



KP-IT Board AI (ML/DL) Certification

Trainer: Fawad Bahadur

Introduction to Python

What Is Python?

- ✓ Python is a high-level, interpreted, general-purpose programming language.
- ✓ Known for its **clear syntax**, readability, and simplicity.
- ✓ Used by **beginners** and **experts** alike due to its **versatility**.



Why Learn Python?

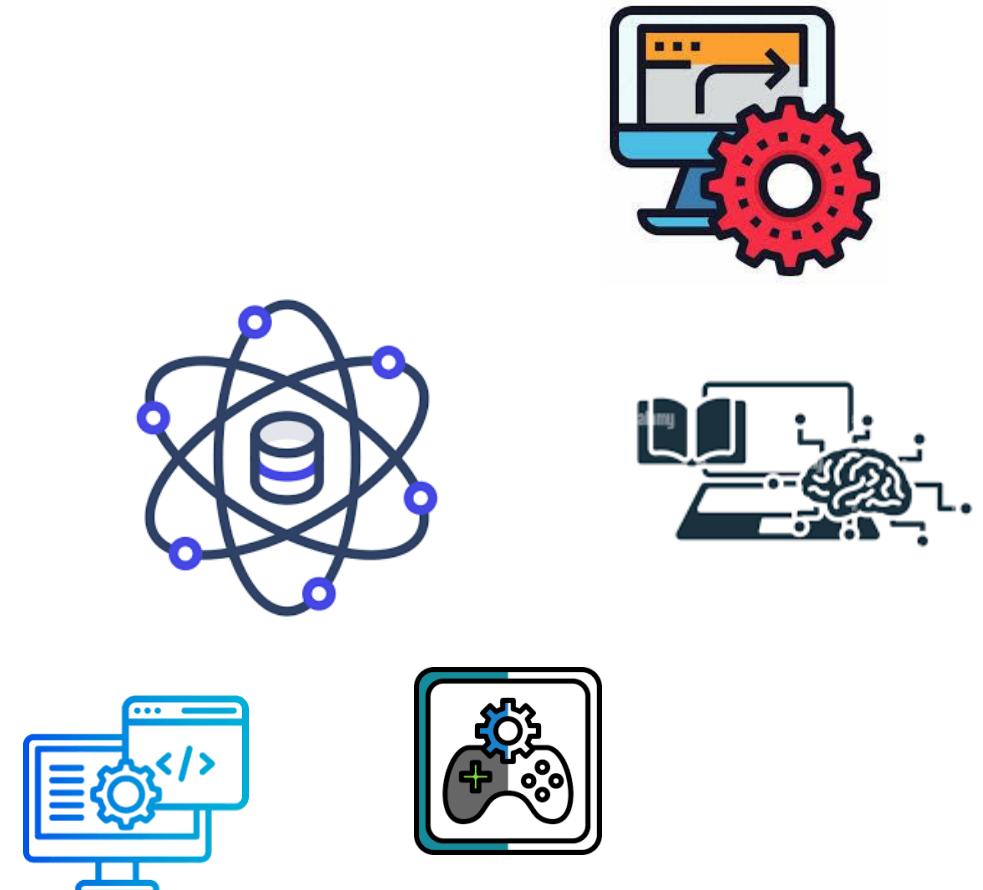
- Beginner-friendly yet powerful for complex applications.
- Widely adopted by Google, NASA, Netflix, and more.



Conti...

Key Features

- **Easy to Learn & Write** - Minimalistic syntax, resembles English.
- **Rich Standard Library** - Pre-built modules for faster development.
- **Strong Community Support** - Active forums, tutorials, and open-source contributions.
- **Versatile** - Used in:
 - ✓ Web Development (Django, Flask)
 - ✓ Data Science & AI (Pandas, TensorFlow)
 - ✓ Automation & Scripting
 - ✓ Game Development (Pygame)



Key characteristics



High-Level

Simple, abstracted coding



Object-Oriented

Classes, objects support



Interpreted

Runs without compilation



General-Purpose

Versatile, multi-domain use

Conti...



Extensible

Integrates external code



Vast Ecosystem

Rich libraries, tools



Dynamically Typed

Flexible variable types



Cross-Platform

Runs on all OS



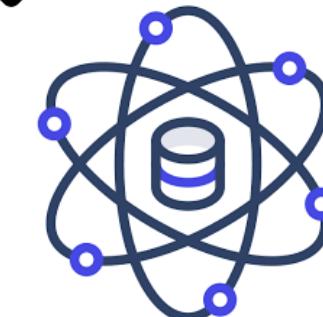
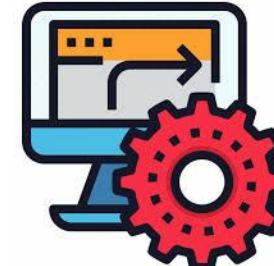
Large Community

Active, supportive network

Python Programming

Applications

- Web Development
- Data Science
- Data Analytics
- Machine Learning & AI
- Automation and Scripting
- Software Development and Prototyping
- Desktop GUI applications
- Scientific and Numeric Computing
- Game Development
- Education and Research



Download & Install Python

1

Download Python

Visit python.org/downloads

Choose the latest stable version (e.g., Python 3.12).

