**PYTHON NOTES (1)**

**The Python print() Function**

The print() function is Python's primary way to display information (output) to the console.

**1. Basic Syntax and Simplicity**

Python's print() is straightforward and easy to use compared to other languages:

* It is a **built-in function**, meaning you do not need to import any libraries or set up a main function to start.
* To display "Hello World," you only need one line of code:

| Description | Code Example |
| --- | --- |
| **Simple String** | print("Hello World") |

**2. Printing Different Data Types**

The print() function is flexible and can handle various data types passed inside the parentheses.

| Data Type | Code Example |
| --- | --- |
| **String** | print("My name is Alex") |
| **Integer (Number)** | print(42) |
| **Boolean** | print(True) |

**3. Printing Multiple Items (The Comma Separator)**

You can pass multiple values into a single print() function. These values must be separated by a **comma (,)**.

| Description | Code Example | Output |
| --- | --- | --- |
| **Multiple Strings** | print("India", "Pakistan", "Nepal") | India Pakistan Nepal |
| **Mixed Data Types** | print("Answer is", 42, True) | Answer is 42 True |

**4. Customizing Output with Parameters**

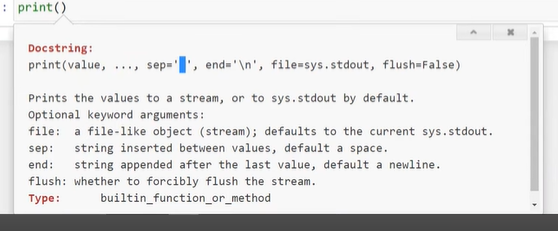
The print() function has powerful **optional parameters** that allow you to control how the output is formatted.

**A. The sep Parameter (Separator)**

The sep parameter controls what characters are placed **between** the multiple items you print.

* **Default Behavior:** By default, sep is set to a **single space** (' ').
* **Use Case:** You use sep to change the default space to a custom character (like a hyphen, slash, or pipe).

| Description | Code Example | Output |
| --- | --- | --- |
| **Using a Slash** | print("user", "data", "01", sep='/') | user/data/01 |
| **Using a Hyphen** | print(20, 24, sep='-') | 20-24 |



**B. The end Parameter**

The end parameter controls what is printed **after** the last item of a print() function.

* **Default Behavior:** By default, end is set to a **newline character** (\n), which causes every new print() statement to start on the next line.
* **Use Case:** You can use end to make multiple print() statements appear on the same line.

| Description | Code Example | Output |
| --- | --- | --- |
| **Printing on Same Line** | print("Hello", end=' ') print("World") | Hello World |
| **Printing with an Ellipsis** | print("Loading", end='...') print("Done") | Loading...Done |
|  |  |  |

**PYTHON NOTES (2)**

**Python Data Types 📊**

In Python, all data types are broadly categorized into three main groups: **Basic Types**, **Container Types** (or collections), and **User-Defined Types**.

**I. Basic Data Types (Fundamental Types)**

These are the most fundamental building blocks for storing single values.

| Data Type | Description | Key Feature | Examples |
| --- | --- | --- | --- |
| **Integer (int)** | Whole numbers (positive, negative, or zero). | Python integers have **arbitrary precision**, meaning they can store numbers of virtually any size. There is no practical limit, unlike in many other programming languages. | 10, -500, 1234567890,1e308(max) |
| **Float (float)** | Numbers that contain a decimal point. | They represent real numbers and also handle very large magnitudes. | 3.14, -0.001, 1.7e308(max) |
| **Boolean (bool)** | Represents truth values. | Has only two possible values: **True** and **False**. Used extensively for logical operations, conditions, and control flow (e.g., in if/else statements). | True, False |
| **String (str)** | A sequence of characters used to store text. | Can be defined using **single quotes** ('...'), **double quotes** ("..."), or **triple quotes** ('''...''' or """...""") for multi-line strings. | 'hello', "Python", '''This is a multiline note''' |
| **Complex (complex)** | Numbers with a real part and an imaginary part, often written as a+bj. | Python has built-in support for complex number arithmetic, useful in scientific and engineering applications. | 5 + 2j, 10 - 3j |

**II. Container Data Types (Collections)**

Container types are used to hold and organize **multiple values** in a single variable. They will be studied in great detail in subsequent lessons.

| Container Type | Description | Defining Character(s) | Example |
| --- | --- | --- | --- |
| **List (list)** | An ordered, mutable (changeable) sequence of items. | **Square Brackets** [] | [10, 'apple', True] |
| **Tuple (tuple)** | An ordered, **immutable** (unchangeable) sequence of items. | **Parentheses** () | ('Alice', 25, 'Engineer') |
| **Set (set)** | An unordered collection of **unique** items. | **Curly Braces** {} | {'red', 'green', 'blue'} |
| **Dictionary (dict)** | An unordered collection of **key-value pairs**. | **Curly Braces** {} | {'name': 'Alex', 'age': 30, 'city': 'NY'} |

**Key Differences to Note:**

* **List vs. Tuple:** Lists use [] and are **changeable** (mutable). Tuples use () and are **unchangeable** (immutable).
* **Set vs. List/Tuple:** Sets do not maintain order and automatically remove duplicate items.
* **Dictionary Structure:** Dictionaries store data as key: value pairs, providing extremely fast access to values using their unique keys.

**III. User-Defined Data Types**

These types are created by the programmer when writing code that follows the principles of Object-Oriented Programming (OOP).

* **Classes and Objects:** These are custom blueprints for creating data structures that combine both data (attributes) and functions (methods) into a single unit. They are the focus of a later, more advanced topic in Python programming.

**PYTHON NOTES (3)**

**📝 Comments in Python**

**1. Definition and Purpose**

A **comment** is a section of code that is completely **ignored by the Python interpreter**. This means comments do not affect how your program runs.

**The Core Objective**

The main goal of using comments is to **improve code readability** and serve as internal documentation for your program.

**Why Comments Are Essential:**

1. **Explaining Complex Logic:** To clarify the intent or rationale behind a piece of code that might not be immediately obvious to another developer.
2. **Self-Documentation:** To remind your future self about why you chose a particular approach, which is vital when returning to older projects.
3. **Temporary Disabling:** To temporarily "comment out" lines of code for testing or debugging without permanently deleting the code.

**2. Standard Commenting (Single-Line)**

The official and most common way to create a comment in Python is by using the **hash symbol** (#). Everything written after the # on a given line is treated as a comment.

**Example 1: Full-Line Comment**

Python

# This comment explains that the function below calculates the area of a circle.

def calculate\_area(radius):

return 3.14 \* radius \* radius

**Example 2: Inline Comment**

You can place a comment on the same line as an executable statement.

Python

user\_age = 25 # The age of the user is stored here.

print(user\_age)

**3. Multi-Line Comments (Workarounds)**

Python does **not** have a dedicated syntax for a true multi-line comment block (like the /\* ... \*/ found in languages like C++). Therefore, you must use one of two common workarounds:

**A. Repeated Single-Line Comments (Standard Method)**

The most Pythonic and common way to handle multi-line notes is simply to preface every line with a hash symbol.

Python

# The following section initializes the database connection

# and loads the user configuration files before starting the main loop.

# This should only run once at the beginning of the program.

db\_connect()

load\_config()

**B. Using Triple Quotes (Docstrings)**

A technically distinct feature called a **docstring** (documentation string) can be used to write large blocks of text. Since an unassigned triple-quoted string is ignored by the interpreter, it behaves like a multi-line comment.

This method is generally preferred for function or module documentation, but can be used as a comment block:

Python

"""

License information:

This script is the property of [Your Company Name].

Unauthorized copying or distribution is strictly prohibited.

Last updated: October 2025.

"""

print("Starting up...")

**PYTHON NOTES (4)**

**Variables in Python: Dynamic Typing & Binding**

**I. Variables: The Concept**

A **variable** is a name used to refer to a value stored in memory. They act as **containers for future use** where the actual data might not be known when you are writing the program (e.g., a user's name or input number).

* **Analogy:** A variable is like a placeholder or a label that you attach to a specific piece of data.

**II. Variable Creation in Python**

Unlike languages like C or Java, Python does not require you to explicitly declare a variable before using it.

**A. Simple Assignment**

You create a variable the moment you assign a value to it using the **assignment operator** (=).

| Description | Code Example |
| --- | --- |
| Assigning a string | name = "Nitish" |
| Assigning an integer | age = 30 |
| Assigning a boolean | is\_active = True |

**B. Multiple Assignment Syntaxes**

Python offers convenient ways to assign values to multiple variables simultaneously:

| Syntax | Description | Code Example |
| --- | --- | --- |
| **Assigning unique values** | Assign different values to different variables in a single line. | a, b, c = 5, 6, 7 |
| **Assigning the same value** | Assign the same value to multiple variables. | x = y = z = 100 |

**III. Python's Unique Features**

Python variables have two major features that differentiate them from variables in many other languages (like C++ or Java).

**A. Dynamic Typing**

**Dynamic Typing** means you do not have to specify the **data type** of a variable when you create it. Python is "smart enough" to automatically figure out the data type based on the value you assign to it.

| Feature | Python (Dynamic Typing) | C/Java (Static Typing) |
| --- | --- | --- |
| **Declaration** | name = "Alice" (No type needed) | String name = "Alice"; (Type must be specified) |

**B. Dynamic Binding**

**Dynamic Binding** means that a variable is **not permanently bound** to a single data type for its lifetime. You can change a variable's data type at any point in your code.

| Concept | Description |
| --- | --- |
| **Dynamic Binding** | A variable can hold a value of one type, and then later hold a value of a completely different type. |

Export to Sheets

| Code Example | Explanation |
| --- | --- |
| data = "Hello" | data holds a **string** (str). |
| data = 42 | data is immediately reassigned to hold an **integer** (int). |
| data = False | data is reassigned again to hold a **boolean** (bool). |

This flexibility is a hallmark of Python and contributes to its reputation as an easy-to-use and expressive programming language.

**PYTHON NOTES (5)**

**📝 Keywords and Identifiers in Python**

This video introduces the essential concepts of **Keywords** (reserved words used by Python) and **Identifiers** (names you choose for your code elements).

**I. Python Case Sensitivity**

Before discussing Keywords and Identifiers, it is crucial to note that **Python is a case-sensitive programming language**.

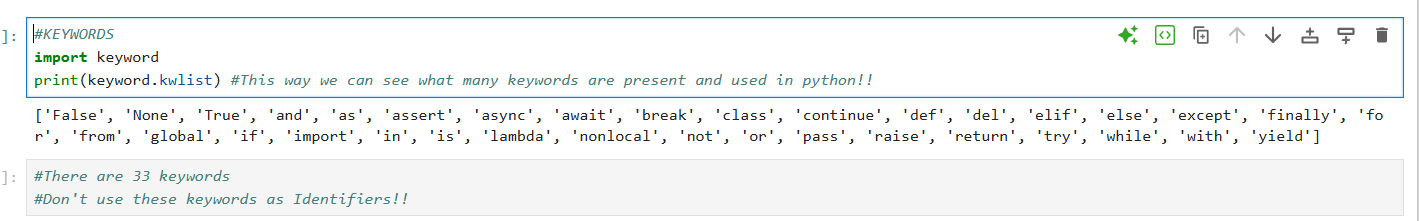
* This means that the interpreter treats Name, name, and NAME as three completely different entities (e.g., three separate variables).

**II. Keywords (Reserved Words)**

**Keywords** are words that have a **special, reserved meaning** to the Python interpreter. They are used to define the language's structure and commands (like loops, conditions, and functions).

**A. Key Concept**

You **cannot** use keywords as names for your variables, functions, classes, or any other identifier. Using them will cause a syntax error because the interpreter will expect the keyword to perform its special function.



**B. Examples of Keywords**

Python has 33 keywords (as of the video's recording). You do not need to memorize them; you will learn them as you code.

| Category | Examples | Purpose |
| --- | --- | --- |
| **Control Flow** | if, else, elif, for, while, break, continue | Used for decision making and loops. |
| **Logic/Boolean** | True, False, and, or, not | Used for logical operations and truth values. |
| **Functions/Classes** | def, class, return, yield | Used for defining functions and custom data types. |
| **Module/Handling** | import, from, as, try, except, with | Used for managing external code and handling errors. |

**C. Finding All Keywords**

If needed, you can programmatically view the full list of keywords:

Python

import keyword

print(keyword.kwlist)

**III. Identifiers (User-Defined Names)**

An **Identifier** is a **name** given by the programmer to various elements in the code, such as variables, functions, classes, modules, or objects.

**A. Rules for Naming Identifiers**

When creating identifiers, you must follow three strict rules:

1. **Valid Starting Characters:** An identifier must start with either a **letter** (A-Z, a-z) or an **underscore** (\_).
   * **Valid:** name, \_data, Name\_1
   * **Invalid:** 1name (Cannot start with a digit)
2. **Allowed Characters:** After the first character, an identifier can contain **letters**, **underscores** (\_), and **digits** (0-9).
   * **Valid:** user\_age\_2, final\_score
   * **Invalid:** first-name (Hyphens/dashes - are not allowed)
3. **Keyword Exclusion:** An identifier **cannot** be the same as any Python keyword.

**B. Identifier Examples**

| Status | Code Example | Reason |
| --- | --- | --- |
| **Valid** | first\_name = "Alex" | Starts with a letter, contains letters and an underscore. |
| **Valid** | \_temp\_var = 10 | Starts with an underscore, which is allowed. |
| **Valid** | sum2numbers = 50 | Contains letters and digits, but does not start with a digit. |
| **Invalid** | 2nd\_score = 95 | **Violates Rule 1:** Starts with a digit. |
| **Invalid** | my-variable = 10 | **Violates Rule 2:** Contains a hyphen (-). |
| **Invalid** | if = True | **Violates Rule 3:** Cannot use the reserved keyword if. |

**PYTHON NOTES (6)**

**📝 Taking User Input in Python**

The ability to take input from a user is crucial for creating **dynamic software** (like YouTube or WhatsApp), where the program interacts back and forth with the user. Python uses the built-in input() function for this purpose.

**I. The input() Function**

The input() function pauses the program's execution and waits for the user to type something into the console and press Enter.

**A. Basic Usage**

The function is called with optional text inside the parentheses, which serves as a prompt for the user.

| **Description** | **Code Example** | **Console Output** |
| --- | --- | --- |
| **Taking input** | name = input("Enter your name: ") | Enter your name: [User types here] |

**B. The Importance of the Prompt**

While you can call input() without any arguments, it is highly recommended to provide a **prompt string** inside the parentheses.

* **Purpose:** The prompt tells the user exactly what kind of data they need to enter (e.g., "Enter your age," not just a blinking cursor).

| **Code Example (Bad)** | **Code Example (Good)** |
| --- | --- |
| data = input() | data = input("Please enter your age: ") |

**II. The Critical Rule: Input is Always a String**

The single most important property of the input() function is that the data it returns to the programmer is **always a string (str)**, regardless of what the user types.

**A. The Universal Format**

Python returns a string because it is the most **universal format**. A string can hold a number, text, or a boolean value, whereas a type like an integer cannot hold text (e.g., you can store the number 56 as a string "56", but you cannot store the word "hello" as an integer).

**B. Demonstrating the Issue**

If you try to perform arithmetic on two inputs, the program will perform **string concatenation** (joining the strings) instead of mathematical addition.

| **Code** | **Output (if user inputs 56 and 76)** | **Explanation** |
| --- | --- | --- |
| num1 = input("First: ") num2 = input("Second: ") result = num1 + num2 print(result) | 5676 | Since num1 is "56" and num2 is "76" (both strings), the + operator joins them like text. |

**III. Verification (The type() Function)**

To confirm the data type of any variable in Python, you can use the built-in **type() function**.

| **Description** | **Code Example** | **Output** |
| --- | --- | --- |
| **Checking the type** | data = input("Enter a number: ") print(type(data)) | <class 'str'> |
| **Checking a literal** | print(type(42)) | <class 'int'> |

This function confirms that even if the user types a number, the resulting variable holds a value of type str.

**IV. The Solution: Type Conversion**

To fix the concatenation problem and perform arithmetic, you must **convert** the string input into the desired numeric data type (e.g., int or float). This process is called **Type Conversion** and will be covered in the next topic.

**PYTHON NOTES (7)**

**📝 Type Conversion in Python**

**Type Conversion** is the process of changing a value from one data type (like a string) to another (like an integer). This is essential in Python, especially after taking user input, which is always returned as a string.

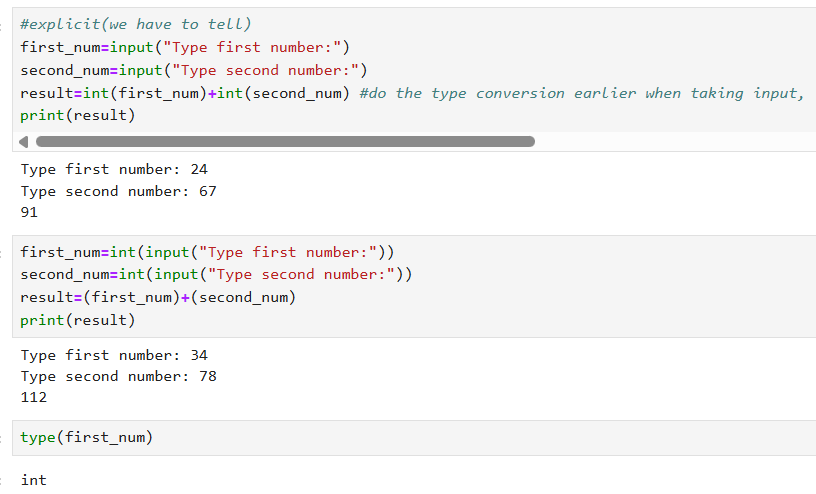
**I. Types of Type Conversion**

Type conversion is broadly divided into two categories based on who performs the conversion.

**A. Implicit Type Conversion (Automatic)**

**Implicit Conversion** happens automatically when the Python interpreter handles the conversion behind the scenes without the programmer's explicit instruction.

* **When it Occurs:** This typically happens when mixing data types in an arithmetic operation, where Python automatically promotes the "smaller" type to the "larger" type (e.g., converting an integer to a float).
* **Goal:** To prevent data loss and ensure the operation is possible.



| **Code Example** | **Explanation** |
| --- | --- |
| result = 4 + 5.5 | The integer 4 is automatically converted to a float 4.0 before addition, resulting in 9.5. |
| result = 5 + (6 + 7j) | The integer 5 is automatically converted to a complex number before addition. |

**B. Explicit Type Conversion (Manual)**

**Explicit Conversion** (also called **Type Casting**) is when the programmer manually tells Python to convert a value from one type to another.

* **When it Occurs:** This is necessary when an automatic conversion is not possible or not desired, such as converting a user's string input to a number for arithmetic.
* **Goal:** To perform an operation that requires a specific data type.

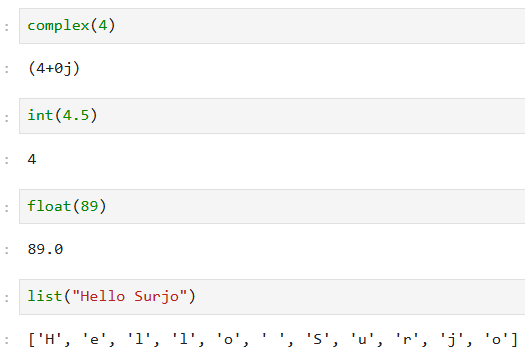
**II. Performing Explicit Type Conversion**

Python provides built-in functions for performing explicit conversion, with a function named after the data type you want to convert **to**.

**A. Core Conversion Functions**

To convert a value, simply wrap the value or variable inside the conversion function:

| **Function** | **Converts To...** | **Example (Conversion)** | **Result** |
| --- | --- | --- | --- |
| **int()** | Integer | int("42"), int(4.9) | 42, 4 |
| **float()** | Float | float("12.3"), float(10) | 12.3, 10.0 |
| **str()** | String | str(123), str(True) | "123", "True" |
| **bool()** | Boolean | bool(1), bool(0) | True, False |

****

**B. Logical Requirement (Compatibility)**

* Conversion is only possible if the data types are **compatible**.
* You **cannot** convert a string that is not a valid number into an integer.
  + **Invalid:** int("Kolkata") would result in an error.

**C. Solution to the Input Problem**

To correctly add two numbers taken from a user via input(), you must explicitly convert the string inputs to integers or floats:

| **Problem Code (String Concatenation)** | **Solution Code (Type Conversion)** |
| --- | --- |
| num1 = input("First: ") # "56" num2 = input("Second: ") # "76" result = num1 + num2 # "5676" | num1 = int(input("First: ")) # Converts to 56 num2 = int(input("Second: ")) # Converts to 76 result = num1 + num2 # **132** |

**III. Important Note on Type Conversion**

**Type conversion is NOT a permanent operation on the original variable.**

When you convert a variable's type, you are creating a **new temporary value** of the converted type. The original variable retains its original value and data type.

Example:-

a = 4.5 # 'a' is a float

b = int(a) # 'b' is a new integer value (4)

print(a) # Output: 4.5 (Original 'a' is unchanged)

print(b) # Output: 4 (The new integer value)

****

**PYTHON NOTES (8)**

📝 Literals in Python

A Literal is the raw data or fixed value that you assign to a variable or use in an expression. It is the data given to a variable.

In Python, literals are categorized into four main types: Numeric, String, Boolean, and Special.

I. Numeric Literals

These are the fixed numerical values assigned to integer, float, or complex variables.

A. Integers (Default Base 10)

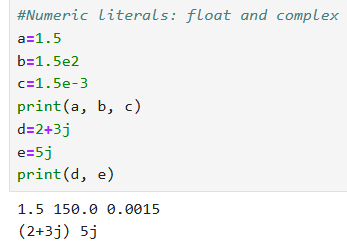
Integers can be assigned using bases other than the standard base 10 (decimal), which can be useful in low-level programming or electronics.

| Base | Prefix | Description | Example (Input) | Value (Output) |
| --- | --- | --- | --- | --- |
| Decimal | *(None)* | Standard base 10 (0-9) | a = 10 | 10 |
| Binary | 0b or 0B | Base 2 (0 and 1) | b = 0b1010 | 10 |
| Octal | 0o or 0O | Base 8 (0-7) | c = 0o12 | 10 |
| Hexadecimal | 0x or 0X | Base 16 (0-9, A-F) | d = 0xa | 10 |



B. Floats and Complex

| Type | Syntax | Description | Example | Resulting Value |
| --- | --- | --- | --- | --- |
| Scientific Notation | Use e or E to represent "times 10 to the power of". | Useful for representing very large or very small numbers. | f1 = 1.5e2 f2 = 1.5e-3 | 150.0 0.0015 |
| Complex | The imaginary part is suffixed with j or J. | Can be initialized with only the imaginary part. | c1 = 4 + 5j c2 = 5j | (4+5j) (0+5j) |



II. String Literals

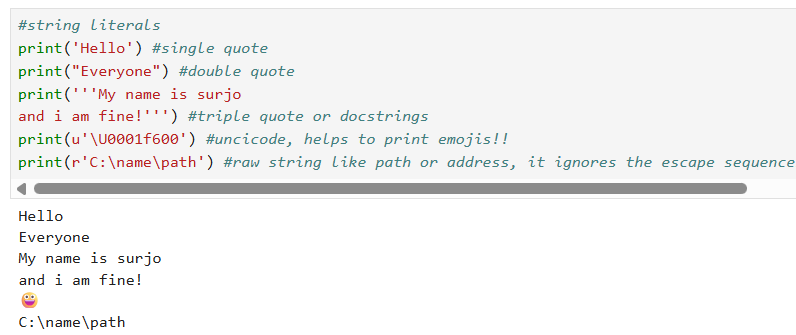
String literals are sequences of characters used to represent text.

