

პროგრამირება

Python

ლექცია 11-15: **Web Frameworks** - დანიშნულება და გამოყენების
უპირატესობები; **Flask**-ის სტრუქტურა, ვებ აპლიკაციის არქიტექტურა

ლიკა სვანაძე

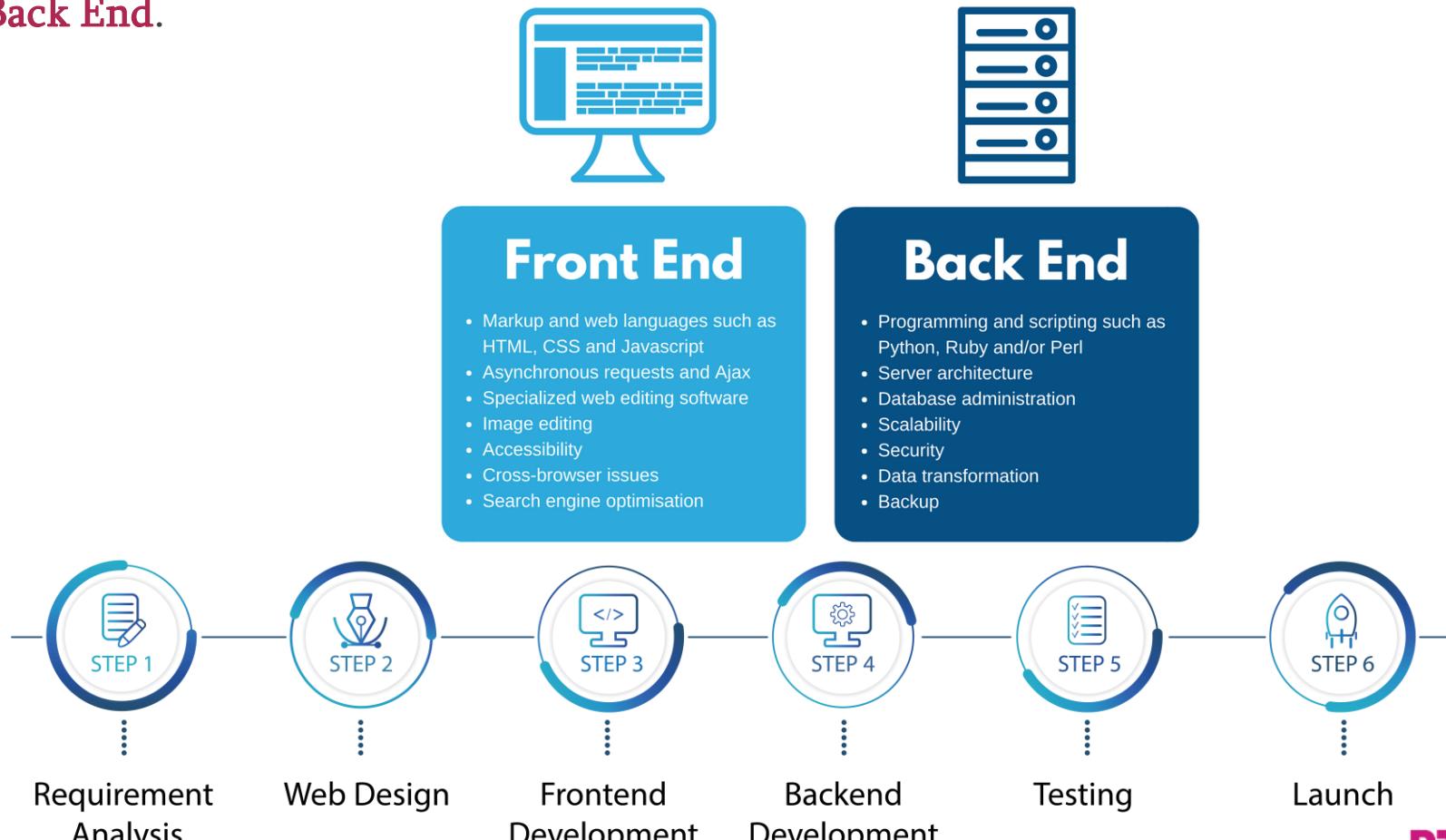
lika.svanadze@btu.edu.ge

განსახილველი საკითხები

- * Web Frameworks (Flask, Django)
- * Flask architecture - First web page
- * Templates/შაბლონები
- * Template inheritance, CSS library
- * HTTP Requests – GET, POST; Web Forms
- * Authorization, Session
- * მონაცემთა ბაზები: SQLAlchemy
- * Static folders (css&js files, images)
- * Flash ფუნქცია
- * Deployment (სერვერზე განთავსება)

Web

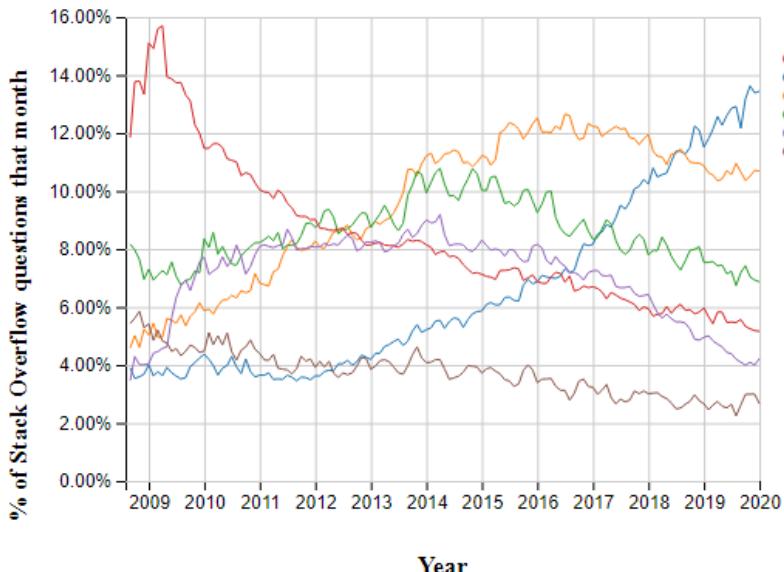
* ვებ საიტების დეველოპენტი დაყოფილია 2 ძირითად ნაწილად: **Front End** და **Back End**.



Web

* **Back End**-ის დეველოპმენტის დროს გამოიყენება სხვადასხვა

პროგრამირების ენა; უმეტესად: PHP, C# (.NET), Python, Java, Ruby და სხვ.



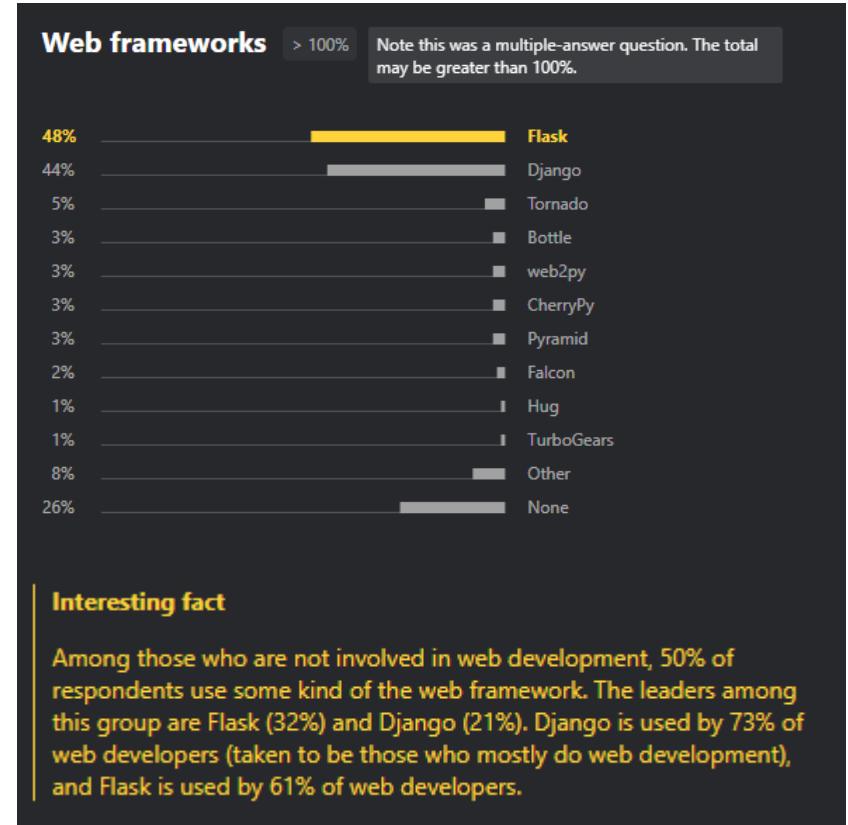
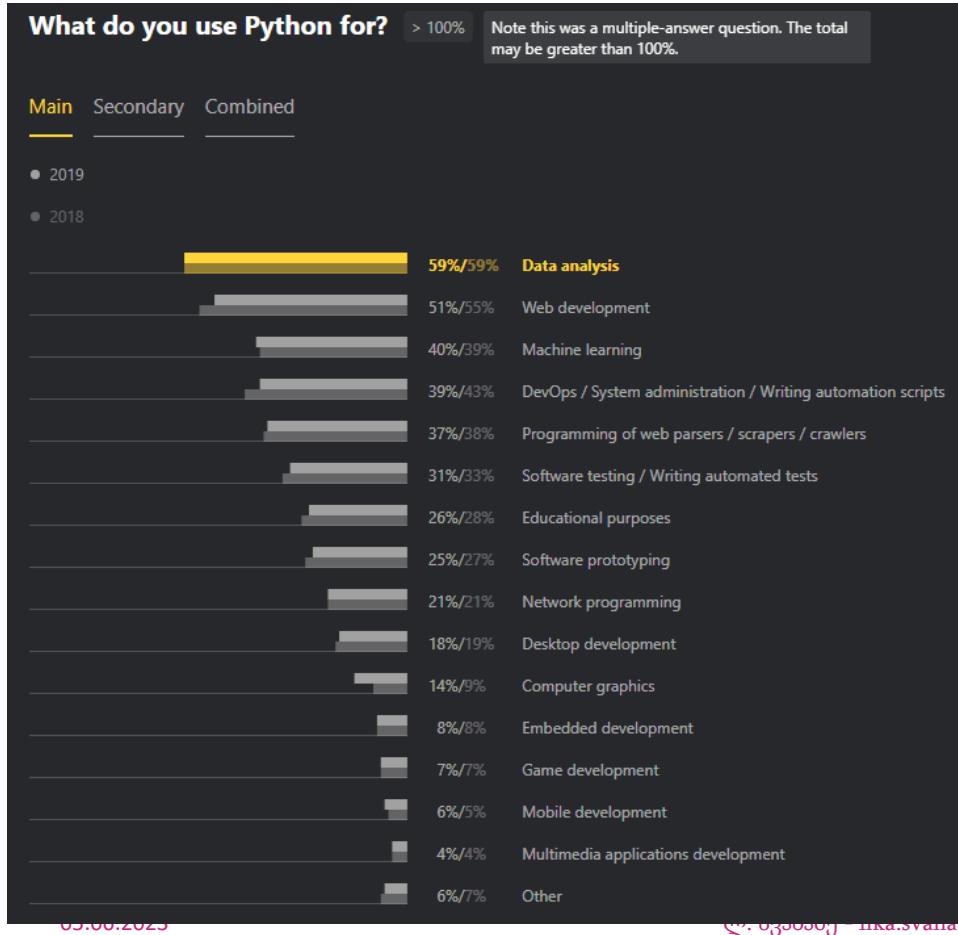
პითონის Web Frameworks

- * პითონში არსებული მოდულების/პაკეტების გამოყენებით შესაძლებელია ვებ აპლიკაციების/სერვისების აგება.
- * არსებობს ორი ტიპის ვებ ფრეიმვორქი:
 - **Full-Stack Frameworks** - რომელიც მოიცავს სრულ პაკეტს: სერვერის მხარეს, ORM-ს (Object-relational mapping - მონაცემთა ბაზებს), ავტორიზაციის მოდულს, ajax toolkit-ს. ესეთი ფრეიმვორქებია: **Django**, **TurboGears**, **web2py**, სხვ.
 - **Micro (Non Full-Stack) Frameworks** – რომელიც მოიცავს საბაზისო აპლიკაციის სერვერს, ხოლო დამატებითი კომპონენტების საჭიროების შემთხვევაში უნდა მოხდეს შესაბამისი ბიბლიოთეკის (extensions) გამოყენება. მიკრო ფრეიმფორქებია: **Flask**, **Bottle**, **CherryPy**, **Falcon**, სხვ.



