Iraqi Journal for Computer Science and Mathematics

Journal Homepage: http://journal.esj.edu.iq/index.php/IJCM

ISSN: 2788-7421

New Virtual Environment Based on Python Programming

Omar Abdulwahabe Mohamad[1]

[1]Computer Department, College of Education, Al-Iraqia University, Baghdad, Iraq

*Corresponding Author: Omar Abdulwahabe Mohamad

DOI: https://doi.org/10.52866/ijcsm.2022.02.01.012

Received April 2022; Accepted May 2022; Available online June 2022

ABSTRACT: Python is an amazing language for large and complicated programming ventures. This language uses codes that are simple to use or modify for any software programmer. In this paper, one annoying issue during the project expansion was described. Once a package is updated to the new version, this package is no longer compatible with previous projects, causing trouble when working on the program. To solve this problem, Python supports the programmer with a function called virtual environment. This function can construct an independent environment for any venture to avoid conflict. Various steps have been explained to create a virtual environment in Python. Different examples have then shown the benefits of Python language to building shapes, animations, and interactions.

Keywords: Virtual Environment; Python; Built-in Functions; User-defined Functions

1. INTRODUCTION

Python, as most other current programming tools, has its very own extraordinary method for storing, resolving the package, and downloading. While this has its points of interest, some intriguing choices made around package storage and resolution type lead to certain issues—especially about where and how packages are put away [1]. To keep the dependencies required via various projects, the virtual environment tool has been used to create isolated python virtual environments for each project. Most Python developers used these important tools [2].

During project development, one annoying issue is that once packages are updated to new versions, those packages are no longer compatible with previous projects, causing troubles when running the program. Python virtual environment, virtualenv, can solve the problem by constructing an independent environment for each project to avoid conflicts. In addition, using a virtual environment allows projects to be easily transferred or shared with other developers. The entire project only needs to be copied and its built-in virtual environment activated without being troubled by installing all required packages for the project [3]. A virtual environment write as (virtualenv) can find the solution to dependency conflicts in various projects. The basic idea of this solution summarizes by creating an isolated environment for each project which contains all packages for the specific project [4].

The most difficult aspect of installing a particular software package is usually to obtain all the dependencies right. That is, the package requires the existence of a lot of other packages on the computer system. These packages also have their dependencies. In addition, some packages only work with certain versions of other packages. Obtaining dependencies and software versions right quickly becomes a challenging problem. Even more critical is that installing a new set of packages brings in other versions of some software that affect the behavior of previously installed software on the computer system [5]. To be specific, package **A** depends on packages **B** and **C** in their respective versions 1.0 and 1.2 or newer. Package **D**, which also requires package **C**, is then installed but in an older 0.6 version. Package **C** in v0.6 overwrites v1.2, causing package **A** to break [5].

In this paper, a good solution to the problem of incompatible dependencies, and to handle dependencies in general, is to create an isolated virtual environment where the packages in the versions can be installed. The system of the current

study can have several such isolated environments and none of them interfere with the global software system. Also, an environment can be easily deleted when it is no longer needed [5].

The rest of the paper is systematically arranged as follows: Section 1 is the introduction. Section 2 presents the Python functions. Section 3 presents the steps to create a new virtual environment. Then, Section 4 shows the experimental work, and Section 5 describes different examples using the virtual environment in Python. Finally, Section 6 presents the conclusion.

2. PYTHON FUNCTIONS

The codes used to perform a specific action can be named a function. The function gives the best modularity for the applications with a high level of codes used more than once. Python supports various built-in functions, e.g., print, call, and so on. Also, own functions can be created by the user. These functions are known as user-defined functions [6].

The functions can be defined to supply the desired functionality. The basic rules which are used to explain the function are as follows [6]:

- The keyword **def** is used to begin the function blocks and the function name and parentheses ( ).
- A function name is used to uniquely identify it. Function naming follows the same rules of writing identifiers in Python
- The input must be written within the parentheses.
- An optional statement can be defined in the first statement of the function. The (docstring) denotes the documentation string of a function.
- The colon : indicates the start of the code in a function.
- The function body makes up of one or more Python statements. These statements used the same level of common four spaces [7].
- The [expression] indicates the existence of the function and sends hints to the caller which is called a return statement.