2

Run the Installer

Windows/macOS: Double-click the downloaded .exe (Windows) or .pkg (macOS).

3

Setup Options (Windows/macOS)

Check "Add Python to PATH" (Critical for command-line use).

Optional: Install pip (Python's package manager).



4

Verify Installation

Open Command Prompt/Terminal and type:
`python --version`

Install Anaconda for Python

Why Anaconda?

- Includes Python, Jupyter Notebook, and key libraries.
- Simplifies setup for immediate coding.

1. Download Anaconda

- Visit anaconda.com/download
- Choose Python 3.x for your OS (Windows/macOS).

3. Key Installation Options

- **Add Anaconda to PATH** (Recommended ‘easy CLI access’).
- Register Anaconda as default Python (Optional but useful).



2. Run the Installer

- Windows: Double-click .exe → Follow prompts.
- macOS: Open .pkg → Drag to Applications.

4. Verify Installation

- Open Command Prompt/Terminal, run:
- `conda --version`

Set Up Jupyter Notebook

1

Open Anaconda Navigator

Launch from your system after installation.

2

Start Jupyter Notebook

In Navigator, click the Jupyter Notebook icon.

3

Create a New Notebook

Opens in your web browser.

Click New > Python 3 in top-right.



4

Start Coding

Write Python code in notebook cells.

Press Shift + Enter to execute.



KP-IT Board AI (ML/DL) Certification

Trainer: Fawad Bahadur

Variables in Python

What is a Variable?

- A named container to store data in memory.
- Holds values that can be changed during program execution.
- No need to declare type

Key Rules

Names are case-sensitive ($x \neq X$).

Can include letters, digits, _ (no spaces).

Cannot start with a digit.

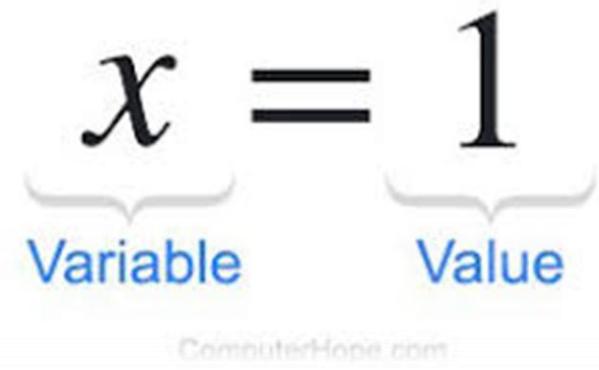
Avoid Python keywords (if, for, etc.).

Syntax

```
variable_name = value
```

Example

```
age = 25  
name = "Alice"
```



Name of your Variable

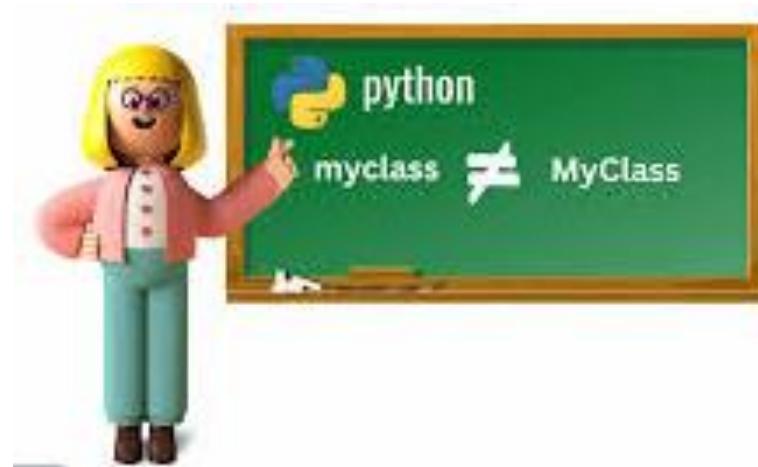
num = 5

Your Data (Integer DataType)

Trainer: Fawad Bahadur Marwat

Python Syntax & Structure

Case-sensitive



Indentation matters (no braces)

Comments: # single-line, "" multi-line ""

Statement ends by newline (no semicolon needed)

```
def greet():
    print("Hello!")
    print("Welcome to Python.")
```

A screenshot of a Python code editor window. The window has a blue header bar with three colored dots (red, yellow, green) on the left. The main area contains Python code: a function definition named 'greet' that prints two lines of text: "Hello!" and "Welcome to Python."

Data Types

What is Data type?

