ЛЕКЦИЯ №6 ПО ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Москва, 2020

Обработка ошибок

```
In [4]: def divide(a,b):
            try:
                return a/b
            except ZeroDivisionError as ex:
                print(f"an error: {ex}")

In [5]: divide(10,2)

Out[5]: 5.0

In [6]: divide(2,0)

        an error: division by zero
```

```
def divide(a,b):
    try:
        res = a/b
        return res
    except ZeroDivisionError as ex:
        print(f"an error: {ex}")
    except:
        print("unknown error!")
```

Блок finally

Полезны при закрытии указателей на файлы. Очистка низкоуровневых ресурсов

```
file = None
try:
    file = open(r'C:\ggfff.txt')
    data = file.read()
finally:
    print('Finally')
    if file:
        file.close()
```

Пример функции ввода числа с проверками

```
def get_int():
    while True:
        try:
            reply = int(input('Enter a number...'))
            return reply
        except:
            print('Not a number! Try again')
            continue
```

Можно создавать свои собственные исключения

Исключения

```
import math
```

```
def calc(a,b,c):
    p = (a+b+c)/2
    s= math.sqrt(p*(p-a)*(p-b)*(p-c))
    return s
```

```
s = calc(-12,12,3)
```

Функция не защищена от некорректных аргументов

Исключения

```
import math
```

```
def calc(a,b,c):
    if (a<=0) or (b<=0) or (c<=0):
        raise ValueError("One of the sides <=0")
    p = (a+b+c)/2
    s= math.sqrt(p*(p-a)*(p-b)*(p-c))
    return s
```

```
s = calc(-12,12,3)
```

ValueError – стандартное исключение

Классы, объекты

```
class Character():
    pass
```

```
obj = Character()
```

```
type(obj)
```

```
__main__.Character
```

Классы, объекты

```
class Character():
    pass
```

```
obj = Character()
```

```
type(obj)
```

```
__main__.Character
```

Конструктор

```
class Character():
    def __init__(self, race, dem1 = 10, dem2 = 15):
        #self - ссылку на текущий объект
        # объявление переменной на уровне класса
        self.race = race
        self.dem1 = dem1
        self.dem2 = dem2
```

```
obj = Character("aa", dem1  =55)
```

```
obj.race
```

```
'aa'
```

Статические поля

```
class Character():
    maxSpeed = 100
    minSpeed = 0
    def __init(self, race):
        self.race = race
```

```
Character.maxSpeed
```

```
100
```

Методы класса

```
class Character():
    maxSpeed = 100
    minSpeed = 0
    def __init(self, race):
        self.race = race
```

```
Character.maxSpeed
```

```
100
```

Приватных и защищенных полей в питоне нет

```
class Character():
    maxSpeed = 100
    minSpeed = 0
    def __init__(self, race):
        self.race = race
    def hit(self, damage):
        self.damage = damage
```

```
Character.maxSpeed
```

```
100
```

```
obj = Character("ff")
```

```
obj.hit(50)
```

```
obj.damage
```

```
50
```

Модификаторы доступа

Два подчеркивания – приватный доступ, одно подчеркивания - защищенный

```
class Character():
    MAX_SPEED = 100
    def __init__(self, damage = 66, race = 55):
        self.damage = damage

        self.__race = race
```

```
c = Character(66)
```

```
c.
```

damage
MAX_SPEED

Модификаторы доступа

Одно подчеркивание – защищенный

```
class Character():
    MAX_SPEED = 100
    def __init__(self, race, damage = 66):
        self.damage = damage

        self.__race = race
        self._health = 55
    def hit(self):
        self._health = 44
```

```
c = Character(77)
```

```
c.hit()
```

Свойства

```
class Character():
    MAX_SPEED = 100
    def __init__(self, race, damage = 66):
        self.damage = damage

        self.__race = race
        self.__health = 55
    @property
    def health(self):
        return self.__health
```

```
c = Character(77)
```

```
c.health
```

```
55
```

Свойства могут иметь логику.
Свойства – это среднее Между методами и полями.

