



CS324 - SKRIPTING JEZICI

Napredni rad u Flask paketu

Lekcija 14

PRIRUČNIK ZA STUDENTE

CS324 - SKRIPTING JEZICI

Lekcija 14

NAPREDNI RAD U FLASK PAKETU

- ✓ Napredni rad u Flask paketu
- ✓ Poglavlje 1: Funkcijsko zatvorenje
- ✓ Poglavlje 2: Dekoratori u Python jeziku
- ✓ Poglavlje 3: Jinja notacija za Python jezik
- ✓ Poglavlje 4: Šabloni za Flask mikroservis
- ✓ Poglavlje 5: Nastavak rada u izradi Flask veb aplikacije
- ✓ Poglavlje 6: Pokazne vežbe
- ✓ Poglavlje 7: Individualne vežbe
- ✓ Poglavlje 8: Domaći zadatak
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

▼ Uvod

UVOD

Uvod u lekciju #14

U ovoj lekciji nastavlja se sa implementacijom Flask mikroservisa za razvoj veb aplikacija u Python programskom jeziku.

Da bi se razumeli napredni koncepti koji su korišćeni, najpre se uvodi pojam *funkcije prve klase* (en. *first class functions*), zatim *funkcijskog zatvorenja* (en. *function closures*), kao i upotreba *dekoratora* (en. *decorators*).

Da bi Python veb aplikacije bile dinamičke, treba znati koristiti Jinja notaciju za šablone koje koriste Flask aplikacije.

Nakon toga, nastavlja se rad u postojećoj Flask aplikaciji sa lekcije #13, prvenstveno kroz uvođenje šablona.

Predstavljene su samo osnovne funkcionalnosti paketa Flask, a kompletno uputstvo (*playlist*) se može pogledati na:

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 1

Funkcijsko zatvorenje

FUNKCIJA PRVE KLASE

Ako programski jezik podržava funkcije prve klase, onda se funkcija može tretirati kao promenljiva.

Funkcija prve klase (en. first class function) jeste sposobnost programskoj jezika da tretira funkciju kao objekat prve klase (en. first class citizen). Objekat prve klase predstavlja entitet koji podržava sve operacije koje su dozvoljene ostalim entitetima, i uključuju prosleđivanje kao argument, povratna vrednost funkcije, kao i dodelu vrednosti promenljivoj.

To znači da jezik podržava slanje funkcija kao argumente drugim funkcijama, pri čemu ih vraća kao vrednost drugih funkcija i dodeljuje ih promenljivama ili ih čuva u strukturama podataka.

Primer #1:

Napisati funkciju **square**, koja uzima parametar **x** i vraća kvadriranu vrednost ulaznog parametra. Zatim štampati funkciju sa nekim prirodnim brojem, kao i štampati funkciju bez promenljivih i bez zagrada (tj. bez poziva funkcije).

```
# funkcija prve klase

def square(x):
    return x * x
print( square(5) )
print( square )
```

Izlaz:

```
>>> 25
>>> <function square at 0x01027808>
```

Može se videti da prilikom drugog poziva štampanja, kada se pozvala funkcija kao promenljiva, izlaz vraća da je objekat funkcija **square** na nekoj memorijskoj lokaciji.

Primer #2

Koristeći funkciju square iz primera #1, napraviti dve promenljive, **f_1** i **f_2**. Prva promenljiva uzima vrednost funkcije za parametar 5, dok druga promenljiva uzima vrednost same funkcije.

```
# primer #2
f_1 = square(5)
f_2 = square

print(f_1)
print(f_2)
print(f_2(5))
```

Izlaz:

```
>>> 25
>>> <function square at 0x01027808>
>>> 25
```

Promenljiva **f_1** je pozvala funkciju **square** za parametar 5, pa se njoj dodelila vrednost izlaza funkcije. Međutim, promenljivoj **f_2** se dodelila sama funkcija, tako da je sada moguće pozvati funkciju **f_2** koja je identična kao funkcija **square**.

PROSLEĐIVANJE FUNKCIJE KAO PARAMETAR

Pri definisanju funkcije, kao parametar moguće je proslediti i neku drugu funkciju.

Ukoliko programski jezik podržava funkcije prve klase, onda se pri definisanju funkcije kao parametar proslediti i neka druga funkcija.

Primer #3:

Napisati funkciju **my_map**, koja ima dva parametara: parametar **func** je funkcija, dok je drugi parametar lista **list_arg**.

Unutar funkcije, za sve elemente parametra **list_arg** pozivati funkciju **func**, i smestiti u listu **result**, koju treba vratiti kao povratnu vrednost.

