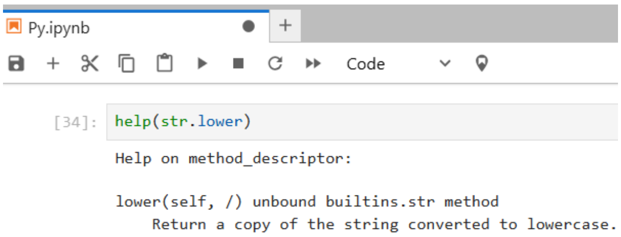
🔧 Methods

- Methods are actions associated with objects. Example:



```
[35]: x = 10
      name = "Ajay"

[36]: name.lower()

[36]: 'ajay'

[ ]:
```

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



```
[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

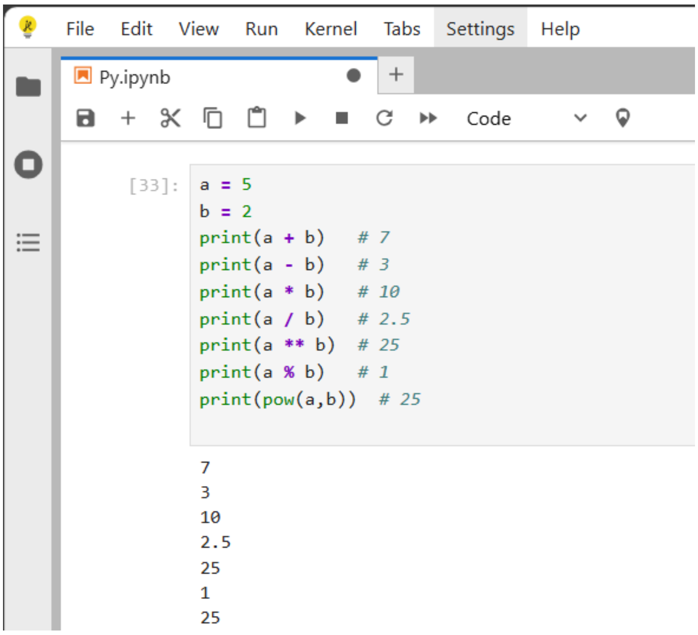
- Use **meaningful variable names**
- Use **snake_case** for variables (e.g. `student_name`)
- Always maintain **proper indentation** (4 spaces)
- Use comments (`#`) to explain steps

🔢 2. Numbers in Python

🧮 Main Number Types:

Type	Description	Example
Integer	Whole numbers	`10`, `-3`
Float	Decimal numbers	`3.14`
Complex	Real + Imaginary	`2 + 3j`

🔗 Arithmetic Operations:



```
[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
```

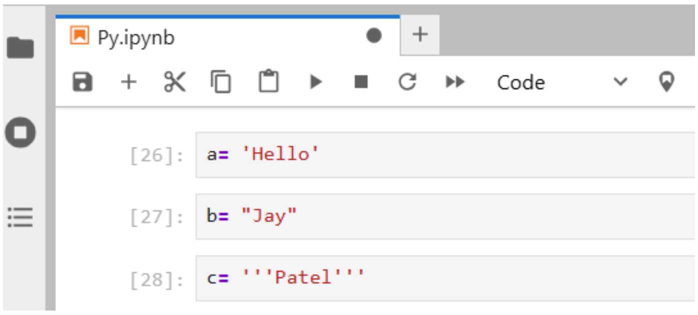
✅ Python follows **PEDMAS** (Parentheses, Exponent, Division/Multiplication, Addition/Subtraction)

🔤 3. Strings in Python

- Strings are sequences of characters.

📝 Ways to Declare Strings:

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



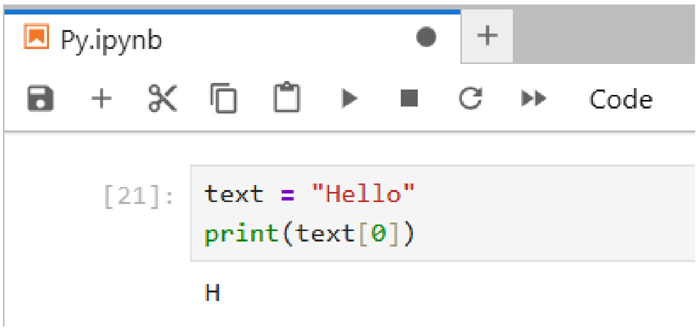
```
[26]: a= 'Hello'
[27]: b= "Jay"
[28]: c= '''Patel'''
```

🔁 Special Characters:

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

✂️ 4. String Operations

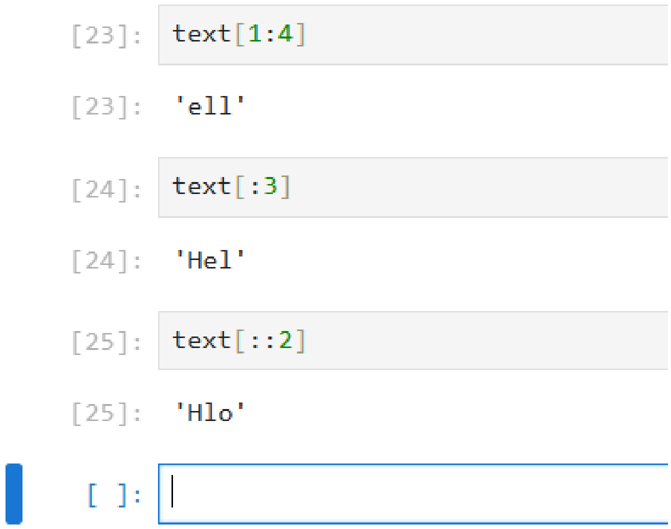
🔢 String Indexing



