

This specific module describes the skills, knowledge and attitude required to Apply Python Programming. This module is intended to prepare learners pursuing TVET Level 5 in Networking and Internet Technologies. Upon completion of this module, the learner will be able to Prepare python environment, write basic python program and Apply object-driven in python

At the end of the module the learner will be able to:

1. Prepare python environment

2. Write basic python program

3. Apply object-driven in python

# 1. Prepare python environment

## 1.1 Python tools are properly selected in accordance with computer operating system

### Python programming overview

Computers and programs are everywhere in today's world. Programs affect many aspects of daily life and society. People depend on programs for communication, shopping, entertainment, health care, and countless other needs. Learning how to program computers opens the door to many careers and opportunities for building a better world.

What is Python?

Python is a popular programming language. It was created by **Guido van Rossum**, and released in 1991.

It is used for:

* web development (server-side),
* software development,
* mathematics,
* system scripting.

What can Python do?

* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
* Python has a simple syntax like the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-oriented way or a functional way.

Python Syntax compared to other programming languages

* Python was designed for readability and has some similarities to the English language with influence from mathematics.
* Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
* Python relies on **indentation**, using **whitespace**, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

**Python** is an i*nterpreted*, *object-oriented*, *high-level programming language* with *dynamic semantics*. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. (Python\_Community)

**Programs** consist of statements to be run one after the other. A **statement** describes some action to be carried out.

The statement **print("Good morning")** instructs Python to output the message **"Good morning"** to the user. The statement **count = 0** instructs Python **to assign the integer 0** to the variable **count**.

A **computer** is an electronic device that **stores** and **processes** information. Examples of computers include smartphones, tablets, laptops, desktops, and servers. Technically, a **program** is a *sequence of instructions that a computer can run*. Programs help people accomplish everyday tasks, create new technology, and have fun.

* **What are the characteristics of Python?**

Python has many important features for programmers, both for those who are starting out as well as for those who already have knowledge and want to try new things or experiment. Some of its main features include:

* **Object oriented**

One of the main characteristics of Python is that it is an object-oriented programming language. This means that Python recognizes the concept of class and object encapsulation, which makes coding with Python more efficient in the long run.

As such, Python makes it easy to create inherited object classes. This means that, building from things that have already been done, you can create new classes that will inherit the attributes of the previous ones, which simplifies and improves the long-term efficiency of the code.

* **Open Source**

Yet another of the main characteristics of Python is that it is an open source programming language. Anyone can create and contribute to its development.

This in turn means that it has a large community that works to improve and facilitate the learning of this programming system. Also, it is free to download for any operating system, including Windows, Mac or Linux.

* **Easy to learn**

Python is a very user-friendly code for all types of developers, from those who already have experience with other languages to those who are learning to program from scratch.

If you already have experience with C, C++, Java or C#, Python is a good system with which to continue growing and expanding your programming knowledge. If you’re just starting out, with the right training, it’s easy to jump in and learn how to do things in a very short time.

* **Integration and adaptation**

Another of the main characteristics of Python is that it is an integrated programming language. This means that it executes the code line by line.

What does this imply? It means that Python, unlike other coding languages, does not compile, which makes the process of debugging code much easier and more efficient. Another advantage of this characteristic of Python is that it makes execution easier and saves time in the long run.

* **GUI support**

GUI stands out for Graphical User Interface, which is a key aspect of any programming language as it helps add style to the code and makes the programmer’s work much more visible.

In this sense, Python is compatible with a wide range of GUIs, which can be easily imported, making it a widely used system in Data Science, as it facilitates data visualization.

* **High level programming**

As we said at the beginning, one of the main characteristics of Python is that it has been designed to become a high-level programming language.

That means that, when working with it, you don’t need to know the code structure, architecture, or memory management. This simplifies the work of programmers.

* **Portable**

Let’s see this feature through a practical case: suppose you are working with Python on Windows, but you need to transfer your work to Mac or Linux. With Python you can do this without having to make any changes in the code.

This portability between different operating systems is something that not all programming languages have, which is why Python has become one of the most portable systems today, a great advantage in its main fields of application.

* **Applications of python**

Python is a versatile programming language with many applications, including **software development, data science, automation,** and **data analytics:**

* **Software development**

Python can help with tasks like build control, bug tracking, and testing. For example, developers can use Python to automate testing for new products or features. Python is also used in web development, particularly for fast web application development.