Свойства

```
class Character():
    MAX_SPEED = 100
    def __init__(self, race, damage = 66):
        self.damage = damage

        self.__race = race
        self.__health = 55
        self.__current_speed = 10
    @property
    def health(self):
        return self.__health

    @property
    def current_speed(self):
        return self.__current_speed
```

```
c = Character(77)
```

```
c.health
```

```
55
```

```
c.current_speed
```

```
10
```

Свойства

```
class Character():
    MAX_SPEED = 100
    def __init__(self, race, damage = 66):
        self.damage = damage

        self.__race = race
        self.__health = 55
        self.__current_speed = 10
    @property
    def health(self):
        return self.__health

    @property
    def current_speed(self):
        return self.__current_speed
```

```
c = Character(77)
```

```
c.health
```

```
55
```

```
c.current_speed
```

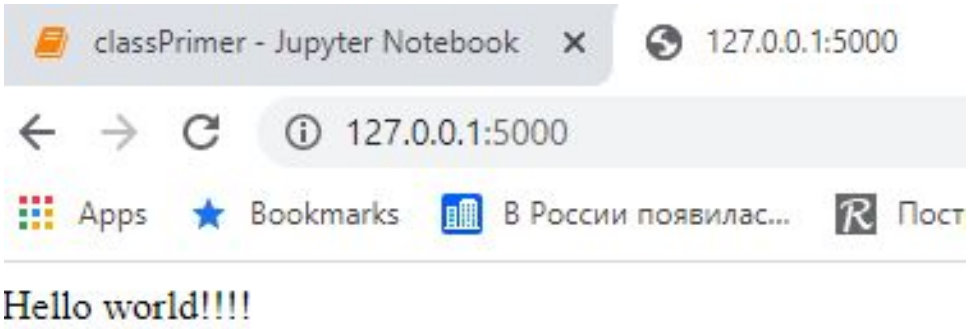
```
10
```

Свойства

```
@property
def current_speed(self):
    return self.__current_speed

@current_speed.setter
def current_speed(self, current_speed):
    if current_speed<0:
        self.__current_speed = 0
    elif current_speed >100:
        self.__curent_speed = 100
    else:
        self.__current_speed = current_speed
```

Библиотека FLASK

conda install -c anaconda flask
set FLASK_APP=flask_ex.py



Веб сервер FLASK

```
from flask import Flask

app = Flask(__name__)
@app.route('/')
def hello() ->str:
      return 'Hello world!!!!'
app.run()
```

Декоратор route позволяет связать URL с существующей функцией hello
route – организует вызов указанной функции веб-сервером Flask когда тот получает запрос /
Затем декоратор ожидает вывод от декорируемой функции чтобы передать его ожидающему веб- браузеру
app.run – запуск веб-сервера

1 **Метод GET**

Браузеры обычно используют этот метод для запроса ресурса с веб-сервера, и он, безусловно, является наиболее часто используемым. (Мы сказали «обычно», потому что метод *GET* можно также использовать для *отправки* данных из браузера на сервер, что довольно странно, но мы не будем заострять внимание на этой возможности.) Все URL из нашего веб-приложения сейчас поддерживают метод GET, который в Flask является HTTP-методом по умолчанию.

2 **Метод POST**

Этот метод позволяет веб-браузеру посылать данные на сервер через HTTP, и он тесно связан с HTML-тегом `<form>`. Вы можете потребовать от своего Flask веб-приложения принимать данные, отправленные браузером, добавив еще один аргумент в строку `@app.route`.

Шаблоны Jinja2

Если мы когда-нибудь захотим изменить статический текст, такой как тот, который появляется в наших заголовках, мы должны отредактировать наши файлы Python
Механизмы шаблонов позволяют программистам применять объектно-
ориентированные понятия наследования и повторного использования при создании текстовых данных, таких как веб-странички.
Вид и оформление веб-сайта можно определить как высокоуровневый HTML-шаблон, известный как базовый шаблон, который затем наследуют другие HTML-странички. Если внести изменения в базовый шаблон, они отразятся на всех HTML-страничках, наследующих его.
В состав Flask входит простой и мощный механизм шаблонов, который называетсяJinja2.
Более подробную информацию о возможностях Jinja2 можно посмотреть по ссылке:
http://jinja.pocoo.org/docs/dev/

Базовый шаблон

```
<!doctype html>
<html>
    <head>
        <title>{{ the_title }}</title>
        <link rel="stylesheet" href="static/hf.css" />
    </head>
    <body>
        {% block body %}

        {% endblock %}
    </body>
</html>
```

Директива Jinja2 указывает, что значение будет вставлено перед отправкой страницы (считайте ее аргументом шаблона).

Таблица стилей определяет внешний вид веб-страничку

Директивы Jinja2 указывают, что сюда будет подставлен блок HTML, который определяется страничкой, наследующей шаблон.

