Programowanie asynchroniczne i aplikacja do rejestracji

Michał Pędziwiatr
Igor Lechowski

Plan prezentacji

- Czym jest programowanie asynchroniczne?
- Do czego i po co je stosować?
- Jak to wygląda w Python (moduł asyncIO)
- Omówienie frameworka Django
- Jak działa nasza aplikacja?

Programowanie asynchroniczne

- W tradycyjnym programowaniu zadania wykonywane są po kolei, dana część kodu musi się wykonać żeby program mógł przejść dalej.
- W podejściu asynchronicznym taka kolejność nie musi zachowana – może zależeć od czynników zewnętrznych, spoza naszego programu.
- Programowanie asynchroniczne umożliwia osiągnięcie **współbieżności**.

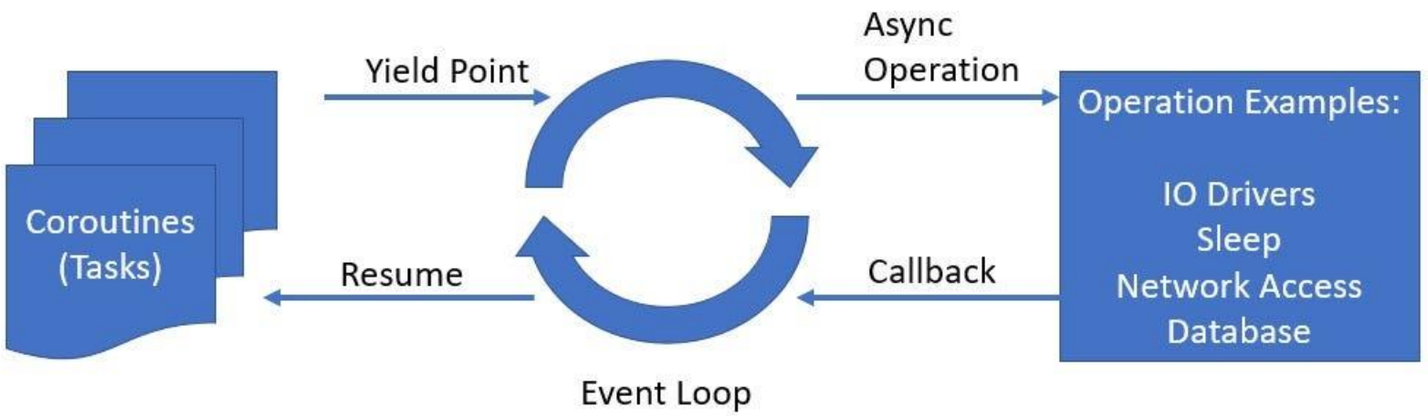
Współbieżność, czyli concurrency

- Współbieżność oznacza że kilka zadań jest wykonywanych “jednocześnie”.
- Osiągnięcie współbieżności pozwala na lepsze wykorzystywanie zasobów, poprawia wydajność i płynność działania aplikacji.
- Może być to osiągane na wiele sposobów, w tym wielowątkowość lub programowanie asynchroniczne.
- Sama asynchroniczność może być realizowana w wielu wątkach, lecz nie musi.

Działanie asynchroniczności na jednym wątku można omówić na przykładzie gry w szachy z wieloma przeciwnikami:

- przy podejściu synchronicznym wykonujesz gry po kolei, musisz skończyć grę z jednym przeciwnikiem aby zacząć grę z kolejnym
- w podejściu asynchronicznym po wykonaniu swojego ruchu zamiast czekać na ruch przeciwnika, idziesz wykonać ruch z kolejnym przeciwnikiem
- Zakładając więc że nie musisz się zbyt długo zastanawiać nad swoimi ruchami (jesteś arcymistrzem), a przeciwnik zastanawia się dość długo, przy 10 przeciwnikach czas potrzebny na wszystkie gry skróciłby się niemal 10-krotnie.

