# Functions, Modules, Packages & Virtual Environment

**Functions**

- A function is a block of code that performs a specific task and it only runs when it is called.
- A function can be called(invoked) any number of times within the program and thereby the code/ functionality written in a function can be reused without having to rewrite it.
- Function break down tasks into smaller units of code.
- We can define a function in Python, using the **def** keyword
- A function can accept input (parameters), performs the required logic and and return an output (return value).

**Example 1 – Functions define and call them**

File name – calculator.py

```
n1=20
n2=15

def add():
        sum = n1+n2                                         # defining function
        print("Sum is:",sum)                                # defining function logic

def sub():
        subtract = n1-n2
        print("Subtraction is:",subtract)

def multi():
        multiplication = n1*n2
        print("Multiplication is:",multiplication)

add()                                                     # calling the function
sub()
multi()
```

**Example 2 – Define Functions, pass parameters in them and return the output**

file name - basic_calci.py

```
def add(n1, n2):
    sum = n1+n2
    return sum

def add(n1, n2):
   addition = n1+n2
   return addition

def sub(n1, n2):
   substract = n1-n2
   return substract

def multi(n1, n2):
   multiplication = n1*n2
   return multiplication

print(add(5,10))
print(sub(20,5))
print(multi(100,5))
```

Output –
```
15
15
500
```
-----------------------------------------------------------------------------------------------------------------------

**Module**

A **module** in Python is a file contains Python code (like functions, classes, variables) that can be used in other Python programs.

“import” keyword is used to import external modules or libraries to access their functions, classes, or variables in our script/ program.

“from” keyword is used to import a specific function of a program in another program

**Benefits of using Module –**

**Code Re-usability -** Once you write a module, you can reuse it in multiple programs.

**Modularization -** Modules help you split your code into separate files based on functionality. This makes your code easier to organize, understand, and maintain

e.g – we created a file basic_calci.py above which has a basic calculator function i.e addition, subtraction , multiplication. Now we want to create a program of advance calculator, instead of writing code for addition, substration and multiplication in advanced_caclci.py, we use the code of the file basic_caclci  as module.

e.g – file name – advance_calci

```
import basic_calci                                # Importing the basic_calci module
print(basic_calci.add(5,10))                      # Calling the functions from the basic_calci module
printbasic_calci.sub(5,3))

print(basic_calci.multi(5,2))
```

Output –
```
15
15
100
```

**Issue 1 –** The output of advance_caclci.py should be -

```
15
2
10
```

But the output coming is –
```
15
15
100
```

**WHY?**

Ans – Add if __name__ == "__main__": in basic_calci.py python file as below –

```
if __name__ == "__main__":
    print(add(5,10))
    print(sub(20,5))
    print(multi(100,5))
```

if __name__ == "__main__": = it tells python to only run this part of the code when this file (basic_calci.py) is run directly, and not when it being imported into another file, if we do not include this then as we run advance_calci.py, it will also run print(add(5, 10)) from basic_calci.py because it gets imported — and you might not want that.

Now the output of advance_calci.py –

```
15
2
10
```

Issue 2 – We imported basic_calci.py python code in advance_calci.py file when both files are in same location. What if basic_calci.py is in D://Python Folder A and advance_calci.py is in E://Advancd Python Folder i.e both files are in different locations –

Ans - this is a very common scenario when our Python project files are organized in different folders. We want to import basic_calci.py (from D://Python Folder A) into adv_calci.py (which is in E://Advanced Python older).
But Python doesn't know where to find it by default — because they're in different directories.

**Solution 1 - Set the PYTHONPATH environment variable**

1. On Windows, go to: System Properties → Environment Variables
Under "User variables" or "System variables", add a new variable:
Name: PYTHONPATH
Value: D:\Python Folder A

2. Restart your code editor (like VS Code or PyCharm) to apply it.

Now you can simply do: import basic_calci as calculator

**Solution 2 - Modify sys.path in adv_calci.py**

```
import sys

sys.path.append("D:/Python Folder A")  # Path to where basic_calci.py lives

import basic_calci as calculator

print(calculator.add(10, 20))
```

**Note – Issue 2 is very common problem to be faced in real time**

**Packages**

A **package** is a collection of Python code (functions, classes, modules) that someone else has written and we can use it.

e.g - boto3 is a python package provided by AWS. boto3 lets our Python code **interact with AWS services** like EC2, S3, Lambda, DynamoDB, etc. It helps you **create, manage, and delete AWS resources** using Python.

