Python

**Day2**: **Advanced Rules for Variables in Python**

1. **Naming Rules:**
   * Can contain letters (a–z, A–Z), digits (0–9), and underscores \_.
   * Cannot start with a digit.
   * Cannot be a Python **keyword** or built-in function name.
   * Case-sensitive (age ≠ Age).( Python, C / C++,Java, JavaScript)but not case sensitive(SQL, BASIC)
2. **Dynamic Typing:**
   * Python variables **don’t require type declaration**.
   * The type is determined automatically based on the value.

x = 10 # int

x = "Ram" # str (type changes dynamically)

1. **Multiple Assignment:**
   * Python allows assigning multiple variables in one line:

a, b, c = 1, 2, 3

**Valid Examples:**

student\_name = "Ram"

\_score = 95

totalMarks123 = 100

1. **Invalid Examples:**

1st\_name = "Ram" # starts with digit ❌

for = 10 # keyword ❌

1. **Memory Reference:**
   * Variables in Python are **references to objects** in memory.
   * Changing the value of a variable points it to a new object.
2. **Global & Local Scope:**
   * Variables can be **global** (accessible anywhere) or **local** (inside a function only).
   * Sure Ram! Here’s a simple explanation of **why we use different naming cases** in Python and programming:

**🔹 Why We Use Different Naming Cases**

1. **Readability** ✅

* Naming conventions make code **easy to read and understand**.
* Example: student\_name = "Ram" # easy to read (snake\_case)

1. **Consistency** 🎯

* Using a standard case across a project keeps code **consistent**.
* Easier for **team collaboration**.

1. **Indicates Purpose / Type** 🔍

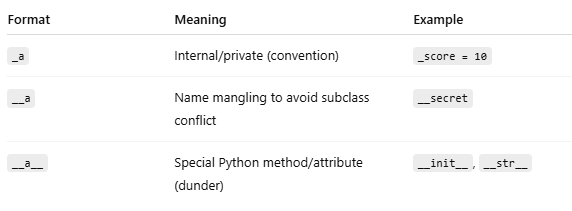
* Different cases help programmers **know what the variable represents**:
  + snake\_case → regular variable or function
  + PascalCase → class name
  + UPPER\_CASE → constant value

1. **Avoids Errors** ⚠️

* Helps avoid naming conflicts and mistakes in large programs.

✅ **Summary Table:**

| **Case** | **Use in Python** | **Example** |
| --- | --- | --- |
| Snake Case (All lowercase letters, words separated by \_) | Variables, functions | student\_name |
| Camel Case(First word lowercase, following words start with uppercase, no spaces) | Rare in Python, common in JS/Java(Variables and functions in some other languages like Java, JavaScript) | studentName |
| Pascal Case(Every word starts with uppercase, no spaces) | Classes | StudentName |
| Kebab Case(not common in Python, more in URLs/filenames) | URLs, filenames | student-name |
| Upper Case(All letters uppercase, words separated by \_) | Constants | PI, MAX\_VALUE |



