Variable :

A variable in Python is a named storage that holds data (a value).

It acts like a label or reference that points to an object stored in memory.

Variable Declaration rules :

1. Variable names can contain **letters (a–z, A–Z)**, **digits (0–9)**, and **underscore (_)**.
2. Variable names **cannot start with a digit**.
3. Variable names are **case-sensitive** (Age, age, AGE are different).
4. No special symbols (@, $, %, -, ., space) are allowed except _.
5. Variable names **cannot be a Python keyword** (if, for, class, while, etc.).
6. Variable names **cannot contain spaces** (use _ instead).
7. Variable names should be **short and meaningful** (best practice).
8. A variable can start with an **underscore (_) but not recommended** unless for special purposes

1. Allowed characters

- A variable name can contain **letters (a–z, A–Z)**, **digits (0–9)**, and **underscore (_)**.

```
name = "Ajay"
age1 = 25
total_amount = 5000
```

---

2. Cannot start with a digit

- Variable names **must not begin with a number**.

```
1value = 100   #  Invalid
value1 = 100   #  Valid
```

---

3. Case-sensitive

- Python treats uppercase and lowercase letters differently.

```
count = 10
Count = 20
print(count, Count)   # Output: 10 20
```

---

4. No special symbols except _

- Symbols like @, $, %, -, . are **not allowed**.

```
price$ = 50   #  Invalid
price_value = 50   #  Valid
```

---

5. Cannot use keywords

- Python **keywords** (reserved words like if, for, class, etc.) cannot be used as variable names.

```
if = 10       # Invalid
class = "A"   #  Invalid

class_name = "A"   #  Valid
```

6. Meaningful names (best practice)

- Always use **descriptive names** for readability.

```
x = 50       # Bad (unclear)
student_marks = 50   #  Good
```

7. No spaces allowed

- Use _ instead of spaces.

```
student name = "Ajay"   #  Invalid
student_name = "Ajay"   #  Valid
```

8. Length of variable name

- Technically no strict limit, but should be **short and meaningful**.

NOTE:

1. **Python always creates the VALUE (object) first**.
    - Example: when you write 10, Python creates an **integer object** in memory.
2. Then Python creates a **name (variable)** and makes it **point to that object**.

Example

- Step 1: Python creates an **integer object 10** in memory.
- Step 2: Python binds the **name x** to that object.

Datatype:

A data type defines the kind of value a variable can hold and what operations can be performed on that value.

Types of data type

1. Primitive
2. Non primitive

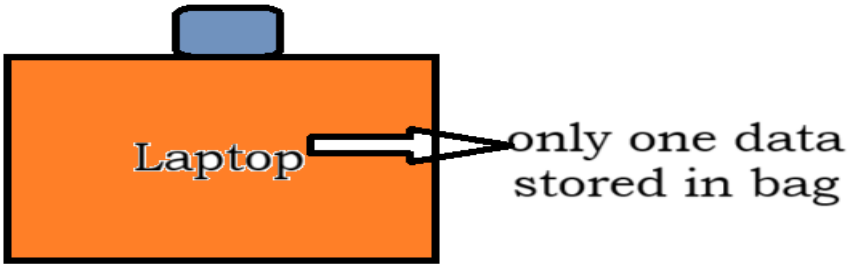
Primitive data type :

Primitive data type allows single value in single variable .

Or

Primitive data types are the **basic built-in data types** in Python that store **single simple values** (not collections).

1. **int** → integers
2. **float** → decimal numbers
3. **complex** → complex numbers
4. **bool** → Boolean values (True/False)
5. **str** → strings (text)



Int data type

The **int (integer)** data type in Python is used to represent **whole numbers** (numbers without a decimal point).

- It can be **positive**, **negative**, or **zero**.
- Python integers can be **very large** (no fixed size limit like in C or Java).

Ex 1:

```
b=10
print(b)
print(type(b))
print(id(b))
```

Ex 2:

```
n=-10
print(n)
print(type(n))
print(id(n))
```

Ex 3:

```
k=0
print(k)
print(type(k))
```

we can store binary and octal and hexadecimal numbers in int data type

ex:1

```
l=0b0101    # 01
print(l)
print(type(l))
```

OCTAL NUMBER

```
k=0o1234567
print(k)
print(type(k))
```

HEXADECIMAL

```
n=0x1239af
print(n)
print(type(n))
```

Variable declaration:

Ex :1

True1=1000 ➔ valid variable

Ex 2:

Student name='mohan' ➔ invalid

Student name='mohan' ➔ valid

Ex 3:

Snake Case

- Words are written in **lowercase**.
- Words are separated using an **underscore _**.
- Common in **Python** variable & function naming.

Examples:

```
student_name = "Ajay"
total_marks = 450
is_valid_user = True
```

Camel Case

- First word starts with **lowercase**.
- Every next word starts with an **Uppercase letter** (no spaces/underscores).
- Common in **JavaScript** variable & function naming.

Examples:

```
studentName = "Ajay";
totalMarks = 450;
isValidUser = true;
```

string data type :
A **string** in Python is a sequence of **characters** enclosed in **single quotes ' ', double quotes " ", or triple quotes ''' ''' / """ """.

```
name = "Ajay"
message = 'Hello Python'
paragraph = """This is a
multi-line string."""
```

Features

1. **Text Representation** – Stores characters, words, sentences.
2. **Immutable** – Cannot be changed after creation.
3. **Indexed & Ordered** – Supports positive and negative indexing.
4. **Length Calculation** – Use len() to get total characters.
5. **Iteration Supported** – Can loop through each character.

Differences between single quotes triple quotes

Feature	**Single Quotes (' or ")**	**Triple Quotes (''' or """)**
Lines Supported	Single line only	Multiple lines supported
Usage	Short strings	Long text, docstrings
Syntax	'Hello' or "Hello"	'''Hello''' or """Hello"""
Common Use	Variable values	Documentation, multi-line text

Ex: 1
```
g="full 'stack' developer"
print(g)
```

ex :2
```
h='123456789'
print(type(h))
print(id(h))
```

Ex 3: swap two variables

```
a=100
b=200
a,b=b,a
print(a)
print(b)
```

Float data type

- A **float** is a **number with a decimal point**.
- It represents **real numbers** (both positive and negative).

Ex :

```
x = 10.5
y = -3.14
z = 2.0
print(type(x))   # <class 'float'>
```

ex :2

```
a = 5.6
b = -10.75
print(a, b)   # 5.6 -10.75
```

Ex:3

```
value = 3.5e3   # 3.5 × 10³    # e= exponential    e=10
print(value)    # 3500.0
```

Boolean data type

- The **Boolean (bool)** data type represents **truth values**:
  - True
  - False
- It is often used in **conditions, comparisons, and logical operations**.
- Internally, True = 1 and False = 0.

Example:

```
x = True
y = False
print(type(x))   # <class 'bool'>
print(type(y))   # <class 'bool'>
```

Features of Boolean Data Type

1. **Only Two Values** → True or False.
2. **Subclass of int** → True == 1, False == 0.
3. **Result of Comparisons** → Any comparison (>, <, ==, !=) gives a Boolean.

Ex :2

```
print(10 > 5)   # True
print(2 == 3)   # False
```

ex 2:

```
print(int(True))   # 1
print(int(False))  #  0
```


10000
C{}DERS
Right path for a Bright Career.