

Prof (Dr) Bishwajeet Kumar Pandey

Department of MCA

GL Bajaj Institute of Technology and
Management, Greater Noida, India

INTRODUCTION TO PYTHON



UNIT-1

Syllabus

1.1 Introduction to Python, Features, Setting up Python Environment

1.2 Python Syntax, Indentation, Comments, Writing & Executing Programs

1.3 Python Data Variables & Data Types

1.4 Operators in Python: Arithmetic, Relational, Logical, Assignment, Bitwise

1.5 id() and type() Functions, Coding Standards

1.6 Input & Output Functions: input(), print()

1.7 Control Structures: if, if-else, elif, Nested if

1.8 Iteration: for, while loops, break, continue, pass

What is Python?

Python is a **high-level, interpreted programming language** that is widely used for building applications, automating tasks, analyzing data, creating websites, developing AI models, and more.

It is known for its **simplicity, readability, and versatility**, making it one of the most popular programming languages in the world.

Difference Between Interpreter and Compiler

Feature	Interpreter	Compiler
How it works	Translates line by line and executes immediately.	Translates entire program into machine code first, then runs it.
Speed of Execution	Slower (translates while running).	Faster (once compiled, runs directly).
Error Detection	Shows errors immediately after each line.	Shows all errors after compiling the whole program.
Output	No separate file; runs directly.	Creates an executable file (e.g., .exe).
Memory Usage	Uses less memory; no need to store compiled code.	Uses more memory; stores compiled machine code.
Example Languages	Python, JavaScript, Ruby, PHP.	C, C++, Java, Go, Rust.
Best Use Case	Scripting, rapid testing, dynamic programs.	Large applications, performance-heavy software.



Key Features of Python

- **Easy to Learn and Read** – Python uses simple, English-like syntax, making it beginner-friendly.
- **Interpreted Language** – Code is executed line by line; no need to compile.
- **High-Level Language** – You focus on solving problems rather than dealing with hardware-level details.
- **Versatile and Cross-Platform** – Runs on Windows, macOS, Linux, and other systems.
- **Dynamically Typed** – No need to declare variable types explicitly.
- **Extensive Libraries** – Huge collection of libraries and frameworks for data science, web development, automation, AI, and more (e.g., NumPy, Pandas, Django, TensorFlow).
- **Open-Source** – Free to use and modify, supported by a large community.

+

•

○

What You Can Do with Python

• **Web Development** (Django, Flask, FastAPI)

• **Data Science & Machine Learning** (Pandas, NumPy, scikit-learn)

• **Artificial Intelligence & Deep Learning** (TensorFlow, PyTorch)

• **Automation & Scripting** (Automate tasks like file management)

• **Game Development** (Pygame)

• **Desktop GUI Apps** (Tkinter, PyQt)

• **Cybersecurity & Ethical Hacking**

• **Internet of Things (IoT)**

Comparison of Python with Java and C++

Feature	Python	C++	Java
Syntax	Simple, readable, like English; fewer lines of code.	Complex; uses lots of symbols and manual memory management.	More verbose than Python but simpler than C++; uses curly braces { }.
Execution	Interpreted (line-by-line). Slower than C++/Java.	Compiled to machine code. Very fast.	Compiled to bytecode, then run on JVM. Faster than Python, slower than C++.
Typing	Dynamically typed (no need to declare variable types).	Statically typed (must declare types).	Statically typed (must declare types).
Ease of Learning	Easiest for beginners.	Hard; steep learning curve.	Medium; easier than C++ but harder than Python.
Use Cases	AI, ML, Data Science, Web Apps, Automation, Scripting.	System programming, game engines, high-performance apps.	Enterprise software, mobile apps (Android), backend systems.
Speed	Slower (interpreted).	Very fast (compiled).	Fast (JVM optimized).
Memory Management	Automatic (garbage collection).	Manual (you must manage memory).	Automatic (garbage collection).
Community Support	Very large; many libraries.	Large but niche (game dev, embedded systems).	Large; popular for enterprise solutions.
Portability	Very high; runs everywhere.	Medium; requires recompilation for each platform.	Very high; JVM runs on all platforms.
Code Example (Print Hello)	<pre>print("Hello")</pre>	<pre>std::cout << "Hello";</pre>	<pre>System.out.println("Hello");</pre>

Docstrings

- Used to document functions, classes, and modules.
- Triple quotes (""" or ''') placed **inside** the function, class, or module.
- Can be **accessed at runtime** using `help()` or `.__doc__`


```
Command Prompt - python  x  +  v
>>> def greet(name):
...     """
...     This function takes a name as input and returns a greeting string.
...     """
...     return f"Hello, {name}!"
...
>>> greet(Bishwajeet)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Bishwajeet' is not defined
>>> name="Bishwajeet"
>>> greet(name)
'Hello, Bishwajeet!'
>>> print(greet.__doc__)

    This function takes a name as input and returns a greeting string.
>>> |
```


Comments

- **Used to explain code logic for developers.**
- Begins with # and is ignored by the Python interpreter.
- Meant **only for humans**
- **Not accessible** at runtime

```
>>> # This function returns a greeting message
>>> def greet(name):
...     return f"Hello, {name}!" # Format the greeting string
...
>>> greet("Bishwajeet")
'Hello, Bishwajeet!'
>>> |
```



Docstring versus Comments

Feature	Docstring	Comment
Syntax	<code>"""Triple quotes"""</code>	<code>#</code>
Purpose	Documentation	Code explanation
Location	Inside functions/classes/modules	Anywhere in code
Runtime Access	Yes (<code>. __doc__</code> , <code>help()</code>)	No
Follows PEP	Yes (PEP 257)	Yes (PEP 8 for line length)
Tools Usage	Used by IDEs, Sphinx, <code>help()</code>	Ignored by tools