```
[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'

[ ]:
```

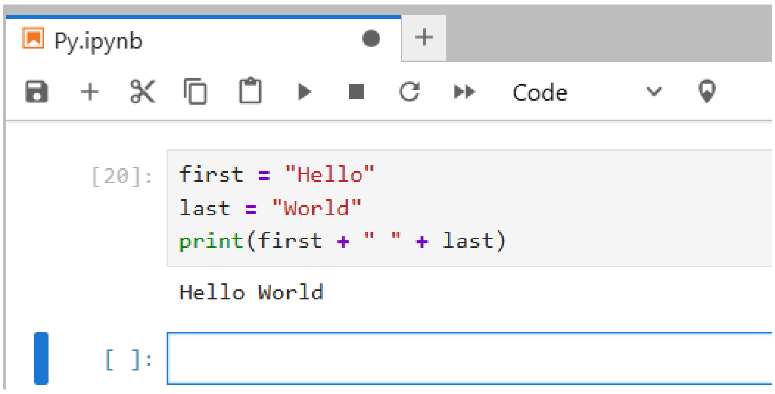
🧰 5. String Methods & Properties

✅ **Strings are immutable** (cannot be changed after creation)

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

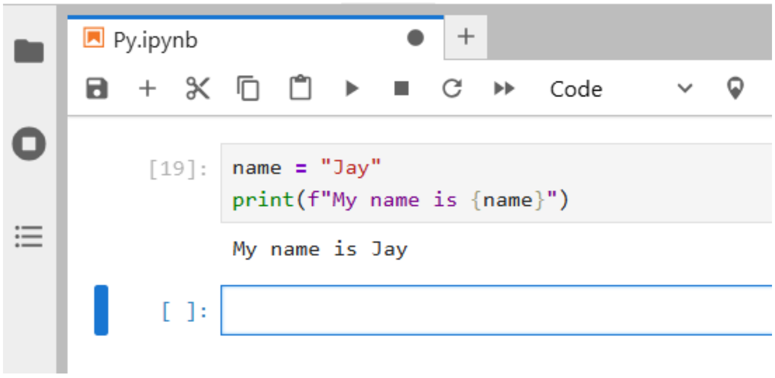
➕ 6. String Concatenation & Formatting

🔗 Concatenation using `+`



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

      Hello World

[ ]:
```