Python Developers Survey 2019 Results

- კვლევაში მონაწილეობდა 24 000 Python დეველოპერი 150-ზე მეტი ქვეყნიდან
- გამოვითხვის ლინკი: <https://www.jetbrains.com/lp/python-developers-survey-2020/>



Flask



- * Flask ვებ ფრეიმვორქი შეიქმნა 2010 წლის 1 აპრილს Werkzeug და Jinja2-ის ბაზაზე, რომელიც წარმოადგენენ პითონის ბიბლიოთეკებს.
- * Werkzeug - წარმოადგენს WSGI (Web Server Gateway Interface) ბიბლიოთეკას, რომელიც გამოიყენება ვებ აპლიკაციებთან სამუშაოდ; მისი საშუალებით შესაძლებელია ვებ აპლიკაციების შექმნა, http request და response-ის მართვა.
- * Jinja - წარმოადგენს ბიბლიოთეკას, რომელიც გამოიყენება template-ების (შაბლონების) დასაგენერირებლად - შაბლონების გენერატორი. მისი საშუალებით შესაძლებელია html ფაილის დარენდერება, რომელშიც ჩართულია Python-ის კოდიც.
- * Flask ბიბლიოთეკის დამატება შესაძლებელია შემდეგი ბრძანებით:

```
pip install flask
```

- * ოფიციალური საიტი: <https://flask.palletsprojects.com/en/2.0.x/>

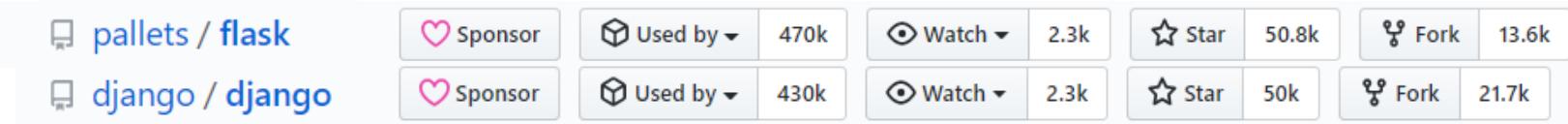
Flask VS Django

Learning Flask first will allow you to learn Django much faster



Framework-ის ტიპი	WSGI framework	Full Stack Web Framework
ORM	SQLAlchemy ბიბლიოთეკა	Built-in ORM
დანიშნულება	მცირე ზომის ვებ აპლიკაციებისთვის, ბლოგებისთვის, ვებ სერვისებისთვის, API-სთვის	კომპლექსური ვებ სისტემებისთვის
Community Support and Documentation	✓ ✓ ✓	✓ ✓ ✓ ✓
Flexibility	✓ ✓ ✓ ✓	✓ ✓
Used by:	Pinterest API, Red Hat, Twilio, Netflix, Uber, LinkedIn, Airbnb	Pinterest, Disqus, Eventbrite, Instagram, Bitbucket
In a nutshell	"web development, one drop at a time"	"web framework for perfectionists with deadlines"

Popularity on GitHub:



* სტატია: Python and Netflix: <https://www.edureka.co/blog/how-netflix-uses-python/>

ვებ აპლიკაციის შექმნა

- * ვებ აპლიკაციის გაშვება ხორციელდება სევერზე (ჩვენს შემთხვევაში ლოკალურ სერვერზე), რომლის მისამართია <http://127.0.0.1:5000> - ეს მისამართი წარმოადგენს საიტის მთავარ გვერდს. ეს ლინკი იგივეა რაც: <http://localhost:5000>

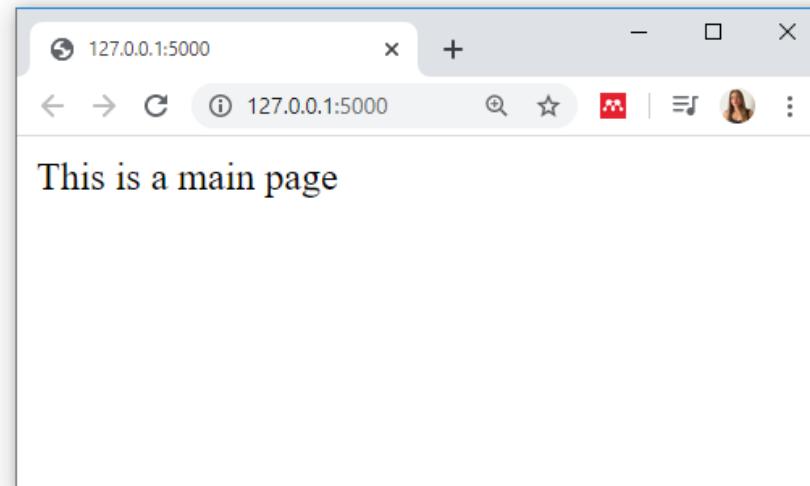
Python

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

@app.route('/')
def home():
    return 'This is a main page'

if __name__ == '__main__':
    app.run(debug=True)
```

Browser-ში:



3ებ ዓპლივაციის შექმნა

Python

```
from flask import Flask

app = Flask(__name__)

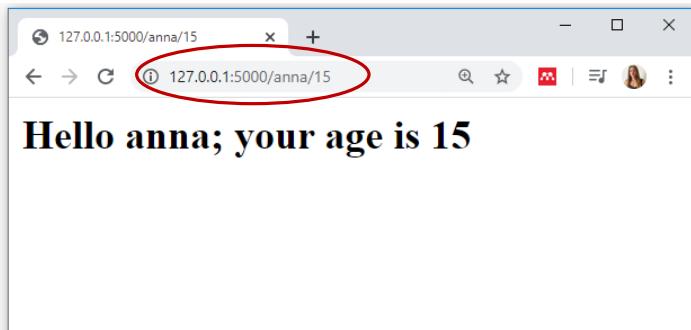
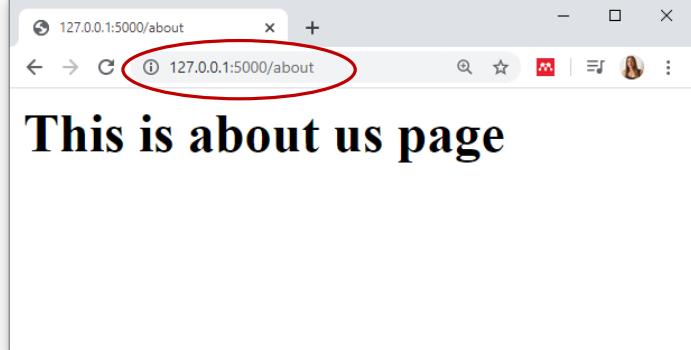
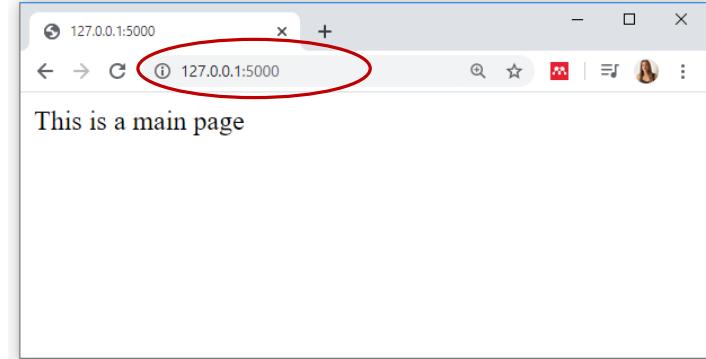
@app.route('/')
def home():
    return 'This is a main page'

@app.route('/about')
@app.route('/aboutus')
def about():
    return '<h1>This is about us page</h1>'

@app.route('/<name>/<int:age>')
def user(name, age):
    return f'<h2>Hello {name}; your age is {age}</h2>'

if __name__ == '__main__':
    app.run(debug=True)
```

შედეგი



გადამისამართება სხვა გვერდზე

- * url_for() ფუნქცია უზრუნველყოფს ლინკის დაგენერირებას. პარამეტრად გადაეცემა სტრიქონი, რომელიც წარმოადგენს იმ ფუნქციის დასახელებას, რომლის შესაბამისი ლინკის აგებაც გვსურს. ფუნქციას შესაძლოა გადაეცეს სხვა პარამეტრებიც, იმ შემთხვევაში თუ პირველ პარამეტრად მითითენულ ფუნქცია არის პარამეტრიანი ფუნქცია, ყველა მისი პარამეტრი შეგვიძლია გადავცეთ url_for()-ში მომდევნო პარამეტრებად (მაგალითი განხილულია მომდევნო სლაიდებზე).
- * redirect() ფუნქცია უზრუნველყოფს პარამეტრად გადაცემულ ლინკზე გადასვლას (გადამისამართებას).

Python

```
from flask import Flask, redirect, url_for

app = Flask(__name__)

@app.route('/')
def home():
    return 'This is a main page'

@app.route('/admin')
def admin():
    return redirect(url_for('home'))

if __name__ == '__main__':
    app.run(debug=True)
```

საიტის admin გვერდზე
გადასვლისას ავტომატურად
მისი გადამისამართება ხდება
მთავარ გვერდზე

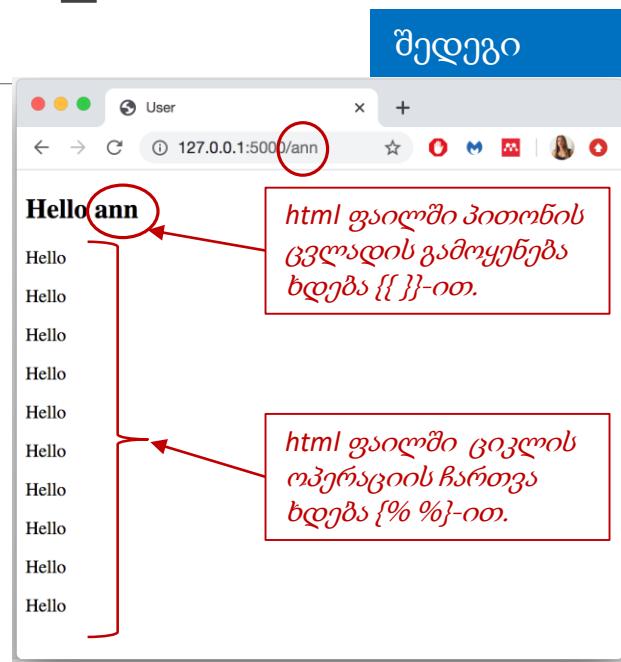


Templates

- * Templates (შაბლონები) წარმოადგენს ფაილებს, რომლებიც შეიცავს სტატიკურ მონაცემებს და placeholder-ებს დინამიური მონაცემებისთვის. შაბლონების დარენდერება ხორციელდება jinja ბიბლიოთეკის გამოყენებით, რომ მივიღოთ საბოლოო დოკუმენტი.
- * უმეტესად ვებ აპლიკაციებისთვის გამოიყენება html ფაილების შაბლონები და მათი მოთავსება ხდება templates საქაღალდეში. საიტის თითოეულ გვერდზე მიბმული უნდა იყოს კონკრეტული html გვერდი. სტატიკური Html გვერდის აგება ხდება front-end დეველოპერის მიერ, რომელიც back-end დეველოპერმა უნდა „გააცოცხლოს“ და გახადოს დინამიური.
- * მაშინ როცა html ფაილში გვსურს პითონის ცვლადების, ფუნქციების და ოპერაციების გამოყენება, დაგვჭირდება jinja-ს გამოყენება. Jinja სინტაქსი ძალიან გავს პითონს. იგი იყენებს სპეციალურ გამყოფ აღნიშვნებს, როგორიცაა **{} {}** და **{% %}**. **{} {}**-ში იწერება გამოსახულება, ხოლო **{% %}**-ში იწერება გარკვეული ოპერაცია როგორიცაა for და if.
- * მომდევნო სლაიდებზე იხილეთ შესაბამისი მაგალითები

Templates - მაგალითი 1

- * საიტის რომელიმე გვერდზე html ფაილის მიბმა ხორციელდება `render_template()` ფუნქციის გამოყენებით. პირველ პარამეტრად ეთითება ფაილის სახელი, რომელიც მოთავსებული უნდა იყოს `templates` საქაღალდეში.
- * თუ html ფაილში გვსურს გამოვიყენოთ პიტონის ფაილიდან რომელიმე ცვლადი, მათი გადაცემა ხორციელდება `render_template()` ფუნქციაში მომდევნო პარამეტრებად. მაგ. `user()` ფუნქციაში ხდება `user.html` ფაილის მიბმა და პარამეტრად გადაეცემა `name` ცვლადის მნიშვნელობა, რომლის გამოყენება html ფაილში ხდება `my_name`-ით (პარამეტრის სახელით)
- * `User.html` ფაილში გამოყენებულია jinja სინტაქსი.



საქაღალდეების სტრუქტურა

პიტონის მთავარი ფაილი

```
Project
  web_api ~/Documents/web_api
    templates
      index.html
      user.html
    venv
    ex.py
```

External Libraries

Scratches and Consoles

`ex.py`

```
from flask import Flask, redirect, url_for, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/<name>')
def user(name):
    return render_template("user.html", my_name=name)

@app.route('/about')
def about():
    return '<h1>This is about us page</h1>'

if __name__ == '__main__':
    app.run(debug=True)
```

User.html ფაილი

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>User</title>
</head>
<body>
    <h2>Hello {{my_name}}</h2>
    {% for i in range(10) %}
        <p>Hello</p>
    {% endfor %}
</body>
</html>
```

Templates - მაგალითი 2

საქაღალდეების სტრუქტურა

პირველი მოდული

User.html ფაილი

```
Project ex.py x user.html x
1: Project ~/Documents/web_api
  web_api
    templates
      index.html
      user.html
    venv
    ex.py
  External Libraries
  Scratches and Consoles

1 from flask import Flask, redirect, url_for, render_template
2
3 app = Flask(__name__)
4
5 @app.route('/')
6 def home():
7     return render_template('index.html')
8
9 @app.route('/<name>')
10 def user(name):
11     return render_template("user.html", my_name=name)
12
13 @app.route('/about')
14 def about():
15     return '<h1>This is about us page</h1>'
16
17
18 if __name__ == '__main__':
19     app.run(debug=True)
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>User</title>
</head>
<body>
    <h2>Hello {{my_name}}</h2>
    {% for i in range(10) %}
        {% if i%2 == 1%}
            <p>{{i}}</p>
        {% endif %}
    {% endfor %}
</body>
</html>
```



