

Python: podstawy programowania



Dzień 4 | v. 1.1.1

Protokół HTTP

HTTP

HTTP

HTTP (**H**yper**T**ext **T**ransfer **P**rotocol) to protokół przekazywania danych między komputerami.

Jeden z nich jest **serwerem**, czyli komputerem, na którym wykonywane jest oprogramowanie back-end i generowane są strony.

Drugi z komputerów jest **klientem**, czyli maszyną, która generuje żądania do serwera, odbiera je i wyświetla na ekranie (w przeglądarce) wynik działania aplikacji.

Jak to działa?

- **Klient** (np. przeglądarka) nawiązuje połączenie z serwerem,
- **klient** wysyła żądanie (**HTTP Request**), a **serwer** je odbiera,
- **serwer** przetwarza żądanie i generuje odpowiedź (**HTTP Response**),
- **serwer** wysyła odpowiedź, a **klient** ją odbiera,
- połączenie jest zamykane,
- **klient** przetwarza odpowiedź (np. wyświetla stronę na ekranie).

Metody HTTP

Metody HTTP

Protokół HTTP przenosi dane między komputerami używając różnych metod:

- **GET,**
- **POST,**
- **HEAD,**
- **PUT,**
- **DELETE,**
- **OPTIONS,**
- **TRACE,**
- **CONNECT,**
- **PATCH.**

Najczęściej używanymi metodami HTTP są metody **GET** oraz **POST**. Są np. powszechnie używane w aplikacjach internetowych, jako metody na przekazywanie danych między przeglądarką, a serwerem.

Metoda GET

GET

Dane przekazywane metodą **GET** są umieszczane w adresie URL:

`https://www.google.pl/search?q=metody+http&oq=metody+http`

Znak zapytania rozdziela adres od danych, po czym następuje seria par klucz – wartość, rozdzielanych znakiem ampersand (&).

Parametry żądania przekazywane tą metodą charakteryzują się:

- Można je cache'ować: oznacza to, że mogą być zapamiętane (razem z adresem),
- Pozostają w historii przeglądarki,
- Mogą być dodane do zakładek,
- Mają ograniczenie długości (zależne od serwera),
- Powinny być używane tylko do pobierania danych (z przyczyn bezpieczeństwa),
- Nie powinno się ich używać do pracy z danymi wrażliwymi.

Przesyłanie danych za pomocą GET

Jest to najprostszy sposób przesyłania danych do aplikacji webowej.

Jest to metoda w której wartości naszych zmiennych są ukryte w adresie URL.

Na końcu adresu dodajemy znak ?. Za nim wpisujemy zmienne które chcemy przekazać w formacie:

<nazwa zmiennej>=<wartość zmiennej>

Kolejne zmienne łączymy ze sobą znakiem &

Link który przesyła do **page2**

``

Znak & rozdziela przesyłane zmienne

Znak ? rozpoczyna informacje o przesyłanych zmiennych

Przesyłamy zmienną **foo** o wartości **32**

Przesyłamy zmienną **bar** o wartości **Some_text**

Metoda POST

POST

Dane przekazywane metodą **POST** są umieszczane w nagłówku żądania HTTP:

```
POST /test/pass-params HTTP/1.1
Host: our-hot-server.com
name1=value1&name2=value2
```

Podobnie, jak w poprzednim przypadku, dane przekazywane są przy pomocy serii par klucz – wartość, rozdzielanych znakiem ampersand (&).

Parametry żądania przekazywane tą metodą charakteryzują się:

- Nie są nigdy cache'owane,
- Nie pozostają w historii przeglądarki,
- Nie mogą być dodane do zakładek,
- Nie mają ograniczenia na długość danych.
- Mogą być używane do modyfikowania stanu aplikacji,
- Mogą być używane do pracy z danymi wrażliwymi.

