**Модуль 7: Створення та встановлення власних пакетів, віртуальне оточення**

Короткі та прості скрипти поміщаються в одному не дуже великому файлі до 400 рядків. Такі скрипти не складно підтримувати і розвивати. Але рано чи пізно настає момент, коли коду стає надто багато. Розмір єдиного файлу робить навігацію файлом складною і пошук у файлі потрібного місця займає надто багато часу. У такій ситуації зручно буде розбити великий скрипт на складові частини і структурувати ваш код. Це зробить роботу з проектом простішою і зручнішою.

Хороша структура проекту значно спрощує підтримку і розробку. Погана ж — навпаки істотно уповільнює роботу. І що більший проект, то це помітніше. Не лінуйтеся приділити час на продумування структури вашого проекту при кожній значній зміні. Ваші зусилля не пройдуть даремно і обов'язково окупляться в майбутньому.

**Створення пакетів та модулів**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson07/lesson-07/#%D1%81%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F-%D0%BF%D0%B0%D0%BA%D0%B5%D1%82%D1%96%D0%B2-%D1%82%D0%B0-%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%96%D0%B2)

Наприклад, усі функції, що мають загальне призначення, можна винести в окремий файл. "Точку входу" (скрипт, який викликається користувачем для початку роботи), допоміжні функції і налаштування в окремі файли, названі відповідно. В результаті ми отримаємо декілька відносно невеликих файлів, код в яких буде призначений для якогось одного завдання (налаштування застосунку, запуск застосунку, основна логіка). Такі файли з кодом у Python називаються модулями.

Ви можете імпортувати код (функції, змінні, що завгодно) із модулей за допомогою оператора виразу **from ... import ....**

Припустимо у вас є файл **salary\_calculations.py** і його вміст виглядає так:

**def add\_bonus(salary, bonus\_percents):  
 return salary \* (1 + bonus\_percents / 100)**

У тій самій папці лежить основний скрипт main.py, у якому ви б хотіли використати функцію **add\_bonus з salary\_calculations.py.** Щоб зробити це, нам достатньо імпортувати цю функцію:

**from salary\_calculations import add\_bonus  
  
salary = 1000  
bonus = 15  
salary\_with\_bonus = add\_bonus(salary, bonus)  
print(salary\_with\_bonus) *# 1150***

Ви також можете імпортувати увесь модуль за допомогою оператора **import** і тоді у вас буде доступ до усього вмісту модуля.

**import salary\_calculations  
  
salary = 1000  
bonus = 15  
salary\_with\_bonus = salary\_calculations.add\_bonus(salary, bonus)  
print(salary\_with\_bonus) *# 1150***

Щоб викликати функцію **add\_bonus з salary\_calculations.py**, коли ви імпортували увесь модуль, ви вказуєте ім'я модуля та ім'я функції через крапку.

Окремі файли з розширенням .**py** — це модулі. Ви можете імпортувати з них потрібні вам сутності (об'єкти, константи тощо) і використати в інших модулях. Зверніть увагу, що під час імпорту не потрібно вказувати розширення файлу **.py**, тільки ім'я.

Зі зростанням проекту кількість модулів (нагадаємо, що рекомендується не давати модулям занадто "розпухати") росте і вже пошук потрібного модуля починає викликати труднощі. У такій ситуації на допомогу приходять пакети Python.

Пакет — це папка, що містить модулі Python. Розташовуючи модулі по папкам, ви можете структурувати модулі за призначенням, і пошук потрібного модуля стане набагато простішим.

Робота з пакетами аналогічна роботі з модулями, ви можете імпортувати потрібну вам сутність (наприклад функцію), імпортувавши модуль з пакету. Припустимо, що модуль **salary\_calculations.py** ви розмістили у папці **calculations** поряд з **main.py**, де вам знадобилася функція **add\_bonus з salary\_calculations.py**. В такому випадку ви можете імпортувати модуль **salary\_calculations.py:**

**from calculations import salary\_calculations  
  
salary = 1000  
bonus = 15  
salary\_with\_bonus = salary\_calculations.add\_bonus(salary, bonus)  
print(salary\_with\_bonus) *# 1150***

або тільки функцію:

