



*FLASK*

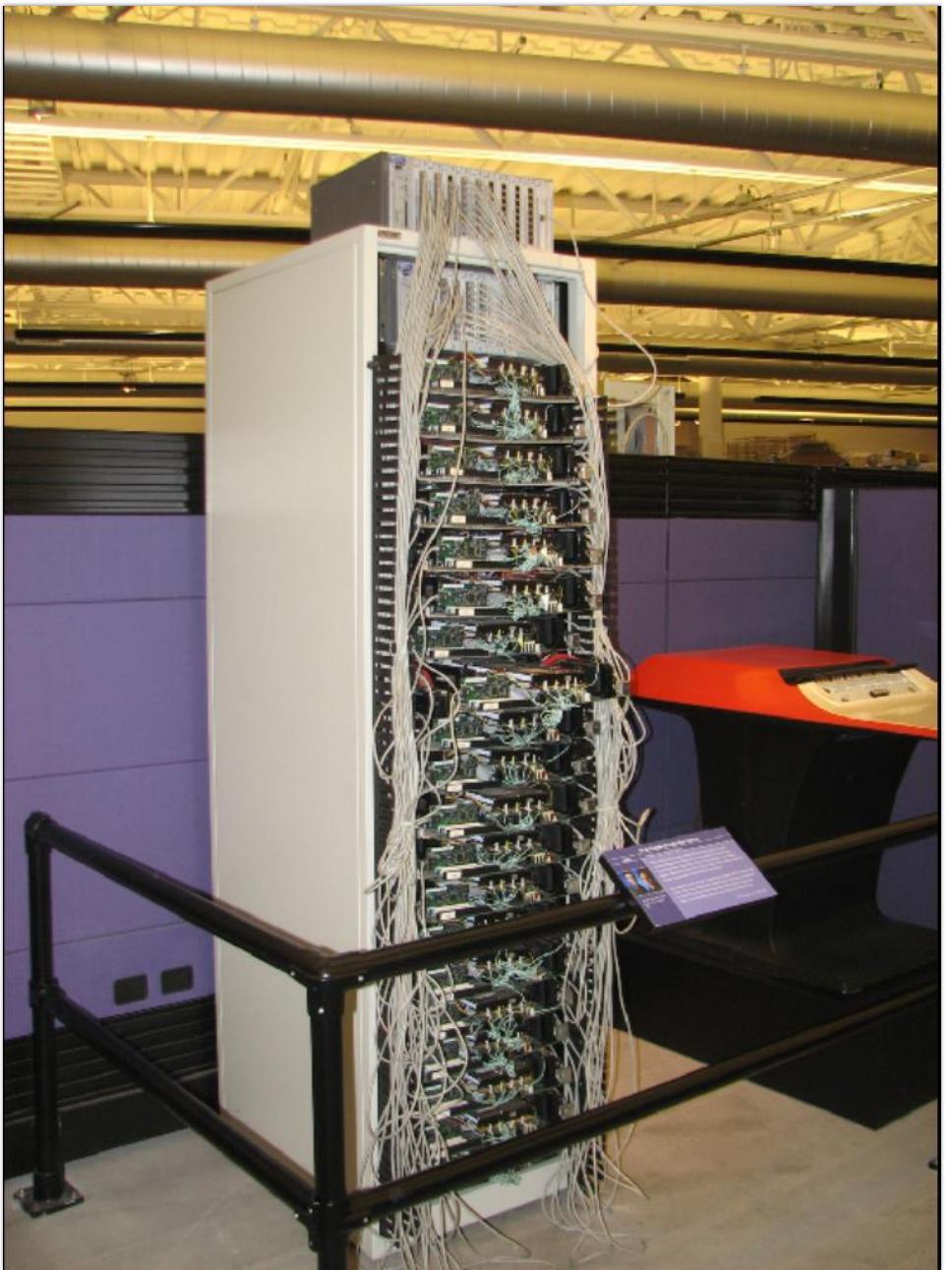
# *CLIENT-SERVER*

# *Klient - serwer*

To czy maszyna (komputer) pełni rolę klienta, czy serwera zależy **wyłącznie** od zainstalowanego na niej oprogramowania.

Jeżeli na komputerze zainstalowano oprogramowanie serwerowe, komputer będzie pełnił rolę serwera. Jeżeli na komputerze zainstalowano oprogramowanie klienckie, komputer będzie pełnił rolę klienta.

Rolą klienta jest wysyłanie żądań oraz odbieranie odpowiedzi. Rolą serwera jest odbieranie i obsługa żądań klienta oraz "serwowania" odpowiedzi (danych) z powrotem do klienta.



*SERWER KIEDYS*



*SERWER DZIS*



Dalekopis (ang. teleprinter)

*KLIENT KIEDYS*



*KLIENT DZIS*



SERWER



Klient

*ARCHITEKTURA  
Klient-Serwer*



SERWER



Klient

*ARCHITEKTURA  
Klient-Serwer*



SERWER



Klient

*ARCHITEKTURA  
Klient-Serwer*



SERWER



Klient

*ARCHITEKTURA  
Klient-Serwer*

# HTTP STATUS CODE

1XX Informational	
<b>100</b>	Continue
<b>101</b>	Switching Protocols
<b>102</b>	Processing
2XX Success	
<b>200</b>	OK
<b>201</b>	Created
<b>202</b>	Accepted
<b>203</b>	Non-authoritative Information
<b>204</b>	No Content
<b>205</b>	Reset Content
<b>206</b>	Partial Content
<b>207</b>	Multi-Status
<b>208</b>	Already Reported
<b>226</b>	IM Used
3XX Redirectional	
<b>300</b>	Multiple Choices
<b>301</b>	Moved Permanently
<b>302</b>	Found
<b>303</b>	See Other
<b>304</b>	Not Modified
<b>305</b>	Use Proxy
<b>307</b>	Temporary Redirect
<b>308</b>	Permanent Redirect
4XX Client Error	
<b>400</b>	Bad Request
<b>401</b>	Unauthorized
<b>402</b>	Payment Required
<b>403</b>	Forbidden
<b>404</b>	Not Found
<b>405</b>	Method Not Allowed
<b>406</b>	Not Acceptable
<b>407</b>	Proxy Authentication Required
<b>408</b>	Request Timeout
4XX Client Error Continued	
<b>409</b>	Conflict
<b>410</b>	Gone
<b>411</b>	Length Required
<b>412</b>	Precondition Failed
<b>413</b>	Payload Too Large
<b>414</b>	Request-URI Too Long
<b>415</b>	Unsupported Media Type
<b>416</b>	Requested Range Not Satisfiable
<b>417</b>	Expectation Failed
<b>418</b>	I'm a teapot
<b>421</b>	Misdirected Request
<b>422</b>	Unprocessable Entity
<b>423</b>	Locked
<b>424</b>	Failed Dependency
<b>426</b>	Upgrade Required
<b>428</b>	Precondition Required
<b>429</b>	Too Many Requests
<b>431</b>	Request Header Fields Too Large
<b>444</b>	Connection Closed Without Response
<b>451</b>	Unavailable For Legal Reasons
<b>499</b>	Client Closed Request
5XX Server Error	
<b>500</b>	Internal Server Error
<b>501</b>	Not Implemented
<b>502</b>	Bad Gateway
<b>503</b>	Service Unavailable
<b>504</b>	Gateway Timeout
<b>505</b>	HTTP Version Not Supported
<b>506</b>	Variant Also Negotiates
<b>507</b>	Insufficient Storage
<b>508</b>	Loop Detected
<b>510</b>	Not Extended
<b>511</b>	Network Authentication Required
<b>599</b>	Network Connect Timeout Error

H T T P   S T A T U S   C O D E S

When a browser requests a service from a web server, an error may occur.  
This is a list of HTTP status messages that might be returned.

## Statusy wykorzystywane w popularnych API

API	Status Codes
Twitter	200, 304, 400, 401, 403, 404, 406, 410, 420, 422, 429, 500, 502, 503, 504
Stripe	200, 400, 401, 402, 404, 429, 500, 502, 503, 504
Github	200, 400, 422, 301, 302, 304, 307, 401, 403
Pagerduty	200, 201, 204, 400, 401, 403, 404, 408, 500
NewRelic Plugins	200, 400, 403, 404, 405, 413, 500, 502, 503, 503
Etsy	200, 201, 400, 403, 404, 500, 503
Dropbox	200, 400, 401, 403, 404, 405, 429, 503, 507
Spotify	200, 201, 204, 304, 400, 401, 403, 404, 429, 500, 502, 503
Google Cloud	200, 301, 304, 307, 308, 400, 401, 403, 404, 405, 409, 411, 412, 416, 429, 500, 501, 503
HipChat	200, 201, 202, 400, 401, 403, 404, 405, 429, 500, 503
Telegram	200, 303, 400, 401, 403, 404, 420, 500
Pocket	200, 400, 401, 403, 503
Uber	200, 201, 400, 401, 403, 404, 406, 409, 422, 429, 500

Źródło: <https://gist.github.com/vkostyukov/32c84c0c01789425c29a>

# HTTP Status Codes



*HTTP STATUS  
CODE*

*WBUDOWANE  
KONWERTERY  
URL*

<code>string</code>	(default) accepts any text without a slash
<code>int</code>	accepts positive integers
<code>float</code>	accepts positive floating point values
<code>path</code>	like <code>string</code> but also accepts slashes
<code>uuid</code>	accepts UUID strings

*MOST COMMON FIELD TYPES*

*SZABLONY*

**Szablon** - plik tekstowy definiujący sposób wyświetlania danych, buduje tzw. **warstwę prezentacji aplikacji**.

*Szablon może służyć do wygenerowania dowolnego formatu tekstu. Treść szablonu generowana jest automatycznie za pomocą instrukcji tzw. **języka szablonów**.*

*JEZYKI SZABLONOW*

**Język szablonów** służy do **automatycznego generowania treści**. Format automatycznie wygenerowanej treści zależy od użytego języka szablonów. Do najpopularniejszych języków szablonów należą języki generujące treść w formacie *xml (XML-based template language)* oraz języki generujące treści w formacie *html (HTML-based template language)*.

*Każdy język programowania posiada swoje, dedykowane języki szablonów.*

*Dla Pythona:*

- *najpopularniejsze języki szablonów przeznaczone do generowania plików XML to m.in. Genshi XML i Zope's TAL*
- *najpopularniejsze języki szablonów generujące pliki html dla Pythona to: Jinja, Mako, Genshi, Mustache. Django posiada swój własny język szablonów.*

*Język szablonów Django używamy głównie do generowania plików html, ale (jak większość języków szablonów) może być z powodzeniem użyty do generowania dowolnego formatu tekstowego (email, csv, json, ...).*

*Szersze omówienie języków szablonów dla Pythona można znaleźć [tutaj](#)*

*Logika wyrażona za pomocą języka szablonów powinna dotyczyć wyłącznie zagadnień związanych z prezentacją danych. Za pomocą języka szablonów nie należy implementować logiki biznesowej aplikacji.*

*JEZYKI  
SZABLONOW HTML  
DLA PYTHONA*



Hyperfast and lightweight templating for the Python platform.

[Home](#) | [Community](#) | [Documentation](#) | [Download](#)

## Mako Templates for Python

Mako is a template library written in Python. It provides a familiar, non-XML syntax which compiles into Python modules for maximum performance. Mako's syntax and API borrows from the best ideas of many others, including Django and Jinja2 templates, Cheetah, Myghty, and Genshi. Conceptually, Mako is an embedded Python (i.e. Python Server Page) language, which refines the familiar ideas of componentized layout and inheritance to produce one of the most straightforward and flexible models available, while also maintaining close ties to Python calling and scoping semantics.

Mako is used by [reddit.com](#) where it delivers over [one billion page views per month](#). It is the default template language included with the [Pylons](#) and [Pyramid](#) web frameworks.

### Nutshell:

```
<%inherit file="base.html"/>
<%
    rows = [[v for v in range(0,10)] for row in range(0,10)]
%>






<%def name="makerow(row)">
    <tr>
        % for name in row:
            <td>${name}</td> \
        % endfor
    </tr>
<%enddef>
```



### Philosophy:



Szukaj

Zaloguj się | Ustawienia | Pomoc/Przewodnik | O systemie Trac

Home | Trac | Trac Hacks | **Genshi** | Babel | Bitten

Wiki

Historia

Plan prac

Przeglądaj repozytorium

Zobacz zgłoszenia

Szukaj

wiki: WikiStart

Strona poczatkowa | Indeks | Historia

## Genshi

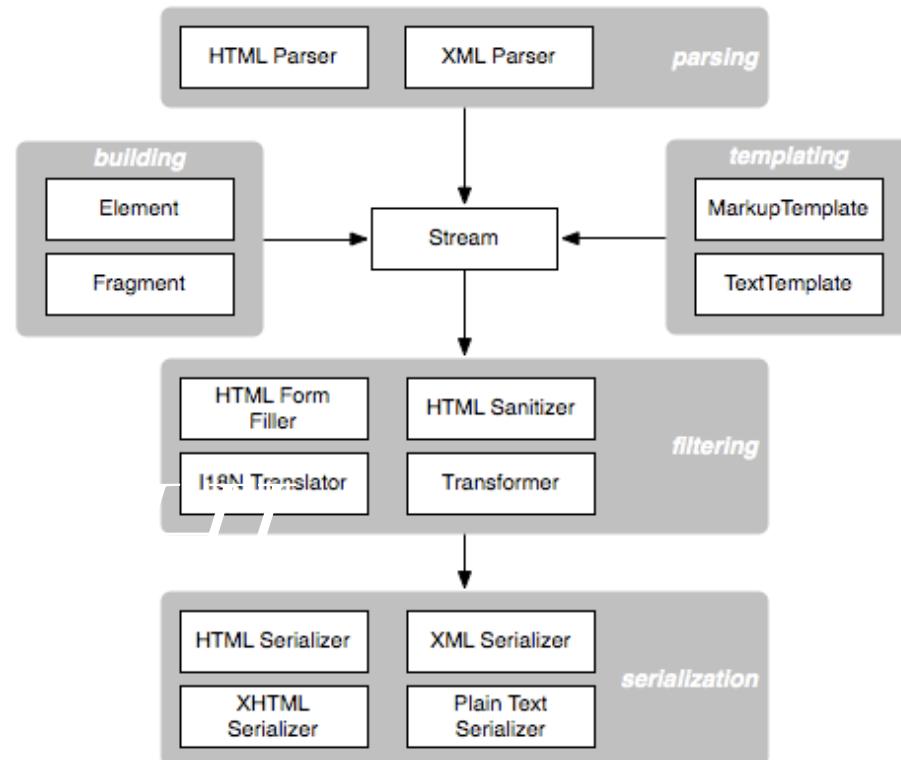
### *Python toolkit for generation of output for the web*

**Genshi** is a Python library that provides an integrated set of components for parsing, generating, and processing HTML, XML or other textual content for output generation on the web.

