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** 🔍

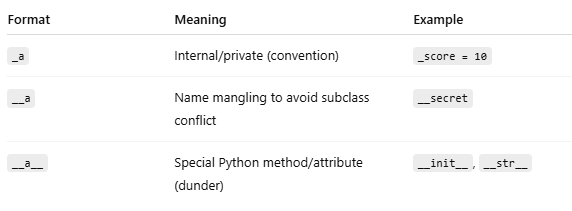
* Different cases help programmers **know what the variable represents**:
  + snake\_case → regular variable or function
  + PascalCase → class name
  + UPPER\_CASE → constant value

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 |



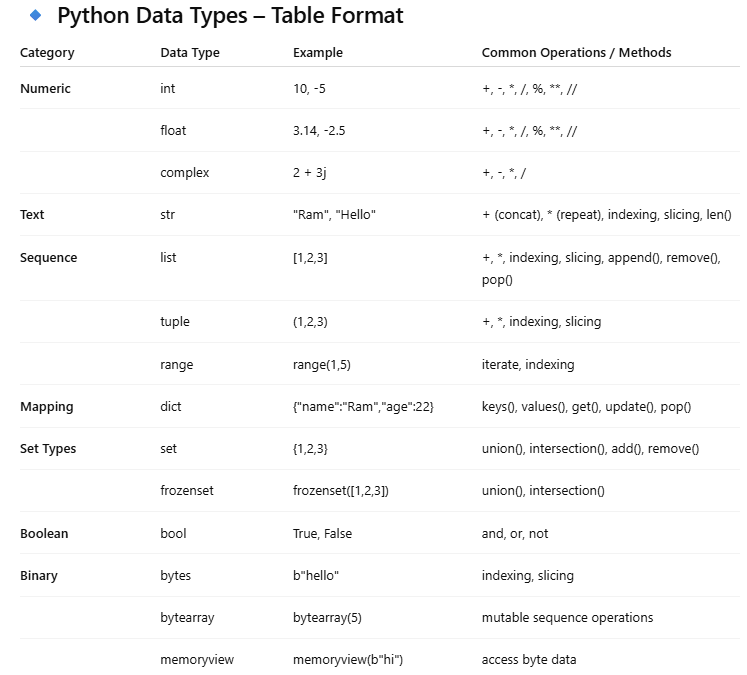
Paxadi ramro snaga details ma xa

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+)**

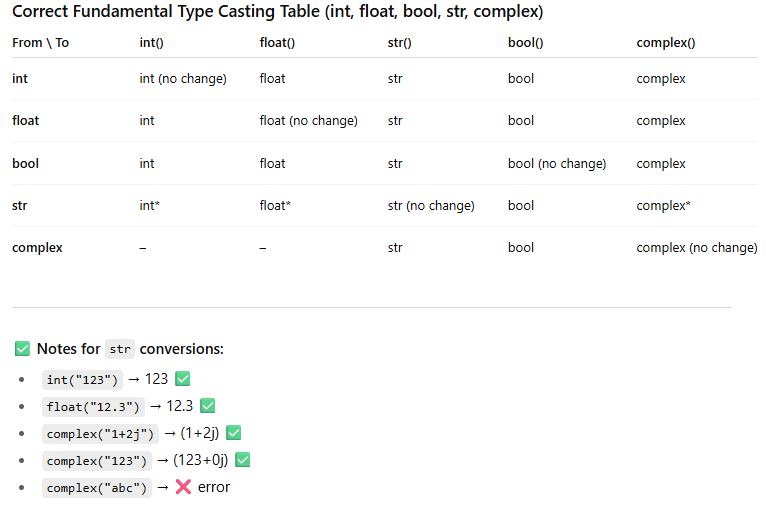
name = "Ram"

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