A. Quoting Methods

| Quoting Style | Use Case | Example |
| --- | --- | --- |
| Single Quotes | Standard string definition. | 'Hello' |
| Double Quotes | Standard string definition (interchangeable with single quotes). | "World" |
| Triple Quotes | Used for defining multi-line strings or docstrings. | '''Line 1\nLine 2''' |

B. Special String Types

| Type | Prefix | Description | Example |
| --- | --- | --- | --- |
| Unicode Strings | u or U | Allows representation of characters outside the standard ASCII set, such as emojis. | u'\U0001f600' (Represents a smiley face emoji) |
| Raw Strings | r or R | Treats backslashes (\) as literal characters instead of escape sequences. Essential for regular expressions and file paths. | r'C:\name\path' |



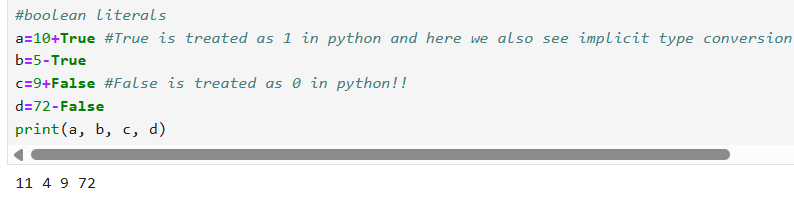
III. Boolean Literals

Boolean literals represent truth values and are limited to two values: True and False.

A. Using Booleans in Arithmetic

In Python, True and False are treated as the integer values 1 and 0, respectively, during implicit type conversion.

| Code Example | Explanation |
| --- | --- |
| result = 10 + True | Python converts True to 1. The result is 11. |
| result = 5 + False | Python converts False to 0. The result is 5. |



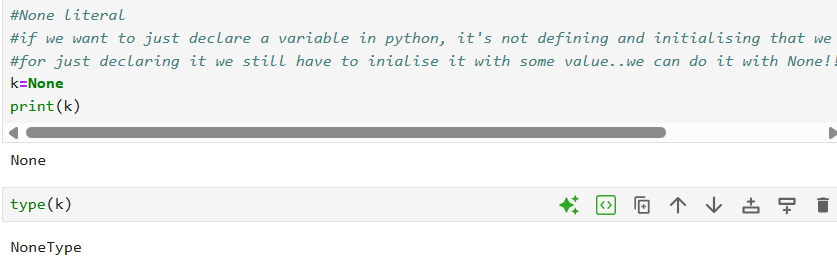
IV. Special Literals

A. The None Literal

The None literal is a special constant used to represent the absence of a value or a null object.

* Purpose: It is used to initialize a variable when you need the variable to exist but you do not want to assign it a meaningful value (like 0 or an empty string) yet.
* Usage: It is often used to declare a variable initially, which is a common practice in team coding even though Python doesn't require pre-declaration.

| Code Example | Explanation |
| --- | --- |
| user\_data = None | The variable user\_data exists but currently holds no value. |
| print(type(None)) | Returns <class 'NoneType'> |



**PYTHON NOTES (9)**

**📝 Operators in Python**

**Operators** are special symbols used to perform operations on variables and values (called operands). Python categorizes its operators into several groups.

**I. Arithmetic Operators**

These are used for basic mathematical calculations.

| **Operator** | **Name** | **Description** | **Example (x=5, y=2)** | **Result** |
| --- | --- | --- | --- | --- |
| + | Addition | Adds two operands. | x + y | 7 |
| - | Subtraction | Subtracts the right operand from the left. | x - y | 3 |
| \* | Multiplication | Multiplies two operands. | x \* y | 10 |
| / | Division | Divides the left operand by the right, always resulting in a float. | x / y | 2.5 |
| % | Modulus | Returns the remainder of the division. | x % y | 1 |
| \*\* | Exponentiation | Raises the left operand to the power of the right. | x \*\* y | 25 |
| // | Floor Division | Divides and returns the integer part of the quotient (rounds down to the nearest whole number). | x // y | 2 |

**II. Comparison (Relational) Operators**

These operators compare two values and return a **Boolean** result (True or False).

| **Operator** | **Name** | **Description** | **Example (x=5, y=2)** | **Result** |
| --- | --- | --- | --- | --- |
| == | Equal | Checks if the values of two operands are equal. | x == y | False |
| != | Not Equal | Checks if the values are not equal. | x != y | True |
| > | Greater Than | Checks if the left operand is greater than the right. | x > y | True |
| < | Less Than | Checks if the left operand is less than the right. | x < y | False |
| >= | Greater Than or Equal To | Checks if the left operand is greater than or equal to the right. | x >= y | True |
| <= | Less Than or Equal To | Checks if the left operand is less than or equal to the right. | x <= y | False |

**III. Logical Operators**

These operators combine conditional statements and return a Boolean result.

| **Operator** | **Name** | **Description** | **Example (a=True, b=False)** | **Result** |
| --- | --- | --- | --- | --- |
| and | AND | Returns True if **both** statements are true. | a and b | False |
| or | OR | Returns True if **at least one** of the statements is true. | a or b | True |
| not | NOT | Reverses the logical state of the operand. | not a | False |

**IV. Assignment Operators**

These are used to assign values to variables.

**A. Simple Assignment**

* **=**: Assigns the value from the right side to the left side variable.
  + **Example:** a = 10

**B. Compound Assignment**

These are shorthand operators combining an arithmetic operation with assignment.

| **Operator** | **Shorthand** | **Equivalent Long Form** | **Example (a=10)** | **Result** |
| --- | --- | --- | --- | --- |
| += | Add & Assign | a = a + 3 | a += 3 | 13 |
| -= | Subtract & Assign | a = a - 3 | a -= 3 | 7 |
| \*= | Multiply & Assign | a = a \* 3 | a \*= 3 | 30 |
| /= | Divide & Assign | a = a / 3 | a /= 3 | 3.33... |
| \*\*= | Power & Assign | a = a \*\* 2 | a \*\*= 2 | 100 |

**Note:** Python does **not** support the increment/decrement operators ++ or -- (e.g., a++ or ++a) found in languages like C/Java. You must use a += 1 or a = a + 1.

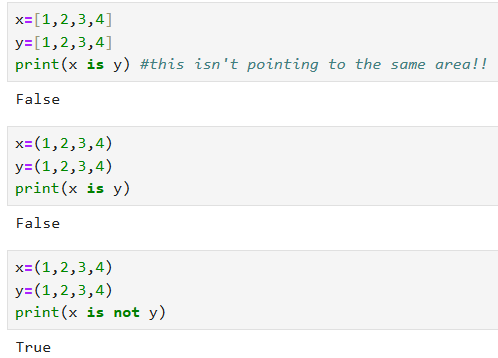
**V. Identity Operators**

These operators compare the **memory location** (object identity) of two variables, not just their value.

| **Operator** | **Name** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- | --- |
| **is** | Is | Returns True if both variables point to the **same object** in memory. | x = [1, 2] y = [1, 2] x is y | False (They are separate lists, even though they look the same.) |
| **is not** | Is Not | Returns True if both variables do **not** point to the same object. | x is not y | True |



**VALUES WHICH LOOK SAME DOES NOT NECESSARILY MEAN THAT THEY POINT TO THE SAME MEMORY LOCATION!!**



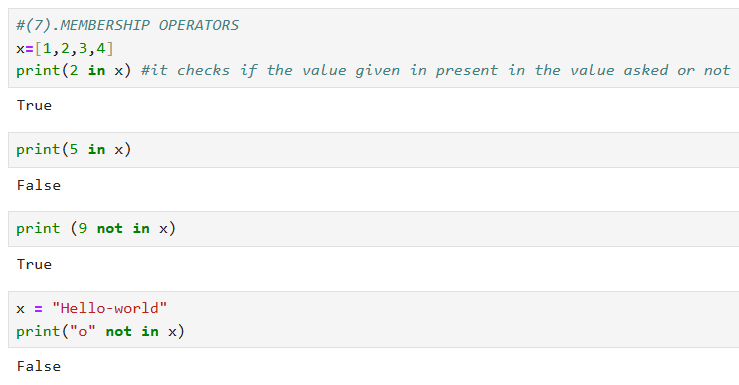
**VI. Membership Operators**

These operators test whether a sequence (string, list, tuple, etc.) contains a specific element.

| **Operator** | **Name** | **Description** | **Example (s='Delhi')** | **Result** |
| --- | --- | --- | --- | --- |
| **in** | In | Returns True if the specified value is found in the sequence. | 'D' in s | True |
| **not in** | Not In | Returns True if the specified value is **not** found in the sequence. | 'z' not in s | True |

**(\*). THIS OPERATOR IS A SPECIAL OPERATOR WE ONLY FIND IN PYTHON!!**

**IT IS NOT PRESENT IN ANY OTHER LANGUAGE, IF WE WANT TO FIND THAT WHETHER A VALUE IS PRESENT IN A GIVEN THING OR NOT THEN WE HAD TO USE LOOP AND IF-ELSE STATEMENTS TO CHECK!!**



**VII. Bitwise Operators**

These operators perform operations bit by bit on the binary representation of integers. They are generally used for specialized tasks like graphics processing or low-level operations.

| **Operator** | **Name** | **Description** |
| --- | --- | --- |
| & | Bitwise AND | Sets each bit to 1 if both bits are 1. |
| | | BITWISE OR | Sets answer to 0 if both bits are 0. |
| ^ | Bitwise XOR | Sets each bit to 1 if only one of the two bits is 1. |
| ~ | Bitwise NOT | Inverts all the bits (unary operator). |
| << | Left Shift | Shifts the bits of the number to the left. |
| >> | Right Shift | Shifts the bits of the number to the right. |

**PYTHON NOTES (10)**

📝 If-else Statements (Decision Control) in Python

If-else statements are fundamental decision control statements that allow a program to execute different blocks of code based on whether a specific condition is True or False. This introduces branching into your program's flow.

I. The Basic if...else Structure

The core structure allows the program to choose between two paths: one for when the condition is met, and one for when it is not.

A. Syntax

Python uses the colon (:) to start the code block and relies on indentation (usually four spaces) to define which statements belong inside the if or else block.

Python

if [condition is True]:

# Code to execute if the condition is True (The 'if' block)

print("The condition was met.")

else:

# Code to execute if the condition is False (The 'else' block)

print("The condition was NOT met.")

B. Example

Python

correct\_password = "123"

user\_input = input("Enter password: ")

if user\_input == correct\_password:

print("Login successful! Welcome.")

else:

print("Incorrect password. Please try again.")

II. The elif Statement (Multiple Conditions)

When you have more than two possible outcomes (multiple branches), you use the elif (Else-If) statement to check a second, third, or subsequent condition.

A. Structure

The flow checks the if condition first. If it's False, it moves to the first elif. This continues until an elif condition is True or, if all fail, the else block executes.

Python

if [first condition is True]:

# Executes only if the first condition is True

print("Case 1")

elif [second condition is True]:

# Executes if the first is False AND the second is True

print("Case 2")

else:

# Executes if all preceding conditions were False

print("All cases failed")

B. Example (Login Scenario with Partial Success)

This example checks for two conditions:

1. Correct Login: Both email AND password are correct.
2. Correct Email, Wrong Password: Email is correct, but the password is wrong.

Python

correct\_email = "test@example.com"

correct\_pass = "1234"

user\_email = input("Email: ")

user\_pass = input("Password: ")

if user\_email == correct\_email and user\_pass == correct\_pass:

print("Welcome! Full access granted.")

elif user\_email == correct\_email and user\_pass != correct\_pass:

print("Password incorrect. Please try again or reset.")

else:

print("Email and/or Password incorrect. Access denied.")

III. Nested if-else Statements

Nesting occurs when you place an entire if-elif-else block inside another if or elif block. This allows you to check a *sub-condition* only after an *initial condition* has been met.

A. Structure and Indentation

Nesting is defined solely by indentation. Each nested block must be indented further than its parent block.

Python

if [Outer Condition is True]:

print("Outer condition passed.")

if [Inner Condition is True]:

# This code runs ONLY if BOTH the Outer and Inner conditions are True

print("Inner condition passed.")

else:

# This code runs if Outer is True, but Inner is False

print("Inner condition failed.")

else:

print("Outer condition failed.")

B. Example (Re-prompting for Password)

If the user gives a correct email but a wrong password, you can nest an if-else block to give them a second chance to enter the password.

Python

correct\_pass = "1234"

# ... Assume user\_email was correct but user\_pass was wrong ...

if user\_email == correct\_email and user\_pass != correct\_pass:

# --- Start of Nested Block ---

print("Password incorrect. One final try.")

second\_pass = input("Enter password again: ")

if second\_pass == correct\_pass:

print("Welcome! Login successful on second attempt.")

else:

print("Second password attempt failed. Account locked.")

# --- End of Nested Block ---

else:

# ... Other cases

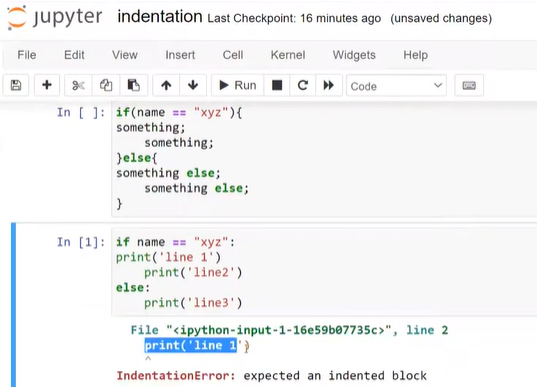
pass



**PYTHON NOTES (11)**

**IMPORTANT**

**INDENTATION**



📝 Indentation in Python

Indentation is a mandatory and critical feature in Python. Unlike most other programming languages (like C++ or Java), which use curly braces ({}) and semicolons (;) to define code blocks, Python uses whitespace (specifically, leading spaces) to determine the structure and hierarchy of your code.

I. Indentation vs. Code Blocks

In Python, indentation serves the purpose that curly braces serve in other languages.

* Code Block: A group of statements that belong together, such as the body of an if statement, a for loop, or a function definition.
* Defining a Block: A code block begins immediately after a line ending in a colon (:) and must be uniformly indented.

| Language | Block Delimiter | Line Terminator |
| --- | --- | --- |
| C/Java | Curly Braces {} | Semicolon ; |
| Python | Indentation (Whitespace) | *(None)* |

Example of Block Definition

Python

# The colon marks the start of a new block

if True:

# This is the IF block. It is indented.

print("This line is part of the IF block.")

print("So is this line.")

# This line is not indented, so it is outside the block.

print("This line runs regardless of the IF.")

II. The Rules of Indentation

Python enforces strict rules regarding how indentation must be applied.

1. Consistency is Key (The Golden Rule)

Within a single block of code, you must use a consistent number of spaces for indentation. The Python standard is four spaces per indentation level.

2. Error on Inconsistency

If you mix tabs and spaces, or if you use a different number of spaces for different lines within the same block, Python will throw an IndentationError.

3. Nesting with Indentation

When you create a nested structure (e.g., an if statement inside another if statement), you simply add another level of indentation.

Example of Nested Indentation

Python

if (condition\_A):

# First level of indentation (4 spaces)

print("Condition A is True")

if (condition\_B):

# Second level of indentation (8 spaces)

print("Condition B is True as well")

# Back to the first level

print("Processing A is complete")

III. Indentation Errors

Since indentation is mandatory, incorrect indentation is a syntax error that prevents your code from running. Common errors include:

| Error Type | Example | Python Error |
| --- | --- | --- |
| Unexpected Indent | if True: print("Correct") print("Error") | IndentationError: unexpected indent (The inner line has too many spaces for the current block.) |
| Expected Indent | if True: print("Error") | IndentationError: expected an indented block (The line following the colon must be indented.) |

Best Practice: Always use the Tab key in your code editor. Most modern Python editors are configured to insert the standard four spaces when the Tab key is pressed, helping to prevent inconsistent spacing.

**PYTHON NOTES (11)**

**📝 The while Loop in Python**

A **Loop** is a control structure that allows a block of code to be executed **repeatedly** based on a certain condition. They are essential for tasks requiring repetition, known as **Iteration Control**.

**I. When to Use Loops (Real-World Application)**

Loops are used whenever you need to display or process data that shares the same structure.

* **Example (E-commerce):** When building a page like a Flipkart or Amazon search results page, you would not create thousands of individual product containers.
  1. Create only **one single product container** (the code structure).
  2. Use a loop to retrieve data for all 4,000 products from the database.
  3. The loop runs 4,000 times, and in each iteration, it plugs in the unique data (image, price, description) into the single container structure and displays it.

**II. Types of Loops in Python**

Python primarily offers two types of loops:

1. **while Loop** (Covered here)
2. **for Loop** (Covered in the next topic)

**III. The while Loop Structure**

The while loop executes a block of code **as long as** its given condition remains True.

**A. Syntax**

Similar to if statements, the while loop uses a **colon (:)** to mark the start of the block and relies on **indentation** to define the code inside the loop.

Python

# 1. Initialization (Start variable)

i = 1

# 2. Condition (Loop continues as long as this is True)

while i <= 5:

# 3. Code Block (Indented)

print("Loop iteration:", i)

# 4. Update (Crucial to change the loop variable to avoid infinite loop)

i = i + 1

**B. Steps for Effective while Loop Programming**

A correctly written while loop must follow four key steps:

1. **Initialization:** Set up a starting value for the loop control variable (e.g., i = 1).
2. **Condition:** Define the termination condition that keeps the loop running (e.g., while i < 11).
3. **Code Block:** Write the code to be repeated.
4. **Update:** **Crucially**, update the control variable within the loop (e.g., i = i + 1 or i += 1) so that the condition eventually becomes False and the loop stops.

**C. Example: Printing a Multiplication Table**

This program prints the multiplication table for a number entered by the user.

Python

# 1. Get user input and convert it to an integer

number = int(input("Enter the number for the table: "))

# 2. Initialize the loop variable

i = 1

# 3. Define the termination condition (runs 10 times: i=1 to i=10)

while i <= 10:

# Print the table line in a formatted way

print(number, "x", i, "=", number \* i)

# 4. Update the loop variable

i = i + 1

**PYTHON NOTES (12)**

**📝 Python Guessing Game: while Loop and if-else Implementation**

This video demonstrates a practical application of the while loop and if-else statements by creating a number guessing game.

**I. Game Logic Flow**

The goal of the game is to have the user guess a randomly selected number between 1 and 100.

1. **Generate Jackpot Number:** The program randomly selects a number (the "jackpot").
2. **User Input:** The user makes an initial guess.
3. **Loop & Comparison:** A while loop runs continuously as long as the user's guess is **not** equal to the jackpot number.
4. **Feedback:** Inside the loop, if-elif-else statements provide feedback:
   * If the guess is too low, the program prompts "Guess Higher."
   * If the guess is too high, the program prompts "Guess Lower."
5. **Re-Guess:** The user is asked to input a new guess.
6. **Termination:** Once the guess matches the jackpot, the loop terminates, and the program announces a win, showing the number of attempts.

**II. Required Concepts and Code**

**1. Generating Random Numbers**

To generate a random "jackpot" number, you must use the random module.

* **Import:** You must first import the module using the import keyword.
* **Function:** Use the **randint(a, b)** function, which generates a random **integer** between a and b, **inclusive** of both endpoints.

Python

import random

# Generates a random integer between 1 and 100 (inclusive)

jackpot = random.randint(1, 100)

**2. Initialization and Loop Setup**

Before the while loop starts, you need to set up variables for the guess and the attempt counter.

* The **initial guess** must be taken *before* the loop, as the while condition needs a value to evaluate.
* The initial guess must be type-converted from a string to an integer using int().

Python

# Initialize counter

counter = 0

# Get first guess and convert to integer

guess = int(input("Guess a number between 1 and 100: "))

counter += 1

# Loop continues as long as the guess is NOT EQUAL to the jackpot

while guess != jackpot:

# Logic runs inside here

# ...

**3. Loop Logic (if-else and Counter Update)**

Inside the while loop, you provide feedback and update the guess and the counter.

Python

while guess != jackpot:

# 1. Provide feedback (Decision Control)

if guess < jackpot:

print("Guess Higher")

elif guess > jackpot:

print("Guess Lower")

# 2. Get the new guess from the user (Crucial to update the loop variable)

guess = int(input("Enter your next guess: "))

# 3. Update the counter

counter += 1

# This code runs AFTER the loop terminates (meaning guess == jackpot)

print("Correct! You took", counter, "attempts.")

**(FROM HERE THE NOTES ARE IN HINGLISH)**

**PYTHON NOTES (13)**

**🐍 Python For Loop: Detailed Notes**

Python ka for loop baki languages (C, C++, Java) se kaafi alag hota hai. C++ mein hum condition aur increment khud likhte hain, par Python mein ye **Sequence** ya **Iterables** ke upar chalta hai.

**1. Range Function (range())**

Video mein bataya gaya hai ki for loop seekhne se pehle range() samajhna zaroori hai. Ye function integers generate karta hai.

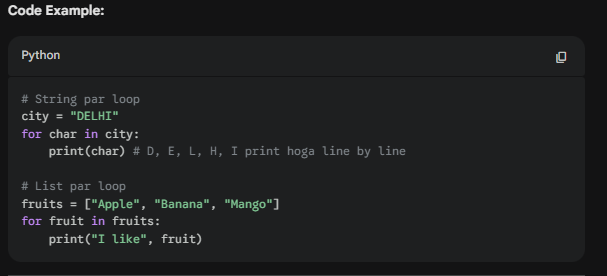
* **Syntax:** range(start, stop, step)
* **Important Points:**
  + start: Jahan se shuru karna hai (Default 0 hota hai).
  + stop: Jahan tak jaana hai (**par ye number include nahi hota**, i.e., n-1 tak chalta hai).
  + step: Kitne ka gap chahiye (Default 1 hota hai).



**2. Sequence Par Iteration**

Python mein for loop kisi bhi sequence (list, string, tuple, etc.) ke har item par ek-ek karke ja sakta hai.

* **Strings:** Har character par iterate karta hai.
* **Lists/Tuples:** Har element par iterate karta hai.



**3. For Loop vs While Loop (Kab kya use karein?)**

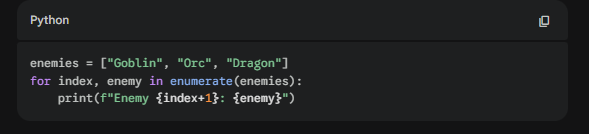
Video mein ek bohot achhi tip di gayi hai jo interview mein bhi puchi jaati hai:

* **For Loop:** Jab tumhe **pehle se pata ho** ki loop kitni baar chalana hai (e.g., list ke saare items print karne hain).
* **While Loop:** Jab tumhe **nahi pata** ki loop kitni baar chalega, aur wo kisi condition par depend karta hai (e.g., Guessing game jab tak user sahi answer na de de).

**4. 💡 Extra Knowledge**

Tumhe game developer banna hai, toh ye extra points bohot kaam aayenge:

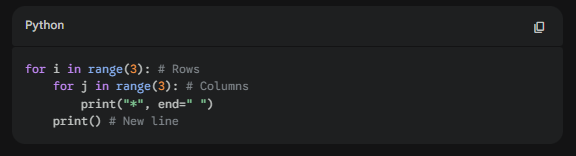
* **enumerate() Function:** Kabhi kabhi loop mein hume item ke saath uska **index** (position) bhi chahiye hota hai.



 **break and continue:**

* break: Loop ko turant rokne ke liye.
* continue: Current step ko skip karke agle step par jaane ke liye.

 **Nested For Loop:** Loop ke andar loop. Ye 2D grids ya maps banane mein kaam aata hai.



**Summary Checklist:**

1. **For Loop** iteration ke liye hota hai, condition checking ke liye nahi (C++ ki tarah).
2. **Range function** mein hamesha stop value exclude hoti hai.
3. Python loops mein **Indentation** (space) ka khaas dhyan rakhna, varna error aayega.
4. Ye loops **Lists, Strings, Tuples, Sets, aur Dictionaries** sab par kaam karte hain.

**PYTHON NOTES (14)**

**NESTED LOOPS**

**1. Nested Loops Kya Hain? (What?)**

Jab ek loop ke andar dusra loop likha jata hai, use **Nested Loop** kehte hain.

