

Python Virtual Environment – Edge Cases

Test Case 1: Duplicate environment name

Practice Task: Create virtual environment with same name twice

Input: `python -m venv myenv`

Steps:

- Open command prompt / terminal
- Navigate to project directory
- Run command: `python -m venv myenv`
- Run the same command again

Expected Output: Expected Output: Environment should not be recreated; system should indicate directory already exists or overwrite warning

How to avoid these/ Actual output

- When the command is run again, Python detects the existing folder.
- The environment is not recreated.
- Existing files remain unchanged.
- User must delete manually to recreate.

Test Case 2: Python not installed

Practice Task: Create a virtual environment when Python is not installed

Input:

`python -m venv myenv`

Steps:

- Open command prompt / terminal
- Navigate to any directory
- Run command: `python -m venv myenv`

Expected Output:

The system should display an error indicating that Python is not recognized or not installed.

How to avoid these/ Actual output

- System cannot find Python executable.
- Environment is not created.

Test Case 3: Permission denied

Practice Task: Create a virtual environment in a restricted directory

Input:

```
python -m venv myenv
```

Steps:

- Open command prompt / terminal
- Navigate to a protected directory (e.g., system folder)
- Run command: `python -m venv myenv`

Expected Output:

The system should deny permission and display an access-related error message.

Actual Output

- System throws permission denied error.
- No files are created.

What is required / Will Happen?

- Os will not permit us to access the files unless it has admin rights

Test Case 4: Invalid env name

Practice Task: Create a virtual environment with an invalid name

Input:

```
python -m venv my/env
```

Steps:

- Open command prompt / terminal
- Navigate to project directory
- Run command: `python -m venv my/env`

Expected Output:

The system should reject the invalid environment name and display an error.

Actual Output :

- Python interprets the name as an invalid path or directory structure.
- An error is raised indicating an invalid directory or path.

What is required?

Avoid symbols like @, #, or \$. Use alphanumeric characters and underscores.

Test Case 5: Space in env name

Practice Task: Create a virtual environment with a space in its name

Input:

```
python -m venv my env
```

Steps:

- Open command prompt / terminal
- Navigate to project directory
- Run command: python -m venv my env

Expected Output:

The system should fail or require proper quoting of the environment name.

Actual Output:

- Shell treats space as separator.
- Command fails or misinterprets name.
- Environment not created correctly.
- Quotes are required.

Test Case 6: Disk full

Practice Task: Create a virtual environment when disk space is insufficient

Input:

```
python -m venv myenv
```

Steps:

- Open command prompt / terminal
- Navigate to project directory
- Ensure disk has very low or no free space
- Run command: python -m venv myenv

Expected Output:

The system should fail to create the environment and display a disk space error.

Actual Output:

- **System runs out of storage.**
- Partial files may exist.

Test Case 7: Unsupported Python version

Practice Task: Create a virtual environment using an unsupported Python version

Input:

```
python -m venv myenv
```

Steps:

- Open command prompt / terminal
- Use an outdated or incompatible Python version
- Run command: `python -m venv myenv`

Expected Output:

The system should display an error indicating that the Python version is not supported.

Actual Output:

- Older Python lacks venv support.

Test Case 8: Activate env without creation

Practice Task: Activate a virtual environment that does not exist

Input:

```
myenv\Scripts\activate (Windows) / source myenv/bin/activate (Linux)
```

Steps:

- Open command prompt / terminal
- Navigate to project directory
- Attempt to activate `myenv` without creating it

Expected Output:

The system should display an error indicating that the environment does not exist.

Actual Output:

- Activation script is missing.
- Shell reports file not found.
- Environment does not activate.
- Creation is required first.

Test Case 9: Wrong activation Cmd

Practice Task: Use an incorrect command to activate the virtual environment

Input:

```
activate myenv
```

Steps:

- Open command prompt / terminal
- Navigate to project directory
- Run an incorrect activation command

Expected Output:

The system should display an error indicating an invalid or unknown command.