Coding Standards in Python

- Python follows a set of best practices for writing clean, readable, and maintainable code — collectively known as **PEP 8** (Python Enhancement Proposal 8).
- **1. Indentation**
- **2. Variable & Function Naming**
- **3. Class Naming**
- **4. Constants**
- **5. Line Length**
- **6. Blank Lines**
- **7. Import Statements**
- **8. Avoid Unused Imports and Variables**
- **9. Use Docstrings**
- **10. Use `is / is not` for None**
- **11. Use Meaningful Names**

1. Indentation

- Use **4 spaces per indentation level** (not tabs).

```
def greet():
```

```
    print("Hello") # 4 spaces
```

2. Variable & Function Naming

Use **snake_case** for variable and function names.

```
user_name = "Alice"
```

```
def calculate_total(): ...
```

3. Class Naming

Use **CamelCase** for class names.

```
class StudentRecord:
```

```
    pass
```

4. Constants



USE **UPPERCASE** WITH
UNDERSCORES.



PI = 3.14



MAX_SIZE = 100

5. Line Length

Limit lines to **79 characters** for code, **72** for docstrings/Comments.

Docstrings Used to document functions, classes, and modules. Syntax: Triple quotes (""" or ''') placed **inside** the function, class, or module.

2. Comments Used to explain code logic for developers. Syntax: Begins with # and is ignored by the Python interpreter.

6. Blank Lines

Use	Import	Import	Import	Import
<p>Use blank lines to separate:</p> <ul style="list-style-type: none">• Functions and classes• Logical sections of code	<pre>import os</pre>	<pre>import sys</pre>	<pre>import numpy as np</pre>	<pre>import mymodule</pre>

7. Import Statements

- Imports should be:
 - Standard libraries
 - Third-party libraries
 - Local modules

8. Avoid Unused Imports and Variables

```
# BAD
```

```
import math
```

```
x = 10
```

9. Use Docstrings

Document functions, classes,
and modules using triple quotes:

```
def add(a, b):
```

```
    """Return the sum of a and b."""
```

```
    return a + b
```

10. Use `is` `/ is not` for `None`



if value is `None`:



...

11. Use Meaningful Names



BAD



`a = 10`



GOOD



`student_count = 10`

Difference between = and ==

A single equal sign (=) is used to set variables to a new value.

The double equals (==) is used to compare one variable to another to see if they're equal.

What is PEP?

A PEP is a Python Enhancement Proposal.

A PEP is a document that describes a way in which Python can be made better. Some don't affect the language at all (such as the style guide), whereas others may add or remove a feature from Python. Others describe processes such as how to submit a PEP or hand an existing PEP to another developer.

```
Microsoft Windows [Version 10.0.26100.4652]  
(c) Microsoft Corporation. All rights reserved.
```

```
C:\Users\pushp>python  
Python 3.12.10 (tags/v3.12.10:0cc8128, Apr 8 2025, 12:21:36) [MSC v.1943 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print "Hello, World!"  
File "<stdin>", line 1  
    print "Hello, World!"  
    ^^^^^^^^^^^^^^^^^^^^^  
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?  
>>> print ("Hello, World!")  
Hello, World!  
>>>
```

Print Hello World

- `print "Hello, World!"`
- That syntax is from **Python 2**, but you're using **Python 3.12**, which requires **parentheses** with `print`.
- `print("Hello, World!")`

```
>>> import sys
>>> sys.ps1="Bishwajeet>>"
Bishwajeet>>print ("Hello World!")
Hello World!
Bishwajeet>>
```

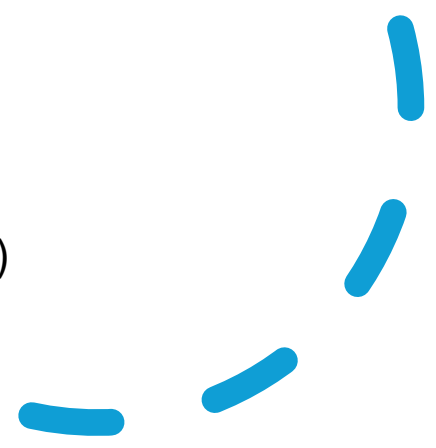
Print Hello
World on your
Personalize
Prompt

- To change default Prompt, we can customize the `sys.ps1` and `sys.ps2` variables:
- `>>> import sys`
- `>>> sys.ps1 = "Bishwajeet>> "`

```
Bishwajeet>>def cls():  
...     import os  
...     os.system('cls' if os.name == 'nt' else 'clear')  
... |
```

Clear Screen

- Instead of writing `os.system` every time, define a quick function:
- `def cls():`
- `import os`
- `os.system('cls' if os.name == 'nt' else 'clear')`
- Then call `cls()` any time to clear.



Simple Mathematics in Python

```
Command Prompt - python

Bishwajeet>>5+1
6
Bishwajeet>>5-1
4
Bishwajeet>>5*4
20
Bishwajeet>>%/1
File "<stdin>", line 1
    %/1
    ^
SyntaxError: invalid syntax
Bishwajeet>>5/1
5.0
```

Print Random Number in Python

- The `random.random()` function in Python is used to **generate a random floating-point number** between `0.0` and `1.0` (excluding `1.0`).
- `import random`
- `random.random()`

```
Command Prompt - python

Bishwajeet>>import random
Bishwajeet>>random.random()
0.16110801069186553
Bishwajeet>>random.random()
0.04312120930480978
Bishwajeet>>random.random()
0.10710197078413208
Bishwajeet>>random.random()
0.00659885623666312
Bishwajeet>>|
```