* **Outer Loop:** Ye "Bahar" wala loop hai jo slow chalta hai.
* **Inner Loop:** Ye "Andar" wala loop hai jo har ek outer loop ki value ke liye **poora** chalta hai.

**Hinglish Logic:** Agar outer loop 3 baar chal raha hai aur inner loop 5 baar, toh inner loop ke andar ka code total 3 \times 5 = 15 baar execute hoga.

**2. Kab aur Kyu use karte hain? (When & Why?)**

Hum nested loops tab use karte hain jab hamara data **Multi-dimensional** ho ya hume har element ko baaki sab elements se compare karna ho.

* **2D Arrays/Matrices:** Jaise koi Excel sheet ya image (jo pixels se bani hoti hai).
* **Game Grids:** Agar tumhe ek 10x10 ka map banana hai, toh ek loop rows ke liye hoga aur uske andar dusra loop columns ke liye.
* **Patterns:** Star patterns ya triangle patterns banane ke liye.
* **Comparing Items:** Jaise ek list mein check karna ki koi do numbers ka sum target ke barabar hai ya nahi.

**3. Time Complexity Kaise Badhti Hai?**

Yahi sabse main baat hai. Time complexity measure karti hai ki input badhne par code kitna slow hoga.

1. **Linear Complexity O(n):** Agar ek simple loop hai jo n tak chalta hai, toh wo n operations karega.
2. **Quadratic Complexity O(n^2):** Agar ek loop n tak chalta hai aur uske andar dusra loop bhi n tak chalta hai, toh total operations n \times n = n^2 ho jayenge.

**Example:** \* Agar n = 10 hai, toh 10^2 = 100 steps. (Theek hai!)

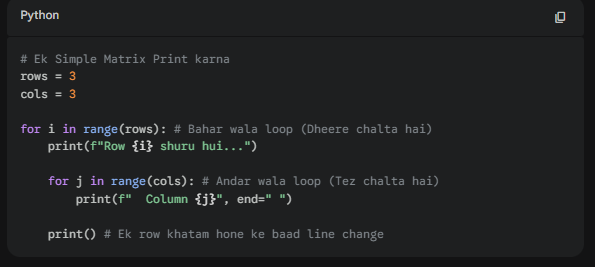
* Agar n = 10,000 hai, toh 10,000^2 = 100,000,000 (10 Crore) steps!

**Game Dev Context:** Agar tumhare game mein 1000 enemies hain aur tum har frame par har enemy ko baaki 999 enemies se compare kar rahe ho nested loop se, toh tumhara game **lag** karne lagega.

**4. Kyu Preferred Nahi Hai? (The Downsides)**

Professional coding mein nested loops (khaskar 2 se zyada levels) ko avoid kiya jata hai kyunki:

* **Performance Hit:** Jaise input badhta hai, execution time bohot tezi se badhta hai (O(n^2), O(n^3)).
* **Readability:** Code bohot complex dikhne lagta hai (ise "Pyramid of Doom" bhi kehte hain).
* **Better Alternatives:** Aaj kal hum Hash Maps, Dictionaries, ya Sorting use karke wahi kaam O(n \log n) ya O(n) mein kar sakte hain.



**Key Takeaways for your Notes:**

* **Outer Loop** "Row" ki tarah kaam karta hai.
* **Inner Loop** "Column" ki tarah kaam karta hai.
* **Time Complexity:** Agar loop ke andar loop hai toh complexity multiply ho jati hai.
* **Tip:** Try karo ki loop 2-level se zyada deep na ho (Avoid O(n^3)).

**PYTHON NOTES (15)**

**STAR PATTERNS**

**🌟 Star Patterns in Python: The Building Block Method**

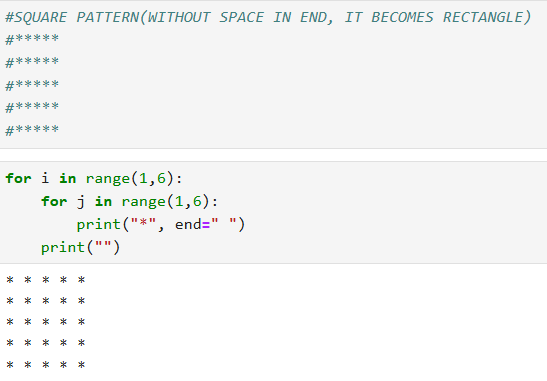
Video ka main mantra hai: **Saare patterns sirf 2 basic blocks se bante hain.**

1. **Increasing Triangle**
2. **Decreasing Triangle**

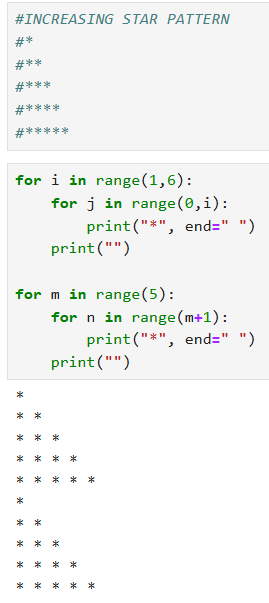
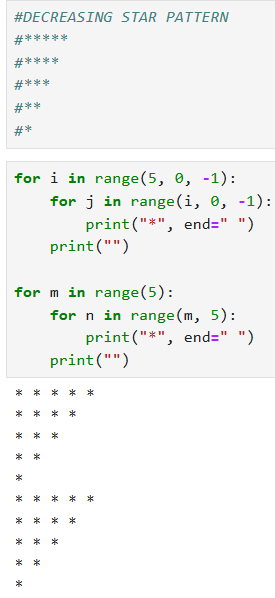
Pehle kuch rules dhyan rakho:

* Hum hamesha **Row by Row** print karte hain.
* Screen ke left side se printing shuru hoti hai.
* Agar beech mein gap hai, toh wo **Spaces** print karke aata hai (hum peeche nahi ja sakte). **TO SPACES KE LIYE ALAG FOR LOOP USE HOGA AND STARS KE LIYE ALAG USE HOGA!!**
* end=" " use karte hain taaki next star usi line mein aaye, aur print() use karte hain line badalne ke liye.

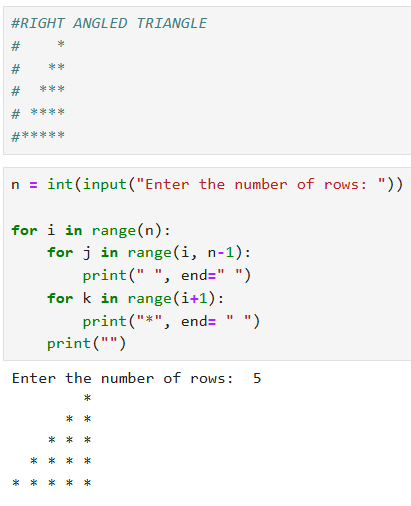
**SQUARE PATTERN**



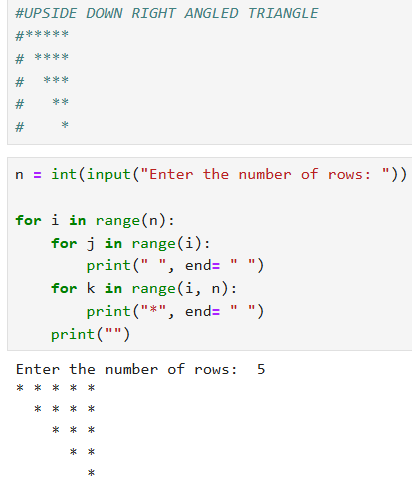
**INCREASING STAR PATTERN DECREASING STAR PATTERN**

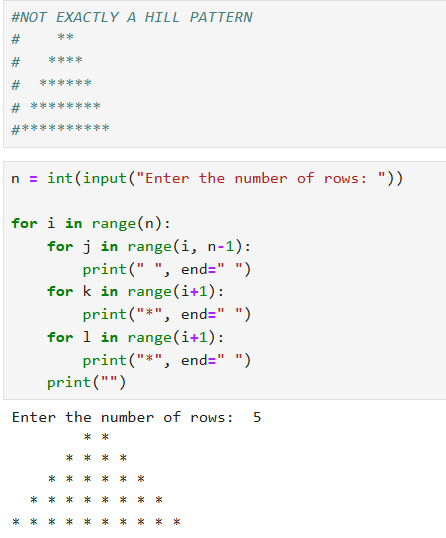
**RIGHT-ANGLED TRIANGLE**



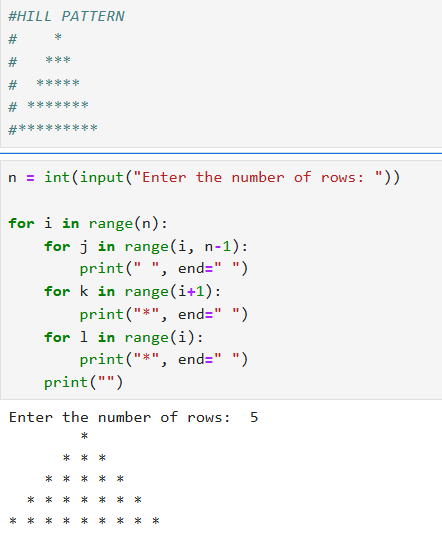
**UPSIDE-DOWN RIGHT-ANGLED TRIANGLE**



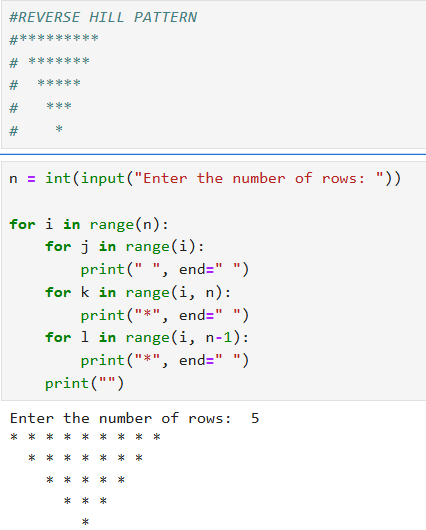
**NOT EXACTLY A HILL PATTERN**



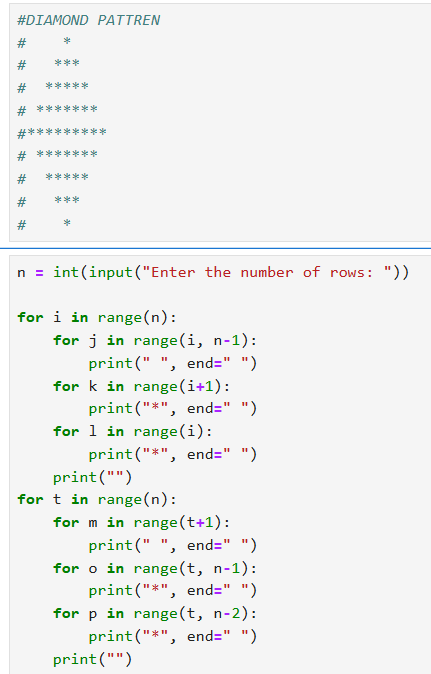
**HILL PATTERN**

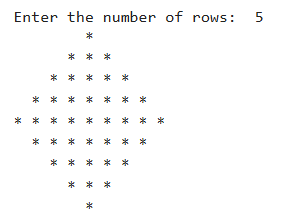


**REVERSE HILL PATTERN**

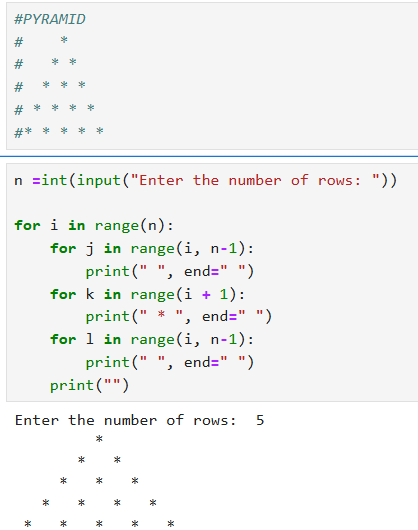


**#DIAMOND PATTERN**

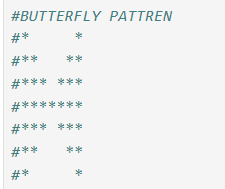




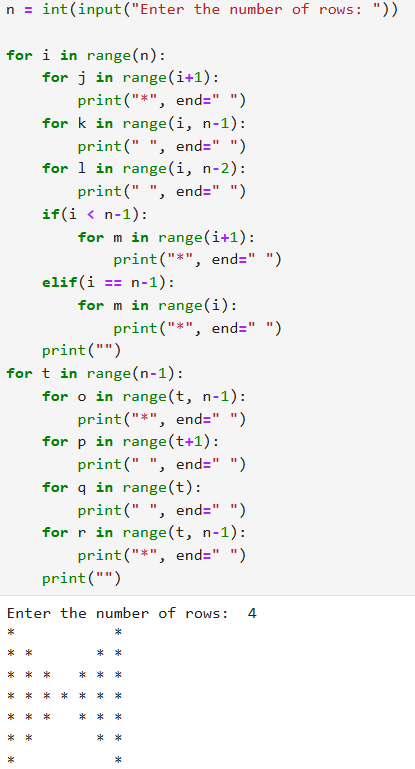
**PYRAMID**



**BUTTERFLY PATTERN**



**THIS IS IMPORTANT AS YAHA I HAVE TO USE IF-ELSE STATEMENT TOO!!**



**PYTHON NOTES (16)**

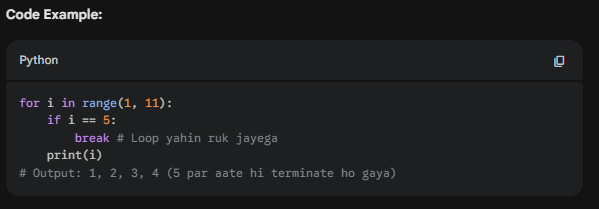
**🛑 Python Control Statements: Break, Continue & Pass**

In statements ka main kaam loops ke normal flow ko control ya alter karna hota hai.

**1. Break Statement**

break ka kaam hai loop ko turant khatam (**terminate**) kar dena. Jaise hi Python ko break milta hai, wo loop se bahar nikal aata hai, bhale hi loop ki condition bachi ho.

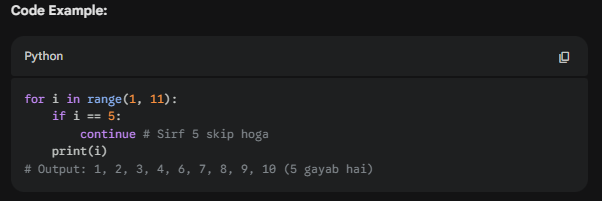
* **Logic:** Jab tumhe kisi specific condition par loop ko stop karna ho.
* **Real-world Example:** **Linear Search.** Maano tumhare database mein 1 lakh users hain aur tumhe "Rahul" ko dhundhna hai. Agar Rahul 100th number par mil gaya, toh baaki ke 99,900 users ko check karne ki zarurat nahi hai. Wahan break laga kar program ko fast banaya jata hai.



**2. Continue Statement**

continue loop ko khatam nahi karta, balki current **iteration** (chakkar) ko skip kar deta hai aur agle item par chala jata hai.

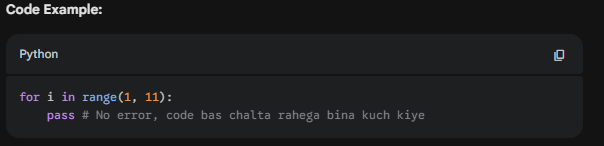
* **Logic:** Us specific step ke niche likha hua code execute nahi hoga, lekin loop agle number ke liye chalta rahega.
* **Real-world Example:** **E-commerce Website.** Maano tum Flipkart par products dikha rahe ho. Tum loop chalaoge saare products par, lekin agar koi product "Out of Stock" hai, toh tum continue use karoge taaki uska display wala code skip ho jaye aur agla product dikhe.



**3. Pass Statement**

pass ek **placeholder** ya "null operation" hai. Ye kuch nahi karta, bas error aane se bachata hai.

* **Logic:** Kabhi kabhi hume pata hota hai ki loop ya function banana hai, par uske andar ka logic hum baad mein likhna chahte hain. Python mein khali loop error deta hai, isliye hum pass likh dete hain.
* **Hinglish Meaning:** "Baad mein dekhunga, abhi aage badho."



**💡 Game Dev Context (For You):**

1. **Break:** Agar player ki health 0 ho jaye, toh break use karke game loop ko stop kiya ja sakta hai (Game Over).
2. **Continue:** Agar tum enemies ko update kar rahe ho aur koi enemy screen se bahar hai, toh continue use karke uska rendering logic skip kar do (Performance optimization).
3. **Pass:** Jab tum naya game level design kar rahe ho aur abhi functions ke naam likh rahe ho par code nahi likha, toh pass use karo taaki game crash na ho.

**Summary Checklist:**

* break: Loop ko "Tala" laga deta hai (Stop).
* continue: Current step ko "Skip" kar deta hai.
* pass: "Error" se bachane ke liye placeholder hai.

**PYTHON NOTES (17)**

**🛠️ Python Built-in Functions**

Functions basically ek "box" ki tarah hote hain: tum unhein kuch input dete ho, aur wo tumhe output dete hain.

**1. Basic Functions (Jo tum pehle se jaante ho)**

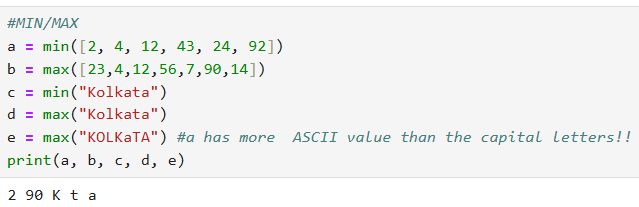
* **print()**: Screen par kuch bhi dikhane ke liye use hota hai.
* **input()**: User se data lene ke liye. Yaad rakhna, ye hamesha **string** format mein output deta hai.
* **type()**: Kisi variable ka data type (int, float, string, etc.) check karne ke liye.
* **Type Conversion**: int(), float(), str(), list(), tuple() — ek type ko dusre mein badalne ke liye.

**2. Mathematical Functions**

* **abs() (Absolute)**: Ye modulus ki tarah kaam karta hai. Negative number ko bhi positive kar deta hai (e.g., abs(-4) is 4).
* **pow() (Power)**: Do numbers leta hai, base aur exponent. pow(2, 3) ka matlab hai 2^3 = 8.
* **round()**: Decimal values ko nearest integer ya specific decimal places tak round karne ke liye. Jaise round(3.1415, 2) output dega 3.14.
* **divmod()**: Ye ek saath do cheezein return karta hai: **Quotient** (bhagphal) aur **Remainder** (sheshphal). divmod(5, 2) output dega (2, 1).

**3. Sequence & Aggregation Functions**

* **min() aur max()**: Kisi bhi list, tuple ya string mein se sabse choti ya badi value nikalne ke liye. Strings mein ye **ASCII values** ke basis par kaam karta hai.



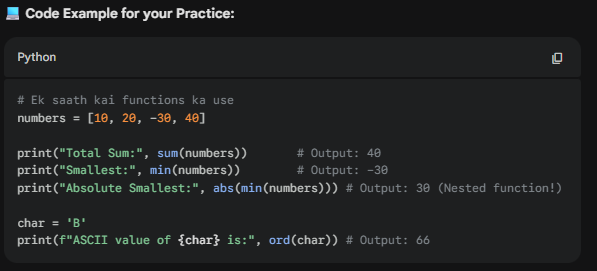
* **sum()**: Kisi list ya tuple ke saare numbers ka total (addition) nikalne ke liye.
* **len()**: Kisi bhi string, list, ya dictionary ki total length (kitne items hain) batata hai.

**4. Conversion & Memory Functions**

* **bin(), oct(), hex()**: Kisi number ko Binary, Octal, ya Hexadecimal format mein badalne ke liye.
* **id()**: Kisi variable ka memory address (location) batata hai. Ye har computer par alag ho sakta hai.
* **ord()**: Kisi single character ka ASCII ya Unicode value batata hai. Jaise ord('A') output dega 65.

**5. Documentation Function**

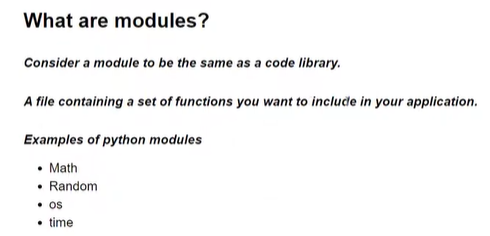
* **help()**: Agar tum kisi bhi function ke baare mein bhool jao, toh help(function\_name) likho, Python tumhe uska poora documentation dikha dega.



**💡 Game Dev Tip (For You):**

* **abs()**: Player aur enemy ke beech ka distance hamesha positive rakhne ke liye kaam aata hai.
* **min()/max()**: Player ki health ko ek limit mein rakhne ke liye use hota hai (e.g., health 100 se upar na jaye).
* **round()**: Jab physics calculations mein lambi decimal values aayein, toh unhein UI par dikhane ke liye round kar liya jata hai.

**PYTHON NOTES (17)**



**📦 Python Built-in Modules: Detailed Notes**

**Module Kya Hai?** Module ek Python file hoti hai jisme pehle se likhe huye functions hote hain. Jaise baki languages mein hum "Library" bolte hain, Python mein usey "Module" kehte hain. Inka main maqsad **Code Reusability** hai taaki humein har cheez ka code khud se na likhna pade.

**1. Modules ko use kaise karein?**

* **List check karna:** Apne system mein available modules dekhne ke liye help('modules') type kar sakte ho.
* **Import karna:** Kisi module ko use karne ke liye import keyword ka istemal hota hai (e.g., import math).
* **Functions dekhna:** module\_name. likh kar Tab press karne se uske saare functions ki list mil jati hai.

**2. Important Modules (Video Examples)**

**A. Math Module (math)**

* **math.pi**: Pi ki value (3.14...) deta hai.
* **math.factorial(n)**: Kisi number ka factorial nikalta hai.
* **math.ceil(x)**: Hamesha upar wali value deta hai (e.g., 6.3 -> 7).
* **math.floor(x)**: Hamesha niche wali value deta hai (e.g., 6.9 -> 6).

****

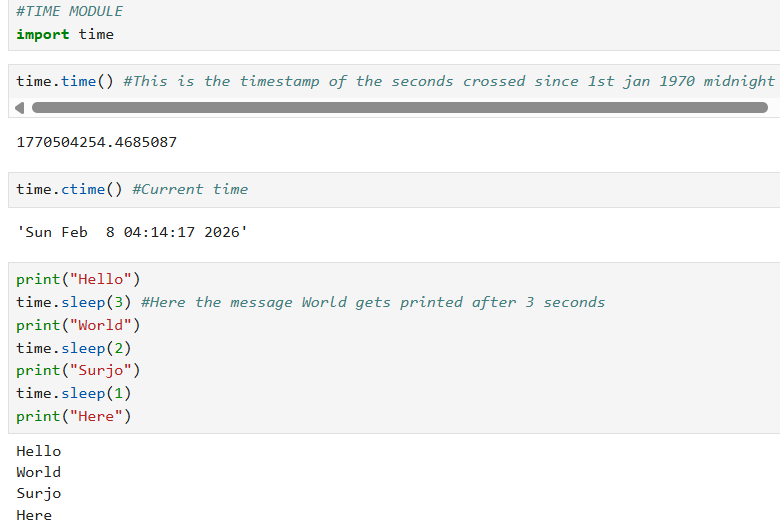
**B. Random Module (random)**

* **random.randint(a, b)**: Do numbers ke beech ek random integer deta hai.
* **random.shuffle(list)**: Kisi bhi list ke items ko mix (shuffle) kar deta hai (jaise cards shuffle karte hain).