Actual Output:

- The command is not recognized by the shell.
- **An error such as command not found or not recognized is displayed.**

Test Case 10: Deleting Active Virtual Environment

Practice Task: Delete a virtual environment while it is active

Input:

```
rm -rf myenv (Linux) / Delete folder manually (Windows)
```

Steps:

- Activate the virtual environment
- Attempt to delete the environment directory
- Continue using the terminal

Expected Output:

The system should restrict deletion or cause the environment to become invalid.

Actual Output:

- The environment folder is deleted while still active.
- The terminal continues to reference a non-existent environment.
- Subsequent Python or pip commands may fail.
- Restarting the shell or deactivating resolves the issue.

Test Case 11: Nested Virtual Environments

Practice Task: Create a virtual environment inside another virtual environment

Input:

```
python -m venv innerenv
```

Steps:

- Activate an existing virtual environment
- Navigate inside its directory
- Run command: `python -m venv innerenv`

Expected Output:

The system should allow creation but may cause confusion or conflicts.

Actual Output:

- The inner virtual environment is created successfully.
- Python uses the currently active interpreter to create it.

What does it leads to?

- This may lead to dependency confusion.

Test Case 12: Wrong Python Executable Used

Practice Task: Create a virtual environment using an incorrect Python executable

Input:

- `python3.6 -m venv myenv` (when version is not installed)

Steps:

- I. Open command prompt / terminal
- II. Navigate to project directory
- III. Run the command with a wrong or unavailable Python version

Expected Output:

- The system should display an error indicating the Python executable is not found or invalid.

Actual Output

- The system fails to locate the specified Python executable.
- Version which may be installed is not supported
- The virtual environment is not created
- Using a valid installed Python version resolves the issue.

Test Case 13: Corrupted venv folder

Practice Task: Use a virtual environment with missing or corrupted files

Input:

- `python -m venv myenv`

Steps:

- I. Create a virtual environment
- II. Manually delete or modify files inside the *myenv* folder
- III. Try to activate or use the environment

Expected Output:

- The system should fail to activate or use the corrupted environment.

Actual Output

- If a crash happens during pip install, the site-packages might be half-written.
- Best way to fix is to Delete and recreate to run.

Test Case 14: Read-only folder

Practice Task: Create a virtual environment in a read-only folder

Input:

- `python -m venv myenv`

Steps:

- I. Navigate to a read-only directory
- II. Attempt to create a virtual environment
- III. Run the venv command

Expected Output:

- The system should block creation and show a write-permission error.

Actual Output / Answer: Virtual env needs to write scripts to the bin or Scripts folder.

Test Case 15: Missing pip

Practice Task: Create a virtual environment without `pip` installed

Input:

- `python -m venv myenv --without-pip`

Steps:

- I. Open command prompt / terminal
- II. Navigate to project directory
- III. Create the environment using `--without-pip`
- IV. Activate the virtual environment

Expected Output:

- The environment should be created, but `pip` should not be available.

Actual Output:

- Sometimes `python -m venv --without-pip` is called, leaving you with no way to install packages.

Test Case 16: Old Pip Version

Practice Task: Use an old `pip` version inside a virtual environment

Input:

- `pip install <package>`

Steps:

- I. Create and activate a virtual environment
- II. Check pip version
- III. Attempt to install a modern package

Expected Output:

- The system should warn about an outdated pip version or fail installation.

Actual Output:

- Pip displays a notice to upgrade.
- Some packages may not been install correctly.
- Always run `python -m pip install --upgrade pip` immediately after activation.

Test Case 17: Env not activated

Practice Task: Run Python packages without activating the virtual environment

Input:

- `python app.py`

Steps:

- I. Create a virtual environment
- II. Do not activate the environment
- III. Run a Python application

Expected Output:

- The system should use global Python and may miss dependencies.

How to resolve?

- Sometimes you install packages, but they go to your Global Python, this leads to **fragment the files**.

- Installed Virtual Env packages are not found.