Templates - მაგალითი 3

საქადალდების სტრუქტურა

პითონის მთავარი ფაილი

User.html ფაილი

```
Project
  web_api ~/Documents/
    templates
      index.html
      user.html
    venv
    ex.py
  External Libraries
  Scratches and Consoles
```

ex.py

```
from flask import Flask, redirect, url_for, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/<name>')
def user(name):
    subjects = ['Python', 'Math', 'English']
    return render_template("user.html", my_name=name, sub=subjects)

@app.route('/about')
def about():
    return '<h1>This is about us page</h1>'

if __name__ == '__main__':
    app.run(debug=True)
```

user.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>User</title>
</head>
<body>
    <h2>Hello {{my_name}}</h2>
    {% for i in sub %}
        <p>საგანი: {{i}}; </p>
        {% if i=='Python' %}
            <p>გამოცდა: 1 ივლისი</p>
        {% elif i=='Math' %}
            <p>გამოცდა: 2 ივლისი</p>
        {% else %}
            <p>გამოცდა: 3 ივლისი</p>
        {% endif %}
    {% endfor %}
</body>
</html>
```

შედეგი

User

localhost:5000/ann

Hello ann

საგანი: Python;

გამოცდა: 1 ივლისი

საგანი: Math;

გამოცდა: 2 ივლისი

საგანი: English;

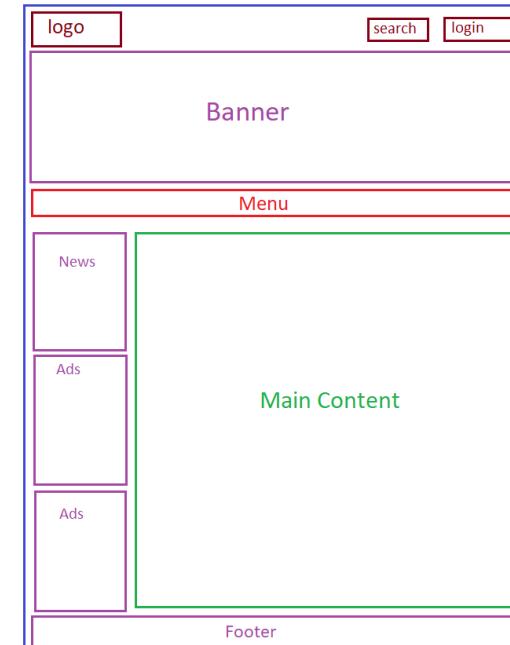
გამოცდა: 3 ივლისი

Web App – Part 2

Templates - შაბლონების ურთიერთკავშირი

- * უმეტესწილად საიტის აწყობისას საიტის სხვადასხვა გვერდს აქვს ერთი დაიგივე კომპონენტები, მაგალითად ყველა გვერდზე გვხვდება ერთი და იგივე header, მენიუ, ბანერი, footer, ხოლო ზოგიერთი ნაწილი სხვადასხვანაირია სხვადასხვა გვერდზე.
- * Jinja-ს ერთ-ერთი მნიშვნელოვანი ნაწილია შაბლონების დაკავშირება, რაც გვაძლევს იმის საშუალებას, რომ შევქმნათ base.html (ან layout.html) **ბაზისური** შაბლონი, რომელიც შეიცავს საიტის ყველა გვერდის საერთო ნაწილებს და სადაც შეგვიძლია განვსაზღვროთ ის **ბლოკები (დასახელებები მხოლოდ)** რომლებიც სხვადასხვა გვერდისთვის ცალ-ცალკე უნდა განისაზღვროს.
- * სხვა შვილობილი გვერდები (html ფაილები) აგებულია base.html-ის ბაზაზე და ბლოკები, რომლებიც განსაზღვრულია ბაზისურ ფაილში, უნდა განისაზღვროს კონკრეტულ გვერდებზე ცალ-ცალკე.
- * ბაზისურ ფაილში სასურველი ბლოკების განსაზღვრა ხდება შემდეგნაირად, სადაც უნდა მიეთითოს ბლოკის სახელი და მისი დასასრული. მათ შორის შეიძლება ეწეროს html კოდი ან იყოს ცარიელი. **{% block title %} {% endblock %}**
- * შვილობილ ფაილში პირველ რიგში უნდა მიეთითოს ბაზისური ფაილის სახელწოდება შემდეგნაირად. შემდეგ კი უნდა განისაზღვროს საჭირო html ბლოკები (რომლებიც შემოღებულია ბაზისურ ფაილში) სურვილისამებრ:

```
{% extends 'base.html' %}  
{% block title %}Home page{% endblock %}
```



Templates - შაბლონების ურთიერთკავშირი (2)

base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}  {% endblock %}</title>
</head>
<body>
    <p>Hello User</p>
    {% block content %} Default text {% endblock %}
</body>
</html>
```

index.html

```
{% extends 'base.html' %}
{% block title %}Home page {%endblock%}

{% block content %}
    {{ super() }} ←
    <h3>This is a home page</h3>
{% endblock %}
```

შედეგი

Home page

127.0.0.1:5000

Hello User

Default text

This is a home page

თუ გვსურს ბაზისურ ფაილში გაწერილი
ამავე ბლოკის (სახელად content) შიგთავსიც
წამოიღოს, ვიყენებთ super() ფუნქციას და
ვამატებთ სასურველ ტექსტს

Bootstrap – css framework

- * ვებ აპლიკაციებში ხშირად გვჭირდება სხვადასხვა ფორმის აწყობა, როგორიცაა რეგისტრაციის, ავტორიზაციის, ძებნის ფორმები. ამ დროს (და არა მხოლოდ ამ დროს), გვინდა რომ საიტის ელემენტები ვიზუალურად ლამაზად გამოიყურებოდეს. ამისათვის front-end დეველოპერი იყენებს css, რომ სტილისტიკურად და ვიზუალურად გაალამაზოს საიტის ელემენტები.
- * ვინაიდან ჩვენ მხოლოდ back-end-ის მიმართულებას განვიხილავთ, დიდ დროს არ დავკარგავთ html-ისა და css-ის განხილვაზე და პრობლემაც მარტივად რომ გადავჭრათ გამოვიყენოთ bootstrap ფრეიმვორქი, რომელიც წარმოადგენს css-ის ერთ-ერთ ყველაზე ცნობილ ფრეიმვორქს და რომელსაც აქტიურად იყენებენ front-end დეველოპერები.
- * ოფიციალური საიტი: <https://getbootstrap.com/> გადადით დოკუმენტაციის გვერდზე
- * იხილეთ სავარჯიშო ატვირთულ სავარჯიშოში.



Bootstrap (2)

- * Documentation გვერდზე გადასვლისას გაეცანით ძირითად ინსტრუქციებს. იმისათვის რომ გამოიყენოთ Bootstrap-ის კომპონენტები ბაზისური გვერდის html-ის head ტეგში უნდა მოათავსოთ css ლინკი (ლინკი იხილეთ აღნიშნულ გვერდზე). ხოლო javascript-ის ლინკები მოათავსეთ body ტეგის ბოლოში.

The screenshot shows the Bootstrap documentation website. The top navigation bar includes links for Home, Documentation (which is highlighted with a red border), Examples, Icons, Themes, Expo, and Blog. A version indicator 'v4.5' is also present. On the left, a sidebar lists 'Getting started' sections: Introduction (highlighted with a red border), Download, Contents, Browsers & devices, JavaScript, Theming, Build tools, Webpack, Accessibility, Layout, Content, Components, Utilities, Extend, Migration, and About. The main content area features a 'Quick start' section with instructions on adding Bootstrap via CDN or downloading source files. Below it is a 'CSS' section with a code snippet for linking the stylesheet. The 'JS' section follows, with a code snippet for including jQuery, Popper.js, and Bootstrap's JS. A note at the bottom explains components requiring jQuery, JS, and Popper.js.

1

Home Documentation Examples Icons Themes Expo Blog v4.5

Search...

Getting started

2

Introduction Download Contents Browsers & devices JavaScript Theming Build tools Webpack Accessibility Layout Content Components Utilities Extend Migration About

3

Quick start

Looking to quickly add Bootstrap to your project? Use BootstrapCDN, provided for free by the folks at StackPath. Using a package manager or need to download the source files? [Head to the downloads page](#).

CSS

Copy-paste the stylesheet `<link>` into your `<head>` before all other stylesheets to load our CSS.

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css" integrity="sha384-9aIt2nRpC12Uk9g59baD141NQApFr" />
```

JS

Many of our components require the use of JavaScript to function. Specifically, they require [jQuery](#), [Popper.js](#), and our own JavaScript plugins. Place the following `<script>`s near the end of your pages, right before the closing `</body>` tag, to enable them. jQuery must come first, then Popper.js, and then our JavaScript plugins.

We use [jQuery's slim build](#), but the full version is also supported.

```
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zY4C+0GpamoFVy38MVBN+EIbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvb1yZfJoft+2mJbHaEWldlvI9IOY5n3zV9zzTtmI3Ul" crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js" integrity="sha384-OgVRvuATP1z7JjHLkuOU7Xw704+h835Lr+6QL9UvYjZE3Ip" crossorigin="anonymous"></script>
```

Curious which components explicitly require jQuery, our JS, and Popper.js? Click the show components link below. If you're at all unsure about the general page structure, keep reading for an example page template.

Our `bootstrap.bundle.js` and `bootstrap.bundle.min.js` include [Popper](#), but not [jQuery](#). For more information about what's included in Bootstrap, please see our [contents](#) section.

Bootstrap (3)

- * მარცხენა მენიუდან აირჩიეთ სასურველი კომპონენტი, რისი დამატებაც გსურთ თქვენს საიტზე. მარჯვენა მხარეს გამოჩენდება სხვადასხვანაირი კომპონენტები ვიზუალური სახით და შესაბამისი html ფრაგმენტი. შეგიძლიათ აირჩიოთ სასურველი ელემენტი, დააკოპიროთ მისი html კოდი და ჩასვით სასურველ html ფაილში.

The screenshot shows the Bootstrap 4.5 Documentation website at getbootstrap.com/docs/4.5/components/forms/. The navigation bar includes links for Home, Documentation (which is active), Examples, Icons, Themes, Expo, and Blog. A purple sidebar on the left lists various components under 'Components' (1) and 'Forms' (2). The main content area displays a form example with three numbered callouts: 3 highlights the 'Email address' input field, 4 highlights the entire code snippet, and 5 highlights the 'Form controls' section below.

3 Email address

We'll never share your email with anyone else.

Password

Check me out

Submit

4

```
<form>
  <div class="form-group">
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp">
    <small id="emailHelp" class="form-text text-muted">We'll never share your email with anyone else.</small>
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword1">
  </div>
  <div class="form-group form-check">
    <input type="checkbox" class="form-check-input" id="exampleCheck1">
    <label class="form-check-label" for="exampleCheck1">Check me out</label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

5 Form controls

Textual form controls—like `<input>`s, `<select>`s, and `<textarea>`s—are styled with the `.form-control` class. Included are styles for general

HTTP methods: GET, POST

- * HTTP (Hypertext Transfer Protocol) წარმოადგენს საკომუნიკაციო არხს კლიენტსა და სერვერს შორის. იგი გულისხმობს სერვერზე request-ის გაგზავნას და სერვერიდან response-ის მიღებას.
- * ყველაზე ხშირად გამოყენებადი HTTP მეთოდებია: GET და POST
- * GET გამოიყენება მაშინ, როდესაც request-ის გაგზავნა ხდება URL-ის მეშვეობით. შესაბამისად, URL-ში ეთითება საჭირო პარამეტრები (პარამეტრის სახელი და მნიშნველობა). გამოიყენება ‘&’ სიმბოლო პარამეტრების ერთმანეთისგან გამოსაყოფად და ‘?’ ძირითადი მისამართისა და პარამეტრების ერთმანეთისგან გამოსაყოფად
- *


http://www.example.com/users.html?name1=value1&name2=value2
- * GET-ით შესაძლებელია ლიმიტირებული მონაცემების გაგზავნა სერვერზე, იგი რჩება ბრაუზერის history-ში, გადაცემული ინფორმაცია ხილვადია ყველასთვის (არ გამოიყენება დაფარული ინფორმაციის გადასაცემად)
- * POST გამოიყენება მონაცემების გასაგზავნად სერვერზე, რომლის შენახვა (ან განახლება) უნდა მოხდეს ბაზაში (ან სხვა რესურში). POST მეთოდი გამოიყენება ფორმებში შეყვანილი ინფორმაციის სერვერზე გასაგზავნად.
- * POST მეთოდის ელემენტები არ არის ხილვადი, არ ხდება მისი ქეშირება, არ რჩება ბრაუზერის history-ში, მისი ელემენტების სიგრძე არ არის ლიმიტირებული.
- * დამატებითი ინფო იხილეთ: <https://flask.palletsprojects.com/en/1.1.x/api/#flask.Request>

HTTP methods: GET, POST (2)

- * Flask-ში HTTP request-ებთან სამუშაოდ დაგვჭირდება request ობიექტი: from flask import request
- * თუ გვსურს მოვწვდეთ url-ის ატრიბუტების მნიშვნელობას, უნდა გამოვიყენოთ request-ის args რომელიც წარმოდგენილია ლექსიკონის სახით.

```
http://www.example.com/users.html?name1=value1&name2=value2
```

```
request.args['name1'] # დააბრუნებს value1-ს
```

- * POST მეთოდისას, ვინაიდან მონაცემების გადაცემა ხდება html ფორმის (<form> ტეგი) მეშვეობით, უმეტესად ფორმას აქვს submit (დადასტურების) ღილაკი. ფორმის ელემენტებს კი აქვთ სახელები, რომლის მეშვეობითაც მოგვიანებით შესაძლებელია მათზე წვდომა. დადასტურების ღილაკზე დაწვაპებისას ხდება POST request-ის გაგზავნა და შესაბამისად ფორმაში შეყვანილი ელემენტებზე წვდომა პითონში შესაძლებელია ფორმის ელემენტების სახელების მეშვეობით, რომლებიც შენახულია form ლექსიკონის სახით.

```
request.form['username'] # დააბრუნებს username ველში შეყვანილ მონაცემს
```

- * შენიშვნა: შესაძლებელია ფორმის მეშვეობით არა მარტო POST request-ის არამედ, GET request-ის გაგზავნაც (URL-ის შეცვლა). იხილეთ საკლასო სავარჯიშოები.
- * POST მეთოდის გამოყენებისას, route-ში უნდა მიეთითოს გამოსაყენებელი http მეთოდის ტიპ(ები) შემდეგნაირად:

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

Session

- * ვებ აპლიკაციებთან მუშაობის დროს ხშირად საწიროა დროებითი ინფორმაციის გამოყენება სხვადასხვა გვერდზე. მაგალითად, თუ მომხმარებელი გაივლის ავტორიზაციას, მისი მონაცემები (username) ხელმისაწვდომი უნდა იყოს საიტის სხვადასხვა გვერდზე და უნდა აჩვენებდეს რომ ავორტიზირებულია ყველგან. ესეთ შემთხვევაში იყენებენ სესიებს (sessions). სესია ინახავს ყველა საწირო ინფორმაციას მომხარებლის შესახებ.
- * Flask Session დოკუმენტაცია: <https://flask.palletsprojects.com/en/1.1.x/api/?highlight=session#flask.session>
- * სესიების მუშაობის ხანგრძლივობა ვარიაბელურია. სესია ასრულებს მუშაობას (მასში შენახული ინფორმაცია იკარგება), როცა ბრაუზერის დახურვა ხდება, საიტის დახურვისას ან ასევე წინასწარ შეიძლება განისაზღვროს სესიის მოქმედების ხანგრძლივობა (მაგ. 30 დღე). თუ მომხარებლის მიერ საიტის თავიდან გახსნისას, კვლავ დალოგინებულია, ეს ნიშნავს რომ რომ სესიაში ინფორმაცია შენახულია. ესეთ დროს გამოიყენება სესიის cookies-ში შენახვა.
- * Flask-ში სესიების გამოსაყენებლად უნდა მოვახდინოთ შესაბამისი კომპონენტის დაიმპორტება შემდეგნაირად:

```
from flask import session
```

- * Flask-ში სესიები წარმოადგენილია ლექსიკონის სახით. სესიის სასურველი მონაცემის გასაწერად ვმუშაობთ როგორც ლექსიკონზე, შემდეგნაირად: `session['user'] = user` დეტალურად იხილეთ ქვემოთ მითითებულ ამოცანაში

```
@app.route('/login', methods=['POST', 'GET'])
def login():
    if request.method == 'POST':
        user = request.form['username']
        session['user'] = user
        return redirect(url_for('user'))
    else:
        return render_template('login.html')
```

Session (2)

- * იმისათვის რომ გაეშვას პროგრამა, საჭიროა secret_key-ის მითითება. იგი წარმაოდგენს სესიის მოანცემების cookies-ში შენახვის დაცვით მექანიზმს. მისი მნისვნელობა სასურველია იყოს რანდომული ტექსტი ან ნებისმიერი სტრიქონი, რომელიც მიეთითება app ობიექტის შემოღების შემდეგ

```
app.secret_key = 'Anylongtexthere'
```

ან

```
app.config['SECRET_KEY'] = 'Anylongtexthere'
```

- * სესიის ელემენტის შექმნის შემდეგ კი ნებისმიერ ადგილას (ფუნქციაში, html ფაილში) წვდომადია სესიის ელემენტი.

```
@app.route('/user')
def user():
    if 'user' in session:
        name = session['user']
        return f"Hello {name}"
    else:
        return redirect(url_for('login'))
```

- * სესიის დასრულება იგივეა რაც logout-ის გაკეთება. თუ გვსურს რომ გავწეროთ კოდში, ესეთ დროს უნდა მოვახდინოთ სესიის ლექსიკონში შესაბამისი ელემენტის წაშლა pop ფუნქციის გამოყენებით.

```
@app.route('/logout')
def logout():
    session.pop('user', None)
    return 'you are logged out'
```

- * იმ შემთხვევაში, თუ გვსურს გამოვიყენოთ სესიების სტანდარტულზე მეტ ხანს (ბრაუზერის თავიდან გახნისას ავტომატურად იყოს დალოგინებული), უნდა გამოვიყენოთ შესაბამისი ბრძანება: **session.permanent = True**
დამატებით უნდა მიუთითოთ დღეების რაოდენობა, რამდენ ხანსაც გვსურს სესიის შენახვა **app.permanent_session_lifetime** ცვლადში (იხილეთ დოკუმენტაცია).