The main feature is a [template language](#) that is smart about markup: unlike conventional template languages that only deal with bytes and (if you're lucky) characters, Genshi knows the difference between tags, attributes, and actual text nodes, and uses that knowledge to your advantage. For example:

- Intelligent automatic escaping greatly reduces the risk of opening up your site to [cross-site scripting](#) attacks (XSS).
- [Template directives](#) are often less verbose than those in most other template languages, as they can be attached directly to the elements they act upon.
- [Independence from a specific serialization format](#) lets you instantly switch between generating well-formed HTML 4.01 and XHTML 1.0 (or other formats).
- [Stream-based filtering](#) allows you to apply various transformations as a template is being processed, without having to parse and serialize the output again.
- [Match templates](#) let you enforce a common structure on template output, and more. This, in combination with [XInclude](#) support, is used instead of the more rigid inheritance feature commonly found in other template languages.

For those cases where you don't want to generate markup, Genshi also provides a simple [text-based template language](#).





Logic-less templates.

Available in [Ruby](#), [JavaScript](#), [Python](#),  
[Erlang](#), [Elixir](#), [PHP](#), [Perl](#), [Raku](#), [Objective-C](#),  
[Java](#), [C#/.NET](#), [Android](#), [C++](#), [CFEngine](#),  
[Go](#), [Lua](#), [ooc](#), [ActionScript](#), [ColdFusion](#),  
[Scala](#), [Clojure\[Script\]](#), [Clojure](#), [Fantom](#),  
[CoffeeScript](#), [D](#), [Haskell](#), [XQuery](#), [ASP](#), [Io](#),  
[Dart](#), [Haxe](#), [Delphi](#), [Racket](#), [Rust](#), [OCaml](#),  
[Swift](#), [Bash](#), [Julia](#), [R](#), [Crystal](#), [Common Lisp](#),  
[Nim](#), [Pharo](#), [Tcl](#), [C](#), [ABAP](#), [Elm](#), [Kotlin](#), and  
for [SQL](#)

Works great with TextMate Vim Emacs

## Project Links

- [Donate](#)
- [PyPI Releases](#)
- [Source Code](#)
- [Issue Tracker](#)
- [Website](#)
- [Twitter](#)
- [Chat](#)

## Quick search

 Go

  
Stop searching. Start doing.  
We simplify search for complex enterprises. [Ask for a 30 Day Free Trial](#)  
Ad by EthicalAds · Monetize your site



Jinja is a fast, expressive, extensible templating engine. Special placeholders in the template allow writing code similar to Python syntax. Then the template is passed data to render the final document.

### Contents:

- [Introduction](#)
  - [Installation](#)
- [API](#)
  - [Basics](#)
  - [High Level API](#)
  - [Autoescaping](#)
  - [Notes on Identifiers](#)
  - [Undefined Types](#)
  - [The Context](#)
  - [Loaders](#)
  - [Bytecode Cache](#)
  - [Async Support](#)
  - [Policies](#)
  - [Utilities](#)
  - [Exceptions](#)
  - [Custom Filters](#)
  - [Custom Tags](#)

## Documentation

Search 3.2 documentation



# The Django template language

This document explains the language syntax of the Django template system. If you're looking for a more technical perspective on how it works and how to extend it, see [The Django template language: for Python programmers](#).

Django's template language is designed to strike a balance between power and ease. It's designed to feel comfortable to those used to working with HTML. If you have any exposure to other text-based template languages, such as [Smarty](#) or [Jinja2](#), you should feel right at home with Django's templates.



## Philosophy

If you have a background in programming, or if you're used to languages which mix programming code directly into HTML, you'll want to bear in mind that the Django template system is not simply Python embedded into HTML. This is by design: the template system is meant to express presentation, not program logic.

The Django template system provides tags which function similarly to some programming constructs – an `if` tag for boolean tests, a `for` tag for looping, etc. – but these are not simply executed as the corresponding Python code, and the template system will not execute arbitrary Python expressions. Only the tags, filters and syntax listed below are supported by default (although you can add your own extensions to the template language as needed).

## Support Django!



WashDrop donated to the Django Software Foundation to support Django development. [Donate today!](#)

## Contents

- [The Django template language](#)
  - [Templates](#)
  - [Variables](#)
  - [Filters](#)
  - [Tags](#)
  - [Comments](#)
  - [Template inheritance](#)

Getting Help

Language: en

Documentation version: 3.2

*JEZYK SZABLONOW DJANGO*  
*(DTL – DJANGO TEMPLATE LANGUAGE)*

# *PRZYKŁAD SZABLONU DJANGO*

```
{% extends "base_generic.html" %}

{% block title %}{{ section.title }}{% endblock %}

{% block content %}
<h1>{{ section.title }}</h1>

{% for story in story_list %}
<h2>
    <a href="{{ story.get_absolute_url }}">
        {{ story.headline|upper }}
    </a>
</h2>
<p>{{ story.tease|truncatewords:"100" }}</p>
{% endfor %}
{% endblock %}
```

# *ELEMENTY SKLADNIOWE JĘZYKA SZABLONÓW DJANGO*

1. **Zmienne**  
*Zastępowane wartościami podczas generowania szablonu.*
2. **Tagi (znaczniki)**  
*Instrukcje obsługujące logikę szablonu, w tym instrukcje sterujące (warunki, pętle).*
3. **Filtry**  
*Funkcje modyfikujące wartości zmiennych.*
4. **Parametry filtra**  
*Dodatkowe parametry przyjmowane przez niektóre filtry, modyfikujące działanie tych filtrów*
5. **Notacja z kropką**  
*Odwołanie po kluczu, po atrybucie obiektu i po indeksie (dokładnie w tej kolejności) realizowane jest w ten sam sposób - za pomocą kropki.*
6. **Komentarze**

PC File Edit View Navigate Code Refactor Run Tools VCS Window DB Navigator Help intro - hello.html

intro > hello > hello\_app > templates > hello\_app > hello.html

Project DB Browser Structure Favorites

File Project settings.py hello\urls.py hello\_app\urls.py views.py hello.html

\_\_pycache\_\_  
\_init\_.py  
asgi.py  
settings.py  
urls.py  
wsgi.py  
hello\_app  
migrations  
templates  
hello\_app  
hello.html  
\_init\_.py  
admin.py  
apps.py  
models.py  
tests.py  
urls.py  
views.py  
manage.py  
venv library root

Terminal: Local +

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Powitanie</title>
</head>
<body>
    Witaj, {{name}}!
</body>
</html>
```

Microsoft Windows [Version 10.0.19042.1288]  
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.  
(venv) C:\Users\jerem\PycharmProjects\intro>

TODO Problems DB Execution Console Terminal Python Console

Packages installed successfully: Installed packages: 'Django' (today 20:01)

1. ZMIENNE  
{{ NAZWA\_ZMIENNEJ }}

# ZMienne

- Nazwa zmiennej może być dowolną kombinacją znaków alfanumerycznych i podkreśnika (z wyłączeniem kombinacji, które zaczynają się od podkreśnika lub są liczbami)
- Jeżeli wprowadzona w szablonie zmienna nie istnieje, silnik języka szablonów Django w jej miejscu wstawi wartość zmiennej `string_if_invalid` (której domyślna wartość to pusty napis - ''). Odwołanie do nieistniejącej zmiennej NIE wywoła błędu!

File Edit View Navigate Code Refactor Run Tools VCS Window DB Navigator Help intro - fruits.html

intro > hello > hello\_app > templates > hello\_app > fruits.html

Project DB Browser Structure Favorites

fruits.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Owoce</title>
</head>
<body>
    <ul>
        {% for fruit in fruits %}
            <li>{{ fruit }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

(venv) C:\Users\jerem\PycharmProjects\intro>

2. TAGI

{% NAZWA\_TAGA %}

# *TAGI (ZNACZNIKI)*

- *Niektóre znaczniki posiadające swój odpowiednik zamykający i wymagają jego użycia, np.:*  
`{% for ... %}{% endfor %}`  
`{% if ... %}{% endif %}`  
`{% block %}{% endblock %}`
- *Jinja pozwala na definiowanie własnych tagów (patrz dodatki na końcu tej prezentacji).*

Variable	Description
<i>loop.index</i>	The current iteration of the loop. (1 indexed)
<i>loop.index0</i>	The current iteration of the loop. (0 indexed)
<i>loop.revindex</i>	The number of iterations from the end of the loop (1 indexed)
<i>loop.revindex0</i>	The number of iterations from the end of the loop (0 indexed)
<i>loop.first</i>	True if first iteration.
<i>loop.last</i>	True if last iteration.
<i>loop.length</i>	The number of items in the sequence.
<i>loop.cycle</i>	A helper function to cycle between a list of sequences. See the explanation below.
<i>loop.depth</i>	Indicates how deep in a recursive loop the rendering currently is. Starts at level 1
<i>loop.depth0</i>	Indicates how deep in a recursive loop the rendering currently is. Starts at level 0
<i>loop.previtem</i>	The item from the previous iteration of the loop. Undefined during the first iteration.
<i>loop.nextitem</i>	The item from the following iteration of the loop. Undefined during the last iteration.
<i>loop.changed(*val)</i>	True if previously called with a different value (or not called at all).

# LOOP VARIABLES

PC File Edit View Navigate Code Refactor Run Tools VCS Window DB Navigator Help intro - fruits.html

intro > hello > hello\_app > templates > hello\_app > fruits.html

Project DB Browser Structure Favorites

intro C:\Users\jerem\PycharmProjects\intro  
hello  
  hello  
    \_\_pycache\_\_  
    \_\_init\_\_.py  
    asgi.py  
    settings.py  
    urls.py  
    wsgi.py  
  hello\_app  
    migrations  
    templates  
      hello\_app  
        fruits.html  
        hello.html  
        \_\_init\_\_.py  
        admin.py  
        apps.py  
        models.py  
        tests.py  
        urls.py  
        views.py  
        manage.py  
  venv library root

Terminal: Local +

(venv) C:\Users\jerem\PycharmProjects\intro>

1 <!DOCTYPE html>  
2 <html lang="en">  
3 <head>  
4     <meta charset="UTF-8">  
5     <title>Owoce</title>  
6 </head>  
7 <body>  
8     <p>{{ fruits|length }}</p>  
9 </body>  
10 </html>

3. FILTRY  
{{ NAZWA\_ZMIENNEJ | NAZWA\_FILTRA }}

TODO Problems DB Execution Console Terminal Python Console  
PyCharm 2020.3.5 available // Update... (2 minutes ago)

# *FILTRY*

- Przykład filtra: filtr *lower* w działaniu na zmienną *name* - `{{ name | lower }}`
- Filtry można łańcuchować `{{ text | safe| upper}}`
- *Jinja* pozwala na tworzenie własnych filtrów (patrz dodatki na końcu tej prezentacji)

## List of Builtin Filters

<code>abs()</code>	<code>forceescape()</code>	<code>map()</code>	<code>select()</code>	<code>unique()</code>
<code>attr()</code>	<code>format()</code>	<code>max()</code>	<code>selectattr()</code>	<code>upper()</code>
<code>batch()</code>	<code>groupby()</code>	<code>min()</code>	<code>slice()</code>	<code>urlencode()</code>
<code>capitalize()</code>	<code>indent()</code>	<code>pprint()</code>	<code>sort()</code>	<code>urlize()</code>
<code>center()</code>	<code>int()</code>	<code>random()</code>	<code>string()</code>	<code>wordcount()</code>
<code>default()</code>	<code>items()</code>	<code>reject()</code>	<code>striptags()</code>	<code>wordwrap()</code>
<code>dictsort()</code>	<code>join()</code>	<code>rejectattr()</code>	<code>sum()</code>	<code>xmlattr()</code>
<code>escape()</code>	<code>last()</code>	<code>replace()</code>	<code>title()</code>	
<code>filesizeformat()</code>	<code>length()</code>	<code>reverse()</code>	<code>tojson()</code>	
<code>first()</code>	<code>list()</code>	<code>round()</code>	<code>trim()</code>	
<code>float()</code>	<code>lower()</code>	<code>safe()</code>	<code>truncate()</code>	

*FILTRY*

File Edit View Navigate Code Refactor Run Tools VCS Window DB Navigator Help intro - passengers.html

intro > hello > hello\_app > templates > hello\_app > passengers.html

Project DB Browser Structure Favorites

passengers.html

1 <!DOCTYPE html>  
2 <html lang="en">  
3 <head>  
4 <meta charset="UTF-8">  
5 <title>Pasażerowie</title>  
6 </head>  
7 <body>  
8 <!-- Odwoływanie po indeksie -->  
9 <p>{{ passengers.0 }}</p>  
10 <!-- Odwoływanie po kluczu -->  
11 <p>{{ passengers.0.age }}</p>  
12 <!-- Odwoływanie po atrybucie -->  
13 <p>{{ passengers.0.age.real }}</p>  
14 </body>

passengers.html

venv library root External Libraries Scratches and Consoles

Terminal: Local +

(venv) C:\Users\jerem\PycharmProjects\intro>

TODO Problems DB Execution Console Terminal Python Console

PyCharm 2020.3.5 available // Update... (39 minutes ago)

## 5. NOTACJA Z KROPKĄ

# KOMENTARZE

- *Zawartość komentarzy Jinja nie jest traktowana przez silnik szablonów jako instrukcje Jinja.*
- *Komentarze Jinja w odróżnieniu od komentarzy HTML będą niewidoczne w ciele odpowiedzi. Silnik szablonów wycina wszystkie komentarze w trakcie generowania treści szablonu.*

# *PRZYKŁAD SZABLONU DJANGO (OMÓWIENIE)*