* **Data science**

Python is a good programming language for data science because of its readability and libraries like Pandas and NumPy. These libraries provide tools for data manipulation, statistics, and visualization, which can help with tasks like data visualization, manipulation, and machine learning. Python is also used in data analytics, data engineering, and image processing.

* **Automation**

Python can be used for automation, such as web scraping, which can pull in large amounts of data from websites for real-world processes like job listings, price comparison, development, and research.

* **Data analytics**

Python can be used for data analytics with modules like Scikit-learn, which is an open-source tool that can implement algorithms like classification, regression, and clustering.

* **Identifications of python tools**

There are various python tools, including **Python Development Tools**, Python Web Scraping Tools, **Python Web Development**, Tools **Python Data Analysis Tools**, **Python Data Visualization** Tools, and Python Machine Learning Tools

1. **Python Development Tools**

Development tools help us to build fast and reliable Python solutions. It includes Integrated Development Environment (IDE), Python package manager, and productive extensions. These tools have made it easy to test the software, debug, and deploy solutions in production.

* **Pip**

**Pip** is a tool that uses **Python Package Index** to install and manage Python software. There are 393,343 projects for you to download and install with lightning speed. The Python ecosystem works on it.

The Python Package Index (**PyPI**) is a **repository** of software for the Python programming language. **PyPI** helps you find and install software developed and shared by the Python community. Package authors use PyPI to distribute their software. (Python\_comm.)

pip install <package\_name>

This code installs a Python package using the pip package manager. It should be replaced with the name of the package that the user wants to install. The **pip install** command will download and install the specified package and any dependencies it requires.

* **VSCode**

Visual Studio Code is free, lightweight, and a powerful code editor. You can build, test, deploy, and maintain all types of applications without leaving the software window.

1. **Python Data Visualization Tools**

Data visualization gives life to data analysis. If you want to explain things to non-technical executives, you need to tell a data story by displaying a bar chart, line plot, scatter plot, heat maps, and histograms. The visualization tools help data analytics create interactive, colorful, and clean visualization with few lines of code.

* **Matplotlib**

Matplotlib is a gateway to the world of data visualization. You will learn about it in many data visualization introductions.

With Matplotlib, you can create fully customizable static, animated, and interactive visualizations. It’s intuitive, and you can use it to plot 3D, multilevel, and detailed visualization. There are hundreds of examples of different visualizations available in the gallery.

1. **Python Data Analysis Tools**

Data analysis tools allow users to ingest, clean, and manipulate data for statistical analysis. Every data professional must understand the core functionality of these tools to perform data analysis, machine learning, data engineering, and business intelligence tasks.

* **Pandas**

Pandas is a gateway into the world of data science. The first thing you learn as a beginner is to load a CSV file using **read\_csv().** pandas is an essential tool for all data professionals.

You can load a dataset, clean it, manipulate it, calculate statistics, create visualizations, and save the data into various file formats. The pandas API is simple and intuitive. You can load and save CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 file format.

* **Numpy**

Numpy is a fundamental Python package for scientific computations, and most modern tools are built upon it. As a data scientist, you will use the Numpy array for mathematical calculations and data wrangling. It provides multidimensional array objects to perform fast operations such as logical, shape manipulation, sorting, selection, basic statics operation, and random simulation.

Numpy will help you understand the fundamentals of mathematics in data science and how to convert complex equations into Python code. You can use it to create machine learning models, customized statical formulas, scientific simulations, and perform advanced data analytics tasks.

## 1.2 Python tools are properly installed in accordance with Operating System

* **Identification of computer system requirements**

Python's computer system requirements include hardware and software:

1. **Operating system**

Linux Ubuntu 16.04 to 17.10, or Windows 7 to 10. Slides for Python via .NET also supports macOS and Linux 64-bit operating systems.

1. **RAM**

2 GB of RAM, but 4 GB is preferable. Enthought Python requires 4 GB of RAM.

1. **Disk space**

5 GB of free disk space.

1. **CPU**

An x86 64-bit CPU (Intel or AMD architecture). TIBCO Documentation recommends 4 cores (Intel Core i3 or equivalent) at a minimum, and 8 cores or more (Intel Xeon E5 or equivalent) at a recommendation.