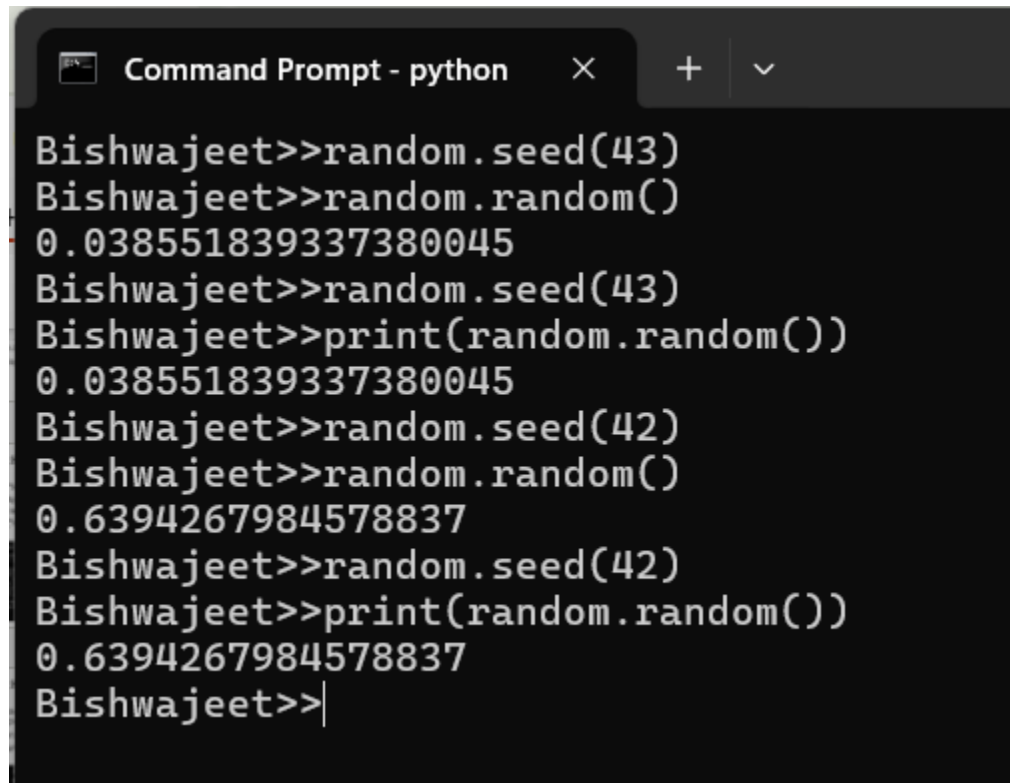

Print Random Number in Python

- If we want a random float in a different range (e.g., 1 to 10):

```
Bishwajeet>>random.uniform(1, 10)
7.55046187282845
Bishwajeet>>random.uniform(1, 10)
8.591053252029491
Bishwajeet>>random.uniform(1, 10)
5.986724476134066
Bishwajeet>>random.uniform(1, 10)
8.36014231967378
Bishwajeet>>|
```

Print Random Number in Python

- Use Seed For repeatability (same results every time):

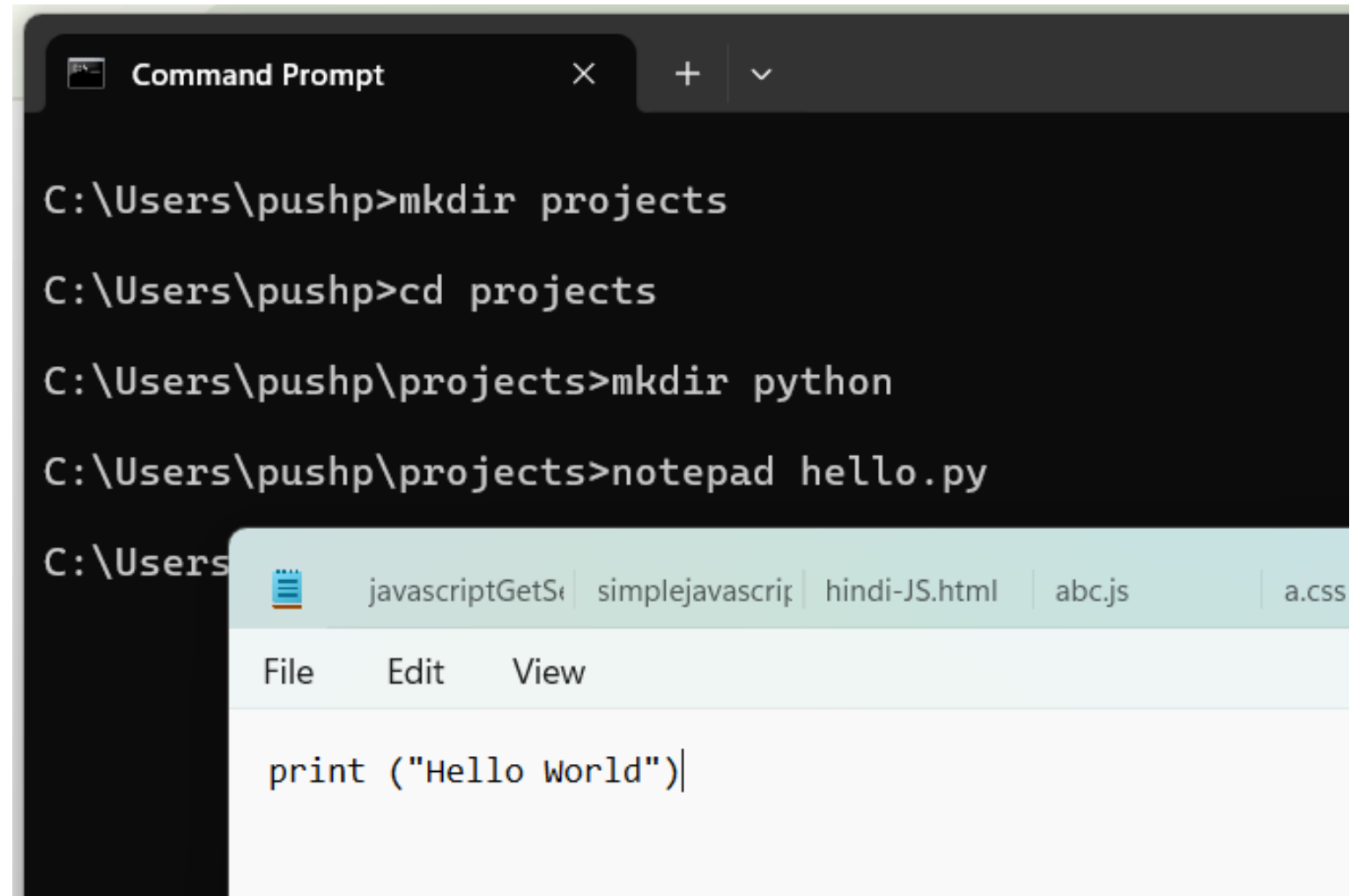


```
Command Prompt - python
Bishwajeet>>random.seed(43)
Bishwajeet>>random.random()
0.038551839337380045
Bishwajeet>>random.seed(43)
Bishwajeet>>print(random.random())
0.038551839337380045
Bishwajeet>>random.seed(42)
Bishwajeet>>random.random()
0.6394267984578837
Bishwajeet>>random.seed(42)
Bishwajeet>>print(random.random())
0.6394267984578837
Bishwajeet>>
```