**from calculations.salary\_calculations import add\_bonus  
  
salary = 1000  
bonus = 15  
salary\_with\_bonus = add\_bonus(salary, bonus)  
print(salary\_with\_bonus) *# 1150***

Розбиття на пакети і модулі може тривати стільки, скільки ви вважаєте за необхідне і зручне. Будь-який пакет може містити пакети в собі, а ті — ще пакети, і так далі. Основне правило — це називати пакети та модулі так само, як і змінні Python (тільки літери, цифри та \_, ім'я не починається з цифри). Також дуже важливо продумати структуру проекту так, щоб вам було зручно потім шукати у ньому потрібні модулі.

**Навіщо потрібен init.py[​](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson07/lesson-07/" \l "%D0%BD%D0%B0%D0%B2%D1%96%D1%89%D0%BE-%D0%BF%D0%BE%D1%82%D1%80%D1%96%D0%B1%D0%B5%D0%BD-initpy" \o "Пряме посилання на навіщо-потрібен-initpy)**

У версіях Python до 3.3 в пакетах обов'язково потрібно було розмістити допоміжний файл **\_\_init\_\_.py**. Якщо цього не зробити, то Python не сприймав папку як пакет та імпортувати з такої папки нічого не міг. Зараз в цьому немає потреби, але часто такі файли створюються для зворотної сумісності зі старими версіями.

Файл **\_\_init\_\_.py**— це службовий файл, який інтерпретатор обов'язково виконає під час першого імпорту пакету. Таким чином, якщо вам потрібно виконати якісь дії під час імпорту пакету, ви можете прописати їх у **\_\_init\_\_.py**.

Зазвичай **\_\_init\_\_.py**— порожній і нічого не робить. Але, коли структура пакету не занадто проста і там багато модулів та/або пакетів, про які користувачеві знати не обов'язково, ви можете імпортувати те, що користувачеві потрібно у **\_\_init\_\_.py**. У такому випадку користувач зможе вже у своєму коді прописати скорочені варіанти імпортів. Наприклад, у пакеті **utility** є два пакети: **useful** та **dummy**. В кожному з них є модуль **functions.py**(у кожного свій). А в цих модулях вже є функції **nice\_function**та **not\_bad**відповідно. Користувачеві пакету **utility** необов'язково знати про внутрішню структуру пакету, вона зроблена для зручності розробника пакету. Розробник написав **utility** так, щоб надати користувачеві доступ до **nice\_function** та **not\_bad**.

Якщо файл **\_\_init\_\_.py** порожній, то використання **nice\_function** та **not\_bad**буде виглядати якось так:

**import utility  
  
utility.useful.functions.nice\_function()  
utility.dummy.functions.not\_bad()**

або:

**from utility.useful.functions import nice\_function  
from utility.dummy.functions import not\_bad  
  
nice\_function()  
not\_bad()**

Обидва варіанти припускають, що користувачеві пакету потрібно буде розібратися, де там що лежить.

Якщо ж розробник подумав про користувача і **\_\_init\_\_.py** виглядає ось так:

**from utility.useful.functions import nice\_function  
from utility.dummy.functions import not\_bad  
  
\_\_all\_\_ = ['nice\_function', 'not\_bad']**

Тоді можна скористатися таким імпортом:

**from utility import nice\_function, not\_bad  
  
nice\_function()  
not\_bad()**

або:

**from utility import \*  
  
nice\_function()  
not\_bad()**

Зверніть увагу на константу **\_\_all\_\_**— це список модулів або пакетів, які імпортуються, якщо у виразі **from ... import \***в кінці вказаний символ **\***.

**Встановлення пакетів за допомогою pip**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson07/lesson-07/#%D0%B2%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D0%BF%D0%B0%D0%BA%D0%B5%D1%82%D1%96%D0%B2-%D0%B7%D0%B0-%D0%B4%D0%BE%D0%BF%D0%BE%D0%BC%D0%BE%D0%B3%D0%BE%D1%8E-pip)

Встановлення додаткових пакетів в Python здійснюється менеджерами пакетів. Стандартний (але не єдиний) менеджер пакетів Python — це **pip. pip**може:

* встановлювати пакети <https://pypi.org/>
* встановлювати пакети з локального репозиторію або розташованого на будь-якому ресурсі, що підтримує протокол **pip**.
* видаляти пакети
* показувати список встановлених доступних пакетів (з версіями або без)
* показувати, які пакети застаріли (зі встановлених) і їх можна оновити
* видаляти пакети

pip написаний на Python і є пакетом Python, який входить в стандартне постачання (попередньо встановлений). Ви можете, звичайно, окремо його встановити або оновити як будь-який інший пакет, якщо потрібно.