SQLAlchemy

- * SQLAlchemy წარმოადგენს ბიბლიოთეკას და ORM (object-relational mapper)-ს, რომელიც გამოიყენება SQL მონაცემთა ბაზებთან სამუშაოდ სხვადასხვა ტიპის აპლიკაციებისთვის.
<https://docs.sqlalchemy.org/en/13/>
- * უშუალოდ ვებ აპლიკაციებთან და flask-ში სამუშაოდ არსებობს Flask-SQLAlchemy ბიბლიოთეკა, რომელიც წარმოადგენს SQLAlchemy-ის ადაპტირებულ ვერსიას და მორგებულია Flask-ის აპლიკაციებთან სამუშაოდ.
- * ინსტალაცია: pip install Flask-SQLAlchemy
- * დოკუმენტაცია: <https://flask-sqlalchemy.palletsprojects.com/en/2.x/>
- * SQLAlchemy მუშაობს შემდეგ მონაცემთა ბაზებთან: SQLite, Postgresql, MySQL, Oracle, MS-SQL, Firebird, Sybase.
- * Flask აპლიკაციაში მოდულის დაიმპორტება:

```
from flask_sqlalchemy import SQLAlchemy
```
- * თავდაპირველად უნდა განისაზღვროს ბაზასთან სამუშაო კონფიგურაციის ელემენტები:

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///books_db.sqlite'  
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False  
db = SQLAlchemy(app)
```

SQLAlchemy (2)

- * SQLAlchemy წარმოადგენს ORM (object-relational mapper)-ს, რაც გულისხმობს, რომ ბაზასთან მუშაობა ხორციელდება ობიექტების (კლასების) მეშვეობით. სანამ ბაზასთან სამუშაო ბრძანებებს გამოვიყენებთ, იმისათვის რომ მოვახდინოთ SELECT, INSERT, UPDATE, და ა.შ, მანამდე უნდა შევქმნათ მოდელი.
- * SQLAlchemy-ში აწერილია კლასი Model, რომელიც წარმოადგენს ბაზისურ მოდელს, რომელზე დაფუძნებითაც შეგვიძლია შევქმნათ ახალი მოდელი.
- * იხილეთ მარტივი მაგალითი დოკუმენტაციაში: <https://flask-sqlalchemy.palletsprojects.com/en/2.x/quickstart/>
- * ყველა ცხრილისთვის, რომელთანაც გვსურს მუშაობა უნდა გაიწეროს ცალკე მოდელი, შემდეგნაირად:

```
class Books(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(30), nullable=False)
    author = db.Column(db.String(40), nullable=False)
    price = db.Column(db.Float, nullable=False)

    def __str__(self):
        return f'Book title:{self.title}; Author: {self.author}; Price: {self.price}'
```

- * თუ ესეთი ცხრილი ბაზაში არ არსებობს, db.create_all() ბრძანებით შეგვიძლია აპლიკაციიდან შეიქმნას შესაბამისი ცხრილი. იმ შემთხვევაში თუ უკვე არსებობს, დასახელებები უნდა ემთხვეოდეს ცხრილის და მისი ველების დასახელებებს ან შეგვიძლია დამატებით განვსაზღვროთ ბაზაში არსებული სახელები. მაგ:

```
t = db.Column('title', db.String(30), nullable=False)
```

- * ცხრილებთან სამუშაოდ უნდა შემოვიღოთ Books კლასის ობიექტი და ობიექტების მეშვეობით ვახდენთ მოქმედებებს.

SQLAlchemy (3)

- * ცხრილში SELECT-ის გაკეთება შესაძლებელია ფუნქციების გამოყენებით შემდეგნაირად: გაითვალისწინეთ, შედეგად, ბაზიდან წამოღებულ თითო ჩანაწერს აბრუნებს Books კლასის ობიექტის სახით და გსურთ შემოწმების მიზნით დაბეჭდოთ ობიექტი, მაშინ კლასში გაწერილი უნდა გქონდეს `__str__()` ან `__repr__()` ფუნქცია.

```
b1 = Books.query.first() # books ცხრილიდან პირველი ჩანაწერის წამოღება
all_books = Books.query.all() # books ცხრილიდან ყველად ჩანაწერის წამოღება
all_books = Books.query.filter_by(author='William Shakespeare').all() # books ცხრილიდან ყველად ჩანაწერის წამოღება
# ავტორის მიხედვით
for each in all_books:
    print(each)
```

შედეგი

```
Book title:Hamlet; Author: William Shakespeare; Price: 10.5
Book title:Macbeth; Author: William Shakespeare; Price: 29.0
```

- * ცხრილში INSERT-ის გაკეთება შესაძლებელია `db.session.add` ფუნქციის გამოყენებით. ცვლილებების ასახვისთვის საჭიროა `db.session.commit()` ფუნქციის გაშვება. გაითვალისწინეთ, რომ ჩანაწერის დამატებისას, `add()` ფუნქციაში პარამეტრად უნდა გადასცეთ ჩანაწერი Books კლასის ობიექტის სახით შემდეგნაირად:

```
db.session.add(Books(title='სიბრძნე სიცრუისა', author='სულხან საბა ორბელიანი', price=15))
db.session.commit()
```

- * ამგვარად, შეგიძლიათ html ფორმიდან მოხმარებლის მიერ შემოტანილი ინფორმაცია ჩაამატოთ ბაზაში ჩანაწერის სახით (იხილეთ საკლასო py ფაილი).

Web App – Part 3

Flash

* ვებ აპლიკაციებში ხშირად საჭიროა მომხარებელს გამოუტანოს გარკვეული მოქმედების შემდეგ ინფორმაცია/შეტყობინება (feedback); Flask-ში ეს საკითხი მარტივად არის გადაჭრილი flash-ის გამოყენებით. მისი მეშვეობით შესაძლებელია გარკვეული მესიჯის შენახვა და მისი გამოტანა მხოლოდ და მხოლოდ მომდევნო request-ზე.

* დოკუმენტაცია: <https://flask.palletsprojects.com/en/1.1.x/patterns/flashing/>

* flash() ფუნქციის გამოყენებით შესაძლებელია მესიჯის დამახსოვრება მომდევნო გვერდზე დაგასვლისას. ფუნქციაში მეორე პარამეტრად შეგვიზღია მივუთითოთ კატეგორიის სახელწოდება (მაგ. 'error', 'info', 'warning', ან ნებისმიერი სხვა სახელწოდება სურვილისამებრ). **შენიშვნა:** გაითვალისწინეთ, flask მოდულიდან უნდა დააიმპორტოთ flash ფუნქცია

```
flash('მონაცემი დამატებულია')
```

```
flash('მონაცემი დამატებულია', 'info')
```

* {% with var1=value %} - **with** ოპერაცია წარმოადგენს ცვლადის განსაზღვრის (მინიჭების) ოპერაციას. აღწერილი ცვლადი მოქმედებს {% endwith %} ბრძანებამდე.

* html ფაილში უნდა მოვახდინოთ შესაბამის ადგილას სასურველი მესიჯის გამოტანა

```
{% with messages = get_flashed_messages() %}
    {% for msg in messages %}
        <p>{{ msg }}</p>
    {% endfor %}
{% endwith %}
```

* შესაძლებელია ასევე კატეგორიის სახელის წამოღებაც

```
{% with messages = get_flashed_messages(with_categories=true) %}
    {% for category, msg in messages %}
        <p class="{{category}}>{{ msg }}</p>
    {% endfor %}
{% endwith %}
```

* შესაძლებელია კონკრეტული კატეგორიის მესიჯების წამოღება. ასეთ შემთხვევაში **get_flashed_messages()** ფუნქციაში უნდა მივუთითოთ **category_filter** პარამეტრი შემდეგნაირად: **get_flashed_messages(category_filter=['info'])**

* იხილეთ შესაბამისი მაგალითები საკლასო ფაილში.

Static folders – css, js, images

* აპლიკაციის საბოლოო სახე წარმოდგენილია საქაღალდეში არსებული ქვესაქაღალდეებისა და ფაილების სახით.

* დოკუმენტაცია: <https://flask.palletsprojects.com/en/1.1.x/quickstart/#static-files>

* სტატიკური ფაილები, როგორიცაა css, js და სურათები წარმოდგენილი უნდა იყოს static საქაღალდეში.

* css ფაილის ჩასმა ხორციელდება html-ში შემდეგი ბრძანების გამოყენებით:

```
<link rel="stylesheet" href="static/styles.css">
```

* ან url_for ფუნქციის გამოყენებით:

```
<link rel="stylesheet" href="{{ url_for('static', filename='style.css')}}">
```

```
/yourapplication  
yourapplication.py  
/static  
    style.css  
/templates  
    layout.html  
    index.html  
    login.html  
    ...
```

* ანალოგიურად შეგიძლიათ მოიქცეთ სურათების ჩასმის დროს. ესეთ დროს სასურველია თუ static საქაღალდეში დაამატებთ საქაღალდეს სურათებისთვის, მაგ. img საქაღალდე. ასევე შესაძლებელია css ფაილები static საქაღალდეში ცალკე ქვე საქაღალდეში იყოს მოთავსებული. შესაბამისად filename ატრიბუტში უნდა მიეთითოს static საქაღალდის ქვესაქაღალდის დასახელება და ფაილის სახელი ერთად:

```

```

Blueprint

- * Flask იყენებს Blueprint-ის კონცეფციას შედარებით დიდი და კომპლექსური აპლიკაციებისთვის. აპლიკაციას უწოდებენ მოდულურ აპლიკაციას, როცა იგი შედგება სხვადასხვა მოდულებისგან. იმისათვის რომ მარტივად მოხდეს flask-ის მოდულურ აპლიკაციებთან მუშაობა, შემოღებულია Blueprint ობიექტები. იგი გავს Flask-ის ობიექტის აპლიკაციას, თუმცა არ არის აპლიკაცია. იგი არის ოპერაციების ერთობლიობა, რომელიც უნდა ”დარეგისტრირდეს” აპლიკაციაში.
- * Blueprint-ის დარეგისტრირება აპლიკაციაში ხდება ქვედომენის/პრეფიქსის განსაზღვრის გზით.
- * დოკუმენტაცია: <https://flask.palletsprojects.com/en/1.1.x/blueprints/>
- * იხილეთ პრაქტიკული ამოცანები

admin.py

```
from flask import Blueprint

admin_page = Blueprint('admin', __name__,
static_folder='static', template_folder='templates')

@admin_page.route('/')
def admin():
    return "Admin page"

@admin_page.route('/user')
def user():
    return "Admin's user page"
```

main.py

```
from admin import admin_page

app = Flask(__name__)

app.register_blueprint(admin_page,
url_prefix='/admin')
```

Deployment

-
- * ვებ აპლიკაციის შექმნის საბოლოო ეტაპი არის მისი ატვირთვა სერვერზე და ოფიციალური გაშვება. ამისათვის საჭიროა გქონდეთ ჰოსტინგი და დომენი. ჰოსტინგის პროვაიდერები თავად აძლევენ ინსტრუქციას თუ როგორ უნდა მოხდეს საიტის ფაილების ატვირთვა სერვერზე.
 - * არსებობს cloud პლატფორმები, რომლებიც სთავაზობენ სერვერულ მხარდაჭერას. მაგ. <https://www.pythonanywhere.com/>
 - * დოკუმენტაცია: <https://flask.palletsprojects.com/en/1.1.x/tutorial/deploy/>

#Barebones App

```
from flask import Flask
app = Flask(__name__)
@app.route('/hello')
def hello():
    return 'Hello, World!'
if __name__ == '__main__':
    app.run(debug=True)
```

#Routing

```
@app.route('/hello/<string:name>') # example.com/hello/Anthony
def hello(name):
    return 'Hello ' + name + '!' # returns hello Anthony!
```

#Allowed Request Methods

```
@app.route('/test') #default, only allows GET requests
@app.route('/test', methods=['GET', 'POST']) #allows only GET and POST.
@app.route('/test', methods=['PUT']) #allows only PUT
```

#Configuration

```
#direct access to config
app.config['CONFIG_NAME'] = 'config value'
```

```
#import from an exported environment variable with a path to a config file
app.config.from_envvar('ENV_VAR_NAME')
```

#Templates

```
from flask import render_template

@app.route('/')
def index():
    return render_template('template_file.html', var1=value1, ...)
```

#JSON Responses

