Python Essentials 2:
Module 1
Modules, Packages and PIP

In this module, you will learn about:

- importing and using Python modules;
- using some of the most useful Python standard library modules;
- constructing and using Python packages;

PIP (Python Installation Package) and how to use it to install and uninstall ready-to-use packages from PyPI.

Python packaging ecosystem and how to use it

Python is a very powerful instrument – we hope you've experienced this yourself already. Many people from around the world feel this way, and they use Python on a regular basis to develop what they can do in many completely different fields of activity. This means that Python has become an interdisciplinary tool employed in countless applications. We can't go through all the spheres in which Python brilliantly shows off its abilities, so let us just tell you about the most impressive ones.

First of all, Python has turned into a leader of research on artificial intelligence. Data mining, one of the most promising modern scientific disciplines, utilizes Python as well. Mathematicians, psychologists, geneticists, meteorologists, linguists – all these people already use Python, or if they don't already, we're sure that they will very soon. There is no escaping this trend.

Of course, it doesn't make any sense to get all Python users to write their code from scratch, keeping them perfectly isolated from the outside world and from other programmers' achievements. This would be both unnatural and counterproductive.

The most preferable and efficient thing is to enable all Python community members to freely exchange their codes and experiences. In this model, nobody is forced to start work from scratch, as there's a high probability that someone else has been working on the same (or a very similar) problem.

As you know, Python was created as open-source software, and this also works as an invitation for all coders to maintain the whole Python ecosystem as an open, friendly, and free environment. To make the model work and evolve, some additional tools should be provided, tools that help the creators to publish, maintain, and take care of their code.

These same tools should help users to make use of the code, both the already existing code, and also the new code appearing every day. Thanks to that, writing new code for new challenges is not like building a new house, starting at the foundations.

Moreover, the programmer is free to modify someone else's code in order to adapt it to their own needs, and in effect build a completely new product that can be used by another developer. The process seems to have no end. Fortunately.

To make this world go round, two basic entities have to be established and kept in motion: a centralized repository of all available software packages; and a tool allowing users to access the repository. Both these entities already exist and can be used at any time.

Python packaging ecosystem and how to use it: continued

The repository (or repo for short) we mentioned before is named PyPI (it's short for Python Package Index) and it's maintained by a workgroup named the Packaging Working Group, a part of the Python Software Foundation, whose main task is to support Python developers in efficient code dissemination.

You can find their website here:
https://wiki.python.org/psf/PackagingWG

The PyPI website (administered by PWG) is located at the address:
https://pypi.org/

When we popped in there for a while at the beginning of June 2020, we found that PyPI was home to 237,515 projects, consisting of 2,877,545 files managed by 427,487 users.

These three numbers alone clearly show the potency of the Python community and the importance of developer cooperation.

We must point out that PyPI is not the only existing Python repository. On the contrary, there are lots of them, created for projects and led by many larger and smaller Python communities. It's likely that someday you and your colleagues may want to create your own repos.

Anyway, PyPI is the most important Python repo in the world. If we modify the classic saying a little, we can state that “all Python roads lead to PyPI”, and that’s no exaggeration at all.

The PyPI repo: the Cheese Shop

The PyPI repo is sometimes referred to as the Cheese Shop. Really.

Does that sound a little strange to you? Don't worry, it’s all perfectly innocent.

We refer to the repo as a shop, because you go there for the same reasons you go to other shops: to fulfill your needs. If you want some cheese, you go to the cheese shop. If you want a piece of software, you go to the software shop. Fortunately, the analogy ends here – you don't need any money to take some software out of the repo shop.

PyPI is completely free, and you can just pick a code and use it – you’ll encounter neither cashier nor security guard. Of course, it doesn't absolve you from being polite and honest. You have to obey all the licensing terms, so don't forget to read them.

“OK”, you say, “the shop is clear now, but what does cheese have to do with Python?”

The Cheese Shop is one of the most famous Monty Python sketches. It depicts the surrealist adventure of an Englishman trying to buy some cheese. Unfortunately, the shop he visits (immodestly named Ye National Cheese Emporium) has no cheese in stock at all.

Of course, it's meant to be ironic. As you already know, PyPI has lots of software in stock and it's available 24/7. It's fully entitled to identify itself as Ye International Python Software Emporium.

The PyPI repo: the Cheese Shop (continued)

PyPI is a very specific shop, not just because it offers all its products for free. It also requires a special tool to make use of it.

Fortunately, this tool is also free, so if you want to make your own digital cheeseburger by using the goods offered by the PyPI Shop, you'll need a free tool named pip.