Використання **pip** передбачає виконання його як скрипту Python з аргументами під час виклику. Наприклад, щоб вивести список встановлених пакетів:

**python3 -m pip list**

Значення -**m** вказує, що потрібно викликати пакет як скрипт, що виконується, **list** — це аргумент, за яким **pip** "розуміє", що потрібно зробити (вивести список встановлених пакетів).

Часто в системі прописано, що **pip** запускається за допомогою Python і ви можете викликати **pip** напряму:

**pip list**

Повна документація з **pip** доступна на [сторінці офіційної документації](https://pip.pypa.io/en/stable/).

* встановлення останньої версії пакету **some\_package: pip install some\_package**
* встановлення версії **1.12.3** пакету **some\_package: pip install some\_package==1.12.3**
* встановлення версії пакету **some\_package** новішого за **1.12.3: pip install some\_package>=1.12.3**
* встановлення версії пакету **some\_package** давнішого за **1.12.3: pip install some\_package<=1.12.3**
* видалення пакету **some\_package: pip uninstall some\_package**
* список встановлених пакетів: **pip list**
* список встановлених пакетів з версіями: **pip freeze**
* встановлення пакетів з текстового файлу **requirements.txt: pip install -r requirements.txt**

**Створення віртуального оточення**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson07/lesson-07/#%D1%81%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B2%D1%96%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE-%D0%BE%D1%82%D0%BE%D1%87%D0%B5%D0%BD%D0%BD%D1%8F)

Встановлювати пакети Python в систему глобально — не найкраща ідея, оскільки встановлені пакети захаращують систему і можуть конфліктувати. Конфлікти встановлених пакетів — це одна з проблем, що найважче виправляються, при роботі з Python. "Поламані" залежності можуть забрати багато часу навіть у дуже досвідченого розробника. Тим паче складніше впоратися з цим новачкам.

Щоб уникнути проблем із залежностями, і встановлення потрібних для якогось проекту пакетів не впливала на всю вашу операційну систему та інші проекти, рекомендується використати віртуальне оточення Python.

Оскільки пакет у Python — це папки з файлами-модулями, ми можемо створити окрему папку для пакетів-залежностей всередині кожного проекту. Якщо вказати Python, що працюючи в папці проекту, імпортувати потрібно саме з цієї папки, то тоді ми зможемо ізолювати нашу операційну систему і наші проекти один від одного. Така папка з пакетами-залежностями — це і є вуртуальне оточення.

Звичайно, існує безліч інструментів, які автоматизують для нас створення /видалення/ активування віртуального оточення. Найпростіший і найстандартніший інструмент — це **venv.**

Почитати, що вміє **venv**, можна на [сторінці офіційної документації](https://docs.python.org/3/library/venv.html). Так само як і **pip, venv**— це консольний Python скрипт.

Щоб створити нове віртуальне оточення, виконайте: **python3 -m venv /path/to/new/virtual/environment або venv /path/to/new/virtual/environment**. Після виконання цієї команди у **/path/to/new/virtual/environment** буде скопійований Python з мінімальним набором стандартних пакетів. І ви зможете використати цей Python окремо від основного системного або будь-якого іншого.

Щоб почати використовувати Python з віртуального оточення, виконайте в консолі: **\path\to\new\virtual\environment\Scripts\activate.bat** або **source** **/path/to/new/virtual/environment/bin/activate** для Linux / Mac OS. Якщо ви використовуєте PowerShell, то можна виконати **\path\to\new\virtual\environment\Scripts\Activate.ps1.**

Після активації в цій консолі виклик **python** буде викликати не системний Python, а версію з **/path/to/new/virtual/environment** зі своїми пакетами.