```
import json
@app.route('/returnstuff')
def returnstuff():
    num_list = [1,2,3,4,5]
    num_dict = {'numbers' : num_list, 'name' : 'Numbers'}

    #returns {'output' : {'numbers' : [1,2,3,4,5], 'name' : 'Numbers'}}
    return jsonify({'output' : num_dict})
```

#Access Request Data

```
request.args['name'] #query string arguments
request.form['name'] #form data
request.method #request type
request.cookies.get('cookie_name') #cookies
request.files['name'] #files
```

#Redirect

```
from flask import url_for, redirect
@app.route('/home')
def home():
    return render_template('home.html')

@app.route('/redirect')
def redirect_example():
    return redirect(url_for('index')) #sends user to /home
```

#Abort

```
from flask import abort()
@app.route('/')
def index():
    abort(404) #returns 404 error
    render_template('index.html') #this never gets executed
```

#Set Cookie

```
from flask import make_response
@app.route('/')
def index():
    resp = make_response(render_template('index.html'))
    resp.set_cookie('cookie_name', 'cookie_value')
    return resp
```

#Session Handling

```
import session
app.config['SECRET_KEY'] = 'any random string' #must be set to use sessions

#set session
@app.route('/login_success')
def login_success():
    session['key_name'] = 'key_value' #stores a secure cookie in browser
    return redirect(url_for('index'))

#read session
@app.route('/')
def index():

    if 'key_name' in session: #session exists and has key
        session_var = session['key_value']
    else: #session does not exist
```

#Useful Plugins

```
Flask-PyMongo - http://readthedocs.org/docs/flask-pymongo/
Flask-SQLAlchemy - http://pypi.python.org/pypi/Flask-SQLAlchemy
Flask-WTF - http://pythonhosted.org/Flask-WTF/
Flask-Mail - http://pythonhosted.org/Flask-Mail/
Flask-RESTFUL - https://flask-restful.readthedocs.org/
Flask-Upserts - https://flask-uploads.readthedocs.org/en/latest/
Flask-User - http://pythonhosted.org/Flask-User/
Flask-Login - http://pythonhosted.org/Flask-Login/
```

#Useful Links

```
Flask Website - http://flask.pocoo.org
Pretty Printed Website - http://prettyprinted.com
Pretty Pretty YouTube Channel - https://www.youtube.com/c/prettyprintedtutorials
```



Flask: The Cheat Sheet

Installation

```
$ pip install Flask
```

Hello World

```
# myapp.py
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World'

if __name__ == '__main__':
    app.run(debug=True)
```

URL Routing

```
@app.route('/foo/<name>/<int:age>')
def view(name, age):
    return '%s is %d years old' % (
        name, age
    )
```

Template Rendering

```
from flask import render_template

return render_template(
    'foo.html',
    var1=value1, var2=value2, ...
)
```

Flask for Django users

Action	Django	Flask
Retrieve query params ("GET data")	request.GET request.GET['name']	request.args request.args['name']
Retrieve form params ("POST data")	request.POST request.POST['first_name']	request.form request.form['name']
Checking request type	request.method	request.method
Retrieve cookie value	request.COOKIES	request.cookies
Set a cookie	response.set_cookie(key, value, ...)	response.set_cookie(key, value, ...)
Sessions	request.session['foo']	session['foo']
Access settings	from django.conf import settings settings.my_name	app.config app.config['my_name']

Useful functionality

From flask import..	This is a...	usage
Flask	Flask application object	app = Flask(__name__)
request	Thread-local request object	request.args['test'] request.form['name']
session	Thread-local session object	session['name'] = 'value'
url_for	Builds URLs based on route names	url_for('blog', id=45, slug='hi')
redirect	Generates a redirect response	return redirect('/hello')

Basic App Layout Package App Layout

my_app.py	my_app/
static/	__init__.py
logo.png	application.py
base.css	models.py
templates/	static/
blog_post.html	logo.png
	base.css
	templates/
	blog_post.html
	index.html

More Info

- <http://flask.pocoo.org> (main site & docs)
- <http://flask.pocoo.org/docs/quickstart>
- <http://github.com/mitsuhiko/flask>

დავალება:

ააგეთ საიტი, სადაც დაამატებთ 3 გვერდს:

1. Home page - განათავსეთ სასურველი ტექსტი
2. About us გვერდი - განათავსეთ სასურველი ტექსტი
3. staff-ის გვერდი - სადაც ჩამოთვლილი იქნება თანამშრომელთა სახელი გვარები; პითონის ცვლადში შეინახეთ თანამშრომლების სახელი და გვარები მაგ. list-ის სახით. html ფაილში მოახდინეთ list-ის მონაცემების გამოტანა.
4. ყველა გვერდზე დაამატეთ Home-ზე, About us-ზე და staff გვერდებზე გადასასვლელი ლინკები
5. დაამატეთ გვერდი, სადაც ლინკში შეიძლება გადაეცეს ცვალებადი მონაცემი (username) და რიცხვი (ასაკი). შესაბამისად, გვერდზე გამოიტანოს მისალემების ტექსტი user-ის სახელით და დაითვალოს რამდენ წელში გახდება პენსიონერი (დავუშვათ, რომ პენშიანი გადიან 65 წლიდან). დაამატეთ ლინკი მთავარ გვერდზე გადასასვლელად
6. სურვილისამებრ შეგიძლიათ დაამატოთ გვერდებზე სურათები და სტილისტიკური ელემენტები (css styles)

პროგრამირება

Python

ლექცია 9-10: ვებ გვერდების Scraping/Parsing; Beautiful Soup

ლიკა სვანაძე

lika.svanadze@btu.edu.ge

რა არის Web Scraping/Parsing?

- * Web Scraping/Parsing გამოიყენება ვებ გვერდებიდან დიდი ინფორმაციის წამოსაღებად და წასაკითხად ავტომატურ რეჟიმში. Scraping ნიშნავს ინფორმაციის წამოღებას, ხოლო parsing ნიშნავს წამოღებული ინფორმაციის დაყოფას ნაწილებად და საჭირო კონტენტზე წვდომას.
- * Web Scraping გამოიყენება სხვადასხვა მიზნებისთვის. მაგ. მარკეტინგისთვის, როცა კომპანიას აინტერესებს პროდუქტების შესახებ ფასების შედარება სხვადასხვა საიტზე, სოციალურ ვებ-გვერდებიდან ინფორმაციის ამოსაკრებად (მაგ. ტვიტების ანალიზი). თუმცა ამ დროს გასათვალისწინებელია ლეგალური და ეთიკური ასპექტები.
- * ვებ გვერდზე განთავსებული ინფორმაცია არ არის სტრუქტურირებული. იმ შემთხვევაში, თუ გვინდა ინფორმაცია წამოვიღოთ სტრუქტურირებული სახით, მაშინ ვიყენებთ ვებ სერვისებს, API-ს.



Web Scraping

-
- * Python-ში შეგიძლიათ გამოიყენოთ შემდეგი framework-ები საიტიდან მონაცემების წამოსაღებად:
 1. Beautiful Soup –ის მეშვეობით შესაძლებელია html ან xml კოდის პარსინგი, დამუშავება და სასურველი ინფორმაციის წამოღება
 2. Selenium - გარდა html-ისა აქვს Javascript-ის scraping-ის მხარდაჭერა.
 3. Scrapy - გამოიყენება ძალიან დიდი მონაცემების წამოსაღებად ასინქრონულად პარალელურ რეჟიმში (აქვს multithreading-ის - მრავალნაკადიანი პროგრამირების მხარდაჭერა), ავტომატურად ახდენს საიტის ერთი ლინკიდან ამავე საიტის სხვა ლინკებზე გადასვლას (Crawling) – (კომპლექსური Crawling-ის მაგალითია Google Search Engine)

Beautiful Soup

- * დოკუმენტაცია: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
- * ინსტალაცია: pip install beautifulsoup4 ან pip install bs4
- * ეტაპები:
 1. შეარჩიეთ საჭირო URL, საიდანაც გსურთ ინფორმაციის წამოღება
 2. დაათვალიერეთ ლინკის source code (inspect element-ით); მოძებნეთ ის ინფორმაცია რომლის წამოღებაც გსურთ საიტიდან
 3. დაწერეთ კოდი და გაუშვით, რომელიც წამოიღებს ინფორმაციას საიტიდან და დაამუშავებს მას
 4. შეინახეთ წამოღებული ინფორმაცია საჭირო ფორმატში



1. ვებ გვერდის URL

- * თავდაპირველად შეარჩიეთ ვებ საიტი საიდანაც გსურთ ინფორმაციის წამოღება. გაითვალისწინეთ, რომ შესაძლებელია საიტს ქონდეს robots.txt ფაილი, სადაც მითითებული იმ ქვე გვერდების ლინკები, რომლებზე წვდომაც ნებადართულია. robots.txt ფაილი უმეტესად გამოიყენება search engine-ებისთვის.
- * ვებ გვერდიდან ინფორმაციის წამოსაღებად საჭიროა ვებ საიტის კონკრეტული გვერდის URL მიუთითოთ კოდში და წამოიღოთ ამ გვერდის html კოდი. ამისათვის უნდა გამოიყენოთ requests მოდული; pip list ბრძანების გამოყენებით შეგიძლიათ ნახოთ არის თუ არა requests მოდული დაყენებული; თუ არ გაქვთ, დააყენეთ შემდეგი ბრძანების გამოყენებით pip install requests
- * Requests მოდულის დოკუმენტაცია: <https://requests.readthedocs.io/en/master/>
- * requests მოდულში არის get() ფუნქცია, რომელსაც პარამეტრად გადაეცემა url და გვიბრუნებს პასუხად http სტატუსის კოდს. მაგ. <Response [200]> პასუხი ნიშნავს, რომ წარმატებით დაუკავშირდა სერვერს. შესაძლებელია სერვერიდან დაბრუნდეს სხვა პასუხი: მაგ. 301, 404, 502
- * Response-ის კოდზე წვდომა შესაძლებელია status_code-ის გამოყენებით (იხილეთ ქვემოთ მოცემულ კოდში)
- * დაბრუნებულ ობიექტს შეგვიძლია მივმართოთ text ატრიბუტით, რომელიც გვიბრუნებს საიტის html კოდს.

```
import requests

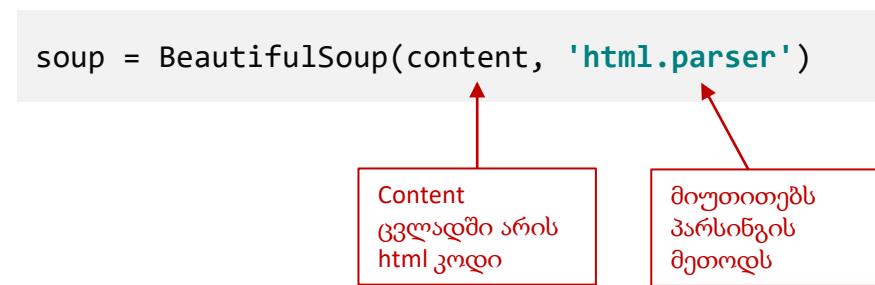
url = 'https://www.amazon.com/b?node=16225007011'
r = requests.get(url)
print(r) # დაბუჭის 
print(r.status_code) # დაბუჭის 200
content = r.text
```

2. ლინკის source code-ის გარჩევა

- * სანამ გადახვალოთ html კოდის დამუშავებაზე, მანამდე უნდა გაარჩიოთ საიტის source კოდი inspect element-ის გამოყენებით (F12 ღილაზე დაჭრით გამოდის).
 - * უნდა დაათვალიეროთ თუ როგორ არის აწყობილი საიტი და რა ტეგებია გამოყენებული. მოძებნეთ თქვენთვის საჭირო არეალი საიტის html კოდში და საჭირო ბლოკები რომლებიც უნდა წაიკითხოთ კოდში.

3. კოდის დაწერა

- * საიტიდან წამოღებული კონტენტის დასამუშავებლად გამოიყენება Beautiful Soup
მოდული - იგივეა რაც bs4; მასში აღწერილია BeautifulSoup კლასი, რომლის
დაიმპორტებაა საჭირო, რომ გამოვიყენოთ შესაბამისი ბრძანებები. დაიმპორტება
შესაძლებელია შემდეგი ბრძანებით `from bs4 import BeautifulSoup`
- * BeautifulSoup კლასის გამოყენებით უნდა შეიქმნას ობიექტი, რომლის საშუალებითაც
დავამუშავებთ html კოდს.



- * მომდევნო სლაიდებზე არსებული ბრძანებები გამოყენებულია html ფაილზე,
რომელიც ნაჩვენებია შემდეგ სლაიდზე

HTML გაგალითი

```
<!DOCTYPE html>
<html>
  <head>
    <title>Header</title>
    <meta charset="utf-8">
  </head>

  <body>
    <h2>Operating systems</h2>

    <ul class="mylist" id="list1" style="width:150px">
      <li>Solaris</li>
      <li>FreeBSD</li>
      <li>Debian</li>
      <li>NetBSD</li>
      <li>Windows</li>
    </ul>

    <h2>Programming languages</h2>
    <ul class="mylist" id="list2" style="width:150px">
      <li>Python</li>
      <li>C/C++</li>
      <li>Java</li>
      <li>PHP</li>
    </ul>

    <a href="https://www.google.com/" class="my_source"> More information</a>
    <p>
      FreeBSD is an advanced computer operating system used to
      power modern servers, desktops, and embedded platforms.
    </p>
    <p>
      Debian is a Unix-like computer operating system that is
      composed entirely of free software.
    </p>
  </body>
</html>
```

Operating systems

- Solaris
- FreeBSD
- Debian
- NetBSD
- Windows

Programming languages

- Python
- C/C++
- Java
- PHP

[More information](#)

FreeBSD is an advanced computer operating system used to power modern servers, desktops, and embedded platforms.

Debian is a Unix-like computer operating system that is composed entirely of free software.

Html tags, text, tag name, attrs

- * soup ცვლადში მოთავსებულია html ტექსტი და წარმოადგენს BeautifulSoup კლასის ობიექტს. შესაბამისად მასზე შეგვიძლია გამოვიყენოთ სხვადასხვა ატრიბუტები, ფუნქციები და დეკორატორები
- * თუ მივმართავთ soup-ს ატრიბუტებს, სადაც ატრიბუტი არის html-ის რომელიმე ტეგი, დაგვიბრუნებს ამ ატირბუტს სრულად html ტექსტიდან (პირველივე შემხვედრს)

```
print(soup.title) # დაბეჭდავს <title>Header</title>
print(soup.h2) # დაბეჭდავს <h2>Operating systems</h2>
print(soup.a) # დაბეჭდავს <a href="https://www.google.com/">More information</a>
```

- * თუ კონკრეტული ტეგის დასახლების შემდეგ გამოვიყენებთ text ატრიბუტს, მივიღებ ამ ტეგის შიგნით არსებულ ტექსტს (გამხსნელ და დამხურავ ტეგებს შორის არსებულ ტექსტს), ხოლო name ატრიბუტი აბრუნებს ტეგის დასახელებას

```
print(soup.title.text) # დაბეჭდავს Header
print(soup.h2.text) # დაბეჭდავს Operating systems
print(soup.li.text) # დაბეჭდავს More information
print(soup.h2.name) # დაბეჭდავს h2
```

- * attrs აბრუნებს ტეგის დასახელების შიგნით მითითებულ დამატებით ატრიბუტების ჩამონათვალს ლექსიკონის სახით. თითოეული ტეგის კონკრეტულ ატრიბუტზე წვდომაც შესაძლებელია

```
print(soup.a.attrs) # დაბეჭდავს {'href': 'https://www.google.com/', 'class': ['my_source']}
print(soup.a['href']) # დაბეჭდავს https://www.google.com/
print(soup.a.attrs['href']) # დაბეჭდავს https://www.google.com/
print(soup.a.get('href')) # დაბეჭდავს https://www.google.com/
```

find() ფუნქცია

- * პარსინგის დროს საჭიროა soup ტექსტში (რომელშიც მოთავსებულია html კოდი) მოვძებნოთ ჩვენთვის საჭირო ნაწილი (ტეგი) და მასში არსებული ინფორმაცია. ამისათვის გამოიყენება ძებნის ფუნქციები: find(), find_all() და სხვა.
- * find() ფუნქცია აბრუნებს მხოლოდ ერთ მნიშვნელობას, პირველივე შემთვედრს.