```
{% extends "base_generic.html" %} Zmienna  
{% block title %}{{ section.title }}{% endblock %}  
  
{% block content %} Tag  
<h1>{{ section.title }}</h1>  
  
{% for story in story_list %} Notacja z kropką  
<h2>  
  <a href="{{ story.get_absolute_url }}">  
    {{ story.headline|upper }}  
  </a>  
</h2> Filtr  
<p>{{ story.tease|truncatewords:"100" }}</p>  
{% endfor %} Parametr filtra  
{% endblock %}
```

# *PRZYKŁADY ZASTOSOWAŃ*

1. Treści statyczne
2. Warunki
3. Pętle
4. Dynamiczne linkowanie
5. Dziedziczenie
6. Kompozycja

*ORM*

*WSTEP*







0010011101101010100110101010111001





0010011101101010100110101010111001

```
CREATE TABLE user
(
    user_id INT NOT NULL PRIMARY KEY,
);
```





ORACLE®

0010011101101010100110101010111001



```
CREATE TABLE user
(
    user_id numeric(10) not null,
    CONSTRAINT user_pk PRIMARY KEY (user_id)
);
```



ORM  
Django

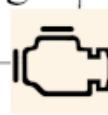
0010011101101010100110101010111001





ORM

Django

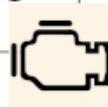


0010011101101010100110101010111001





ORM  
Django



0010011101101010100110101010111001

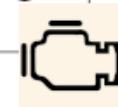


```
class User(models.Model):  
    user_id = models.IntegerField(primary_key=True)
```



ORM

Django



0010011101101010100110101010111001



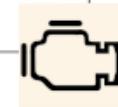
```
class User(models.Model):
    user_id = models.IntegerField(primary_key=True)
```

```
CREATE TABLE user
(
    user_id INT NOT NULL PRIMARY KEY,
);
```



ORM

Django

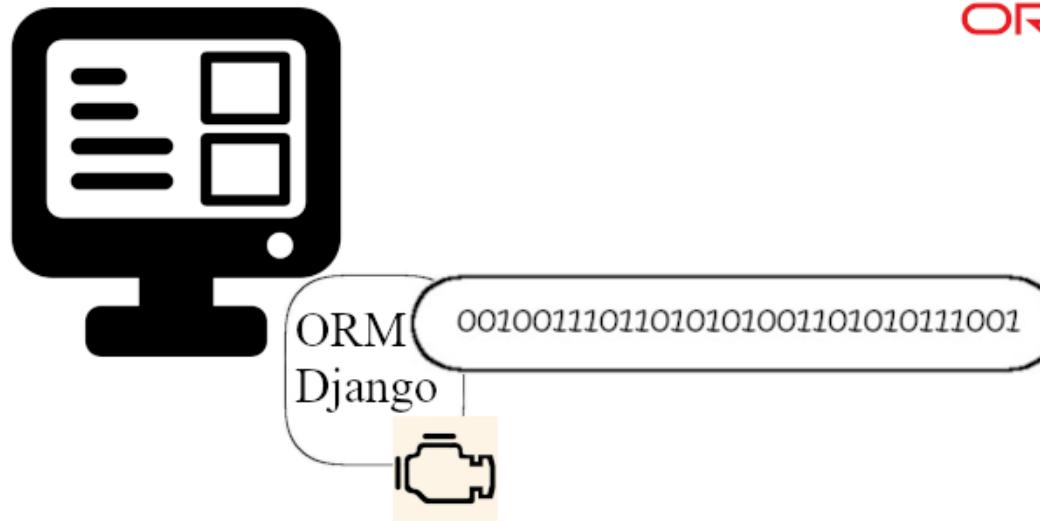


0010011101101010100110101010111001

ORACLE®



```
class User(models.Model):  
    user_id = models.IntegerField(primary_key=True)
```



```
class User(models.Model):  
    user_id = models.IntegerField(primary_key=True)
```

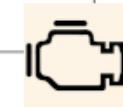


```
CREATE TABLE user  
(  
    user_id numeric(10) not null,  
    CONSTRAINT user_pk PRIMARY KEY (user_id)  
>);
```



ORM

Django



001001101101010100110101010111001

```
class User(models.Model):  
    user_id = models.IntegerField(primary_key=True)
```



PostgreSQL



MySQL®

ORACLE®



# *PYTHONORM*

*PRZEGŁAD*

peewee — peewee 3.14.4 documentation

Niezabezpieczona | docs.peewee-orm.com/en/latest/

peewee latest

Docs » peewee

Edit on GitHub

# peewee



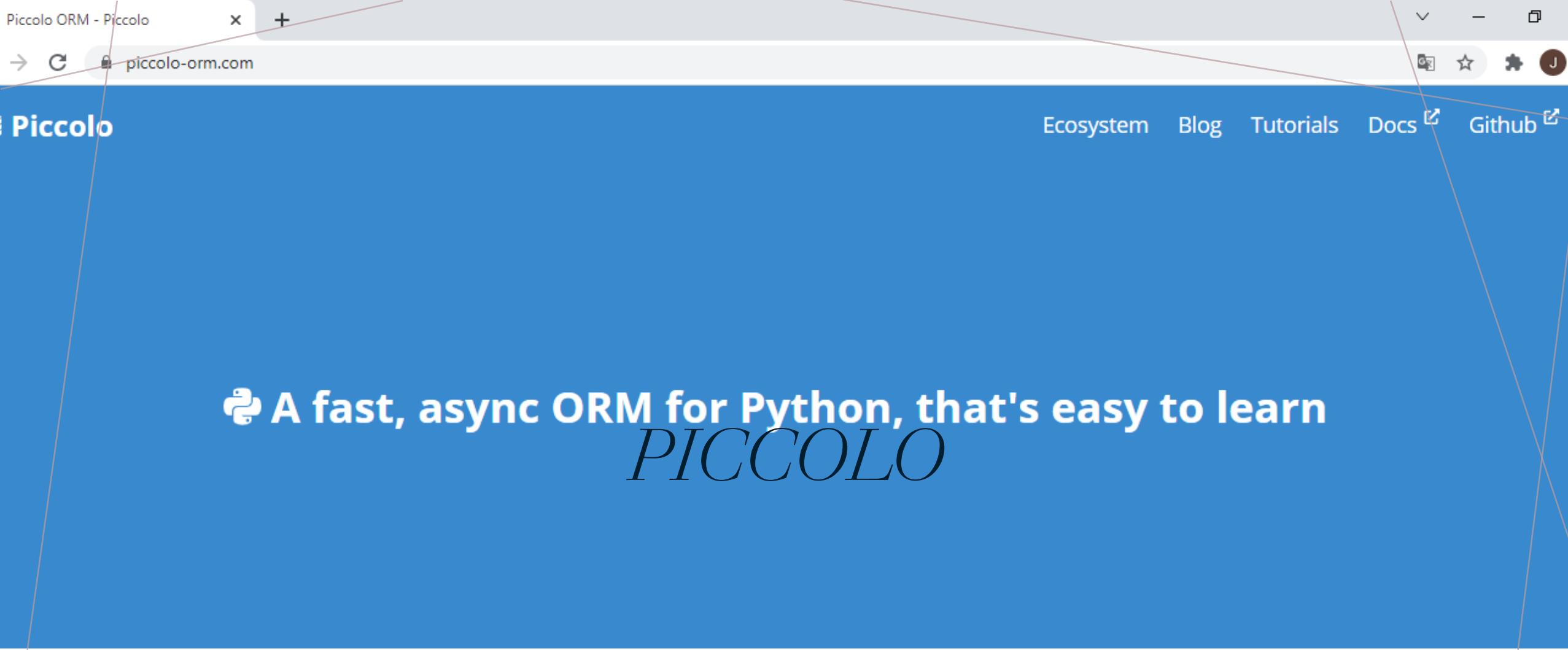
Peewee is a simple and small ORM. It has few (but expressive) concepts, making it easy to learn and intuitive to use.

- a small, expressive ORM
- python 2.7+ and 3.4+ (developed with 3.6)
- supports sqlite, mysql, postgresql and cockroachdb
- tons of extensions

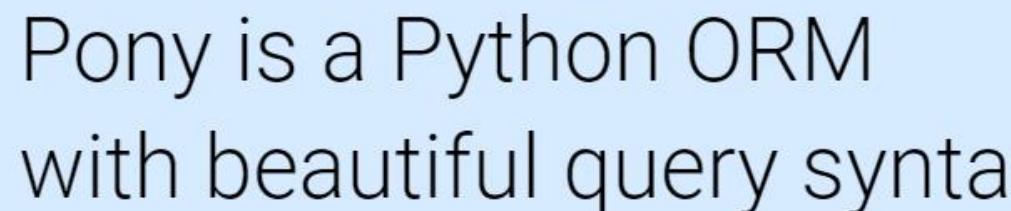
Peewee's source code hosted on [GitHub](#).

New to peewee? These may help:



## Benefits

- ✓ Supports Postgres and SQLite



Write your database queries using Python generators & lambda

Try PonyORM now

Support PonyORM

Free open-source software

PONY



[Github](#)



Twitt



Telegra



 Join the newsletter

## Custom software development

The Web framework for perfectionists with deadlines.

OVERVIEW DOWNLOAD DOCUMENTATION NEWS COMMUNITY CODE ISSUES ABOUT DONATE

Django makes it easier to build better Web apps more quickly and with less code.

Get started with Django

# DJANGO ORM

## Meet Django

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.



Ridiculously fast.

Django was designed to help developers take applications from concept to completion

Download latest release: 3.2.5

DJANGO DOCUMENTATION >

Support Django!



McInnes Cooper donated to the Django Software Foundation to support Django development. [Donate today!](#)

# SQLAlchemy

THE DATABASE TOOLKIT FOR PYTHON

## The Python SQL Toolkit and Object Relational Mapper

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

It provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.

### SQLALCHEMY'S PHILOSOPHY

SQL databases behave less like object collections the more size and performance start to matter; object collections behave less like tables and rows the more abstraction starts to matter. SQLAlchemy aims to accommodate both of these principles.

SQLAlchemy considers the database to be a relational algebra engine, not just a collection of tables. Rows can be selected from not only tables but also joins and other select statements; any of these units can be composed into a larger structure. SQLAlchemy's expression language builds on this concept from its core.

SQLAlchemy is most famous for its object-relational mapper (ORM), an optional component that provides the **data mapper pattern**, where classes can be mapped to the database in open ended, multiple ways - allowing the object model and database schema to develop in a cleanly decoupled way from the beginning.

SQLAlchemy's overall approach to these problems is entirely different from that of most other SQL / ORM tools, rooted in a so-called **complimentarity**- oriented approach; instead of hiding away SQL and object relational details behind a wall of automation, all processes are **fully exposed** within a series of composable, transparent tools. The library takes on the job of automating redundant tasks while the developer remains in control of how the database is organized and how SQL is constructed.

The main goal of SQLAlchemy is to change the way you think about databases and SQL!

Read some **key features** of SQLAlchemy, as well as **what people are saying** about SQLAlchemy.

### CURRENT RELEASES

**1.4.21** - 2021-07-14 - [announce](#)  
[changes](#) | [migration notes](#) | [docs](#)

**1.3.24** - 2021-03-30 - [announce](#)  
[changes](#) | [migration notes](#) | [docs](#)

PagerDuty  
Use tools,  
not snake oil.  
[Listen to Podcast](#)

Let software do the work of driving culture change.

ADS VIA CARBON

### SPONSOR SQLALCHEMY!

[Donate](#)

Donate to SQLAlchemy through [PayPal](#)



Sponsor SQLAlchemy through the Tidelift Subscription

### LATEST NEWS

SQLAlchemy 1.4.21 Released  
Wed, 14 Jul 2021

SQLAlchemy 1.4.20 Released  
Mon, 28 Jun 2021

SQLAlchemy 1.4.19 Released



## Project Link

[Donate](#)  
[PyPI Release](#)  
[Source Code](#)  
[Issue Tracker](#)  
[Website](#)  
[Twitter](#)  
[Chat](#)

Contents

# Flask-SQLAlchemy

## User Guide

## API Reference

## Additional Information

## Quick search

6