Building Blocks of Python Programs

1. Numeric Types

- *Int, Float, Complex*

2. Sequence Types

- *Str, list, tuple*

3. Mapping Types

- *dict*

4. Set Types

- *Set*

5. Boolean Types

- *bool*

6. None Types

- *None*



Integers in Python



What is a Integers?

- A whole number (positive, negative, or zero) without decimals.
- Used for counting, indexing, and mathematical operations.



Key Properties

Immutable: Cannot be changed after creation (a new object is created on modification).

Unlimited Size: Python handles large numbers seamlessly.



```
variable_name = integer_value
```



Example

```
age = 30
```

```
temperature = -10
```

```
count = 0
```

Floating-Point Numbers (Floats) in Python



What is a Floats?

- Represents real numbers (with decimal points or scientific notation).
- Used for measurements, scientific calculations, and fractional values.



Key Properties

Precision: Limited by hardware (64-bit double-precision).

Immutable: Like integers, floats are immutable objects.



Syntax

```
variable_name = float_value
```



Example

```
pi = 3.14159
```

```
temperature = -12.5
```

```
scientific_notation =
```

```
2.5e3 # 2500.0
```

Strings in Python



What is a String?

- A sequence of characters enclosed in quotes (' ' or " ").
- Used to represent text data in Python.



Syntax

```
string_name = "text" # or 'text'
```



Key Properties

Immutable: Cannot be changed after creation.

Indexed: Access characters using indices (starts at 0).

Iterable: Can loop through characters.



Example

```
name = "Alice"  
greeting = 'Hello,  
World!'  
multiline = """This is a  
multi-line string"""
```

Boolean (bool) in Python



Definition

Represent True or False values.



Key Uses:

- Control flow in if and while statements.
- Result of logical operations (e.g., $5 > 3 \rightarrow \text{True}$).
- Represent states (e.g., on/off, open/closed).

Conversions:

`bool(0) → False,`

`bool(1) → True.`



Example

```
is_logged_in = True  
if is_logged_in:  
    print("User is logged in")  
# Output: User is logged in
```



True



False



Working with boolean data type in python

List in Python

Definition

Ordered, mutable collections of items.

Key Features

- Can store mixed data types (e.g., integers, strings, lists).
- Indexed from 0.
- Supports operations: `.append()`, `.remove()`, `.sort()`.

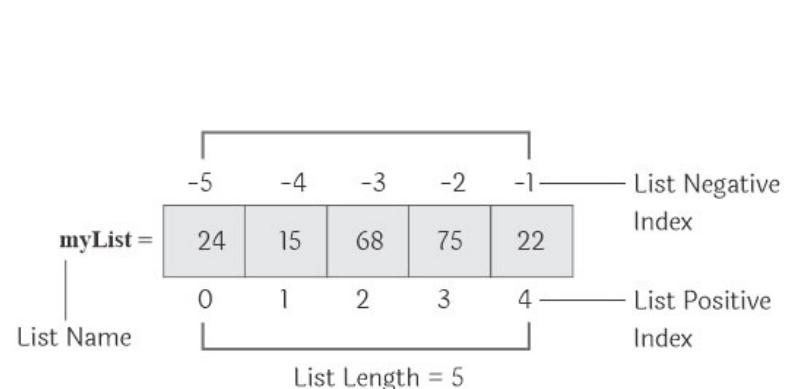
Use Cases

Store sequences (e.g., names, numbers, objects).



Example

```
fruits = ["apple", "banana", "cherry"]  
fruits.append("orange")  
print(fruits)  
# Output: ['apple', 'banana', 'cherry', 'orange']
```



Tuple in Python

Definition

Ordered, immutable collections of items.

Key Features

- Values cannot be changed after creation.
- Indexed from 0, like lists.
- Can store mixed data types (e.g., integers, strings, tuples).

Use Cases

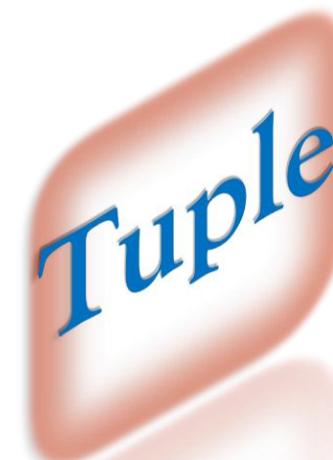
Fixed data sets (e.g., coordinates, color values).

Unpacking: $x, y = (10, 20)$.



Example

```
coordinates = (34.0522, -118.2437)  
print(coordinates[0]) # Output: 34.0522
```



Tuples in Python

```
t = ( 1 , 2 , 'python' , tuple () , ( 42 , ' hi' ) )  
      |     |     |     |  
      t [0] t [1] t [2] t [3] t [4]
```

Dictionary in Python

Definition

Stores data in key-value pairs for fast lookups.



