Университет ИТМО

Лабораторная работа №3,4,5,6 по дисциплине «Технологии программирования»

Выполнили:
Нагавкина София
Богданова Ксения
Кубинская Екатерина
P3410

Санкт-Петербург
2019

лаб-3

```
def extr_name(filename):
    rate = []
    mRate = {}
    fRate = {}
    lines = readLines(filename)
    year = findYear(lines)
    findNames(lines,mRate,fRate)

    fullRate = dict(mRate)
    fullRate.update(fRate)

    rate.append(year);
    for i in sorted (fullRate):
            rate.append('{} {}'.format(i,fullRate[i]))
    return rate
def findYear(lines):
    yearPattern = 'Popularity in (?P<year>\d{4})</h2></caption>'
    for line in lines:
            m = re.search(yearPattern,line)
            if m != None:
                    return m.group('year')
def findNames(lines, mRate, fRate):
    ratePattern = '<tr
align="right"><td>(?P<index>\d+)</td><td>(?P<maleN>[A-Z][a-z]+)</td><td>(?P
<femaleN>[A-Z][a-z]+)</td>$'
    for line in lines:
            m = re.search(ratePattern,line)
            if m!=None:
                    maleN = m.group('maleN')
                    femaleN = m.group('femaleN')
                    index = m.group('index')
                    mRate[maleN] = index
                    fRate[femaleN] = index
def sort_last(i):
    return int(i[1])
def printTopRate(rate, string):
    count = 0
    print ('{} top10:'.format(string))
    for k,v in sorted(rate.items(),key = sort_last):
            count+=1
            print ('{} {}'.format(k,v))
            if count==10:
                    break
    print('\n')
```

```
def readLines(filename):
    f = open(filename,"r")
    return f.readlines()
def top(filenames):
    count=0
    bufMRate = {}
    bufFRate = {}
    mRate = {}
    fRate = {}
    for filename in filenames:
            lines = readLines(filename)
            findNames(lines,bufMRate,bufFRate)

            for k,v in mRate.items():
                    tempValue = bufMRate.pop(k,0)
                    if tempValue != 0:
                            if int(tempValue) < int(v):
                                    mRate[k] =
int(tempValue)
            mRate.update(bufMRate)

            for k,v in fRate.items():
                    tempValue = bufFRate.pop(k,0)
                    if tempValue != 0:
                            if int(tempValue) < int(v):
                                    fRate[k] = int(tempValue)
            fRate.update(bufFRate)
            bufFRate = {}
            bufMRate = {}
    print("\n")
    printTopRate(fRate,"Female")
    printTopRate(mRate,"Male")
    return 0
def main():
    args = sys.argv[1:]
    if not args:
            print ('use: [--file] file [file ...]')
            sys.exit(1)
    index = 0
    while index < len(args):

            print('{}:\n {}'.format(args[index], extr_name(args[index])))
            index += 1
    top(args)
```

лаб-4

```
import random
import re

def getNewText(text):
    r={}
    l=len(text)
    temp=[[' '] * l for i in range(l)]
    i=0
    for wrd in text:
        j=i
        num=text.index(wrd)
        if i==num:
            r[i]=wrd
        i=num+1
        i=j+1
    i=-1
    for wrd in text:
        i=i+1
        for key in r:
            if r[key]==wrd:
                if i<l-1:
                    temp[key][i]=text[i+1]
    wrd=text[0]
    empty=' '
    out=wrd+empty
    for i in range(l-1):
        for key in r:
            if r[key]==wrd:
                while(True):
                    randindex=random.randint(0,l-1)
                    if temp[key][randindex]==' ':
                        temp[key][randindex]==' '
                    else:
                        out=out+temp[key][randindex]+empty
                        wrd=temp[key][randindex]
                        break
    return out

if __name__ == '__main__':
    filename=input()
    file = open(filename, 'r')
    content = file.read()
    file.close()
    content = content.replace('\n', ' ')
    content = re.sub(r'[ ]{2,}', ' ', content)
    words = content.split(' ')
    d=getNewText(words)
    print(d)
```