```
# .readthedocs.yaml
build.tools:
    python: "3.11"
sphinx:
    configuration: conf.py
python.install:
    - requirements: pip.in
```

**Supercharge your Sphinx docs** with deployment options. Read the Docs. **Sign up today!**

*Ad by EthicalAds*

# Flask-SQLAlchemy

Flask-SQLAlchemy is an extension for [Flask](#) that adds support for [SQLAlchemy](#) to your application. It simplifies using SQLAlchemy with Flask by setting up common objects and patterns for using those objects, such as a session tied to each web request, models, and engines.

Flask-SQLAlchemy does not change how SQLAlchemy works or is used. See the [SQLAlchemy documentation](#) to learn how to work with the ORM in depth. The documentation here will only cover setting up the extension, not how to use SQLAlchemy.

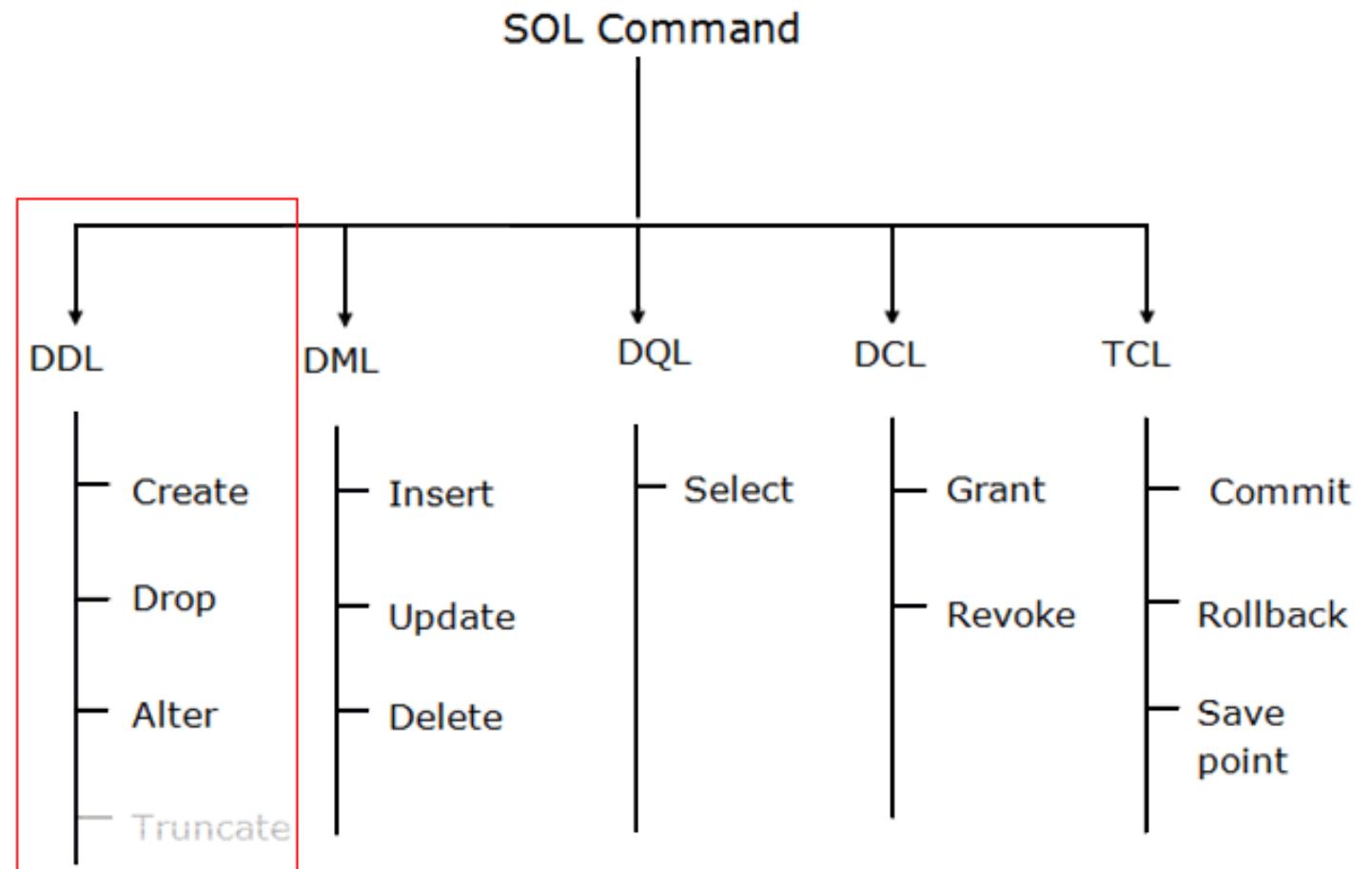
User Guide

- Quick Start
    - [Check the SQLAlchemy Documentation](#)
    - [Installation](#)
    - [Initialize the Extension](#)
    - [Configure the Extension](#)
    - [Define Models](#)
    - [Create the Tables](#)
    - [Query the Data](#)
    - [What to Remember](#)
  - Configuration
    - [Configuration Keys](#)
    - [Connection URL Format](#)
    - [Default Driver Options](#)
    - [Engine Configuration Precedence](#)
    - [Timeouts](#)
  - Models and Tables
    - [Initializing the Base Class](#)
    - [Initialize the Extension](#)
    - [Defining Models](#)
    - [Defining Tables](#)
    - [Reflecting Tables](#)
  - Modifying and Querying Data

# *FLASK-SQLALCHEMY*

# *DDL – DATA DEFINITION LANGUAGE*

- DDL – Data Definition Language
- DML – Data Manipulation Language
- DQL – Data Query Language
- DCL – Data Control Language
- TCL – Transaction Control Language



<b>Integer</b>	an integer
<b>String(size)</b>	a string with a maximum length (optional in some databases, e.g. PostgreSQL)
<b>Text</b>	some longer unicode text
<b>DateTime</b>	date and time expressed as Python <code>datetime</code> object.
<b>Float</b>	stores floating point values
<b>Boolean</b>	stores a boolean value
<b>PickleType</b>	stores a pickled Python object
<b>LargeBinary</b>	stores large arbitrary binary data

## *MOST COMMON FIELD TYPES*

# *DODATKI*



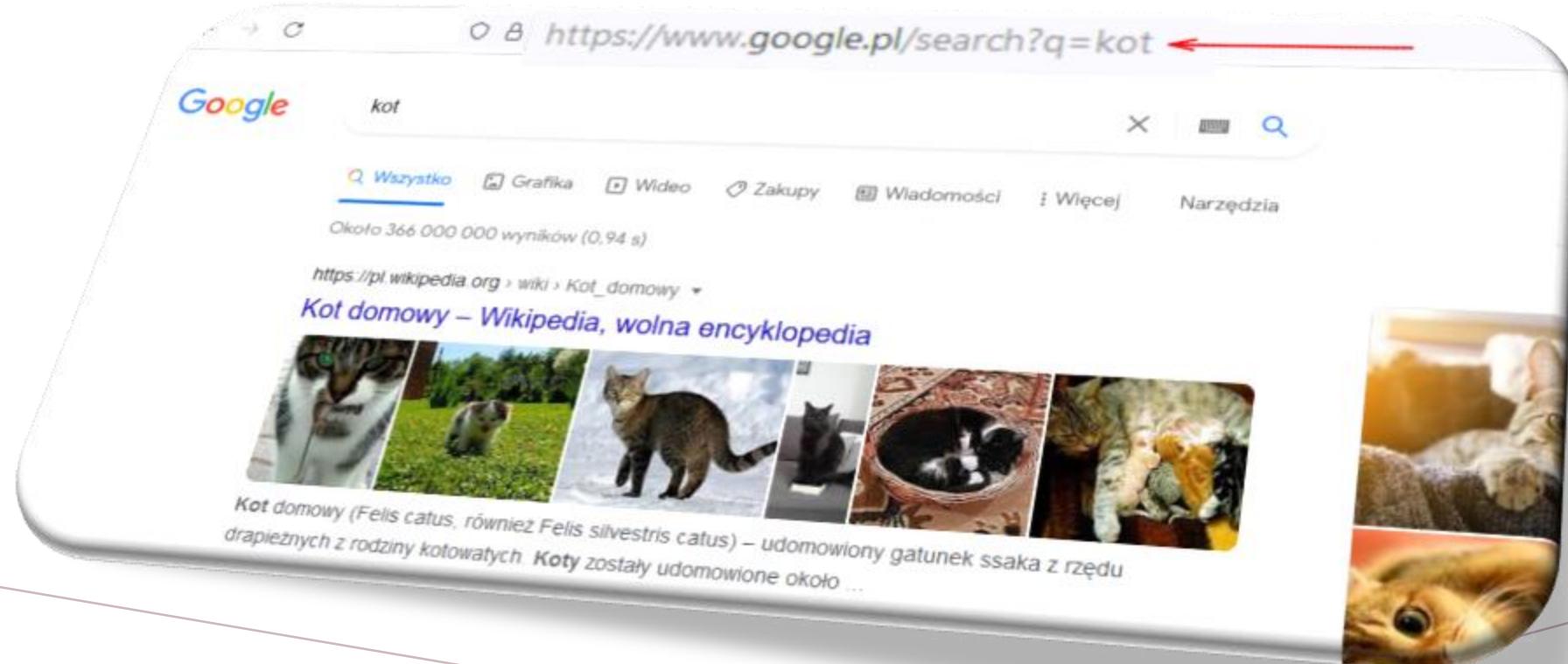
# *PRZEKAZYWANIE DANYCH OD UZYTKOWNIKA*

*PODSTAWOWE METODY PROTOKOLU HTTP*

# *METODA GET*

# DANE GET

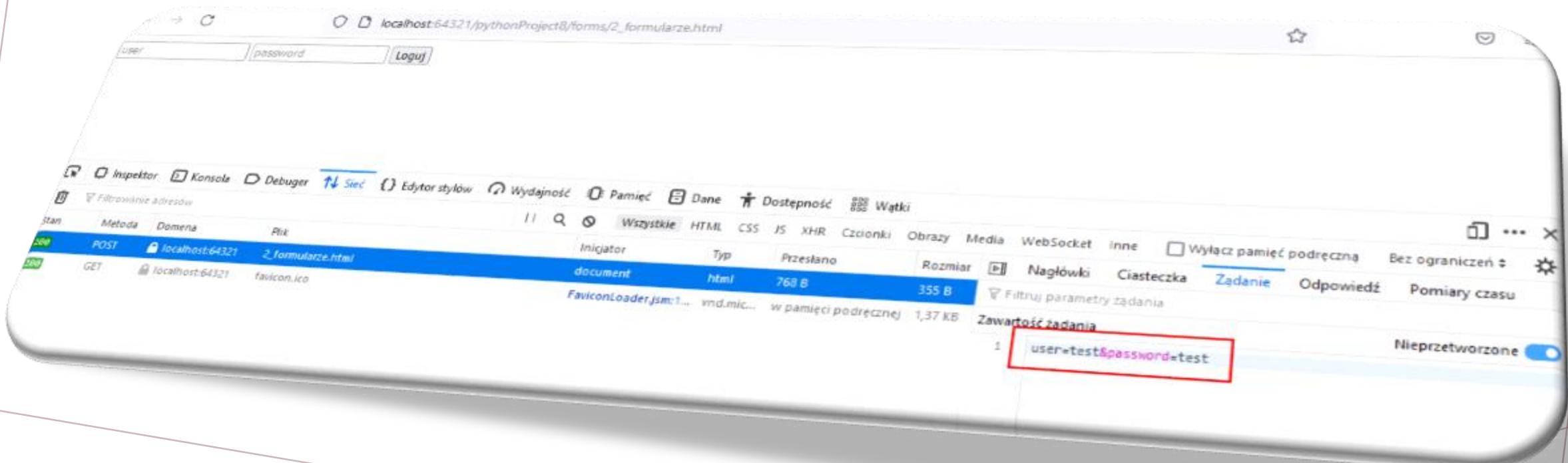
Dane przekazywane są w **adresie URL**, w postaci tzw. **parametrów GET**. Parametry GET **NIE** budują endpointu. Są dodatkowymi informacjami umieszczonymi **ZA** endpointem.



*METODA POST*

# DANE POST

Dane przekazywane są w ciele żądania, w postaci tzw. parametrów POST.



*FORMULARZE HTML*

# *FORMULARZE HTML*

*Formularze to elementy HTML, których celem jest udostępnienie użytkownikowi wygodnego sposobu wprowadzania danych.*

*Dwa podstawowe atrybuty formularza to:*

- ***action** – definiuje url (adres) na jaki zostaną przesłane wprowadzone dane.  
Domyślna wartość atrybutu action to ten sam adres, na którym wyświetlony został formularz.*
- ***method** – definiuje metodę, którą wprowadzone dane zostaną przesłane.  
Domyślna wartość atrybutu method to "GET".*



*ELEMENTY*  
*FORMULARZA HTML 5*

# *ELEMENTY FORMULARZA*

*Elementy formularza (aka pola formularza) to elementy html, z których budujemy formularz.*

*Zadaniem elementów formularza jest udostępnienie miejsca do wprowadzania danych. Dzięki elementom formularza użytkownik może wprowadzić informację do specjalnie przygotowanego pola, a formularz sam umieści wprowadzone w tym polu dane w odpowiednim miejscu podczas wysyłania zapytania (w miejscu zgodnym ze wskazaną w formularzu metodą http).*

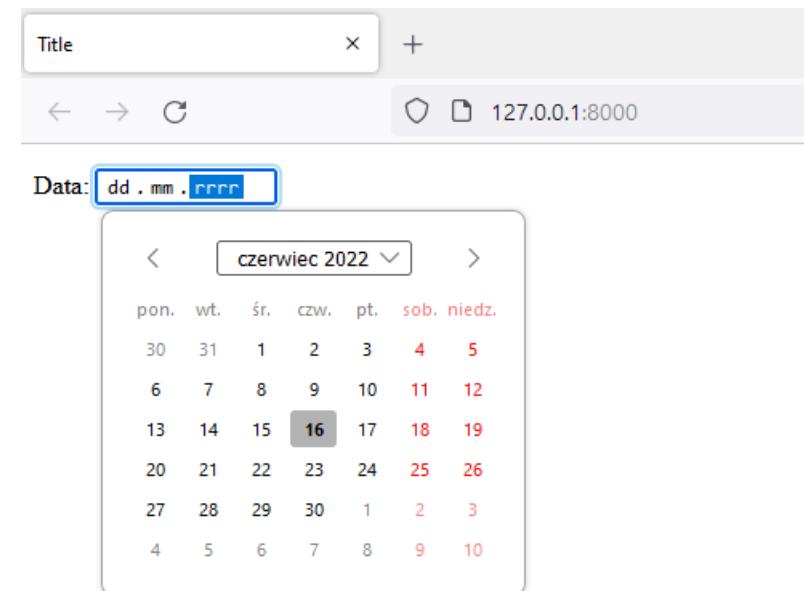
*Wyróżniamy 5 podstawowych elementów formularza:*

- *element <input>*
- *element <button>*
- *element <select>*
- *element <textarea>*
- *element <datalist>*

# ELEMENTY FORMULARZA - WIDGET'

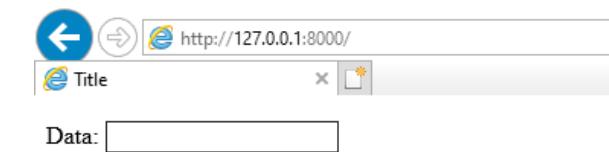
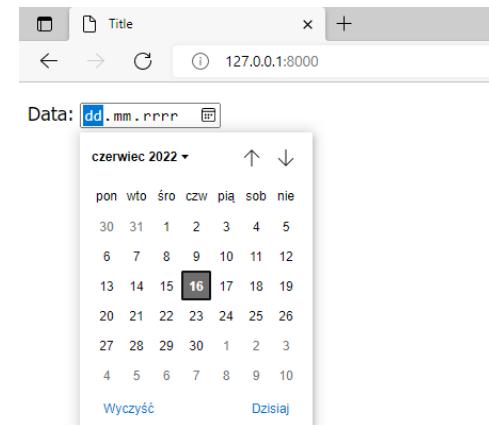
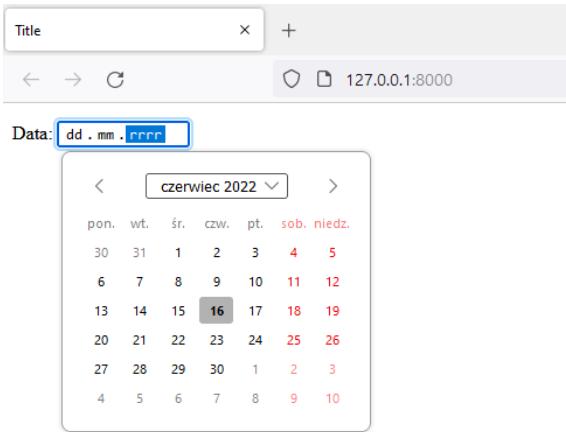
Przeglądarki internetowe, w celu zwiększenia własnej atrakcyjności wyposażają elementy formularza w wygodne **widgety** (elementy graficzne) do wprowadzania danych (np. kalendarz do wprowadzanie informacji o dacie).