The functions can be classified into the following types: [7]

1. **Python Built-in Functions**: These functions are constructed in Python. Many ready-to-use functions are available. The functions explained above are called built-in functions, such as **print ()** to print objects.

The latest Python 3.6 version has 68 built-in functions. A brief description of examples is shown in Table 1 [7].

Table 1. Example of built-in functions [7]

Function	Description
Python abs ()	Return absolute value of a number
Python all ()	Return true when all elements in iterable are true
Python bin ()	Convert integer to binary string
Python dict ()	Create a dictionary
Python len ()	Return a length of an object
Python list ()	Generate a list in Python
Python oct ()	Convert integer to octal

2. **Python User-defined Functions**: The user can define these functions and is referred to as user-defined functions [7]. The functions standardized in Python are known as built-in functions. It tends to be characterized as a library function when the function used is written via other formats in the library [7]. Some advantages of the user-defined functions are as follows:

- A large program can be decomposed into small parts which produce a program that is simple to maintain, understand, and correct.
- This can be used to contain the code if the code is reduplicated in the program and the program is run via call functions if necessary.
- The programmer running on a large work can split the project by making various functions.

3. CREATING A NEW VIRTUAL ENVIRONMENT

Python applications will frequently utilize modules and packages which are added to the standard libraries. The specific structure of the library is necessary for some applications. These applications may need some modification or adaptation for the library interface [8]. This implies that it may not be feasible for one Python establishment to meet the necessities of each application.

If application **A** necessities adaptation 1.0 of a specific module but application **B** needs form 2.0, the prerequisites are in struggle and introducing either form 1.0 or 2.0 will abandon one application unfit to run [8]. Thus, a **virtual environment**, an independent catalog tree that contains a Python establishment for a specific Python variant, should be made in addition to some extra bundles [8].

Diverse applications would then be able to utilize distinctive virtual environments. To determine the prior case of clashing necessities, application **A** can have its virtual condition with adaptation 1.0 introduced while application **B** has another virtual condition with form 2.0. If application **B** requires that a library be moved up to adaptation 3.0, this will not influence application **A**'s environment [8]. A module utilized to make and oversee virtual environments is called **venv**. The **venv** will more often than not introduce the latest form of Python accessible. If different variants of Python are available on a framework, the particular Python form can be chosen via working Python 3 or other needed renditions [8]. Python versions are selected to make the virtual environment. In this paper, these instructions assume that a custom version of Python 3 has already been installed [9].

After it is installed and is using this version, pip3 is run to install **virtualenv**:

```
[server]$ pip3 install virtualenv
Collecting virtualenv
  Downloading virtualenv-15.1.0-py2.py3-none-any.whl (1.8MB)
    100% |████████████████████████████████| 1.8MB 367kB/s
Installing collected packages: virtualenv
Successfully installed virtualenv-15.1.0
```

FIGURE 1. **Running pip3 to install virtualenv [9]**

The full path of a Python library (virtualenv) is then used as shown in Figure 2.

```
[server]$ which virtualenv
/home/username/opt/python-3.6.2/bin/virtualenv
```

FIGURE 2. **Selecting the path [9]**

Using a traditional version of the Python language instead of the server version is common when running a virtual environment. To generate the new virtual environment, the traditional installed version is utilized in the following steps [9]:

1. Make a note of the full file path to the custom version of Python that was just installed, the full path is:

```
[server]$ which python3
/home/username/opt/python-3.6.2/bin/python
```

FIGURE 3. **The full path of Python 3.6.2 [9]**

2. Assign the site directory when the new virtual environment has been created:

```
[server]$ cd ~/example.com
```

FIGURE 4. **Site directory example [9]**

3. Choose the assigned Python version which is needed to be utilized, then start to generate a virtual environment. The command generates the virtualenv called 'my_project' and also utilizes the '-p' flag to assign the path in Python as follows:

```
[server]$ virtualenv -p /home/example_username/opt/python-3.6.2/bin/python3 my_project

Running virtualenv with interpreter /home/example_username/opt/python-3.6.2/bin/python3
Using base prefix '/home/example_username/opt/python-3.6.2'
New python executable in /home/example_username/example.com/env/bin/python3
Also creating executable in /home/example_username/example.com/env/bin/python
Installing setuptools, pip, wheel...done.
```