**PYPI – Python Package Index.** Consider PYPI as a Play Store/ App Store for python packages.

google - [https://pypi.org](https://pypi.org), we can search, find, explore ready-made python packages.

pip – pip is Python's package manager. It connects our machine to PyPI.

We use pip to **install packages** from PyPI.

We can also use it to **uninstall**, **upgrade**, or **list** installed packages.

Syntax - pip install package_name

e.g -

| | |
|---|---|
| **Install boto3 package of AWS -** | **pip install boto3** |
| **Install any specific version of package -** | **Pip install boto3==1.18.0** |
| **Upgrade the latest version of boto3 -** | **pip install --upgrade boto3** |
| **Uninstall boto3 -** | **pip uninstall boto3** |
| **To see list of packages installed in our python code -** | **pip list** |

If pip is not installed, google - install pip for python and download pip as per our OS architecture.

| | |
|---|---|
| **To ensure pip is installed** | **pip3 --version** |

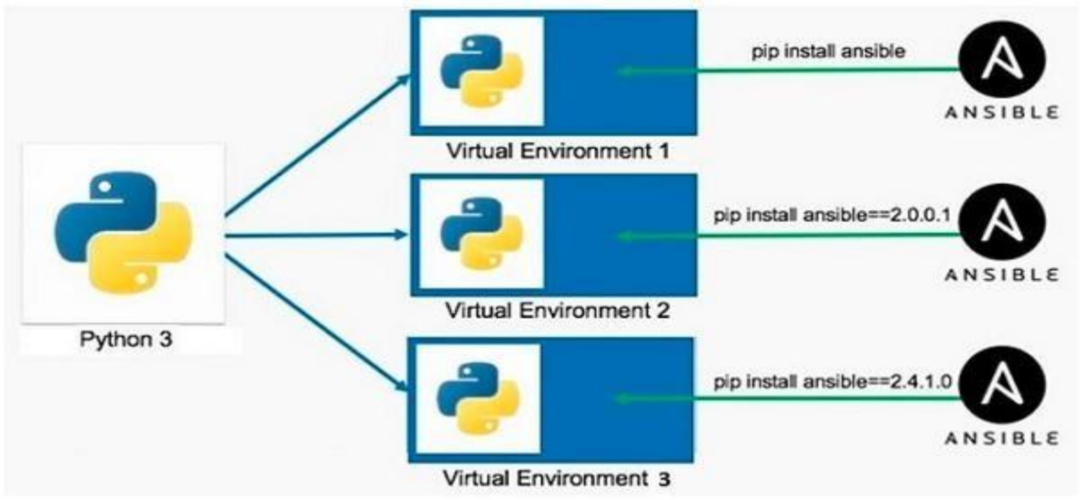
**Virtual Environment**

Virtual Environment creates logical separation for to use the python packages. In real time DevOps engineers used to get a common VM to work. Now problem arise is -

For Project 1, Ansible package 2.0.0.1 version is required and at the same time for another Project 2 (say) Ansible package 2.4.1.0 version is required. But we have a common VM to work. If Ansible 2.0.0.1 is installed then code of Project2 fails and vice-versa for Project 1 code if Ansible 2.4.1.0 version is installed.

This conflict can be avoided by providing two isolated environments of Python in the same machine. These are called virtual environment so that each environment can install the desired version of the package and the installed version is only specific to the environment in which it is installed.

