Приглашаем преподавателей поучаствовать в интервью с нашей командой исследований. Если вы готовы участвовать, пожалуйста, заполните форму. Приглашение на интервью.

Урок aiohttp

Авторские решения

к уроку Использование aiohttp для создания веб-сервисов и запросов

Классная работа

Асинхронный GET-запрос

```
import aiohttp
import asyncio

async def fetch(url):
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as response:
            return await response.text()

async def main():
    url = 'https://api.github.com'
    result = await fetch(url)
    print(result)

asyncio.run(main())
```

Асинхронная обработка нескольких URL

```
import aiohttp
import asyncio

async def fetch(url, session):
    async with session.get(url) as response:
        return await response.text()
```

```
async def fetch_all(urls):
    async with aiohttp.ClientSession() as session:
        tasks = [fetch(url, session) for url in urls]
        return await asyncio.gather(*tasks)

async def main(ulrs):
    results = await fetch_all(urls)
    for result in results:
        print(result)

urls = ['https://httpbin.org/get', 'https://jsonplaceholder.typi
asyncio.run(main(urls))
```

Обработка ошибок в HTTP-запросах

```
import aiohttp
import asyncio

async def fetch_with_error_handling(url):
    async with aiohttp.ClientSession() as session:
        try:
            async with session.get(url, timeout=5) as response:
                return await response.text()
        except aiohttp.ClientError as e:
            return f"Request failed: {e}"
        except asyncio.TimeoutError:
            return "Request timed out"

async def main(url):
    result = await fetch_with_error_handling(url)
    print(result)

url = 'https://nonexistent-url.com'

asyncio.run(main(url))
```

Асинхронное создание HTTP-сервера

```
from aiohttp import web

async def handle(request):
    return web.json_response({'message': 'Hello, world
```

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

web.run_app(app)
```

Реализация простого REST API

```
from aiohttp import web

users = []

async def get_users(request):
    return web.json_response(users)

async def add_user(request):
    user = await request.json()
    users.append(user)
    return web.json_response(user)

async def update_user(request):
    index = int(request.match_info['index'])
    user = await request.json()
    users[index] = user
    return web.json_response(user)

async def delete_user(request):
    index = int(request.match_info['index'])
    users.pop(index)
    return web.Response(status=204)

app = web.Application()
app.add_routes([
    web.get('/users', get_users),
    web.post('/users', add_user),
    web.put('/users/{index}', update_user),
    web.delete('/users/{index}', delete_user),
])

web.run_app(app)
```

Домашняя работа

Асинхронный POST-запрос

```
import aiohttp
import asyncio
```

```
async def post_data(url, data):
    async with aiohttp.ClientSession() as session:
        async with session.post(url, json=data) as response:
            return await response.text()

async def main():
    url = 'https://httpbin.org/post'
    data = {'name': 'Alice', 'age': 30}
    result = await post_data(url, data)
    print(result)

asyncio.run(main())
```

Асинхронное скачивание файлов

```
import aiohttp
import asyncio

async def download_file(url, filename):
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as response:
            with open(filename, 'wb') as f:
                while True:
                    chunk = await response.content.read(1024)
                    if not chunk:
                        break
                    f.write(chunk)
    print(f"Downloaded {filename}")

async def main(url, filename):
    await download_file(url, filename)

url = 'https://httpbin.org/image/png'
filename = 'downloaded_image.png'

asyncio.run(main(url, filename))
```

Обработка параметров запроса на сервере

```
from aiohttp import web

async def handle(request):
    params = request.query
    return web.json_response({'received_params': param
```

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

web.run_app(app)
```

Дополнительные задачи

Асинхронное скачивание нескольких файлов

```
import aiohttp
import asyncio

async def download_file(session, url, filename):
    async with session.get(url) as response:
        with open(filename, 'wb') as f:
            while True:
                chunk = await response.content.read(1024)
                if not chunk:
                    break
                f.write(chunk)
    print(f"Downloaded {filename}")

async def download_files(urls, filenames):
    async with aiohttp.ClientSession() as session:
        tasks = [download_file(session, url, filename) for url,
        await asyncio.gather(*tasks)

async def main(urls, filenames):
    await download_files(urls, filenames)

urls = ['https://httpbin.org/image/jpeg', 'https://httpbin.org/i
filenames = ['file1.jpg', 'file2.png']
asyncio.run(main(urls, filenames))
```

Асинхронный парсинг HTML-страниц

```
import aiohttp
import asyncio
from bs4 import BeautifulSoup

async def fetch_page(url):
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as response:
            return await response.text()
```

```python
async def extract_headings(url):
    html = await fetch_page(url)
    soup = BeautifulSoup(html, 'html.parser')
    headings = [h.text for h in soup.find_all('h1')]
    print(headings)

async def main():
    url = 'https://example.com'
    await extract_headings(url)

asyncio.run(main())
```