Python: Винятки та бібліотеки

Прикладна аналітика при розробці IT

Ігор Мірошниченко
КНУ імені Тараса Шевченка, ФІТ

ЗМІСТ

- Помилки та винятки
- Бібліотеки
- Аргументи командного рядка
- Пакети

ПОМИЛКИ ТА ВИНЯТКИ

SyntaxError

Винятки в Python, як і в інших мовах програмування, відносяться до проблем у вашому коді.

Почнемо з простого прикладу: створимо файл `hello.py`:

```
Terminal
1  code cat.py
```

Напишемо простий код з помилкою:

```
1  print("Привіт, світ!)
```

```
SyntaxError: unterminated string literal (detected at line 1) (4163655757.py, line 1)
```

Ми отримали помилку `SyntaxError`, яка означає, що Python не зміг зрозуміти наш код. Це найпростіший вид помилки, який можна виправити, виправивши помилку в коді.

Запис `unterminated`“, як правило, означає, що я щось почав, але не зупинив. Запис `string` - це послідовність символів, з якою ми вже знайомі. А `literal` зазвичай відноситься до того, що ви буквально набрали.

ValueError

Але в Python є багато інших типів помилок, які можна назвати `RuntimeError`, які трапляються під час роботи вашого коду.

Саме від вас залежить написання додаткового захисного коду для виявлення таких помилок.

Наприклад, давайте створимо файл `number.py`, яка буде приймати число від користувача і виводити це число на екран:

Terminal

```
code number.py
```

```
number = int(input("Введіть число: ")) # 5
print(f'Ваше число: {number}')
```

```
Ваше число: 5
```

ValueError

Але що станеться, якщо користувач введе не число, а текст? Наприклад, введе слово `п'ять`:

```
1  number = int(input("Введіть число: ")) # п'ять
2  print(f'Ваше число: {number}')
```

```
ValueError: invalid literal for int() with base 10: "п'ять"
```

Ми отримали помилку `ValueError`, яка означає, що ми передали функції `int()` значення, яке вона не може перетворити на число.

Це вже не синтаксична помилка, а помилка, яка виникає під час виконання програми. І вирішення цієї помилки вже не залежить від вас, а від користувача, який вводить дані.

Тому вам потрібно написати додатковий код, який буде перевіряти введені дані на коректність. Як це зробити у Python? Виявляється у Python є **ключові слова** які можуть допомогти з цим.

КЛЮЧОВІ СЛОВА `try` ТА `except`

Якщо ви хочете спробувати зробити щось у Python, ви можете використовувати це ключове слово `try`.

За рахунок нього ви можете перевірити, чи сталося щось помилкове. Отже, використовуючи `try`, я можу спробувати щось зробити, але якщо щось піде не так, я можу замість використати ключове слово `except`.

Подивимось, як це працює:

```
1  try:
2      number = int(input("Введіть число: ")) # п'ять
3      print(f'Ваше число: {number}')
4  except ValueError:
5      print('Ви ввели не число')
```

```
Ви ввели не число
```

NameError

Якби я був впевнений, що функція `print()` не призведе до помилки, я міг би переписати код наступним чином і отримати нову помилку:

```
1  try:
2      number = int(input("Введіть число: ")) # п'ять
3  except ValueError:
4      print('Ви ввели не число')
5
6  print(f'Ваше число: {number}')
```

```
Ви ввели не число
NameError: name 'number' is not defined
```

Помилка `NameError` означає, що я використовую змінну, яка не існує. Ця помилка виникла в наслідок порядку виконання операцій.

Помилка виникає в частині коду `int(input("Введіть число: "))`, яка знаходиться праворуч від оператора `=` і це приводить до того, що змінна `number` не створюється.

То як вирішити цю проблему? Ви можете використовувати ключове слово `else`.

КЛЮЧОВЕ СЛОВО `else`

Інтуїція ключового слова `else` схожа на інтуїцію ключового слова `else` в умовних конструкціях.

Якщо в блоці `try` не виникає помилок, то виконується блок `else`:

```
1  try:
2      number = int(input("Введіть число: ")) # п'ять
3  except ValueError:
4      print('Ви ввели не число')
5  else:
6      print(f'Ваше число: {number}')
```

```
Ви ввели не число
```

КЛЮЧОВЕ СЛОВО `else`

З мого боку трохи неввічливо відкидати вхідні дані користувача після того, як він не зміг ввести ціле число, і просто виходити з програми.