```
print(soup.find('ul'))
```

პარამეტრად გადაეცემა
ტეგის დასახელება;
აბრუნებს პირველ
შემთვედრს.

```
print(soup.find('ul', {'class':'mylist'}))
```

შედეგი

```
<ul class="mylist" id="list1" style="width:150px">
<li>Solaris</li>
<li>FreeBSD</li>
<li>Debian</li>
<li>NetBSD</li>
<li>Windows</li>
</ul>
```

მეორე პარამეტრად შესაძლებელია გადაეცეს ლექსიკონი, სადაც მითითებულია
კონკრეტული ატირბუტი და მისი მნიშვნელობა; შედეგად მოძებნის და დააბრუნებს
soup-ში ტეგს, რომელსაც აქვს class ატრიბუტი mylist

```
print(soup.find('ul', {'id':'list2'}))
# იგივე რაც ევემოთ მითითებული ბრძანება
print(soup.find('ul', id='list2'))
```

შედეგი

```
<ul class="mylist" id="list2" style="width:150px">
<li>Python</li>
<li>C/C++</li>
<li>Java</li>
<li>PHP</li>
</ul>
```



find_all() ფუნქცია

- * find_all() ფუნქცია აბრუნებს პარამეტრად მითითებული ტეგების შიგთავს; თუ ეს ტეგი რამდენიმეჯერ არის ნახსენები, აბრუნებს ყველას სიის სახით

შედეგი

```
print(soup.find_all('h2'))
```

```
[<h2>Operating systems</h2>, <h2>Programming languages</h2>]
```

შედეგი

```
print(soup.find_all('ul', {'class':'mylist'}))
```

```
[  
<ul class="mylist" id="list1" style="width:150px">  
  <li>Solaris</li>  
  <li>FreeBSD</li>  
  <li>Debian</li>  
  <li>NetBSD</li>  
  <li>Windows</li>  
</ul>,  
 <ul class="mylist" id="list2" style="width:150px">  
  <li>Python</li>  
  <li>C/C++</li>  
  <li>Java</li>  
  <li>PHP</li>  
</ul>  
 ]
```

4. პარსირებული ინფორმაციის შენახვა

- * ხმირად საჭიროა პარსირებული ინფორმაციის შენახვა ფაილში ან ბაზაში. უმეტესად ესეთ ინფორმაციას ინახავენ csv ფაილის ფორმატში. ამისათვის შეგვიზღვია გამოვიყენოთ სტანდარტული ფაილში ჩაწერის ოპერაციები (write) ან არსებობს ბიბლითეკები csv ფაილთან სამუშაოდ (მაგ. ბიბლიოთეკა csv - განხილულია მომდევნო სლაიდზე).

```
f = open('books.csv', 'w', encoding='utf-8_sig')
f.write('დასახელება' + ',' + 'ავტორი' + ',' + 'ფასი' + ',' + 'ლინკი' + '\n')
.....
f.close()
```

- * იხილეთ დეტალური კოდი ლექცია 12-ის Classwork.py ფაილში

CSV მოდული

- * Python-ში csv მოდულის გამოყენებით შესაძლებელია csv ფაილიდან მონაცემთა იმპორტი/წაკითხვა (Read) და/ან მონაცემების ექსპორტი/ჩაწერა (Write) csv ფაილში. csv (Comma Separated Values) ფაილი გამოიყენება დიდი რაოდენობის მონაცემების შესანახად ტექსტურ ფორმატში. CSV ფაილის გავს excel-ის ფაილს, სადაც მონაცემები წარმოდგენილია ცხრილის სახით (სტრიქონების და სვეტების მეშვეობით). csv ფაილში ცხრილის მონაცემები წარმოდგენილია ტექსტურად, რომლის თითოეული ხაზზე არის ცხრილის თითო სტრიქონის მონაცემი, ხოლო მძიმე გამოიყენება უჯრების (სვეტების) ერთმანეთისგან გამოსაყოფად.
- * შენიშვნა: ზოგიერთ შემთხვევაში შეიძლება გამყოფი იყოს წერტილმძიმე, ორწერტილი ან სხვა სასვენი ნიშანი. უმეტესად ეს დამოკიდებულია ოპერაციული სისტემის პარამეტრებზე (Regional and language settings).
- * დამატებითი ინფორმაცია: <https://docs.python.org/3/library/csv.html>

	A	B	C	D
1	Name	Surname	Faculty	Birthday
2	Ana	Giorgadze	Computer Science	04/05/1995
3	Giorgi	Nikuradze	Business	03/02/1996
4	Davit	Tabatadze	Medicine	31/01/1998
5	Giorgi	Chavchavadze	Computer Science	12/08/1999
6	Tamar	Abashidze	Math	15/03/1993
7	Nika	Lomaia	Medicine	12/08/1996
8				

csv ფაილის მონაცემები Excel-ში გახსნისას

Name, Surname, Faculty, Birthday
Ana, Giorgadze, Computer Science, 04/05/1995
Giorgi, Nikuradze, Business, 03/02/1996
Davit, Tabatadze, Medicine, 31/01/1998
Giorgi, Chavchavadze, Computer Science, 12/08/1999
Tamar, Abashidze, Math, 15/03/1993
Nika, Lomaia, Medicine, 12/08/1996
Giga, Kalandadze, Physics, 12/12/1999

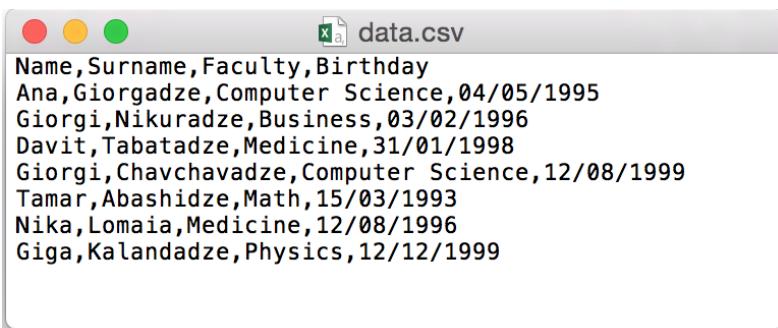
csv ფაილის მონაცემები Notepad-ში გახსნისას

CSV მოდული

ფუნქცია	აღწერა
<code>csv.reader()</code>	აბრუნებს reader-ის ობიექტს, რომლის წაკითხვა შესაძლებელია for ციკლით - თითოეულ იტერაციაზე (ბიჯზე) შესაძლებელია თითო სტრიქონის წაკითხვა. თითოეული ტრიქონი წარმოდგენილია სიის (list) სახით, რომლის ელემენტებია სტრიქონები. გაითვალისწინეთ reader() არის მოდულის ფუნქცია, და არა რომელიმე კლასის მეთოდი.
<code>csv.writer()</code>	აბრუნებს writer-ის ობიექტს, რომელზეც writerow() მეთოდის გამოყენებით შესაძლებელია csv ფაილში ცალკეული სტრიქონების (row)-ის ჩაწერა
<code>Obj.writerow()</code>	csv ფაილში ჩაიწერება პარამეტრად გადაცემული სტრიქონი. პარამეტრად უნდა გადაეცეს თითო სტრიქონი სიის სახით.

```
import csv
f = open("data.csv", "a") ←
write_obj = csv.writer(f)
write_obj.writerow(["Giga", 'Kalandadze', 'Physics', "12/12/1999"])
```

'a' რეჟიმი მიუთითებს არსებული ფაილის ბოლოში
ჩაწერას; ხოლო 'w' რეჟიმის დროს გადააწერეს
ფაილში უკვე არსებულ მონაცემებს



შენიშვნა: Windows-ზე შესაძლოა ბევრი სტრიქონის ჩასმისას,
თითოეულ სტრიქონს შორის ჩასვას ზედმეტი enter, ვინაიდან default-
ად ხდება \r\n-ის ჩასმა. ამის თავიდან ასარიდებლად, open()
ფუნქციაში მიუთითეთ მესამე პარამეტრი: open('data.csv', 'a',
newline='\n')

სურათის scraping/parsing

- * ხშირად პარსინგის დროს საჭიროა საიტიდან სურათის წამოღება შესაბამის ფორმატში (მაგ. jpeg, png, gif). ამ შემთხვევაში უნდა ვიცოდეთ სურათის ლინკი და ანალოგიურად ვახდენთ მის წამოღებას requests.get() ფუნქციის მეშვეობით.
- * იმისათვის რომ მივიღოთ სურათის შეესაბამისი კონტენტი .text ატრიბუტის ნაცვლად ვიყენებთ content ატრიბუტს, რომელიც აბრუნებს ლინკის შიგთავსს ბინარული კოდის სახით. text აბრუნებს ტექსტური მონაცემის სახით.
- * ფაილის სახით შენახვისთვის, უნდა გახსნათ ფაილი ჩაწერის რეჟიმში, მაგრამ ვინაიდან არის ბინარული ფაილი, ფაილის გახსნის რეჟიმად უნდა მივუთითოთ 'wb'

```
import requests
url = 'https://cdn.gweb.ge/buffer/1001285/pictures/slider/7555b47f7d4683e3dd002dc0b18c2ec4.png'

res = requests.get(url)

img_file = open('img1.png', 'wb')

  
img_file.write(res.content)