FIGURE 5. **Creating the virtual environment [9]**

4. To activate the new virtual environment, run the following:

```
[server]$ source my_project/bin/activate
```

FIGURE 6. **Activating the new virtual environment [9]**

5. To deactivate the virtual environment when finished working, run the command “deactivate ”
6. Deleting the specific project folder as shown in Figure 7 is needed to delete the virtual environment.

```
[server]$ rm -rf my_project
```

FIGURE 7. **Delete a virtual environment [9]**

4. EXPERIMENTAL WORK

In this section, the experimental work which used the advantages of the virtual environment library (virtualenv) in Python programming can be explained. First, Anaconda, which is a distribution for both R and Python programming languages, has been used [10–12]. This distribution aims to make easier the deployment and management of the packages. Anaconda has been characterized by including more than 250 packages created automatically together with a virtual environment manager [13, 14]. Figure 8 illustrates the process of virtual environment management in stages [15, 16].

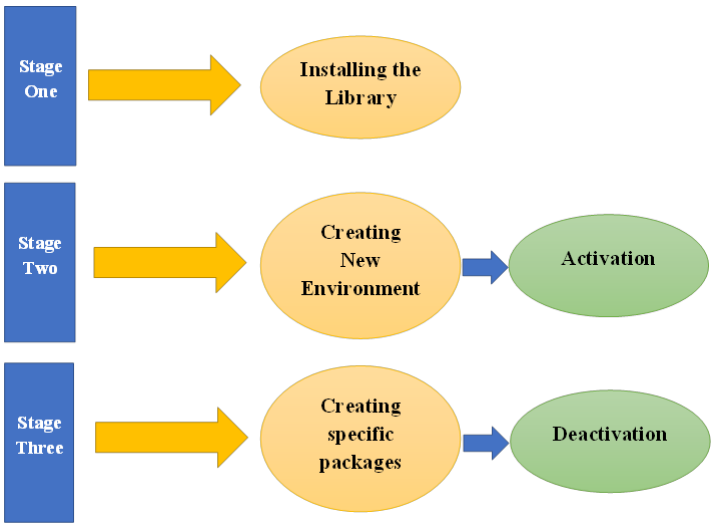

FIGURE 8. **Process of the virtual environment management**

The first stage displays how to install the virtual environment library which is called (virtualenv) using Anaconda Prompt and pip3 install. Figure 9 shows the last version of this library that was installed.

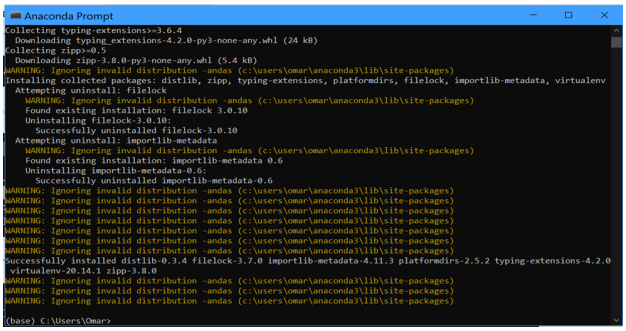

FIGURE 9. Installing the virtual environment library

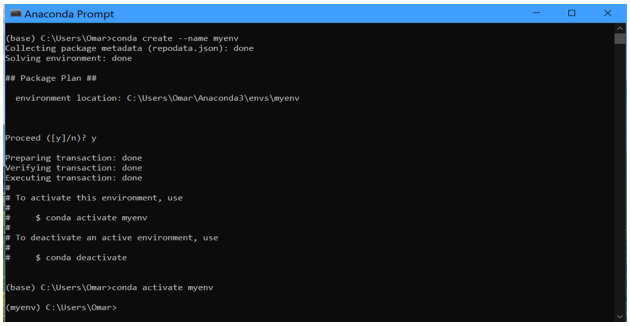

FIGURE 10. Creating new virtual environment with activation

The second stage demonstrates how to create an environment and activate it by using conda in Anaconda Prompt, and the new environment must be activated to use it. Furthermore, the environment must be deactivated after using it. Figure 10 shows how to create the environment.