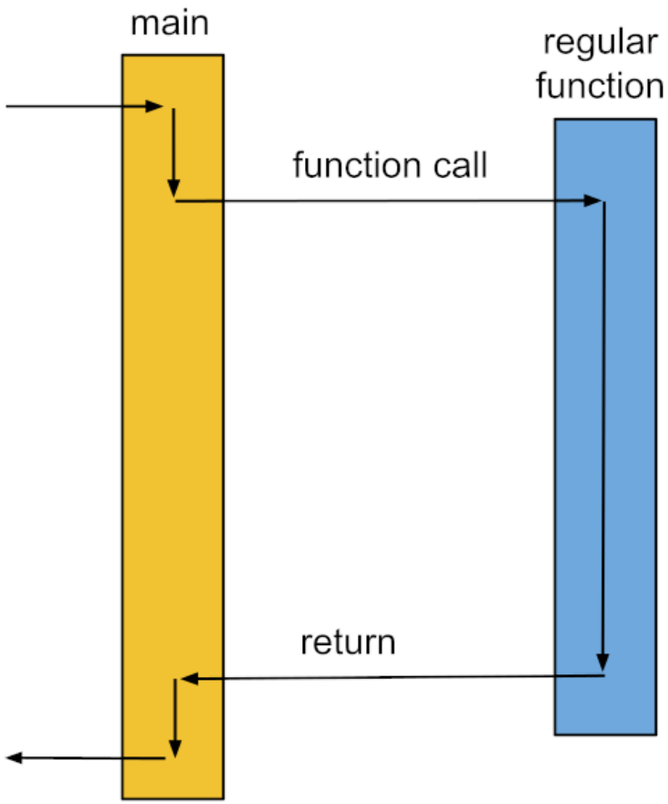
Event loop

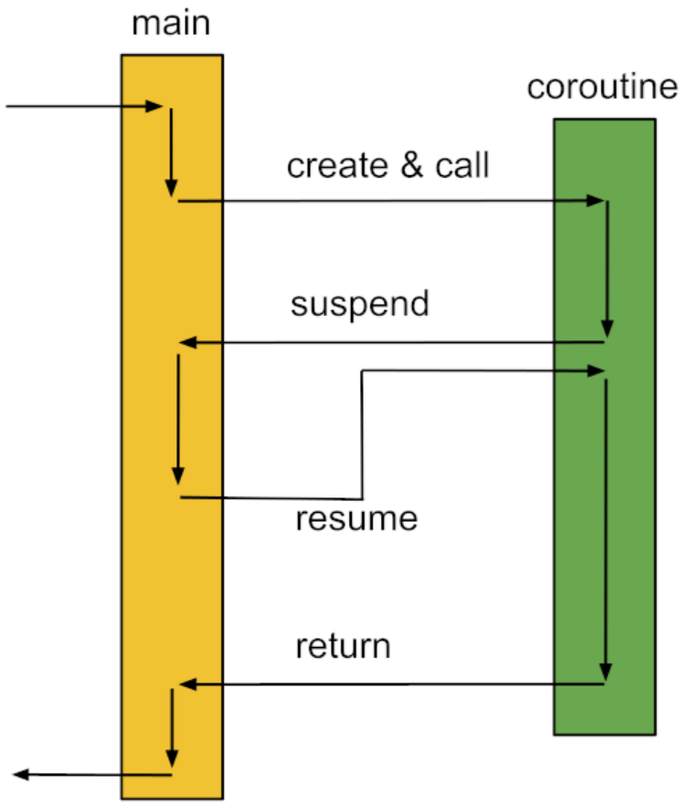


Wracając do programowania, dzięki asynchroniczności, podczas gdy jakaś funkcja czeka np. na pobranie danych z bazy danych, kod może dalej się wykonywać i robić inne rzeczy.

Zadania (**korutyny**) czekają kolejce. Natomiast wybieraniem z kolejki zajmuje się pętla zwana pętlą zdarzeń czyli eventloop.

Podejście klasyczne vs asynchroniczne





Gdzie użyć asynchroniczności?