No, you haven't misheard. Just pip. It's another acronym, of course, but its nature is more complex than the previously mentioned PyPI, as it's an example of a recursive acronym, which means that the acronym refers to itself, which means that explaining it is an infinite process.

Why? Because pip means "pip installs packages", and the pip inside "pip installs packages" means "pip installs packages" and ...

Let's stop there. Thank you for your cooperation.

By the way, there are a few other very famous recursive acronyms. One of them is Linux, which can be interpreted as "Linux is Not Unix".

How to install pip

The question that should be put now is: how to get a proper cheese knife? In other words, how to ensure that pip is installed and ready to work?

The most precise answer is “it depends”. Really.

Some Python installations come with pip, some don't. What’s more, it doesn’t only depend on the OS you use, although this is a very important factor.

Let's start with MS Windows.

pip on MS Windows

The MS Windows Python installer already contains pip, and so no other steps need to be taken in order to install it. Unfortunately, if the PATH variable is misconfigured, pip may unavailable.

To verify that we haven’t misled you, try to do this:

- open the Windows console (CMD or PowerShell, whatever you prefer)
- execute the following command:

  pip –version

- the absence of this message may mean that the PATH variable either incorrectly points to the location of the Python binaries, or doesn't point to it at all; for example, our PATH variable contains the following substring:

  C:\Program Files\Python3\Scripts\;C:\Program Files\Python3\;

- the easiest way to reconfigure the PATH variable is to reinstall Python, instructing the installer to set it for you.

pip on Linux

Different Linux distributions may behave differently when it comes to using pip. Some of them (like Gentoo), which are closely bound to Python and which use it internally, may have pip preinstalled and are instantly ready to work.

Don't forget that some Linuces may utilize more than one Python version concurrently, e.g., one Python 2 and one Python 3 coexisting side by side. Such systems may launch Python 2 as the default version, and it may be necessary to explicitly specify the program name as python3. In this case there may be two different pips identified as pip (or pip2) and pip3. Check it carefully.

Open the terminal window and issue the following command:

pip –version

An answer similar to the one shown in the previous picture determines that you've launched pip from Python 2, so the next try should look as follows:

pip3 –version

pip on Linux: continued

Unfortunately, some Linux distributions don't have pip preinstalled, even if Python itself is installed by default (some versions of Ubuntu may behave this way). In this case, you have two possibilities:

- install pip as a system package using a dedicated package manager (e.g., apt in Debian-like systems)
- install pip using internal Python mechanisms.

The former is definitely better. Although there are some smart scripts that are able to download and install pip by ignoring the OS, we discourage you from using them. This method can get you into trouble.

That's good advice, and we're going to follow it, but it has to be stated that we'll need administrative rights to do it. Don't forget that different Linuces may use different package managers (e.g., it could be pacman if you use Arch Linux, or yum used by distributions derived from Red Hat).

Anyway, all these methods should get pip (or pip3) installed and working.

As you can see, the OS decided to install not only pip itself, but also a couple of additional components needed by pip. This is normal – don't be alarmed.

pip on Linux: continued

When apt finishes its job, we are finally able to utilize pip3:

If you're a Mac user and you've installed Python 3 using the brew installer, pip is already present in your system and ready to work. Check it by issuing the previously mentioned command:

pip3 –version

Dependencies

Now that we're sure that pip is ready at our command, we're going to limit our focus to MS Windows only, as its behavior is (should be) the same in all OSes, but before we start, we need to explain an important issue and tell you about dependencies.

Imagine that you've created a brilliant Python application named redsuspenders, able to predict stock exchange rates with 99% accuracy (by the way, if you actually do that, please contact us immediately).

Of course, you've used some existing code to achieve this goal – e.g., your app imports a package named nyse containing some crucial functions and classes. Moreover, the nyse package imports another package named wallstreet, while the wallstreet package imports other two essential packages named bull and bear.

As you've probably already guessed, the connections between these packages are crucial, and if somebody decides to use your code (but remember, we've already called dibs on it) they will also have to ensure that all required packages are in place.

To make a long story short, we can say that dependency is a phenomenon that appears every time you're going to use a piece of software that relies on other software. Note that dependency may include (and generally does include) more than one level of software development.

Does this mean that a potential nyse package user is obliged to trace all dependencies and manually install all the needed packages? That would be horrible, wouldn't it?

Yes, it's definitely horrible, so you shouldn't be surprised that the process of arduously fulfilling all the subsequent requirements has its own name, and it's called dependency hell.

How do we deal with that? Is every user doomed to visit hell in order to run the code for the first time?