1. **Software**

Python 3.11 and related packages. Python-specific editors and IDEs, such as PyCharm, Spyder, and Thonny, are also available.

A screenshot of a computer

Description automatically generated

Figure 1: Python versions available today

* Installation of python software tools
* **Prerequisites**

You’ll need a computer with the above specifications, having administrative privileges and an internet connection.

Step 1 — Downloading the Python Installer

1. Go to the official [Python download page for Windows](https://www.python.org/downloads/windows/).
2. Find a stable Python 3 release.
3. Click the appropriate link for your system to download the executable file: **Windows installer (64-bit)** or **Windows installer (32-bit)**.



Figure 2:Choosing to download pyton

Step 2 — Running the Executable Installer

1. After the installer is downloaded, double-click the .exe file, for example python-3.10.10-amd64.exe, to run the Python installer.
2. Select the **Install launcher for all users** checkbox, which enables all users of the computer to access the Python launcher application.
3. Select the **Add python.exe to PATH** checkbox, which enables users to launch Python from the command line.
4. If you’re just getting started with Python and you want to install it with default features as described in the dialog, then click **Install Now** and go to [Step 4 - Verify the Python Installation](https://www.digitalocean.com/community/tutorials/install-python-windows-10#step-4-verify-the-python-installation). To install other optional and advanced features, click **Customize installation** and continue.
5. The **Optional Features** include common tools and resources for Python and you can install all of them, even if you don’t plan to use them.

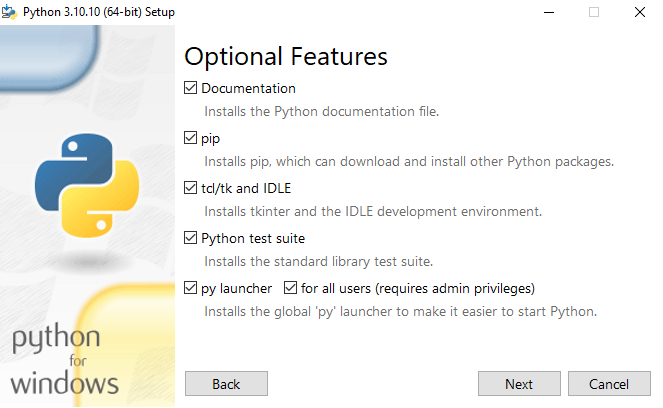


Figure 3 Operational python features

Select some or all the following options:

* 1. **Documentation**: recommended
  2. **pip**: recommended if you want to install other Python packages, such as NumPy or pandas
  3. **tcl/tk and IDLE**: recommended if you plan to use IDLE or follow tutorials that use it
  4. **Python test suite**: recommended for testing and learning
  5. **py launcher** and **for all users**: recommended to enable users to launch Python from the command line

1. Click **Next**.
2. The **Advanced Options** dialog is displayed.

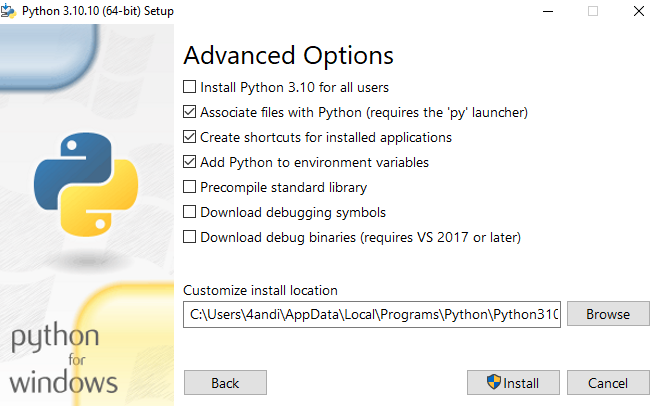


Figure 4: Python advanced options

Select the options that suit your requirements:

* 1. **Install for all users**: recommended if you’re not the only user on this computer
  2. **Associate files with Python**: recommended, because this option associates all the Python file types with the launcher or editor
  3. **Create shortcuts for installed applications**: recommended to enable shortcuts for Python applications
  4. **Add Python to environment variables**: recommended to enable launching Python
  5. **Precompile standard library**: not required, it might down the installation
  6. **Download debugging symbols** and **Download debug binaries**: recommended only if you plan to create C or C++ extensions

Make note of the Python installation directory in case you need to reference it later.