Було б зручніше, якби я просто підказував або перепитував користувача знову і знову.

І для цього я можу використовувати цикл `while`:

```
while True:
    try:
        number = int(input("Введіть число: ")) # п'ять
    except ValueError:
        print('Ви ввели не число')
    else:
        print(f'Ваше число: {number}')
        break
```

КЛЮЧОВЕ СЛОВО `else`

Уявіть ситуацію, що я буду намагатися отримати від користувача досить багато цифр.

Було б непогано створити власну функцію, наприклад `get_int()`, яка буде повторювати запит на введення цілого числа, поки користувач не введе ціле число:

```
 1  def main():
 2      number = get_int()
 3      print(f'Ваше число: {number}')
 4
 5  def get_int():
 6      while True:
 7          try:
 8              number = int(input("Введіть число: ")) # п'ять
 9          except ValueError:
10              print('Ви ввели не число')
11          else:
12              return number
13
14  main()
```

КЛЮЧОВЕ СЛОВО pass

Підемо далі, і припустимо, що ми хочемо ловити помилки, але не хочемо виводити повідомлення про помилку.

Для цього ми можемо використовувати ключове слово pass:

```
 1  def main():
 2      number = get_int()
 3      print(f'Ваше число: {number}')
 4
 5  def get_int():
 6      while True:
 7          try:
 8              number = int(input("Введіть число: ")) # п'ять
 9          except ValueError:
10              pass
11          else:
12              return number
13
14  main()
```

Це просто ще один механізм для обробки помилок.

Іноді ви можете використовувати pass, якщо ви хочете, щоб ваш код був синтаксично правильним, але ви не хочете нічого робити, якщо виникає помилка.

КЛЮЧОВЕ СЛОВО `pass`

Ми можемо скоротити цей код і перенести ключове слово `return` вище:

```
 1  def main():
 2      number = get_int()
 3      print(f'Ваше число: {number}')
 4
 5  def get_int():
 6      while True:
 7          try:
 8              return int(input("Введіть число: ")) # п'ять
 9          except ValueError:
10              pass
11
12  main()
```

Це дозволить дещо скоротити код, але може бути дещо складнішим для розуміння. Який варіант обрати - це вже залежить від вас.

КЛЮЧОВЕ СЛОВО `pass`

Тепер давайте створимо більш загальний варіант цієї програми і позбудемось від жорсткого коду.

Зробимо функцію `get_int()` більш універсальною: було б добре, якби функція `main()` не знала, як функція `get_int()` описує свої змінні. В нашому випадку це `'Введіть число'`, але можна було б написати `'Введіть ціле число'` або `'Введіть додатнє число'` або ще щось.

Для цього ми додамо параметр `prompt` до функції `get_int()`, який буде використовуватися для виведення повідомлення користувачу:

```
 1  def main():
 2      number = get_int('Введіть число: ')
 3      print(f'Ваше число: {number}')
 4
 5  def get_int(prompt):
 6      while True:
 7          try:
 8              return int(input(prompt)) # п'ять
 9          except ValueError:
10              pass
11
12  main()
```

ІНШІ ВИПАДКИ

Це далеко не все, що необхідно знати про винятки.

Наприклад, ви можете використовувати ключове слово `raise`, щоб викликати виняток, або використовувати ключове слово `finally`, щоб виконати певний код незалежно від того, чи виникла помилка, чи ні.

Але це вже виходить за рамки цього курсу. Якщо ви хочете дізнатися більше про винятки, ви можете:

- прочитати документацію за посиланням docs.python.org/3/tutorial/errors.html
- переглянути матеріали W3Schools
- переглянути матеріали GeeksforGeeks

БІБЛІОТЕКИ

ІМПОРТ БІБЛІОТЕК

Бібліотеки - це файли коду, написані людьми, які ви можете використовувати у своїх програмах.

Python підтримує ідею можливості ділитися кодом з іншими і робить це за допомогою **модулів** (англ. *modules*) - це бібліотека, яка зазвичай має низку функцій, класів і інших речей, які ви можете використовувати у своїх програмах.