Test Case 18: PATH Not Updated

Practice Task: Use Python after installation without updating PATH

Input:

- `python -m venv myenv`

Steps:

- I. Install Python without selecting *Add Python to PATH*
- II. Open command prompt / terminal
- III. Run the venv command

Expected Output:

- The system should fail to recognize the Python command.

Actual Output / Answer:

- The python command is not recognized by the system.
- An error message indicating missing PATH configuration is shown.
- The virtual environment is not created.
- Updating PATH or reinstalling Python fixes the issue.

Test Case 19: Antivirus Blocking Virtual Environment Scripts

Practice Task: Create a virtual environment when antivirus blocks script execution

Input:

- `python -m venv myenv`

Steps:

- Open command prompt / terminal
- Navigate to project directory
- Run the virtual environment creation command

Expected Output:

- The system should display an error or block script execution due to antivirus restrictions.

Actual Output / Answer:

- Antivirus software blocks execution of venv scripts.
- Environment creation may fail or be incomplete.
- Security warning or quarantine message is shown.
- Allowing Python in antivirus settings resolves the issue.

Test Case 20: PowerShell Execution Policy Restriction

Practice Task: Activate a virtual environment when PowerShell execution policy is restricted

Input:

.\myenv\Scripts\Activate.ps1

Steps:

- Open PowerShell
- Navigate to project directory
- Attempt to activate the virtual environment

Expected Output:

- The system should block script execution and display an execution policy error.

Actual Output / Answer:

- PowerShell blocks the activation script.
- An execution policy error message is displayed.
- The virtual environment is not activated.
- Changing execution policy allows activation

Test Case 21: Renamed Virtual Environment Folder

Practice Task: Activate a virtual environment after renaming its folder

Input:

• myenv\Scripts\activate

Steps:

- Create a virtual environment
- Rename the environment folder
- Attempt to activate using the old name

Expected Output:

- The system should fail to activate the renamed environment.

Actual Output / Answer:

- The activation path no longer exists.
- An error such as “**The system cannot find the path specified**” is shown.
- The virtual environment is not activated.
- Paths must match the renamed folder.

Test Case 22: Same Environment Name in Different Paths

Practice Task: Create virtual environments with the same name in different directories

Input:

- `python -m venv myenv`

Steps:

- Navigate to Project A directory
- Create virtual environment named myenv
- Navigate to Project B directory
- Create another environment named myenv

Expected Output:

- The system should allow creation since paths are different.

Actual Output:

- Both virtual environments are created successfully.
- Each environment exists in its own directory.
- There is no conflict between the environments.
- Activation depends on the current path.

Test Case 23: Running Application Without Virtual Environment

Practice Task: Run a Python application without activating the virtual environment

Input:

- `python app.py`

Steps:

- Create a virtual environment
- Do not activate the environment
- Run the Python application

Expected Output:

- The application may fail due to missing dependencies.

Actual Output:

- The global Python interpreter is used.
- Required packages from venv are not found.
- Import errors may occur.
- Activating the environment resolves the issue.

Test Case 24: Creating Virtual Environment Inside Docker Container

Practice Task: Create a virtual environment inside a Docker container

Input:

- `python -m venv myenv`

Steps:

- Start a Docker container
- Navigate to application directory inside the container
- Run the venv creation command

Expected Output:

- The virtual environment should be created successfully inside the container.

Actual Output:

- The virtual environment is created inside the container filesystem.
- No conflicts occur with the host system.
- Environment exists only within the container.
- Rebuilding the container removes the environment.

Issues occur:

In Docker, the container *is* the environment, so adding a `venv` inside a container just adds extra layers and complexity.

Test Case 25: Deleted python.exe After Environment Creation

Practice Task: Use a virtual environment after deleting the base Python executable

Input:

- `python app.py`

Steps:

- Create a virtual environment
- Delete or uninstall the system Python
- Attempt to activate or use the virtual environment

Expected Output:

- The virtual environment should fail to run Python commands.

Actual Output / Answer:

- The virtual environment depends on the base Python installation.
- Python commands fail after deletion.
- Activation may succeed but execution fails.
- Reinstalling Python resolves the issue.