Шаблон

После подготовки базового шаблона его можно наследовать, используя директиву
Jinja2 extends. В этом случае наследующие HTML-файлы должны определить
разметку HTML только для именованных блоков в базовом шаблоне
Ниже приведена разметка для первой из наших страничек, с именем entry.html. Эта разметка содержит HTML-форму, с помощью которой пользователь может передать значения phrase и letters, ожидаемые веб-приложением.
Заметьте, что «шаблонная» разметка HTML из базового шаблона не повторяется в этом файле, так как директива extends включает всю его разметку.
Нам нужно определить только разметку HTML для данной конкретной странички, и мы сделаем это, поместив ее внутрь блока Jinja2 с именем body

Шаблон

```
{% extends 'base.html' %}

{% block body %}

<h2>{{ the_title }}</h2>

<form method='POST' action='/search4'>
<table>
<p>Use this form to submit a search request:</p>
<tr><td>Phrase:</td><td><input name='phrase' type='TEXT' width='60'></td></tr>
<tr><td>Letters:</td><td><input name='letters' type='TEXT' value='aeiou'></td></tr>
</table>
<p>When you're ready, click this button:</p>
<p><input value='Do it!' type='SUBMIT'></p>
</form>

{% endblock %}
```

Этот шаблон наследует базовый шаблон и определяет содержимое блока с именем «body».

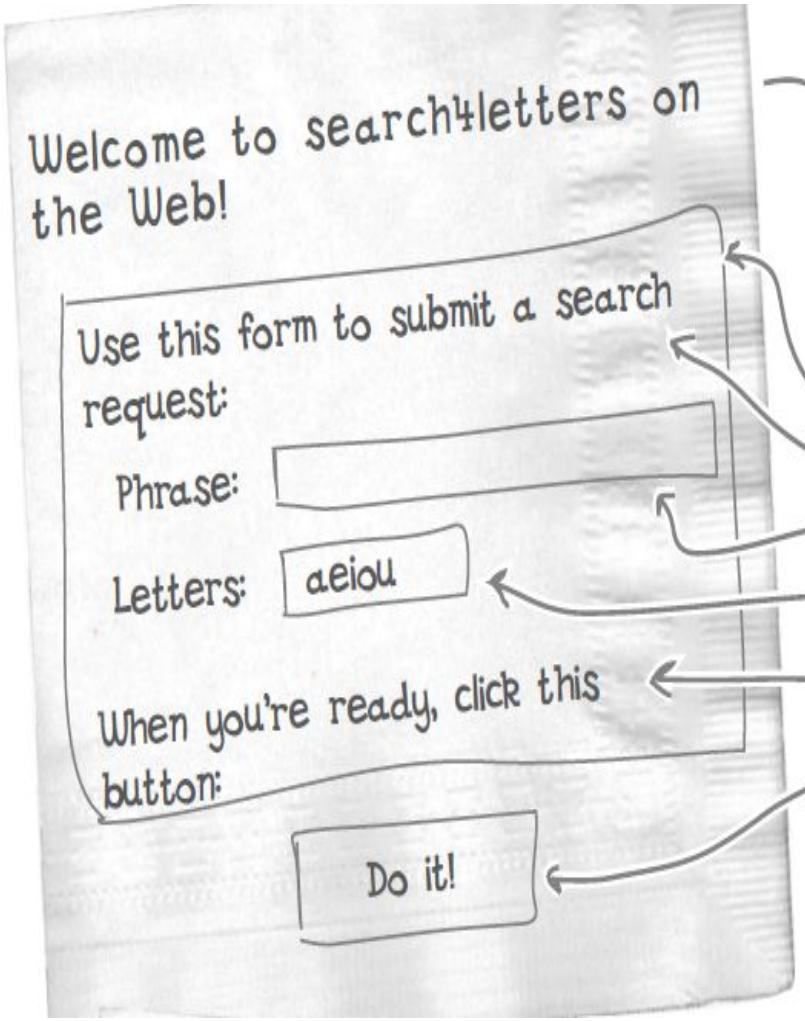
Шаблон

```
{% extends 'base.html' %}

{% block body %}

<h2>{{ the_title }}</h2>

<p>You submitted the following data:</p>
<table>
<tr><td>Phrase:</td><td>{{ the_phrase }}</td></tr>
<tr><td>Letters:</td><td>{{ the_letters }}</td></tr>
</table>

<p>When "{{the_phrase }}" is search for "{{ the_letters }}", the following
results are returned:</p>
<h3>{{ the_results }}</h3>

{% endblock %}
```

Как и «entry.html», этот шаблон также наследует базовый шаблон и также определяет содержимое блока с именем «body».