Request & response

Nagłówki przykładowego żądania (request):

```
GET / HTTP/1.1
Host: coderslab.pl
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/53.0.2785.143 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;
q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: pl-PL,pl;q=0.8,en-
US;q=0.6,en;q=0.4
Cookie: __ym_uid=1464703637313577431;
viewed_cookie_policy=yes; __unam=6c76c78-
155368f4b6a-28a6bd26-1;
_ga=GA1.2.760385936.1464703637
```

Nagłówki odpowiedzi (response):

```
HTTP/1.1 200 OK
Server: nginx
Date: Thu, 13 Oct 2016 11:04:49 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
Link: <http://coderslab.pl/wp-json/>;
rel="https://api.w.org/"
Link: <http://coderslab.pl/>; rel=shortlink
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Content-Encoding: gzip
```




Flask

Flask

- Flask to prosty framework do tworzenia serwerów WWW,
- Jest często wykorzystywany komercyjnie z powodu prostoty i wydajności

Instalacja:

```
pip install Flask
```



Najprostszy serwer

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

Uruchamiamy go, jak każdy skrypt Pythona:

```
python server.py
```

Strona dostępna pod adresem:
<http://localhost:5000>

Najprostszy serwer

```
from flask import Flask  
app = Flask(__name__)
```

Importujemy Flask

Definiujemy, że aktualny plik będzie serwerem Flask. Więcej o magicznej właściwości `__name__` mówić będziemy przy okazji zaawansowanego Pythona.

```
@app.route("/hello")
```

Dekorator definiuje nam, że po wpisaniu adresu `http://<adres_serwera>/hello`, ma uruchomić poniższą funkcję.

```
def hello():  
    return "Hello World!"
```

Definiujemy funkcję `hello()`, która ma za zadanie zwrócić (i wyświetlić na ekranie) napis „Hello World!”

```
app.run()
```

Uruchamiamy serwer

Parametry

Jeśli chcemy przekazać jakiś parametr do URL-a, np. imię, żeby wyświetlić je na stronie, wystarczy zdefiniować je w dekoratorze `@app.route()` podając jego nazwę w znakach `<>`:

```
@app.route("/path/<parameter>")
```

```
@app.route("/hello/<name>")
def hello(name):
    return "Hello " + name + "!"
```

Parametry

Możemy wyspecyfikować jakiego typu danych oczekujemy:

```
@app.route("/path/<type:parameter>")
```

Niektóre dopuszczalne typy:

- **string**: łańcuch tekstowy,
- **int**: liczba całkowita,
- **float**: liczba zmiennoprzecinkowa,
- **path**: w środku parametru dopuszczalne są znaki „/”.

```
@app.route("/hello/<string:name>")  
def hello(name):  
    return "Hello " + name + "!"
```

```
@app.route("/age/<int:num>")  
def age(num):  
    return "You are " + num + " y.o."
```


Metody HTTP we Flask

Standardową metodą, którą używamy podczas wywoływania żądań, jest GET.

Oznacza to, że do tej pory wszystkie funkcje serwera Flask wykonywane były przy użyciu metody GET.

Możemy jednak zażyczyć sobie, żeby Flask przyjmował dane metodą POST.

```
from flask import request

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        # obsłuż dane logowania
    else:
        # wyświetl formularz logowania
```

Metody HTTP we Flask

```
from flask import request
```

```
@app.route('/login', methods=['GET', 'POST'])  
def login():
```

```
    if request.method == 'POST':  
        # obsłuż dane logowania  
    else:  
        # wyświetl formularz logowania
```

Parameter **methods** dekoratora, definiuje nam, jakimi metodami HTTP będzie można się dostać do funkcji.

Obiekt **request** przechowuje wszystkie dane żądania HTTP. Potraktuj **request.method** jak zmienną, która przechowuje nazwę metody HTTP, której użyliśmy do wywołania tego adresu.

Formularze

- Formularze dają większe możliwości niż **GET** w przekazywaniu informacji między stronami.
 - Element **HTML** odpowiedzialny za tworzenie formularza (**<form>**) oraz elementy tworzące poszczególne jego pola (**<input>**, **<select>**, **<textarea>** itd.) już poznaliśmy na kursie.
 - Formularze mogą przekazywać dane metodą **GET** lub **POST**.
- Metoda **GET** przekazuje wartości pól formularza za pomocą adresu **URL** – są one widoczne w pasku adresu przeglądarki (jest to mniej bezpieczna opcja).
 - Metoda **POST** do przekazywania parametrów wykorzystuje nagłówek zapytania, jej parametry nie są widoczne w adresie **URL**. Jest to bezpieczniejsza i częściej używana opcja do przekazywania danych wysłanych formularzem.

