Интенсивы Академии Яндекса. Осень 2022. Разбор задач

Web-программирование на Django

Мы предлагаем лишь вид правильного решения задачи. Вы конечно же могли сделать лучше, эффективнее и красивее.

Вариант №1

Задача №1. В Калькутту

```
import sys


def solve(*data: str) -> str:
    result = ''
    best_len = 0
    best_row = ''
    for row in data:
        if 'S' not in row:
            continue
        b, n = 0, 0
        for x in row:
            if x == 'S':
                n += 1
            else:
                n = 0
            b = max(n, b)
        if b < best_len:
            continue
        if b == best_len and not (len(best_row) < len(row) or (len(best_row) == len(row
) and best_row < row)):
            continue
        best_len = b
        best_row = row
        result = max(row.split('S'), key=len)
    if not result:
        raise SystemError
    return result


def main():
    print(solve(*map(str.strip, sys.stdin)))


if __name__ == '__main__':
    main()
```

Задача №2. Безмолвие

```
class MagicalSilence:
    def __init__(self, country, threshold, magic):
        self.country = country
        self.threshold = threshold
        self.magic = magic

    def __mul__(self, other: 'MagicalSilence'):
        return MagicalSilence(
            ''.join((self.country, other.country)),
            min(self.threshold, other.threshold),
            (self.magic + other.magic) // 2
        )

    def __iadd__(self, other):
        self.magic += other
        self.threshold = max(1, self.threshold - other // 3)
        return self

    def __truediv__(self, other):
        return [
            MagicalSilence(
                self.country,
                self.threshold * other,
                1 if i < self.magic else 0
            )
            for i in range(other)
        ]

    def __call__(self, other):
        return (self.threshold + len(self.country)) * self.magic // other

    def add_magic(self, other):
        self.magic = max(1, self.magic + other)

    def __str__(self):
        return f'Silence in {self.country}, low {self.threshold}, magic {self.magic}.'

    def __repr__(self):
        reprs = ', '.join(map(repr, (self.country, self.threshold, self.magic)))
        return f'MagicalSilence({reprs})'

    def compare(self):
        return self.threshold, self.magic, self.country

    def __eq__(self, other):
        return self.compare() == other.compare()

    def __lt__(self, other):
```

```
        return self.compare() < other.compare()

    def __ne__(self, other):
        return not self.__eq__(other)

    def __le__(self, other):
        return self.__lt__(other) or self.__eq__(other)

    def __gt__(self, other):
        return not self.__le__(other)

    def __ge__(self, other):
        return not self.__lt__(other)
```

Задача №3. Тундра

```
import sys
from argparse import ArgumentParser
from copy import deepcopy
from io import StringIO
from json import dump
from typing import List, Dict, Optional

from requests import get

OUTPUT_FILENAME = 'tundra.json'


def parse(argv: List[str]):
    parser = ArgumentParser()
    parser.add_argument('server', type=str)
    parser.add_argument('port', type=str)
    parser.add_argument('kinds', type=str, nargs='+')
    parser.add_argument('--height', type=int, default=200)
    parser.add_argument('--name', type=str, default='a')
    return parser.parse_args(argv)


def solve(argv: List[str], server_data: Optional[List[Dict]] = None) -> str:
    args = parse(argv)
    if server_data is None:
        server_data = get_data_from_server(args.server, args.port)
    else:
        server_data = deepcopy(server_data)
    result = {}

    for data in server_data:
        if data['kind'] in args.kinds and data['height'] <= args.height and args.name i
n data['name']:
            if data['kind'] not in result:
                result[data['kind']] = []
            result[data['kind']].append([data['name'], data['hardiness']])
            result[data['kind']].sort(key=lambda x: (-x[1], x[0]))

    buffer = StringIO()
    dump(result, buffer, ensure_ascii=False, indent=4)
    return buffer.getvalue()


def get_data_from_server(server, port) -> list:
    return get(f'http://{server}:{port}').json()


def main():
    with open(OUTPUT_FILENAME, 'w', encoding='utf-8') as stream:
```

```
        stream.write(solve(sys.argv[1:]))


if __name__ == '__main__':
    main()
```