****

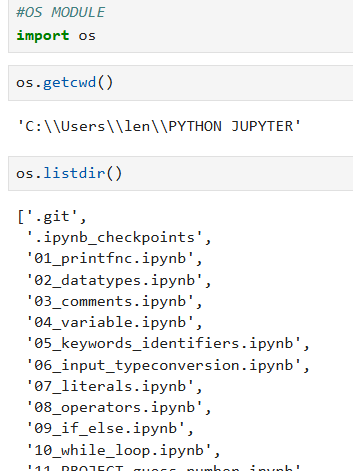
**C. Time Module (time)**

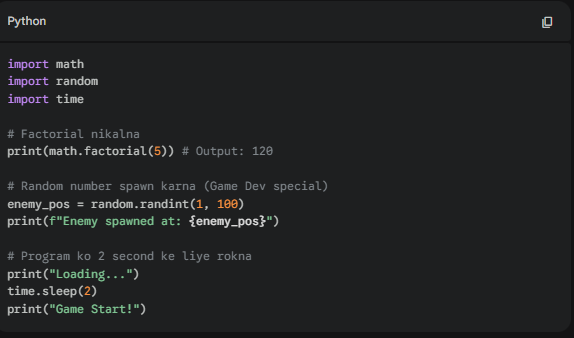
* **time.time()**: 1st Jan 1970 se ab tak ke total seconds (timestamp) batata hai.
* **time.ctime()**: Current date aur time readable format mein dikhata hai.
* **time.sleep(n)**: Program ko n seconds ke liye rok (pause) deta hai.

****

**D. OS Module (os)**

* **os.getcwd()**: Batata hai ki tum abhi kis folder (directory) mein kaam kar rahe ho.
* **os.listdir()**: Us folder ki saari files ki list dikhata hai.



****

**💡 Game Dev Insight:**

Tum game developer banna chahte ho, toh ye modules tumhare best friends hain:

* **random**: Enemies ko random positions par spawn karne ya loot drop rates set karne ke liye.
* **math**: Player ki movement, jumping physics (parabola), aur angles (trigonometry) calculate karne ke liye.
* **time**: Cooldown periods (jaise weapon reload) ya game loop ki speed control karne ke liye.

**PYTHON NOTES (17)**

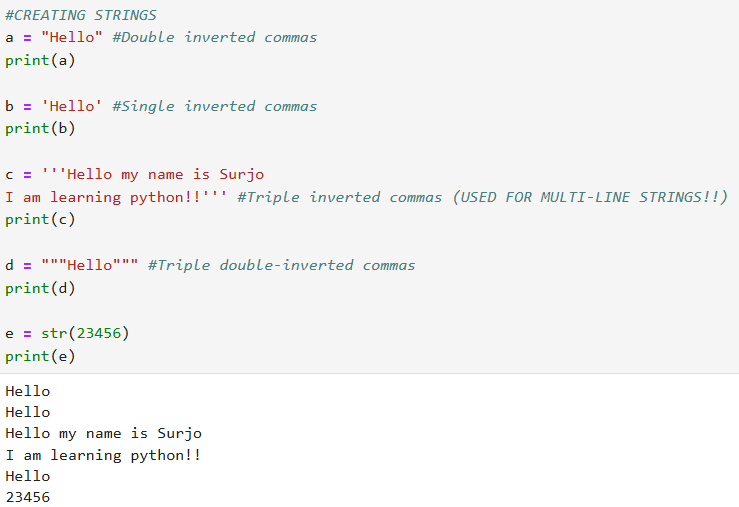
**🧵 Strings in Python: Introduction & Creation**

**String Kya Hai?** String characters ka ek sequence hota hai. Python mein iska matlab hai "**Sequence of Unicode characters**". Machine sirf numbers samajhti hai, isliye characters ko pehle Unicode numbers mein aur phir binary mein badla jata hai.

**1. Strings Create Kaise Karein?**

Python mein strings banane ke kai tarike hain:

* **Single Quotes (' '):** s = 'Hello'.
* **Double Quotes (" "):** s = "Hello".
  + **Zarurat:** Agar tumhare sentence ke beech mein single quote hai (jaise: It's raining), toh bahar double quotes use karna chahiye taaki Python confuse na ho.
* **Triple Quotes (''' ''' ya """ """):** Ye **Multi-line strings** ke liye use hote hain. Agar tumhe poora paragraph ya blog post likhna hai jo kai lines mein ho, toh triple quotes best hain.
* **str() Function:** Type conversion ke zariye bhi string banai ja sakti hai (e.g., str(123))



**💡 Game Dev Context:**

* **User Interface (UI):** Game mein "Start Game", "Score: 100", ya "Game Over" jaise saare messages strings hi hote hain.
* **Dialogues:** Story-driven games mein characters ke beech ki saari baatein (scripts) multi-line strings mein store ki jati hain.
* **Input Handling:** User jab apna name type karta hai, toh wo input hamesha string format mein hi milta hai.

**IMPORTANT**

**ASCII VS UNICODE**

**🔠 ASCII vs Unicode: Why the Change?**

**1. ASCII (American Standard Code for Information Interchange)**

ASCII purana standard hai jo 1960s mein banaya gaya tha.

* **Capacity**: Ye sirf 7-bit ya 8-bit ka hota tha, jisme sirf **128 se 256 characters** hi aa sakte the.
* **Characters**: Isme sirf English alphabets (A-Z, a-z), numbers (0-9), aur kuch basic symbols hote the.
* **Limitation**: Jab programming puri duniya mein phaili, toh China, Japan, ya India (Hindi) jaise languages ke characters ASCII mein fit nahi ho pa rahe the.

**🔢 ASCII: 7-bit vs 8-bit ka Khel**

ASCII shuruat mein **7-bit** ka tha.

* **7-bit ka matlab:** 2^7 = 128 unique patterns (yaani 0 se 127 tak numbers). Isme English ke saare alphabets, numbers, aur punctuation aa jate the.
* **8-bit (Extended ASCII):** Jab computers thode advance huye, toh 1 bit aur jod diya gaya (2^8 = 256 patterns). Is extra jagah ka use special characters (jaise mathematical symbols) ke liye kiya gaya.

**2. Unicode (Universal Character Set)**

Unicode ek modern standard hai jo duniya ki har language ke har character ko ek unique number deta hai.

* **Capacity**: Isme **1 lakh se zyada characters** ki jagah hai.
* **Scope**: Isme Hindi, Arabic, Chinese, emojis, aur yahan tak ki mathematical symbols bhi shamil hain.
* **Bits**: Unicode 8, 16, ya 32 bits use kar sakta hai (jaise UTF-8 jo sabse popular hai).

**🌐 Unicode: 8, 16, aur 32 Bits (UTF)**

Unicode sirf ek list hai jisme har character (chahe wo Hindi ho ya Emoji) ko ek unique number (Code Point) diya gaya hai. Lekin computer us number ko memory mein kaise "save" karega, uske liye **UTF (Unicode Transformation Format)** banaya gaya.

**1. UTF-8 (Sabse Popular)**

* Ye **Variable Length** encoding hai.
* Agar character ASCII wala hai (English), toh sirf **8 bits (1 byte)** lega.
* Agar koi complex character hai (jaise Hindi), toh ye **16 se 32 bits** tak badh sakta hai.
* **Fayda:** Memory bohot save karta hai kyunki English text kam jagah leta hai.

**2. UTF-16**

* Isme har character minimum **16 bits (2 bytes)** leta hi hai.
* Windows aur Java internally isse kaafi use karte hain.

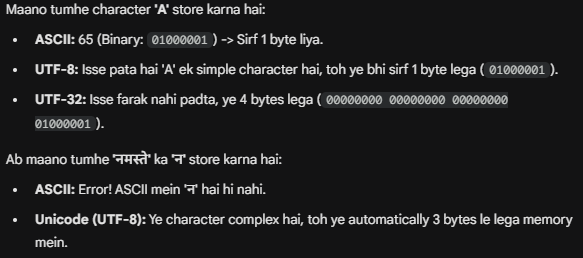
**3. UTF-32**

* Ye **Fixed Length** hai. Har character **32 bits (4 bytes)** lega.
* **Nuksan:** Agar tum sirf "ABC" likhoge, toh bhi ye 12 bytes lega, jabki UTF-8 mein ye sirf 3 bytes leta. Ye memory bohot waste karta hai.

**❓ Kyu Change Karna Pada? (The Problems with ASCII)**

Change karne ke main reasons ye the:

* **Global Communication**: ASCII sirf English-centric tha. Dusri languages (jaise Spanish ke accents ya Indian Devnagari) ke liye koi standard nahi tha.
* **Conflicts**: Alag-alag countries ne apne khud ke 8-bit standards banana shuru kar diye the, jisse ek computer ka text dusre computer par "Garbage characters" dikhata tha.
* **Standardization**: Ek aise "Universal" system ki zarurat thi jo poori duniya mein same rahe, taaki internet aur apps par text sahi se dikhe.



**PYTHON NOTES (18)**

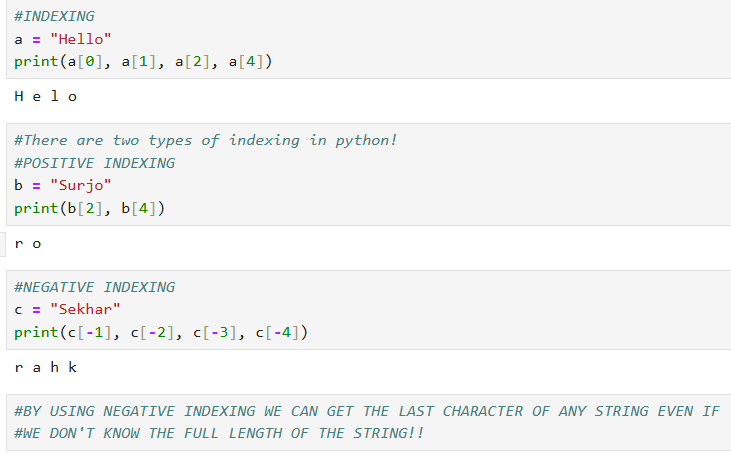
**✂️ Indexing and Slicing in Python Strings (Part 2)**

Strings characters ka sequence hote hain, isliye har character ki ek fixed position hoti hai jise **Index** kehte hain.

**1. Indexing (Ek single character nikalna)**

Python mein indexing do tarah ki hoti hai:

* **Positive Indexing:** Ye **Left se Right** jati hai aur **0** se shuru hoti hai.
  + Example: "HELLO" mein H=0, E=1, L=2, L=3, O=4.
* **Negative Indexing:** Ye **Right se Left** jati hai aur **-1** se shuru hoti hai.
  + Example: "HELLO" mein O=-1, L=-2, L=-3, E=-4, H=-5.
  + **Faayda:** Agar tumhe string ki length nahi pata par last character chahiye, toh seedha [-1] use kar lo.



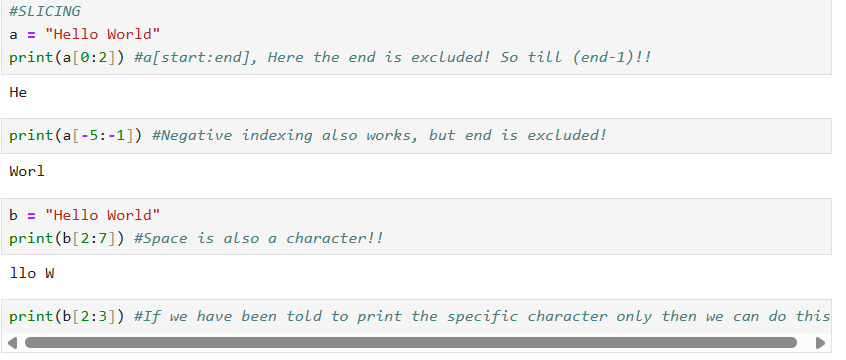
**2. Slicing (Ek chota part/substring nikalna)**

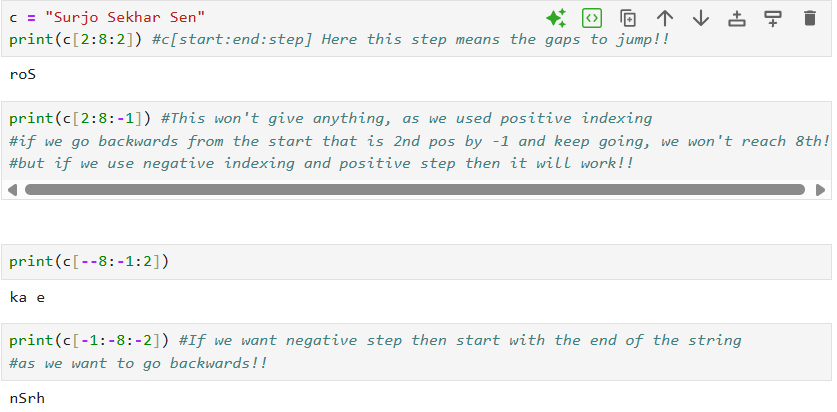
Slicing se hum ek range ke characters nikal sakte hain.

* **Syntax:** string[start : stop : step]
* **Start:** Jahan se shuru karna hai (Included).
* **Stop:** Jahan rukna hai (**Excluded** - yaani stop index se ek pehle tak hi lega).
* **Step:** Kitne ka gap chahiye (Default 1 hota hai).

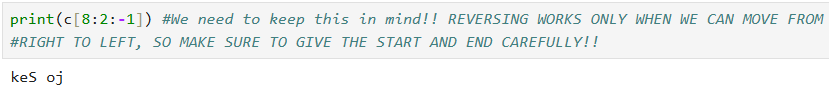
**Variaitions of Slicing:**

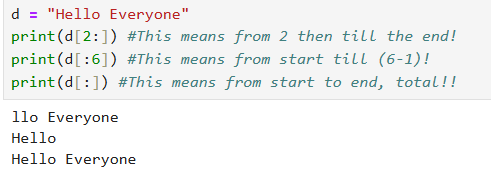
* s[0:5] : 0 se 4 index tak ke characters.
* s[2:] : 2 index se lekar end tak saare.
* s[:3] : Shuru se lekar index 2 tak.
* s[:] : Poori ki poori string.
* **Steps ka use:** s[0:10:2] — 0 se 9 tak har alternate character (1-1 skip karke).

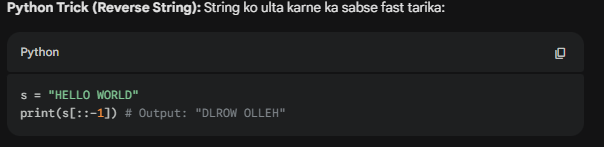




**IMPORTANT CONCEPT: ABOUT REVERSING**







**💡 Game Dev Context (For You):**

* **Parsing Input:** Maano player ne command di /teleport 100 200. Tum slicing use karke /teleport ko alag kar sakte ho aur coordinates ko alag.
* **Inventory Display:** Agar kisi item ka naam bohot bada hai, toh tum slicing use karke usey chota dikha sakte ho (jaise item\_name[:10] + "...").
* **Reversing Effects:** Games mein puzzles ke liye kabhi-kabhi text ko reverse dikhaya jata hai, wahan [::-1] kaam aata hai.

**PYTHON NOTES (18)**

**🚫 Editing and Deleting Strings in Python (Part 3)**

Python mein strings **Immutable** hoti hain. 'Immutability' ka matlab hai ki aap ek baar string create karne ke baad uske kisi bhi hisse ko change nahi kar sakte.

**1. Editing Strings (Kyoon nahi hota?)**

Agar aapke paas ek string hai s = "Hello" aur aap chaho ki H ki jagah X aa jaye (s[0] = 'X'), toh Python aapko **TypeError** dega: *"str object does not support item assignment"*.

* **Reason:** Strings immutable hain, isliye aap specific index par jaakar value change nahi kar sakte.
* **Re-assignment:** Aap poori string ko naye data ke saath badal sakte ho (e.g., s = "World"), lekin purani string ke beech mein kuch change nahi kar sakte.

**2. Adding Characters**

Editing ki tarah hi, aap existing string mein naye characters insert bhi nahi kar sakte.

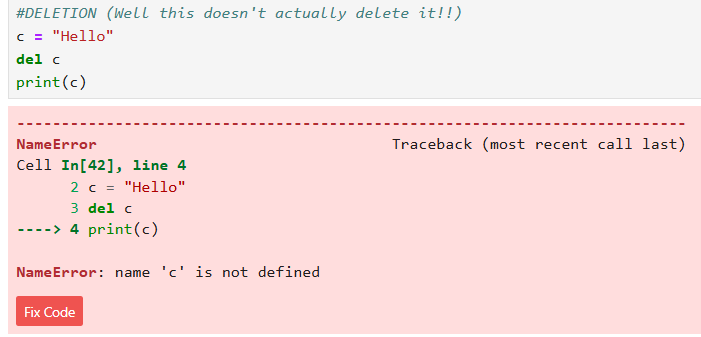
****

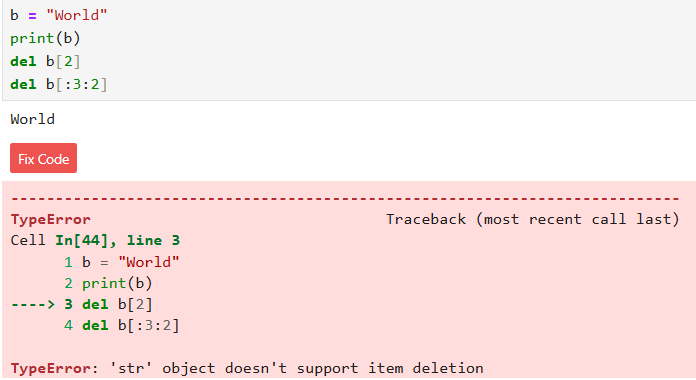
**IMPORTANT:**

****

**3. Deleting Strings**

* **Full Deletion:** Aap del keyword use karke poori string variable ko memory se uda sakte ho (e.g., del s). Iske baad agar aap s ko print karoge, toh error aayega kyunki wo variable ab exist hi nahi karta.
* **Partial Deletion:** Aap string ka ek character ya substring delete nahi kar sakte (e.g., del s[0]). Isse wahi same error aayega: *"str object does not support item deletion"*.





**💡 Game Dev Context (For You):**

* **Memory Safety:** Immutability ki wajah se strings memory mein safe rehti hain. Game development mein agar aapka "PlayerName" ek baar set ho gaya, toh galti se code ka koi doosra part use change nahi kar payega jab tak aap poora re-assign na karo.
* **Level Names:** Maan lo level ka naam hai "DarkForest". Agar aap use change karna chahte ho "FireCave" mein, toh aapko naya string assign karna hoga, Forest ke characters ko edit karke Cave nahi banaya ja sakta.

**PYTHON NOTES (19)**

**🧵 String Operations in Python (Part 4)**

Python mein hum saare mathematical operators strings par use nahi kar sakte. Sirf niche diye gaye operators hi allowed hain:

**1. Arithmetic Operators (Sirf + aur \*)**

* **Addition (+)**: Isse **Concatenation** kehte hain. Ye do strings ko aapas mein jod deta hai.
  + **Code**: "Hello" + "World" ban jayega "HelloWorld".
  + Tum kitne bhi strings ko + laga kar jod sakte ho.
* **Multiplication (\*)**: Isse **String Replication** kehte hain. Ye ek string ko utni baar repeat karta hai.
  + **Code**: "\*" \* 5 output dega \*\*\*\*\*.
  + Ye patterns banane mein bohot kaam aata hai.



**2. Relational Operators (==, !=, >, <, etc.)**

Ye operators do strings ko compare karte hain. Inka comparison **Lexicographical** (Dictionary order) hota hai.

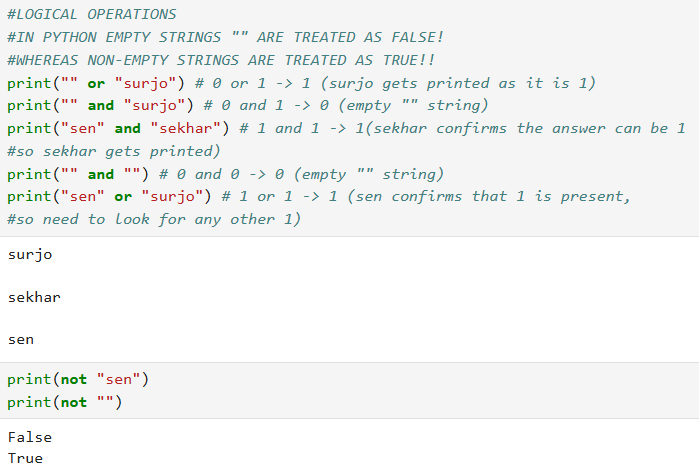
* **Dictionary Logic**: Jo word dictionary mein baad mein aata hai, wo bada mana jata hai.
  + **Example**: "Mumbai" > "Pune" ka output **False** aayega kyunki 'P' dictionary mein 'M' ke baad aata hai, isliye "Pune" bada hai.
* **Capital vs Small**: Small letters ki ASCII value badi hoti hai, isliye 'a' > 'A' hamesha **True** hota hai.



**3. Logical Operators (and, or, not)**

Strings ke saath inka logic simple hai:

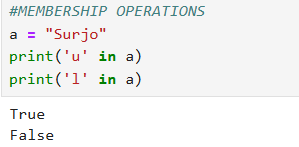
* **Empty String ("")**: Python ise **False** maanta hai.
* **Non-empty String**: Python ise **True** maanta hai.
* **Example**: "Hello" and "World" mein output "World" aayega kyunki pehla 'True' hai toh Python dusra check karta hai.



**4. Membership Operators (in, not in)**

Ye check karne ke liye hota hai ki koi character ya word kisi string ke andar hai ya nahi.

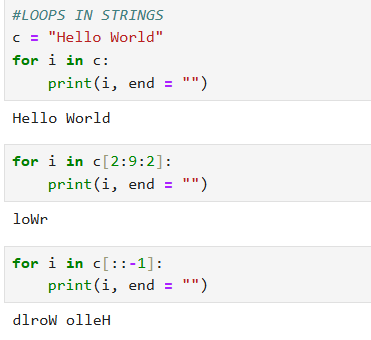
* **Example**: "H" in "Hello" output dega **True**.



**5. Loops on Strings**

Tum string ke har ek character par loop chala sakte ho.

* **Code**: for i in "Hello": print(i) — Ye 'H', 'e', 'l', 'l', 'o' ko alag-alag lines mein print karega.



**💡 Game Dev Context (For You):**

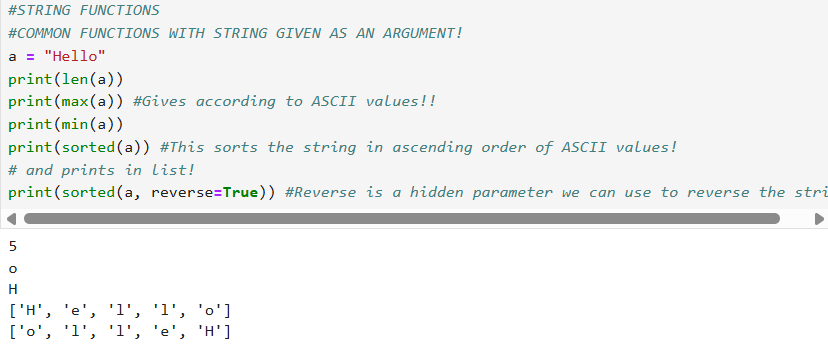
* **UI Text**: Player ka naam aur level jodne ke liye + ka use hota hai.
* **Cheat Codes**: Agar tum check karna chahte ho ki player ne sahi code type kiya hai, toh in ya == operator kaam aayega.
* **Level Design**: Border ya patterns print karne ke liye \* operator kaafi useful hai.