Example

```
student = {"name": "Alice", "age": 22}  
print(student["name"]) # Output: Alice
```

Keys: Unique, immutable (e.g., strings, numbers, tuples).

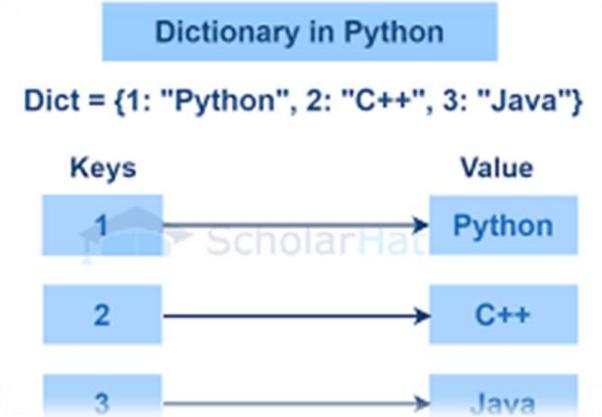
Values: Any data type, mutable.

Use Cases: Structured data (e.g., user profiles, settings, JSON).

Key Operations

Add: `dict['key'] = value`

Remove: `del dict['key']`



Advantage: Fast key-based access via hashing.

Set in Python

Definition

Stores unique, unordered elements..

Keys: Unique, immutable (e.g., strings, numbers, tuples).

Values: Elements must be immutable (e.g., strings, numbers, tuples).

Use Cases:

Membership testing, removing duplicates, mathematical set operations (union, intersection, difference).

Key Operations

Add: `set.add(element)`

Remove: `set.remove(element)`

Advantage: Fast membership testing via hashing; automatic duplicate removal.



Example

```
numbers = {1, 2, 3, 3, 4}
```

```
print(numbers)
```

```
# Output: {1, 2, 3, 4} (order may vary)
```

```
print(2 in numbers)
```

```
# Output: True
```



Comparison of Mutability and Use Case

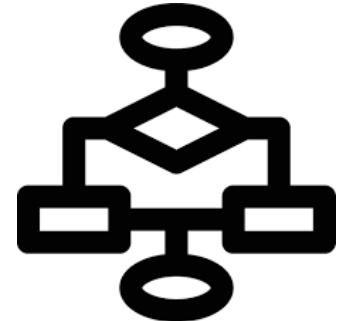
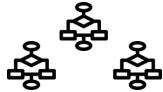
Data Type	Example	Mutable	Use Case
Int	x = 5	-	Whole numbers
Float	y = 3.24	-	Decimals
Str	s = 'text'	✗	Text
Bool	Flag = True	-	True/False
List	[1,2,3]	✓	Modifiable collection
Tuple	(1,2)	✗	Fixed Collection
Dict	{"key" : "value"}	✓	Key-Value Pairs
Set	{1,2,3}	✓	Unique elements

Conditional Statements



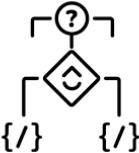
Definition

Conditional statements enable programs to execute different actions based on whether a condition evaluates to True or False.



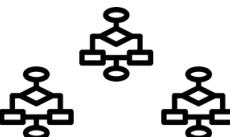
Condition

A logical expression resulting in a boolean (True or False).



Purpose

Controls program flow by enabling decision-making.



Real-Life Analogy

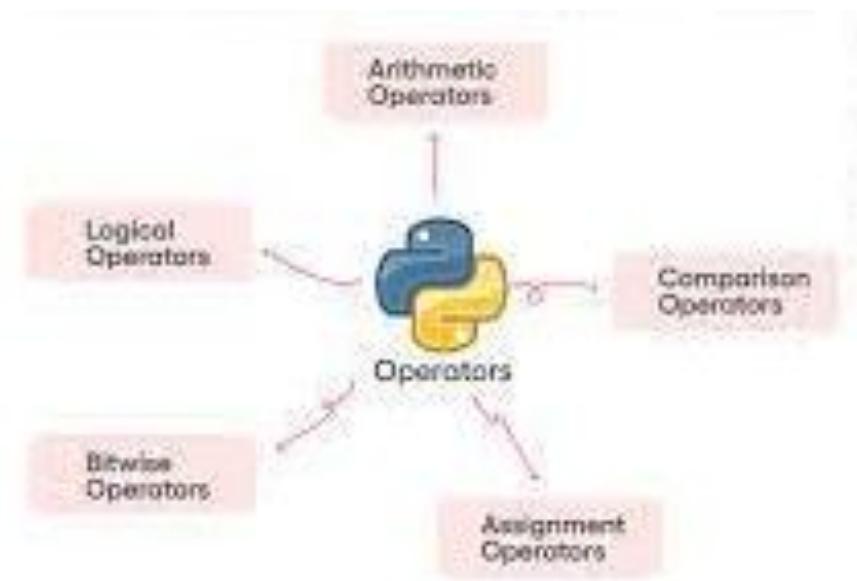
- Traffic light:
 - Green → Cars go.
 - Red → Cars stop.