Exercise

1. Through the command line, create a new folder called "projects" in your home folder. Then, change into that directory and create another folder called "python"
2. In our text editor, create a new file called hello.py in python directory (the one you made in Exercise 1). Type `print hello World` into it then save and close it.
3. See how the file hello.py is?
4. Also run hello.py

Solution



The screenshot shows a Windows Command Prompt window with the following commands and their outputs:

```
C:\Users\pushp>mkdir projects
C:\Users\pushp>cd projects
C:\Users\pushp\projects>mkdir python
C:\Users\pushp\projects>notepad hello.py
```

Below the Command Prompt, a Notepad window is open, showing the file explorer at the top with tabs for 'javascriptGetS...', 'simplejavascrip...', 'hindi-JS.html', 'abc.js', and 'a.css'. The menu bar includes 'File', 'Edit', and 'View'. The main text area contains the Python code:

```
print ("Hello world")
```

Solution

```
Bishwajeet prompt "Bishwajeet>"
```

```
"Bishwajeet>" dir
```

```
Volume in drive C is OS
```

```
Volume Serial Number is AA52-6D10
```

```
Directory of C:\Users\pushp\projects
```

```
07-08-2025  11:18    <DIR>          .
07-08-2025  11:17    <DIR>          ..
07-08-2025  11:18                21 hello.py
07-08-2025  11:17    <DIR>          python
                        1 File(s)                21 bytes
                        3 Dir(s)  105,889,648,640 bytes free
```

```
"Bishwajeet>" python hello.py
```

```
Hello World
```

```
"Bishwajeet>" |
```

Variables in Python

- Variable can hold many different kinds of information.
- In Python, variables are **dynamically typed**, meaning you don't need to declare their type explicitly.
- Python figures out the type based on the value assigned.
- In python, equal sign doesn't means that two things should be equal. It is used as one of ways to assign a value to a variable.

Naming Variables in Python

Rule	Description
Must start with a letter (a–z, A–Z) or underscore _	name, _value
✗ Cannot start with a number	2name ✗
Can contain letters , digits , and underscores	score1, user_name
✗ No spaces , symbols like @, –, !, etc.	user-name ✗
✗ Cannot be a Python keyword	class, for, if ✗

Naming Variables in Python

	Example
Use descriptive names	price, student_name
Use snake_case (recommended in Python)	total_marks
Use lowercase for variables	counter = 0
Constants in UPPERCASE	PI = 3.14
Avoid single letters unless in loops	i, j in for i in range(5)

```
Command Prompt - python  ×  +  ▾  
>>> import keyword  
>>> print(keyword.kwlist)  
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue',  
, 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', '  
in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with',  
, 'yield']  
>>> |
```

Naming Variables in Python

- To check all Python keywords that **cannot** be used as variable names:
- `import keyword`
- `print(keyword.kwlist)`

Types of Variables

Type	Name	Description	Example
int	Integer	Whole numbers	<code>x = 10</code>
float	Floating point	Decimal numbers	<code>pi = 3.14</code>
str	String	Sequence of characters (text)	<code>name = "Alice"</code>
bool	Boolean	Logical value: True or False	<code>is_active = True</code>
list	List	Ordered, changeable collection	<code>fruits = ["apple", "banana"]</code>
tuple	Tuple	Ordered, unchangeable collection	<code>point = (5, 10)</code>
dict	Dictionary	Key-value pairs	<code>student = {"name": "Bob"}</code>
set	Set	Unordered, unique items	<code>nums = {1, 2, 3}</code>
NoneType	None	No value or null	<code>result = None</code>