лаб-5 игра guess by image

```
const test_values = {
    photoURL:
'https://giantbomb1.cbsistatic.com/uploads/original/0/6087/2437349-pikachu.png'
,
    variants: ['pikachu', 'raichu', 'psyduck'],
    answer: 'pikachu',
};

const [EASY, MEDIUM, HARD] = ['easy', 'medium', 'hard'];
```

```
import sys
import json
import random
from urllib.request import Request, urlopen
from lxml import html

from http.server import BaseHTTPRequestHandler, HTTPServer
from urllib.parse import parse_qs
from urllib.parse import urlparse
```

```
const levels = [EASY, MEDIUM, HARD];

const [CORRECT, WRONG, EMPTY] = ['Correct!', 'Wrong!', null];
function App() {
    const [question, setQuestion] = useState(test_values);
    const [selectedOption, setSelectedOption] = useState(null);
    const [result, setResult] = useState(EMPTY);
    const [level, setLevel] = useState(EASY);
    useEffect(() => { getNewQuestion() }, [level]);
    const getNewQuestion = async () => {
        let gameInfo = null;
        try {
            gameInfo = await getJson(`/api?level=${level}`);
            const {photo, answer, variants} = gameInfo;
            setQuestion({
                photoURL: photo,
                variants,
                answer
            });
        } catch (e) {
            console.log(e);
            gameInfo = { error: e };
        }
    };
    const selectOption = (option) => {
        setSelectedOption(option);
    };
    const checkAnswer = () => {
        console.log(selectedOption);
        const newResult = selectedOption === question.answer
            ? CORRECT
            : WRONG;
        setResult(newResult);
    };
    const getNextQuestion = () => {
        setResult(EMPTY);
        setSelectedOption(null);
        getNewQuestion();
    };

     return (
        <div className="App">
            <RadioButton options={levels} selected={level} onSelect={setLevel}/>
            <img src={question.photoURL} className="App-logo" alt="logo" />
            <div className={'selectWrapper'}>
                <span className={'label'}>This is </span>
                <DropDown onSelect={selectOption}
                          selected={selectedOption}
                          variants={question.variants || []}/>
                {selectedOption && <button className={'brk-btn submitButton'}
onClick={checkAnswer}>check</button>}
            </div>
            {result && <div>{result}</div>}
            {result && <button className={'brk-btn nextButton'}
onClick={getNextQuestion}>next</button>}
        </div>
    );
}

const DropDown = (props) => {
    const {onSelect, selected, variants} = props;
    const [isOpen, setIsOpen] = useState(false);
    const handleSelect = (el) => {
        setIsOpen(false);
        onSelect(el);
    };
    const sortedVariants = variants.sort((a, b) => a === selected ? -1 : 1);
    return <span className={'selectWrapper'}>
                <button className={'placeholder'} onClick={() =>
setIsOpen(!isOpen)}>{selected || '      '}</button>
                <span className={`list ${isOpen ? 'visible' : ''}`}>
                    {sortedVariants.map((el, i) =>
                        <button key={i} className="variants" onClick={() =>
handleSelect(el)}>{el}</button>
                    )}
                </span>
            </span>;
};

const getJson = async (input) => {
    const response = await fetch(input, {method: 'GET'});
    return response.json();
};
export default App;
```

```
from serverparams import LEVELS_TO_NUM_VARIANTS, COUNT_ANSWERS

def get_names_list_from_file(filename):
    file = open(filename, "r")
    lines_in_file = file.readlines()

    names = []
    for name in lines_in_file:
        names.append(name)

    return names

def generate_answers(names_list, count_variants):
    generated_answers = []
    index = 0
    while index < count_variants:
        answer = names_list[random.randint(0, len(names_list) - 1)]
        formatted_answer = answer.split('\n')[0]
        if formatted_answer not in generated_answers:
            generated_answers.append(formatted_answer)
            index += 1

    correct_answer = generated_answers[
        random.randint(0, len(generated_answers) - 1)
    ]

    return generated_answers, correct_answer


def get_photo_for_correct_answer(name):
    url =
"https://www.google.ru/search?site=&tbm=isch&source=hp&biw=1600&bih=160
0&q=" + (name + ' png').replace(" ", "%20")

    #url = "https://pokemondb.net/pokedex/national"
    req = Request(
        url,
        headers={'User-Agent': 'Mozilla/5.0'})
    html_content = urlopen(req).read()
    tree = html.fromstring(html_content)

    import random
    return tree.xpath('(.//*[@target="_blank"]/img)[' + str(random.randint(1, 10)) +
']/@src')

class QuizServer(BaseHTTPRequestHandler):
    # GET
    def do_GET(self):
        names = get_names_list_from_file(filename=sys.argv[1])

        params = parse_qs(urlparse(self.path).query)
        if 'level' not in params:
            # do not any action
            return

        level = params['level'][0]
        if level in LEVELS_TO_NUM_VARIANTS:
            num_variants = LEVELS_TO_NUM_VARIANTS[level]
        else:
            num_variants = LEVELS_TO_NUM_VARIANTS[1]

        answer_variants, correct_answer = generate_answers(
            names_list=names,
            count_variants=num_variants
        )

        photo_for_answer =
get_photo_for_correct_answer(name=correct_answer)[0]
        response = {
            'photo': photo_for_answer,
            'answer': correct_answer,
            'variants': answer_variants
        }
        self.send_response(200)
        self.send_header('Content-type', 'application/json')
        self.end_headers()
        self.wfile.write(bytes(json.dumps(response), encoding='utf-8'))

if __name__ == '__main__':
    server_address = ('127.0.0.1', 8085)
    http_server = HTTPServer(server_address, QuizServer)
    http_server.serve_forever()
```

