**Lecture's Description**

**PART 1:**

In this lecture series on Python, our focus today is on variables. We will explore what variables are, the rules for naming them, how to initialize them, and the different types of variables in Python. Variables are fundamental to programming as they allow us to store and manipulate data. Let’s dive into the details.

**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.

**2. 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.

**3. 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.

**4. 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.

**5. 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.

**6. 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

**7. 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.

**8. Conclusion**

**Summary**

* Variables are containers that store data temporarily during program execution.
* Follow specific rules and conventions for naming variables to ensure clean and readable code.
* Python supports various data types, including integers, floats, strings, and Booleans.

**Next Steps**

* In the next lecture, we will explore more data types and work on practical examples using VS Code.

-----------------------------------------------------------------------------------------------------------------------------------------------

**PART 2:**

In the last lecture, we covered the basics of variables in Python and discussed some fundamental data types. Today's lecture continues from the previous one, focusing more on data types in detail, with a hands-on practical approach. This lecture is designed to help you understand Python collections, their best practices, and their practical implementations.

**Today's Lecture Overview**

* Understanding Python collections in depth
* Hands-on practice of collections
* Best practices for using collections effectively
* Setting up and using a virtual environment for Python programming

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

**1. 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

**2. 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

**3. 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

**4. 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

**Setting Up a Virtual Environment in Anaconda**

To ensure a proper development environment, we use Anaconda and always work within a virtual environment. Follow these steps to set up your virtual environment:

**Open Anaconda Prompt**

Navigate to the Python Programming Directory:

cd path\_to\_your\_python\_programming\_folder

**Create a New Virtual Environment**

conda create -n farah python=3.10

**Activate the Virtual Environment**

conda activate farah

**Open VS Code**

code .

Once VS Code is open, confirm that your Python Notebook is running within the correct environment:

* Click on **Select Kernel** in the top-right corner of VS Code.
* Select **Python Environment** and choose **Farah**.
* Allow access if prompted for the first-time setup.

**Best Practices for Using Collections in Python**

To ensure efficient coding, follow these best practices:

* Use **lists** for ordered collections where elements can be modified.
* Use **tuples** for immutable data to protect against accidental changes.
* Use **sets** for unique elements and fast membership checks.
* Use **dictionaries** for key-value pairs when data retrieval is essential.

All best practices have been noted in your provided notebooks. Make sure to revise them and practice implementing these concepts in multiple ways.

**Homework & Next Steps**

* Practice the concepts covered in this lecture by implementing them in your system.
* Experiment with different use cases to strengthen your understanding.
* Review the best practices mentioned in the notebooks.
* Prepare for the next lecture by revising today's topics.

With practice, these concepts will become second nature and will greatly benefit you in your programming career.

**Conclusion**

This concludes today's lecture. We have learned about Python collections, set up a virtual environment, and understood the importance of best practices in Python programming. Keep practicing, and I will see you in the next lecture.