Isprobati funkcionalnost za funkcije **square(x)**, koja vraća x^2 , i funkciju **cube(x)**, koja vraća x^3 .

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

```
# funkcija kao parametar druge funkcije

def my_map(func, arg_lst):
    result = []
    for item in arg_lst:
        result.append(func(item))
    return result

def square(x):
    return x*x
```

```
def cube(x):
    return x*x*x

lst = [1,2,3,4,5]

print(my_map(square, lst))
print(my_map(cube, lst))
```

Izlaz:

```
>>> [1, 4, 9, 16, 25]
>>> [1, 8, 27, 64, 125]
```

POJAM FUNKCIJSKOG ZATVORENJA

Zatvorenje omogućava funkciji da pristupi uhvaćenim promenljivim kroz kopije vrednosti ili referenci koje se odnose na zatvorenje.

Prema wikipediji, pojam zatvorenje, leksičko zatvorenje ili [funkcijsko zatvorenje](#) (en. [closure](#)), je tehnika za implementaciju leksički obuhvaćenog vezivanja imena u jeziku sa funkcijama prve klase.

Praktično, zatvorenje je zapis koji čuva funkciju zajedno sa okruženjem. Okruženje je preslikavanje koje povezuje svaku slobodnu promenljivu neke funkcije (promenljive koje se koriste kao lokalne, ali su definisane u obuhvatajućem opsegu), sa vrednošću ili referencom na koju je ime vezano kada je zatvorenje kreirano. Zatvorenje omogućava funkciji da pristupi uhvaćenim promenljivim kroz kopije vrednosti ili referenci koje se odnose na zatvorenje, čak i kada je funkcija pozvana izvan njihovih opsega.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Primer #4:

Napisati funkciju **outer_func** koja nema ulazne parametre. Unutar funkcije, napraviti lokalnu promenljivu **message** koja ima vrednost "Hello, World!". Takođe, unutar funkcije napraviti novu funkciju **inner_func**, koja nema parametre. U telu unutrašnje funkcije štampati promenljivu **message**.

Kao povratnu vrednost spoljašnje funkcije vratiti a) poziv funkcije **inner_func**, b) samo funkciju **inner_func**. Diskutovati rezultate kada se pozove **outer_func()**.

```
# zatvorenje funkcije
def outer_func():
    message = 'Hello, World!'

    def inner_func():
        print(message)

    return inner_func()
```

```
# return inner_func  
outer_func()
```

▼ Poglavlje 2

Dekoratori u Python jeziku

POJAM DEKORATORA U PYTHON JEZIKU

Dekoratorima je moguće proširiti funkcionalnost funkcije, bez modifikacije osnovne strukture funkcije.

Dekorator (en. **decorator**) jeste šablon projektovanja u Python programskom jeziku koji omogućava korisniku da doda novu funkcionalnost postojećem objektu a da ne menja njegovu strukturu.

Dekoratori se obično javljaju pre definisanja funkcije za koju treba uraditi dekorator.

Dekoratori se najčešće koriste pri razvoju veb aplikacije korišćenjem Flask paketa ili Django radnog okvira.

Dekoratori se konstruišu slično kao funkcije zatvorenja, s tim što se spoljašnjoj funkciji prosleđuje kao parametar funkcija.

Primer #1:

Napraviti dekorator koji štampa "Dodatna funkcionalnost" pre izvršenja originalne funkcije.

```
# primer za dekoratore

def decorator_function(original_function):
    def wrapper_function():
        print('Dodatna funkcionalnost')
        return original_function()
    return wrapper_function

def display():
    print("Display funkcija pokrenuta")

decorated_display = decorator_function(display)
decorated_display()
```

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

KONSTRUKCIJA DEKORATORA ZA BILO KOJI TIP FUNKCIJA

Dekorator se poziva tako što se pre definicije originalne funkcije napiše ime spoljašnje funkcije sa znakom @.

Pri konstrukciji dekoratora, treba obratiti pažnju na oblik originalne funkcije.

Neke funkcije mogu se pozivati bez argumenata, dok se druge moraju pozvati sa argumentima i/ili ključnim rečima.

Konvencija jeste pisati argumente i ključne reči kao **args** i **kwargs**.