```
<form>
    <p>Data: <input type="date" name="reservationDate"></p>
</form>
```



# ELEMENTY FORMULARZA - WIDGET

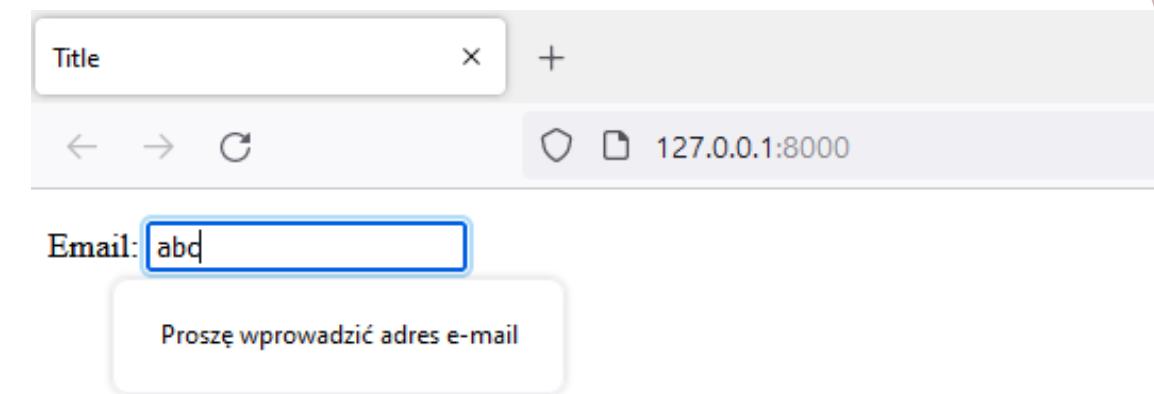
Widgety różnią się pomiędzy przeglądarkami, a niektóre przeglądarki mogą w ogóle nie implementować niektórych widgetów (np. brak dedykowanego widget-u dla elementu `input` typu `date` w IE).



# ELEMENTY FORMULARZA - WALIDATORY

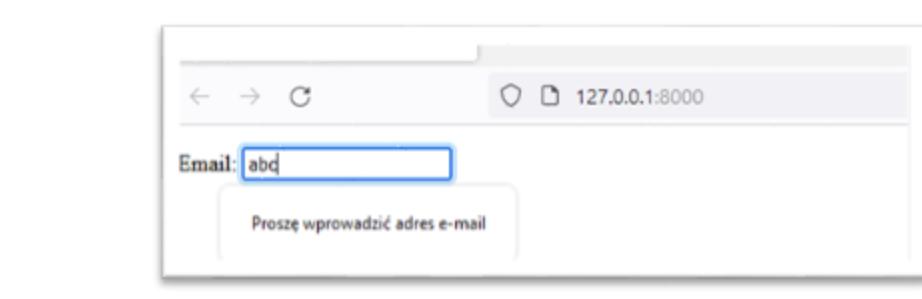
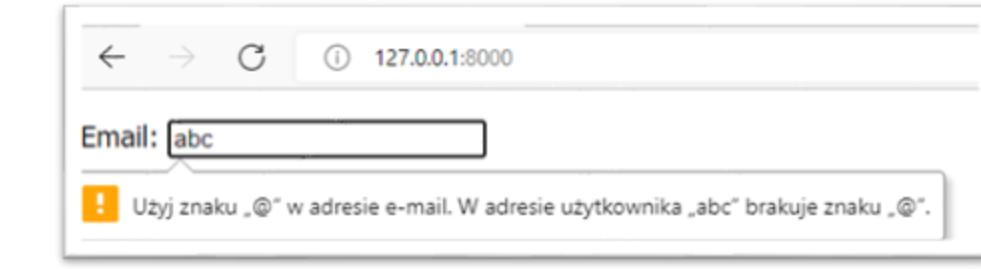
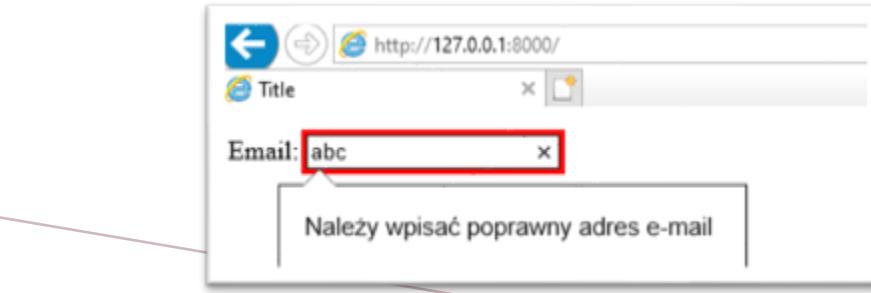
Przeglądarki internetowe, w celu zwiększenia własnej atrakcyjności wyposażają elementy formularza w **validatory danych**. Walidatory przed wysłaniem zapytania sprawdzają, czy dane umieszczone w elemencie formularza mają format zgodny z typem elementu. W przypadku błędного formatu dane nie są wysyłane, a użytkownikowi zostaje wyświetlony odpowiedni komunikat.

```
<form>
  <p>Email: <input type="email" name="userEmail"></p>
</form>
```



# ELEMENTY FORMULARZA - WALIDATORY

Sposób działania validatorów może różnić się pomiędzy przeglądarkami.

 <p>A screenshot of the Mozilla Firefox browser. The address bar shows '127.0.0.1:8000'. A form field labeled 'Email:' contains the value 'abc'. Below the field, a tooltip says 'Proszę wprowadzić adres e-mail'.</p>	
 <p>A screenshot of the Google Chrome browser. The address bar shows '127.0.0.1:8000'. A form field labeled 'Email:' contains the value 'abc'. Below the field, a tooltip says 'Użyj znaku „@” w adresie e-mail. W adresie użytkownika „abc” brakuje znaku „@”.'</p>	
 <p>A screenshot of the Microsoft Internet Explorer browser. The address bar shows 'http://127.0.0.1:8000/'. A form field labeled 'Email:' contains the value 'abc'. The field is highlighted with a red border. Below the field, a tooltip says 'Należy wpisać poprawny adres e-mail'.</p>	

*INPUT*

# INPUT

Element `<input>` to podstawowy element html służący do wprowadzania wartości takich jak tekst, data, wartość logiczna, itp.

Title  x +

← → C 127.0.0.1:8000

Username:  login <input type="text">

Password:  ..... <input type="password">

Gender: Male  Female  <input type="radio">

Enable features: Feature 1  Feature 2  <input type="checkbox">

submit <input type="submit">

reset <input type="reset">

dd, mm , yyyy <input type="date">

czerwiec 2022 <input type="button" value="czerwiec 2022" />

pon.	wt.	śr.	czw.	pt.	sob.	niedz.
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	1	2	3
4	5	6	7	8	9	10

# *TYPY ELEMENTU INPUT (HTML 5)*

<code>&lt;input type="text"&gt;</code>	<code>&lt;input type="tel"&gt;</code>	<code>&lt;input type="month"&gt;</code>	<code>&lt;input type="file"&gt;</code>	<code>&lt;input type="button"&gt;</code>
<code>&lt;input type="password"&gt;</code>	<code>&lt;input type="email"&gt;</code>	<code>&lt;input type="date"&gt;</code>	<code>&lt;input type="image"&gt;</code>	<code>&lt;input type="submit"&gt;</code>
<code>&lt;input type="hidden"&gt;</code>	<code>&lt;input type="url"&gt;</code>	<code>&lt;input type="datetime-local"&gt;</code>	<code>&lt;input type="radio"&gt;</code>	
<code>&lt;input type="number"&gt;</code>	<code>&lt;input type="time"&gt;</code>	<code>&lt;input type="search"&gt;</code>	<code>&lt;input type="checkbox"&gt;</code>	
<code>&lt;input type="range"&gt;</code>	<code>&lt;input type="week"&gt;</code>	<code>&lt;input type="color"&gt;</code>	<code>&lt;input type="reset"&gt;</code>	

## *PODSTAWOWA STRUKTURA ELEMENTU INPUT*

```
<input type="inputType" name="inputName" value="InitialValue"/>
```

*Element input posiada trzy podstawowe atrybuty:*

- *type – atrybut reprezentujący typ elementu (np. text, password, radio)*
- *name – atrybut reprezentujący unikalną nazwę elementu*
- *value - atrybut reprezentujący początkową wartość elementu*

# DODATKOWE ATRYBUTY ELEMENTU INPUT

Elementy input posiadają 10 wspólnych, dodatkowych atrybutów. Elementy input typu checkbox i radio posiadają ponadto atrybut checked. Rola dodatkowych atrybutów elementu input jest przeważnie rozszerzenie logiki walidującej wprowadzane dane.

disabled	min	required	value
max	pattern	size	checked (tylko dla checkbox i radio)
maxlength	readonly	step	

*SELECT*

```
!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <form>
        <select name="fruit">
            <option value="0">Arbus</option>
            <option value="1">Pomarańcza</option>
            <option value="2">Jabłko</option>
        </select>
        <input type="submit" value="Wyślij">
    </form>
</body>
```

## SELECT

Select to element formularza implementujący listę rozwijalną.

## *SELECT - WIDGET*

A screenshot of a web browser window titled "title". The address bar shows "127.0.0.1:8000". On the left, there is a dropdown menu with the value "Arbus" selected. To the right of the dropdown is a button labeled "Wyślij". Below the dropdown, a list of options is visible: "Arbus", "Pomarańcza", and "Jabłko".

*Select to element formularza  
implementujący listę rozwijalną*

# *SELECT*

```
<select name="fruit">  
    <option value="0">Arbus</option>  
</select>
```

Nazwa zmiennej  
Wartość zmiennej      Etykieta

*Struktura html/*

# ATRYBUTY ELEMENTU SELECT

- *selected - początkowo wybrana etykieta*
- *size – liczba widocznych wpisów. Domyślna wartość to 1. Dla wartości 1 po kliknięciu w element rozwija się lista. Dla wartości większej od 1 lista nie jest rozwijalna. Zamiast tego wyświetla się liczba wpisów odpowiadająca wartości atrybutu size z paskiem przewijania po prawej stronie.*
- *multiple - możliwość wybrania więcej niż jednej wartości (domyślnie z listy można wybrać tylko jedną wartość)*

The screenshot shows the PyCharm IDE interface with two code editors and a terminal window.

**Project Structure:** The left sidebar shows the project structure for `test_08`. The `views.py` file is open in the editor.

**Code Editor (views.py):**

```
from django.shortcuts import render, redirect
from django.http import HttpResponse

def index(request):
    if request.method == "GET":
        print(request.GET.get('fruit')) 2

    return render(
        request,
        'formapp/form.html'
    )
```

**Code Editor (form.html):**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <form>
        <select name="fruit">
            <option value="0">Arbus</option>
            <option value="1">Pomarańcza</option>
            <option value="2">Jabłko</option>
        </select>
        <input type="submit" value="Wyslij">
    </form>
</body>
```

**Terminal:**

```
[01/May/2022 12:42:50] "GET /?fruit=0 HTTP/1.1" 200 386
None
[01/May/2022 12:43:05] "GET / HTTP/1.1" 200 386
0 3
[01/May/2022 12:43:07] "GET /?fruit=0 HTTP/1.1" 200 386
```

**Annotations:**

- Red Number 2:** Above the line `print(request.GET.get('fruit'))`.
- Red Number 1:** Above the line `<option value="0">Arbus</option>`.
- Red Number 3:** Above the line `0` in the terminal output.

*SELECT  
(WDJANGO)*

## *SELECTZ ATRYBUTEM MULTIPLE*

A screenshot of a web browser window titled "Title". The address bar shows the URL "127.0.0.1:8000". On the left, there is a multiple select dropdown menu containing three options: "Arbus", "Pomarańcza", and "Jabłko". The option "Arbus" is currently selected and highlighted in blue. To the right of the dropdown is a "Wyślij" (Send) button.

*Atrybut multiple umożliwia wybór wielu elementów*

The screenshot shows a PyCharm IDE interface with two code editors and a terminal window.