Задача №4. Цепочка

```
import csv
import os
import shutil
import sqlite3
from collections import OrderedDict

DB_FILENAME = 'rescue.db'
FUNC_NAME = 'chain'
OUTPUT_CSV = 'restored_chain.csv'
DELIMITER = ':'


def chain(*args):
    result = solve(*args)
    with open(OUTPUT_CSV, 'w', encoding='utf-8') as stream:
        writer = csv.DictWriter(stream, list(result[0].keys()), delimiter=DELIMITER)
        writer.writeheader()
        writer.writerows(result)


def solve(*data, generation=False):
    if generation:
        data = run_queries(*data)

    con = sqlite3.connect(DB_FILENAME)
    cur = con.cursor()

    headers = 'name:length:strength'.split(':')
    result = []
    for _id in data:
        cur.execute(
            f"SELECT l.name, c.length, c.strength "
            f"FROM Chain c "
            f"JOIN Links l ON l.id = c.type_id "
            f"WHERE c.id = ? ",
            (_id,)
        )
        result.append(OrderedDict(zip(headers, cur.fetchone())))

    result.sort(key=lambda x: (x['strength'], x['name']), reverse=True)
    result = [
        OrderedDict(number=number, **row)
        for number, row in enumerate(result, start=1)
    ]

    con.close()
    if generation:
        os.unlink(DB_FILENAME)
```

```
    return result


def run_queries(*data):
    if 'SQL:' in data:
        input_data_length = data.index('SQL:')

        con = sqlite3.connect(DB_FILENAME)
        for query in data[input_data_length + 1:]:
            con.execute(query)
        con.commit()
        con.close()

        return data[:input_data_length]

    shutil.copy('sample.db', DB_FILENAME)
    return data
```

Задача №5. Процессия

```
import csv
import json
import sys

from flask import Flask
from argparse import ArgumentParser

ROUTE = '/holiday'
HOST_ARG = 'host'
PORT_ARG = 'port'
FILE_ARG = 'file'
DELIMITER = ','
HEADERS = 'id,nationality,country,custom'.split(DELIMITER)


def parse_args(argv):
    parser = ArgumentParser()
    parser.add_argument(f'--{HOST_ARG}')
    parser.add_argument(f'--{PORT_ARG}', type=int)
    parser.add_argument(f'--{FILE_ARG}')
    parser.add_argument(f'--country')
    args = parser.parse_args(argv)
    return vars(args)


def data():
    result = {}
    for row in FILE_DATA:
        if row['country'] == ARGS['country']:
            if row['nationality'] not in result:
                result[row['nationality']] = []
            result[row['nationality']].append(row['custom'])
            result[row['nationality']].sort(reverse=True)
    return result


def solve(args, file_data):
    ARGS.update(args)
    FILE_DATA[:] = file_data
    return data()


ARGS = {}
FILE_DATA = []
app = Flask(__name__)


@app.route(ROUTE)
def app_route():
```

```
    return json.dumps(data(), indent=4, ensure_ascii=False)


def main():
    ARGS.update(parse_args(sys.argv[1:]))
    with open(ARGS[FILE_ARG], 'r', encoding='utf-8') as stream:
        FILE_DATA[:] = csv.DictReader(stream, delimiter=DELIMITER)
    app.run(ARGS[HOST_ARG], ARGS[PORT_ARG])


if __name__ == '__main__':
    main()
```