🧩 Formatting using f-strings



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

      My name is Jay

[ ]:
```

🧾 7. List – Ordered & Mutable Collection

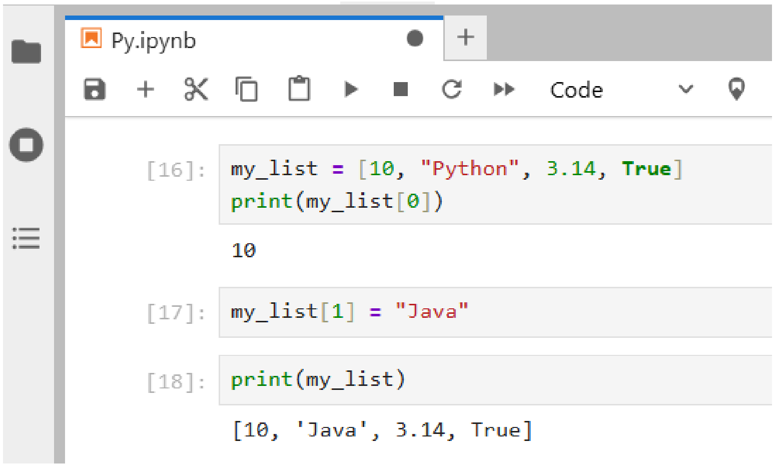
🔍 What is a List?

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

✅ Why Lists Are Useful:

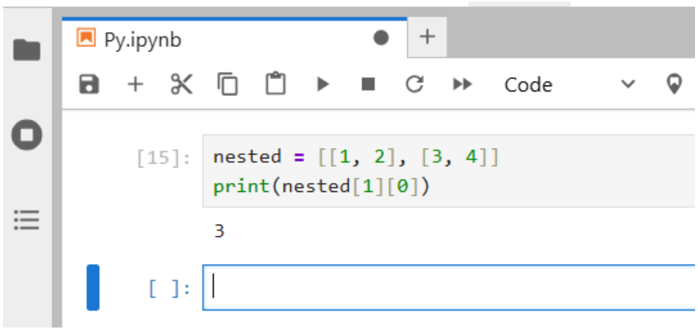
- Dynamic resizing
- Ideal for iteration, collections, queues, and loops
- Nesting possible: lists inside lists

🔢 Example:



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

      10

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

[18]: print(my_list)

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

➕ Nested List Example:



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

      3

[ ]:
```

🧰 List Methods:

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

📘 8. Dictionary – Key-Value Mapping

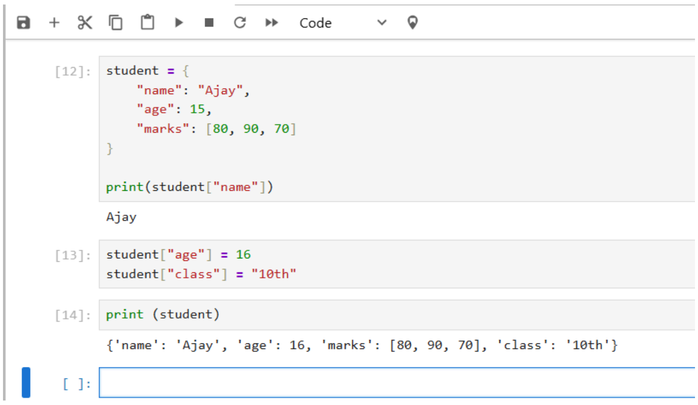
🔍 What is a Dictionary?

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

✅ Why Dictionaries are Useful:

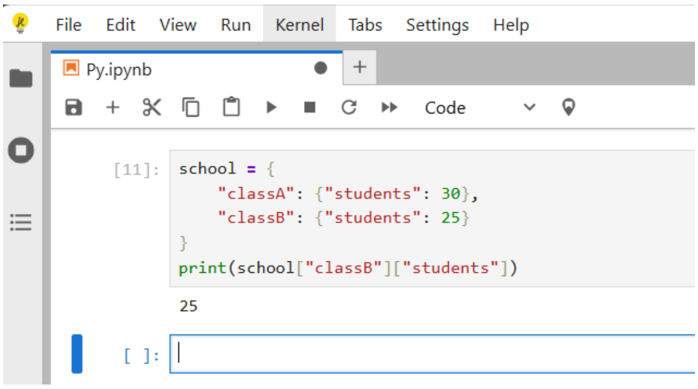
- Fast data retrieval
- Real-life mapping (student data, user profiles)
- Can hold any type of value, including lists & other dictionaries