img_file.close()
```



არის თუ არა scraping ლეგალური?

- * Web Scraping არის ლეგალური, თუ:
 - * გათვალისწინებულია robots.txt ფაილის პირობები და საიტის web scraping policy. თუ მსგავსი პირობები არ არის, სასურველია საიტის მფლობელთან დაკავშირება და ნებართვის აღება.
 - * თუ წვდომას ხორციელდება საჯარო ინფორმაციაზე და წამოღებული ინფორმაცია გამოიყენება ანალიზის/კვლევის მიზნით
 - * თუ სერვერზე request-ის გაგზავნის სიხშირე დროში ოპტიმალურია. სასურველია რიქვესთებს შორის ინტერვალი იყოს მინიმუმ 12-15 წმ.
- * არ არის ლეგალური, თუ:
 - * კონფიდენციალურ ინფორმაციაზე წვდომა ხორციელდება შემდგომი გაყიდვის და მოგების ნახვის მიზნით
 - * წამოღებული ინფორმაცია გავრცელებულია როგორც საკუთარი ინფორმაცია წყაროს მითითების გარეშე (Copyright - საავტორო უფლებების დარღვევა)

US court says scraping a site without permission isn't illegal

მონაცემებზე წვდომა ეთიკურად და
ლეგალურად გამართლებულია API-ის
გამოყენებისას!!!! იხილეთ მომდევნო
სლაიდებზე.



Web Parsing-ის სირთულეები

-
- * **Variety** (განსხვავებულობა) - ყველა ვებ გვერდს ჭირდება ინდივიდუალური პარსინგის კოდის დაწერა, ვინადან ყველა ვებ გვერდს აქვს განსხვავებული html კოდი.
 - * **Durability** (მდგრადობა) - საიტის html კოდის ცვლილების შემთხვევაში, არსებულ პარსინგის კოდშიც საწირო გახდება ცვლილებების განხორციელება

სავარჯიშო:

1. აიღეთ თქენთვის სასურველი ნებისმეირი ვებ გვერდი, სადაც ჩამოთვლილია კონკრეტული მონაცემები (პროდუქტები, ვაკანსიები, გასაყიდი ობიექტები, ან სხვა). დაწერეთ ამ გვერდის პარსინგი და წამოღებული ინფორმაცია შეინახეთ csv ფაილში ან ბაზაში. ხშირად ესეთ გვერდზე არის paging (რამდენიმე გვერდზე მონაცემები) და საჭირო ხდება მომდევნო გვერდებიდანაც ავტომატურ რეჟიმში გადასვლა. შეეცადეთ მოიფიქროთ როგორ არის შესაძლებელი ამის განხორციელება პროგრამაში.

პროგრამირება

Python

ლექცია 6-8: Web API

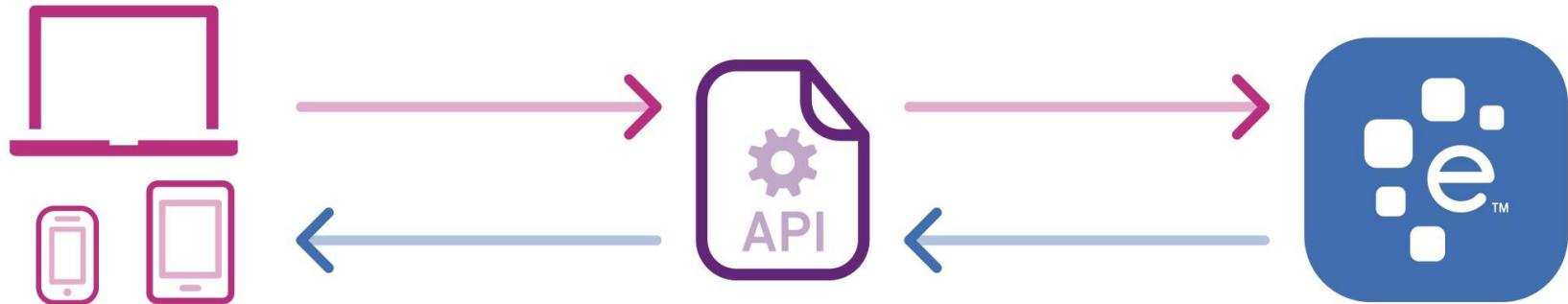
HTTP პროტოკოლი. HTTP-ის გამოყენება Python-ში

ლიკა სვანაძე

lika.svanadze@btu.edu.ge

API

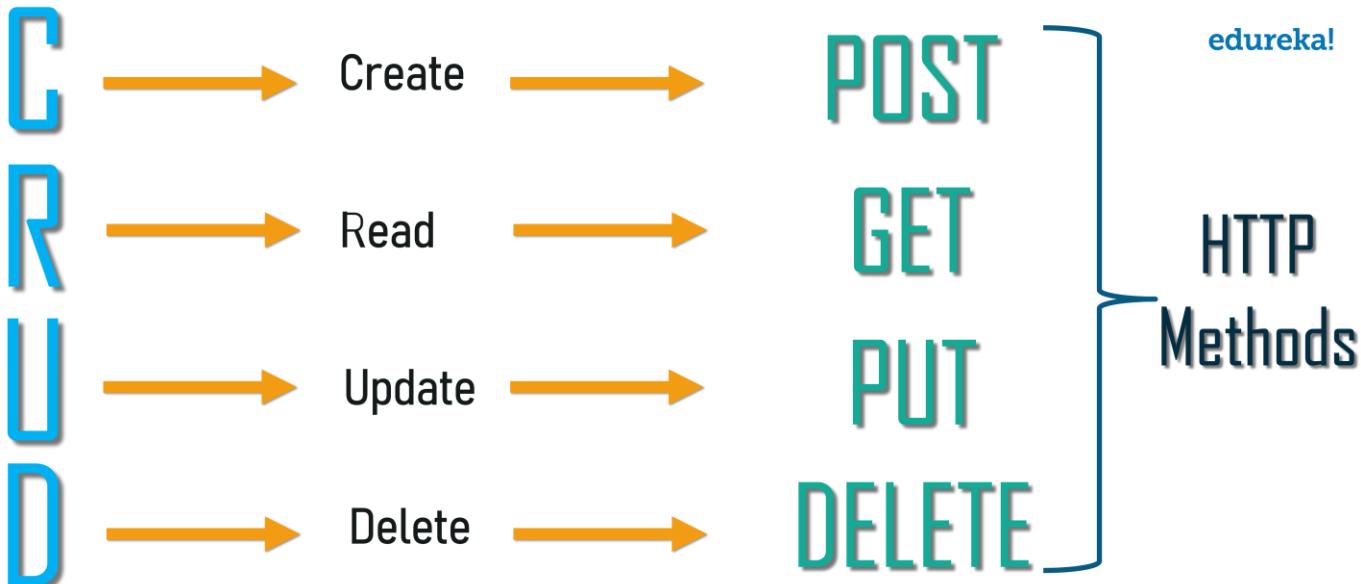
- * API (Application Programming Interfaces) წარმოადგენს მედიატორს კლიენტსა და სერვერს შორის, მისი მეშვეობით შესაძლებელია ინფორმაციის გაცვლა ორ სხვადასხვა დამოუკიდებელ software-ს შორის.
- * API-ის მაგალითებია:
 - ამინდის პროგნოზის საიტიდან მონაცემების გაგზავნა სხვა საიტზე და განთავსება
 - საერთაშორისო ფრენების, სასტუმროების შესახებ ინფორმაციის განთავსება სამოგზაურო საიტებზე
 - Youtube-ის ვიდეოების ჩასმა (Embedding) სხვა საიტზე



Requests მოდული

- * API-ს მეშვეობით ვიღებთ ჩვენთვის საჭირო მონაცემებს (data) სერვერიდან, რისთვისაც საჭიროა რომ შევქმნათ request
- * სერვერთან სამუშაოდ python-ში გვჭირდება შესაბამისი მოდული requests. pip list ბრძანების გამოყენებით შეგიძლიათ ნახოთ არის თუ არა requests მოდული დაყენებული; თუ არ გაქვთ, დააყენეთ შემდეგი ბრძანების გამოყენებით pip install requests
- * მოდულის დოკუმენტაცია: <https://docs.python-requests.org/en/master/>
- * სერვერსა და კლინეტს შორის ინფორმაციის გაცვლა ხდება HTTP პროტოკოლის მიერ.

HTTP Requests



- **GET:** აბრუნებს საჭირო ინფორმაციას სერვერიდან, რომელიც შესაძლებელია API-ს გამოყენებით იყოს ხელმისაწვდომი.
- **POST:** ამატებს ინფორმაციას სერვერზე. მაგ. მისი სასუალებით შესაძლებელია დაემატოს მომხმარებელი თავისი პირადი ინფორმაციით სარეგისტრაციო ფორმის შევსების გზით.
- **PUT:** ცვლის არსებულ ინფორმაციას. მაგ. მისი საშუალებით შესაძლებელია პროდუქტის მახასიატებლების ცვლილება.
- **DELETE:** შლის არსებულ ინფორმაციას

HTTP methods: GET, POST

- * HTTP (Hypertext Transfer Protocol) წარმოადგენს საკომუნიკაციო არხს კლიენტსა და სერვერს შორის. იგი გულისხმობს სერვერზე request-ის გაგზავნას და სერვერიდან response-ის მიღებას.
- * ყველაზე ხშირად გამოყენებადი HTTP მეთოდებია: GET და POST
- * GET გამოიყენება მაშინ, როდესაც request-ის გაგზავნა ხდება URL-ის მეშვეობით. შესაბამისად, URL-ში ეთითება საჭირო პარამეტრები (პარამეტრის სახელი და მნიშნველობა). გამოიყენება ‘&’ სიმბოლო პარამეტრების ერთმანეთისგან გამოსაყოფად და ‘?’ ძირითადი მისამართისა და პარამეტრების ერთმანეთისგან გამოსაყოფად

Get Request

<http://www.example.com/users.html?name1=value1&name2=value2>

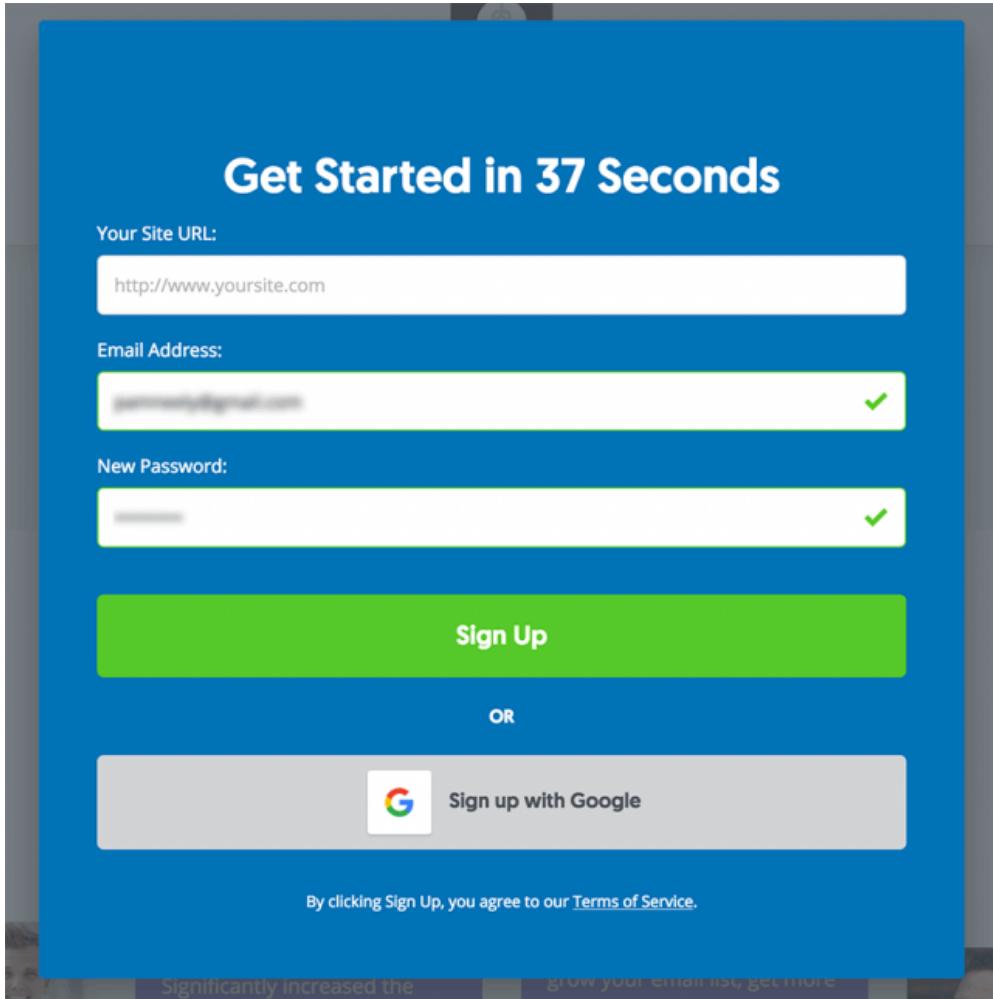
- * GET-ით შესაძლებელია ლიმიტირებული მონაცემების გაგზავნა სერვერზე, იგი რჩება ბრაუზერის history-ში, გადაცემული ინფორმაცია ხილვადია ყველასთვის (არ გამოიყენება დაფარული ინფორმაციის გადასაცემად)
- * POST გამოიყენება მონაცემების გასაგზავნად სერვერზე, რომლის შენახვა უნდა მოხდეს ბაზაში (ან სხვა რესურში). POST მეთოდი გამოიყენება ფორმებში შეყვანილი ინფორმაციის სერვერზე გასაგზავნად.
- * POST მეთოდის ელემენტები არ არის ხილვადი, არ ხდება მისი ქეშირება, არ რჩება ბრაუზერის history-ში, მისი ელემენტების სიგრძე არ არის ლიმიტირებული.

Get ფუნქცია

- * requests მოდულში არის get() ფუნქცია, რომელსაც პარამეტრად გადაეცემა url და გვიბრუნებს პასუხად http სტატუსის კოდს. მაგ. <Response [200]> პასუხი ნიშნავს, რომ წარმატებით დაუკავშირდა სერვერს. შესაძლებელია სერვერიდან დაბრუნდეს სხვა პასუხი: მაგ. 301, 404, 502
- * Response-ის კოდზე წვდომა შესაძლებელია status_code-ის გამოყენებით
- * დაბრუნებულ ობიექტს შეგვიძლია მივმართოთ text ატრიბუტით, რომელიც გვიბრუნებს საიტის html კოდს.

```
import requests  
resp = requests.get('https://btu.edu.ge/')
```

Post



* HTML ფორმების შევსება

* ფაილების ატვირთვა



* საშუალებას იძლევა დიდი

ოდენობის მონაცემი

გაიგზავნოთს ერთიანად

Response

- * get() ფუნქცია აბრუნებს Response კლასის ობიექტს, რომელიც წარმოადგენს სერვერიდან მიღებულ პასუხს. მას გააჩნია სხვადასხვა პარამეტრები:
 - * Status_code - სერვერიდან მიღებული პასუხის სტატუსი (იხილეთ მომდევნო სლაიდზე)
 - * headers - სერვერიდან მიღებული პასუხის დამატებითი ინფორმაცია. მაგ. სერვერის დასახლება, კონტენტის ტიპი, ა.შ. შედეგი არის dict ტიპის.
 - * text – Response-ის შიგთავსი წარმოდგენილი ტექსტის სახით
 - * Content - Response-ის შიგთავსი (არა ტექსტური მონაცემებისთვის, მაგ. სურათი)

```
import requests
resp = requests.get('https://btu.edu.ge/')
print(resp)
print(resp.status_code)
print(resp.headers)
print(resp.headers['Content-Type'])
print(resp.text)
```

Status Codes

HTTP STATUS CODES

2xx Success

200 Success / OK

3xx Redirection

- 301** Permanent Redirect
- 302** Temporary Redirect
- 304** Not Modified

4xx Client Error

- 401** Unauthorized Error
- 403** Forbidden
- 404** Not Found
- 405** Method Not Allowed

5xx Server Error

- 501** Not Implemented
- 502** Bad Gateway
- 503** Service Unavailable
- 504** Gateway Timeout

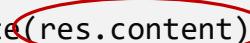
სურათის URL get ფუნქციაში

- * Get ფუნქციაში შესაძლებელია მიეთითოს სურათის URL
- * ამ შემთხვევაში უნდა ვიცოდეთ სურათის ლინკი და ანალოგიურად ვახდენთ მის წამოღებას requests.get() ფუნქციის მეშვეობით.
- * იმისათვის რომ მივიღოთ სურათის შესაბამისი კონტენტი .text ატრიბუტის ნაცვლად ვიყენებთ content ატრიბუტს, რომელიც აბრუნებს ლინკის შიგთავსს ბინარული კოდის სახით. text აბრუნებს ტექსტური მონაცემის სახით.
- * ფაილის სახით შენახვისთვის, უნდა გახსნათ ფაილი ჩაწერის რეჟიმში, მაგრამ ვინაიდან არის ბინარული ფაილი, ფაილის გახსნის რეჟიმად უნდა მივუთითოთ ‘wb’