Zbog toga, prilikom definisanja unutrašnje funkcije, potrebno je navesti argumente i ključne reči koje bi primila originalna funkcija:

```
def outer_func(orig_func):  
  
    def inner_func(*args, **kwargs):  
  
        # dodatna funkcionalnost  
        # ...  
        # ...  
  
        orig_func(*args, **kwargs)  
  
        # dodatna funkcionalnost  
        # ...  
        # ...  
  
    return inner_func
```

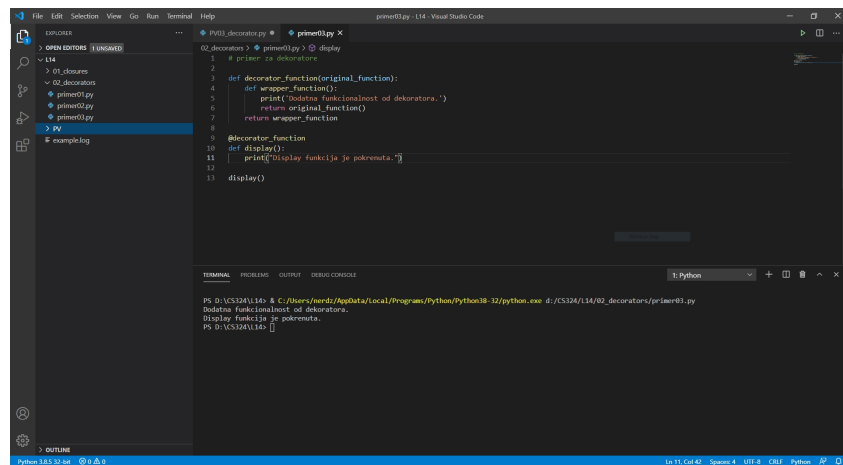
Na ovaj način moguće je pozvati bilo koju funkciju u dekorator.

Pozivanje dekoratora

Dekorator se poziva tako što se pre definicije originalne funkcije napiše ime spoljašnje funkcije sa znakom @:

```
@outer_func  
def orig_func(*args, **kwargs):  
    # ...
```

Nakon toga, potrebno je samo pozvati originalnu funkciju.



```
02_decorators.py | prime01.py | display
1 # primer za dekoratore
2
3 def decorator_function(original_function):
4     def wrapper_function():
5         print('Dodana funkcionalnost od dekoratora.')
6         return original_function()
7     return wrapper_function
8
9 @decorator_function
10 def display():
11     print('Display funkcija je pokrenuta.')
12
13 display()
```

Terminal

```
PS D:\CS324\14\2\decorators> python.exe d:/CS324/14/02_decorators/prime01.py
Dodana funkcionalnost od dekoratora.
Display funkcija je pokrenuta.
PS D:\CS324\14\2>
```

Slika 2.1 Pokretanje dekoratora. [Izvor: Autor]

▼ Poglavlje 3

Jinja notacija za Python jezik

INSTALACIJA JINJA2 PAKETA

Flaks koristi Jinja2 jezik za šablone pri izrazi veb aplikacija.



Slika 3.1 Jinja logo. [Izvor: <https://jinja.palletsprojects.com/en/2.11.x/>]

Da bi se razvile napredne veb aplikacije u Python jeziku korišćenjem Flask-a, potrebno je poznavati jezik za šablone Jinja.

Jezik za šablone (en. **templating language**) sadrži promenljive i logiku koja, kada se izvršava (ili renderuje u HTML), zamenjuje sa pravim vrednostima.

Jinja šabloni su jednostavne .html datoteke. Po konvenciji, u Flask projektima nalaze se u direktorijumu /templates.

Instalacija Jinja paketa

Instalacija jinja2 paketa vrši se preko pip komande:

```
>>> pip install jinja2
```

```
PS D:\CS324\L14> pip install jinja2
Collecting jinja2
  Downloading Jinja2-2.11.2-py3-none-any.whl (125 kB)
    |#####| 125 kB 3.3 MB/s
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\nerdz\appdata\local\programs\python\python38-32\lib\site-packages (from jinja2) (1.1.1)
Installing collected packages: jinja2
Successfully installed jinja2-2.11.2
PS D:\CS324\L14>
```

Slika 3.2 Instalacija jinja2 paketa. [Izvor: Autor]

Prilikom instalacije Flask, jinja2 se automatski instalira, ali poželjno je proveriti korišćenjem pip list komande.

JINJA 2 NOTACIJA

Jinja notacija poseduje tagove koji kontrolišu logiku šablona, i koji su inspirisani Django radnim okvirom za Python jezik

Paket Flask koristi Jinja2 notaciju za šablone prilikom generisanja HTML stranica.

Jinja2 šablon predstavlja tekstualnu datoteku. Jinja može generisati bilo koji tekstualni format (HTML, XML, CSV, LaTeX, i dr.). Jinja šablon ne poseduje sopstvenu ekstenziju već koristi neke od postojećih.