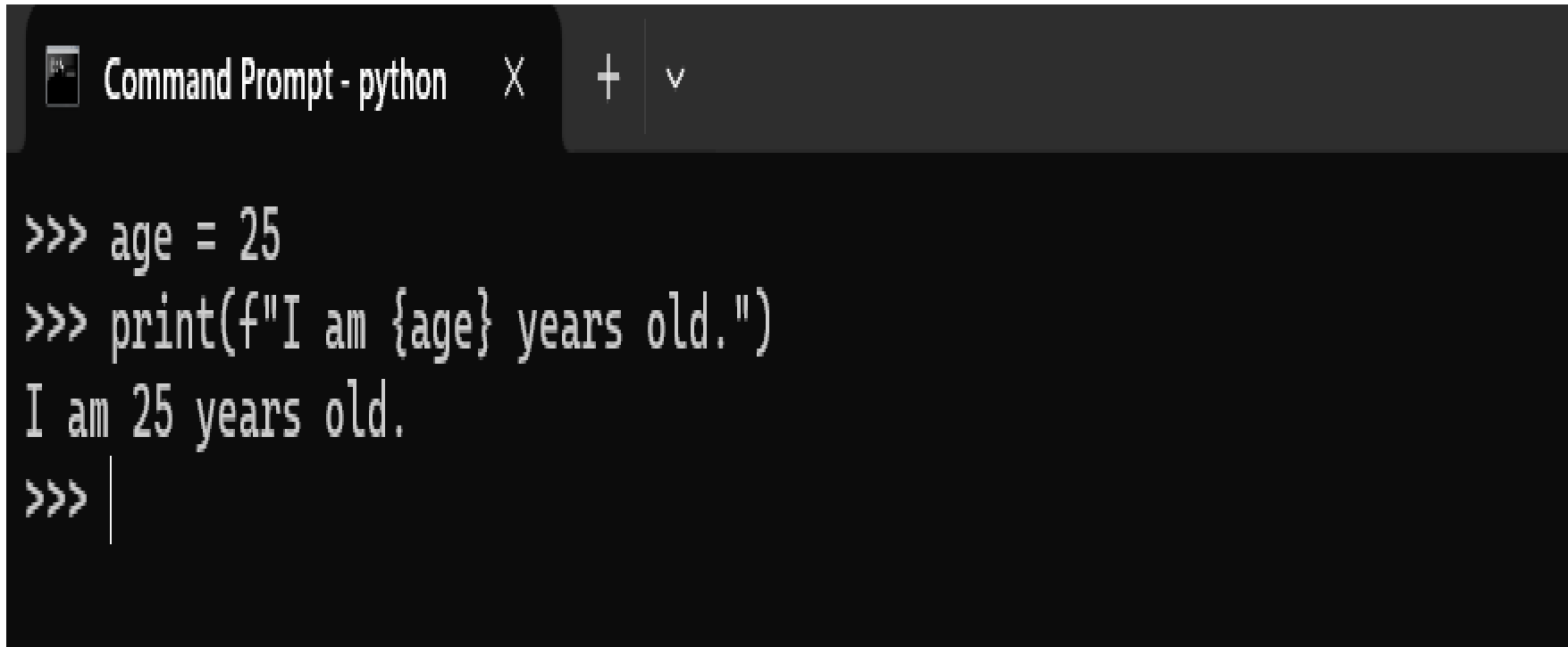
F-string

- The `f` in `f"Hello, {name}!"` is short for a **formatted string literal**, commonly called an **f-string** in Python.
- An **f-string** is a way to **embed expressions inside string literals** using curly braces `{}`. It was introduced in **Python 3.6**.
- The `f` before the string tells Python to **evaluate** any expressions inside `{}`. We can use variables, expressions, even function calls inside the `{}`.

```
>>> name="Bishwajeet"
>>> greeting = f"Hello, {name}"
>>> print(greeting)
Hello, Bishwajeet
>>> |
```

F-string

- Simple variable insertion:

A screenshot of a Windows Command Prompt window titled "Command Prompt - python". The window has a dark background and a light-colored text. The text shows a Python interactive session where a variable 'age' is assigned the value 25, and then an f-string is used to print "I am {age} years old.", which results in "I am 25 years old." being printed. The prompt is currently at the start of a new line.

```
>>> age = 25
>>> print(f"I am {age} years old.")
I am 25 years old.
>>> |
```

F-string

Inline calculations:

```
Command Prompt - python  ×  +  ∨  
  
>>> x = 5  
>>> y = 3  
>>> print(f"Sum is {x + y}")  
Sum is 8  
>>> |
```

F-string

Function call inside:

```
>>> def get_name():  
...     return "Bishwajeet"  
...  
>>> print(f"Hello, {get_name()}!")  
Hello, Bishwajeet!  
>>> |
```

Type Function

- In Python, the built-in **type()** function is used to check the **data type** of any variable.
- `type(variable)`

```
Command Prompt - python
>>> x = 10
>>> print(type(x))
<class 'int'>
>>> name = "Alice"
>>> print(type(name))
<class 'str'>
>>> pi = 3.14
>>> print(type(pi))
<class 'float'>
>>> is_valid = True
>>> print(type(is_valid))
<class 'bool'>
>>> data = [1, 2, 3]
>>> print(type(data))
<class 'list'>
>>> user = {"id": 1}
>>> print(type(user))
<class 'dict'>
>>> |
```


Type Function

- Combining type when doing math.

```
Command Prompt - python  ×  +  ▾

>>> type(1+1)
<class 'int'>
>>> type(1.0+1)
<class 'float'>
>>> type(3+2j + 1)
<class 'complex'>
>>> type (True+1)
<class 'int'>
>>> type(False + 2.0)
<class 'float'>
>>> type(True*False)
<class 'int'>
>>> type(int(True))
<class 'int'>
>>> typle(float(false))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'typle' is not defined. Did you mean: 'tuple'?
>>> type(float(false))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'false' is not defined. Did you mean: 'False'?
>>> type(float(False))
<class 'float'>
>>> complex(1,5)
(1+5j)
>>> |
```

Integer and Float divide

```
>>> 1/2
0.5
>>> 1/2.0
0.5
>>> 1/float(2)
0.5
>>> 1/int(2)
0.5
>>> 1%2
1
>>> 1//int(2)
0
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> |
```

Operators in Python

Operators in Python, grouped by category

1. Arithmetic Operators

2. Assignment Operators

3. Comparison Operators

4. Logical Operators

5. Identity Operators

6. Membership Operators

7. Bitwise Operators

1. Arithmetic Operators

Operator	Description	Example	Result
+	Addition	5 + 2	7
-	Subtraction	5 - 2	3
*	Multiplication	5 * 2	10
/	Division (float)	5 / 2	2.5
//	Floor Division	5 // 2	2
%	Modulus (remainder)	5 % 2	1
**	Exponentiation	2 ** 3	8

2. Assignment Operators

Operator	Example	Meaning
=	x = 5	Assign 5 to x
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
//=	x //= 3	x = x // 3
%=	x %= 3	x = x % 3
**=	x **= 3	x = x ** 3

3. Comparison Operators

Operator	Description	Example	Result
==	Equal to	5 == 5	True
!=	Not equal to	5 != 3	True
>	Greater than	5 > 3	True
<	Less than	5 < 3	False
>=	Greater or equal	5 >= 5	True
<=	Less or equal	5 <= 4	False

4. Logical Operators

Operator	Description	Example	Result
and	True if both are True	True and False	False
or	True if at least one is True	True or False	True
not	Inverts result	not True	False

Check whether two variables refer to the same object in memory.