Python встановлюється з низкою базових модулів, які ви можете використовувати у своїх програмах: робота з файлами, мережею, даними, датами і часом, математичними функціями і багато іншого. Ці модулі називаються **стандартними бібліотеками** (англ. *standard libraries*).

> **Примітка**
>
> Документація до стандартних бібліотек доступна за посиланням https://docs.python.org/3/library/.

ІМПОРТ БІБЛІОТЕК

Щоб використовувати бібліотеку, її потрібно імпортувати.

Це робиться за допомогою ключового слова `import` і назви бібліотеки.

Приклад підкидання монетки:

Оскільки ми маємо справу з випадковим процесом (підкидання монетки), то нам знадобиться модуль `random`. Щоб його використати, ми повинні його імпортувати:

```
1  import random
2
3  coin = ['орел', 'решка']
4  print(random.choice(coin))
```

```
орел
```

Запис `random.choice(coin)` означає, що ми використовуємо функцію `choice()` з модуля `random` і передаємо їй список `coin` як аргумент. Ця функція повертає випадковий елемент зі списку `coin`, який ми виводимо на екран.

Примітка

Документація до модуля `random` доступна за посиланням docs.python.org/3/library/random.html.

РАНДОМІЗАЦІЯ

Але що, якщо нам потрібно використати лише одну функцію з цієї бібліотеки? Тоді ми можемо імпортувати лише цю функцію за допомогою комбінації ключових слів `from` та `import`:

```
1  from random import choice
2
3  coin = ['орел', 'решка']
4  print(choice(coin))
```

```
решка
```

Тепер ми можемо використовувати функцію `choice()` без префіксу `random`.

РАНДОМІЗАЦІЯ

Продовжимо досліджувати модуль `random`.

Цього разу це буде функція `random.randint(a, b)`, яка повертає випадкове ціле число від `a` до `b` включно.

Давайте використаємо цю функцію, щоб симулювати підкидання грального кістки з 20 гранню[1]:

```
1  import random
2
3  d20 = random.randint(1, 20)
4  print(d20)
```

```
14
```


1. 20-гранна кістка використовується у деяких настільних іграх, наприклад, у Dungeons & Dragons

ВІДТВОРЮВАНІСТЬ

Щоб зробити наші експерименти відтворюваними, ми можемо використовувати функцію `random.seed()`.

Ця функція приймає один аргумент - ціле число, яке використовується для генерації випадкових чисел.

```
1  import random
2
3  random.seed(42)
4  print(random.randint(1, 20))
```

```
4
```

```
1  import random
2
3  random.seed(2023)
4  print(random.randint(1, 20))
```

```
13
```

```
1  import random
2
3  random.seed(1)
4  print(random.randint(1, 20))
```

```
5
```

ТАСУВАННЯ КАРТ

Розглянемо наступну функцію `random.shuffle(x)`, яка перемішує елементи списку `x` у випадковому порядку. Давайте використаємо цю функцію, щоб перемішати колоду карт:

```
1  import random
2
3  cards = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']
4
5  random.shuffle(cards)
6
7  for card in cards:
8      print(card, end=' ')
```

```
4 Q 2 A 8 7 5 10 9 K 6 3 J
```

Попередження

Зверніть увагу, що функція `random.shuffle(x)` **не повертає** перемішаний список `x`, а просто перемішує його **на місці**.

Уважно читайте документацію!

СТАТИСТИКА

Python також постачається з бібліотекою `statistics`, яка містить різноманітні статистичні функції для проведення базового дослідження мір центральної тенденції, варіації та залежностей.

Створимо програму, яка буде розраховувати середнє арифметичне значення списку чисел:

Імпортуємо модуль `statistics` та розрахуємо середнє значення поточної успішності студента за допомогою функції `mean()`:

```
1  import statistics
2
3  grades = [
4      12, 10, 7, 12, 9, 10, 12,
5      8, 11, 12, 10, 9, 8, 11, 12,
6      10, 9, 8, 11, 12, 10, 9, 8, 11
7  ]
8
9  print(statistics.mean(grades))
```

```
10.041666666666666
```

Примітка

АРГУМЕНТИ КОМАНДНОГО РЯДКА

АРГУМЕНТИ КОМАНДНОГО РЯДКА

Існує ще більше функціональних можливостей, які постачаються з Python. Серед них є функції для роботи з аргументами командного рядка (англ. *command-line arguments*).