```
import requests
url = 'https://cdn.gweb.ge/buffer/1001285/pictures/slider/7555b47f7d4683e3dd002dc0b18c2ec4.png'

res = requests.get(url)

img_file = open('img1.png', 'wb')

  
img_file.write(res.content)

img_file.close()
```



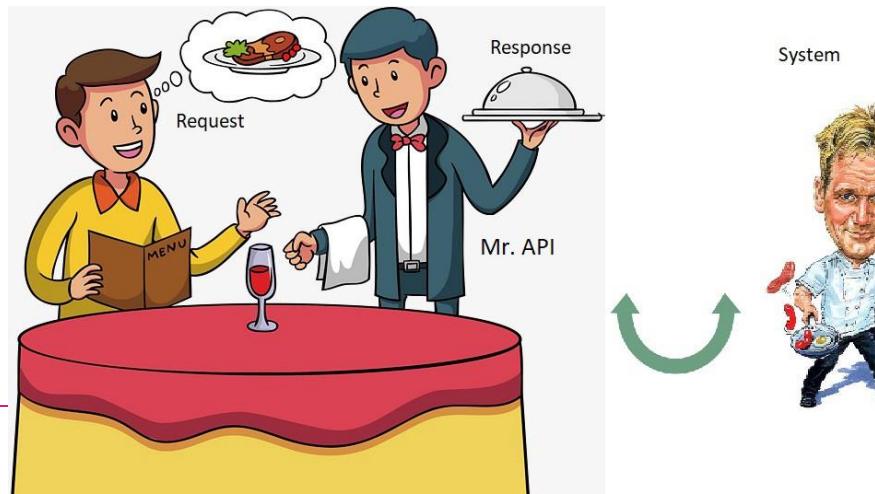
Most Popular API by Category

-
- [Weather](#)
 - [Sports](#)
 - [SMS](#)
 - [Food & Restaurant](#)
 - [Music](#)
 - [News](#)
 - [Dictionary](#)
 - [Travel](#)
 - [Real Estate](#)
 - [IP Geolocation](#)
 - [Coupon](#)
 - [Video Games](#)
 - [Alcohol](#)
 - [Cryptocurrency](#)
 - [Anime & Manga](#)
 - [Events](#)
 - [Machine Learning](#)
 - [Payments](#)
 - [Mapping](#)
 - [NFL](#)
 - [Analytics](#)
 - [NLP](#)
 - [Email](#)
 - [Facial Recognition](#)
 - [Search](#)
 - [Finance](#)
 - [Cloud](#)
 - [Blogging](#)
 - [Shipping](#)
 - [Streaming](#)
 - [Animation](#)
 - [Booking](#)
 - [PDF](#)
 - [Emotion](#)
 - [OCR](#)
 - [Company Information](#)
 - [Text Summarization](#)
 - [Chat](#)
 - [Data](#)
 - [Math](#)
 - [IP & Domain](#)
 - [Wiki](#)
 - [HTML5](#)

<https://rapidapi.com/blog/most-popular-api/>

API

- * ზოგიერთი საიტი უზრუნველყოფს API-ს (Application Programming Interfaces), რომელიც საშუალებას გვაძლევს მივწვდეთ საიტზე არსებულ ინფორმაციას წინასწარ განსაზღვრული გზით. API-ის მეშვეობით, თავიდან ვირიდებთ HTML-ის პარსინგს და მონაცემებზე წვდომა გვაქვს JSON და XML ფორმატის გამოყენებით.
- * API-ის გამოყენებისას საჭიროა შესაბამისი დოკუმენტაციის გარჩევა
- * მაგალითად, ძალიან პოპულარულია ამინდის პროგნოზის API. <https://openweathermap.org/api> საიტი უზრუნველყოფს API key-ს, რომელიც მიეთითება კოდში და დოკუმენტაციის გამოყენებით შესაძლებელია პროგრამაში მიეთითოს რომელი ქალაქის ამინდის პროგნოზის წამოღება გვსურს. შესაბამისად ავაწყობთ URL-ს, ვაგზავნით get requests, და შედეგად ვიღებთ json ფაილს.



API - მაგალითი

```
import requests
city = 'Kutaisi'
key = 'b0382a9da8d31051dd5eecdcc220673dc'
url = f'http://api.openweathermap.org/data/2.5/weather?q={city}&appid={key}&units=metric'
r = requests.get(url)
print(r)
print(r.headers)
print(r.text)
```

შედეგი:

Response

Response [200]

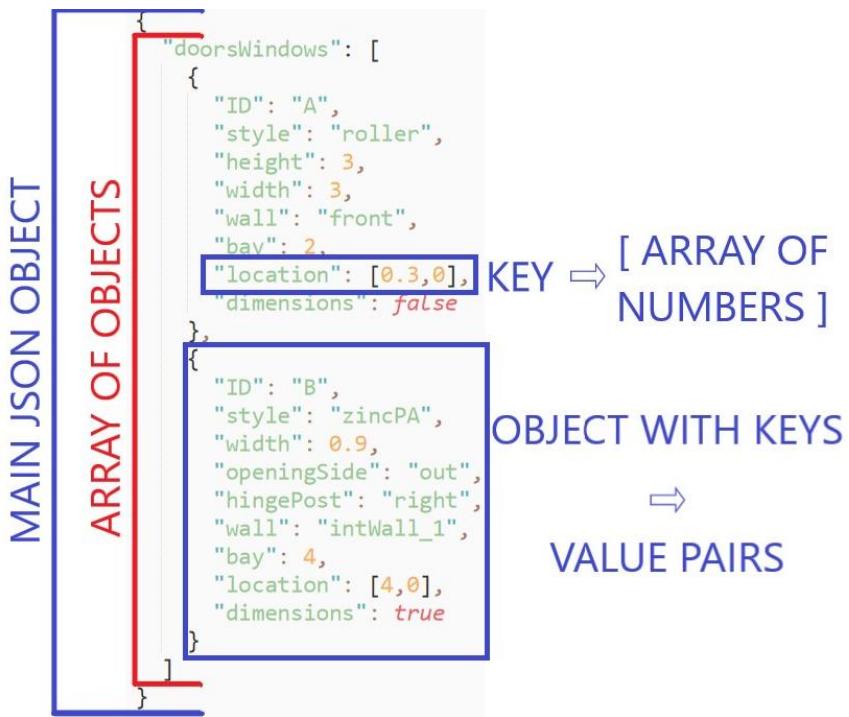
Response-ის header

Response-ის შეგთვევის json-ის სტრუქტურა

```
{"Server": "openresty", "Date": "Sun, 18 Apr 2021 09:39:43 GMT", "Content-Type": "application/json; charset=utf-8", "Content-Length": "472", "Connection": "keep-alive", "X-Cache-Key": '/data/2.5/weather?q=kutaisi&units=metric', "Access-Control-Allow-Origin": "*", "Access-Control-Allow-Credentials": "true", "Access-Control-Allow-Methods": "GET, POST"}  
{"coord": {"lon": 42.6997, "lat": 42.2496}, "weather": [{"id": 500, "main": "Rain", "description": "light rain", "icon": "10d"}], "base": "stations", "main": {"temp": 26, "feels_like": 26, "temp_min": 26, "temp_max": 26, "pressure": 1014, "humidity": 38}, "visibility": 10000, "wind": {"speed": 2.57, "deg": 130}, "rain": {"1h": 0.42}, "clouds": {"all": 75}, "dt": 1618738778, "sys": {"type": 1, "id": 8856, "country": "GE", "sunrise": 1618712613, "sunset": 1618761186}, "timezone": 14400, "id": 613607, "name": "Kutaisi", "cod": 200}
```

JSON

- * API-ს გამოყენებით შედეგად სერვერი გვიბრუნებს json ფაილს.
- * Json ფაილში არსებული მონაცემის სტრუქტურა ძალიან გავს dictionary-ის.



JSON

```
{  
  "firstName": "Jane",  
  "lastName": "Doe",  
  "hobbies": ["running", "sky diving", "singing"],  
  "age": 35,  
  "children": [  
    {  
      "firstName": "Alice",  
      "age": 6  
    },  
    {  
      "firstName": "Bob",  
      "age": 8  
    }  
  ]  
}
```

JSON

- * `r = requests.get()` ფუნქციით დაბრუნებული `json` ფაილის კონტენტის წასაკითხად, შეგვიძლია გამოვიყენოთ როგორც `r.text` რომელიც გვიბრუნებს `json` ფაილს სტრიქონის სახით, ასევე `r.json()` ფუნქცია, რომელიც გვიბრუნებს `json` ფაილს `dict` სახით.
- * პითონში `json` ფაილთან სამუშაოდ შესაძლოა გამოვიყენოთ `json` ბიბლიოთეკა, რომლის მეშვეობითაც უფრო გამარტივებულია მოქმედებები.
- * Json მოდულის დოკუმენტაცია: <https://docs.python.org/3/library/json.html>
- * Json მოდულიდან შეგვიძლია გამოვიყენოთ ფუნქცია `loads()`, რომელიც ტექსტურ მონაცემს (რომელშიც მოთავსებულია `json` ობიექტი) გარდაქმნის პითონის `dict` ობიექტად. შედეგად შეგვიძლია, `json`-ის მონაცემები წავიკითხოთ `dict`-ის მეშვეობით. ანალოგიურად, არსებობს `load()` ფუნქცია, რომელსაც პარამეტრად გადაეცემა არა `str` ტიპის მიმდევარი და ფაილში არსებულ `json` ობიექტს გარდაქმნის პითონის ობიექტად (`dict`).
- * პითონში `json` ობიექტის დაბეჭდვისას, შედეგი იბეჭდება ერთ ხაზზე და `json`-ის სტრუქტურის აღქმა გამნელებულია. იმისათვის რომ უფრო ლამაზად გამოვიტანოთ ეკრანზე, რაც გაგვიადვილებს შემდეგ მასზე მუშაობას, უმჯობესია გამოვიყენოთ `dumps()` ფუნქცია, სადაც პირველ პარამეტრად გადაეცემა `json`-ის ობიექტი, ხოლო მეორე ატრიბუტად უნდა გადავდეთ `indent` შესაბამისი მნიშვნელობით. მაგ. `res_structured = json.dumps(res, indent=4)`. შედეგად ვიღებთ `json` ფორმატით ჩაწერილ სტრიქონის ობიექტს და ვიზუალურად სტრუქტურირებულია. ანალოგიურად, არსებობს `dump()` ფუნქცია, რომელსაც მეორე პარამეტრად გადაეცემა ფაილის დასახელება, და ახდენს ფაილში `json` ობიექტად ინფორმაციის ჩაწერას. თუ გადაეცემა `indent` პარამეტერი, შედეგი იქნება ვიზუალურად სტრუქტურირებული. გამოიყენება `json` ფაილის შესაქმნელად.
- * იხილეთ ამოცანა მომდევნო სლაიდზე და `Classwork.py` ფაილში

Python	JSON
<code>dict</code>	<code>object</code>
<code>list, tuple</code>	<code>array</code>
<code>str</code>	<code>string</code>
<code>int, long, float</code>	<code>number</code>
<code>True</code>	<code>true</code>
<code>False</code>	<code>false</code>
<code>None</code>	<code>null</code>

JSON - მაგალითი

- * openweathermap-ის გამოყენებით, ნებისმიერი ქალაქის ტემპერატურის და ტენიანობის დაბეჭდვა API-ის გამოყენებით

```
import requests
import json
api_key = 'b0382a9da8d31051dd5eecdcc220673dc'
city = input("Enter the City: ") ←
url = f'http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric'
r = requests.get(url)
result_json = r.text
res = json.loads(result_json)
res_structured = json.dumps(res, indent=4)
print(res_structured) ←
m = res['main']
temp = m['temp']
humidity = m['humidity']
print('ტემპერატურა: ', temp, 'ცელსიუსი')
print('ტენიანობა: ', humidity, '%')
```

შედეგი

Enter the City: Tbilisi

```
{
  "coord": {
    "lon": 44.83,
    "lat": 41.69
  },
  "weather": [
    {
      "id": 800,
      "main": "Clear",
      "description": "clear sky",
      "icon": "01d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 32,
    "feels_like": 29.35,
    "temp_min": 32,
    "temp_max": 32,
    "pressure": 1016,
    "humidity": 27
  },
  "visibility": 10000,
  "wind": {
    "speed": 4.1,
    "deg": 170
  },
  "clouds": {
    "all": 0
  },
  "dt": 1591788823,
  "sys": {
    "type": 1,
    "id": 8862,
    "country": "GE",
    "sunrise": 1591752361,
    "sunset": 1591806873
  },
  "timezone": 14400,
  "id": 611717,
  "name": "Tbilisi",
  "cod": 200
}
```

სასურველი ქალაქის
შეყვანა

სერვერიდან
მიღებული json
ფაილის შიგთავსი

შედეგი

ტემპერატურა: 32 ცელსიუსი
ტენიანობა: 27 %

API-ის საიტები

Links	Description
www.openweathermap.org	Weather forecasts, nowcasts and history in a fast and elegant way
www.api-football.com	Football API
https://api.nasa.gov	NASA API
https://github.com/ExpDev07/coronavirus-tracker-api	კორონა ვირუსის API
www.chandan-02.github.io/anime-facts-rest-api/	Anime facts API
www.thronesapi.com	Game of Thrones Character Api
www.tvmaze.com/api	TV information about movies
https://www.spaceflightnewsapi.net/	Spaceflight Related News API
https://hp-api.onrender.com/	Harry Potter characters API



WEB API - ლინკები

- კორონა ვირუსის WEB API: <https://github.com/ExpDev07/coronavirus-tracker-api>
- პოპულარული WEB API-ების ჩამონათავალი: <https://github.com/public-apis/public-apis>
- არსებული Python Wrapper-ები Web API-სთვის:
<https://github.com/realpython/list-of-python-api-wrappers>

სავარჯიშო:

1. გამოიყენეთ <https://openweathermap.org/api> ამინდის პროგნოზის API, რომლის საშუალებითაც იპოვით ნებისმიერი ქალაქის ამინდის მონაცემებს (შეაყვანინეთ მომხმარებელს ქალაქის დასახლება); სასურველია გამოიყენოთ API-ის ის ჭვე მოდული რომელიც ლექციაზე არ განხილულა. მაგალითად: ამჟამინდელი ამინდი კონკრეტულ არის (კვადრატი ან წრე) ყველა ქალაქში (მაგ. იმერეთის რეგიონში), მომდევნო 16 დღიანი ამინდის პროგნოზი, ან საათობრივი ამინდის პროგნოზი
2. გამოიყენეთ Nasa-ს რომელიმე api (სასურველი ის, რომელიც ლექციაზე არ განხილულა). წამოიღეთ თქვენთვის სასურველი ინფორმაცია, დაამუშავეთ json მონაცემები პითონში.
3. მოძებნეთ სასურველი საიტი (წინა სლაიდზე მოცემულია მსგავსი საიტების ლინკი), რომელიც გთავაზობთ API-ს და დაწერეთ შესაბამისი პროგრამა. გამოიყენეთ json ფაილთან სამუშაო ფუნქციები.