Активувавши віртуальне оточення, ви можете встановлювати/видаляти пакети за допомогою **pip** у віртуальному оточенні. Встановлення/видалення пакетів впливає тільки на віртуальне оточення і не зачіпає систему або інші проекти.

Щоб повернутися до системного Python, виконайте: **deactivate.**

Щоб повністю видалити віртуальне оточення, достатньо видалити папку **\path\to\new\virtual\environment\** з усім її вмістом.

**Створення встановлюваних пакетів, структура setup.py**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson07/lesson-07/#%D1%81%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B2%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BB%D1%8E%D0%B2%D0%B0%D0%BD%D0%B8%D1%85-%D0%BF%D0%B0%D0%BA%D0%B5%D1%82%D1%96%D0%B2-%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0-setuppy)

Кінцевим результатом роботи програміста є застосунок. Щоб користувач міг встановити і використати ваш застосунок, ви можете зробити встановлюваний пакет з вашого застосунку.

Щоб створити пакет, вам потрібно дотримуватися ряду вимог.

**Структура пакету:**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson07/lesson-07/#%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0-%D0%BF%D0%B0%D0%BA%D0%B5%D1%82%D1%83)

Припустимо у вас є пакет **useful**. У цьому пакеті є модулі **some\_code.py та another\_code.py.** У цих модулях знаходиться код, який виконує корисну роботу.

Щоб ваш пакет можна було встановити, вам варто розмістити сам пакет всередині папки, яка називається так само:

**useful/  
 useful/  
 \_\_init\_\_.py  
 some\_code.py  
 another\_code.py  
 setup.py**

Поряд з самим пакетом варто розмістити модуль з інструкціями зі встановлення — **setup.py**. До вмісту **setup.py** ми повернемося трохи пізніше, а поки що продовжимо розгляд структури пакету.

Python за замовчуванням проігнорує під час встановлення усі файли, які не закінчуються на **.py** Якщо у вашому пакеті не буде **\_\_init\_\_.py**, то вміст **useful/useful/**теж буде проігноровано (сподіваюся це виправлять в наступних версіях). Назва пакету повинна відповідати вимогам до імен змінних Python.

**setup.py**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson07/lesson-07/#setuppy)

Модуль, що містить інструкції зі встановлення, викликає функцію **setup** з пакету **setuptools.** Функція **setup** здійснює встановлення пакету в системі і містить параметри, що конфігурують встановлення.

Детальні інструкції з написання **setup.py**ви можете отримати на [сторінці документації](https://docs.python.org/3/distutils/setupscript.html).

Приклад вмісту **setup.py:**

**from setuptools import setup  
  
setup(name='useful',  
 version='1',  
 description='Very useful code',  
 url='http://github.com/dummy\_user/useful',  
 author='Flying Circus',  
 author\_email='flyingcircus@example.com',  
 license='MIT',  
 packages=['useful'])**

В цьому прикладі ми викликаємо **setup** з додатковими інформаційними параметрами, які будуть доступні користувачам. А саме, ми вказали ім'я пакету, версію, короткий опис пакету, адресу, де можна подивитися початковий код, ім'я автора, його email, ліцензію, набір пакетів, які містяться у постачанні.

Що, якщо наш пакет досить великий і прописувати вручну усі модулі **packages**незручно, та існує ризик помилитися? Тоді у **setuptools**є функція **find\_namespace\_packages**, яка допоможе знайти всі модулі і не пропустити нічого:

**from setuptools import setup, find\_namespace\_packages  
  
setup(  
 name='useful',  
 version='1',  
 description='Very useful code',  
 url='http://github.com/dummy\_user/useful',  
 author='Flying Circus',  
 author\_email='flyingcircus@example.com',  
 license='MIT',  
 packages=find\_namespace\_packages()  
)**

Такий пакет можна опублікувати на PyPi і тоді його можна буде встановити за допомогою **pip**, або опублікувати початковий код, і тоді можна буде встановити з початкових кодів.

