Python

AioHttp

Менеджер пакетов

`Python Package Index (PyPI)` – репозиторий пакетов, открытый для всех Python разработчиков. На данный момент более 300 000 пакетов.

`pip` - консольная утилита для скачивания и установки пакетов.

Менеджер пакетов

`pip install package_name` - установка пакета (пакетов).
`pip uninstall package_name` - удаление пакета (пакетов).
`pip list` - список установленных пакетов.
`pip show package_name` - показывает информацию об установленном пакете.
`pip help` - помощь по доступным командам.

Виртуальная среда Python

- одновременно в среде может быть только одна версия пакета;
- разным проектам требуются разные наборы пакетов;
- требуется легко воспроизводить окружение;
- закрыт доступ к основному каталогу с python.

Способы виртуализации

1. Virtualenv
2. Conda environments

Virtualenv

```
pip install virtualenv
virtualenv <path to environment>

<path to environment>\Scripts\activate.bat  # для windows
source <path to environment>/bin/activate  # для OS X и Linux
```

PyCharm

File => Settings => Project: <name> => Project Interpreter

Как работает браузер?

- Разрешение DNS
- На полученный ip отправляет запрос по HTTP/HTTPS
- Запрос обрабатывает веб-сервер (nginx, apache...)
- Веб-сервер передает запрос в приложение
- Приложение возвращает ответ веб-серверу
- Веб-сервер возвращает ответ в браузер

Из чего состоит HTTP запрос?

- Method (GET, POST, PUT, DELETE ...)
- Path - путь до запрашиваемого ресурса
- Версия http протокола
- Headers - заголовки запроса (Host, Cookie, Authorization, кэширование, информация о теле запроса ...)
- Body - тело запроса

Из чего состоит HTTP запрос?

```
POST /sample_page.html HTTP/1.1
Host: www.example.org
User-Agent: Mozilla/5.0 (Windows NT x.y; rv:10.0)
Cookie: yummy_cookie=choco; tasty_cookie=strawberry

<Body>
```

HTTP-методы

Запросы без body:

GET, HEAD, OPTIONS, CONNECT, TRACE

Запросы с body:

POST, PUT, DELETE, PATCH

Из чего состоит HTTP ответ?

- Версия http протокола
- Status Code
- Status Message
- Headers - заголовки запроса (Host, Set-Cookie, кэширование, информация о теле запроса ...)
- Body - тело ответа

Из чего состоит HTTP ответ?

```
HTTP/1.1 200 OK
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Content-Length: 29769
Content-Type: text/html

<Body>
```

HTTP-коды ответа

1xx - информирование о процессе передачи
2xx - успех
3xx - перенаправление
4xx - ошибка клиента (418 I’m a teapot)
5xx - ошибка сервера

AioHttp

`AioHttp` - асинхронный HTTP клиент/сервер для asyncio.

Документация по HTTP-серверу:
https://docs.aiohttp.org/en/stable/web.html

Установка AioHttp

Менеджер пакетов

```
pip install aiohttp
```

PyCharm

Settings => Project:<name> => Project Interpreter

Alt+Insert (или знак “+” в правом углу)

Ищем нужный пакет, далее Install Package

Простое приложение

Создадим директорию lecture_x/web_app и в ней файл app.py:

```
from aiohttp import web

async def hello(request):
    return web.Response(text="Hello, world")

app = web.Application()
app.add_routes([web.get('/', hello)])

if __name__ == '__main__':
    web.run_app(app)
```

Простое приложение

app.py:

```
from aiohttp import web  # импортируем модуль web

async def hello(request): # корутина принимает request
    return web.Response(text="Hello, world") # возвращает response

app = web.Application()  # Создаем приложение
app.add_routes([web.get('/', hello)]) # Указываем путь и функцию

if __name__ == '__main__':
    web.run_app(app)  # Запускаем приложение
```

Простое приложение

```
async def new_page(request):
    return web.Response(text='Привет! Это новая страница!')

app.add_routes([
    web.get('/', hello),
    web.get('/new', new_page)
])
```

Теперь на странице http://localhost:8080/new будет выведен текст “Привет! Это новая страница!”

Простое приложение с параметрами

```
async def new_page(request):
    return web.Response(text=f'Text: {request.match_info["word"]}')

app.add_routes([web.get('/new/{word}', new_page)])
```

Страница:
http://127.0.0.1:5000/new/ - не найдена.
http://127.0.0.1:5000/new/test - выведен текст.