Задача №6. Нет дороги

```
import csv
import json
import sys

INPUT_DELIMITER = '='
INPUT_HEADERS = 'id=city=distance=railway'.split(INPUT_DELIMITER)
OUTPUT_FILENAME = 'roads.json'


def solve(stdin, file_data):
    result = []
    for row in file_data:
        if row['railway'] == '1':
            result.append({
                'city': row['city'],
                'distance': int(row['distance'])
            })
    result.sort(key=lambda x: x['city'], reverse=True)
    return result


def main():
    stdin = list(map(str.strip, sys.stdin))

    with open(stdin[0], 'r', encoding='utf-8') as stream:
        file_data = list(csv.DictReader(stream, delimiter=INPUT_DELIMITER))

    output_data = solve(stdin, file_data)

    with open(OUTPUT_FILENAME, 'w', encoding='utf-8') as stream:
        json.dump(output_data, stream, ensure_ascii=False, indent=4)


if __name__ == '__main__':
    main()
```

Задача №7. Суперклава

```
from collections import Counter

n = int(input())
pr = list(map(int, input().split()))
not_needed = int(input())
queue = Counter(map(int, input().split()))
for i in range(n):
    if queue[i + 1] > pr[i]:
        print("YES")
    else:
        print("NO")
```

Вариант №2

Задача №1. Без пенни в кармане

```
import sys


def solve(*data: str) -> int:
    result = 0
    best_count = 0
    best_row = ''
    for row in data:
        count = sum(x.islower() for x in row)
        if count < best_count:
            continue
        if count == best_count and len(row) < len(best_row):
            continue
        if count == best_count and len(row) == len(best_row) and row < best_row:
            continue
        best_count = count
        best_row = row
        b, n = '', ''
        for x in row:
            if x.islower():
                n += x
            else:
                n = ''
            b = max(b, n, key=len)
        result = len(b)
    if not result:
        raise SystemError
    return result


def main():
    print(solve(*map(str.strip, sys.stdin)))


if __name__ == '__main__':
    main()
```

Задача №2. Драгоценный разум

```
class PreciousMind:
    def __init__(self, creature, volume, level):
        self.creature = creature
        self.volume = volume
        self.level = level

    def __sub__(self, other: 'PreciousMind'):
        result = PreciousMind(
            self.creature,
            abs(self.volume - other.volume),
            min(self.level, other.level)
        )
        max(self, other, key=lambda x: x.volume).volume -
= abs(self.volume - other.volume)
        return result

    def __iadd__(self, other):
        self.volume += other // 4
        self.level = min('W', chr(ord(self.level) + 1))
        return self

    def __mul__(self, other):
        result = [
            PreciousMind(
                f'{self.creature}_{i + 1}',
                self.volume // other,
                self.level
            )
            for i in range(other)
        ]
        self.volume %= other
        return result

    def __call__(self, other):
        return (self.volume * (ord(self.level) - ord('A')) + len(self.creature)) // oth
er

    def upgrade(self, other):
        self.level = min('W', chr(ord(self.level) + other))
        return 'DONE!'

    def __str__(self):
        return f'{self.creature} with brain volume {self.volume} and intelligence {self
.level}.'

    def __repr__(self):
        reprs = ', '.join(map(repr, (self.creature, self.volume, self.level)))
        return f'PreciousMind({reprs})'
```

```
def compare(self):
    return self.level, self.volume, self.creature

def __eq__(self, other):
    return self.compare() == other.compare()

def __lt__(self, other):
    return self.compare() < other.compare()

def __ne__(self, other):
    return not self.__eq__(other)

def __le__(self, other):
    return self.__lt__(other) or self.__eq__(other)

def __gt__(self, other):
    return not self.__le__(other)

def __ge__(self, other):
    return not self.__lt__(other)
```