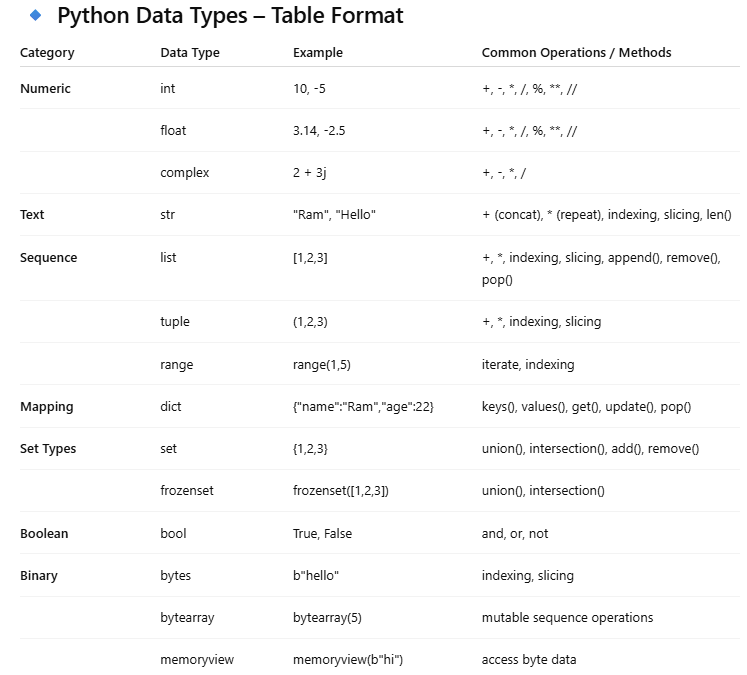
Paxadi ramro snaga details ma xa

Mainly 36 reserver word/keyword

Code : Import keyword

Print(keyword.kwlist)

['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']



**🔹 Number Systems in Python**

Python supports **different number systems** for integers.

| **System** | **Prefix in Python** | **Example** | **Base** |
| --- | --- | --- | --- |
| **Binary** | 0b or 0B | 0b1010 | 2 |
| **Decimal** | No prefix | 10 | 10 |
| **Octal** | 0o or 0O | 0o12 | 8 |
| **Hexadecimal** | 0x or 0X | 0xA | 16 |

**🔹 Examples in Python**

# Binary

x = 0b1010

print(x) # 10 (decimal)

# Decimal

y = 10

print(y) # 10

# Octal

z = 0o12

print(z) # 10 (decimal)

# Hexadecimal

h = 0xA

print(h) # 10 (decimal)

**🔹 Key Points**

1. Python **automatically converts** to decimal when doing calculations.
2. Use **bin(), oct(), hex()** to convert decimal numbers to other bases:

n = 10

print(bin(n)) # 0b1010

print(oct(n)) # 0o12

print(hex(n)) # 0xa

**🔹 id() in Python**

**Definition:**

* The **id() function** in Python returns the **unique identity (address) of an object** in memory.
* Every object in Python has a **unique ID**, which is its **memory location**.

**Day3:** Complex number data type:

**1. Syntax**

z = a + bj

* a → Real part
* b → Imaginary part
* j → Imaginary unit (√-1)

**2. Example**

z1 = 3 + 4j

z2 = 1 - 2j

print(z1) # (3+4j)

print(type(z1)) # <class 'complex'>

**3. Accessing Parts**

* real → real part
* imag → imaginary part

z = 3 + 4j

print(z.real) # 3.0

print(z.imag) # 4.0

**4. Arithmetic Operations**

z1 = 3 + 4j

z2 = 1 + 2j

print(z1 + z2) # (4+6j)

print(z1 - z2) # (2+2j)

print(z1 \* z2) # (-5+10j)

print(z1 / z2) # (2.2-0.4j)

**5. Built-in Functions**

import cmath

z = 3 + 4j

print(abs(z)) # 5.0 (magnitude)

print(cmath.phase(z)) # 0.927 radians (angle θ)

print(cmath.polar(z)) # (5.0, 0.927) → (r, θ)

**6. Create using complex() function**

z = complex(5, -2)

print(z) # (5-2j)

**✅ Boolean Data Type in Python**

**1. What is Boolean?**

* **Boolean (bool)** is a built-in data type in Python that can hold only **two values**:
  + True(1)(aagadaiko letter jahile ni capital hunu parxa hai bro)
  + False(0)
* It is mainly used in **logical conditions**, **comparisons**, and **decision making**.

**2. Example**

x = True

y = False

print(x) # True

print(type(x)) # <class 'bool'>

print(x+y) #output is 1 bro

**3. Boolean from Comparisons**

Booleans are often produced by **comparison operators**:

a = 10

b = 20

print(a > b) # False

print(a < b) # True

print(a == 10) # True

**4. Boolean Operators**

Boolean values can be combined using **logical operators**:

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| and | Returns True if both are True | True and False → False |
| or | Returns True if at least one is True | True or False → True |
| not | Reverses the value | not True → False |

Example:

x = True

y = False

print(x and y) # False

print(x or y) # True

print(not x) # False

**5. Boolean Conversion**

You can use the bool() function to convert other data types into Boolean.