Why Use Them?

- Execute actions when a condition is True.
- Skip actions when a condition is False.

Operators in Python

- Arithmetic: +, -, *, /, **, %
- Comparison: ==, !=, <, >, <=, >=
- Logical: and, or, not
- Assignment: =, +=, -= etc.



Conditions in Python



Definition

Expressions evaluating to True or False



Usage

Control flow in if, if-else, if-elif-else statements

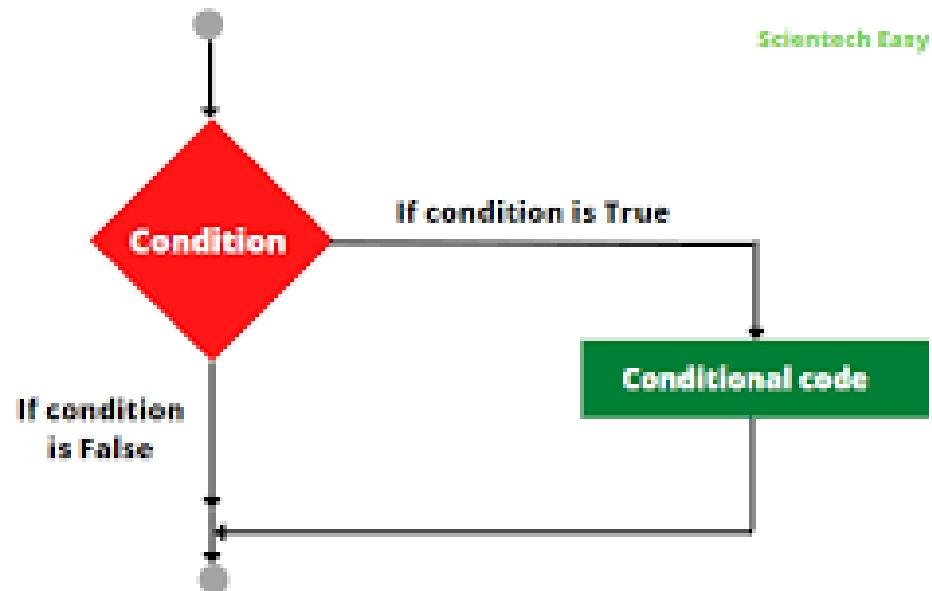


Example

How It Works: $x > 5$ evaluates to True, triggering the print statement.

```
x = 5
if x > 5:
    print("x is greater than 5")
```

Scientechnica Easy



If Statements in Python

Purpose

Execute a code block only if a condition is True

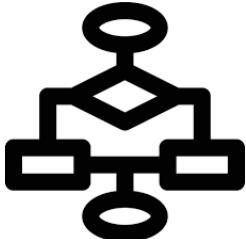
Syntax

if keyword, followed by a condition, then a colon :

Behavior

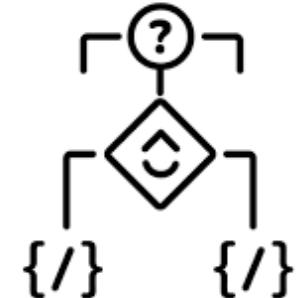
True condition: Code block executes

False condition: Code block is skipped



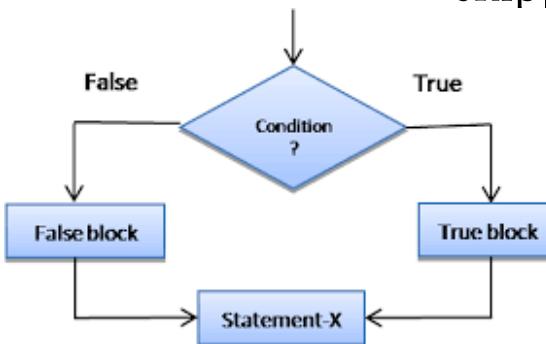
Example

```
num = 10  
if num > 5:  
    print("The number is greater than 5")
```



`num > 5` evaluates to True, so the message is printed.

If `num <= 5`, the `print` statement is skipped.



If-else Statements in Python



Purpose

Execute one code block if a condition is True, otherwise execute a different code block.



Syntax

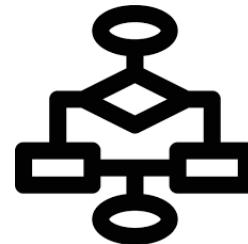
If a condition is True, then execute a code block; else, execute a different code block.