Asynchroniczności używa się najczęściej w zadaniach limitowanych przez wyjście lub wejście (I/O bound)

Przykłady:

- czytanie z i zapis do plików czy bazy danych
- zapytania do serwera
- tworzenie responsywnego UI

asyncIO

- Jest to biblioteka w Python
- Umożliwia pisanie 1-wątkowego kodu asynchronicznego
- Pozwala na pracę z **korutynami**
- Bazuje na składni **async/await**
- Dokumentacja asyncIO:
  https://docs.python.org/3/library/asyncio-task.html

async i await

- Async służy do definiowania korutyny
- Await odpowiada za zawieszenie i wznawianie wykonywania tej korutyny

```python
import asyncio
import time


3 usages
async def count():
    print("One")
    await asyncio.sleep(1)
    print("Two")

1 usage
async def main():
    await asyncio.gather(count(), count(), count())

if __name__ == "__main__":
    s = time.perf_counter()
    asyncio.run(main())
    elapsed = time.perf_counter() - s
    print(f"{__file__} executed in {elapsed:0.2f} seconds.")
```

```
One
One
One
Two
Two
Two
D:\Dokumenty\Studia Teleinformatyka\JPWP\Przykłady\async_count.py executed in 1.00 seconds.
```

- asyncio.sleep zawiesza wykonanie korutyny
- asyncio.gather spowoduje, że obiekty zostaną uruchomione współbieżnie
- asyncio.run uruchamia korutyne main()

```python
import time

1 usage
def count():
    print("One")
    time.sleep(1)
    print("Two")

1 usage
def main():
    for _ in range(3):
        count()

if __name__ == "__main__":
    s = time.perf_counter()
    main()
    elapsed = time.perf_counter() - s
    print(f"{__file__} executed in {elapsed:0.2f} seconds.")
```

```
One
Two
One
Two
One
Two
D:\Dokumenty\Studia Teleinformatyka\JPWP\Przykłady\sync_count.py executed in 3.00 seconds.
```

- Kod wykonujący 3 takie same zadania, ale w wersji klasycznej (synchronicznej).
- Wykonuje się 3x dłużej.

Asynchroniczność w Django

W naszej aplikacji używamy frameworka Django.

Stosunkowo niedawno, bo wersji Django 5.0 (powstała 4.12.2023) wprowadzono asynchroniczne wersje funkcji do systemu logowania i rejestracji użytkowników:

- aauthenticate()
- alogin()
- alogout()
- auser()

Aplikacja do logowania

- Django models
- Django forms
- System uwierzytelniania w Django (Django.contrib.auth)
- Django AppConfig

Models

- Każdy model jest klasą w Pythonie, która jest podklasą django.db.models.Model
- Każdy atrybut modelu reprezentuje pole bazy danych
- Dzięki temu, Django daje automatycznie wygenerowane database-access API
- Przykład użycia:

```python
from django.db import models


class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

Forms

- Służy do tworzenia formularzy na stronie internetowej
- Są 2 metody HTTP używane przez forms: POST i GET
- POST służy do wprowadzania zmian, np. w bazie danych, koduje przesyłane dane więc jest bezpieczny do rejestracji i logowania
- GET zwykle jest używany do wyszukiwania czegoś

Przykład użycia Forms

```python
class LoginForm(forms.Form):
    username = forms.CharField(max_length=64, label="Nazwa użytkownika")
    password = forms.CharField(
        max_length=128, widget=forms.PasswordInput, label="Hasło"
    )

    Michał Pędziwiatr
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        for visible in self.visible_fields():
            visible.field.widget.attrs["class"] = "form-control"
```

Uwierzytelnianie użytkowników

- Służy do tego funkcja authenticate()
- Sprawdza ona poświadczenia w bazie danych i zwraca obiekt użytkownika, jeśli są prawidłowe.
- Jeśli są nieprawidłowe lub backend zwróci PermissionDenied, funkcja authenticate() zwraca None.
- Musi być poprzedzona funkcją login(), aby zalogować użytkownika.

Widoki w Django można zabezpieczyć przed nieautoryzowanym dostępem dzięki dekoratorom:

- @login_required
- @user_passes_test

```python
@login_required
@transaction.atomic
def profile(request):
    if request.method == "POST":
        user_form = UserForm(request.POST, instance=request.user)
        profile_form = ProfileForm(request.POST, instance=request.user.profile)
        location_form = LocationForm(request.POST, instance=request.user.profile)
        sensitive_form = SensitiveForm(request.POST, instance=request.user.profile)
        if (
```

- Trzeba taki dekorator umieścić przed fragmentem kodu, który chcemy zabezpieczyć.