👉 Rule:

* **Falsy values → False**: 0, 0.0, "", [], {}, set(), None
* **Truthy values → True**: everything else

print(bool(0)) # False

print(bool(42)) # True

print(bool("")) # False

print(bool("Hello")) # True

print(bool([])) # False

print(bool([1,2,3])) # True

**6. Usage in Conditions**

is\_logged\_in = True

if is\_logged\_in:

print("Welcome User!")

else:

print("Please login")

**📝 String Data Type in Python**

-Not char in python( all string ho bro)

* A **string** in Python is a sequence of **characters** enclosed in **single quotes '...'**, **double quotes "..."**, or **triple quotes '''...''' or """..."""**.
* Strings are **immutable** (once created, they cannot be changed).

**2. Creating Strings**

str1 = 'Hello'

str2 = "World"

str3 = '''This is

a multi-line

string.''' #multiline line kolagi chai triple “”” or’’’ use bro

print(str1) # Hello

print(str2) # World

print(str3)

**3. Accessing Characters**

Strings are like arrays of characters → you can access them using **indexing**.

* Index starts from 0.
* Negative index starts from the end.

s = "Python"

print(s[0]) # P

print(s[3]) # h

print(s[-1]) # n (last character)

**4. String Slicing**

You can extract a portion (substring) using slicing:

s = "Python"

print(s[0:4]) # Pyth (from index 0 to 3)

print(s[:3]) # Pyt (from start to 2)

print(s[2:]) # thon (from 2 to end)

print(s[::-1]) # nohtyP (reversed string)

**5. String Operations**

a = "Hello"

b = "World"

# Concatenation

print(a + " " + b) # Hello World

# Repetition

print(a \* 3) # HelloHelloHello

# Membership

print("H" in a) # True in → checks if a substring exists inside a string.

print("z" not in a) # True not in → checks if a substring does not exist.

**6. Useful String Methods**

Python provides many built-in string methods:

s = " Python Programming "

print(s.upper()) # PYTHON PROGRAMMING

print(s.lower()) # python programming

print(s.strip()) # "Python Programming" (removes spaces)

print(s.replace("Python", "Java")) # Java Programming

print(s.split()) # ['Python', 'Programming']

print(s.startswith("Py")) # True

print(s.endswith("ing")) # True

print(len(s)) # 22 (length)

**7. String Formatting**

**f-strings (modern way, Python 3.6+)**

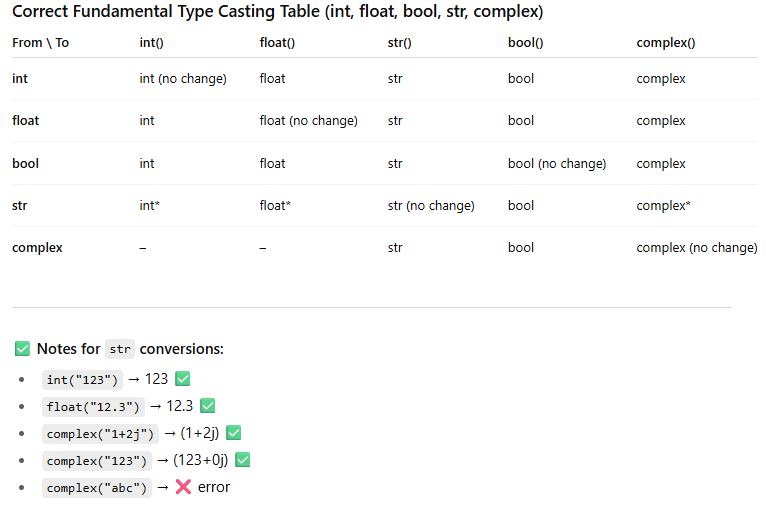
name = "Ram"

age = 21

print(f"My name is {name} and I am {age} years old.")

**format() method**

print("My name is {} and I am {} years old.".format(name, age))

Type casting:

**Fundamental Data Types**

* int, float, complex, bool, str

**Immutability**

* All fundamental data types are **immutable**.
* Once created, the object **cannot be changed**.
* Any modification creates a **new object**.

**Who destroys the old object?**

👉 **Python’s Garbage Collector (GC)** automatically destroys objects that are **no longer referenced**.