Fortunately not - pip can do all of this for you. Really. It can discover, identify, and resolve all dependencies. Moreover, it can do it in the cleverest way, avoiding any unnecessary downloads and reinstalls.

How to use pip

Now we're ready to ask pip what it can do for us. Let's do it – issue the following command:

pip help

Don't forget that you may be obliged to replace pip with pip3 if your environment requires this.

The list produced by pip summarizes all the available operations, and the last of them is help, which we've just used already.

If you want to know more about any of the listed operations, you can use the following form of pip invocation:

pip help operation

For example, the line:

pip help install

will show you detailed information about using and parameterizing the install command.

If you want to know what Python packages have been installed so far, you can use the list operation – just like this:

pip list

As you can see, there are two columns in the list, one showing the name of the installed package, and the other showing the version of the package. We can't predict the state of your Python installation.

The only thing we know for sure is that your list contains the two lines we see on our list: pip and setuptools. This happens because the OS is convinced that a user wanting pip will very likely need the setuptools soon. It's not wrong.

How to use pip: continued

The pip list isn't very informative, and it may happen that it won't satisfy your curiosity. Fortunately, there's a command that can tell you more about any of the installed packages (note the word installed). The syntax of the command looks as follows:

pip show package_name

We're going to use it in a slightly deceptive way – we want to convince pip to confess something about itself. This is how we do it:

pip show pip

You may ask where this data comes from? Is pip really so perceptive? Not at all – the information appearing on the screen is taken from inside the package being shown. In other words, the package's creator is obliged to equip it with all the needed data (or to express it more precisely – metadata).

Look at the two lines at the bottom of the output. They show:

which packages are needed to successfully utilize the package (Requires:)
which packages need the package to be successfully utilized (Required-by:)
As you can see, both properties are empty. Feel free to try to use the show command in relation to any other installed package.

The power of pip comes from the fact that it's actually a gateway to the Python software universe. Thanks to that, you can browse and install any of the hundreds of ready-to-use packages gathered in the PyPI repositories. Don't forget that pip is not able to store all PyPI content locally (it's unnecessary and it would be uneconomical).

In effect, pip uses the Internet to query PyPI and to download the required data. This means that you have to have a network connection working whenever you're going to ask pip for anything that may involve direct interactions with the PyPI infrastructure.

One of these cases occurs when you want to search through PyPI in order to find a desired package. This kind of search is initiated by the following command:

pip search anystring

The anystring provided by you will be searched in:

the names of all the packages;
the summary strings of all the packages.
Be aware of the fact that some searches may generate a real avalanche of data, so try to be as specific as possible. For example, an innocent-looking query like this one:

pip search pip

produces more than 100 lines of results (try it yourself – don't take our word for it). By the way – the search is case insensitive.

If you're not a fan of console reading, you can use the alternative way of browsing PyPI content offered by a search engine, available at https://pypi.org/search.

How to use pip: continued

Assuming that your search is successful (or you're determined to install a specific package of an already known name) you can use pip to install the package onto your computer.

Two possible scenarios may be put into action now:

you want to install a new package for you only – it won't be available for any other user (account) existing on your computer; this procedure is the only one available if you can't elevate your permissions and act as a system administrator;
you've decided to install a new package system-wide – you have administrative rights and you're not afraid to use them.
To distinguish between these two actions, pip uses a dedicated option named --user (note the double dash). The presence of this option instructs pip to act locally on behalf of your (non-administrative) user.

If you don't add this, pip assumes that you're as a system administrator and it'll do nothing to correct you if you're not.

In our case, we're going to install a package named pygame – it's an extended and complex library allowing programmers to develop computer games using Python.

The project has been in development since the year 2000, so it's a mature and reliable piece of code. If you want to know more about the project and about the community which leads it, visit https://www.pygame.org.

If you're a system administrator, you can install pygame using the following command:

pip install pygame

If you're not an admin, or you don't want to fatten up your OS by installing pygame system-wide, you can install it for you only:

pip install --user pygame

It's up to you which of the above procedures you want to take place.

Pip has a habit of displaying fancy textual animation indicating the installation progress, so watch the screen carefully – don't miss the show! If the process is successful, you'll see something like this:

We encourage you to use:

pip show pygame

and

pip list

to get more information about what actually happened.

How to use pip: a simple test program
Now that pygame is finally accessible, we can try to use it in a very simple test program. Let's comment on it briefly.