Takođe, Jinja notacija poseduje tagove koji kontrolišu logiku šablona, i koji su inspirisani Django radnim okvirom za Python jezik.

U nastavku je minimalni Jinja šablon koji ilustruje osnovnu Jinja konfiguraciju:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>My Webpage</title>
</head>
<body>
    <ul id="navigation">
        {% for item in navigation %}
            <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
        {% endfor %}
    </ul>

    <h1>My Webpage</h1>
    {{ a_variable }}

    {% # a comment %}
</body>
</html>
```

Podrazumevani Jinja tagovi

Naredbe (en. **statements**) se pišu između sledećih tagova:

```
{% ... %}
```

Izrazi (en. **expressions**) se pišu između sledećih tagova:

```
{{ ... }}
```

Komentari (en. **comments**) se pišu između sledećih tagova:

```
{# ... #}
```

Linijske naredbe (en. **line statements**) se pišu između sledećih tagova:

```
# ... ##
```

JINJA2 I PYTHON

Kada se izvršava (render-uje) Jinja šablon, potrebno je pozvati metodu `.render()`, sa argumentom koji daje vrednost nekom od korišćenih tagova.

Kada se Jinja2 uvozi u Python, treba uraditi sledeće

```
from jinja2 import Template
```

Ne uvozi se ceo jinja paket, već samo modul Template koji je potreban.

Najpre se pravi objekat klase Template koji kao argument može imati string sa nekim od Jinja tagova.

Kada se izvršava (render-uje) Jinja šablon, potrebno je pozvati metodu `.render()`, sa argumentom koji daje vrednost nekom od korišćenih tagova.

Primer:

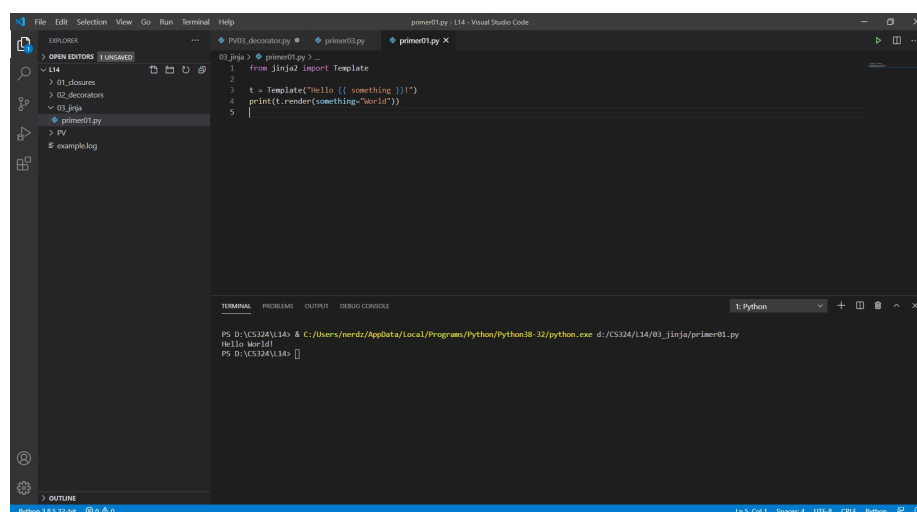
Napisati *"Hello world!"* aplikaciju koristeći Jinja šablone u Python jeziku, gde je world promenljiva *something* koju treba render-ovati.

Rešenje:

```
from jinja2 import Template

t = Template("Hello {{ something }}!")

print(t.render(something="World"))
```



Slika 3.3 Jinja2 Hello world aplikacija. [Izvor: Autor]

▼ Poglavlje 4

Šabloni za Flask mikroservis

POČETNA FLASK APLIKACIJA (BEZ ŠABLONA)

Flask podrazumevano pravi serverski proces na localhost-u (IP adresi 127.0.0.1) na portu 5000.

U nastavku obradiće se korišćenje Jinja šablona unutar Flask veb aplikacije.

Početna *flask* aplikacija jeste sledeća:

```
# $env:FLASK_APP = "flask_primer"
# $env:FLASK_DEBUG = 1
# flask run

from flask import Flask
app = Flask(__name__)

@app.route('/')
@app.route('/home')
def hello_world():
    return "<h2> Pocetna stranica! </h2>"