1. Click **Install** to start the installation.
2. After the installation is complete, a **Setup was successful** message displays.

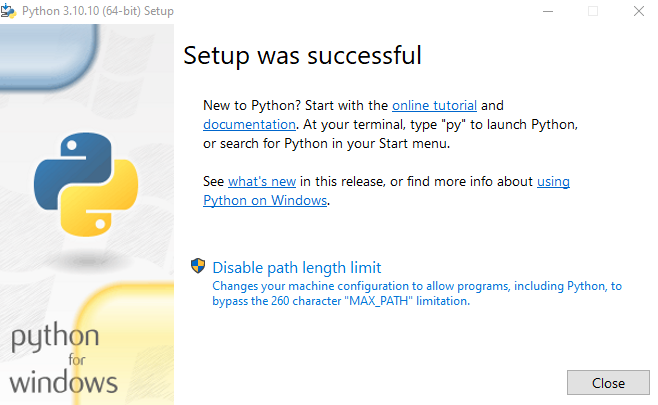


Figure 5:Successful installation

**Step 3 — Adding Python to the Environment Variables (optional)**

Skip this step if you selected **Add Python to environment variables** during installation.

If you want to access Python through the command line but you didn’t add Python to your environment variables during installation, then you can still do it manually.

Before you start, locate the Python installation directory on your system. The following directories are examples of the default directory paths:

* C:\Program Files\Python310: if you selected **Install for all users** during installation, then the directory will be system wide
* C:\Users\Sammy\AppData\Local\Programs\Python\Python310: if you didn’t select **Install for all users** during installation, then the directory will be in the Windows user path

Note that the folder name will be different if you installed a different version but will still start with Python.

1. Go to **Start** and enter advanced system settings in the search bar.
2. Click **View advanced system settings**.
3. In the **System Properties** dialog, click the **Advanced** tab and then click **Environment Variables**.
4. Depending on your installation:
   * If you selected **Install for all users** during installation, select **Path** from the list of **System Variables** and click **Edit**.
   * If you didn’t select **Install for all users** during installation, select **Path** from the list of **User Variables** and click **Edit**.
5. Click **New** and enter the Python directory path, then click **OK** until all the dialogs are closed.

**Step 4 — Verify the Python Installation**

You can verify whether the Python installation is successful either through the command line or through the Integrated Development Environment (IDLE) application, if you chose to install it.

Go to **Start** and enter cmd in the search bar. Click **Command Prompt**.

Enter the following command in the command prompt:

**python --version**

An example of the output is:

**Python 3.12.10**

* Configure python virtual environment

**Virtual Environment**

It is suggested to have a dedicated virtual environment for each Django project, and one way to manage a virtual environment is [venv](https://docs.python.org/3/tutorial/venv.html" \t "_blank), which is included in Python.

The name of the virtual environment is your choice, in this tutorial we will call it level5NIT.

Type the following in the command prompt, remember to navigate to where you want to create your project:

**Windows:**

py -m venv myworld

This will set up a virtual environment, and create a folder named " level5NIT " with subfolders and files, like this:

level5NIT  
  Include  
  Lib  
  Scripts  
  pyvenv.cfg

Then you must activate the environment, by typing this command:

level5NIT\Scripts\activate.bat

Once the environment is activated, you will see this result in the command prompt:

(level5NIT) C:\Users\*Your Name*>

**Note:** You must activate the virtual environment every time you open the command prompt to work on your project.

**Why virtual environment?**

A virtual environment is a tool that helps to keep dependencies required by different projects in separate places, by creating isolated environments for each project. Here’s why it's important to use a dedicated virtual environment, like the one described for a Django project:

**1. Dependency Isolation:**

* **Problem**: Different projects may require different versions of the same package or library. If all projects use the global Python environment, there could be conflicts when one project needs version 1.0 of a library and another needs version 2.0.
* **Solution**: A virtual environment allows you to install specific versions of packages for each project without affecting other projects. This ensures that your Django project has a consistent environment that won’t break due to external changes.

**2. Avoiding Conflicts:**

* **Problem**: Installing packages globally might cause conflicts between different projects or with the system's Python packages, leading to unpredictable behavior.
* **Solution**: By using a virtual environment, each project has its own set of dependencies, minimizing the risk of conflicts.