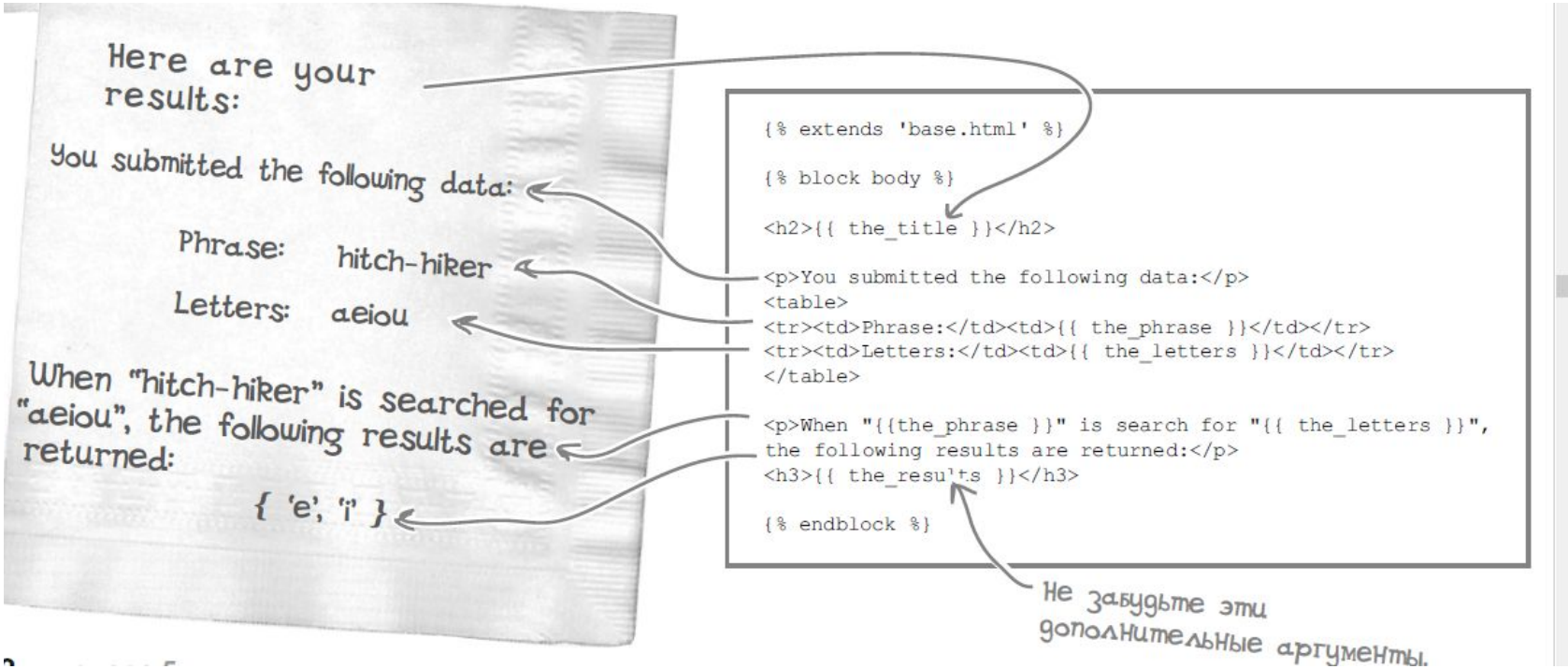
Обратите внимание на дополнительные аргументы, значения которых должны быть определены перед отображением.

Шаблон




```
{% extends 'base.html' %}

{% block body %}

<h2>{{ the_title }}</h2>

<form method='POST' action='/search4'>
<table>
<p>Use this form to submit a search request:</p>
<tr><td>Phrase:</td><td><input name='phrase' type='TEXT'
width='60'></td></tr>
<tr><td>Letters:</td><td><input name='letters' type='TEXT'
value='aeiou'></td></tr>
</table>
<p>When you're ready, click this button:</p>
<p><input value='Do it!' type='SUBMIT'></p>
</form>

{% endblock %}
```

Шаблон


Here are your results:
You submitted the following data:
Phrase: hitch-hiker
Letters: aeiou
When "hitch-hiker" is searched for "aeiou", the following results are returned:
{ 'e', 'i' }
{% extends 'base.html' %}
{% block body %}
<h2>{{ the_title }}</h2>
You submitted the following data:
<table>
<tr><td>Phrase:</td><td>{{ the_phrase }}</td></tr>
<tr><td>Letters:</td><td>{{ the_letters }}</td></tr>
</table>
When "{{the_phrase }}" is search for "{{ the_letters }}",
the following results are returned:
<h3>{{ the_results }}</h3>
{% endblock %}
Не забудьте эти дополнительные аргументы.

Отображение шаблонов из Flask

Чтобы отобразить HTML-форму с помощью шаблона `entry.html`, нужно внести несколько изменений в код выше.