Створимо файл `name.py`, в якому будемо використовувати новий модуль `sys`:

Terminal

```
code name.py
```

Модуль `sys` дозволяє взаємодіяти з інтерпретатором Python. Цей модуль містить різноманітні функції, які дозволяють отримати доступ до аргументів командного рядка, змінних і функцій, які використовуються інтерпретатором Python.

> **Примітка**
>
> Документація до модуля `sys` доступна за посиланням docs.python.org/3/library/sys.html.

sys.argv

Ми зосередимо увагу на функції `sys.argv`, яка повертає список аргументів командного рядка, переданих програмі. Давайте використаємо цю функцію, щоб вивести на екран другий аргумент командного рядка:

```
1 import sys
2
3 print('Привіт, мене звати ' + sys.argv[1] + '!')
```

Але цього разу замість команди `python name.py` ми виконаємо команду `python name.py Ігор`, яка передасть ім'я `Ігор` у якості аргумента командного рядка.

Terminal

```
1 python name.py Ігор
```

В результаті ми отримаємо наступне повідомлення:

```
1 Привіт, мене звати Ігор!
```

sys.argv

Якщо ж виконати цей код без аргументів командного рядка, то ми отримаємо помилку:

```
1  Traceback (most recent call last):
2    File "name.py", line 4, in <module>
3      print('Привіт, мене звати ' + sys.argv[1] + '!')
4
5  IndexError: list index out of range
```

Це означає, що ми намагаємося отримати доступ до елемента списку, якого не існує.

Це тому, що список `sys.argv` не містить жодного елемента, якщо не передати аргументів командного рядка.

sys.argv

Давайте виправимо цю помилку, додавши перевірку наявності аргументів командного рядка:

```
1  import sys
2
3  try:
4      print('Привіт, мене звати ' + sys.argv[1] + '!')
5  except IndexError:
6      print('Введіть своє ім\'я як аргумент командного рядка!')
```

Виконаємо код з аргументами командного рядка:

Terminal

```
1  python name.py Ігор
```

```
1  Привіт, мене звати Ігор!
```

Виконаємо код без аргументів командного рядка:

Terminal

```
1  python name.py
```

sys.argv

З іншої сторони, ми можемо перевірити кількість аргументів командного рядка за допомогою функції `len()`:

```
1  import sys
2
3  if len(sys.argv) < 2:
4      print('Мало аргументів командного рядка!')
5  elif len(sys.argv) > 2:
6      print('Багато аргументів командного рядка!')
7  else:
8      print('Привіт, мене звати ' + sys.argv[1] + '!')
```

```
Багато аргументів командного рядка!
```

Виконаємо код з аргументами командного рядка:

Terminal

```
python name.py Ігор
```

```
Привіт, мене звати Ігор!
```

Виконаємо код без аргументів командного рядка:

Terminal

```
python name.py
```

```
Мало аргументів командного рядка!
```

Виконаємо код з багатьма аргументами командного рядка:

Terminal

```
python name.py Ігор Мірошниченко
```

```
Багато аргументів командного рядка!
```

sys.argv

Одна з речей, яка мені не подобається у попередній версії коду полягає у тому, що суть моєї програми винесено у блок `else`. Є щось приємне в тому, щоб тримати всю обробку помилок окремо від коду, який вас дійсно цікавить. Існує кращий спосіб зробити це, використовуючи функцію `exit()`:

```
1  import sys
2
3  # Перевіряємо кількість аргументів командного рядка
4  if len(sys.argv) < 2:
5      sys.exit('Мало аргументів командного рядка!')
6  elif len(sys.argv) > 2:
7      sys.exit('Багато аргументів командного рядка!')
8
9  # Виводимо повідомлення
10 print('Привіт, мене звати ' + sys.argv[1] + '!')
```

Функція `exit()` приймає один аргумент - повідомлення, яке буде виведено на екран, і зупиняє виконання програми. Тепер ми можемо використовувати функцію `print()` без блоку `else`.

