__init__, argumenty linii, shutil

Paweł Gliwny

Wykonywanie skryptów Pythona z linii komend

- Skrypty Pythona można uruchamiać bezpośrednio z linii komend.
- Instrukcje na najwyższym poziomie w skrypcie Pythona wykonują się zarówno przy wywołaniu skryptu, jak i przy jego imporcie.
- Przykład bezpośredniego wywołania funkcji na najwyższym poziomie skryptu:
- Uruchomienie `python always_say_it.py` lub zaimportowanie skryptu w innym module spowoduje wykonanie `say_it()`.

```
def say_it():
    greeting = 'Cześć'
    target = 'Joe'
    message = f'{greeting} {target}'
    print(message)
say_it()
```

Rola `__name__` w skryptach Pythona

- Zmienna `__name__` w Pythonie odgrywa specjalną rolę.
- Jest ustawiona na `"__main__"` gdy skrypt jest uruchamiany bezpośrednio.
- Gdy skrypt jest importowany jako moduł, `__name__` równa się nazwie modułu.
- Użyj `__name__`, aby kontrolować, kiedy funkcje skryptu są wykonywane:

```
def say_it():
    ...

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

Zapewnia to, że `say_it()` uruchamia się tylko wtedy, gdy skrypt jest wykonany bezpośrednio, a nie przy imporcie.

Co to jest `sys.argv`?

- `sys.argv` jest listą w Pythonie, która zawiera argumenty wiersza poleceń przekazane do skryptu.
- Jest dostępny poprzez moduł `sys`, który należy zaimportować przed użyciem.
- Pierwszym elementem tej listy jest zawsze ścieżka do uruchamiania skryptu.
- Kolejne elementy to dodatkowe argumenty przekazane do skryptu.

```
import sys

print(sys.argv)
```

Uruchomienie skryptu: `python script.py arg1 arg2`
Wynik: `['script.py', 'arg1', 'arg2']`

argparse

`argparse` to moduł w standardowej bibliotece Pythona, używany do pisania przyjaznych dla użytkownika interfejsów linii poleceń.

Automatyczne generuje wiadomości pomocy za pomocą flagi - - help

parser można postrzegać jako kontener, który przechowuje informacje o tym, jakich argumentów oczekuje program i jak powinny być przetwarzane.

argparse inicjalizacja

Inicjalizacja Parsera**:** pierwszym krokiem w użyciu **`argparse`** jest utworzenie obiektu **`ArgumentParser`**.

- Ten obiekt będzie obsługiwał wszystkie argumenty linii poleceń, które są przekazywane do programu.
- Parametr `description` daje krótki opis tego, co robi program i jest wyświetlany, gdy wywoływana jest wiadomość pomocy

```
parser = argparse.ArgumentParser(description='Opis twojej aplikacji')
```

Dodawanie argumentów

Metoda add_argument: kluczowym elementem w pracy z `argparse` jest metoda `add_argument`, która pozwala definiować nowe argumenty.

- **Definiowanie Argumentów:** parser musi wiedzieć, jakich argumentów powinien się spodziewać.
  a. Te argumenty są definiowane przez metodę `add_argument`.
  b. Każdy argument może być argumentem pozycyjnym (obowiązkowym) lub opcjonalnym (zaczyna się zwykłe od `-` lub `--`).

```
parser.add_argument('message',
                    help='Komunikat do wyświetlenia')
parser.add_argument('--twice', '-t',
                    help='Zrób to dwukrotnie',
```

Parsowanie i wykorzystanie argumentów

- **Parsowanie Argumentów:** `parse_args()` służy do przetwarzania argumentów linii poleceń.

  args = parser.parse_args()

  x=args.x

Wykorzystanie `nargs='+'` w Argparse

- **Definicja `nargs='+'`** jest opcją w metodzie **`add_argument()`** biblioteki **`argparse`**, która pozwala na przyjęcie jednego lub więcej wartości dla danego argumentu. Jest to przydatne, gdy oczekujesz, że użytkownik przekaże serię wartości, zamiast tylko jednej.
  - **`parser.add_argument('filenames', nargs='+', help='List of files to process')`**
  - Używając **`nargs='+'`**, argument `filenames` może teraz przyjąć wiele wartości, np. **`script.py file1.txt file2.txt file3.txt`**.

`shutil`

`shutil` to moduł używany do operacji na plikach na wysokim poziomie.

Zapewnia funkcje do efektywnego kopiowania, przenoszenia, zmieniania nazw i usuwania plików oraz katalogów.

- Kopiowanie plików i katalogów: Umożliwia kopiowanie plików i katalogów, w tym opcję kopiowania całych drzew katalogów.
- Przenoszenie plików i katalogów: Ułatwia przenoszenie plików i katalogów z jednego miejsca do drugiego.
- Kopiowanie i usuwanie drzew katalogów: Pozwala na kopiowanie całych drzew katalogów oraz ich późniejsze usuwanie.

`shutil.copytree` vs `shutil.copy`

- **`shutil.copytree(src, dst)`**: Ta funkcja służy do kopiowania całego drzewa katalogów. Kopiuje wszystkie foldery, podfoldery i pliki z katalogu źródłowego (**`src`**) do katalogu docelowego (**`dst`**). Jest to przydatne, gdy chcesz zrobić pełną kopię jakiegoś katalogu wraz z jego zawartością.
- **`shutil.copy(src, dst)`**: Ta funkcja służy do kopiowania pojedynczego pliku. Kopiuje plik ze ścieżki źródłowej (**`src`**) do ścieżki docelowej (**`dst`**). Jeśli **`dst`** jest katalogiem, plik jest kopiowany do tego katalogu, zachowując swoją nazwę.

Obie te funkcje są niezwykle przydatne w różnych scenariuszach zarządzania plikami i katalogami w Pythonie.