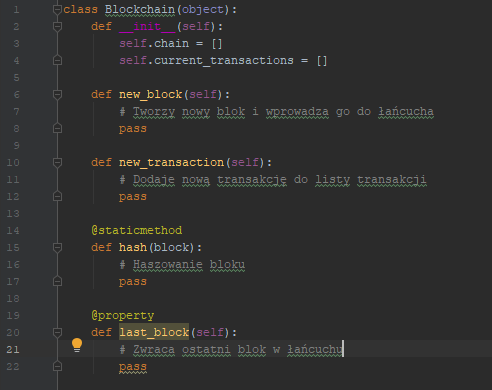
**Technologia blockchain**

**Python**

**Krok 1: Tworzenie własnego Blockchain**

**Szkielet Blockchain:**

Stwarzam klasę blockchain, której konstruktor utworzy początkowy pusty arkusz (do przechowywania blockchain), a drugi do przechowywania transakcji. Oto rysunek naszej klasy:

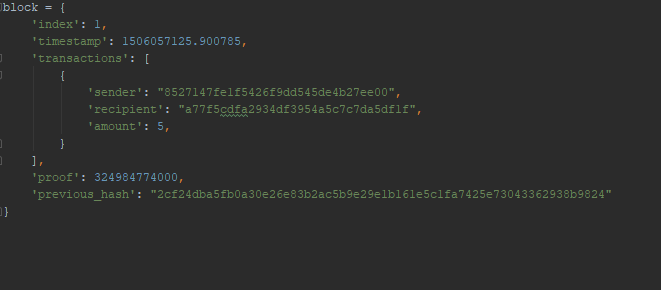


Klasa Blockchain jest odpowiedzialna za zarządzanie łańcuchem. Będzie przechowywać transakcje, a także będzie miała kilka pomocniczych metod wprowadzania nowych bloków do łańcucha.

**Jak wygląda blok?**

Każdy blok zawiera indeks, (czas uniksowy), listę transakcji, dowód i skrót poprzedniego bloku.

Oto przykład, jak wygląda jeden blok:



Każdy nowy blok zawiera w sobie hash poprzedniego bloku. Jest to fundamentalnie ważne, ponieważ zapewnia niezmienność łańcucha bloków: jeśli atakujący włamie się do poprzedniego bloku, wówczas wszystkie inne bloki będą zawierały niepoprawne hashe.

## Wniesienie tranzakcji w blok

## Potrzebujemy sposobu na wprowadzenie transakcji do bloku. Metoda new\_transaction () jest za to odpowiedzialna i jest dość prosta:

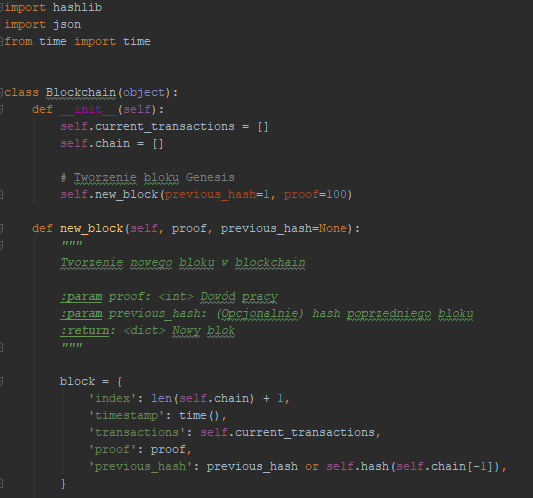
## 

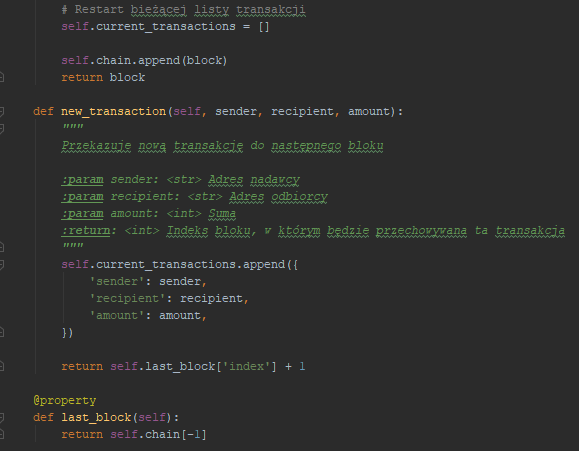
Gdy new\_transaction () wprowadzi transakcję na listę, zwróci indeks bloku, w którym transakcja zostanie wprowadzona - a mianowicie następna. W przyszłości będzie to przydatne dla użytkownika wysyłającego transakcję.