Виконаємо код без аргументів командного рядка:

Terminal

```
1  python name.py
```

```
1  Мало аргументів командного рядка!
```

sys.argv

Припустимо, що я хочу, щоб друкувалося не тільки моє ім'я, але й імена інших користувачів. Для цього ми можемо використати зрізи (англ. *slices*) списків. Для цього використовуються квадратні дужки `[]`, які вказують, які елементи списку потрібно вивести. Перепишемо програму `name.py`:

```
1  import sys
2
3  if len(sys.argv) < 2:
4      sys.exit('Мало аргументів командного рядка!')
5
6  for name in sys.argv[1:]:
7      print('Привіт, мене звати ' + name + '!')
```

Виконаємо код з аргументами командного рядка:

Terminal

```
1  python name.py Ігор Анна Яромир Артур Святослав
```

```
1  Привіт, мене звати Ігор!
2  Привіт, мене звати Анна!
3  Привіт, мене звати Яромир!
4  Привіт, мене звати Артур!
5  Привіт, мене звати Святослав!
```

ПАКЕТИ

ПАКЕТИ

Однією з причин того, що Python є настільки популярним і потужним в наші дні, є те, що існує багато бібліотек сторонніх розробників, також відомих як **пакети** (англ. *packages*).

Строго кажучи, у самій мові Python є термін “пакети”, який по суті є модулем, реалізованим у папці, не просто у файлі. Але в більш загальному сенсі, **пакети** - це бібліотека сторонніх розробників, яку ми з вами можемо встановити на наш власний комп’ютер і отримати доступ до ще більшої функціональності, яку інші люди реалізували для нас.

Одне з місць, де ви можете отримати всі ці пакунки, називається **PyPI** (*The Python Package Index*). Це репозиторій, де розробники можуть розміщувати свої пакети, щоб інші люди могли їх встановити. Існує багато інших репозиторіїв, але PyPI є найбільш популярним.

Для встановлення пакетів, Python використовує менеджер пакетів `pip` - це програма, яка поставляється з Python і дозволяє встановлювати пакети.

COWSAY

Пакет `cowsay`, який дозволяє виводити повідомлення на екран у вигляді корови. Давайте встановимо пакет `cowsay` за допомогою команди `pip install cowsay`:

Terminal

```
pip install cowsay
```

> **Примітка**
>
> Документація до пакету `cowsay` доступна за посиланням pypi.org/project/cowsay/.

Тепер давайте створимо файл `cow.py`, який буде виводити повідомлення на екран у вигляді корови:

Terminal

```
code cow.py
```

Імпортуємо модуль `cowsay` та використаємо функцію `cowsay.cow()` для виведення повідомлення на екран у вигляді корови:

```
import cowsay

cowsay.cow('Привіт, мене звати Бакбик!')
```

```
 _____________________________
| Привіт, мене звати Бакбик! |
 =============================
                            \
                             \
                               ^__^
                               (oo)\_______
                               (__)\       )\/\
                                   ||----w |
                                   ||     ||
```

API

API (англ. *Application Programming Interface*) - це спосіб, за допомогою якого програми можуть взаємодіяти одна з одною.

Пакет, з яким ми будемо знайомитися, називається `requests`. Цей пакет дозволяє нам виконувати HTTP-запити до веб-сайтів.

Для його встановлення виконаємо команду:

Terminal

```
pip install requests
```

Примітка

Документація до пакету `requests` доступна за посиланням pypi.org/project/requests/.

API

Створимо новий файл `itunes.py`, який буде використовувати API iTunes.

Apple має власний API для свого сервісу iTunes, яке надає вам можливість завантажувати і шукати музику і пісні, а також іншу інформацію.

Terminal

```
code itunes.py
```

Якщо переглянути документацію до API iTunes, то ми побачимо, що для пошуку пісень потрібно використовувати адресу `entity=song`.

Для перегляну інформації про одну пісню необхідно до базової адреси додати `limit=1`.

Для пошуку пісень гурту Korn необхідно до базової адреси додати `term=korn`.