Behavior

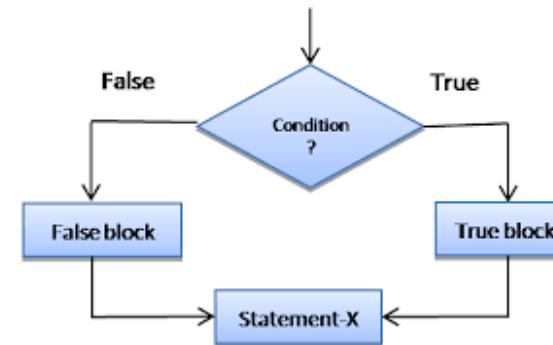
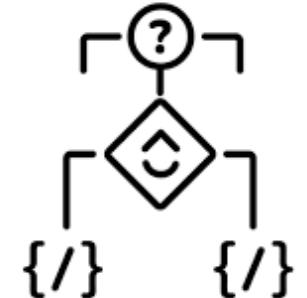
True condition: Code block 1 executes

False condition: Code block 2 executes



Example

```
num = 10
if num > 5:
    print("The number is greater than 5")
else:
    print("The number is not greater than
5")
```



If-Elif-Else Statements in Python



Purpose

Evaluate multiple conditions sequentially



Syntax

if: Checks the first condition

elif: Checks additional conditions if previous ones are **False**

else: Runs if no conditions are **True** (optional fallback)



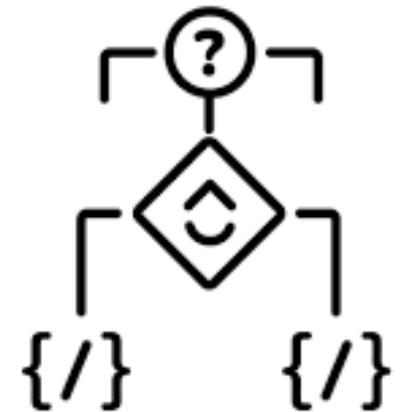
Behavior

Executes the first **True** condition's block or the **else** block if none are **True**



Example

```
score = 75
if score >= 90:
    print("Grade: A")
elif score >= 80:
    print("Grade: B")
elif score >= 70:
    print("Grade: C")
else:
    print("Grade: D")
```





KP-IT Board AI (ML/DL) Certification

Trainer: Fawad Bahadur

Introduction to Loops in Python

⌚ What Are Loops?

- Programming structures that repeat instructions until a condition is met.
- Enable iteration over data structures or repetitive task automation.



Why Use Loops?

- Reduce code redundancy
- Automate repetitive tasks
- Improve code efficiency and readability



Types of Loops

- **while Loop:** Repeats while a condition is 'True'
- **for Loop:** Iterates over a sequence (e.g., lists, tuples, strings)



Iterating with Python Loops



What Is Iteration?

- Accessing each item in a collection (e.g., list, tuple, string) one by one.



How Loops Work

- Iterate over Python data structures: lists, dictionaries, sets, strings.
- Enable sequential processing of items in an iterable.



Key Points

- Loops provide sequential access to elements.
- Use loops to read or modify items in sequences (e.g., lists, strings).

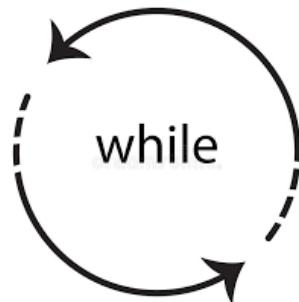


The while Loop in Python



How It Works

- Executes a code block as long as a condition is True.
- Stops when the condition becomes False.



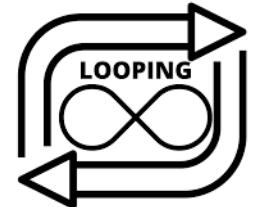
Useful Scenarios

- When the number of iterations is unknown (e.g., reading a file until its end).



Example

```
count = 0
while count < 3:
    print("Counting:", count)
    count += 1
```



Explanation

- Checks if `count < 3` (True initially).
- Prints `count` and increments by 1.
- Stops when `count` reaches 3 (False).

The for Loop in Python



How It Works

Iterates over each item in an iterable
(e.g., list, string, dictionary) until all items are processed.



Useful Scenarios

When the number of iterations is known or
iterating over a collection (e.g., lists, strings).



Example

```
colors = ["red", "green", "blue"]  
for color in colors:  
    print("Color:", color)
```



Explanation

- List: Loops through colors, printing each item.

The range() Function in Python



Purpose

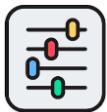
Generates a sequence of numbers for iterating a specific number of times.