**PYTHON NOTES (20)**

**🧵 String Functions in Python (Part 5)**

**1. Common Functions (Jo sab par kaam karte hain)**

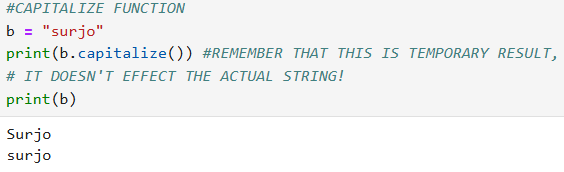
* **len()**: String ki total length (kitne characters hain) batata hai.
* **max()**: String mein sabse bada character (ASCII value ke hisaab se) batata hai.
* **min()**: String mein sabse chota character batata hai.
* **sorted()**: String ke saare characters ko list ke form mein sort (ascending order) kar deta hai. reverse=True karne par ye descending order mein sort karega.



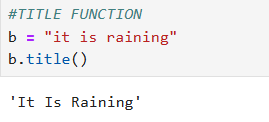
**2. String Specific Functions (Sirf Strings ke liye)**

**Capitalization & Case Functions:**

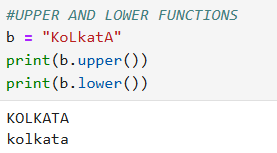
* **capitalize()**: Sirf pehle character ko capital karta hai aur baaki sabko small.



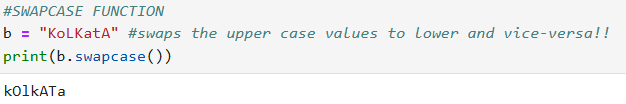
* **title()**: Har word ke pehle character ko capital kar deta hai (Jaise book ka title hota hai).



* **upper()**: Saare characters ko UPPERCASE mein badal deta hai.
* **lower()**: Saare characters ko lowercase mein badal deta hai.



* **swapcase()**: Upper ko lower aur lower ko upper mein badal deta hai.

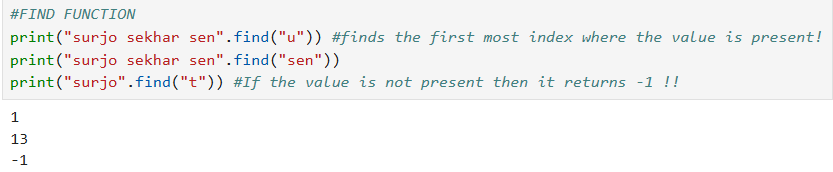


**Search & Count Functions:**

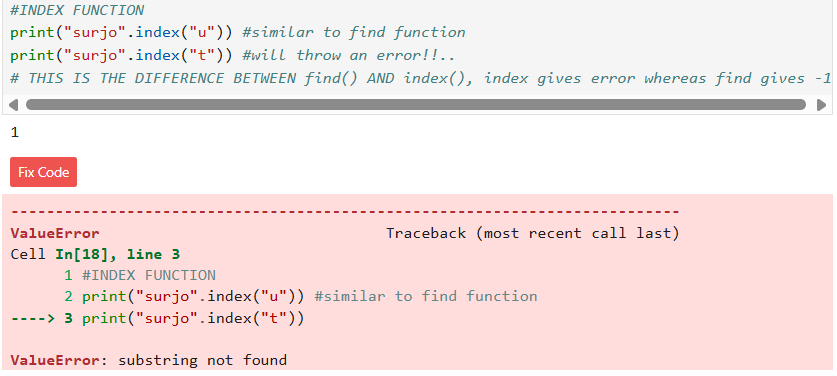
* **count()**: Kisi character ya substring ka total frequency (kitni baar aaya hai) batata hai.



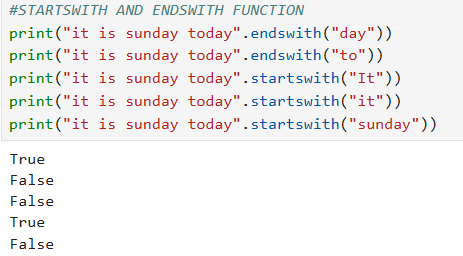
* **find()**: Kisi substring ka index batata hai. Agar nahi milta, toh ye **-1** return karta hai.



* **index()**: find() ki tarah kaam karta hai, par agar substring nahi milti toh ye **Error** throw karta hai.

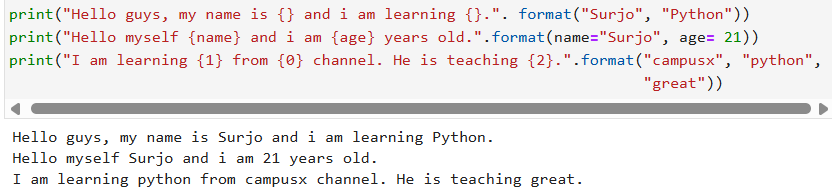


* **startswith() / endswith()**: Check karta hai ki string kisi specific word se shuru ya khatam ho rahi hai ya nahi (Output: True/False).



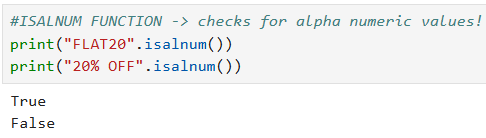
**Formatting Function:**

* **format()**: String ke beech mein dynamic values daalne ke liye use hota hai. Ye login screens ya profiles mein user ka naam dikhane ke liye bohot kaam aata hai.

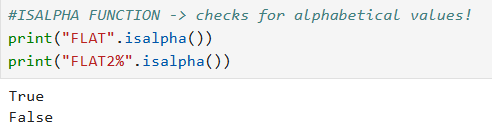
****

**Validation Functions (True/False):**

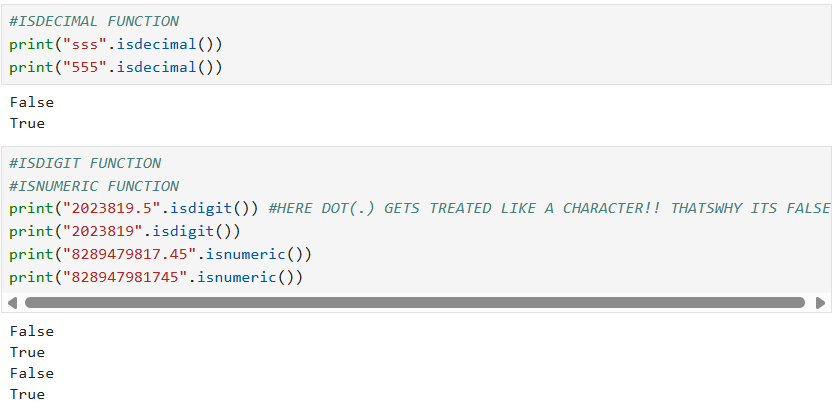
* **isalnum()**: Check karta hai ki string sirf Alphanumeric (alphabet + numbers) hai ya nahi.



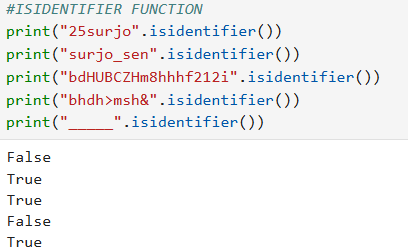
* **isalpha()**: Sirf Alphabets ke liye check karta hai.



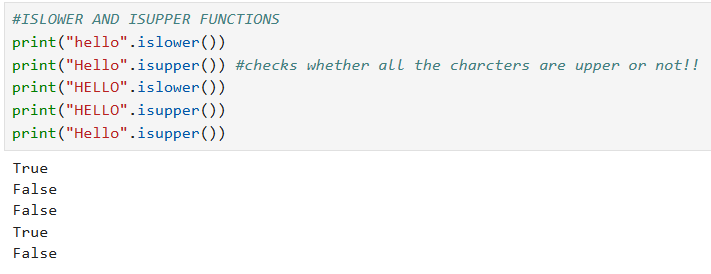
* **isdigit()**: Check karta hai ki sirf digits hain ya nahi.



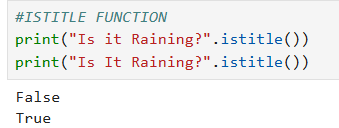
* **isidentifier()**: Check karta hai ki kya ye string ek valid variable name (identifier) ban sakti hai.



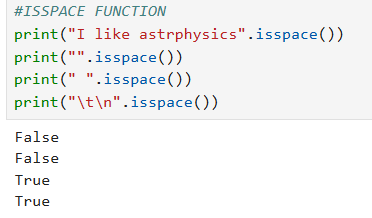
**ISLOWER() AND ISUPPER() FUNCTIONS**



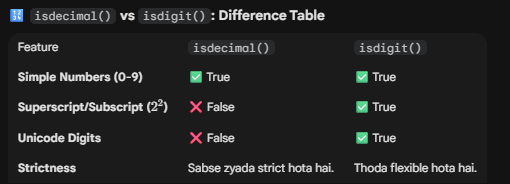
**ISTITLE() FUNCTION**

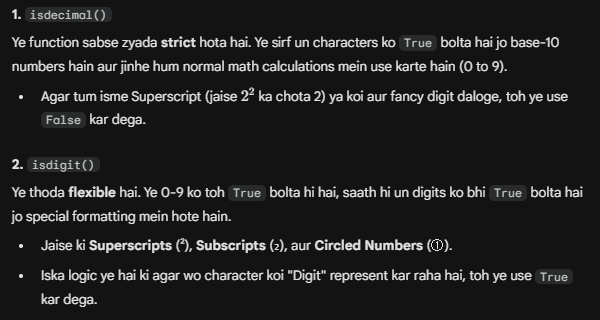


**ISSPACE() FUNCTION**



**IMPORTANT**: **ISDECIMAL VS ISDIGIT**

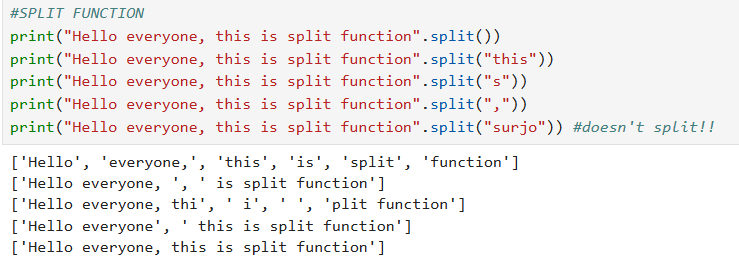






**Split & Join Functions:**

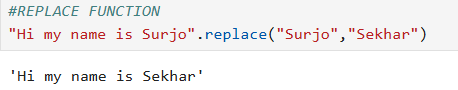
* **split()**: Ek lambi string ko tod kar words ki **List** bana deta hai.



* **join()**: Ek list ke words ko jod kar wapas **String** bana deta hai. URL formatting mein ye bohot useful hai.



**REPLACE FUNCTION**

****

**Cleaning Function:**

* **strip()**: String ke shuruat aur aakhiri mein bache huye faltu **Spaces** ko hata deta hai.



**💡 Game Dev Context:**

* **strip()**: Jab user "Player Name" type kare aur galti se space chhor de, toh usey saf karne ke liye use hota hai.
* **split()**: Game commands parse karne ke liye (e.g., "/move up 10" ko split karke direction aur speed nikalna).
* **upper()/lower()**: Jab user se "YES/NO" input lo, toh input ko lower() karke check karo taaki "Yes", "YES", ya "yes" sab par game sahi chale.

**PYTHON NOTES (21)**

**📦 Python Lists: Complete Explanation**

**Definition:** List ek aisa data type hai jahan tum ek saath multiple items (cheezein) store kar sakte ho.

**1. List vs Arrays (Main Differences)**

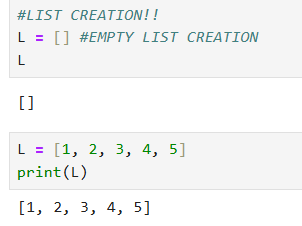
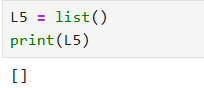
* **Heterogeneous:** Arrays mein sirf ek hi tarah ka data (jaise sirf numbers) aa sakta hai, lekin List mein tum **numbers, strings, aur floats** sab ek saath rakh sakte ho.
* **Memory:** Arrays memory mein ek ke baad ek (continuous) jagah lete hain,

**Lists mein aisa zaruri nahi hai**.

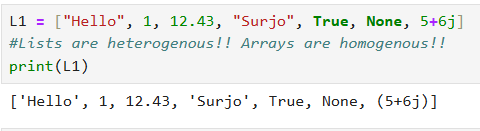
* **Speed:** Arrays fast hote hain, par Lists **Programmer Friendly** hoti hain (kaam karna aasaan hai).

**2. Creating Lists**

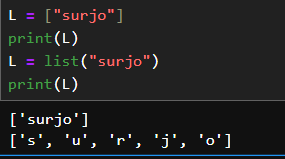
* **Empty List:** l = [] ya l = list()

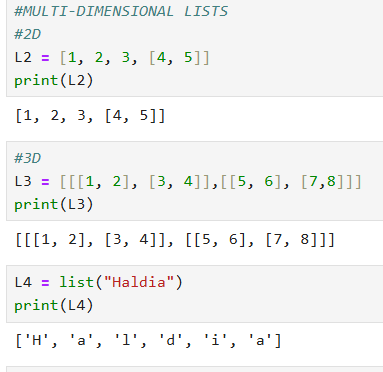
* **Heterogeneous List:** l = [1, "Hello", 3.4]



**##DIFFERENCE IN LIST CREATION USING FUNCTION list() AND NORMAL ASSIGNING!!**

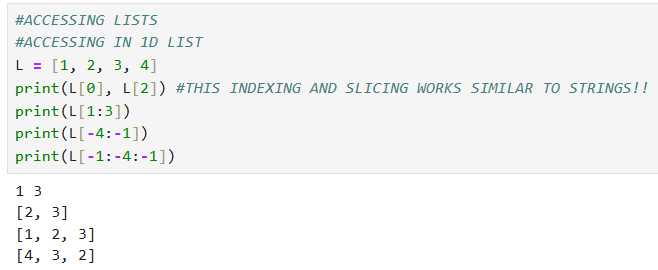


* **Multi-dimensional (2D/3D):** List ke andar list.
  + 2D: [[1,2], [3,4]]
  + 3D: [[[1,2], [3,4]], [[5,6], [7,8]]]

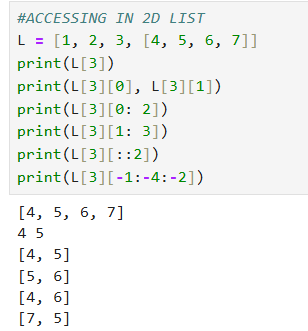


**3. Accessing Items (Indexing & Slicing)**

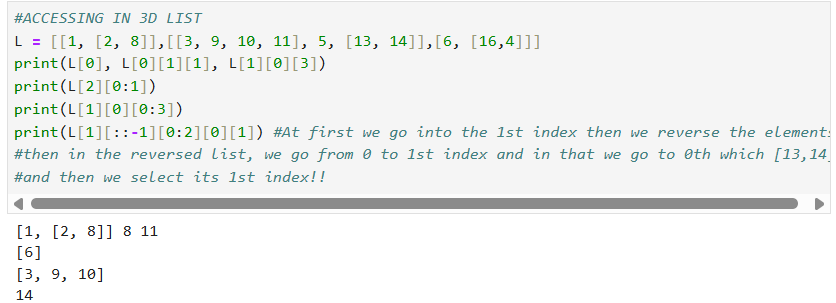
* **Indexing:** Same strings ki tarah. l[0] se pehla item, l[-1] se aakhiri item.



* **2D/3D Access:** Jitne dimensions, utne brackets.
  + Example: l[1][0] ka matlab dusri list ka pehla item.



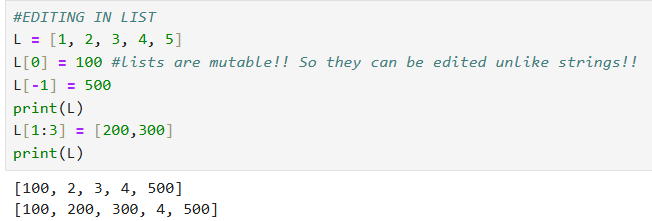
**V.V IMPORTANT**



**4. Editing & Adding Items (Lists are Mutable)**

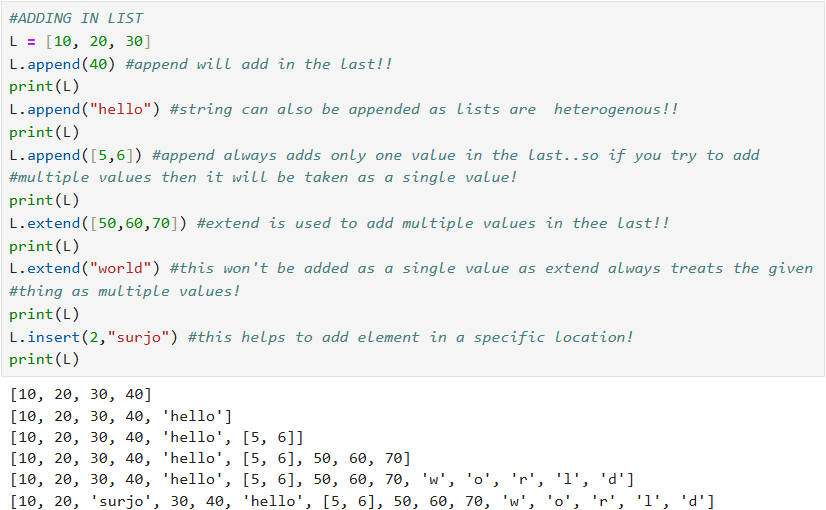
Strings badli nahi ja sakti thi, par Lists ko tum **edit** kar sakte ho.

* **Edit:** l[0] = 100 (Pehla item 100 ban jayega).



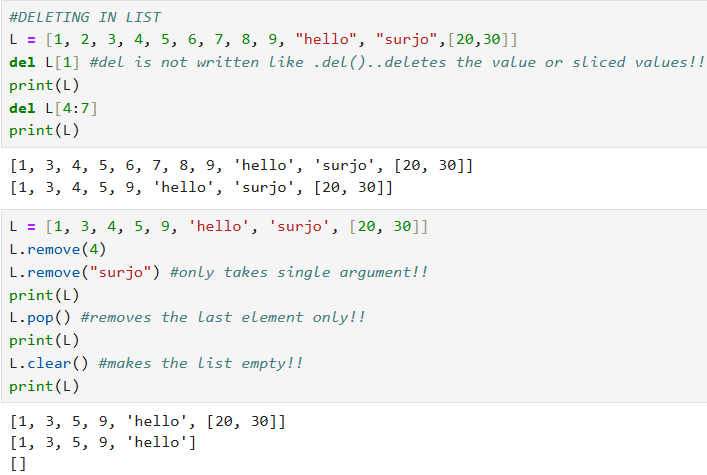
 **THIS EDITED AND EVEN ADDED VALUES!!**

* **Add Items:**
  1. **append(item)**: List ke **aakhiri** mein sirf ek item jodne ke liye.
  2. **extend([list])**: Ek saath multiple items jodne ke liye.
  3. **insert(index, item)**: Kisi specific position par item daalne ke liye.



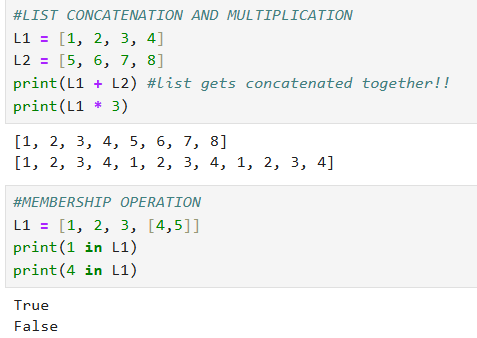
**5. Deleting Items**

* **del l[0]**: Index ke basis par delete karna.
* **remove(item)**: Item ke naam se delete karna (agar index nahi pata).
* **pop()**: Hamesha aakhiri item ko uda deta hai.
* **clear()**: Poori list khali kar deta hai (par variable rehta hai).

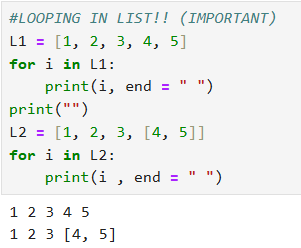


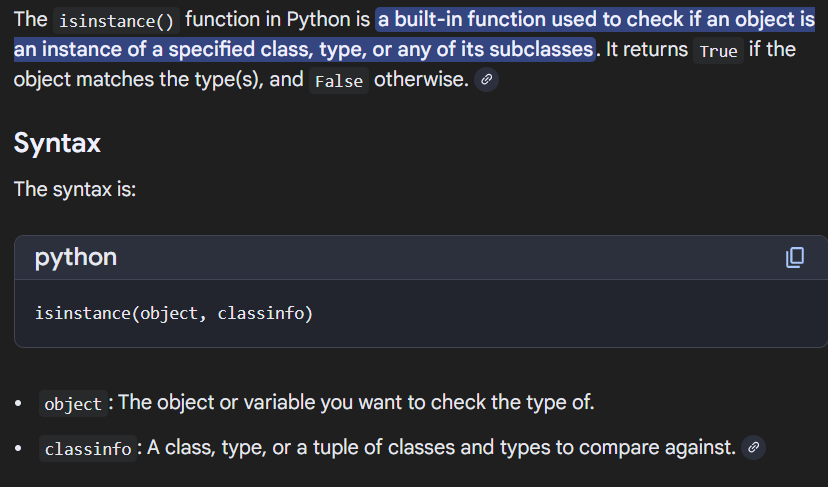
**6. Important Operations & Functions**

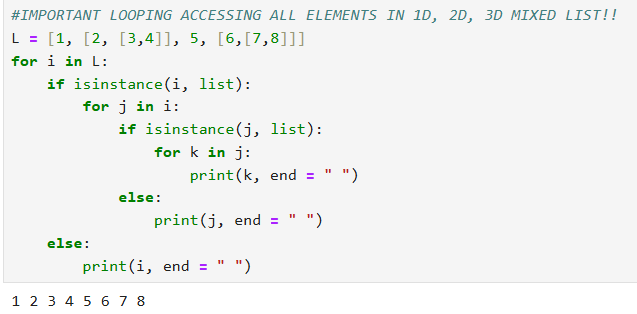
* **Concatenation (+)**: Do lists ko jodna.
* **Multiplication (\*)**: List ko repeat karna.
* **Membership (in)**: Check karna ki koi cheez list mein hai ya nahi.

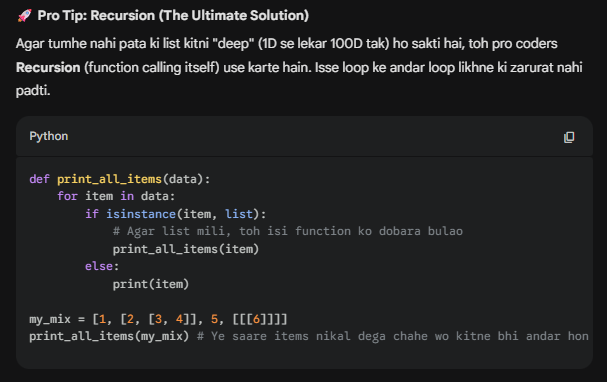


**V.V IMPORTANT (LOOPING IN LIST)**

****

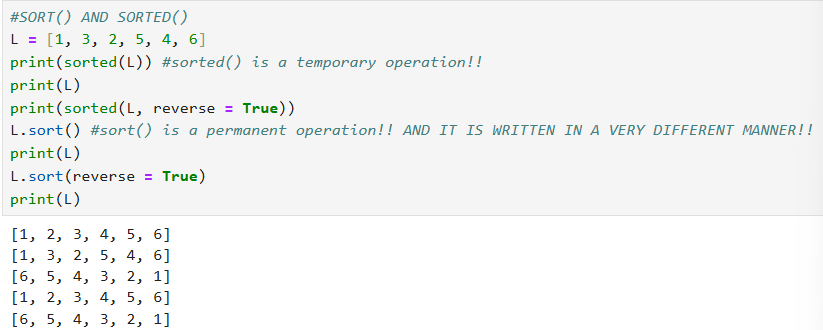
****

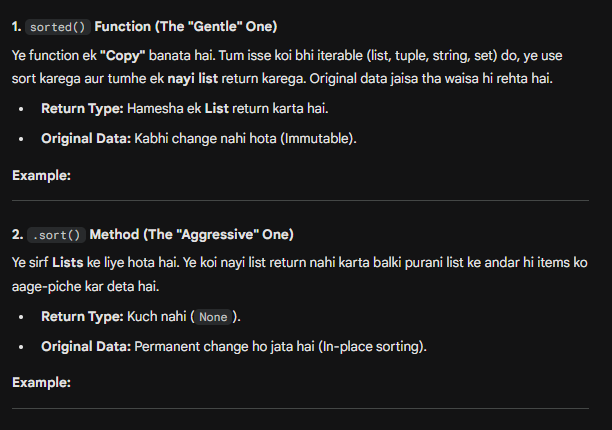
****

****

* **sort() vs sorted()**: sort() original list ko badal deta hai, jabki sorted() ek naya sorted list deta hai.