Таким чином, ми отримаємо наступну адресу:

```
https://itunes.apple.com/search?entity=song&limit=1&term=korn
```

API

Якщо перейти за цією адресою, ми отримаємо текстовий файл, зміст котрого буде виглядати приблизно наступним чином:


```
{
 "resultCount":1,
 "results": [
{"wrapperType":"track", "kind":"song", "artistId":466532, "collectionId":423045626, "trackId":423045744, "artistName":"Korn", "collectionName":"Korn",
"trackName":"Blind", "collectionCensoredName":"Korn", "trackCensoredName":"Blind", "artistViewUrl":"https://music.apple.com/us/artist/korn/466532?
uo=4", "collectionViewUrl":"https://music.apple.com/us/album/blind/423045626?i=423045744&uo=4",
"trackViewUrl":"https://music.apple.com/us/album/blind/423045626?i=423045744&uo=4",
"previewUrl":"https://audio-ssl.itunes.apple.com/itunes-assets/AudioPreview115/v4/ef/a4/dd/efa4dd64-e8e0-6c12-c88b-
a3a996b24b12/mzaf_13798363446300996858.plus.aac.p.m4a", "artworkUrl30":"https://is2-ssl.mzstatic.com/image/thumb/Music114/v4/09/2b/e7/092be7d0-7697-
220d-c000-97f366e723e4/mzi.anacpwuj.jpg/30x30bb.jpg", "artworkUrl60":"https://is2-ssl.mzstatic.com/image/thumb/Music114/v4/09/2b/e7/092be7d0-7697-
220d-c000-97f366e723e4/mzi.anacpwuj.jpg/60x60bb.jpg", "artworkUrl100":"https://is2-ssl.mzstatic.com/image/thumb/Music114/v4/09/2b/e7/092be7d0-7697-
220d-c000-97f366e723e4/mzi.anacpwuj.jpg/100x100bb.jpg", "collectionPrice":9.99, "trackPrice":1.29, "releaseDate":"1994-08-01T07:00:00Z",
"collectionExplicitness":"explicit", "trackExplicitness":"notExplicit", "discCount":1, "discNumber":1, "trackCount":12, "trackNumber":1,
"trackTimeMillis":258267, "country":"USA", "currency":"USD", "primaryGenreName":"Metal", "isStreamable":true}]
}
```


Наповнення виглядає дещо незрозуміло, воно має свою структуру: JSON (англ. *JavaScript Object Notation*).

Це формат, який використовується для передачі даних між програмами.

Існує багато інших форматів, таких як XML, CSV, YAML, але JSON є одним з найпопулярніших форматів.

API

Цей текст містить інформацію з бази даних Apple про одну пісню Korn. Давайте використаємо пакет `requests`, щоб отримати цю інформацію і вивести її на екран:

```
import requests
import sys

if len(sys.argv) != 2:
    sys.exit('Введіть назву гурту як аргумент командного рядка!')

url = 'https://itunes.apple.com/search?entity=song&limit=1&term=' + sys.argv[1]
response = requests.get(url)
print(response.json())
```

```
{'resultCount': 1, 'results': [{'wrapperType': 'track', 'kind': 'song', 'artistId': 466532, 'collectionId': 423045626, 'trackId': 423045744,
'artistName': 'Korn', 'collectionName': 'Korn', 'trackName': 'Blind', 'collectionCensoredName': 'Korn', 'trackCensoredName': 'Blind', 'artistViewUrl':
'https://music.apple.com/us/artist/korn/466532?uo=4', 'collectionViewUrl': 'https://music.apple.com/us/album/blind/423045626?i=423045744&uo=4',
'trackViewUrl': 'https://music.apple.com/us/album/blind/423045626?i=423045744&uo=4', 'previewUrl': 'https://audio-ssl.itunes.apple.com/itunes-
assets/AudioPreview115/v4/ef/a4/dd/efa4dd64-e8e0-6c12-c88b-a3a996b24b12/mzaf_13798363446300996858.plus.aac.p.m4a', 'artworkUrl30': 'https://is1-
ssl.mzstatic.com/image/thumb/Music114/v4/09/2b/e7/092be7d0-7697-220d-c000-97f366e723e4/mzi.anacpwuj.jpg/30x30bb.jpg', 'artworkUrl60': 'https://is1-
ssl.mzstatic.com/image/thumb/Music114/v4/09/2b/e7/092be7d0-7697-220d-c000-97f366e723e4/mzi.anacpwuj.jpg/60x60bb.jpg', 'artworkUrl100': 'https://is1-
ssl.mzstatic.com/image/thumb/Music114/v4/09/2b/e7/092be7d0-7697-220d-c000-97f366e723e4/mzi.anacpwuj.jpg/100x100bb.jpg', 'collectionPrice': 9.99,
'trackPrice': 1.29, 'releaseDate': '1994-08-01T07:00:00Z', 'collectionExplicitness': 'explicit', 'trackExplicitness': 'notExplicit', 'discCount': 1,
'discNumber': 1, 'trackCount': 12, 'trackNumber': 1, 'trackTimeMillis': 258267, 'country': 'USA', 'currency': 'USD', 'primaryGenreName': 'Metal',
'isStreamable': True}]}
```

