Python Prerequisite for DevOps

Topic 1: Variables & Assignments

Subtopic: Single Value vs Multiple Assignment

◈ What is a Variable?

- A **variable** in Python is like a “container” that stores data.
- Instead of remembering numbers or strings, we give them a name to use later.
- Think of it like a **label stuck on a box** – the box can hold something (a value), and we can change what’s inside later.

Example:

```
server_name = "web-server-01"

port = 8080
```

Here:

- server_name holds a **string** value.
- port holds a **number**.

◈ Single Value Assignment

In Python, assigning one value to one variable:

```
username = "devops_admin"
```

- The variable username is created.
- The value "devops_admin" is stored inside it.

◈ Multiple Assignment

Python allows assigning values to multiple variables in **one line**.

Case 1: Assign same value to multiple variables

```
x = y = z = "ready"
```

- All three variables (x, y, z) hold the value "ready".

Case 2: Assign different values to multiple variables

```
region, zone, instance = "us-east-1", "zone-a", "server-01"
```

- region = "us-east-1"
- zone = "zone-a"
- instance = "server-01"

This is called **unpacking assignment**. It’s very common in DevOps scripts.

DevOps Relevance

🔹 Why does this matter in DevOps?

Variables are the **building blocks of automation**. In DevOps, you often deal with:

- **Configuration files** (like storing server IPs, usernames, ports).
- **Script parameters** (passing credentials, cloud region, container names).
- **Reusable values** across automation scripts.

If you don’t use variables, you’ll end up **hardcoding** values again and again, making scripts rigid.
Using variables makes scripts **dynamic and flexible**.

🔹 DevOps Scenarios

✅ Scenario 1: Automating Cloud Deployment

```
cloud_provider = "AWS"

region = "us-east-1"

instance_type = "t2.micro"


print("Deploying to", cloud_provider, "in region", region, "with instance", instance_type)
```

Output:

```
Deploying to AWS in region us-east-1 with instance t2.micro
```

👉 Instead of typing these values repeatedly, store them in variables.
If tomorrow you shift to **Azure**, just change cloud_provider = "Azure".

✅ Scenario 2: Managing Multiple Servers

```
server1, server2, server3 = "10.0.0.1", "10.0.0.2", "10.0.0.3"

print("Connecting to servers:", server1, server2, server3)
```

Output:

Connecting to servers: 10.0.0.1 10.0.0.2 10.0.0.3

👉 Multiple assignment lets you declare all server IPs in one line.
Later, you can loop through them for deployment.

---

3. Tasks for Students

👉 **Task 1: Single Assignment**

- Create a variable called tool_name and assign it the value "Docker".
- Print: Tool in use: Docker.

👉 **Task 2: Multiple Assignment**

- Assign "Dev", "Test", "Prod" to three variables: env1, env2, env3.
- Print them in one line:
- Environments available: Dev, Test, Prod

👉 **Task 3: Same Value Assignment**

- Assign "active" to three variables status1, status2, status3.
- Print them one by one.

👉 **Task 4: DevOps Mini Scenario**

- Assign:
    - provider = "AWS"
    - region = "ap-south-1"
    - service = "EC2"
- Print:

Launching EC2 service in ap-south-1 region on AWS

Variables & Assignments

◆ What is a Variable?

- A **variable** is a name given to a value stored in memory.
- Think of it like a **label on a container** — the label is the variable name, and the container holds a value.
- Python allows us to create variables simply by assigning a value using the = operator.

Example:

```
username = "student01"

age = 22
```

Here:

- username is a variable storing a string "student01".
- age is a variable storing a number 22.

◆ Single Value Assignment

Assign one value to one variable:

```
tool = "Python"

print(tool)
```

Output:

```
Python
```

◆ Multiple Assignment

Case 1: Same value to multiple variables

```
x = y = z = 100

print(x, y, z)
```

Output:

```
100 100 100
```

Case 2: Different values to multiple variables

```
a, b, c = 10, 20, 30

print(a, b, c)
```

Output:

```
10 20 30
```

This is called **unpacking assignment**.

Python Data Types

Python has different **data types** depending on the kind of value stored in a variable.

◈ Numbers

- **Integer (int)** → Whole numbers: 5, -10, 100
- **Float (float)** → Decimal numbers: 3.14, 10.0
- **Complex (complex)** → Numbers with imaginary part: 2+3j

Example:

```
x = 10      # int
y = 3.14    # float
z = 2 + 3j  # complex
```

◈ Strings

- A **string** is a sequence of characters inside quotes (' ' or " ").
- Strings are commonly used to store names, messages, or any text data.

Example:

```
name = "Python"
print(name.upper())  # Convert to uppercase
```

Output:

```
PYTHON
```

◈ Lists

- Ordered collection of items.
- **Mutable** → can change after creation.
- Useful for storing multiple related values.

Example:

```
numbers = [1, 2, 3, 4, 5]
numbers.append(6)   # Add new element
print(numbers)
```

Output:

```
[1, 2, 3, 4, 5, 6]
```

◆ Tuples

- Ordered collection of items.
- **Immutable** → cannot change once created.

Example:

```
coordinates = (10, 20)
print(coordinates[0])  # Access first element
```

Output:

```
10
```

◆ Dictionaries

- Store values as **key-value pairs**.
- Useful for mapping relationships.

Example:

```
student = {"name": "Alice", "age": 21, "marks": 85}
print(student["name"])
```

Output:

```
Alice
```

Python Practice Tasks

❖ **Variables & Assignments**

1. Assign "Python" to a variable called language and print it.
2. Assign values 100, 200, 300 to three variables x, y, z in one line. Print them.
3. Assign the same value 0 to three variables a, b, c. Print their sum.

❖ **Numbers**

1. Write a program that takes two numbers and prints their **sum, difference, product, and quotient**.
2. Calculate the **square and cube** of a given number.
3. Convert temperature from **Celsius to Fahrenheit** using:
4. F = (C * 9/5) + 32

❖ **Strings**

1. Take a string input and print it in **reverse order**.
2. Count how many vowels are in a given string.
3. Check if a string is a **palindrome** (same forwards and backwards).

❖ **Lists**

1. Create a list of 10 numbers and print only the **even numbers**.
2. Find the **largest and smallest** number in a list without using max() or min().
3. Merge two lists into one and print the result.

❖ **Tuples**

1. Create a tuple of 5 numbers and calculate their **sum**.
2. Check if a given element exists in a tuple.
3. Convert a tuple into a list, add one element, then convert back to a tuple.

❖ **Dictionaries**

1. Create a dictionary of 3 students with their marks. Print the **average marks**.
2. Write a program to count how many times each character appears in a string (frequency dictionary).
   Example: "hello" → {'h':1, 'e':1, 'l':2, 'o':1}
3. Merge two dictionaries into one.