Formularze - tworzenie

- Teraz zajmiemy się odbieraniem elementów i sprawdzaniem wartości, które użytkownik wpisał do formularza.
- Adres skryptu, do którego mają być wysłane dane z formularza, definiujemy w atrybucie **action**.
- Metodę wysłania parametrów definiujemy w atrybucie **method**.

```
<form action="/" method="POST">  
...  
</form>
```

Do jakiej strony
prześle nas formularz

Jaką metodą zostaną
przesłane dane (mamy do
wyboru **POST** albo **GET**)

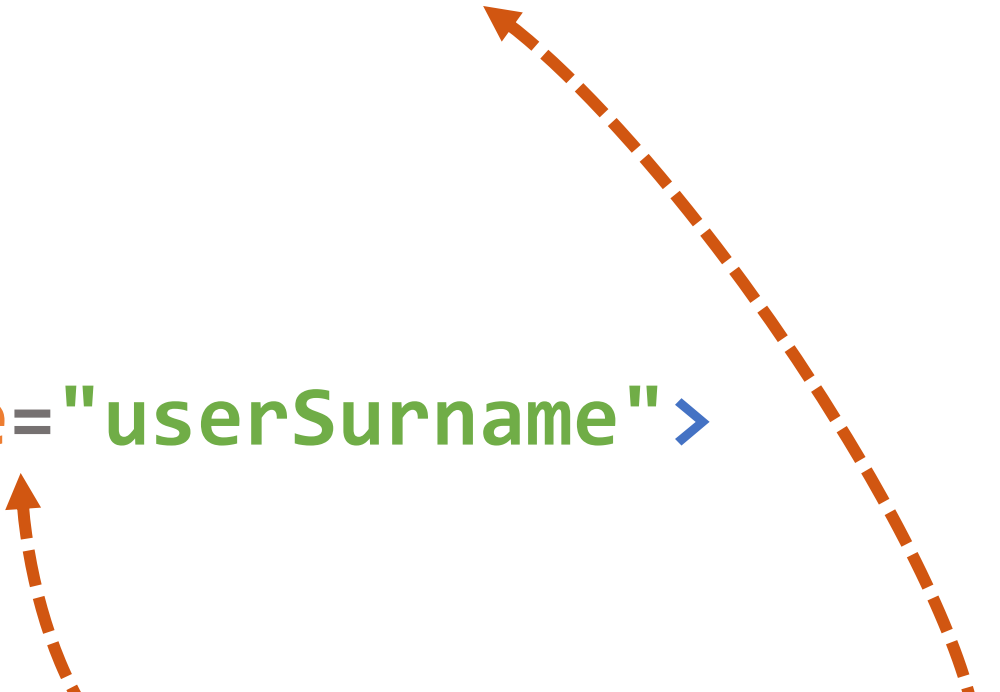
Formularze – pola

Następnie musimy wpisać do formularza pola w które użytkownik będzie mógł wpisywać dane.

Najważniejszy z poziomu back endu jest atrybut **name** który musi się znaleźć w każdym z inputów.

Określa on pod jakim kluczem będzie znajdowała się wartość z danego inputa po przesłaniu danych.

```
<label>  
  Imię:  
  <input type="text" name="userName">  
</label>  
  
<label>  
  Nazwisko:  
  <input type="text" name="userSurname">  
</label>
```



Ważne jest żeby nigdy nie zapomnieć o atrybucie **name**

Formularze – pole submit

Na końcu formularze musi znajdować się specjalny **input** o typie **submit**. Dzięki niemu użytkownik będzie mógł wysłać przygotowany formularz.

```
<label>  
    <input type="submit">Wyślij</input>  
</label>
```


Odbieranie parametrów

Odbieranie danych

Wszystkie dane które przesyłamy zarówno metodą GET, jak i POST znajdują się w obiekcie **request** jako słownik **form**. Dostać się do nich możemy używając konstrukcji **request.form[nazwa-klucza]**. Wartość zmiennej znajduje się pod takim samym kluczem (stringiem!) jak atrybut **name** danego pola formularza.

Jeżeli dwa pola będą miały taką samą wartość atrybutu **name**, to będziemy w stanie odebrać dane tylko z jednego z nich.

Odbieranie parametrów

Fragment HTML-a z formularzem

```
<label>
    Imię:
    <input type="text" name="userName">
</label>

<label>
    Nazwisko:
    <input type="text" name="userSurname">
</label>
```

Fragment kodu aplikacji z użyciem Flask:

```
def get_from_form():
    if request.method == "POST":
        user_name =
            request.form["userName"]
        userSurname =
            request.form["userSurname"]

    # zwróć uwagę na klucze słownika
    # request.form: są to nazwy pól
    # (atrybut name) z formularza
    # po lewej stronie.
```