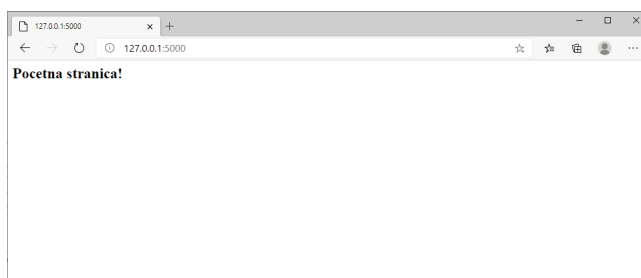
@app.route('/about')
def about():
    return "About stranica!"
```

Komentarisane linije koda jesu komande za flask unutar *powershell* terminala za Windows. Ukoliko se koristi *Linux* ili *macOS*, treba koristiti komandu *set*

Kao i ranije, najpre se importuje *flask*, i pravi objekat klase *Flask*.

Definišu se rute kroz dekoratore, sa nekim početnim vrednostima za početnu (*home*) i *about* stranice.

Flask podrazumevano pravi serverski proces na *localhost*-u (IP adresi 127.0.0.1) na portu 5000.



Slika 4.1 Počenta home stranica. [Izvor: Autor]

Slika 4.2 Početna about stranica. [Izvor: Autor]

POČETNA FLASK APLIKACIJA (SA HTML ŠABLONIMA)

Efikasniji način upisa HTML-a od direktnog jeste pravljenje šablona.

Unutar tela funkcije **home()** moguće je napisati i ceo HTML koji se želi prikazati, ali to nije praktično.

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

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

    <!DOCTYPE html>
    <html>
        <body>

            <h1>My First Heading</h1>
            <p>My first paragraph.</p>

        </body>
    </html>
    ...

@app.route('/about')
def about():
    return "About stranica!"
```

Efikasniji način upisa HTML-a jeste pravljenje šablona.

Za početak, treba napraviti direktorijum za šabloni (**templates**) u radnom direktorijumu:

```
|—templates
|—__pycache__
```

Unutar direktorijuma za šablone, napraviti šablon za svaku od stranica. Šablon predstavlja HTML dokument.

Za home stranicu, datoteka home.html izgledaće:

```
<!DOCTYPE html>
<html>
  <head>
    <title>CS324 - Skripting jezici</title>
  </head>
  <body>

    <h1>Pocetna stranica</h1>

  </body>
</html>
```

PRIKAZ FLASK APLIKACIJE SA HTML ŠABLONIMA

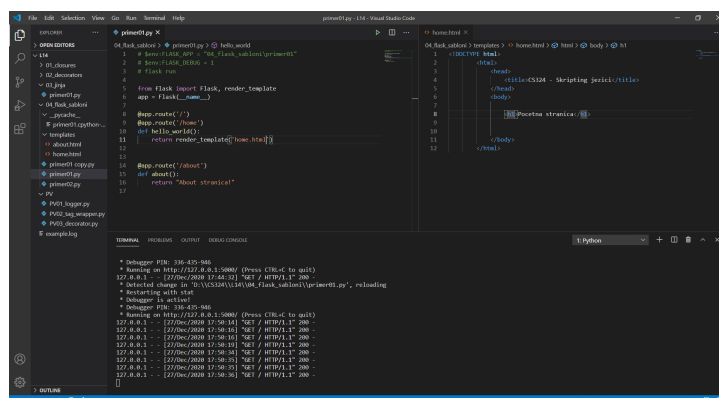
Modul `render_template` potražiće dokumente u "templates" direktorijumu flask projekta.

Nakon kreacije HTML šablona u odgovarajućem direktorijumu, potrebno je i uvesti u modul `render_template` iz `flask` paketa.

Nakon toga, umesto vraćanja direktno kodiranog HTML-a, funkcija treba da vrati poziv funkcije `render_template`, koji kao string ima HTML dokument. Podrazumevano, ovaj dokument tražiće se u direktorijumu `templates`.

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

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



Slika 4.3 Izgled HTML šablona u Flask aplikaciji. [Izvor: Autor]

Slika 4.4 Izgled početne stranice sa render-ovanim šablonom. [Izvor: Autor]

Slika 4.5 Izgled izvornog koda početne stranice. [Izvor: Autor]

▼ Poglavlje 5

Nastavak rada u izradi Flask veb aplikacije

DINAMIČKE VEB STRANICE I FLASK

Tokom rada savremene veb aplikacije, podaci se stalno ažuriraju.

Pri radu sa savremenim veb aplikacijama statičke stranice nisu dovoljne.

Tokom rada aplikacije, podaci se stalno ažuriraju.

