

PART 1:

What Are Variables?

Definition

- A variable is a container that temporarily stores data during the execution of a program.
- The value stored in a variable can change during the program's execution, which is why it is called a "variable."

Example:

- Imagine a glass of water. The glass is the container, and the water is the data stored in it.
- Similarly, in programming, a variable acts as a container that holds data temporarily.

Rules for Naming Variables

Key Rules:

- Start with a Letter or Underscore: Variable names must begin with a letter (a-z, A-Z) or an underscore (_).
Example: `username`, `_firstname`
- Cannot start with a Number: Variable names cannot begin with a digit.
Example: `1user` is invalid, but `user1` is valid.
- Use Only Letters, Numbers, and Underscores: Variable names can only contain letters, numbers, and underscores.
Example: `user_name`, `age1`
- Case-Sensitive: Variable names are case-sensitive.
Example: `FirstName` and `firstname` are considered two different variables.
- Avoid Reserved Words: Do not use Python's reserved keywords (e.g., `print`, `if`, `for`) as variable names.
Example: `print = 10` is invalid because `print` is a reserved word.

- **Meaningful Names:** Use meaningful and descriptive names for variables to improve code readability.
Example: Use `student_name` instead of `sn`.

Variable Naming Conventions

1. **Snake Case:** All letters are in lowercase, and words are separated by underscores.
Example: `user_name`, `first_name`.
2. **Camel Case:** The first word is in lowercase, and the first letter of each subsequent word is capitalized.
Example: `userName`, `firstName`.
3. **Pascal Case:** The first letter of each word is capitalized, and there are no separators.
Example: `UserName`, `FirstName`.

Initializing Variables

Syntax:

To create a variable, you need three things:

- **Variable Name:** The name of the variable (e.g., `fruit`).
- **Assignment Operator:** The equal sign (`=`) is used to assign a value to the variable.
- **Value:** The data you want to store in the variable (e.g., `apple`).

Example:

```
fruit = "apple"
```

Here, `fruit` is the variable name, and `"apple"` is the value assigned to it.

Example: Assigning an integer value:

```
a = 10
```

Here, `a` is the variable name, and `10` is the value assigned to it.

Data Types in Python

1. Integer

- Represents whole numbers (positive or negative) without decimal points.
- Example: 10, -5, 1000.

2. Float

- Represents decimal numbers (positive or negative).
- Example: 3.14, -0.001.

3. String

- Represents a sequence of characters enclosed in single or double quotes.
- Example: 'hello', "python".
- Note: Numbers enclosed in quotes are treated as strings.
 - Example: "123" is a string, not an integer.

4. Boolean

- Represents two values: True or False.
- Used in conditions and logical operations.
- Example: is_student = True.

Practical Examples

Example 1: Assigning Values to Variables

```
country = "Pakistan" # Assigning a string value
age = 25             # Assigning an integer value
price = 99.99        # Assigning a float value
is_active = True     # Assigning a Boolean value
```

Example 2: Case-Sensitivity

```
FirstName = "Ali"
firstname = "Ahmed"
```

Here, **FirstName** and **firstname** are two different variables.

Example 3: Invalid Variable Names

```
1user = "Ali" # Invalid - cannot start with a number
print = 10    # Invalid - cannot use reserved words
```

Best Practices for Variable Naming

1. Follow a Consistent Naming Convention

- Stick to one naming convention (e.g., snake case, camel case, or Pascal case) throughout your code.
- Example: If your project uses snake case, name all variables in snake case (e.g., `user_name`, `first_name`).

2. Use Meaningful Names

- Choose descriptive names that reflect the purpose of the variable.
- Example: Use `student_name` instead of `sn`.

3. Avoid Reserved Words

- Do not use Python's reserved keywords (e.g., `print`, `if`, `for`) as variable names.

PART 2:

Understanding Python Data Structures: Dictionary, List, Tuple, and Strings

Python Dictionary

A dictionary in Python is a collection of key-value pairs. Each key is unique, and values can be accessed, updated, or removed using various dictionary operations.

Creating a Dictionary

A dictionary is defined using curly braces `{}`:

```
person = {  
    "name": "John",  
    "age": 30,  
    "city": "New York"  
}
```

Accessing Elements in a Dictionary

We can retrieve values using keys:

```
print(person["name"]) # Output: John
```

Using `.get()` method is a best practice as it prevents exceptions:

```
print(person.get("name")) # Output: John  
print(person.get("gender", "Not Found")) # Output: Not Found
```

Adding and Updating Elements

If the key exists, its value is updated; otherwise, a new key-value pair is added.

```
person["country"] = "USA" # Adding new key-value pair
person["city"] = "London" # Updating an existing value
```

Removing Elements

We can remove elements using the `del` statement:

```
del person["age"]
print(person) # Output: {'name': 'John', 'city': 'London', 'country': 'USA'}
```

Checking Key Existence

To verify whether a key exists in a dictionary:

```
if "name" in person:
    print("Key exists") # Output: Key exists
```

Lists in Python

A list is a collection of ordered elements that allows duplicate values and supports modifications.

Creating a List

```
numbers = [10, 20, 30, 40]
```

Accessing Elements

Each element in a list has an index, starting from 0:

```
print(numbers[0]) # Output: 10
print(numbers[-1]) # Output: 40 (Last element using negative index)
```

Appending and Inserting Elements

Append adds elements to the end.

```
numbers.append(50)
```

Insert adds elements at a specific index.

```
numbers.insert(2, 15) # Inserts 15 at index 2
```

Removing Elements

`pop()` removes the last element.

```
numbers.pop()
```

Checking List Mutability

Lists are mutable, meaning elements can be modified after creation.

```
numbers[0] = 99 # Modifying the first element
```

Strings in Python

A string is a sequence of characters enclosed in quotes.

String Concatenation

```
first_name = "John"
last_name = "Doe"
full_name = first_name + " " + last_name # Output: John Doe
```

String Repetition

```
print("ha" * 3) # Output: hahaha
```

Accessing Characters in a String

```
greeting = "Hello"
print(greeting[0]) # Output: H
print(greeting[1:5]) # Output: ello (Slicing)
```

Using f-strings (Best Practice)

```
age = 25
print(f"My age is {age}") # Output: My age is 25
```

Tuples in Python

A tuple is similar to a list but immutable (cannot be modified after creation).

Creating Tuples

```
person_tuple = ("Bob", 30, "Engineer")
```

Accessing Elements

```
print(person_tuple[0]) # Output: Bob
print(person_tuple[1:]) # Output: (30, 'Engineer')
```

Tuple Immutability

Tuples do not support item assignment.

```
person_tuple[1] = 35 # This will raise an error
```

Tuple Unpacking

```
name, age, job = person_tuple  
print(name, age, job) # Output: Bob 30 Engineer
```