That means:

* If no variable points to an object, it becomes **unreachable**.
* Then the Garbage Collector **frees the memory**.

✅ Example:

x = 10

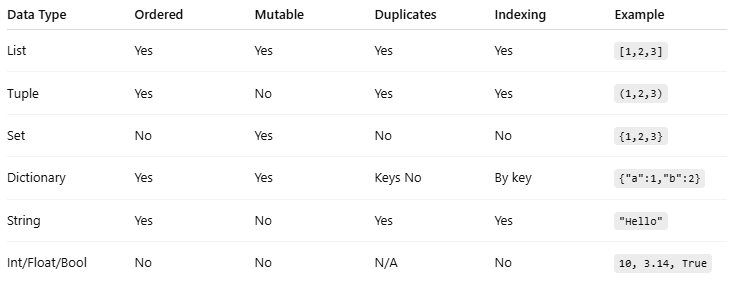
print(id(x)) # address of 10

x = x + 5 # now x refers to new object 15

print(id(x)) # different address

# object 10 has no reference now → Garbage Collector will destroy it

Day 4



**Python List Data Type**

**Definition:**

A **list** in Python is an **ordered, mutable collection** of items.

* **Ordered**: Elements have a defined order, and you can access them by index.
* **Mutable**: You can **change, add, or remove elements** after the list is created.
* **Heterogeneous**: A list can contain **different data types** (int, float, string, other lists, etc.).

**Syntax:**

my\_list = [element1, element2, element3, ...]

**Example:**

fruits = ["apple", "banana", "cherry"]

* **Ordered ≠ unchangeable order**.
  + Being ordered just means **Python remembers the sequence of elements**.
  + The **order can be changed** by operations like insert(), append(), sort(), reverse().

**Key Features of Lists:**

1. **Indexing and Slicing**
2. fruits[0] # "apple" -> first element
3. fruits[-1] # "cherry" -> last element
4. fruits[0:2] # ["apple", "banana"] -> slicing
5. **Mutable**
6. fruits[1] = "orange" # change second element
7. **Heterogeneous**
8. mixed = [1, "apple", 3.5]
9. **Dynamic Size**
   * Can add or remove elements anytime.

**Common List Operations in Python**

**1. Adding Elements**

**a) append()**

* Adds a single element at the **end** of the list.
* **Syntax:** list.append(element)
* **Example:**

fruits = ["apple", "banana"]

fruits.append("cherry")

print(fruits) # Output: ['apple', 'banana', 'cherry']

**b) extend()**

* Adds **multiple elements** from another list (or iterable) at the **end**.
* **Syntax:** list.extend(iterable)
* **Example:**

fruits = ["apple", "banana"]

fruits.extend(["cherry", "mango"])

print(fruits) # Output: ['apple', 'banana', 'cherry', 'mango']

**c) insert()**

* Inserts an element at a **specific index**.
* **Syntax:** list.insert(index, element)
* **Example:**

fruits = ["apple", "banana"]

fruits.insert(1, "cherry")

print(fruits) # Output: ['apple', 'cherry', 'banana']

**2. Removing Elements**

**a) remove()**

* Removes **first occurrence** of a specified element.
* **Syntax:** list.remove(element)
* **Example:**

fruits = ["apple", "banana", "cherry"]

fruits.remove("banana")

print(fruits) # Output: ['apple', 'cherry']

**b) pop()**

* Removes element at a **specific index** and **returns it**. Default is last element.
* **Syntax:** list.pop(index)
* **Example:**

fruits = ["apple", "banana", "cherry"]

last = fruits.pop()

print(last) # Output: cherry

print(fruits) # Output: ['apple', 'banana']

**c) clear()**

* Removes **all elements** from the list.
* **Syntax:** list.clear()
* **Example:**

fruits = ["apple", "banana", "cherry"]

fruits.clear()

print(fruits) # Output: []

**3. Other Useful Operations**

**a) len()**

* Returns the **number of elements** in the list.
* **Example:**

fruits = ["apple", "banana", "cherry"]

print(len(fruits)) # Output: 3

**b) sort()**

* Sorts the list **in ascending order** (by default).
* **Syntax:** list.sort(key=None, reverse=False)
* **Example:**

numbers = [3, 1, 4, 2]

numbers.sort()

print(numbers) # Output: [1, 2, 3, 4]

numbers.sort(reverse=True)

print(numbers) # Output: [4, 3, 2, 1]