The third stage shows how to create an environment with a specific version of Python and create an environment with a specific package or even with a specific version of a package along with the environment deactivation. Figure 11 presents the process at this stage.

Finally, some important actions that are used in the virtual environment were noted. These actions can be classified as sharing, restoring, updating, and removing the environment. To run the project without encountering any problems, the project environment must be shared with some person. The restoring environment offers to easily restore some work in the project by returning to the previous version. In the updating environment, the programmer may want some extra package to run the project or need a specified number of the version. Also, removing any unnecessary environment is easy.

5. DIFFERENT EXAMPLES USING THE PYTHON VIRTUAL ENVIRONMENT

In this section, the different examples using the benefits of the virtual environment in Python programming will be discussed.

Example 1 in reference [17] showed that the author used a virtual environment in Python to create a simple virtual world using the Python library. The virtual world is updated in real-time based on measurements from the Arduino and the ultrasonic sensor (HC-SR04). This project shows how to combine Arduino, Python, and Python libraries to create a simple virtual world that is dynamically updated based on sensor measurements. Figures 12 and 13 show the screenshots that display the results of this project.

Also, the author can change the view of the shape. In this example, the author changes the cylinder view as shown in Figure 14.

Example 2 in [18] showed that the authors used the virtual Python environment to build shapes, animations, and interactions (Figures 15 and 16).

Example 3 in [19] showed that the authors used Python language via different functions to create and manipulate the 3D objects in the games as shown in Figure 17.