U primeru, veb aplikacija koja je u izradi predstavlja blog. Recimo da je pozivom iz baze podataka dobijena lista imenika koji sadrži sledeće informacije

```
posts_var = [  
    {  
        'author': 'Autor 1',  
        'title': 'Blog post 1',  
        'content': 'Sadrzaj prvog posta',  
        'date_posted': datetime.now()  
    },  
    {  
        'author': 'Autor 2',  
        'title': 'Blog post 2',  
        'content': 'Sadrzaj drugog posta',  
        'date_posted': datetime.now()  
    }  
]
```

Najpre neka ovaj sadržaj bude direktno kodiran u .py datoteci.

```
# $env:FLASK_APP = "04_flask_sablioni\primer01"  
# $env:FLASK_DEBUG = 1  
# flask run  
  
from flask import Flask, render_template  
app = Flask(__name__)  
from datetime import datetime  
  
posts = [  
    {
```

```
        'author': 'Autor 1',
        'title': 'Blog post 1',
        'content': 'Sadrzaj prvog posta',
        'date_posted': datetime.now()

    },
    {
        'author': 'Autor 2',
        'title': 'Blog post 2',
        'content': 'Sadrzaj drugog posta',
        'date_posted': datetime.now()

    }

]

@app.route('/')
@app.route('/home')
def hello_world():
    return render_template('home.html', posts=posts_var)

@app.route('/about')
def about():
    return render_template('about.html')
```

U funkciji koja vraća glavnu stranicu, u **render_template** funkciji dodat je argument **posts=posts_var**, koji označava da u šablonu, kada naiđe na **posts**, treba da zameni sa promenljivom **posts_var**.

JINJA2 ŠABLONI U FLASK APLIKACIJI

Često se koriste ista imena za promenljive koje se nalaze u šablonu i koje se prosleđuju šablonu.

Unutar šablona za home stranicu (`templates/home.html`) treba ubaciti sadržaj promenljive **posts_var**.

Po pozivu funkcije **render_template**, sadržaj datoteke **posts_var** nalazi se u posts datoteci (**posts=posts_var**).

Često se koriste ista imena za promenljive koje se nalaze u šablonu i koje se prosleđuju šablonu. Međutim, ovde je namerno odvojeno da bi se razumelo koja se promenljiva gde navodi.

Izgled HTML datoteke sada je:

```
<!DOCTYPE html>
<html>
  <head>
    <title>CS324 - Skripting jezici</title>
```

```
</head>
<body>

    {% for post in posts %}
    <h1>{{ post.title }}</h1>
    <p> By {{ post.author }} on {{ post.date_posted }}</p>
    <p>{{ post.content }}</p>
    {% endfor %}

</body>
</html>
```

U okviru HTML datoteke sada se koristi Jinja notacija za šablone, i to:

Naredba:

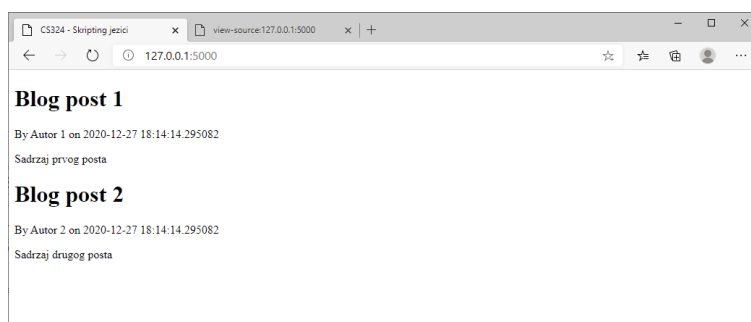
```
{% for post in posts %}
...
{% endfor %}
```

prolazi kroz sve elemente promenljive posts. U Python datoteci to je promenljiva **posts_var**.

Zatim, **izrazi**:

```
<h1>{{ post.title }}</h1>
<p> By {{ post.author }} on {{ post.date_posted }}</p>
<p>{{ post.content }}</p>
```

Ovi izrazi nalaze se unutar *for petlje* i pišu se samo u dvostrukim velikim zagradama. Kada se sačuva HTML dokument, stranica izgleda:



Slika 5.1 Izgled stranice nakon ubacivanja Jinja notacije. [Izvor: Autor]

USLOVNO GRANANJE U JINJA NOTACIJI

Moguće je dodati uslovno grananje u Jinja notaciji za šablone.