**c) reverse()**

* Reverses the **order of elements** in the list.
* **Example:**

fruits = ["apple", "banana", "cherry"]

fruits.reverse()

print(fruits) # Output: ['cherry', 'banana', 'apple']

**d) copy()**

* Returns a **shallow copy** of the list.
* **Example:**

fruits = ["apple", "banana"]

new\_fruits = fruits.copy()

print(new\_fruits) # Output: ['apple', 'banana']

**e) count()**

* Returns the **number of times an element occurs**.
* **Example:**

numbers = [1, 2, 2, 3]

print(numbers.count(2)) # Output: 2

**f) index()**

* Returns the **index of first occurrence** of an element.
* **Example:**

fruits = ["apple", "banana", "cherry"]

print(fruits.index("banana")) # Output: 1



**Advanced Python Tuples –**

**1. Definition Recap**

👉 Tuple is an **ordered, immutable, heterogeneous collection**.

* **Ordered:** Index system cha (0, 1, 2, …).
* **Immutable:** Once tuple create garyo, elements change, add, remove garna mildaina.
* **Heterogeneous:** Different datatypes store garna sakcha.
* **Syntax:**

my\_tuple = (1, "apple", 3.5, True)

**2. Nested Tuples**

Tuple bhitra **tuple** or **list** rakhna milcha.

nested = (1, 2, (3, 4), [5, 6])

print(nested[2]) # (3, 4)

print(nested[3][1]) # 6

⚡ Note: Tuple immutable ho, tara inside list chai mutable huncha (change garna milcha).

**3. Tuple Unpacking (Destructuring)**

Multiple elements lai directly variables ma unpack garna milcha.

point = (10, 20, 30)

x, y, z = point

print(x, y, z) # 10 20 30

**Extended Unpacking (with \*)**

numbers = (1, 2, 3, 4, 5)

a, \*b, c = numbers

print(a) # 1

print(b) # [2, 3, 4]

print(c) # 5

**4. Tuple Operations**

| **Operation** | **Example** | **Output** |
| --- | --- | --- |
| Concatenation (+) | (1,2) + (3,4) | (1,2,3,4) |
| Repetition (\*) | (1,2) \* 2 | (1,2,1,2) |
| Membership (in) | 2 in (1,2,3) | True |
| Indexing | t[0] | First element |
| Slicing | t[1:3] | Sub-tuple |

**5. Tuple Methods (Only 2)**

Tuples immutable huda list jasto methods hudaina. Available methods:

1. count(x) → number of times x occurs
2. index(x) → first index of x

t = (1, 2, 2, 3)

print(t.count(2)) # 2

print(t.index(3)) # 3

**6. Single Element Tuple**

Ek element ko tuple banaunda **comma mandatory** ho.

single = (5,)

print(type(single)) # <class 'tuple'>

not\_tuple = (5)

print(type(not\_tuple)) # <class 'int'>

**7. Immutability Deep Dive**

* Tuple itself immutable → t[1] = 99 → ❌ Error
* But inside mutable element (like list) change garna milcha:

t = (1, [2, 3], 4)

t[1][0] = 99

print(t) # (1, [99, 3], 4)

👉 Tuple pointer fixed huncha, tara inside list modify garna milcha.

**8. Tuple as Dictionary Keys**

* Immutable huda tuple lai **dictionary key** banauna milcha.
* List cannot be a key (mutable huney le).

my\_dict = {(1,2): "value"}

print(my\_dict[(1,2)]) # value

**9. Tuple Packing & Unpacking**

* **Packing:** Multiple values → tuple automatically

t = 1, 2, 3 # Packed tuple

* **Unpacking:** Tuple → multiple variables

a, b, c = t

print(a, b, c) # 1 2 3

**10. Named Tuples (Advanced Use)**

collections.namedtuple → Tuple with named fields.