5. Identity Operators

Operator	Description	Example
<code>is</code>	Same object	<code>x is y</code>
<code>is not</code>	Not same object	<code>x is not y</code>

Operator	Description	Example
<code>in</code>	Value exists	<code>'a' in 'apple'</code>
<code>not in</code>	Value doesn't exist	<code>'x' not in 'apple'</code>

- Check if a value exists in a sequence (like list, string).

6. Membership Operators

Operator	Description	Example
&	AND	5 & 3 → 1
∨	∨	OR
^	XOR	5 ^ 3 → 6
~	NOT (invert bits)	~5 → -6
<<	Left shift	5 << 1 → 10
>>	Right shift	5 >> 1 → 2

- Operate at the binary level.

7. Bitwise Operators

```
Command Prompt - python  X + v
>>> x=10;
>>> print(id(x))
140720047999704
>>> y=7
>>> print(id(y))
140720047999608
>>> |
```

id() Function in Python

Interning in Python

- Interning is an optimization technique where Python stores only one copy of a particular immutable object (like certain integers or strings) and reuses it wherever possible.
- Integer Interning:
- Python **pre-allocates integers** in the range **-5 to 256**. These integers are reused throughout the program.

```
Command Prompt - python

>>> x=100
>>> y=100
>>> id(x)
140713029686808
>>> id(y)
140713029686808
>>> x=300
>>> y=300
>>> id(x)
2837098726032
>>> id(y)
2837098741328
>>>
```

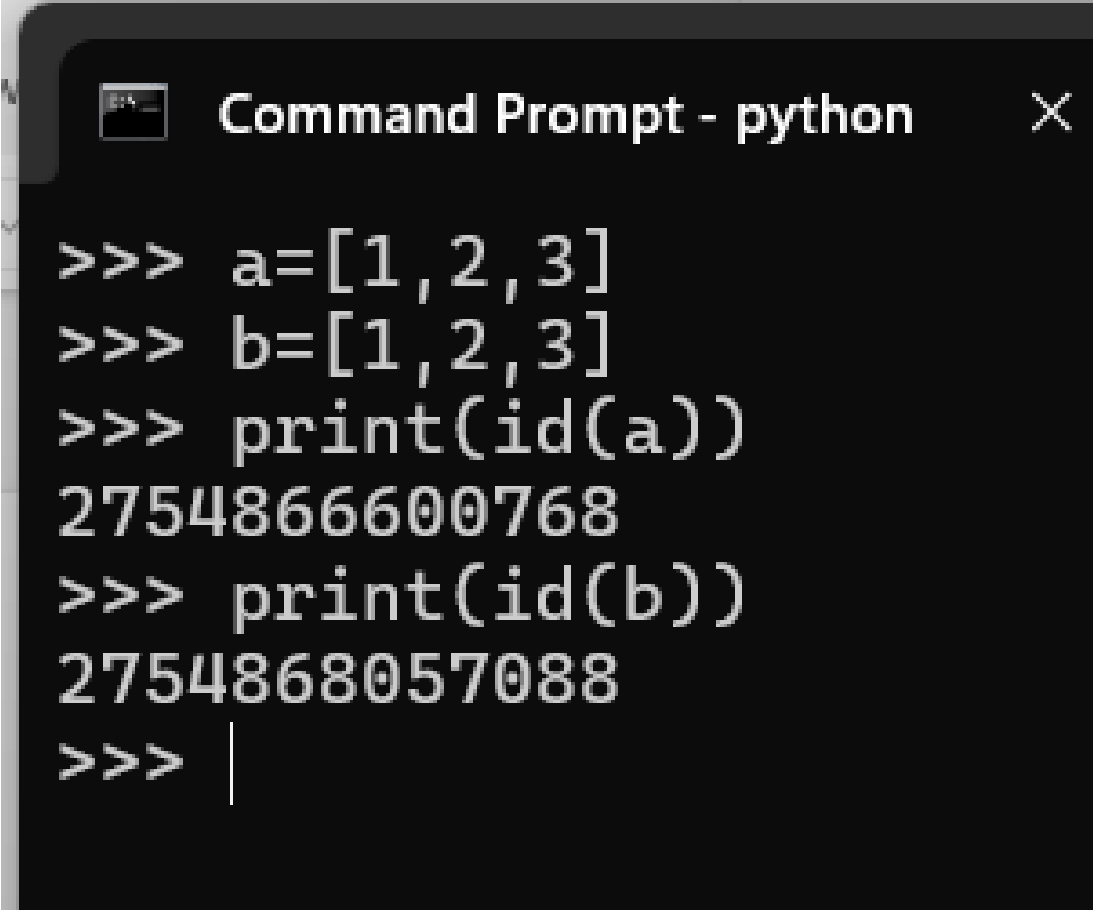
Interning in Python

- String Interning:
- Short strings and identifiers (like variable names) are often interned automatically.
- But dynamically created strings may not be interned.
- You can force interning of strings manually.

```
Command Prompt - python
>>> s1="hello"
>>> id(s1)
2837098856640
>>> s2="hello"
>>> id(s2)
2837098856640
>>> s3=''.join(['hel','lo'])
>>> id(s3)
2837098862976
>>> print(s1==s3)
True
>>> print(s1 is s3)
False
>>> print(s1 is s2)
True
>>> import sys
>>> s3 = sys.intern(s3)
>>> print(s1 is s3) # Now True
True
>>> |
```

id() Function in Python

- `a = [1, 2, 3]`
- `b = [1, 2, 3]`
- `print(id(a))` # different from `id(b)`
- `print(id(b))`
- Here, `a` and `b` look the same, but they are **different objects**, so `id(a) != id(b)`.



```
Command Prompt - python

>>> a=[1,2,3]
>>> b=[1,2,3]
>>> print(id(a))
2754866600768
>>> print(id(b))
2754868057088
>>> |
```