Below diagram showing purpose of virtual environment in Python -

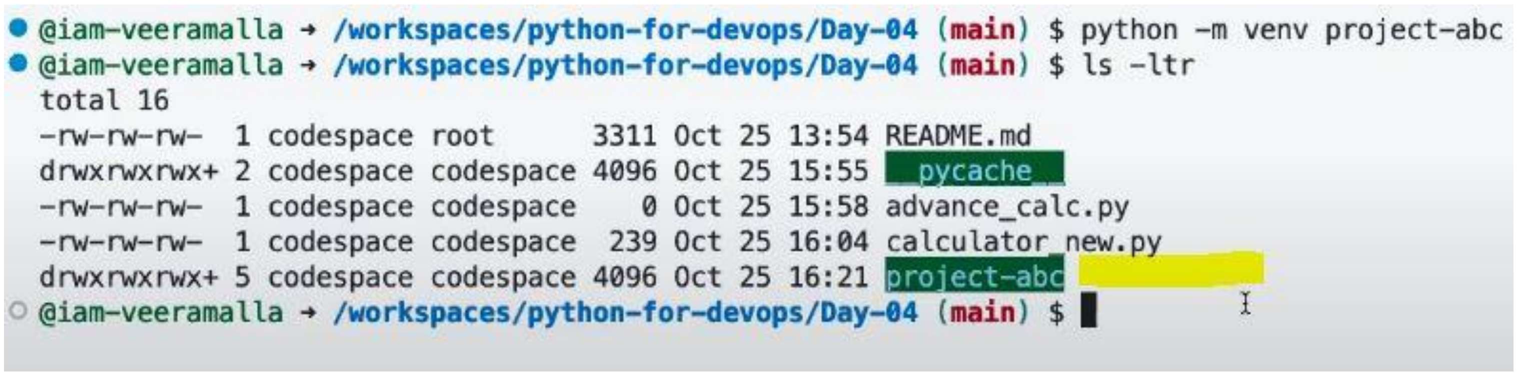


How to create and use Virtual Environment in Python -

Step 1 - Install Virtual Environment tool --> pip install virtualenv

Step 2 - Create project folder --> python -m venv project-abc

As we execute the above command, a folder with name Project-ABC is create, we can check in terminal by "ls" command as below –

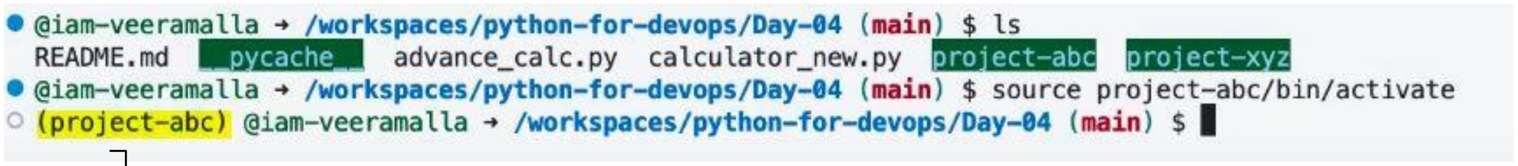


```
● @iam-veeramalla → /workspaces/python-for-devops/Day-04 (main) $ python -m venv project-abc
● @iam-veeramalla → /workspaces/python-for-devops/Day-04 (main) $ ls -ltr
  total 16
  -rw-rw-rw-  1 codespace root      3311 Oct 25 13:54 README.md
  drwxrwxrwx+ 2 codespace codespace 4096 Oct 25 15:55 __pycache__
  -rw-rw-rw-  1 codespace codespace    0 Oct 25 15:58 advance_calc.py
  -rw-rw-rw-  1 codespace codespace  239 Oct 25 16:04 calculator_new.py
  drwxrwxrwx+ 5 codespace codespace 4096 Oct 25 16:21 project-abc
○ @iam-veeramalla → /workspaces/python-for-devops/Day-04 (main) $
```

Create another folder for another Project - python -m venv project-xyz

Step 3 - SSH to the folder --> source project-abc/bin/activate

As we execute above command we entered in project-abc folder, here install Ansible 2.0.0.1 package then this package libraries are specific to the project-abc only as below



```
● @iam-veeramalla → /workspaces/python-for-devops/Day-04 (main) $ ls
  README.md  __pycache__  advance_calc.py  calculator_new.py  project-abc  project-xyz
● @iam-veeramalla → /workspaces/python-for-devops/Day-04 (main) $ source project-abc/bin/activate
○ (project-abc) @iam-veeramalla → /workspaces/python-for-devops/Day-04 (main) $
```

the yellow highlighted portion showing we are in project-abc folder

any package installed here via pip will be for project-abc folder and no interaction with project-xyz folder.