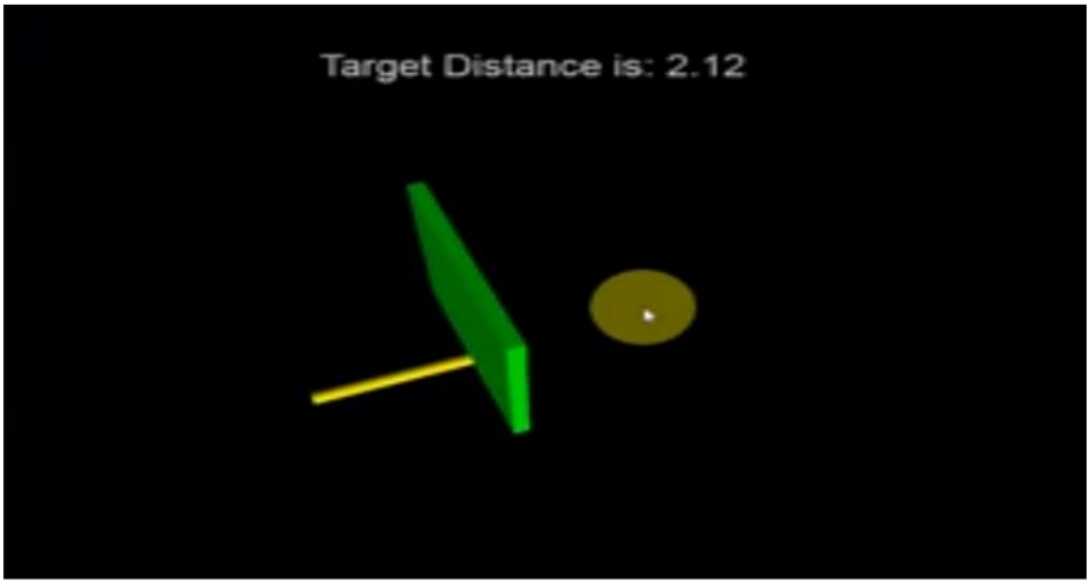

FIGURE 11. **Creating specific packages with deactivation**

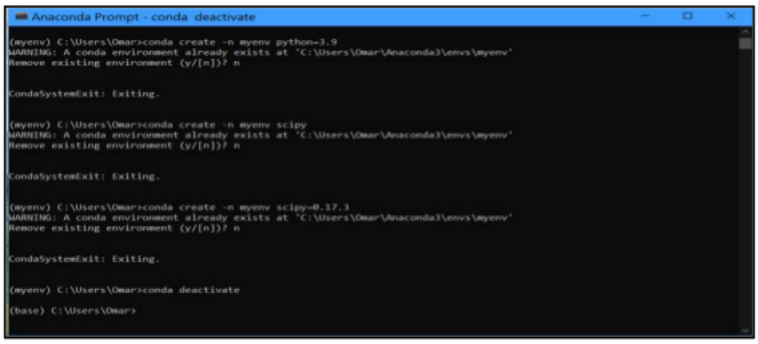

FIGURE 12. **Example 1: Python with ultrasonic sensor [17]**

FIGURE 13. **Example 1: Python with ultrasonic sensor and the cylinder as results [17]**

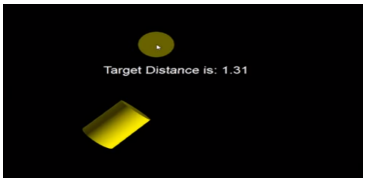

FIGURE 14. **Example 1: Change the view of the shape [17]**

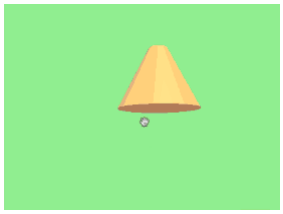

FIGURE 15. Example 2: Display shape via the virtual environment [18]

FIGURE 16. Example 2: Display interactions in a browser [18]

FIGURE 17. Example 3: Create and manipulate the 3D game [19]

6. CONCLUSION

Python is a programming language that can be used for a general purpose on different types of operating systems. It supports working with images, text, values, and anything saved on a computer. In this paper, the most popular function in the Python language, called Virtual Environment, has been presented. The steps which are used to create this function have also been explained. Various examples illustrating the importance and effective role of a Virtual Environment function in the Python language were then presented.

ACKNOWLEDGEMENT

The first author would like to thank the reviewers for providing useful suggestions, allowing for the improved presentation of this paper.

CONFLICTS OF INTEREST

The authors declare no conflict of interest.

REFERENCES

[1] M. Herman, *Python Virtual Environments: A Primer", Real Python Company, Online Access*. 2022.
[2] M. Agrawal, "Python Virtual Environment", GeeksforGeeks Company, Online Access," 2022.
[3] I.-N. Liao, "Tutorial - Python Virtual Environment", A Medium Corporation," 2018.
[4] O. Terpollari, "Virtual Environments in Python Made Easy", Sitepoint Company," 2015.
[5] H. P. Langtangen and A. Johansen, "Creating Virtual Python Software Environments with Virtualenv," 2016.
[6] B. V. Roy, "Python Tutorial," 2016.
[7] "Programiz Company, "python-programming/function", Parewa Labs Pvt. Ltd, Online Access," 2022.
[8] *Python Software Foundation, "Virtual Environments and Packages", Python Software Foundation Corporation, Online Access*. 2022.
[9] D. Company, "Installing and using virtualenv with Python 3", Online Access," 2022.
[10] Wikipedia and Anaconda, "Python distribution)", Online Access," 2022.
[11] P. Wang and T. Oliphant, "About Anaconda, Anaconda, Inc," 2020.
[12] I. Anaconda, "Products and Pricing", anaconda.com," 2020.

[13] I. Anaconda, “Conda - Conda documentation,” 2016.
[14] B. Team, “What’s the difference between Anaconda, conda, and Miniconda?,” 2020.
[15] I. Anaconda, . . Miniconda, and C. Io, 2018.
[16] I. Anaconda, “Anaconda | Understanding Conda and Pip,” 2021.
[17] P. Mcwhorter, “Python with Arduino: simple virtual world using ultrasonic sensor,” 2022.
[18] “Introduction to Virtual Reality”, Online Access, CodeHS Company,” 2022.
[19] “Virtual Reality on Raspberry Pi With BeYourHero”, Instructables Company,” *JeanotP*, 2019.