Unutar šablona za about stranicu, dodaće se još jedna promenljiva, **title**:

```
@app.route('/about')
def about():
    return render_template('about.html', title='About')
```

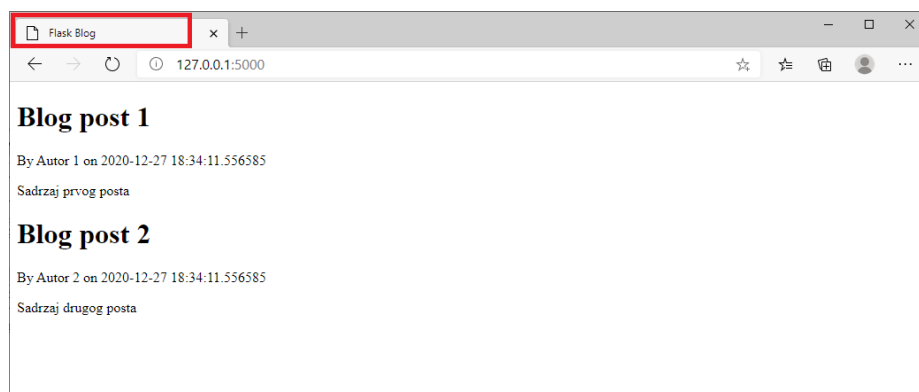
U HTML šablonu obe stranice, može se postaviti uslov da ukoliko postoji promenljiva title, da sadržaj promenljive title bude naslov stranice, a ukoliko nije ispunjen uslov, da bude neko podrazumevani naslov.

```
<!DOCTYPE html>
<html>
  <head>
    {% if title %}
      <title>Flask Blog - {{ title }}</title>
    {% else %}
      <title>Flask Blog</title>
    {% endif %}
  </head>
  <body>

    <h1>About stranica</h1>

  </body>
</html>
```

Na ovaj način, about stranica imaće naslov "Flask blog - About", dok će početna stranica imati podrazumevani naslov.



Slika 5.2 Početna stranica sa podrazumevanim naslovom. [Izvor: Autor]

Slika 5.3 About stranica sa novim naslovom. [Izvor: Autor]

NASLEĐIVANJE ŠABLONA

Moguće je naslediti opšti šablon, dok pojedinačni šabloni menjaju samo određene blokove

Primećuje se da je šablon koji se koristi za home i about stranicu jako sličan, samo se razlikuje deo sa tagom **<body>**. Zbog toga, moguće je napraviti opšti šablon, kojeg će ostali šabloni naslediti, dok će određene blokove zadržati.

Najpre, unutar templates direktorijuma treba napraviti novu datoteku **layout.html**. Ovo će biti datoteka opšteg šablona.

```
<!DOCTYPE html>
<html>
  <head>
    {% if title %}
      <title>Flask Blog - {{ title }}</title>
    {% else %}
      <title>Flask Blog</title>
    {% endif %}
  </head>
  <body>
    {% block content %}
    {% endblock %}
  </body>
</html>
```

Svi identični delovi jesu isti, samo je dodato da se u telu HTML strane nalazi blok **content**:

```
{% block content %}
{% endblock %}
```

Sada, u pojedinačnim šablonima treba dodati sledeće:

About:

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

{% block content %}
  <h1>About stranica</h1>
{% endblock %}
```

Home:

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

{% block content %}

    {% for post in posts %}

        <h1>{{ post.title }}</h1>
        <p> By {{ post.author }} on {{ post.date_posted }}</p>
        <p>{{ post.content }}</p>
    {% endfor %}

{% endblock %}
```

Na ovaj način funkcionalnost se ne gubi, a sve što je bilo zajedničko za šablone nasledilo se od glavnog šablona **layout.html**, komadom:

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

✓ Poglavlje 6

Pokazne vežbe

ZADATAK #1

Zadatak #1 odnosi se na funkcije prve klase i okvirno se radi 15 minuta.

Zadatak #1 (15 minuta)

Napisati program koji pravi aktivnosti pozvanih funkcija. Definirati funkciju logger, koja uzima kao parametar funkciju func.

Unutar ove funkcije, definirati novu funkciju log_func, sa proizvoljnim brojem argumenata, koja vraća text koji kaže koja se funkcija pokreće i sa kojim parametrima, i štampa pokrenutu funkciju func. Zati, vraća log_func kao povratnu vrednost.

Definirati jednostavne funkcije za sabiranje i oduzimanje dva broja, napraviti odgovarajuće logger funkcije, i isprobati funkcionalnost.

Koristiti sledeći paket za logovanje:

```
# uvoz paketa
import logging
logging.basicConfig(filename='example.log', level=logging.INFO)

# pravljenje log datoteke:
logging.info(
    'Running "{}" with arguments {}'.format(func.__name__, args))
```

```
# Pokazna vezba Zadatak #1

import logging
logging.basicConfig(filename='example.log', level=logging.INFO)

def logger(func):
    def log_func(*args):
        logging.info(
            'Running "{}" with arguments {}'.format(func.__name__, args))
        print(func(*args))
    return log_func

def add(x, y):
```



```
    return x+y

def sub(x, y):
    return x-y

add_logger = logger(add)
sub_logger = logger(sub)

add_logger(3, 3)
add_logger(4, 5)

sub_logger(10, 5)
sub_logger(20, 10)
```

ZADATAK #2

Zadatak #2 odnosi se na zatvorenje funkcije i okvirno se radi 15 minuta.