**IMPORTANT (SORT() VS SORTED())**





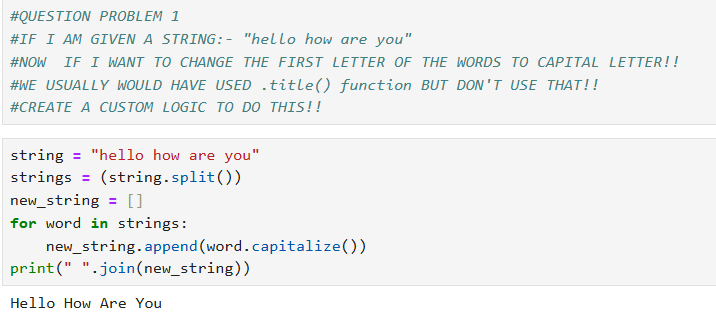
**SOME OTHER FUNCTIONS WHICH ARE USED WITH LIST!!**



**💻 Game Dev Case Study (From Video Logic):**

Video ke aakhir mein kuch problems solve kiye gaye hain jo tumhare logic ke liye best hain:

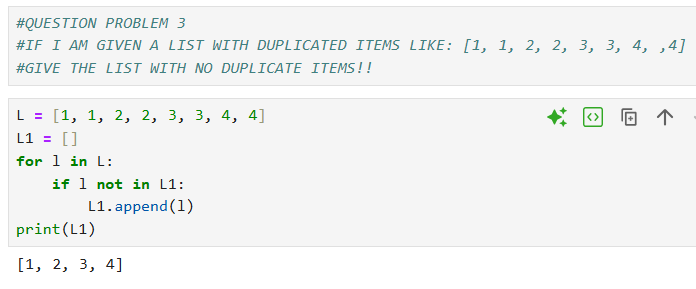
1. **Remove Duplicates:** Agar game mein player galti se ek hi item do baar utha le aur tumhe duplicate hatana ho, toh ek naya list banao aur if item not in new\_list wala logic use karo.
2. **Email Parsing:** split() function se email se username nikalna.
3. **QUESTION PRACTICE**



1. **QUESTION PRACTICE**



1. **QUESTION PRACTICE**



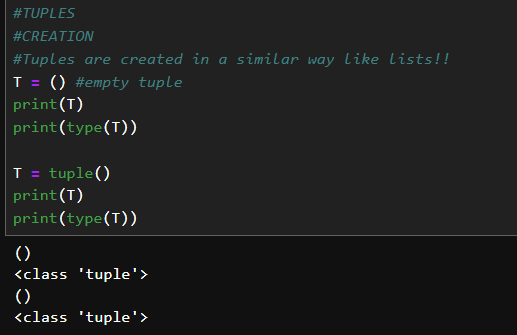
**PYTHON NOTES (22)**

**📦 Python Tuples**

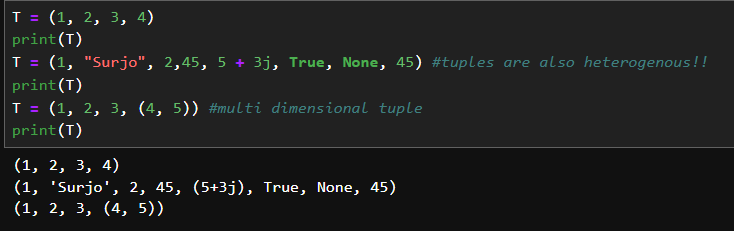
**Definition:** Tuples ek aisa data type hai jo Lists ki tarah hi multiple items store karta hai, lekin ek bohot bade twist ke saath—ye **Immutable** hote hain.

**1. Creating Tuples**

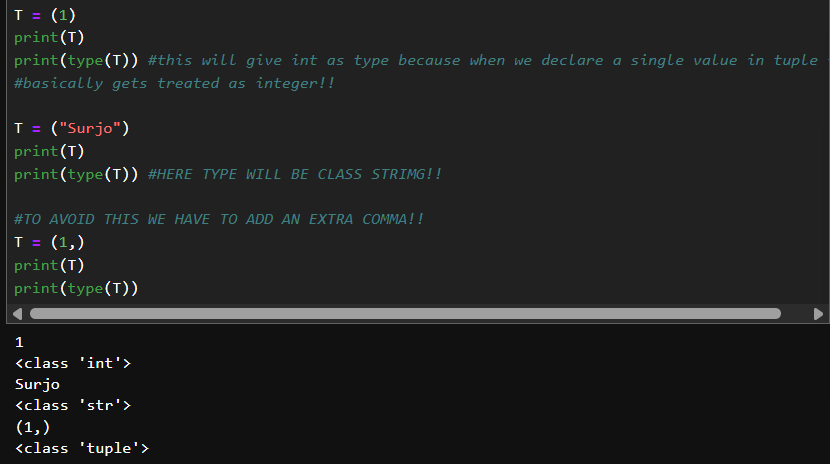
* **Syntax:** Tuples hamesha **Parentheses ()** mein likhe jate hain (jabki lists square brackets [] mein hote the).
* **Empty Tuple:** t = ()



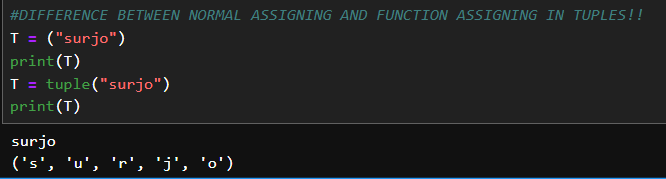
* **Types:** Ye bhi **Heterogeneous** hote hain, yani isme number, string, float sab ek saath aa sakte hain.

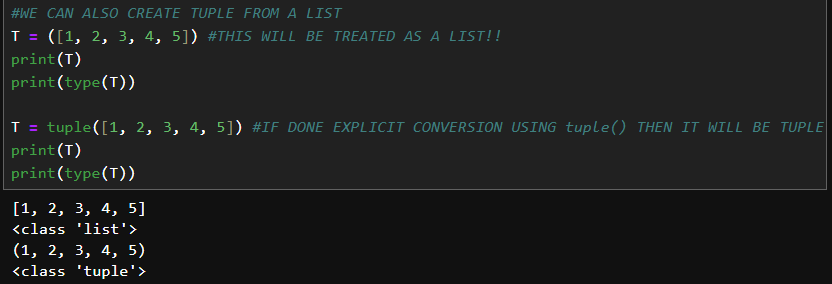


* **Single Item Tuple (The Trap):** Agar tum sirf t = (5) likhoge, toh Python ise integer maanta hai. Single item tuple banane ke liye hamesha **Comma** lagana padta hai: t = (5,).



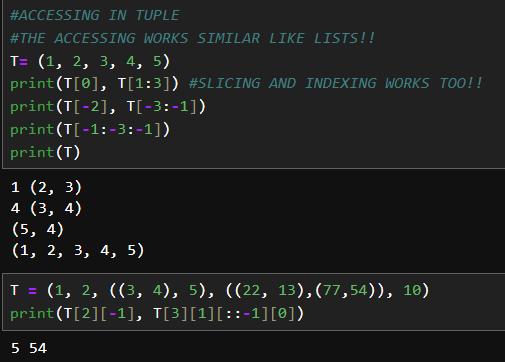
**##DIFFERENCE IN TUPLE CREATION USING FUNCTION tuple() AND NORMAL ASSIGNING!!**





**2. Accessing Items**

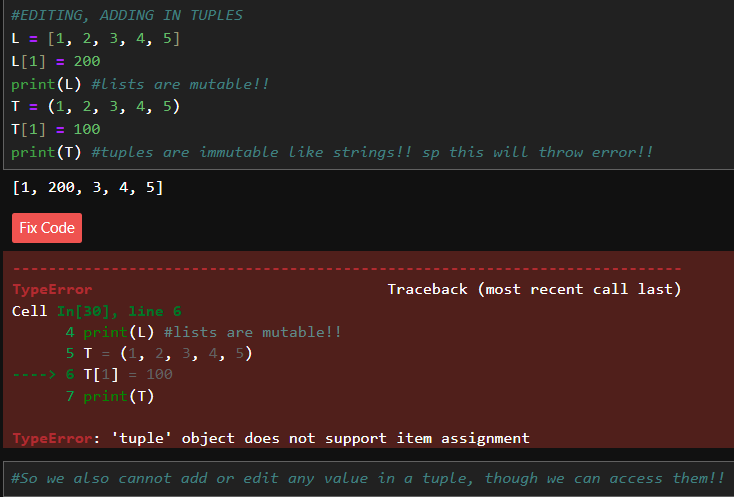
* **Indexing & Slicing:** Ye bilkul Lists aur Strings ki tarah hi kaam karta hai.
* **Positive Indexing:** t[0] (Pehle item ke liye).
* **Negative Indexing:** t[-1] (Aakhiri item ke liye).



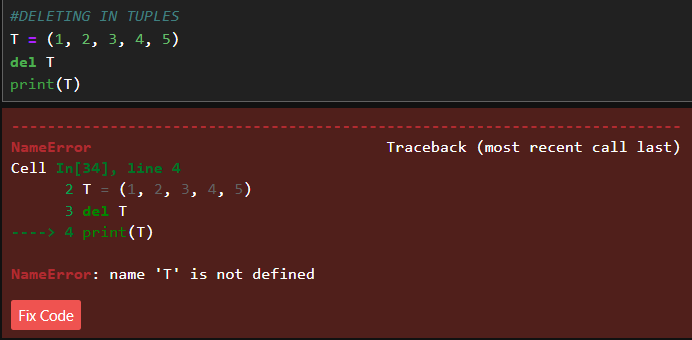
**3. 🛑 The Big Difference: Immutability**

Ye sabse main point hai jo tumhe hamesha yaad rakhna hai:

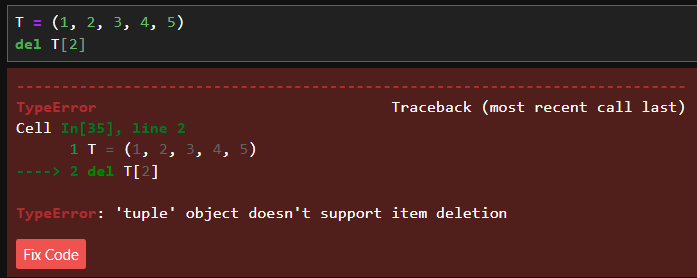
* **No Editing:** Tum tuple ke andar ke kisi item ko change nahi kar sakte (e.g., t[0] = 100 error dega).
* **No Adding/Deleting:** Tum ek bane huye tuple mein naya item add nahi kar sakte aur na hi kisi ek item ko delete kar sakte ho.
* **Read-Only Data:** Tuples ko hum **"Read-Only"** data types bhi kehte hain.



DELETING



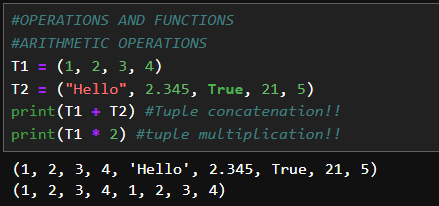
**DELETING CAN BE DONE ONLY IF YOU WANT TO DELETE THE WHOLE TUPLE! THIS DOESN’T EMPTY THE TUPLE, IT REMOVES IT!!**

****

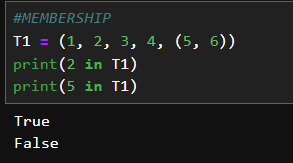
**THOUGH WE CANNOT DELETE A SPECIFIC ELEMENT FROM TUPLE AS IT IS IMMUTABLE!**

**4. Operations & Functions**

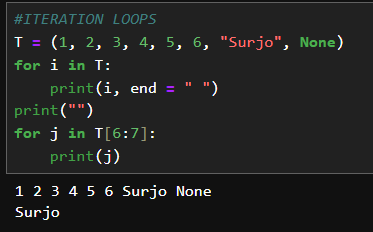
* **Arithmetic:** + se do tuples ko jod sakte ho aur \* se repeat kar sakte ho.



* **Membership:** in operator se check kar sakte ho ki koi cheez tuple mein hai ya nahi.

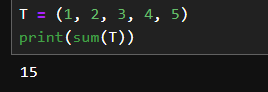


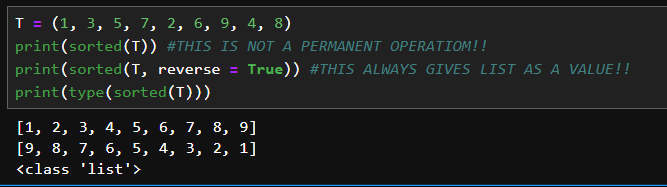
* **Iteration:** for loop se tum tuple ke har item par ja sakte ho.



* **Functions:** len(), min(), max(), sum(), aur sorted() (sorted hamesha ek nayi sorted **list** return karta hai).





**V. V IMPORTANT**

**💡 Why use Tuples? (The "Data Integrity" Factor)**

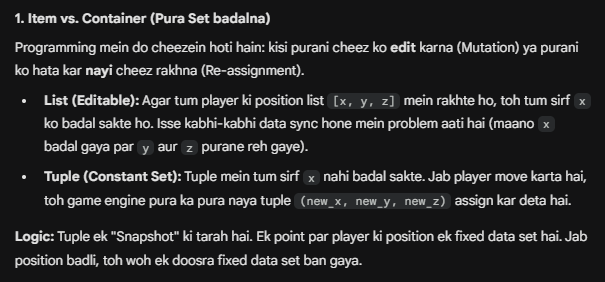


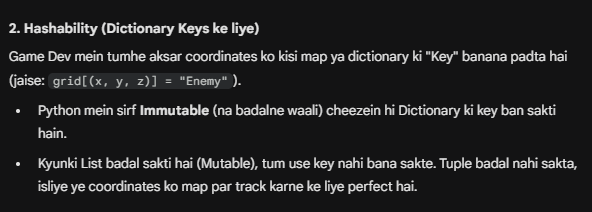
Video mein ek bohot achha example diya gaya hai: Maan lo tum ek college result website bana rahe ho. Agar student ka score **List** mein hai, toh koi bhi developer galti se ya jaan-boojh kar usse change kar sakta hai. Lekin agar data **Tuple** mein hai, toh wo "Write-Protected" hai—usse koi badal nahi sakta. Ise hi **Data Integrity** kehte hain.

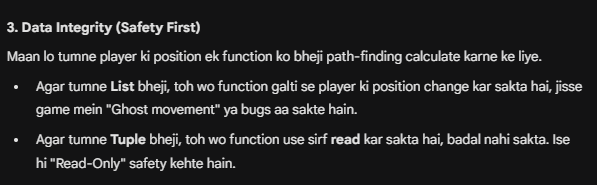
**🎮 Game Dev Extra Point:**

Game developer banna chahte ho toh ye baat hamesha yaad rakhna:

* **Coordinates:** Player ki position (x, y, z) aksar tuple mein store ki jati hai kyunki wo fix rehti hai (ek coordinate set ke taur par).

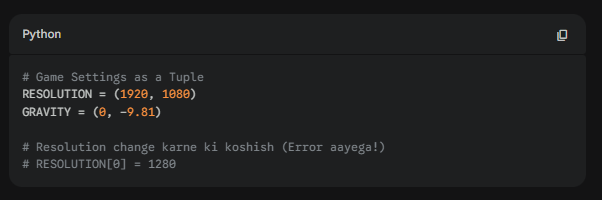






* **Constants:** Game ki aisi settings jo pure game mein nahi badalni chahiye (jaise gravity value ya screen resolution), unhe tuples mein rakhna safe hota hai.

**Example Code:**



**📝 Summary Checklist:**

1. Tuples () use karte hain.
2. Single item tuple ke liye comma , zaruri hai.
3. Ye **Immutable** (non-changeable) hain.
4. Ye Lists se thode **fast** hote hain memory mein.

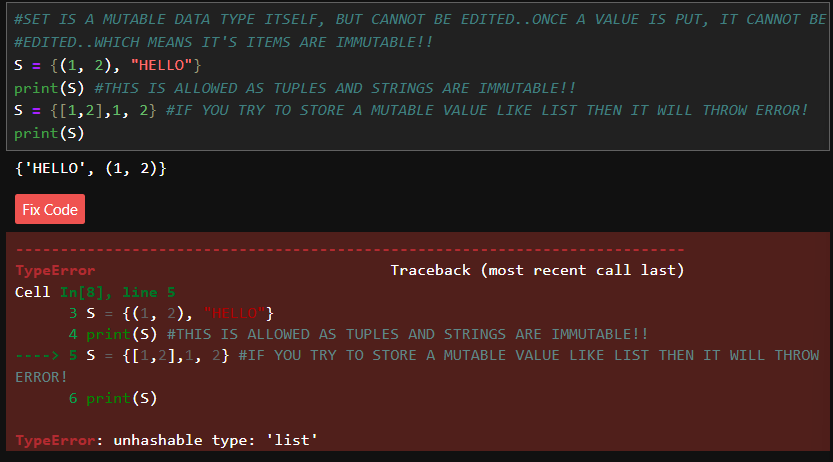
**PYTHON NOTES (23)**

**🎡 Python Sets**

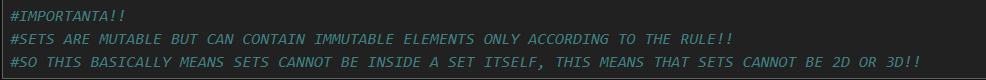
**Definition:** Set ek unordered collection hai items ka, jahan har item **unique** hota hai aur wo **immutable** (badla nahi ja sakta) hona chahiye.

**1. Sets ki Khasiyat (Characteristics)**

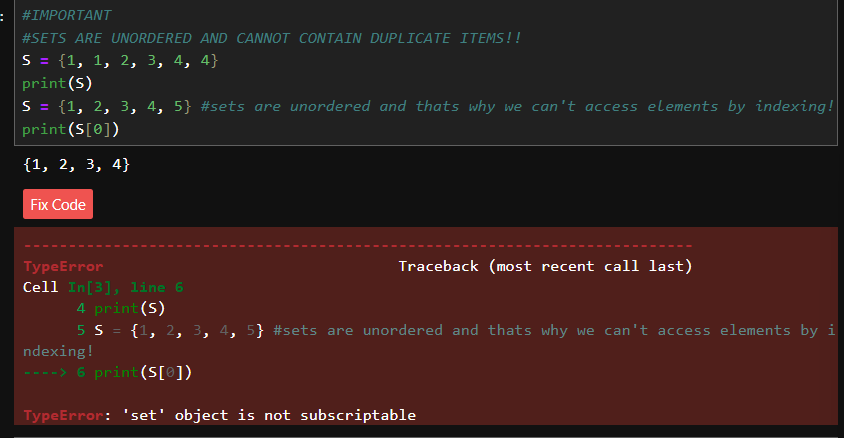
* **Unordered:** Isme items ka koi fix sequence nahi hota. Tumne pehle 'A' dala toh zaroori nahi wo pehle hi dikhe.
* **Unique Items:** Isme duplicates allowed nahi hote. Agar tum ek set mein teen baar 1 daaloge, toh wo use ek hi baar count karega.
* **Mutable Set, Immutable Items:** Tum poore set mein naye items add kar sakte ho, lekin jo item andar chala gaya use edit nahi kar sakte.

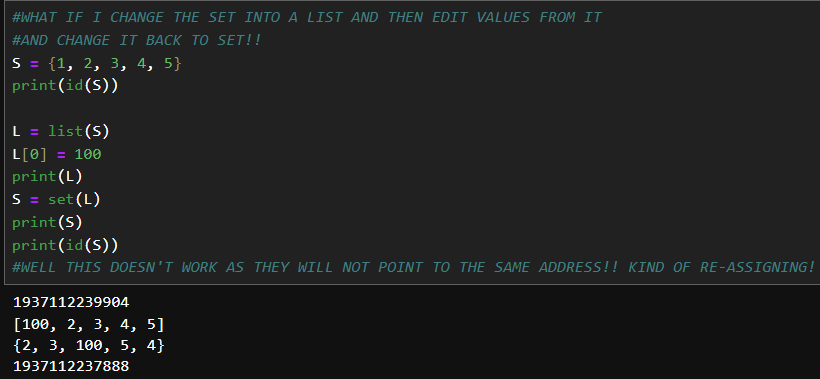


**#IMPORTANT (SETS CANNOT BE MULTI-DIMENSIONAL)**



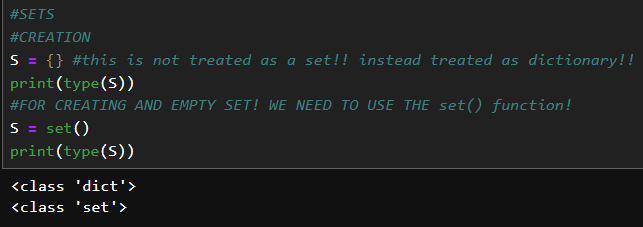
* **No Indexing/Slicing:** Kyunki iska koi order nahi hai, isliye tum s[0] karke item nahi nikaal sakte.



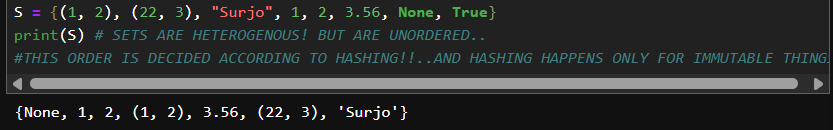


**2. Creating Sets**

* **Syntax:** Sets hamesha **Curly Brackets {}** mein likhe jaate hain.
* **Empty Set:** s = set() (Dhyan rakhna, {} likhne se empty dictionary ban jati hai, set nahi).



* **Mixed Types:** Isme numbers, strings, aur tuples aa sakte hain (lekin lists ya dictionaries nahi, kyunki wo mutable hain).

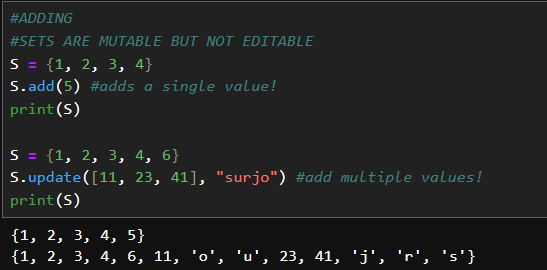


**IMPORTANT (HASHING ON SETS)**

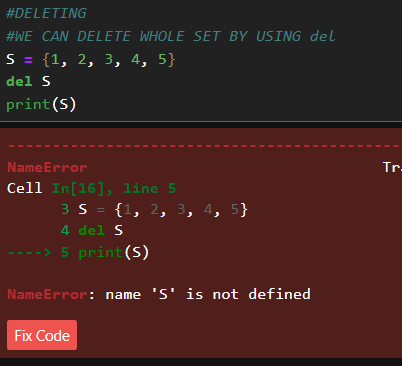


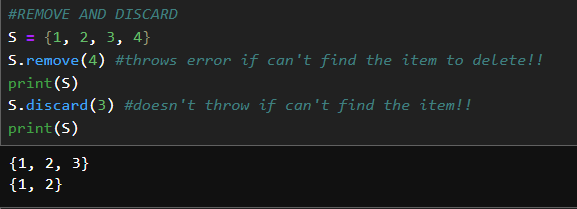
**3. Adding & Deleting Items**

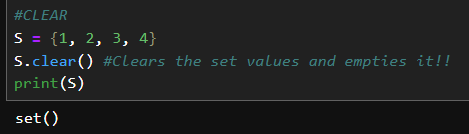
* **Add:** s.add(item) se ek naya item joda jata hai.
* **Update:** s.update([1, 2, 3]) se ek saath kai items jode ja sakte hain.