Ми отримали той самий відформатований текст, але він був стандартизований у вигляді словника Python: Apple API віддає нам JSON, а пакет `requests` перетворює його у словник Python.

API

Давайте відформатуємо вивід, щоб він був більш читабельним. Для цього використаємо бібліотеку `json`, яка вбудована в Python:

```python
import json
import requests
import sys

if len(sys.argv) != 2:
    sys.exit('Введіть назву гурту як аргумент командного рядка!')

url = 'https://itunes.apple.com/search?entity=song&limit=1&term=' + sys.argv[1]
response = requests.get(url)
print(json.dumps(response.json(), indent=4))
```


```
{
  "resultCount": 1,
  "results": [
    {
      "wrapperType": "track",
      "kind": "song",
      "artistId": 466532,
      "collectionId": 423045626,
      "trackId": 423045744,
      "artistName": "Korn",
      "collectionName": "Korn",
      "trackName": "Blind",
      "collectionCensoredName": "Korn",
      "trackCensoredName": "Blind",
      "artistViewUrl": "https://music.apple.com/us/artist/korn/466532?uo=4",
      "collectionViewUrl": "https://music.apple.com/us/album/blind/423045626?i=423045744&uo=4",
      "trackViewUrl": "https://music.apple.com/us/album/blind/423045626?i=423045744&uo=4",
      "previewUrl": "https://audio-ssl.itunes.apple.com/itunes-assets/AudioPreview115/v4/ef/a4/dd/efa4dd64-e8e0-6c12-c88b-
a3a996b24b12/mzaf_13798363446300996858.plus.aac.p.m4a",
      "artworkUrl30": "https://is1-ssl.mzstatic.com/image/thumb/Music114/v4/09/2b/e7/092be7d0-7697-220d-c000-
97f366e723e4/mzi.anacpwuj.jpg/30x30bb.jpg",
      "artworkUrl60": "https://is1-ssl.mzstatic.com/image/thumb/Music114/v4/09/2b/e7/092be7d0-7697-220d-c000-
97f366e723e4/mzi.anacpwuj.jpg/60x60bb.jpg",
      "artworkUrl100": "https://is1-ssl.mzstatic.com/image/thumb/Music114/v4/09/2b/e7/092be7d0-7697-220d-c000-
97f366e723e4/mzi.anacpwuj.jpg/100x100bb.jpg",
      "collectionPrice": 9.99,
      "trackPrice": 1.29,
      "releaseDate": "1994-08-01T07:00:00Z",
      "collectionExplicitness": "explicit",
```

API

Тепер ми отримали відформатований текст, який більш читабельний. Але що, якщо ми хочемо вивести лише назву пісні? Для цього нам потрібно звернутися до словника Python, який міститься у змінній `response.json()`. Цей словник містить ключ `results`, який містить список пісень. Цей список містить словник, який містить ключ `trackName`, який містить назву пісні. Пропоную модифікувати URL-адресу і вивести 10 треків (`limit=10`) гурту Korn:

```
 1  import json
 2  import requests
 3  import sys
 4
 5  if len(sys.argv) != 2:
 6      sys.exit('Введіть назву гурту як аргумент командного рядка!')
 7
 8  url = 'https://itunes.apple.com/search?entity=song&limit=10&term=' + sys.argv[1]
 9  response = requests.get(url)
10  data = response.json()
11
12  for result in data['results']:
13      print(result['trackName'])
```

```
Blind
Shoots and Ladders
Faget
Clown
Ball Tongue
Divine
Need To
Helmet in the Bush
Daddy
Fake
```

Примітка

Документація до пакету `json` доступна за посиланням docs.python.org/3/library/json.html.

