Python Variables

1. What is a Variable?

A **variable** in Python is a symbolic name that refers to a memory location where data is stored. Unlike statically typed languages (e.g., C, Java), Python does not require explicit declaration of variable types, as it dynamically assigns types based on the assigned value.

Key Characteristics of Variables in Python

- Variables are dynamically typed, meaning you don't need to specify their type.
- A variable is created when a value is assigned to it.
- The type of a variable can change during execution.
- Variables store references to objects in memory, not the actual data itself.

---

2. Declaring and Assigning Variables

In Python, you assign values to variables using the **assignment operator (`=`)**.

Basic Assignment

```
x = 10          # Integer
y = 3.14        # Float
name = "Alice"  # String
is_valid = True # Boolean
```

Dynamic Typing Example

```
a = 100       # Initially an integer
a = "Hello"   # Now becomes a string
a = 3.14      # Now becomes a float
```

Python allows reassigning variables to different data types.

---

3. Naming Rules and Conventions for Variables

Rules for Naming Variables

1. Must **start** with a letter (`a-z`, `A-Z`) or an underscore (`_`).
2. Can **contain** letters, digits (`0-9`), and underscores (`_`).
3. Cannot **start** with a number.
4. Cannot use Python **reserved keywords** (e.g., `if`, `else`, `def`, `class`, etc.).
5. Python is **case-sensitive** (`myVar` and `myvar` are different variables).

Examples of Valid and Invalid Variable Names

```
valid_var = 10         # Valid
_var_name = "hello"   # Valid
var_123 = 3.14         # Valid

2nd_variable = 5       # Invalid (cannot start with a number)
class = "Python"       # Invalid (class is a reserved keyword)
my-variable = 20       # Invalid (hyphens are not allowed)
```

Best Practices for Naming Variables

- Use **descriptive names** for clarity.
- Use **snake_case** for variable names (`user_name` , `total_price`).
- Use **UPPERCASE** for constants (`PI = 3.14159`).

---

4. Assigning Multiple Variables in a Single Line

Python allows multiple assignments in a single line.

Assigning Different Values

```
x, y, z = 5, 10, 15
print(x, y, z)  # Output: 5 10 15
```

Assigning the Same Value to Multiple Variables

```
a = b = c = 100  # All three variables point to the same value
print(a, b, c)   # Output: 100 100 100
```

---

5. Data Types and Variables in Python

Python provides various built-in **data types**:

Data Type	Example	Description
Integer (`int`)	`x = 10`	Whole numbers
Float (`float`)	`y = 3.14`	Decimal numbers
Complex (`complex`)	`z = 3 + 5j`	Complex numbers
String (`str`)	`s = "Hello"`	Sequence of characters
Boolean (`bool`)	`is_valid = True`	True or False
List (`list`)	`lst = [1, 2, "apple"]`	Ordered, mutable collection
Tuple (`tuple`)	`tpl = (10, 20, "banana")`	Ordered, immutable collection

Data Type	Example	Description
Set (`set`)	`st = {1, 2, 3, "apple"}`	Unordered, unique items
Dictionary (`dict`)	`d = {"name": "John", "age": 25}`	Key-value pairs

Checking Data Type

To check a variable's type, use `type()`.

```python
x = 42
print(type(x))  # Output: <class 'int'>

y = "Hello"
print(type(y))  # Output: <class 'str'>
```

Type Conversion (Casting)

```python
a = "100"
b = int(a)     # Convert string to integer
c = float(b)  # Convert integer to float
d = str(c)     # Convert float to string

print(b, c, d)  # Output: 100 100.0 '100.0'
```

---

6. Global and Local Variables

Local Variables

A **local variable** is defined inside a function and is only accessible within that function.

```python
def my_function():
    local_var = "I am inside the function"
    print(local_var)

my_function()
# print(local_var)  # Error: local_var is not defined outside the function
```

Global Variables

A **global variable** is defined outside any function and can be accessed inside functions.

```python
global_var = "I am global"

def my_function():
    print(global_var)  # Accessible inside the function

my_function()
print(global_var)  # Accessible outside the function
```

Modifying Global Variables Inside a Function

```python
x = 10

def modify_global():
    global x  # Use the global keyword
    x = 20    # Modify the global variable

modify_global()
print(x)  # Output: 20
```

---

7. Deleting Variables

You can delete a variable using the `del` keyword.

```python
x = 100
del x
# print(x)  # Error: x is not defined
```

---

8. Mutable vs Immutable Variables

Type	Mutable?	Example
Numbers (`int` **,** `float` **,** `complex` **)**	No	`x = 10`
Strings (`str` **)**	No	`s = "Hello"`
Tuples (`tuple` **)**	No	`t = (1, 2, 3)`
Lists (`list` **)**	Yes	`lst = [1, 2, 3]`
Sets (`set` **)**	Yes	`st = {1, 2, 3}`
Dictionaries (`dict` **)**	Yes	`d = {"a": 1, "b": 2}`

Example:

```python
lst = [1, 2, 3]
lst[0] = 100  # Allowed

tup = (1, 2, 3)
# tup[0] = 100  # Error: Tuples are immutable
```

Conclusion

- Python variables are **dynamically typed**.
- They can store different types of **data**.
- **Global and local scope** affects variable accessibility.
- Constants are indicated using **uppercase names**.
- Mutable and immutable types affect how data is stored.