Example

```
for i in range(1, 6):  
    print(i)
```

```
for i in range(5, 0, -1):  
    print(i)
```

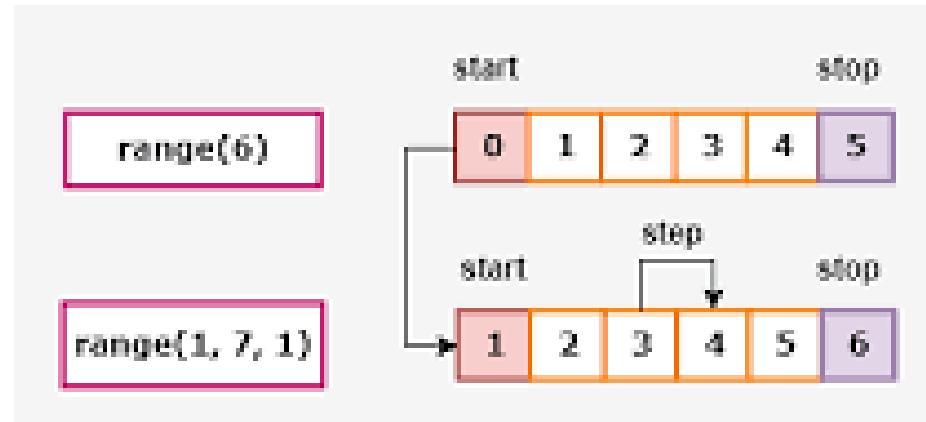


Parameters

Start: starting number (default: 0)

Stop: end number (non-inclusive)

Step: increment/decrement (default: 1)



The break Statement in Python



Purpose

Exits a loop prematurely when a specific condition is met.



When to Use

To stop a loop immediately (e.g., when finding a target element in a list).



Example

```
for number in range(10):
    if number == 5:
        break
    print(number)
```



Explanation

Loops through numbers 0 to 9.
When number == 5, break stops the loop.
Output: Prints 0, 1, 2, 3, 4.

The `continue` Statement in Python



Purpose

Skips the current iteration and proceeds to the next one without exiting the loop.



When to Use

To bypass specific iterations (e.g., skipping unwanted values in a dataset).



Example

```
for i in range(5):
    if i == 2:
        continue
    print(i)
```



Explanation

- Loops through numbers 0 to 4.
- When `i == 2`, `continue` skips the `print` statement.

The Pass Statement in Python



- A null operation; acts as a placeholder when no action is needed.
- Used where code is syntactically required but not yet implemented.



When to Use

As a temporary placeholder (e.g., in empty loops, functions, or try blocks).



Example

```
for i in range(5):
    if i < 3:
        pass
    print(i)
```



Explanation

- Loops through numbers 0 to 4.
- When $i < 3$, pass does nothing, and the loop continues.
- Output: Prints 0, 1, 2, 3, 4.



KP-IT Board AI (ML/DL) Certification

Trainer: Fawad Bahadur

Introduction to Function in Python



Definition

Named, reusable code blocks for specific tasks.



Purpose

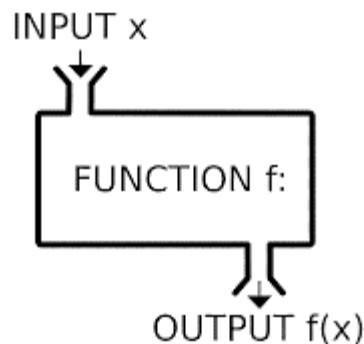
- Organize code into manageable parts.
- Promote reusability, reduce redundancy.
- Enable multiple calls throughout a program.

Syntax

Use def keyword, function name, and parameters in () .



Functions



Example

```
def greet(name):  
    print(f"Hello, {name}!")  
greet("Alice") # Output: Hello, Alice!
```



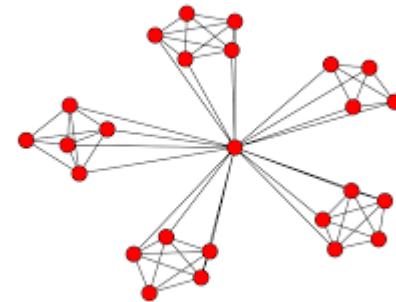
Explanation

greet function takes name parameter and prints a greeting.

Functions: Modularity & Reusability

Modularity

- Encapsulate functionality for organized, navigable code.
- Develop, test, and debug functions independently.
- Enhances software quality and maintainability.



Code Reusability

- Reuse functions across programs or sections.
- Eliminates redundant code.
- Keeps codebase clean and maintainable.



Function Definition in Python

Syntax

Use **def** keyword, function **name**, and **()** for parameters.

Parameters (optional) accept inputs for the function.



Purpose



Functions

- Define reusable code to perform specific tasks.
- Return results using **return** (optional).

```
def keyword      name      parameter
def fahr_to_celsius(temp):
    name
    parameter
    return ((temp - 32) * (5/9))
return statement      return value
```



Example

```
def add(a, b):
    return a + b
print(add(3, 4)) # Output: 7
```



Explanation

- **add** function takes parameters **a** and **b**.
- Returns their sum using **return**.

Executing Functions in Python



Function Execution

Runs when called, executing the function's code body.



Calling Syntax

- Use function name followed by () .
- Include arguments in () if required.



Multiple Calls

Reuse functions by calling them multiple times.