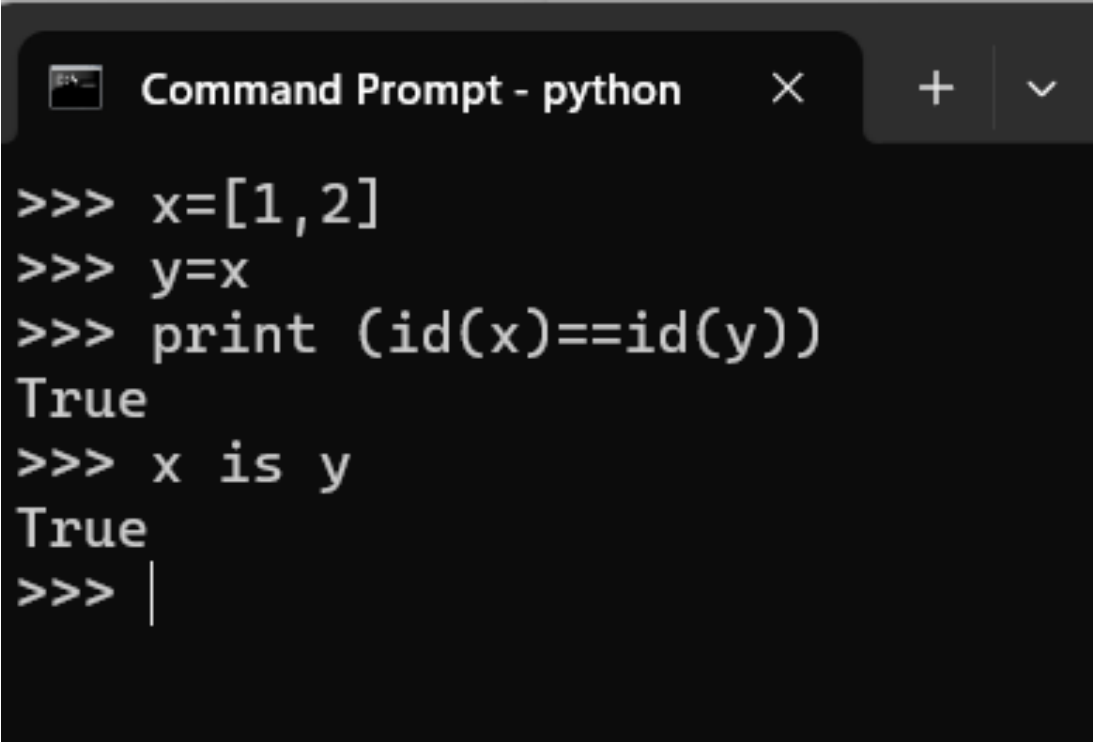
id() Function in Python

- `id()` vs `is`:
- `id(a) == id(b)` → checks if two variables point to the same object
- `a is b` → is the same as `id(a) == id(b)`

```
Command Prompt - python
>>> x=[1,2]
>>> y=x
>>> print (id(x)==id(y))
True
>>> x is y
True
>>> |
```

id() Function in Python

- `x = [1, 2]`
- `y = x`
- `print(id(x) == id(y))` # True — both point to same list

A screenshot of a Windows Command Prompt window titled "Command Prompt - python". The window has a dark background and a light-colored text. The code being executed is as follows:

```
>>> x=[1,2]
>>> y=x
>>> print (id(x)==id(y))
True
>>> x is y
True
>>> |
```

The output shows that both `id(x) == id(y)` and `x is y` return `True`, confirming that both variables point to the same list object in memory.

Result of $1 + 2 * 3$

- The answer is 7 because multiplication comes before addition.

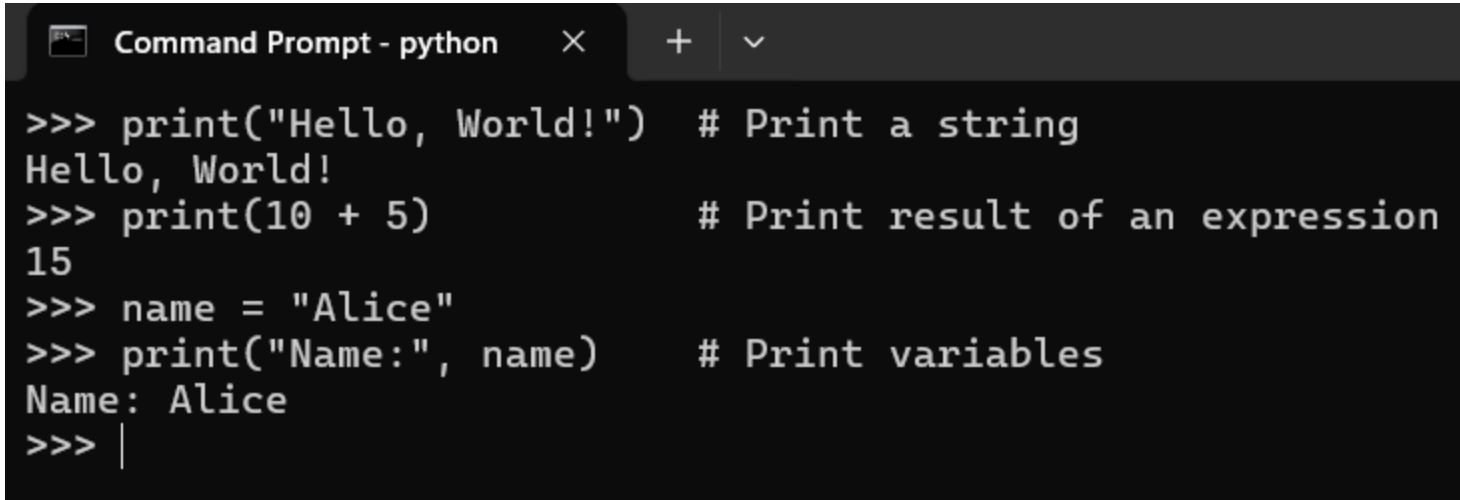
✓ Python Order of Operations (PEMDAS)