ВЛАСНІ ПАКЕТИ

Ви самі можете створювати власні бібліотеки.

Досі ми писали всі наші функції в одному файлі. Хорошою практикою було б якось об’єднати код, який ви повторно використовуєте, і створити власний модуль або пакет Python. Ви можете тримати його локально на власному комп’ютері або хмарному сервері, або ж ви можете пройти через певні кроки, щоб зробити його безкоштовним і з відкритим вихідним кодом і викласти його, наприклад на PyPI, щоб інші також могли ним користуватися.

Створимо новий файл `saying.py`, який буде моїм власним модулем:

Terminal

```
code saying.py
```

ВЛАСНІ ПАКЕТИ

В цьому файлі я створю функцію `say_hello()`, яка буде виводити вітальне повідомлення на екран, функцію `say_goodbye()`, яка буде виводити прощальне повідомлення на екран. Щоб переконатися, що ці функції працюють належним чином, я поміщу їх у функцію `main()`:

```
 1  def main():
 2      say_hello('Гаррі')
 3      say_goodbye('Гаррі')
 4
 5  def say_hello(name):
 6      print(f'Привіт, {name}')
 7
 8  def say_goodbye(name):
 9      print(f'До побачення, {name}')
10
11  main()
```

```
Привіт, Гаррі
До побачення, Гаррі
```

ВЛАСНІ ПАКЕТИ

Тепер я хочу використовувати ці функції так, ніби я дійсно створив власну бібліотеку, який робить доступною весь його функціонал для будь кого. Для цього створимо новий файл `say.py`:

Terminal

```
code say.py
```

В цьому файлі я імпортую функції з модуля `saying` і використовую їх:

```
import sys
from saying import say_hello

if len(sys.argv) == 2:
    say_hello(sys.argv[1])
```

Виконаємо код з аргументом командного рядка:

Terminal

```
python say.py Гаррі
```

І отримаю наступний результат:

```
Привіт, Гаррі
До побачення, Гаррі
Привіт, Гаррі
```

ВЛАСНІ ПАКЕТИ

```
1  Привіт, Гаррі
2  До побачення, Гаррі
3  Привіт, Гаррі
```

Чому так відбувається? Хоч я все зробив згідно з нашою попередньою практикою, це не зовсім правильний спосіб виклику `main()`.

Нюанс в тому, що у файлі `saying.py` я викликаю функцію `main()` у самому кінці. І навіть коли я імпортую цей модуль у новий файл `say.py`, то функція `main()` все одно буде викликана.

Код `from saying import say_hello` просить Python знайти модуль `saying`, прочитати його, а потім імпортувати функцію `say_hello()`. На жаль, до того часу, як Python прочитає файл зверху вниз зліва направо, останній рядок коду викликає функцію `main()`, що призводить до **обов'язкового виклику** всього коду.

ВЛАСНІ ПАКЕТИ

Для виправлення ситуації необхідно замість виклику функції `main()` у кінці коду використовувати запис:

```
1  if __name__ == '__main__':
2      main()
```

Запис `__name__` - це спеціальна змінна Python, яка містить ім’я поточного модуля.

Якщо модуль виконується **як програма**, то ця змінна містить рядок `'__main__'`.

Якщо ж **модуль** імпортується, то ця змінна містить ім’я модуля.

Таким чином, ми можемо **перевірити**, чи виконується модуль як програма, і якщо так, то викликати функцію `main()`.

ВЛАСНІ ПАКЕТИ

Давайте подивимось. Спочатку виправимо нам модуль `saying.py`:

```python
def main():
    say_hello('Гаррі')
    say_goodbye('Гаррі')

def say_hello(name):
    print(f'Привіт, {name}')

def say_goodbye(name):
    print(f'До побачення, {name}')

if __name__ == '__main__':
    main()
```

Імпортуємо модуль `saying`:

```python
import sys
from saying import say_hello

if len(sys.argv) == 2:
    say_hello(sys.argv[1])
```

Виконаємо код з аргументом командного рядка:

Terminal

```
python say.py Гаррі
```

```
Привіт, Гаррі
```

ДЯКУЮ ЗА УВАГУ!

Матеріали курсу

ihor.miroshnychenko@kneu.ua

Data Mirosh

@ihormiroshnychenko

@aranaur

aranaur.rbind.io