Example

```
def greet(name):
    print(f"Hello, {name}!")
greet("Alice") # Output: Hello, Alice!
greet("Bob")   # Output: Hello, Bob!
greet("Charlie") # Output: Hello, Charlie!
```



Functions



Explanation

greet function is called multiple times with different arguments.

Function Arguments in Python



Definition

Values (arguments) passed to a function for processing.
Used as inputs for calculations or manipulations.



Importance

Enable generic, flexible functions.
Avoid hardcoding, work with varied data.



Benefits

Dynamic functionality with different inputs.
Improves code readability and maintainability.

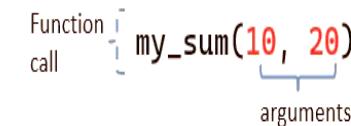
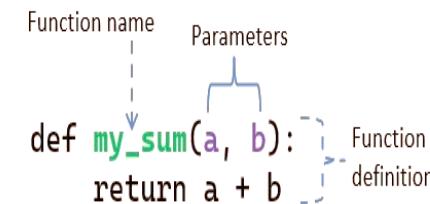


Example

```
def multiply(a, b):  
    return a * b  
  
result = multiply(4, 5)  
print(result) # Output: 20
```



Functions



Positional Arguments in Python



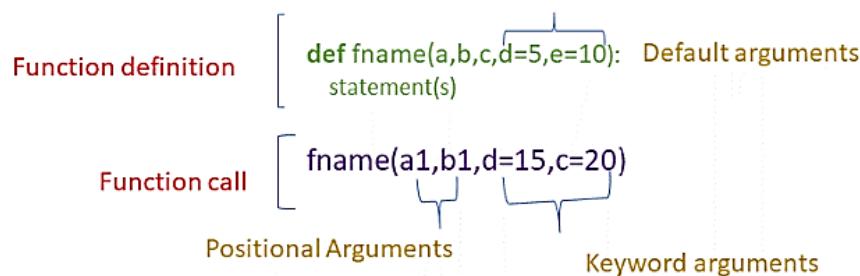
Definition

Arguments passed in the order of function parameters.
First argument matches first parameter, and so on.



Key Point

Order matters; incorrect order causes logical errors.



Example



```
def divide(numerator, denominator):
    if denominator == 0:
        return "Error: Cannot divide by zero."
    return numerator / denominator
result1 = divide(10, 2) # Output: 5.0
result2 = divide(2, 10) # Output: 0.2
print(result1)
print(result2)
```

Explanation

Correct: `divide(10, 2) → 10 / 2 = 5.0.`
Incorrect order: `divide(2, 10) → 2 / 10 = 0.2.`

Keyword Arguments in Python



Definition

Specify parameter names and values when calling a function.
Allows arguments to be passed in any order.



Benefits

Clarity: Clearly indicates what each argument represents.
Flexibility: Order-independent, ideal for optional parameters.

```
def num(a,b):  
    print(a,b)  
num(b=3,a=5)
```



Keyword Arguments



Functions



Example

```
def profile(name, age, city):  
    return f"{name} is {age} years old and lives in {city}."  
info = profile(age=30, name="Alice", city="New York")  
print(info) # Output: Alice is 30 years old and lives  
in New York.
```

Explanation

Arguments passed using parameter names,
ignoring order.

*args in Python



Definition

Allows a function to accept any number of positional arguments.
Arguments are collected as a tuple.



Benefits

Ideal when the number of arguments is unknown.

```
def num(a,b):  
    print(a,b)  
num(b=3,a=5)
```



Keyword Arguments



Example



Functions

```
def concatenate_strings(*args):  
    return " ".join(args)  
  
result = concatenate_strings("Hello", "world",  
"from", "Python!")  
print(result) # Output: Hello world from Python!
```

Explanation

- *args gathers all arguments into a tuple.
- join() combines them into a single string.

Global vs. Local Variables in Python

Global Variables

- Defined outside functions.
- Accessible anywhere in the code.

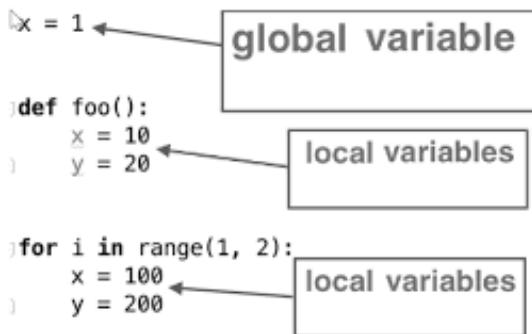


Example

```
global_var = 10 # Global variable
def function():
    local_var = 5 # Local variable
    print(global_var) # Output: 10
function()
# print(local_var) # Error: local_var is not
accessible
```

Local Variables

- Defined inside a function.
- Accessible only within that function's scope.



Explanation

- global_var is accessible everywhere.
- local_var is limited to the function; accessing it outside raises an error.



THANK
YOU