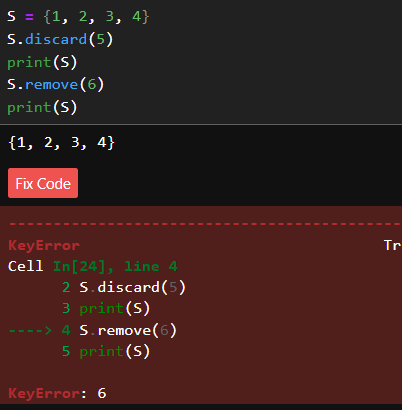


* **Remove/Discard:** \* s.remove(item): Agar item nahi mila toh error dega.
  + s.discard(item): Agar item nahi mila toh chup-chap baitha rahega (error nahi dega).

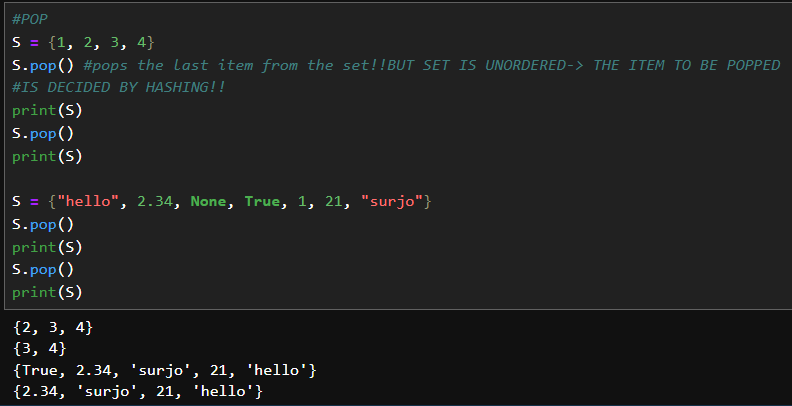




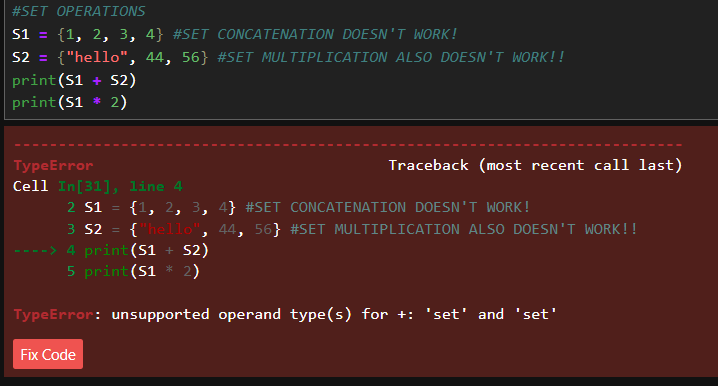


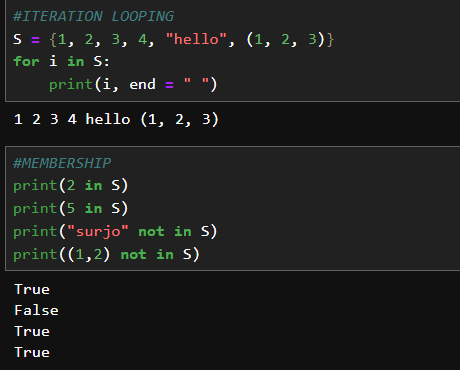


* **Pop:** Kisi bhi random item ko nikaal deta hai.

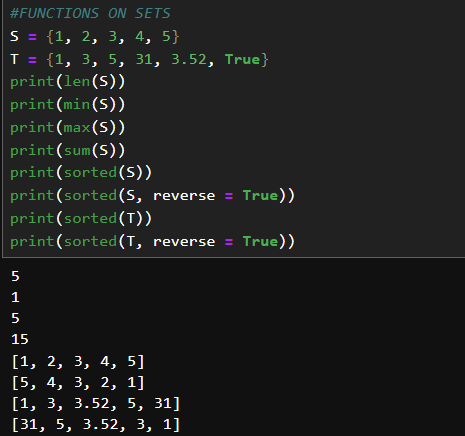


**OPERATIONS ON SETS**

****

****

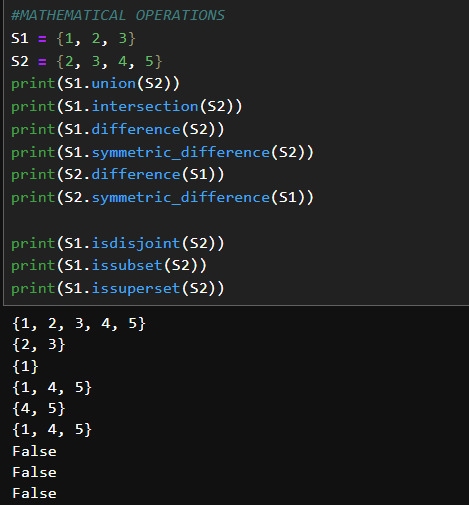
**FUNCTIONS ON SETS**

****

**4. Mathematical Operations (Sabse Important)**

Sets ka asli dum in operations mein hai:

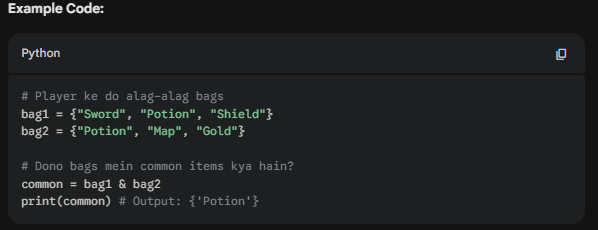
* **Union (|):** Do sets ke saare unique items ko mila dena.
* **Intersection (&):** Sirf wo items jo dono sets mein common hain.
* **Difference (-):** Pehle set ke wo items jo dusre mein nahi hain.
* **Symmetric Difference (^):** Wo items jo ya toh pehle mein hain ya dusre mein, par dono mein common nahi hain.



**🎮 Game Dev Context:**

Game dev mein Sets ka use kab karoge?

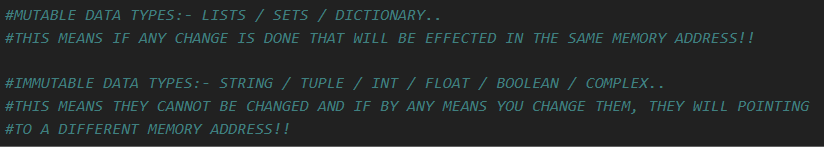
1. **Unique Inventory:** Agar player ne 10 baar "Wood" uthaya, par tum list mein sirf dikhana chahte ho ki uske paas "Wood" hai (quantity nahi, sirf type), toh Set best hai.
2. **Achievement Unlocks:** Har achievement unique hoti hai. Tum check kar sakte ho if achievement in unlocked\_sets.
3. **Collision Detection:** Do sets ka intersection nikal kar check kar sakte ho ki kaunse do objects aapas mein takraye hain.



**📝 Key Takeaways:**

* Sets tab use karo jab tumhe **Duplicates** nahi chahiye.
* Isme indexing nahi hoti, isliye loop chalane ke liye for item in s: use karna padta hai.
* Ye bohot fast hote hain jab tumhe check karna ho item in set (List se bhi fast).

**PYTHON NOTES (24)**

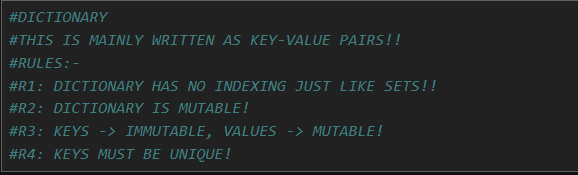


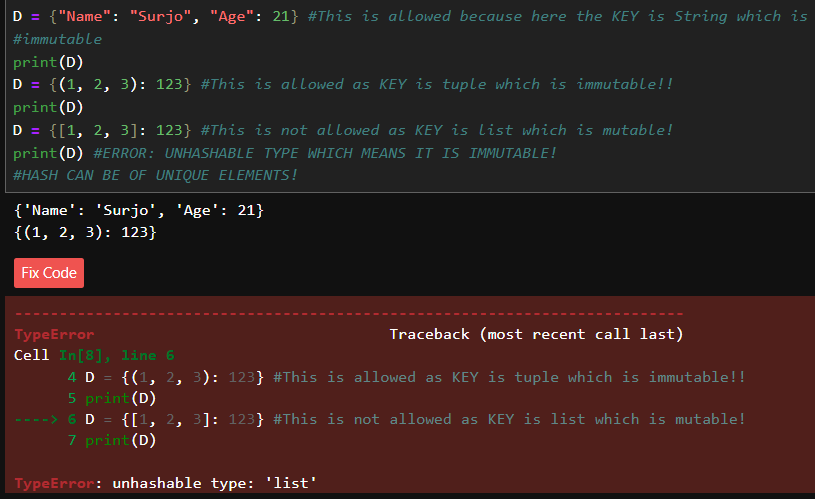
**📖 Python Dictionary**

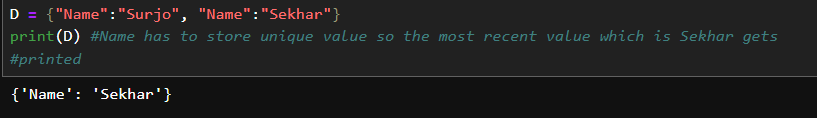
**Definition:** Dictionary ek unordered collection hai jo **Key-Value pairs** ke format mein data store karta hai. Ise curly braces {} mein likha jata hai.

**1. Dictionary ke 4 Bade Rules (Characteristics)**

* **No Indexing**: Isme 0, 1, 2 jaisa koi index nahi hota. Data ko access karne ke liye tumhe uski **Key** ka naam pata hona chahiye.
* **Mutable Data Type**: Tum bani-banayi dictionary mein badlav kar sakte ho, jaise naye items jodna ya purani values badalna.
* **Keys vs Values**:
  + **Keys** hamesha **Immutable** honi chahiye (jaise String, Integer, ya Tuple).
  + **Values** kuch bhi ho sakti hain—Mutable (List) ya Immutable.
* **Unique Keys**: Ek dictionary mein do keys ka naam same nahi ho sakta. Agar tumne repeat kiya, toh Python latest wali value ko hi rakhega.

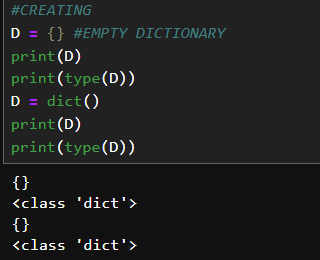




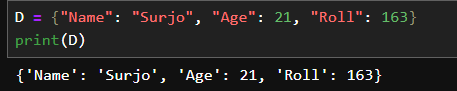


**2. Creating Dictionaries**

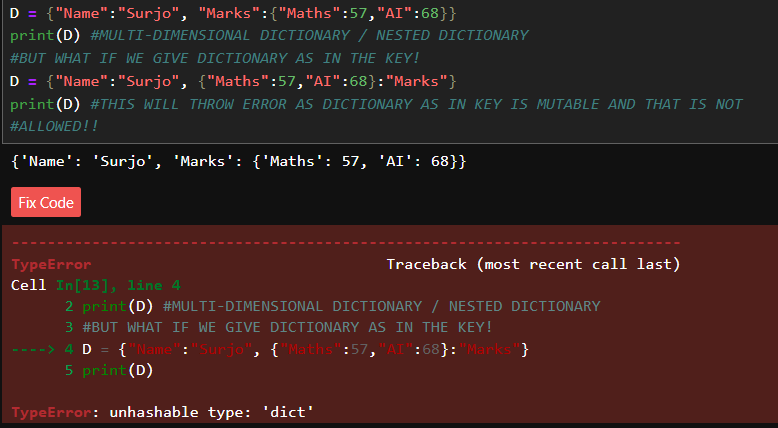
* **Empty Dictionary**: d = {}.



* **Normal Dictionary**: d = {"name": "Surjo", "age": 20}.

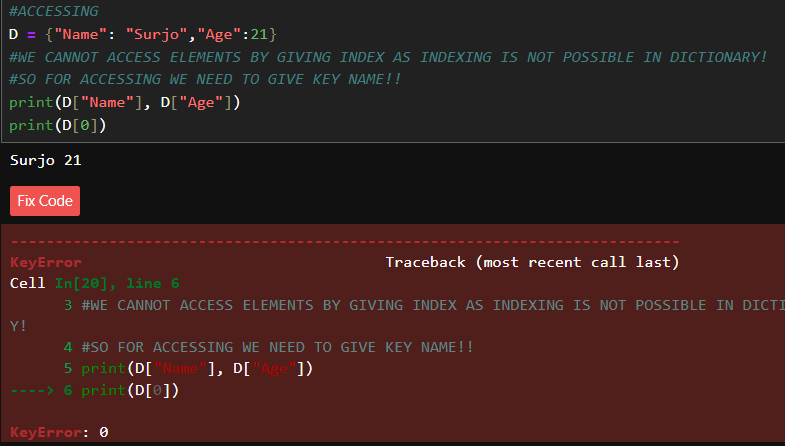


* **2D / Nested Dictionary**: Dictionary ke andar doosri dictionary.
  + *Example:* s = {"name": "Rahul", "marks": {"maths": 80, "ds": 90}}.



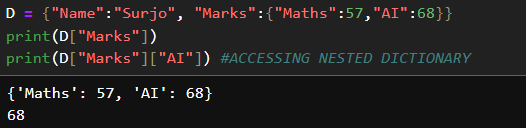
**3. Accessing & Editing Items**

* **Accessing**: d["name"] likhne se value mil jayegi.

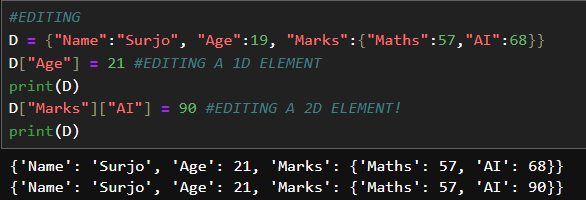




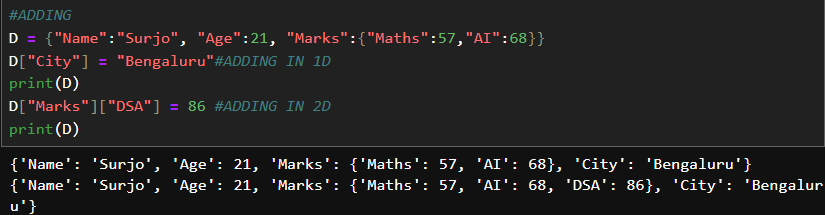
* **Nested Access**: s["marks"]["ds"] likhne se 90 milega.



* **Editing**: Purani key ko naya value assign kar do: d["age"] = 21.

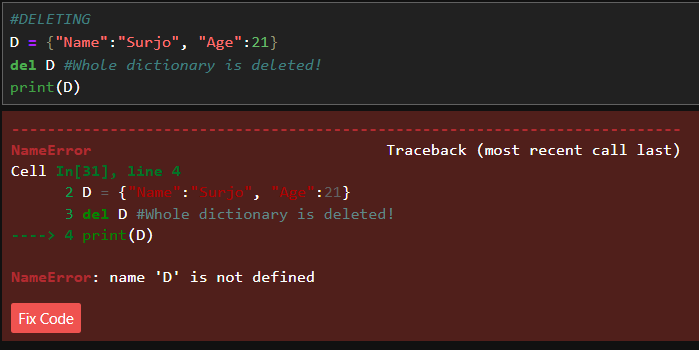


* **Adding New Key**: Bas ek nayi key aur value likh do: d["city"] = "Mumbai".



**4. Deleting & Functions**

* **del d["key"]**: Kisi specific key-value jode ko uda deta hai.

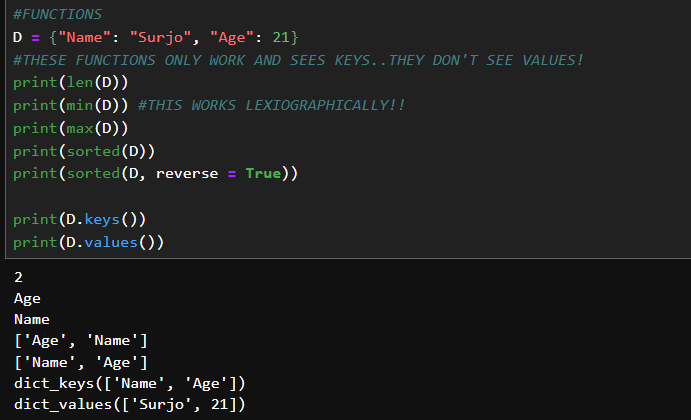


* **clear()**: Dictionary ko poori tarah khali kar deta hai.

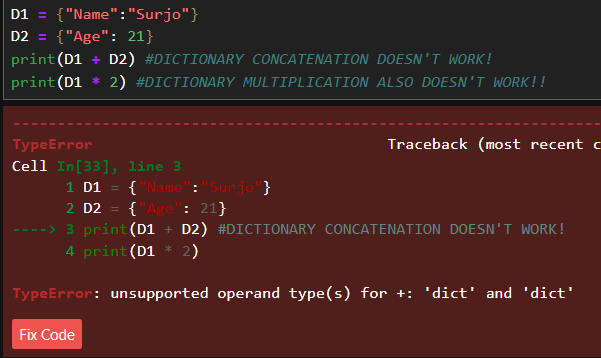


* **keys()**: Saari keys ki ek list deta hai.
* **values()**: Saari values ki ek list deta hai.

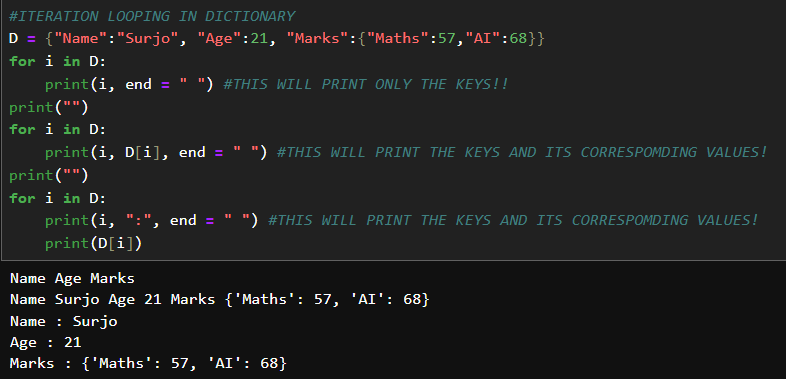
**FUNCTIONS WITH DICTIONARY**

****

**IMPORTANT**

****

**ITERATION LOOPING DICTIONARY**

****

**MEMBERSHIP OPERATIONS**



**🛡️ Mutable vs Immutable Summary**

Ye table yaad rakhna bohot zaruri hai:

* **Mutable (Changeable)**: List, Set, Dictionary.
* **Immutable (Non-changeable)**: String, Tuple, Int, Float, Boolean, Complex.

**🎮 Game Dev Connection (Extra Points):**

Dictionary game dev mein **JSON data** handle karne ke liye use hoti hai.

* **Inventory**: inventory = {"sword": 1, "potion": 5, "coins": 100}.
* **Enemy Stats**: enemy = {"type": "Orc", "hp": 150, "speed": 10}.

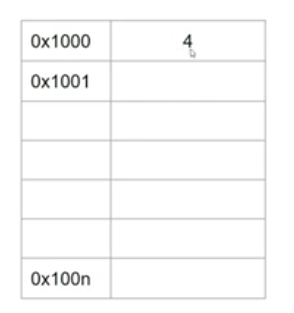
**PYTHON NOTES (25)**

**🧠 Python Deep Dive: Memory, Variables & Mutabili ty**

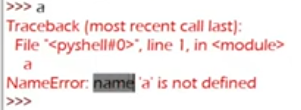
Is video mein samjhaya gaya hai ki Python memory (RAM) ke andar data kaise store karta hai aur variables kaise kaam karte hain.