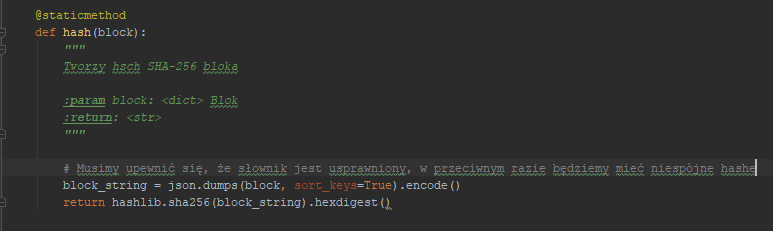
**Tworzenie nowych bloków**

Po otrzymaniu kopii łańcucha bloków trzeba zasadzić w nim blok genezy - pierwszy blok bez poprzedników. Musze również zrobić „Proof” w naszym bloku genezowym, który jest wynikiem wydobycia (dowód wykonanej pracy).

Oprócz tworzenia bloku genezy w konstruktorze, będziemy także wdrażać metody dla new\_block (), new\_transaction () i hash ():







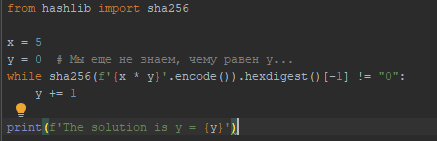
**Zrozumienie potwierdzenia pracy**

Algorytm Proof of Work (PoW) to sposób tworzenia lub wydobywania nowych bloków w Blockchain. Celem PoW jest znalezienie numeru, który rozwiązuje problem. Liczba powinna być taka, że ​​trudno ją znaleźć, ale łatwo jest ją potwierdzić (mówiąc o obliczeniach) przez każdego w Internecie. To jest główne zadanie algorytmu.

**Wyjaśnie to prostym przykładzie:**

Powiedzmy, że hash danej liczby x razy inna liczba musi kończyć się na zero. Zatem hash (x \* y) = ac23dc ... 0. W tym uproszczonym przykładzie wyobrażamy sobie że x = 5.

W Pythonie to będzie wyglądać tak:



Rozwiązaniem jest tutaj: y = 21, ponieważ utworzony hash kończy się na zero:



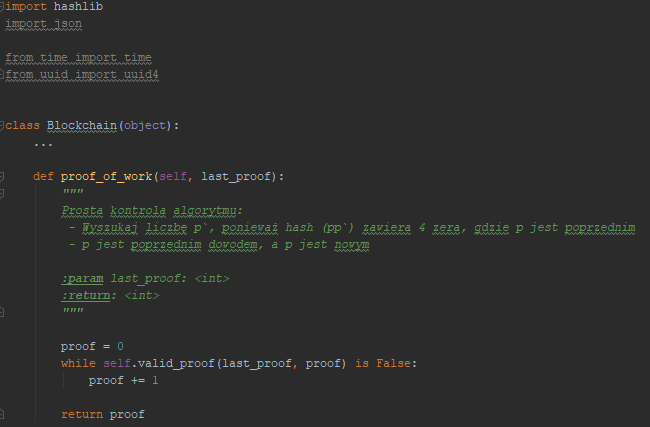


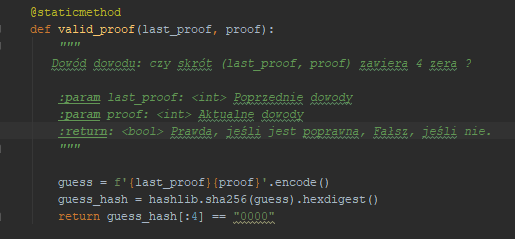
W bitcoin taki algorytm nazywa się Hashcash. I nie różni się szczególnie od mego przykładu. Jest to algorytm, który pokolenie wydobywaczy próbuje rozwiązać, aby utworzyć nowy blok. Zasadniczo złożoność zależy od liczby znaków branych pod uwagę w wierszu. Wydobywacze są dobrze wynagradzani za rozwiązanie problemu otrzymania coina w transakcji.

