ЕКЗЕКУТОРИ ТА ПУЛИ ПОТОКІВ

Питання 12.4.

Багатопоточні футури

- Додані в Python 3.2, реалізують концепцію ThreadPoolExecutor з Java
- Однією з найбільш обчислювально дорогих задач є запуск потоків. Можемо:
  - одноразово створити потік, а потім постійно годувати його новими задачами (тасками).
  - одноразово створити низку потоків, перш ніж делегувати їм численні задачі протягом усього їхнього життя.
- Об'єкти класу Executor з модуля concurrent.futures дають можливість виконувати багато різних викликів у багатопоточному стилі.
  - Можуть використовуватись напряму або за допомогою менеджерів контексту, виконуючи тривалі задачі за один прохід.
- `executor = ThreadPoolExecutor(max_workers=3)`
  - створюємо екземпляр ThreadPoolExecutor і передаємо максимальну кількість робочих потоків.
- Щоб надати змогу потокам (ниткам) з ThreadPoolExecutor щось зробити, можна викликати функцію submit(), яка приймає функцію як основний параметр:
  - `executor.submit(myFunction())`

```
from concurrent.futures import ThreadPoolExecutor
import threading
import random

def task():
    print("Executing our Task")
    result = 0
    i = 0
    for i in range(10):
        result = result + i
    print("I: {}".format(result))
    print("Task Executed {}".format(threading.current_thread()))

def main():
    executor = ThreadPoolExecutor(max_workers=3)
    task1 = executor.submit(task)
    task2 = executor.submit(task)

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

Створення ThreadPoolExecutor. Приклад 1

- досить змішаний вивід у результаті виконання обох наших завдань, а результат нашого обчислення буде надруковано в командному рядку.
  - Потім використовуємо функцію threading.current_thread(), щоб визначити, яка нитка виконала це завдання.
  - Два значення, задані як виходи, є різними потоками-демонами.

```
Executing our Task
I: 45
Executing our Task
I: 45
Task Executed <Thread(<concurrent.futures.thread.ThreadPoolExecutor object
at 0x102abf358>_1, started daemon 123145333858304)>
Task Executed <Thread(<concurrent.futures.thread.ThreadPoolExecutor object
at 0x102abf358>_0, started daemon 123145328603136)>
```

Приклад 2 (менеджер контексту)

```
from concurrent.futures import ThreadPoolExecutor

def task(n):
    print("Processing {}".format(n))

def main():
    print("Starting ThreadPoolExecutor")
    with ThreadPoolExecutor(max_workers=3) as executor:
        future = executor.submit(task, (2))
        future = executor.submit(task, (3))
        future = executor.submit(task, (4))
    print("All tasks complete")

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

- Інший популярний спосіб інстанціювання ThreadPoolExecutor - використовувати його як менеджер контексту.
  - **with ThreadPoolExecutor(max_workers=3) as executor:**
  - робить ту ж саму роботу, але синтаксично виглядає краще.

```
Starting ThreadPoolExecutor
Processing 2
Processing 3
Processing 4
All tasks complete
```

Відображення (Maps) та пули потоків

```
from concurrent.futures import ThreadPoolExecutor
from concurrent.futures import as_completed

values = [2,3,4,5,6,7,8]

def multiplyByTwo(n):
    return 2 * n

def main():
    with ThreadPoolExecutor(max_workers=3) as executor:
        results = executor.map(multiplyByTwo, values)

    for result in results:
        print(result)

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

Можемо відобразити всі елементи ітератора у функцію і представити їх як незалежні завдання для ThreadPoolExecutor

- **`results = executor.map(multiplyByTwo, values)`**

Скорочується багатослівний синтаксис:

- **`for value in values:`**

  `executor.submit(multiplyByTwo, (value))`

```
$ python3.6 03_threadPoolMap.py
4
6
8
10
12
14
16
```

```
import time
import random
from concurrent.futures import ThreadPoolExecutor

def someTask(n):
    print("Executing Task {}".format(n))
    time.sleep(n)
    print("Task {} Finished Executing".format(n))

def main():
    with ThreadPoolExecutor(max_workers=2) as executor:
        task1 = executor.submit(someTask, (1))
        task2 = executor.submit(someTask, (2))
        executor.shutdown(wait=True)
        task3 = executor.submit(someTask, (3))
        task4 = executor.submit(someTask, (4))

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

Завершення роботи об'єктів-екзекуторів

- Мається на увазі неможливість прийому ним нових завдань.
  - Уже розпочаті завдання виконуються до свого завершення.
  - Визначимо функцію, яка «працюватиме» n секунд.
  - Сформуємо (submit) кілька завдань та викличемо метод shutdown().
  - Обробка винятку в коді не передбачена.

```
Executing Task 1
Executing Task 2
Task 1 Finished Executing
Task 2 Finished Executing
Traceback (most recent call last):
  File "C:/Users/Lenovo/PycharmProjects/execShutdown/execShutdown.py", line 19, in <module>
    main()
  File "C:/Users/Lenovo/PycharmProjects/execShutdown/execShutdown.py", line 15, in main
    task3 = executor.submit(someTask, (3))
  File "C:\Users\Lenovo\AppData\Local\Programs\Python\Python37-32\lib\concurrent\futures\thread.py", line 151, in subm
    raise RuntimeError('cannot schedule new futures after shutdown')
RuntimeError: cannot schedule new futures after shutdown
```

Об'єкти Future (футури) - об'єкти, яким у майбутньому буде надано значення

- Методи класу:
  - ***result():*** надає будь-які значення, що надходять від футури.

    ```
    futureObj.result(timeout = None)
    ```

    Якщо футура не встигає виконатись за вказаний період часу, виникає timeout-помилка.
  - ***add_done_callback():*** задає callback-функцію, яка буде виконуватись після завершення футури. Не потрібно відстежувати стан футури

    ```
    futureObj.add_done_callback(fn)
    ```
  - ***running():*** перевіряє, чи запущена футура в даний момент, поверне True або False

    ```
    futureObj.running()
    ```
  - ***cancel():*** намагається відмінити роботу футури, спрацьовує лише до його завершення:

    ```
    futureObj.cancel()
    ```
  - ***exception():*** дозволяє одержати винятки, які генеруватимуть футуру. Задавши час на виконання (у секундах), матимемо concurrent.futures.TimeoutError, якщо за цей період футура не завершить свою роботу

    ```
    futureObj.exception(timeout = None)
    ```
  - ***done():*** поверне True або False залежно від того, було успішно завершено роботу футури чи відмінено його виконання

    ```
    futureObj.done()
    ```

ДЯКУЮ ЗА УВАГУ!

Наступне питання: Екзекутори та пули потоків