**3. Easier Management:**

* **Problem**: Managing dependencies across multiple projects can become complex, especially when you need to ensure that a project works consistently on different machines or environments.
* **Solution**: A virtual environment encapsulates all dependencies, making it easier to manage and replicate the environment across different setups. This also makes it easier to work in teams or deploy projects.

**4. Clean Global Python Environment:**

* **Problem**: Installing all packages globally can clutter your global Python environment, making it harder to track which packages are used for which project.
* **Solution**: Virtual environments keep the global Python environment clean by only installing packages locally within the project directory.

**5. Reproducibility:**

* **Problem**: Without a virtual environment, it’s challenging to ensure that your project will work the same way on another machine, especially if it relies on specific package versions.
* **Solution**: A virtual environment allows you to create a requirements.txt file that lists all the dependencies with their versions. This file can be used to recreate the same environment on another machine, ensuring that the project behaves consistently.

**6. Security:**

* **Problem**: Installing packages globally can expose your system to potential vulnerabilities if a malicious package is installed.
* **Solution**: A virtual environment contains the installation within a controlled directory, reducing the risk to your system.

**Using venv with Django:**

In the example you provided, the command py -m venv myworld creates a virtual environment named "level5NIT". By activating this environment, any packages you install **(like Django)** will only apply to this project, keeping it isolated from other Python projects and maintaining a clean, conflict-free environment.

Matplotlib, NumPy, and Pandas are three essential libraries in Python, particularly for data analysis, visualization, and scientific computing. Here's a brief overview of each:

**1. Matplotlib:**

* **Purpose**: Matplotlib is a plotting library used for creating static, animated, and interactive visualizations in Python. It's highly versatile and allows you to create a wide range of plots, such as line graphs, bar charts, histograms, scatter plots, and more.
* **Common Use Cases**:
  + Visualizing data distributions and trends.
  + Creating complex plots like 3D graphs or multi-subplot figures.
  + Generating publication-quality figures in a variety of formats.
* **Installation**: pip install matplotlib

**2. NumPy:**

* **Purpose**: NumPy is a library for numerical computations in Python, providing support for large, multi-dimensional arrays and matrices. It also includes a vast collection of mathematical functions to operate on these arrays efficiently.
* **Common Use Cases**:
  + Performing mathematical operations on arrays and matrices (e.g., addition, multiplication, trigonometric functions).
  + Working with large datasets, where performance is critical.
  + Serving as the foundation for other data science libraries like Pandas and machine learning frameworks.
* **Installation**: pip install numpy

**3. Pandas:**

* **Purpose**: Pandas is a powerful data manipulation and analysis library that provides data structures like DataFrames and Series. It allows for easy manipulation, analysis, and visualization of structured data.
* **Common Use Cases**:
  + Handling and analyzing large datasets, such as CSV files or SQL databases.
  + Cleaning, transforming, and filtering data.
  + Performing time series analysis, merging/joining data, and calculating statistics.
* **Installation**: pip install pandas

**How They Work Together:**

These three libraries often work together in data science and analysis workflows:

* **NumPy** is used for numerical operations and array manipulation.
* **Pandas** is built on top of NumPy and provides more sophisticated data manipulation tools, particularly for tabular data.
* **Matplotlib** uses NumPy and Pandas to plot and visualize data, turning numerical results into understandable graphics.

Here's a simple example of how you might use all three in a single script:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Create a NumPy array

data = np.random.randn(100)

# Create a Pandas DataFrame

df = pd.DataFrame(data, columns=['Random Data'])

# Plot the data using Matplotlib

df.plot(kind='hist', title='Histogram of Random Data')

plt.show()

## 1.3 Python installation is successfully tested based on output of python version command

# 2. Write basic python program

## 2.1 Python concepts are effectively applied based on python standards

## 2.2 Control structures are properly applied in accordance with python standards

## 2.3 Functions are properly applied in accordance with python standards

## 2.4 Collections are properly applied in accordance with python standards

## 2.5 File handling is properly performed in accordance with python standards

# 3. Apply object-driven in python

## 3.1 OOP Concepts are properly applied in line with python standards

## 3.2 Python date and time concepts are applied according to python standards

## 3.3 Libraries are properly used in accordance with python standards

## 3.4 System automation is properly applied based on specific task