- line 1: import pygame and let it serve us;
- line 3: the program will run as long as the run variable is True;
- lines 4 and 5: determine the window's size;
- line 6: initialize the pygame environment;
- line 7: prepare the application window and set its size;
- line 8: make an object representing the default font of size 48 points;
- line 9: make an object representing a given text – the text will be anti-aliased (True) and white (255,255,255)
- line 10: insert the text into the (currently invisible) screen buffer;
- line 11: flip the screen buffers to make the text visible;
- line 12: the pygame main loop starts here;
- line 13: get a list of all pending pygame events;
- lines 14 through 16: check whether the user has closed the window or clicked somewhere inside it or pressed any key;
- line 15: if yes, stop executing the code.

```
import pygame

run = True
width = 400
height = 100
pygame.init()
screen = pygame.display.set_mode((width, height))
font = pygame.font.SysFont(None, 48)
text = font.render("Welcome to pygame", True, (255, 255, 255))
screen.blit(text, ((width - text.get_width()) // 2, (height - text.get_height()) // 2))
pygame.display.flip()
while run:
   for event in pygame.event.get():
      if event.type == pygame.QUIT\
      or event.type == pygame.MOUSEBUTTONUP\
      or event.type == pygame.KEYUP:
         run = False
```

How to use pip: continued

The pip install has two important additional abilities:

it is able to update a locally installed package – e.g., if you want to make sure that you're using the latest version of a particular package, you can run the following command:

pip install -U package_name

where -U means update. Note: this form of the command makes use of the --user option for the same purpose as presented previously;

it is able to install a user-selected version of a package (pip installs the newest available version by default); to achieve this goal you should use the following syntax:

pip install package_name==package_version

(note the double equals sign) e.g.,

pip install pygame==1.9.2

How to use pip: continued

If any of the currently installed packages are no longer needed and you want to get rid of them, pip will be useful, too. Its uninstall command will execute all the needed steps.

The required syntax is clear and simple:

pip uninstall package_name

so if you don't want pygame anymore you can execute the following command:

pip uninstall pygame

Pip will want to know if you're sure about the choice you're making – be prepared to give the right answer.

Use pip!
Pip's capabilities don't end here, but the command set we've presented to you is enough to start successfully managing packages that aren't a part of the regular Python installation.

We hope we've encouraged you to carry out your own experiments with pip and the Python package universe. PyPI invites you to dive into its extensive resources.

Some say that one of the most important programming virtues is laziness. Don't get us wrong – we don't want you to spend all day napping on the couch and dreaming of Python code.

A lazy programmer is a programmer who looks for existing solutions and analyzes the available code before they start to develop their own software from scratch.

This is why PyPI and pip exist – use them!

Key takeaways

1. A repository (or repo for short) designed to collect and share free Python code exists and works under the name Python Package Index (PyPI) although it's also likely that you come across a very niche name The Cheese Shop. The Shop's website is available at https://pypi.org/.

2. To make use of The Cheese Shop the specialized tool has been created and its name is pip (pip installs packages while pip stands for... ok, don't mind). As pip may not be deployed as a part of standard Python installation, it is possible that you will need to install it manually. Pip is a console tool.

3. To check pip's version one the following commands should be issued:

pip --version

or

pip3 --version

Check yourself which of these works for you in your OS' environment.

4. List of main pip activities looks as follows:

- pip help operation - shows brief pip's description;
- pip list - shows list of currently installed packages;
- pip show package_name - shows package_name info including package's dependencies;
- pip search anystring - searches through PyPI directories in order to find packages which name contains anystring;
- pip install name - installs name system-wide (expect problems when you don't have administrative rights);
- pip install --user name - install name for you only; no other your platform's user will be able to use it;
- pip install -U name - updates previously installed package;
- pip uninstall name - uninstalls previously installed package;

Exercise 1

Where does the name "The Cheese Shop" come from?

Check
It's a reference to an old Monty Python's sketch of the same name.

Exercise 2

Why should I ensure which one of pip and pip3 works for me?

Check
When Python 2 and Python 3 coexist in your OS, it's likely that pip identifies the instance of pip working with Python 2 packages only.

Exercise 3

How can I determine if my pip works with either Python 2 or Python 3?

Check
pip --version will tell you that.

Exercise 4

Unfortunately, I don't have administrative right. What should I do to install a package system-wide?

Check
You have to ask your sysadmin - don't try to hack your OS!

Congratulations! You have completed PE2: Module 1.

Well done! You've reached the end of Module 1 and completed a major milestone in your Python programming education. Here's a short summary of the objectives you've covered and got familiar with in Module 1:

- working with Python modules; importing, creating, and using modules;
- using selected Python STL modules (math, random, and platform)
- constructing and using packages in Python;

PIP (Python Package Installer.)

You are now ready to take the module quiz and attempt the final challenge: Module 1 Test, which will help you gauge what you've learned so far.