**Realizacja PoW**

Implementuje podobny algorytm dla swego blockchaina. Reguła będzie podobna do podanej wcześniej:

„Znajść liczbę „p”, która jest mieszana z poprzednio utworzonym rozwiązaniem blokowym z hashem zawierającym 4 zera”





Aby dostosować złożoność algorytmu, możemy zmienić liczbę zer. W naszym przypadku wystarczy 4.

Klasa jest już prawie gotowa, więc można zacząć z nią wchodzić w interakcje za pośrednictwem żądań HTTP.

**Krok 2: Blockchain jako interfejs API**

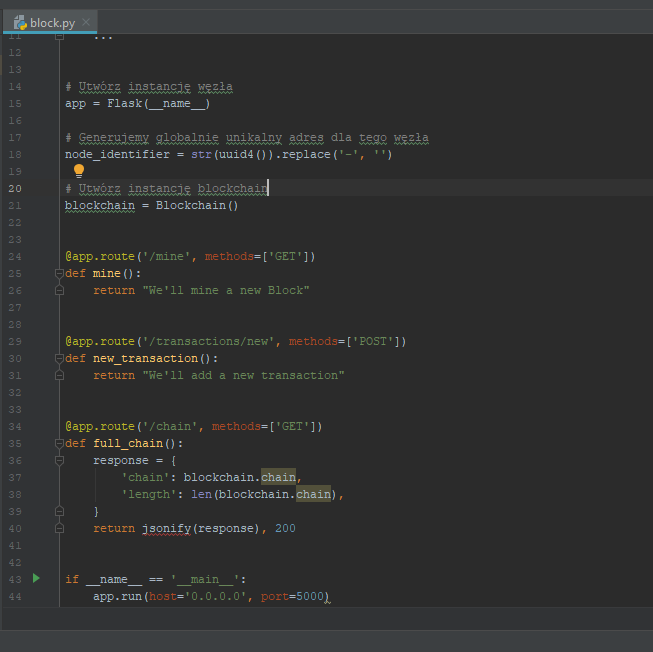
Tutaj używam frameworka o nazwie Flask. Jest to struktura makro-framework, która znacznie ułatwia mapowanie punktów końcowych na funkcje Pythona. Pozwala nam to na interakcję z naszym blockchainem w Internecie za pomocą żądań HTTP.

Tworze trzy metody:

* **/transactions/new** aby utworzyć nową transakcję w bloku;
* /**mine,** wskazać serwerowi, że konieczne jest wydobycie nowego bloku;
* **/ chain,** aby zwrócić cały łańcuch bloków

**Konfiguracja Flack:**

„serwer” utworzy pojedynczy węzeł w sieci blockchain:



Krótkie wyjaśnienie:

Wiersz 15: Tworzenie instancji węzła.

Wiersz 18: Tworzenie losowej nazwy dla węzła;

Wiersz 21: Tworzenie instancji klasy Blockchain;

Linie 24–26: Tworzenie punktu końcowego / **mine**, która jest żądaniem GET;

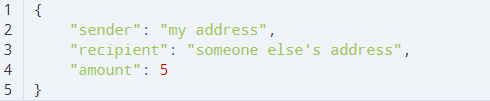
Linie 28-30: Tworzenie punktu końcowego /**transactions/new**, który jest żądaniem POST, ponieważ tam wysyłają się dane;

Linie 32-38: Utwórz punkt końcowy / **chain**, który zwraca cały łańcuch bloków;