**# RAM ME CHIZE STORE HOTI HAI REGISTER ME..EK REGISTER FLIP-FLOPS SE MILKAR BANA HOTA HAI! FLIP-FLOPS BANTA HAI LOGIC GATE SE AUR LOGIC GATE BANTA HAI DIODE SE!!**

**# EK REGISTER ME MAINLY TWO COMPONENTS HOTE HAI..EK MEMORY ADDRESS AUR DUSRA JO CHIZ STORE HO RAHI HAI!! AUR USUALLY MEMORY ADDRESS CONTINUOS HOTA HAI! AND YE MEMORY ADDRESS KO HEXA DECIMAL VALUE ME LIKHA JATA HAI..**

** **

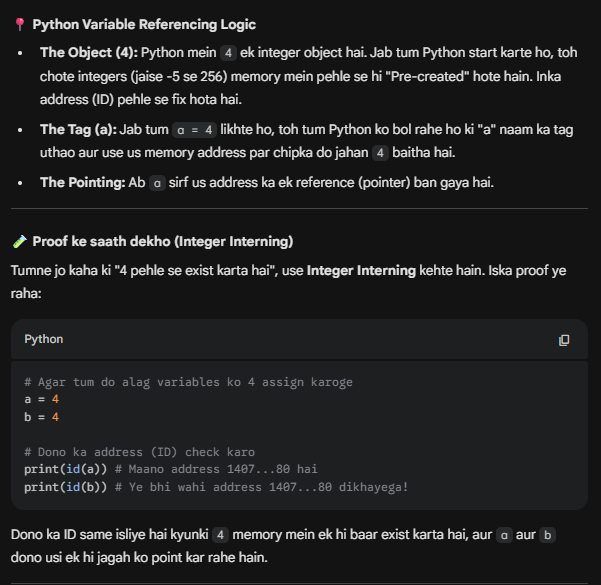
**ACTUAL WAY OF STORING HOTA HAI JAISE RIGHT SIDE VALE PICTURE ME DIKH RAHA HAI! 4 KO BINARY ME CHANGE KARTE HAI AND USKA PEHLA MEMORY SPACE SE HAM REPRESENT KARTE HAI!**

****

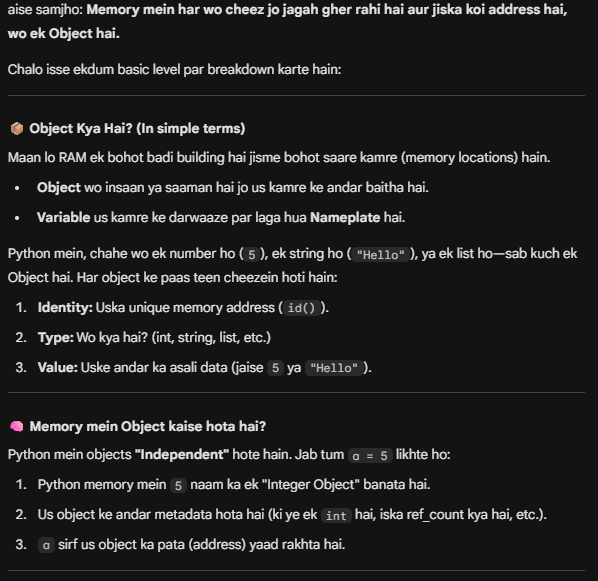
**# KOI BHI VARIABLE KO PYTHON NAME BOLTA HAI..**

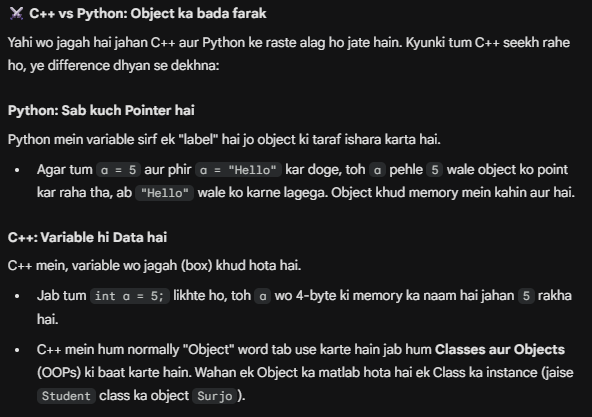
**# PYTHON VARIABLE REFERENCING LOGIC:**

**CONCEPT:- (INTEGER INTERNING)**

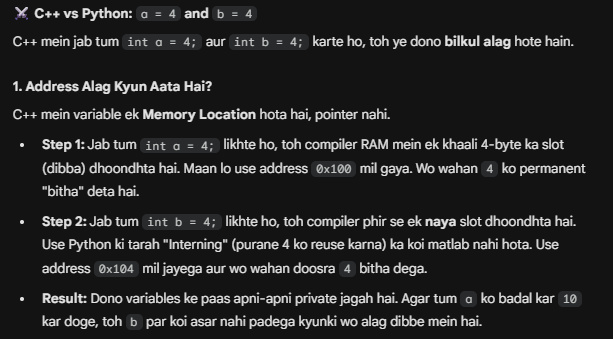
****

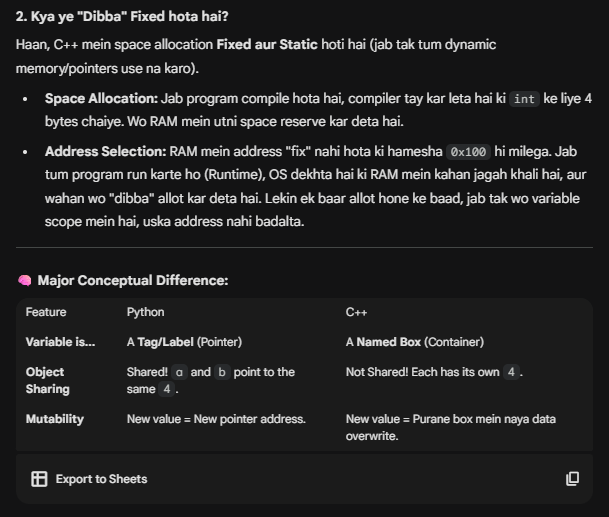
**#IMPORTANT (OBJECTS IN PYTHON)**

****

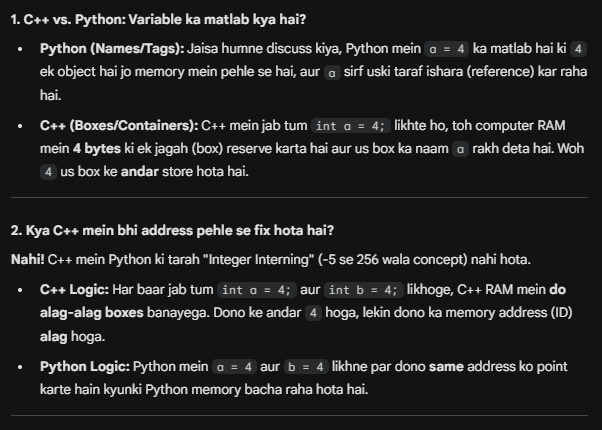
****

**# IF VARIABLE ARE HAVING SAME VALUE..IN C++ AND IN PYTHON..DIFFERENCE??**

****

****

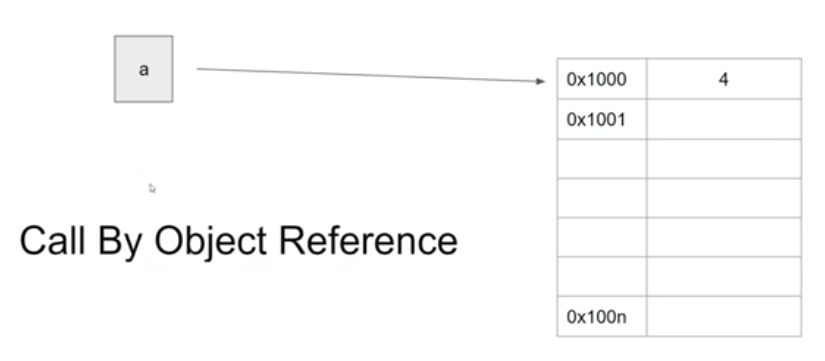
**IMPORTANT (C++ AND PYTHON -> DIFFERENT MEMORY MODELS)**

****

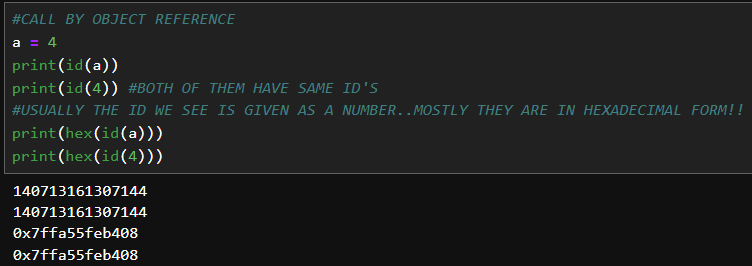
**1. Variable Referencing (Call by Object Reference)**

Python mein jab tum a = 5 likhte ho, toh 'a' ke andar 5 store nahi hota.

* **Reality**: 5 memory mein ek jagah store hota hai, aur a sirf us jagah (address) ka ek **naam/reference** hota hai.
* **ID Function**: id(a) se tum us memory address ko dekh sakte ho jahan 5 store hai.

****

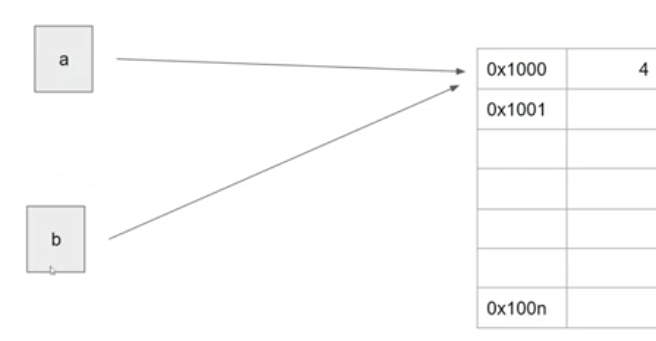
**# YE CHIZ KO CALL BY OBJECT REFERENCE KEHTE HAI!!..JAB KOI VARIABLE KISI OBJECT KO POINT KARTA HAI..AND PYTHON ME YE SAB USUALLY OBJECT HI HOTE HAI!!**

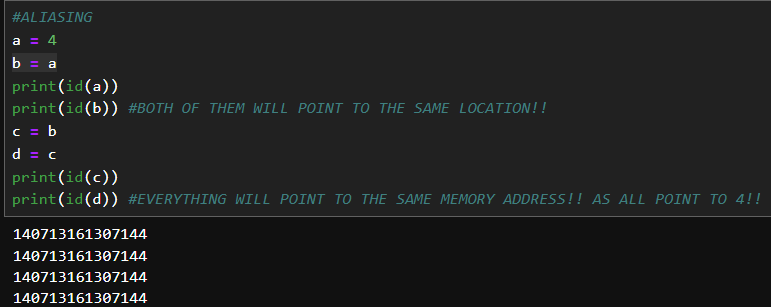


**2. Aliasing**

Jab tum a = 5 aur b = a likhte ho, toh a aur b dono **same** memory address ko point karte hain. Ise **Aliasing** kehte hain.

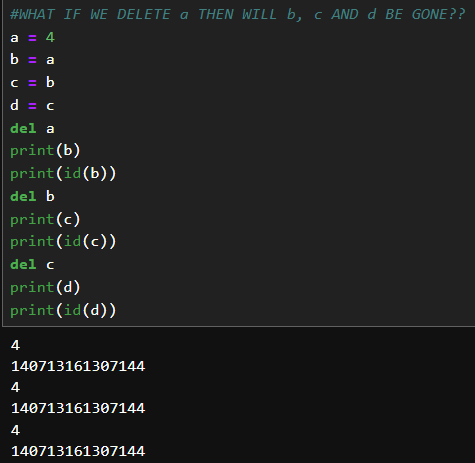
* Dono variables ek hi "Object" (5) ko refer kar rahe hain.



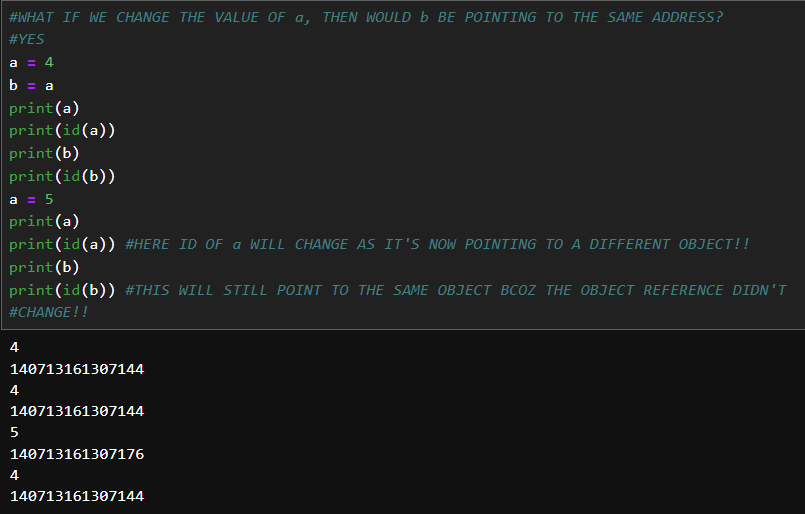


**IMPORTANT CONCEPT ABOUT ALIASING**

**(PROBLEM 1)**

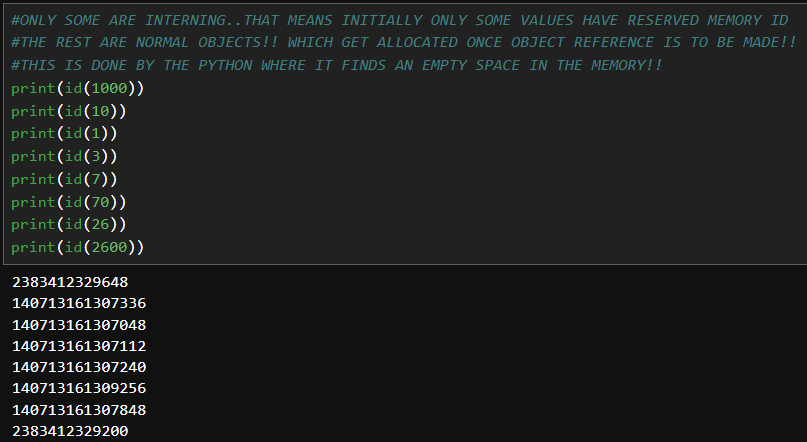


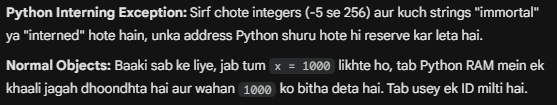
**(PROBLEM 2)**



**#IMPORTANT**

**#NORMAL OBJECTS AND PYTHON INTERNING OBJECTS!!**

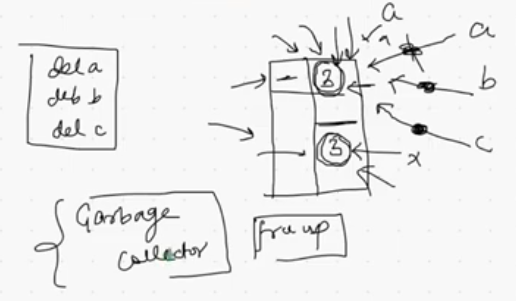
****

****

**3. Garbage Collection**

Python mein memory management automatic hota hai.

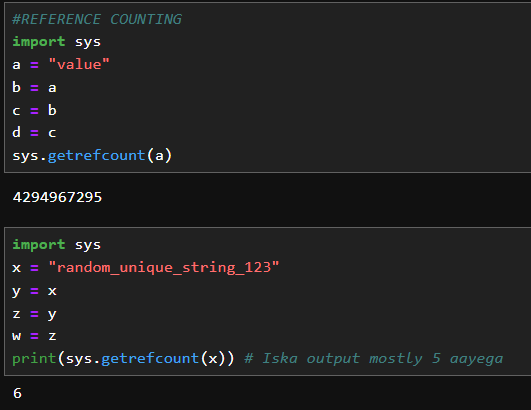
* Agar tumne a = 5 likha aur phir a = 10 kar diya, aur koi doosra variable 5 ko point nahi kar raha, toh 5 "faltu" ho jayega.
* **Garbage Collector**: Ye ek background program hai jo aisi memory ko dhundhta hai jise koi refer nahi kar raha aur use **clear** kar deta hai taaki RAM khali ho jaye.



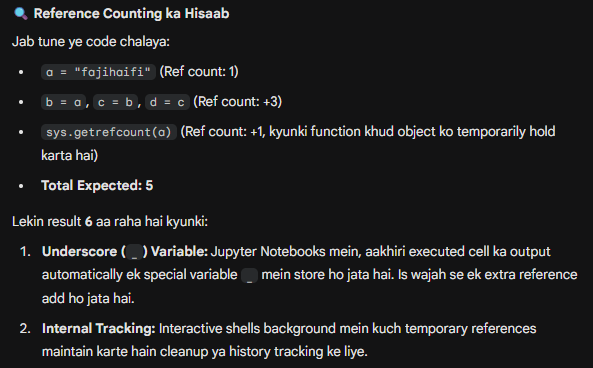
**4. Reference Count**

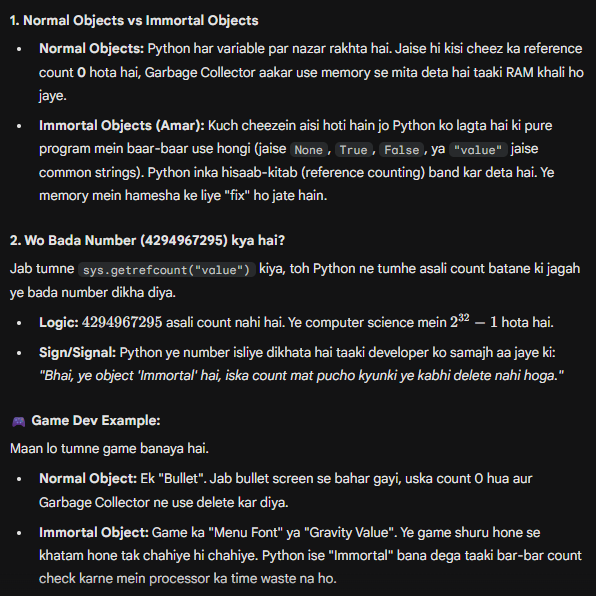
Tum check kar sakte ho ki ek memory address par kitne variables point kar rahe hain:

* sys.getrefcount(a): Ye batata hai ki kitne references hain. (Note: Ye hamesha actual count se 1 zyada dikhayega kyunki function khud bhi ek reference create karta hai).

****

**REASON FOR UNEXPECTED OUTPUTS IN THESE CASES ABOVE:**

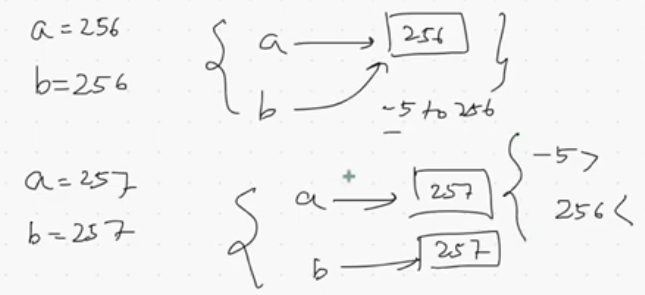
****

****

**5. Python Memory Optimization Tricks (The Interview Stuff)**

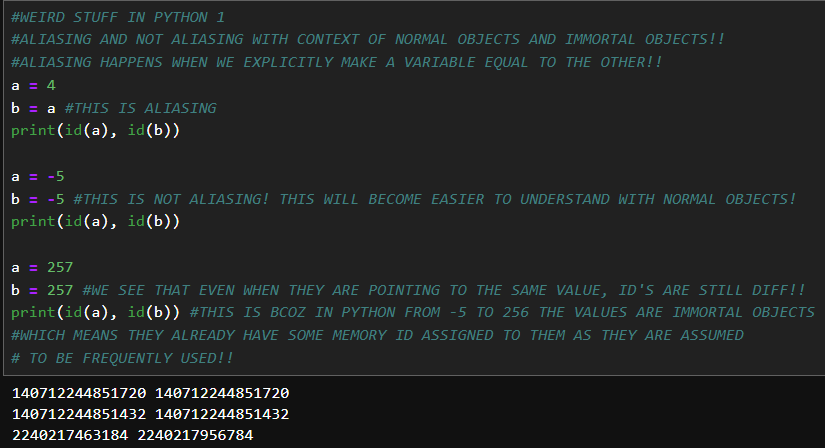
3 bade optimization bataye gaye hain:

* **Integer Interning (-5 to 256)**: Python pehle se hi -5 se 256 tak ke numbers memory mein save kar leta hai. Agar tum a = 10 aur b = 10 likhoge, toh unki ID hamesha **same** hogi kyunki Python naya 10 create nahi karta.



* + Is range ke bahar (a = 300, b = 300), IDs **alag** ho sakti hain (depends on how you run the code).

**# WEIRD STUFF IN PYTHON 1**

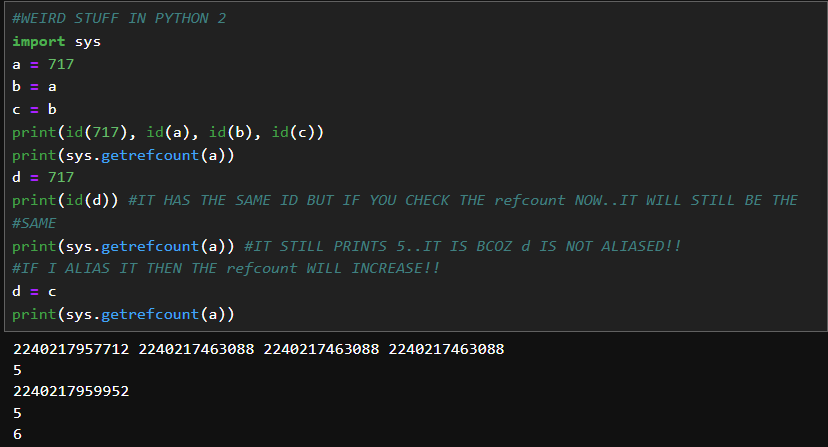


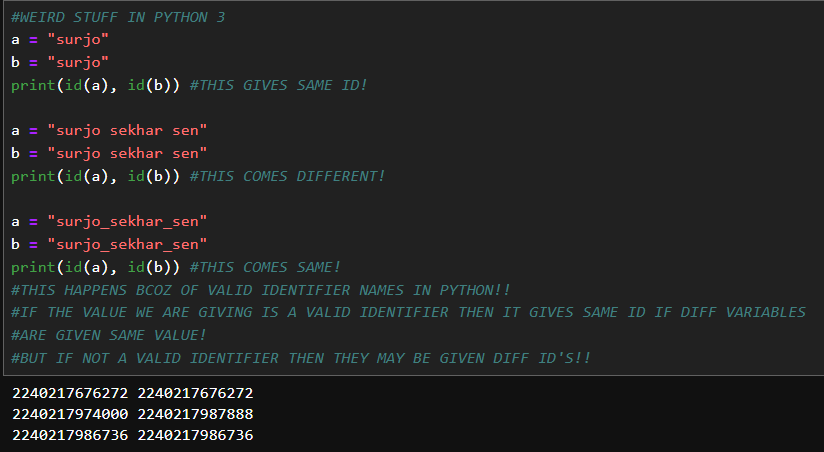
* **String Interning**: Chote strings (jo identifiers jaise dikhte hain, bina spaces ke) Python memory mein save kar leta hai (Interning). Ise memory aur time dono bachte hain.
* **Empty Objects**: Khali tuples ya objects ke liye bhi Python sometimes same memory address use karta hai optimization ke liye.

**# IMPORTANT**

**ALIASING AND REFCOUNT**

**# WEIRD STUFFS IN PYTHON 2**

**# WEIRD STUFFS IN PYTHON 3**

****

**💻 Game Dev Perspective (For You):**

Game dev mein ye knowledge bohot kaam aati hai jab tum lakhon particles ya enemies handle karte ho.

* Agar tumne unki memory references ko sahi se manage nahi kiya (yani unhe delete nahi kiya but variables abhi bhi point kar rahe hain), toh **Memory Leak** ho sakta hai aur tumhara game lag karega.

**🔄 Mutability: The Memory Game**

Video mein samjhaya gaya hai ki Python ke data types do camps mein bate huye hain:

**1. Immutable (Non-changeable)**

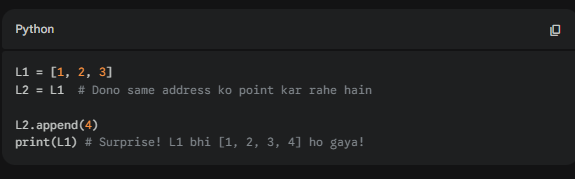
* **Kaun hain?** Int, Float, String, Tuple, Boolean.
* **Memory Behavior:** Agar tum kisi immutable object ko "change" karne ki koshish karte ho, toh Python us purane address par data update nahi karta. Wo ek naya address (ID) allot karta hai.
  + *Example:* s = "Hello", phir s = s + " World". Pehle wala s aur naya s dono ki **ID alag** hogi.

**2. Mutable (Changeable)**

* **Kaun hain?** List, Dictionary, Set.
* **Memory Behavior:** Ye bilkul C++ ke "dibbe" ki tarah behave karte hain. Agar tum list mein item add karte ho (list.append(4)), toh uska **Memory Address (ID) nahi badalta**.
  + *Logic:* Python usi purane address par jaakar data ko overwrite ya expand kar deta hai.

**⚠️ The Dangerous Side of Mutability (Aliasing Bug)**

Video mein ek bohot bada warning diya gaya hai jo developers aksar galti karte hain:



**Kyun hua?** Kyunki L1 aur L2 dono ek hi mutable object (list) ka pata yaad rakhte hain. Ek jagah badlav karoge, har jagah dikhega.

**🧪 Weird Cases (Interview Questions)**

Video ke aakhir mein kuch "Dimaag ghumane wale" questions discuss huye hain:

1. **Tuple inside a List:** List mutable hai, par uske andar ka tuple immutable hi rahega.
2. **List inside a Tuple:** Ye sabse crazy hai! Tuple khud immutable hai (uske andar ke "pointers" nahi badal sakte), lekin agar us pointer ke aage koi **List** baithi hai, toh tum us list ke andar ka data badal sakte ho! Isse "Tuple ki immutability" khatam nahi hoti, kyunki tuple abhi bhi usi list address ko point kar raha hai.

**🎮 Game Dev Takeaway:**

* **Efficiency:** Mutable objects (Lists) memory bachte hain kyunki naya object bar-bar create nahi karna padta.
* **Safety:** Immutable objects (Tuples/Strings) safe hain kyunki tumhe darr nahi hota ki koi doosra function galti se tumhara original data badal dega.

**📝 Summary Checklist:**

* **is operator:** Ye IDs compare karta hai (a is b matlab same memory address).
* **== operator:** Ye sirf values compare karta hai.
* **Copying:** Agar tumhe mutable objects mein pange nahi chahiye, toh L2 = L1.copy() use karo taaki dono ki IDs alag ho jayein.