**Left Editor (views.py):**

```
from django.shortcuts import render, redirect
from django.http import HttpResponse

def index(request):
    if request.method == "GET":
        print(request.GET.get('fruit'))

    return render(
        request,
        'formapp/form.html'
    )
```

**Right Editor (form.html):**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <form>
        <select name="fruit" multiple>
            <option value="0">Arbus</option>
            <option value="1">Pomarańcza</option>
            <option value="2">Jabłko</option>
        </select>
        <input type="submit" value="Wyślij">
    </form>
</body>
```

**Terminal Output:**

```
[01/May/2022 14:25:42] "GET /?fruit=0 HTTP/1.1" 200 395
None
[01/May/2022 14:26:38] "GET / HTTP/1.1" 200 395
2
[01/May/2022 14:26:41] "GET /?fruit=0&fruit=2 HTTP/1.1" 200 395
```

*SELECTZ ATRYBUTEM MULTIPLE - GET  
(METODA GET ZWRACA OSTATNIE ELEMENT LISTY)*

The screenshot shows the PyCharm IDE interface with two code editors open. The left editor displays `views.py` and the right editor displays `form.html`. The `views.py` file contains Python code for a Django view:

```
from django.shortcuts import render, redirect
from django.http import HttpResponse

def index(request):
    if request.method == "GET":
        print(request.GET.getlist('fruit'))

    return render(
        request,
        'formapp/form.html'
    )
```

The `request.GET.getlist('fruit')` line is highlighted with a red underline. The `form.html` file contains HTML code for a form:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <form>
        <select name="fruit" multiple>
            <option value="0">Arbus</option>
            <option value="1">Pomarańcza</option>
            <option value="2">Jabłko</option>
        </select>
        <input type="submit" value="Wyślij">
    </form>
</body>
```

The `multiple` attribute of the `<select>` element and the `value` attribute of the first `<option>` element are highlighted with red underlines. Below the code editors, the terminal window shows the server logs:

```
Quit the server with CTRL-BREAK.
[]
[01/May/2022 14:28:02] "GET / HTTP/1.1" 200 395
['0', '2']
[01/May/2022 14:28:06] "GET /?fruit=0&fruit=2 HTTP/1.1" 200 395
```

**SELECTZ ATRYBUTEM MULTIPLE - GETLIST**  
(METODA GETLIST ZWRACA WSZYSTKIE ZAZNACZONE ELEMENTY)

*TEXTAREA*

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <form>
        <textarea name="foo"></textarea>
        <input type="submit" value="Wyślij">
    </form>
</body>
</html>
```

## TEXTAREA

*<textarea> to element formularza implementujący wielolinijkowe pole tekstowe.*

## *TEXTAREA*

A screenshot of a web browser window. The title bar says "Title". The address bar shows "127.0.0.1:8000". The main content area contains a text area with the following text:

```
To jest mój  
wielolinijkowy napis
```

The word "wielolinijkowy" is underlined with a red wavy line, indicating it might be misspelled. There is a small icon of a document with a diagonal line through it next to the text area. To the right of the text area is a blue button labeled "Wyślij" (Send).

*<textarea>* to element formularza implementujący wielolinijkowe pole tekstowe.

The screenshot shows the PyCharm IDE interface with two code editors and a terminal window.

**Code Editors:**

- views.py**: Python code defining a view named `index`. It prints the value of the `'foo'` parameter from the GET request.
- form.html**: HTML template containing a form with a text area and a submit button labeled "Wyślij".

**Terminal:**

- Output: "None"
- Log:
  - [01/May/2022 15:03:26] "GET / HTTP/1.1" 200 236
  - To jest mój  
wielolinijkowy napis
  - [01/May/2022 15:03:28] "GET /?foo=To+jest+m%C3%B3j%0D%0Awielolinijkowy+napis HTTP/1.1" 200 236

# TEXTAREA

(WDJANGO)

# *ATRYBUTY ELEMENTU TEXTAREA*

- *rows* - liczba wierszy pola tekstowego (wysokość pola). Domyślna wartość atrybutu rows różni się pomiędzy przeglądarkami.
- *cols* – liczba kolumn pola tekstowego (szerokość pola). Domyślna wartość atrybutu cols różni się pomiędzy przeglądarkami.

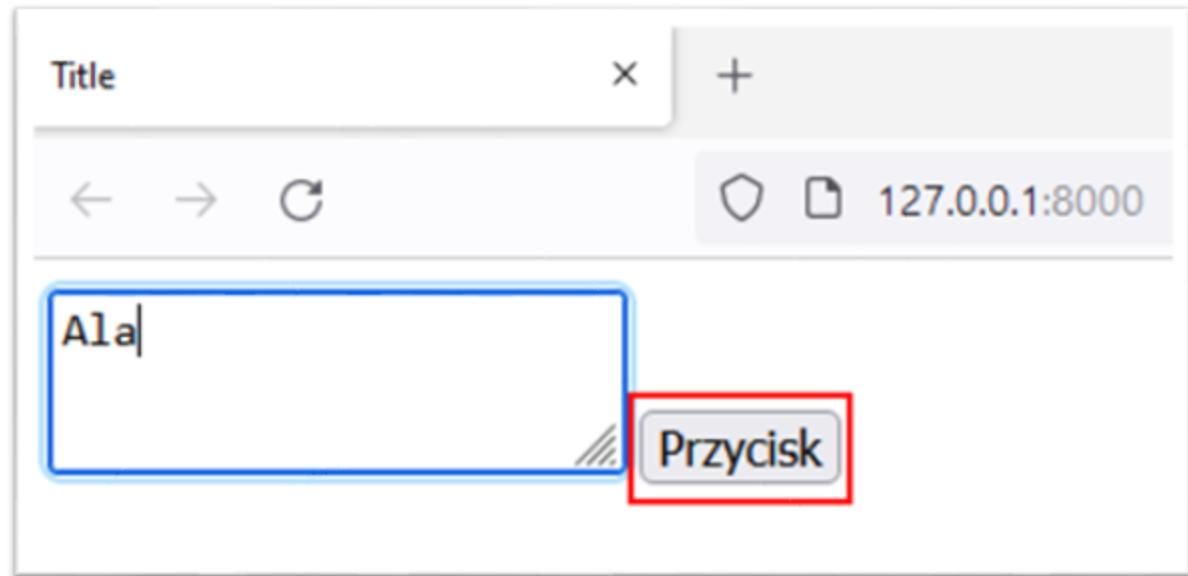
*BUTTON*

# BUTTON

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <form>
        <textarea name="my_str"></textarea>
        <button name="foo" value="pressed">Przycisk</button>
    </form>
</body>
</html>
```

*<button>* to element formularza implementujący "klikalny" przycisk w formularzu.

# BUTTON



*<button>* to element formularza implementujący "klikalny" przycisk w formularzu.

The screenshot shows the PyCharm IDE interface with two code editors and a terminal window.

**Project Structure:** The left sidebar shows the project structure for `test_08`, including `config`, `formapp` (with `migrations`, `templates`, and `views.py`), and `venv`.

**Code Editors:**

- views.py:** Contains Python code for a `index` view. It imports `django.shortcuts` and defines a `index` function that prints the `request.GET` object if the method is `GET`. It then renders the `'formapp/form.html'` template.
- form.html:** Contains an HTML template with a `<form>` block. Inside the form, there is a `<textarea name="my_str"></textarea>` and a `<button name="foo" value="pressed">Przycisk</button>`.

**Terminal:** The bottom terminal window shows the server logs:

```
Quit the server with CTRL-BREAK.  
<QueryDict: {}>  
[01/May/2022 20:29:08] "GET / HTTP/1.1" 200 254  
<QueryDict: {'my_str': ['Ala'], 'foo': ['pressed']}>  
[01/May/2022 20:29:12] "GET /?my_str=Ala&foo=pressed HTTP/1.1" 200 254
```

*BUTTON*  
(*WDJANGO*)

# ATRYBUTY ELEMENTU BUTTON

- *type - typ przycisku. Domyślna wartość atrybutu type to "submit". Dostępne wartości atrybutu to: "button", "submit", "reset".*
  - *submit - przycisk wysyłający na serwer wartości wszystkich pól formularza.*
  - *reset - przycisk resetujący wszystkie wartości wprowadzone w polach formularza (bez komunikacji z serwerem).*
  - *button - typ domyślny. Przycisk do podpinania pod jego zdarzenia odpowiednich funkcji js. Domyślnie po kliknięciu w <button type="button"> nic się nie dzieje.*

*DATALIST*

```
<title>Title</title>
</head>
<body>
<form>
  <input list="colors" name="color">
  <datalist id="colors">
    <option>czerwony</option>
    <option>niebieski</option>
    <option>żółty</option>
    <option>zielony</option>
    <option>czarny</option>
  </datalist>
  <input type="submit" value="Wyślij">
</form>
</body>
</html>
```

## DATALIST

*<datalist>* to element formularza łączący w sobie element *<input>* oraz *<select>*. Dodatkowym atutem elementu jest wyświetlanie na liście wyłącznie tych wartości, które zawierają wprowadzoną w elemencie *<input>* frazę.

## DATALIST

A screenshot of a web browser window titled "Title". The address bar shows "127.0.0.1:8000". Below the address bar is a search input field containing "żółty" (yellow). To the right of the input field is a button labeled "Wyślij" (Send). A dropdown menu is open, listing the following color names: "czerwony" (red), "niebieski" (blue), "żółty" (yellow), "zielony" (green), and "czarny" (black). The word "żółty" is highlighted in blue, indicating it is the selected value.

`<datalist>` to element formularza łączący w sobie element `<input>` oraz `<select>`. Dodatkowym atutem elementu jest wyświetlanie na liście wyłącznie tych wartości, które zawierają wprowadzoną w elemencie `<input>` frazę.

## *DATALIST*

The screenshot shows a web browser window with a title bar labeled "Title". Below the title bar is a toolbar with navigation icons (back, forward, search) and a status bar showing the URL "127.0.0.1:8000". The main content area contains a form. On the left is a text input field with the placeholder "cz" and a dropdown menu below it listing "czerwony" and "czarny". To the right of the input field is a button labeled "Wyślij".

*<datalist>* to element formularza łączący w sobie element *<input>* oraz *<select>*. Dodatkowym atutem elementu jest wyświetlanie na liście wyłącznie tych wartości, które zawierają wprowadzoną w elemencie *<input>* frazę.

The screenshot shows the PyCharm IDE interface with two open files: `views.py` and `form.html`.

**views.py:**

```
from django.shortcuts import render

def index(request):
    if request.method == "GET":
        print(request.GET)

    return render(
        request,
        'formapp/form.html'
    )
```

**form.html:**

```
<title>Title</title>
</head>
<body>
<form>
    <input list="colors" name="color">
    <datalist id="colors">
        <option>czerwony</option>
        <option>niebieski</option>
        <option>żółty</option>
        <option>zielony</option>
        <option>czarny</option>
    </datalist>
    <input type="submit" value="Wyślij">
</form>
</body>
</html>
```

A red box highlights the `<form>` block in the `form.html` file.

**Terminal Output:**

```
Not Found: /favicon.ico
[01/May/2022 21:22:00] "GET /favicon.ico HTTP/1.1" 404 2271
<QueryDict: {'color': ['czerwony']}>
[01/May/2022 21:22:50] "GET /?color=czerwony HTTP/1.1" 200 477
```

*BUTTON  
(WDJANGO)*

## *SELECT*

A screenshot of a web browser window titled "Title". The address bar shows "127.0.0.1:8000". Below the address bar is a dropdown menu with the following options:

- Arbus
- Arbus
- Pomarańcza
- Jabłko

The option "Arbus" is selected and highlighted with a blue border. To the right of the dropdown menu is a button labeled "Wyślij" (Send).

*Select to element formularza implementujący listę rozwijalną.*

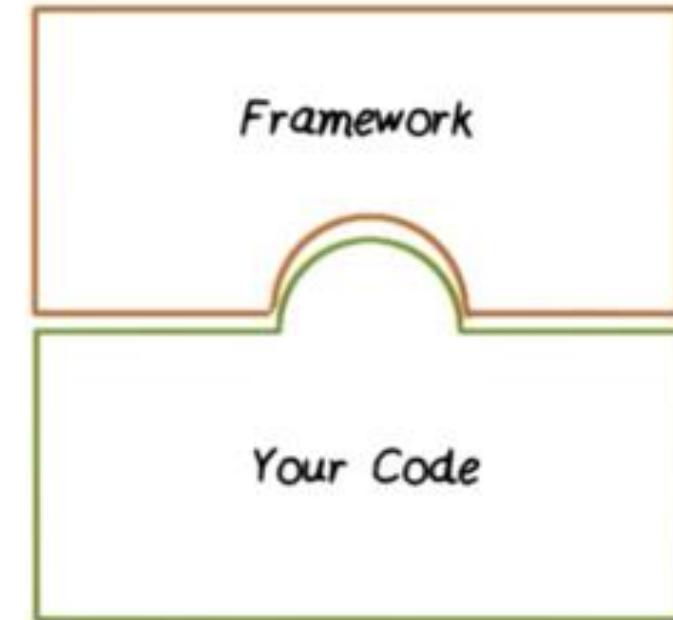
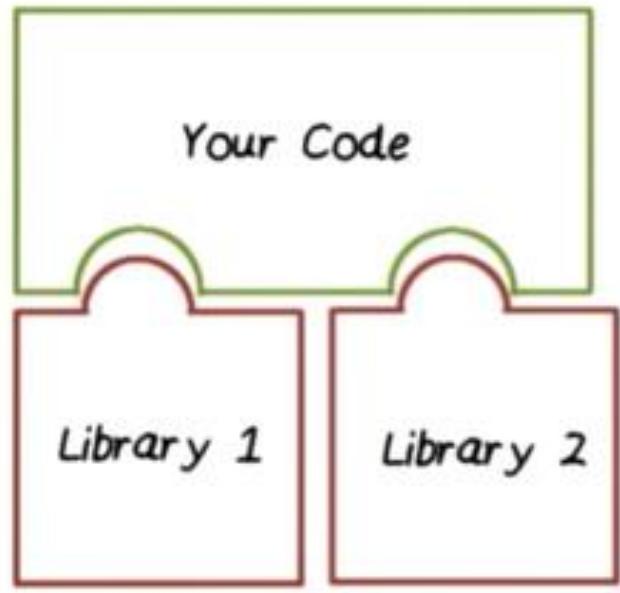
# *ATRYBUTY ELEMENTU DATALIST*