лаб-6 rss-клиент

```
const ITEMS_PER_PAGE = 5;

function App() {
    const [channels, setChannels] = useState([]);
    const [selectedChanel, setSelectedChanel] = useState({});
    const [currentPage, setCurrentPage] = useState(0);
    const [articles, setArticles] = useState(null);
    const [showCreatePopup, setShowCreatePopup] = useState(false);
    useEffect(() => {getChannels();}, []);
    useEffect(() => {fetchArticles();}, [currentPage, selectedChanel]);
    const getChannels = async () => {
        try {
            const result = await getJson('/api?event=getlist');
            setChannels(result);
        } catch (e) {console.error(e);}
    };
    const selectChanel = async (chanel) => {
        setArticles([]);       setCurrentPage(0);
        setSelectedChanel(chanel);
    };
    const fetchArticles = async () => {
        try {
            const result = await getJson(`/api?event=getarticles` +
`&name=${encodeURIComponent('' + selectedChanel)}` +
`&offset=${encodeURIComponent(currentPage * ITEMS_PER_PAGE)}` +
`&limit=${encodeURIComponent(ITEMS_PER_PAGE)}`);
            if(articles.error) {
                console.error(articles.error);
            } else {setArticles(result);}
            setShowCreatePopup(false);
        } catch (e) {console.error(e);}
    };
    const addChanel = async () => {
        const name = document.getElementById("nameInput").value;
        const url = document.getElementById("urlInput").value;
        // todo validate
        if(name && url) {
            try {
                const result = await getJson(`/api?event=add` +
                    `&name=${encodeURIComponent(name)}` +
                    `&url=${encodeURIComponent(url)}`
                );
                setChannels([...Object.values(result)]);
                setShowCreatePopup(false);
            } catch (e) {
                console.error(e);
            }
        }
    };
     return (
        <div className="App">
            <div className="leftBar">
                {channels && channels.map((el, i) => {
                    return (
                        <button key={i} className="chanelItem" onClick={() =>
selectChanel(el)}>{el}</button>
                    );
                })}
                <button className={'brk-btn'} onClick={() =>
setShowCreatePopup(true)}> + </button>
            </div>
            {showCreatePopup && <div className="overlayPopup">
                <div>Name</div>
                <input id="nameInput"/>
                <div>Link</div>
                <input id="urlInput"/>
                <button className={'brk-btn addChanelButton'}
onClick={addChanel}>add new channel</button>
            </div>}
            <div className="articlesListWrapper">
                <button disabled={currentPage < 1} className={'brk-btn
paginationButton'} onClick={() => setCurrentPage(currentPage - 1)}>
                    prev
                </button>
                <div className="articlesList">
                    {articles && articles.map((el, i) => {
                        return(
                            <div key={i} className="articleWrapper">
                                <div className="articleName">{el.name}</div>
                                <a className="articleURL" href={el.url}>link</a>
                            </div>
                        );
                    })}
                </div>
                <button className={'brk-btn paginationButton'} onClick={() =>
setCurrentPage(currentPage + 1)}>
                    next
                </button>
            </div>
        </div>
    );
```

```
from http.server import BaseHTTPRequestHandler, HTTPServer
from urllib.parse import parse_qs
from urllib.parse import urlparse
import json

import feedparser

my_list = {}


class Reader:
    def __init__(self, name, url, lang='ru'):
        self.lang = lang
        self.rss_path = url
        self.title = name

    def get_feed(self):
        d = feedparser.parse(self.rss_path)
        return d['entries']

    def read(self, offset=0, limit=5):
        f = self.get_feed()
        print(f[0].keys())
        result = []
        for i, item in enumerate(f):
            if int(offset) <= i < int(limit) + int(offset):
                real_number = (i + 1)
                result.append({'name': item['title'], 'url': item['link']})

        return result

class HTTPServerQuiz(BaseHTTPRequestHandler):
    # GET
    def do_GET(self):
        print('do get')
        params = parse_qs(urlparse(self.path).query)
        if 'event' not in params:
            return
        event = params['event'][0]
        if event == 'add':
            self.add_rss(params)
            return
        elif event == 'getlist':
            self.get_list()
            return
        elif event == 'getarticles':
            self.get_articles(params)
            return

        return

    def get_articles(self, params):
        global my_list
        if 'name' not in params:
            self.send_message({'error': 'already exist'})
            return
        name = params['name'][0]
        if name not in my_list:
            self.send_message({'error': 'no such list'})
            return

        offset = params['offset'][0]
        limit = params['limit'][0]
        self.send_message(my_list[name].read(offset, limit))
        return

    def get_list(self):
        print('get list')
        self.send_message(list(my_list.keys()))
        print('get list')
        return

    def add_rss(self, params):
        name = params['name'][0]
        url = params['url'][0]
        if name in my_list:
            self.send_message({'error': 'already exist'})
            return

        reader = Reader(name, url)
        my_list[name] = reader
        self.send_message(list(my_list.keys()))
        return

    def send_message(self, response):
        print('resp')
        self.send_response(200)
        self.send_header('Content-type', 'application/json')
        self.end_headers()
        self.wfile.write(bytes(json.dumps(response), encoding='utf-8'))
        return
```

```
}
const getJson = async (query) => {
    const response = await fetch(query, {
        method: 'GET'
    });
    return response.json();
};
export default App;
```

```
if __name__ == '__main__':
    server_address = ('127.0.0.1', 8086)
    httpd = HTTPServer(server_address, HTTPServerQuiz)

    httpd.serve_forever()
```