Order	Operation	Example
1	Parentheses <code>()</code>	<code>(2 + 3) * 4 = 20</code>
2	Exponents <code>**</code>	<code>2 ** 3 = 8</code>
3	Multiplication <code>*</code> , Division <code>/</code> , Floor Division <code>//</code> , Modulus <code>%</code>	<code>10 / 2 * 3 = 15.0</code>
4	Addition <code>+</code> , Subtraction <code>-</code>	<code>5 + 3 - 1 = 7</code>

✓ Operations on the same level are evaluated left to right, except for exponents (`**`), which are evaluated right to left.

Printing on Screen in Python

To **print output** to the screen, you use the `print()` function.

A screenshot of a Windows Command Prompt window titled "Command Prompt - python". The window has a dark background with white text. It shows a series of Python commands and their outputs. The first command is `>>> print("Hello, World!")` with a comment `# Print a string`, followed by the output `Hello, World!`. The second command is `>>> print(10 + 5)` with a comment `# Print result of an expression`, followed by the output `15`. The third command is `>>> name = "Alice"`. The fourth command is `>>> print("Name:", name)` with a comment `# Print variables`, followed by the output `Name: Alice`. The prompt `>>>` is followed by a vertical bar `|` indicating the cursor is at the end of the line.

```
Command Prompt - python  X  +  v

>>> print("Hello, World!")  # Print a string
Hello, World!
>>> print(10 + 5)          # Print result of an expression
15
>>> name = "Alice"
>>> print("Name:", name)   # Print variables
Name: Alice
>>> |
```

Printing on Screen in Python

You can use end and sep parameters:

```
>>> print("Hello", "World", sep="---")    # Output: Hello---World
Hello---World
>>> print("Hello", end=" ")               # Stay on same line
Hello >>> print("again")
again
>>> |
```

Reading Data from Keyboard in Python

- To **get input** from the user, use the `input()` function.

```
>>> name=input("Enter your name:")  
Enter your name:Bishwajeet  
>>> |
```

```
Command Prompt - python  ×  +  ∨  
>>> age = int(input("Enter your age: "))      # Convert input to integer  
Enter your age: 39  
>>> height = float(input("Enter your height: ")) # Convert to float  
Enter your height: 167  
>>> |
```

Reading Data from Keyboard in Python

- All input from `input()` is **treated as a string** by default. We need to convert it:

Control Structures in Python

if-else

elif

Nested if

Iteration Control structures

break, continue, pass

IF-ELSE

The if-else statement is used to make decisions. If the condition is true, one block of code runs, otherwise the else block executes.

```
x = 10
if x > 5:
    print("x is greater than 5")
else:
    print("x is 5 or less")
```

OUTPUT: x is greater than 5

ELIF

The elif (short for *else if*) is used when there are **multiple conditions** to check. It prevents deep nesting of multiple if statements.

```
marks = 75
```

```
if marks >= 90:  
    print("Grade A")  
elif marks >= 75:  
    print("Grade B")  
elif marks >= 50:  
    print("Grade C")  
else:  
    print("Fail")
```

OUTPUT: GRADE B

Nested IF

A nested if means an if statement inside another if. It is used when one condition depends on another.

```
x = 20
if x > 10:
    if x % 2 == 0:
        print("x is greater than 10 and even")
    else:
        print("x is greater than 10 and odd")
```

- OUTPUT: x is greater than 10 and even

Iteration Control Structures (for and while)

```
# for loop
for i in range(1, 6):
    print("For loop:", i)
```

OUTPUT

```
For loop: 1
For loop: 2
For loop: 3
For loop: 4
For loop: 5
```

```
# while loop
count = 1
while count <= 5:
    print("While loop:", count)
    count += 1
```

OUTPUT

```
While loop: 1
While loop: 2
While loop: 3
While loop: 4
While loop: 5
```

break, continue, pass

- **break** → exits the loop immediately.
- **continue** → skips the current iteration and moves to the next.
- **pass** → does nothing; used as a placeholder.

break

- # break
 - for i in range(1, 10):
 - if i == 5:
 - break
 - print("Break example:", i)
- Break example: 1
 - Break example: 2
 - Break example: 3
 - Break example: 4

continue

- # continue
 - for i in range(1, 6):
 - if i == 3:
 - continue
 - print("Continue example:", i)
- Continue example: 1
 - Continue example: 2
 - Continue example: 4
 - Continue example: 5

pass

```
# pass
for i in range(1, 4):
    pass # placeholder
print("Pass example: loop
executed but did nothing")
```

- Pass example: loop executed but did nothing

Why use pass?

- **Placeholder for future code**

- When you are designing a program but haven't written the logic yet.

```
def my_function():  
    pass # logic will be added later
```

- **Empty loops**

- When you need a loop but don't want it to do anything right now.

```
for i in range(5):  
    pass
```

Why use pass?

- **Empty class or function**

- If you define a class or function but don't want to implement it yet.

```
class MyClass:  
    pass
```

- **Empty loops**

- When you need a loop but don't want it to do anything right now.

```
for i in range(5):  
    pass
```