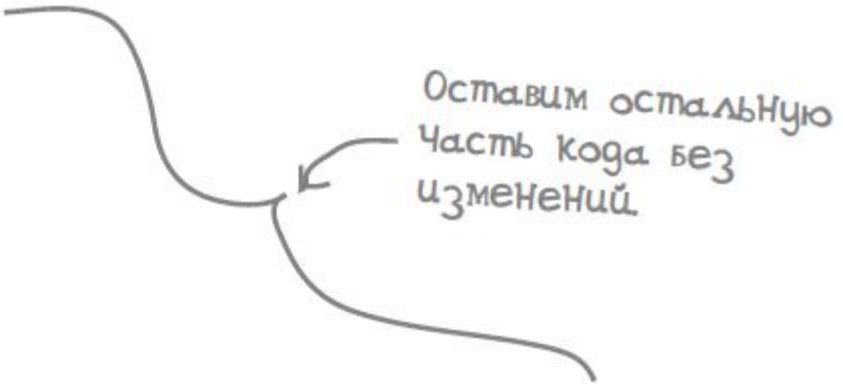
1. **Импортировать функцию** `render_template`.
   Добавим `render_template` в список импорта, в строке `from flask` в начале программы.

2. **Создать новый URL — в данном случае** `/entry`.
   Каждый раз, когда вам понадобится добавить новый URL в Flask веб-приложение, добавьте также новую строку `@app.route`. Мы сделаем это перед строкой с вызовом `app.run()`.

3. **Создать функцию, которая вернет сконструированную разметку HTML.**
   Со строкой `@app.route` можно связать функцию, которая выполнит всю фактическую работу (и сделает веб-приложение более полезным для пользователей). В этой функции мы вызовем `render_template` (и получим результат), передав ей имя файла шаблона (`entry.html` в этом случае), а также значения всех аргументов, необходимых шаблону (в этом случае мы должны передать значение для `the_title`).

Отображение шаблонов из Flask

```
from flask import Flask, render_template
from vsearch import search4letters

app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello world from Flask!'

@app.route('/search4')
def do_search() -> str:
    return str(search4letters('life, the universe, and everything', 'eiru,!'))

@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')

app.run()
```



Коды ошибок html

Есть пять основных категорий кодов состояния: 100-е, 200-е, 300-е, 400-е и 500-е.
Коды из диапазона 100–199 — это информационные сообщения: они говорят, что все в порядке, и сервер сообщает детали, относящиеся к клиентскому запросу.
Коды из диапазона 200–299 —это сообщения об успехе: сервер получил, понял и обработал запрос. Все в порядке.
Коды из диапазона 300– 399 — это сообщения перенаправления: сервер информирует клиента, что запрос может быть обработан в каком-то другом месте.
Коды из диапазона 400–499 — это сообщения об ошибках на стороне клиента: сервер не смог понять и обработать запрос. Как правило, виноват
в этом клиент.
Коды из диапазона 500–599 — это сообщения об ошибках на стороне сервера: сервер получил запрос, но в процессе обработки на сервере возникла ошибка. Как правило, виноват в этом сервер.

Коды ошибок html

Есть пять основных категорий кодов состояния: 100-е, 200-е, 300-е, 400-е и 500-е.
Коды из диапазона 100–199 — это информационные сообщения: они говорят, что все в порядке, и сервер сообщает детали, относящиеся к клиентскому запросу.
Коды из диапазона 200–299 —это сообщения об успехе: сервер получил, понял и обработал запрос. Все в порядке.
Коды из диапазона 300– 399 — это сообщения перенаправления: сервер информирует клиента, что запрос может быть обработан в каком-то другом месте.
Коды из диапазона 400–499 — это сообщения об ошибках на стороне клиента: сервер не смог понять и обработать запрос. Как правило, виноват
в этом клиент.
Коды из диапазона 500–599 — это сообщения об ошибках на стороне сервера: сервер получил запрос, но в процессе обработки на сервере возникла ошибка. Как правило, виноват в этом сервер.

Исходный код программы на питоне для отображения формы

```
from flask import Flask, render_template


app = Flask(__name__)

def search4letters(phrase: str, letters: str='aeiou') -> set:
    """Returns the set of 'letters' found in 'phrase'."""
    return set(letters).intersection(set(phrase))

@app.route('/')
def hello() -> str:
    return 'Hello world from Flask!'


@app.route('/search4', methods=['POST'])
def do_search() -> str:
    return str(search4letters('life, the universe, and everything', 'eiru,!'))

@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')


app.run(debug=True)
```