* More readable than normal tuple.

from collections import namedtuple

Point = namedtuple("Point", ["x", "y"])

p = Point(10, 20)

print(p.x, p.y) # 10 20

**11. Use Cases of Tuples**

✔ Store fixed data (like coordinates, days of week)  
✔ As dictionary keys (immutable requirement)  
✔ Returning multiple values from function  
✔ Safer than lists (no accidental modification)  
✔ More memory efficient & slightly faster

**12. Difference: List vs Tuple**

| **Feature** | **List** | **Tuple** |
| --- | --- | --- |
| Mutability | ✅ Mutable | ❌ Immutable |
| Syntax | [] | () |
| Methods | Many (append, extend, pop, …) | Only count, index |
| Performance | Slower | Faster |
| Use Case | Dynamic, frequently changing data | Fixed, constant data |

**Set in Python**

**1. Definition**

👉 **Set** is a **built-in data type** in Python, which is:

* **Unordered** → elements ko order fixed hudaina.
* **Mutable** → elements add/remove garna milcha.
* **Unique elements only** → duplicates automatically remove huncha.

**Syntax:**

s = {1, 2, 3, 4}

print(s) # {1, 2, 3, 4}

Or use constructor:

s = set([1, 2, 2, 3])

print(s) # {1, 2, 3}

**2. Properties of Set**

* ✅ No duplicates
* ✅ Unordered (no indexing, no slicing)
* ✅ Mutable (change garna milcha)
* ✅ Can only contain **immutable (hashable) elements** (numbers, strings, tuples)
* ❌ Cannot contain mutable elements (list, dict, set)

**3. Creating Sets**

# Empty set

s = set() # ✅ Correct way

s2 = {} # ❌ This creates a dictionary, not a set

# From string

s = set("apple")

print(s) # {'a', 'l', 'e', 'p'}

**4. Common Set Methods**

**Add Elements**

* add(x) → Add single element
* update(iterable) → Add multiple elements

s = {1, 2}

s.add(3)

print(s) # {1, 2, 3}

s.update([4, 5])

print(s) # {1, 2, 3, 4, 5}

**Remove Elements**

* remove(x) → Removes element (error if not found)
* discard(x) → Removes element (no error if not found)
* pop() → Removes random element
* clear() → Removes all elements

s = {1, 2, 3}

s.remove(2)

print(s) # {1, 3}

s.discard(5) # No error even if not found

s.pop() # Removes random element

s.clear() # Set becomes empty

**5. Set Operations**

**Union (| or .union())**

a = {1, 2, 3}

b = {3, 4, 5}

print(a | b) # {1, 2, 3, 4, 5}

print(a.union(b)) # same result

**Intersection (& or .intersection())**

print(a & b) # {3}

**Difference (- or .difference())**

print(a - b) # {1, 2}

**Symmetric Difference (^ or .symmetric\_difference())**

print(a ^ b) # {1, 2, 4, 5}

**6. Other Useful Methods**

* len(s) → number of elements
* copy() → shallow copy of set
* issubset(other) → checks if set is subset
* issuperset(other) → checks if set is superset
* isdisjoint(other) → True if sets have no common elements

a = {1, 2}

b = {1, 2, 3}

print(a.issubset(b)) # True

print(b.issuperset(a)) # True

print(a.isdisjoint({4,5})) # True

**8. Limitation**

* ❌ No indexing or slicing (s[0] → error)
* ❌ Cannot store mutable elements (list, dict, set)

s = {1, [2,3]} # ❌ Error: unhashable type 'list'

**9. Use Cases**

* Removing duplicates from list
* Performing mathematical set operations (union, intersection, …)
* Membership testing (in is faster in sets than lists)

**10. Example: Removing Duplicates**

nums = [1, 2, 2, 3, 4, 4, 5]

unique\_nums = set(nums)

print(unique\_nums) # {1, 2, 3, 4, 5}

**11. Difference: Set vs Frozen Set**

| **Feature** | **Set** | **Frozen Set** |
| --- | --- | --- |
| Mutability | ✅ Mutable | ❌ Immutable |
| Methods | add(), remove(), pop()… | Only read-only (union, intersection, …) |
| Hashable | ❌ No | ✅ Yes |
| Dictionary Key | ❌ No | ✅ Yes |