🧪 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

[ ]:
```

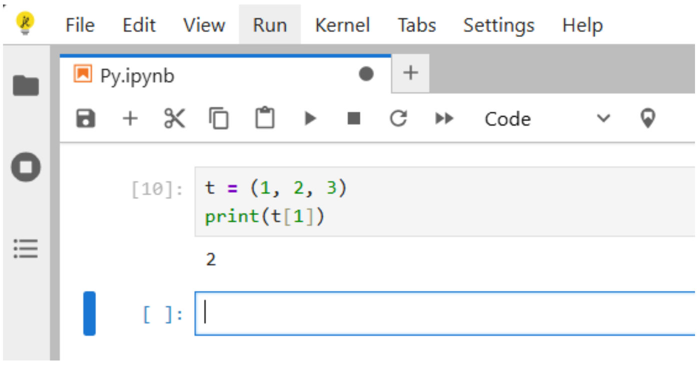
🧰 Dictionary Methods:

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

🧱 9. Tuples and Sets

🔶 Tuples – Immutable, Ordered

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

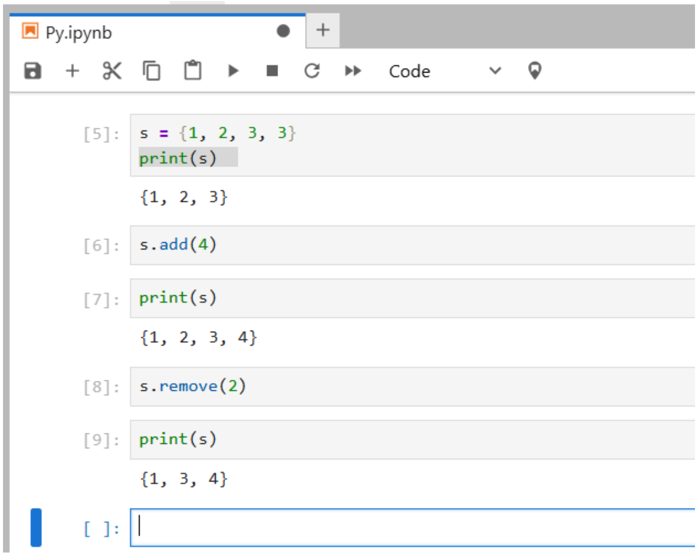


```
[10]: t = (1, 2, 3)
      print(t[1])

      2

[ ]:
```

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

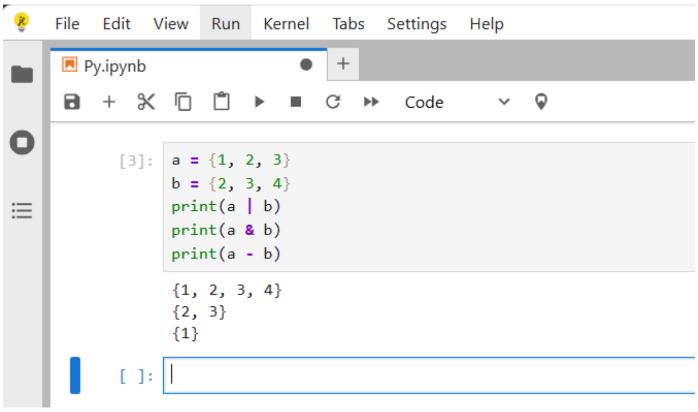
🔶 Sets – Unordered, Unique

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



```
[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}

[ ]:
```

🔁 Set Operations:

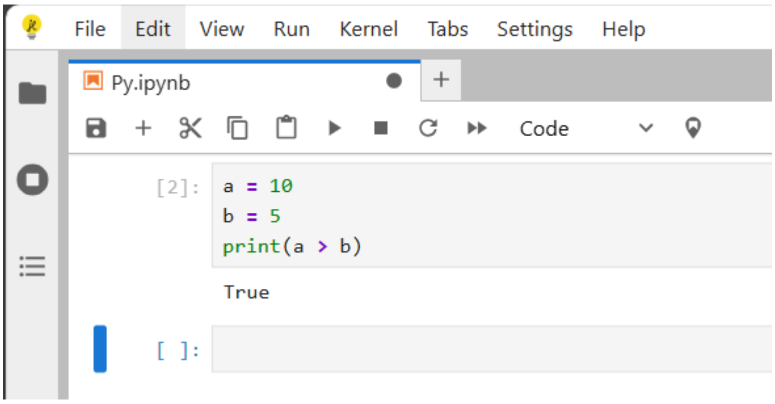


```
[3]: a = {1, 2, 3}
     b = {2, 3, 4}
     print(a | b)
     print(a & b)
     print(a - b)

     {1, 2, 3, 4}
     {2, 3}
     {1}

[ ]:
```

🔁 10. Booleans

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

✅ Example:



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

```
True
```

⚙️ 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**