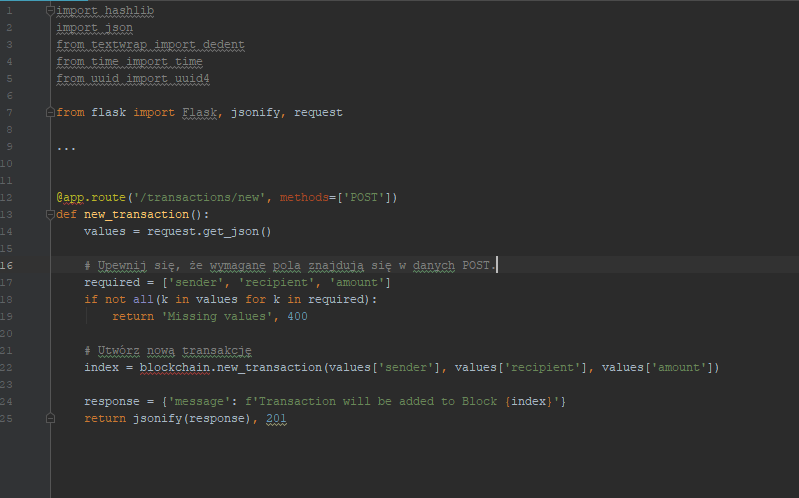
Linie 40–41: Uruchamia serwer na porcie: 5000.

**Punkt końcowy transakcji**

Tak powinno wyglądać żądanie transakcji. Oto, co użytkownik wysyła na serwer:



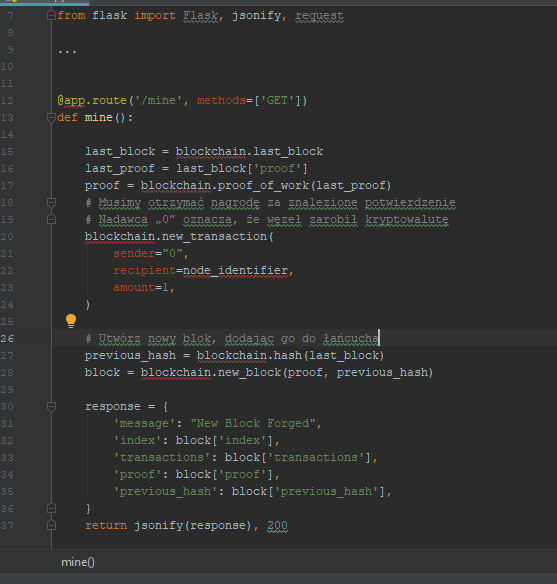
Funkcja do dokonywania transakcji:



**Końcowy punkt wydobywania**

Potrzeba zrobić:

* Policzyć PoW
* Nagroda wydobywacza (nas), dodając transakcję dającą 1 coin;
* Skleić następny blok, wprowadzając go do łańcucha.

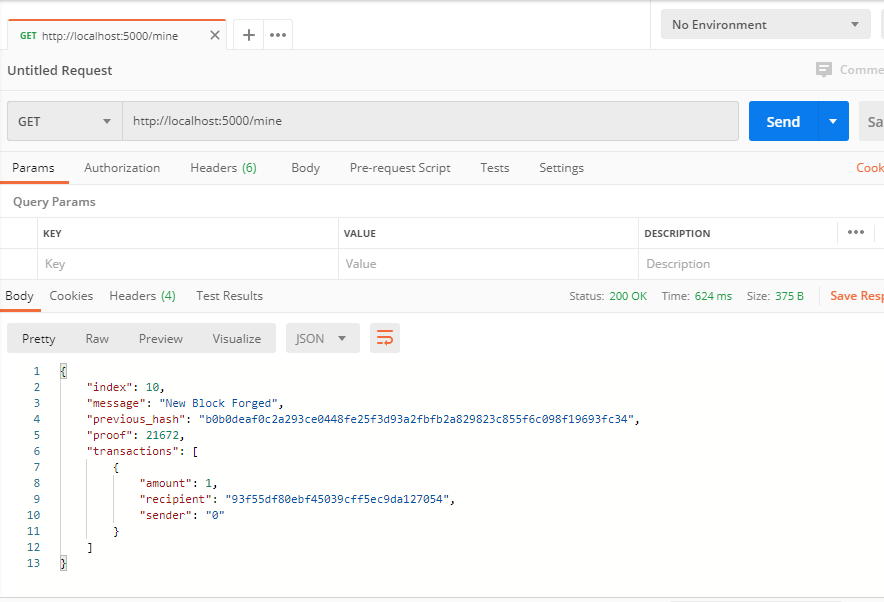


**Krok 3: interakcja z naszym blockchainem**

Uruchamiamy serwer:



Probuje wydobyć blok, tworząc żądanie GET do węzła <http://localhost:5000/mine>:



Sprawdzanie czy cały łańcuch, wykonujuje zapytanie do węzła **http://localhost:5000/chain:**