Можем указывать регулярное выражение:

```
app.add_routes([web.get(r'/new/{word:\d+}', new_page)])
```

Шаблоны

Шаблоны не доступны в основном пакете `aiohttp` и идут в дополнительной библиотеке `aiohttp_jinja2`

```
pip install aiohttp_jinja2
```

Шаблоны

```
async def index():
    user = {'username': 'Nick'}
    result = '''
<html>
    <head>
        <title>Home Page - Microblog</title>
    </head>
    <body>
        <h1>Hello, ''' + user['username'] + '''!</h1>
    </body>
</html>'''
    return web.Response(text=result)
```

Шаблоны

Создадим директорию: templates

Создадим в ней файл index.html

Добавим в файл строку из прошлого примера:

```
<html>
    <head>
        <title>Home Page - Microblog</title>
    </head>
    <body>
        <h1>Hello, username!</h1>
    </body>
</html>
```

Шаблоны

```
import aiohttp_jinja2
import jinja2

async def index(request):
    return aiohttp_jinja2.render_template('index.html', request)


...
aiohttp_jinja2.setup(app,
loader=jinja2.FileSystemLoader('templates'))
```

Шаблонизатор - Jinja2

index.html

```
<html>
    <head>
        <title>{{ title }} - Microblog</title>
    </head>
    <body>
        <h1>Hello, {{ user.username }}!</h1>
    </body>
</html>
```

Шаблонизатор - Jinja2

```
async def index(request):
    context = {'title': 'Home', 'user': {'username': 'Nick'}}
    return aiohttp_jinja2.render_template('index.html', request,
context)
```

Шаблонизатор - Jinja2

Документация:

https://jinja.palletsprojects.com/en/2.10.x/

Условный оператор в Jinja2

```
{% if kenny.sick %}
   Kenny is sick.
{% elif kenny.dead %}
   You killed Kenny!  You bastard!!!
{% else %}
   Kenny looks okay --- so far
{% endif %}
```

Цикл for в Jinja2

```
{% for user in users %}
  {{ user.username }}<br>
{% endfor %}
```

Формы

input_name.html

```
<html>
    <head>
    <title>Home Page - Microblog</title>
    </head>
    <body>
    <h1>Введи имя:</h1>
    <form action="">
        <input type="text" name="username">
        <button type="submit">Отправить!</button>
    </form>
    </body>
</html>
```

Формы

```
async def input_name(request):
    username = request.rel_url.query.get('username')
    if username:
        return render_template('print_name.html', request, {'name':
username})
    return render_template('input_name.html')
```

Формы

print_name.html

```
<html>
    <head>
        <title>Home Page - Microblog</title>
    </head>
    <body>
        <h1>Привет, {{name}}!</h1>
    </body>
</html>
```

POST-запросы

input_name.html

```
<html>
    <head>
    <title>Home Page - Microblog</title>
    </head>
    <body>
    <h1>Введи имя:</h1>
    <form action="" method="post">
        <input type="text" name="username">
        <button type="submit">Отправить!</button>
    </form>
    </body>
</html>
```

POST-запросы

```
async def input_name(request):
    if request.method == 'POST':
        data = await request.post()
        return render_template('print_name.html', request, {'name':
data['username']})
    return render_template('input_name.html')

app.add_routes([
    web.get('/input_name', input_name),
    web.post('/input_name', input_name)
])
```

POST - загрузка файлов

В шаблоне:

```
<form action="" method="post" enctype="multipart/form-data">
    <input name="some_file" type="file" value=""/> ...
```

В хэндлере запроса:

```
async def load_file(request):
    data = await request.post()
    some_file = data['some_file'] # объект класса FileField
    filename = some_file.filename
    content = some_file.file.read()
```

Вспомогательные методы

При определении роута, можно задать ему имя:

```
web.get(r'/new/{word:\d+}', new_page, name='new')

request.app.router['new'].url_for(word='333').with_query({.
..})

async def handler(request):
    location = request.app.router['login'].url_for()
    raise web.HTTPFound(location=location)
```

Заголовки, cookies, json

```
async def handler(request):
    user_agent = request.headers.get('User-Agent')

    count = request.cookies.get('count', 0)
    count = int(count) + 1
    response = web.Response(text=f'Посещений: {count}')
    response.set_cookie('count', str(count))
    return response

async def handler(request):
    data = {'some': 'data'}
    return web.json_response(data)
```

Быстрое прототипирование

https://getbootstrap.com/

Набор html-элементов, css и js для быстрого создания веб-приложений.

Быстрое прототипирование

```
<html>
    <head>
        …
        <link href="..." ...>
        <script src="..." ...></script>
    </head>
    <body>
        …
    </body>
</html>
```

Упражнение

Создать приложение, позволяющее сложить/вычесть/перемножить/поделить 2 числа. Вернуться со страницы результата на ввод новых значений. Сохранять последние 3 операции и их результат для текущего пользователя.

Для выбора операции используйте html тэг select:

```
<select name="operation">
    <option selected value="add">Сумма</option>
    <option value="sub">Вычитание</option>
    <option value="mul">Произведение</option>
    <option value="div">Деление</option>
</select>
```

Домашнее задание

Приложение состоит из 2 страниц:

1. Главная страница - список статей и кнопка добавить новую.
2. Создание статьи - страница с формой создания.

(статьи хранить в глобальной переменной)

Задания:

1. На главной странице вывести список статей (сортировка от новых к старым). Для каждой статьи должны быть: заголовок, дата создания, текст, теги.
   Если статей больше 5, должна быть постраничная навигация с шагом в 5 страниц.

Домашнее задание

Задания:

2. На странице с формой создания статьи:
    a. Добавить ввод названия, текста, списка тегов. Теги вводятся через запятую или пробел.
    b. После создания статьи необходимо перенаправить пользователя на главную страницу.
    c. Если не заполнены какие то из полей, вывести соответствующую ошибку.