- *list - atrybut elementu <input> wskazujący na listę powiązaną z danym inputem.*  
*Wartość atrybutu powinna być identyczna z wartością atrybutu id elementu <datalist>, który chcemy powiązać z inputem*
- *id – unikalny identyfikator elementu <datalist>*

# *DATALIST VS SELECT*

- *Element <datalist> w odróżnieniu od elementu <select> pozwala na wprowadzenie i przesłanie wartości nie znajdującej się na liście. Jeżeli chcemy ograniczyć użytkownikowi zbiór wartości to należy użyć elementu <select>.*
- *Element <datalist> w odróżnieniu od elementu <select> ogranicza listę wyświetlanych wartości do wartości odpowiadających wprowadzanej przez użytkownika frazie.*

# *FRAMEWORK VS LIBRARY*



*ODWRÓCENIE STEROWANIA*

*(INVERSION OF CONTROL)*

Używanie biblioteki



Używanie frameworka



Twój kod Zewnętrzny kod

Kamil.Kwapisz.pl

*ODWRÓCENIE STEROWANIA  
(INVERSION OF CONTROL)*

*URI, URL, URN*

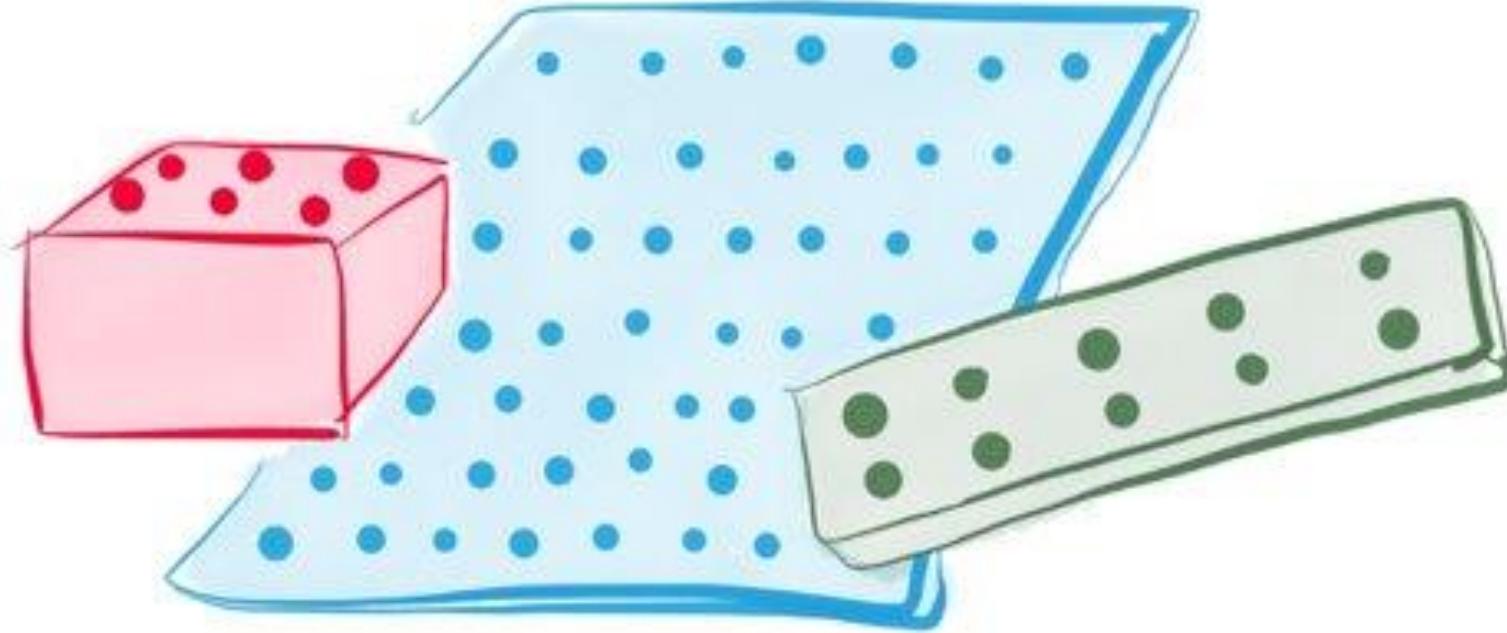
# ***URI (UJEDNOLICONY IDENTYFIKATOR ZASOBÓW)***

Rodzaje URI (Uniform Resource Identifier):

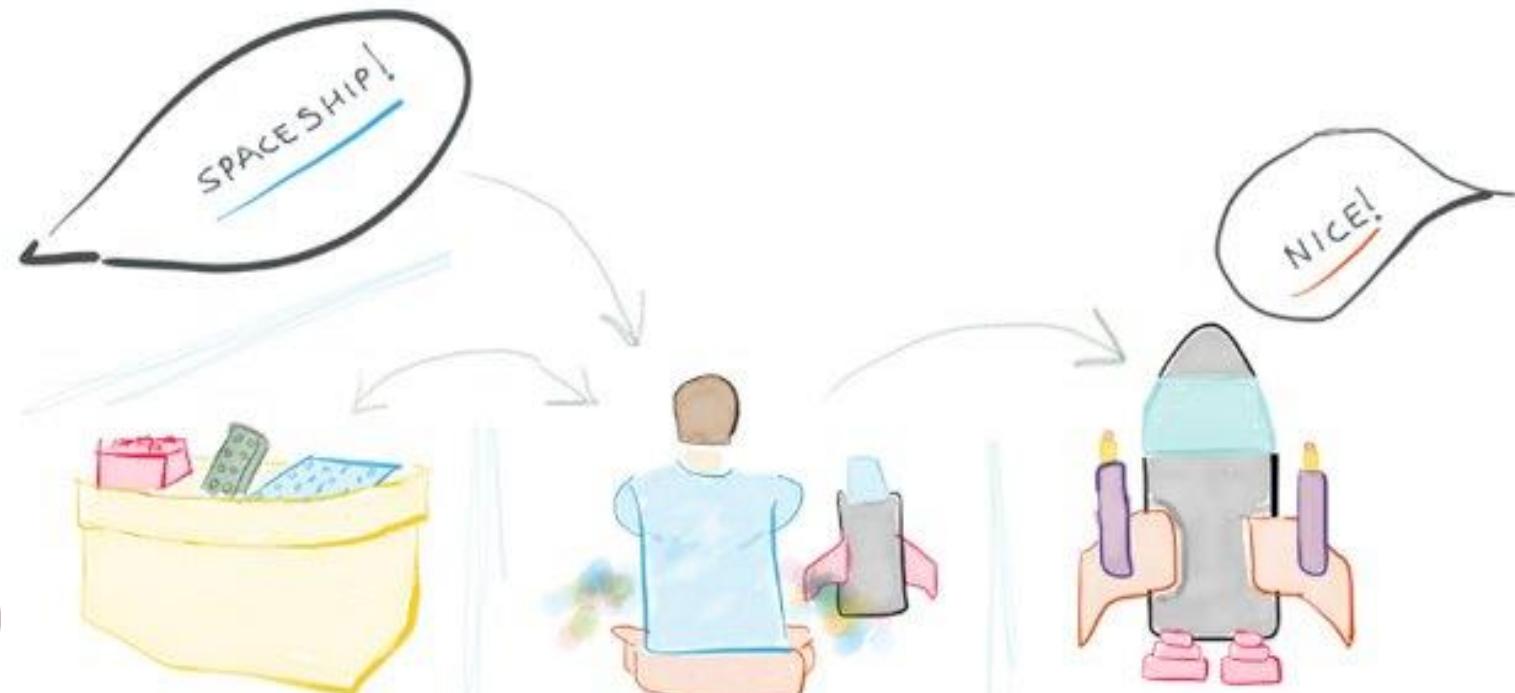
- URL – Uniform Resource Locator
- URI – Uniform Resource Identifier
- URN – Uniform Resource Name