Щоб встановити цей пакет з початкового коду, виконайте в консолі **pip install .**або **pip install -e .** у папці, де лежить **setup.py.**

**Додавання не py файлів у пакет**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson07/lesson-07/#%D0%B4%D0%BE%D0%B4%D0%B0%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F-%D0%BD%D0%B5-py-%D1%84%D0%B0%D0%B9%D0%BB%D1%96%D0%B2-%D1%83-%D0%BF%D0%B0%D0%BA%D0%B5%D1%82)

Коли вам потрібно додати до пакету на **.py**файл (зображення, README.md тощо) варто пам'ятати, що **setuptools** проігнорує всі не **.py**файли. Потрібно явно вказати додати файли в пакет, якщо вам це потрібно. Один зі способів додати файли в постачання — це вказати параметр **include\_package\_data=True**у виклику **setup**та прописати, які файли потрібно додати у постачання у **MANIFEST.in.**

**MANIFEST.in**— це файл поряд з **setup.py**, і в ньому вказуються шляхи до всіх файлів, які потрібно додати у постачання. Приклад **MANIFEST.in** для додавання файлу **README.md**:

**include README.md**

І структура пакету буде:

**useful/  
 useful/  
 \_\_init\_\_.py  
 some\_code.py  
 another\_code.py  
 setup.py  
 MANIFEST.in  
 README.md**

Тепер файл **README.md** буде доданий до пакету і буде доступний. Шлях до файлів, які потрібно включити у постачання, може бути будь-яким відносно пакету.

**Управління залежностями пакету**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson07/lesson-07/#%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%96%D0%BD%D0%BD%D1%8F-%D0%B7%D0%B0%D0%BB%D0%B5%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8F%D0%BC%D0%B8-%D0%BF%D0%B0%D0%BA%D0%B5%D1%82%D1%83)

Якщо в нашому пакеті є залежності, щоб він запрацював, потрібно встановити додаткові пакети, потрібно їх всі прописати в параметрі **install\_requires**:

**from setuptools import setup, find\_namespace\_packages  
  
setup(  
 name='useful',  
 version='1',  
 description='Very useful code',  
 url='http://github.com/dummy\_user/useful',  
 author='Flying Circus',  
 author\_email='flyingcircus@example.com',  
 license='MIT',  
 packages=find\_namespace\_packages(),  
 install\_requires=['markdown'],  
)**

В цьому прикладі наш пакет буде вимагати встановити спочатку пакет **markdown**перед встановленням. Порядок встановлення залежностей визначає сам менеджер пакетів (**pip**наприклад).

**Консольний скрипт як пакет**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson07/lesson-07/#%D0%BA%D0%BE%D0%BD%D1%81%D0%BE%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9-%D1%81%D0%BA%D1%80%D0%B8%D0%BF%D1%82-%D1%8F%D0%BA-%D0%BF%D0%B0%D0%BA%D0%B5%D1%82)

Якщо наш пакет містить застосунок, який можна викликати з консолі, зручно буде додати можливість виклику цього застосунку у будь-якому місці нашої системи з консолі. Для цього у виклику **setup**додамо ще один параметр — **entry\_points**. Цей параметр приймає словник, де ми можемо вказати список "точок входу" для ключа **console\_scripts**.

Наприклад, в нашому пакеті у модулі **some\_code.py**є функція **hello\_world**, яка виводить у консоль повідомлення **Hello World!.** Після встановлення пакету ми зможемо в будь-якому місці нашої системи виконати в консолі команду: **helloworld**і отримаємо у відповідь **Hello World!.**

Щоб це працювало в системі, Python повинен викликатися при виклику файлів з розширенням **.py**та **setup.py** повинен бути змінений:

**from setuptools import setup, find\_namespace\_packages  
  
setup(  
 name='useful',  
 version='1',  
 description='Very useful code',  
 url='http://github.com/dummy\_user/useful',  
 author='Flying Circus',  
 author\_email='flyingcircus@example.com',  
 license='MIT',  
 packages=find\_namespace\_packages(),  
 install\_requires=['markdown'],  
 entry\_points={'console\_scripts': ['helloworld = useful.some\_code:hello\_world']}  
)**

У списку точок входу **console\_scripts**можуть бути файли, що виконуються **(.exe)**, скрипти Bash, cmd, PowerShell і будь-який інший файл, який операційна система зможе виконати.