Zadatak #2 (15 minuta)

Napisati funkciju **html_tag(tag)**, koja ima ulazni parametar string **tag**.

Unutar ove funkcije, definisati funkciju **wrap_text(msg)**, koja štampa string **msg** okružen otvorenim i zatvorenim **html** tagovima.

Funkcija **html_tag** vraća (bez pozivanja) funkciju **wrap_text**.

U glavnom programu, napraviti funkcije za štampanje **h1** i **p** tagova, i ispitati funkcionalnost.

```
def html_tag(tag):

    def wrap_text(msg):
        print('<{0}>{1}</{0}>'.format(tag, msg))

    return wrap_text

print_h1 = html_tag('h1')
print_h1('Test Headline!')
print_h1('Another Headline!')

print_p = html_tag('p')
print_p('Test Paragraph!')
```

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADATAK #3

Zadatak #3 odnosi se na dekoratore i okvirno se radi 15 minuta.

Zadatak #3 (15 minuta)

Napisati funkciju koja računa i štampa faktorial unetog broja. Može se koristiti math modul.

Zatim, napraviti dekorator **vreme_izvršenja** koji računa vreme izvršenja unete funkcije. Koristiti time modul za dobijanje razlike u vremenu.

```
def vreme_izvršenja(orig_f):
    import time

    def wraper_func(*args, **kwargs):

        start = time.time()
        orig_f(*args, **kwargs)
        stop = time.time()

        print(f'Vreme izvršenja funkcije {orig_f.__name__} je {stop - start}')

    return wraper_func

@vreme_izvršenja
def factorial(num):
    import math, time
    print(math.factorial(num))

factorial(52)
```

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 7

Individualne vežbe

INDIVIDUALNE VEŽBE #14

Individualne vežbe #14 rade se okvirno 90 minuta.

Individualne vežbe #14 sastoje se iz dva do tri zadatka.

Vreme za izradu svih zadataka okvirno je 90 minuta.

Studenti u dogovoru sa predmetnim nastavnikom i asistentom dobijaju zadatke individualnih vežbi.

▼ Poglavlje 8

Domaći zadatak

DOMAĆI ZADATAK #14

Domaći zadatak #14 okvirno se radi 2h

Domaći zadatak #14 daje se u dogovoru sa predmetnim nastavnikom i/ili asistentom.

Predaja domaćeg zadatka:

Tradicionalni studenti:

Domaći zadatak treba dostaviti najkasnije 7 dana nakon predavanja, za 100% poena. Nakon toga poeni se umanjuju za 50%.

Internet studenti:

Domaći zadatak treba dostaviti najkasnije 10 dana pred polaganje ispita. Domaći zadaci se brane!

Domaći zadatak poslati dr Nemanji Zdravkoviću: nemanja.zdravkovic@metropolitan.ac.rs

Obavezno koristiti uputstvo za izradu domaćeg zadatka.

Uz .doc dokument (koji treba sadržati i screenshot svakog urađenog zadatka kao i komentare za zadatak), poslati i izvorne i dodatne datoteke.

▼ Poglavlje 9

Zaključak

ZAKLJUČAK

Zaključak lekcije #14

Rezime:

U ovoj lekciji najpre su objašnjeni koncepti funkcija prve klase, zatvorenja funkcija, kao i dekoratera kod Python jezika.

Nakong toga, objašnjena je jinja2 notacija za šablone koja se koristi pri izradi Flask veb aplikacija.

Konačno, nastavljen je rad sa Flask mikroservisom, ovog puta dodavanjem šablona i prikazom promenljivih iz Python dela u HTML deo aplikacije.

Literatura:

- Tarek Ziade;, Python - Razboj mikroservisa, Kompjuter biblioteka, 2017. (prevod iste knjige Packt Publishing-a)
- David Beazley, Brian Jones, Python Cookbook: Recipes for Mastering Python 3, 3rd edition, O'Reilly Press, 2013.
- Mark Lutz, Learning Python, 5th Edition, O'Reilly Press, 2013.
- Andrew Bird, Lau Cher Han, et. al, The Python Workshop, Packt Publishing, 2019.
- Al Sweigart, Automate the boring stuff with Python, 2nd Edition, No Starch Press, 2020.