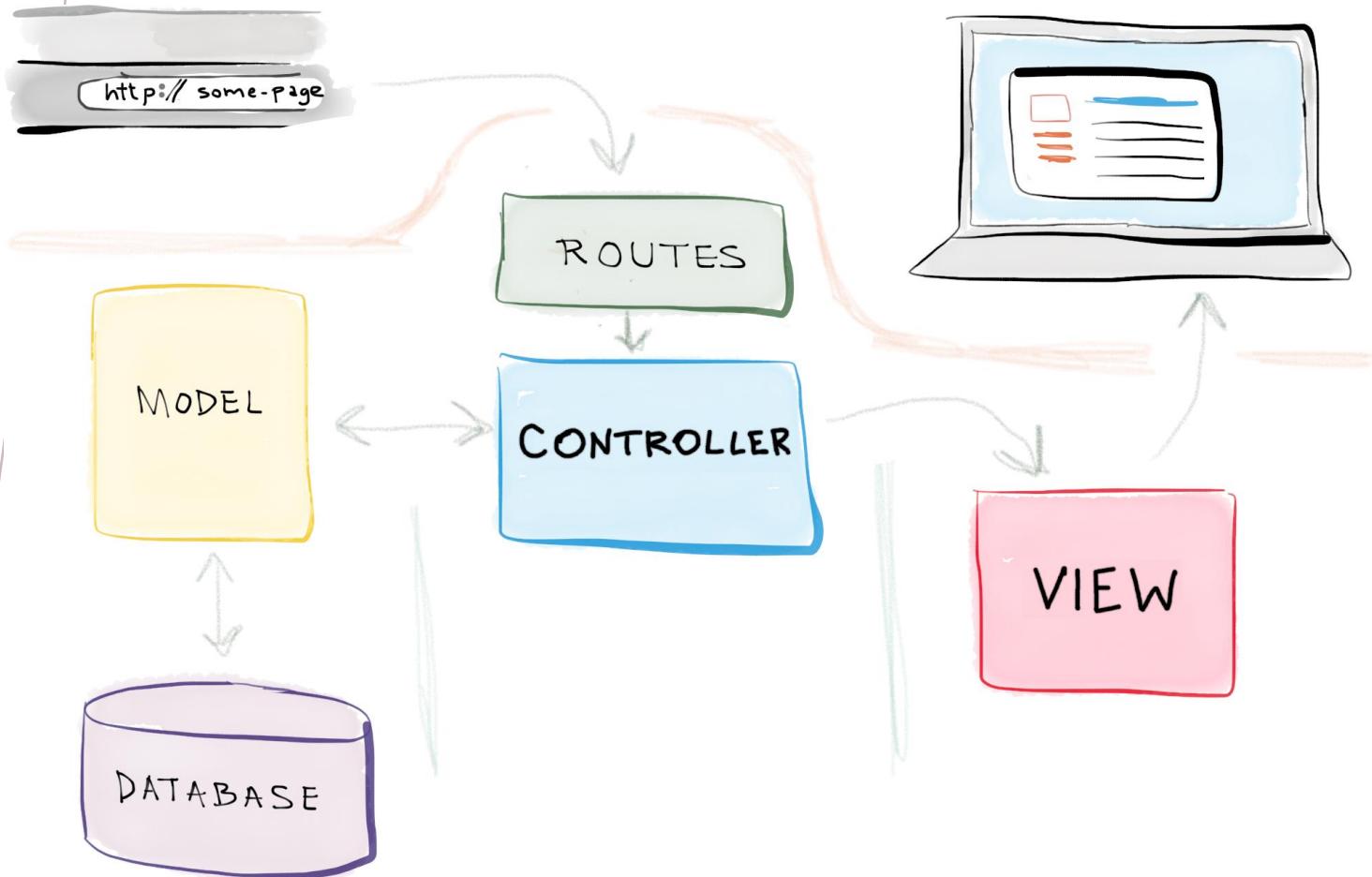
*WZORCE MVC IMT*



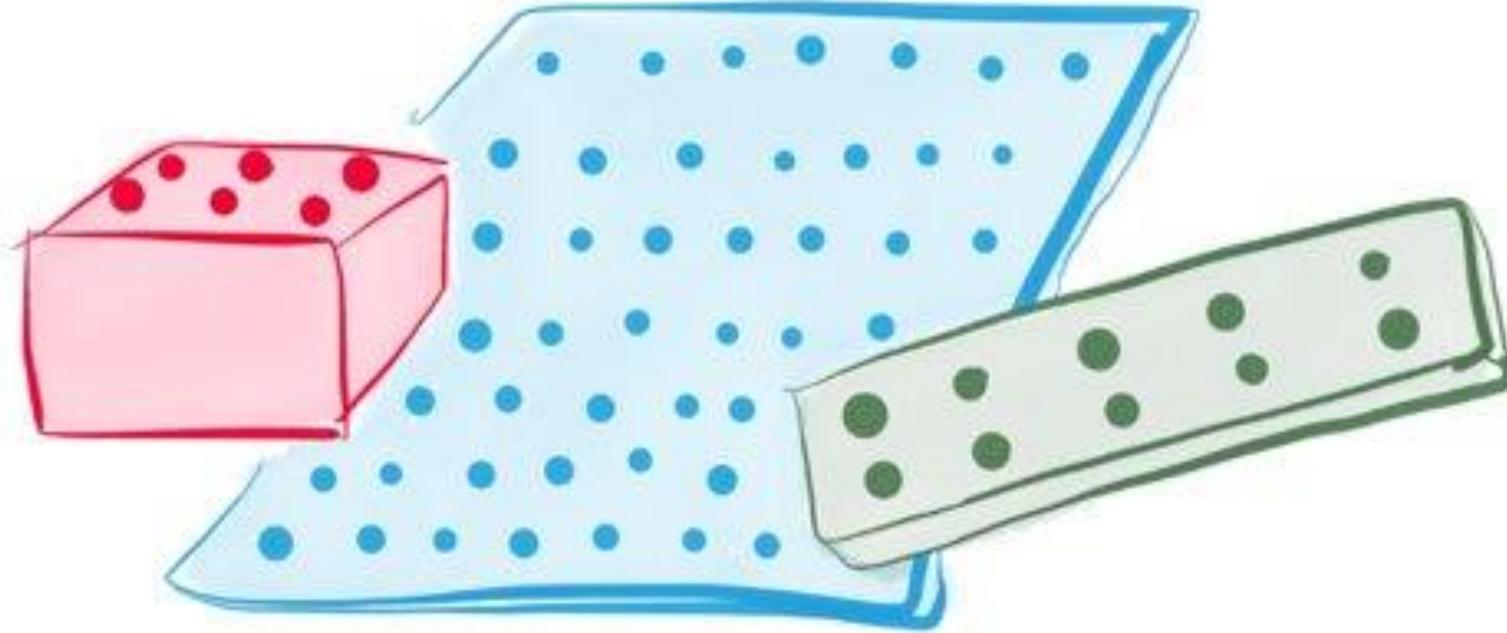
*MVC(MODEL-VIEW-CONTROLLER*



*MVC*



*MVC*



***MTV(MODEL-TEMPLATE-VIEW)***  
*AKA MVT(MODEL-VIEW-TEMPLATE)*

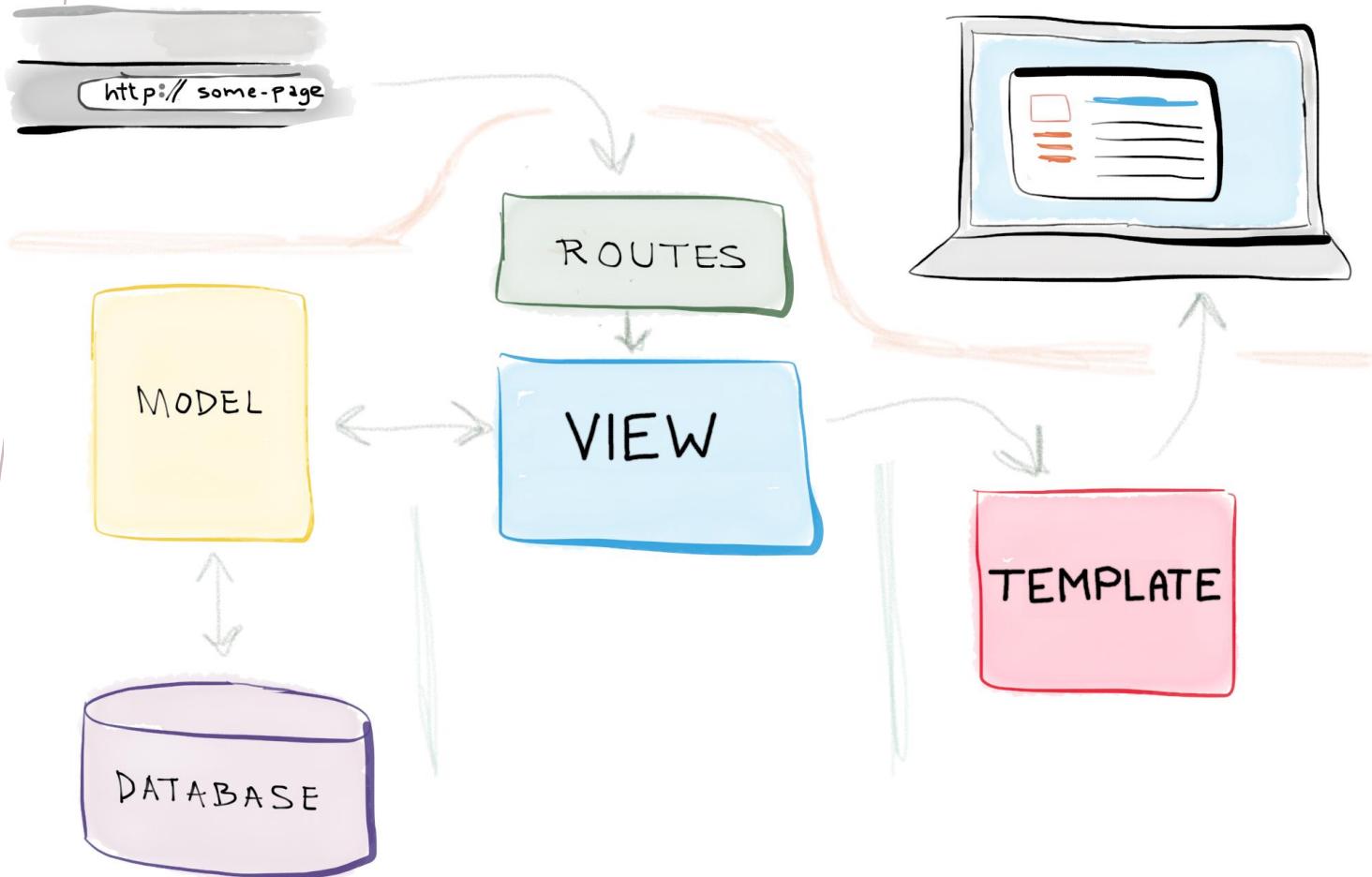
# *WZORZEC MVC A WZORZEC MTV*

W Django widokami (views) nazywamy coś innego niż we wzorcu MVC. Widoki Django to we wzorcu MVC kontrolery, a szablony Django to we wzorcu MVC widoki.

Stąd często spotykana zmiana liter w akronimie:

$$\begin{array}{ccc} V \text{ (w mvc)} & \longleftrightarrow & T \text{ (w mtv)} \\ C \text{ (w mvc)} & \longleftrightarrow & V \text{ (w mtv)} \end{array}$$

Obie nazwy określają jednak ten sam wzorzec projektowy.



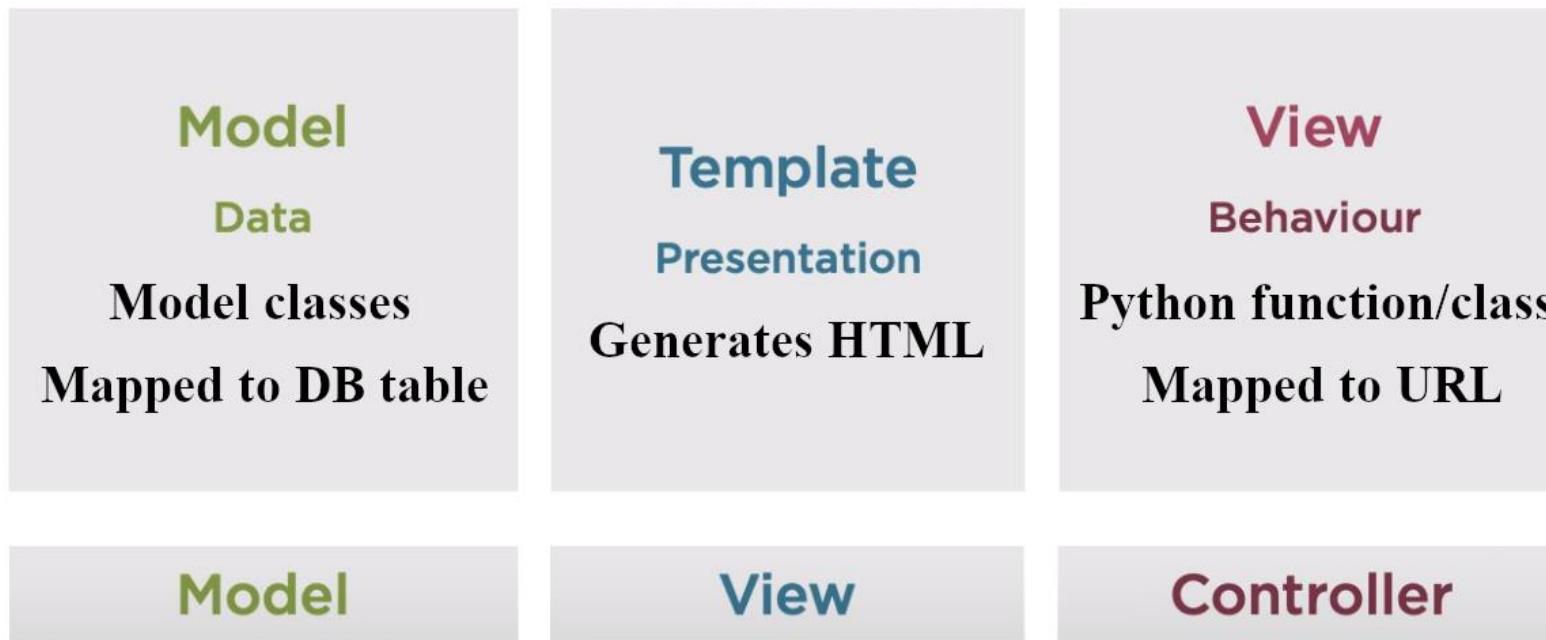
*MTV*

## Model-Template-View



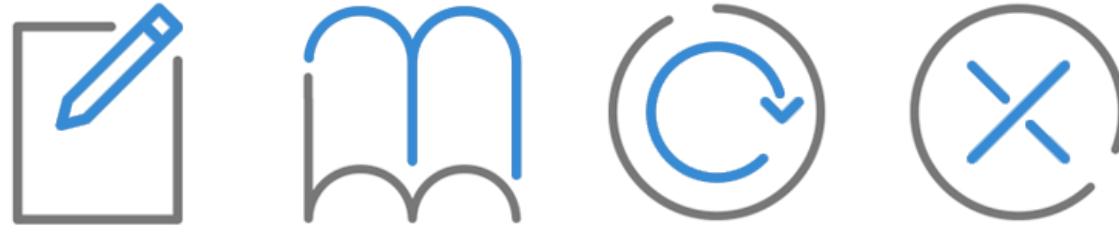
*MTV*

## Model-Template-View



*MTV*

*CRUD*



CREATE      READ      UPDATE      DELETE

---

**C R U D**

*CRUD*



*FLASK*

**ZADANIA**

# ZADANIE 1

Stwórz nową aplikację Flask o nazwie **draw**.

Endpoint aplikacji: <http://127.0.0.1:5000/draw/>

Aplikacja powinna posiadać widok na endpointie:

<http://127.0.0.1:5000/draw/toto-lotek/>

Widok wyświetla szablon draw/toto-lotek.html

W szablonie powinno wyświetlać się sześć losowo wybranych liczb z zakresu 1-49. Do generowania losowych liczb użyj biblioteki random.

# ZADANIE 2

Stwórz nową aplikację Flask o nazwie **mem**.

Endpoint aplikacji: <http://127.0.0.1:5000/mem/>

Aplikacja powinna posiadać widok na endpointie:

<http://127.0.0.1:5000/mem/sample/>

Widok wyświetla szablon mem/sample.html

Szablon powinien wyświetlać jeden, wybrany mem/ilustracje.

Ilustracje należy pobrać z internetu i umieścić w odpowiednim folderze. Dodatkowo szablon powinien posiadać arkusz stylów. Arkusz stylów powinien definiować podstawowy styl dla ilustracji (np. wyczentrowanie).

# ZADANIE 3

Stwórz widok na endpointie:

<http://127.0.0.1:5000/calculator/add/>

Widok wyświetla formularz z dwoma polami do wprowadzenia dwóch liczb. Formularz do wysłania danych używa metody get. Po wprowadzeniu liczb i wysłaniu formularza wyświetlony zostaje szablon calculator/add.html

Szablon wyświetla napis:

<value1> + <value2> = <wynik dodawania value1 i value2>

# ZADANIE 4

Napisz stronę, która wczyta z atrybutu POST dwie zmienne: **start** i **end** (dla uproszczenia załóżmy, że będziemy przesyłać tylko liczby całkowite).

Następnie strona wypisze wszystkie liczby od **start** do **end**. Do wyświetlenia kolejnych liczb użyj znacznika **for** języka szablonów Jinja.

# ZADANIE 5

Używając Flask-SQLAlchemy stwórz w bazie tabelę **Band** z kolumnami:

- name – nazwa zespołu, wartość znakowa o maksymalnej długości 64
- year – rok założenia zespołu
- still\_active – informacja o tym czy zespół jest jeszcze aktywny (pole powinno przyjmować typ boolean z wartością domyślną True)

# ZADANIE 6

Używając Flask-SQLAlchemy stwórz w bazie tabelę **Category** z kolumnami:

- name – wartość znakowa o maksymalnej długości 64
- description – nieograniczonej długości napis, może przyjmować pustą wartość

# ZADANIE 7

Stwórz model **Article** z polami:

- title – napis o maksymalnej długości 128 znaków
- author – napis o maksymalnej długości 64 znaków, może przyjmować pustą wartość
- content – nieograniczonej długości napis
- date\_added – pole typu datetime, wartość automatycznie uzupełniania w chwili dodania nowego wpisu (auto\_now\_add=True)

# ZADANIE 8

Stwórz model Album z polami przechowującymi informacje o:

- tytule albumu
- roku wydania
- średniej ocenami

# ZADANIE 11

Używając Flask-SQLAlchemy wprowadź do tabeli z zadania 5 informacje o swoich trzech ulubionych zespołach.

# Golden thoughts

Projekt Django 1

# Projekt - część 1

Zawsze dobrze jest uprzyjemnić komuś dzień mądrą sentencją. Zbuduj aplikację wyświetlającą użytkownikowi losowo wybraną myśl z zapisanej w bazie listy złotych myśli. Za każdym razem kiedy Twoja strona zostanie załadowana, aplikacja Django powinna w widoku wybrać jeden, losowy wpis z kilkunastu znajdujących się w bazie sentencji i przekazać go do szablonu.

W tej części sentencje możesz zapisać w bazie ręcznie.

# Projekt - część 2

Dorób do istniejącej aplikacji pełny interfejs CRUD dla użytkownika, tzn. poza widokiem wyświetlającym widok z losowo wybraną sentencją z bazy, aplikacja powinna posiadać dostępny dla użytkownika widok z formularzem do dodawania sentencji, widok listy sentencji, widok szczegółu sentencji, widok z formularzem do modyfikacji sentencji i widok do usuwania sentencji.