Задача №3. Амазонка

```
import sys
from argparse import ArgumentParser
from copy import deepcopy
from io import StringIO
from json import dump
from typing import List, Dict, Optional

from requests import get

OUTPUT_FILENAME = 'flow_with_clay.json'


def parse(argv: List[str]):
    parser = ArgumentParser()
    parser.add_argument('server', type=str)
    parser.add_argument('port', type=str)
    parser.add_argument('states', type=str, nargs='+')
    parser.add_argument('--impurity', type=int, default=20)
    parser.add_argument('--exclude', type=str, default='zw')
    return parser.parse_args(argv)


def solve(argv: List[str], server_data: Optional[List[Dict]] = None) -> str:
    args = parse(argv)
    if server_data is None:
        server_data = get_data_from_server(args.server, args.port)
    else:
        server_data = deepcopy(server_data)
    result = {}

    for data in server_data:
        if set(data['states']) & set(args.states) and data['pollution'] <= args.impurit
y \
                and not (set(data['river'].lower()) & set(args.exclude.lower())):
            if data['pollution'] not in result:
                result[data['pollution']] = []
            result[data['pollution']].append(data['river'])
            result[data['pollution']].sort(reverse=True)

    buffer = StringIO()
    dump(result, buffer, ensure_ascii=False, indent=4)
    return buffer.getvalue()


def get_data_from_server(server, port) -> list:
    return get(f'http://{server}:{port}').json()


def main():
```

```
    with open(OUTPUT_FILENAME, 'w', encoding='utf-8') as stream:
        stream.write(solve(sys.argv[1:]))


if __name__ == '__main__':
    main()
```

Задача №4. Помощь

```
import csv
import os
import shutil
import sqlite3
from collections import OrderedDict

DB_FILENAME = 'ships_about.db'
FUNC_NAME = 'life_saving'
OUTPUT_CSV = 'crawl_order.csv'
DELIMITER = ';'


def life_saving(*args):
    result = solve(*args)
    with open(OUTPUT_CSV, 'w', encoding='utf-8') as stream:
        writer = csv.DictWriter(stream, list(result[0].keys()), delimiter=DELIMITER)
        writer.writeheader()
        writer.writerows(result)


def solve(*data, generation=False):
    if generation:
        data = run_queries(*data)

    con = sqlite3.connect(DB_FILENAME)
    cur = con.cursor()

    cur.execute(
        f"SELECT t.leftboard, t.bow, t.starboard, t.stern "
        f"FROM Types_of_ship t "
        f"JOIN Ships s ON s.type_id = t.id "
        f"WHERE s.ship_name = ? ",
        (data[0],)
    )

    headers = list(map(lambda x: x[0], cur.description))
    _data = cur.fetchone()
    row_number = 1
    result = []
    for name, val in zip(headers, _data):
        while val > 0:
            val -= data[1]
            result.append(OrderedDict(
                number=row_number,
                side=name,
                rest=max(val, 0)
            ))
            row_number += 1
```

```
    con.close()
    if generation:
        os.unlink(DB_FILENAME)

    return result


def run_queries(*data):
    if 'SQL:' in data:
        input_data_length = data.index('SQL:')

        con = sqlite3.connect(DB_FILENAME)
        for query in data[input_data_length + 1:]:
            con.execute(query)
        con.commit()
        con.close()

        return data[:input_data_length]

    shutil.copy('sample.db', DB_FILENAME)
    return data
```

Задача №5. Отвага

```
import csv
import json
import sys

from flask import Flask
from argparse import ArgumentParser

ROUTE = '/bravery'
HOST_ARG = 'host'
PORT_ARG = 'port'
FILE_ARG = 'filename'
DELIMITER = ';'
HEADERS = 'id;train_series;need;action;order'.split(DELIMITER)


def parse_args(argv):
    parser = ArgumentParser()
    parser.add_argument(f'--{HOST_ARG}')
    parser.add_argument(f'--{PORT_ARG}', type=int)
    parser.add_argument(f'--{FILE_ARG}')
    parser.add_argument(f'--series')
    args = parser.parse_args(argv)
    return vars(args)


def data():
    result = {}
    for row in FILE_DATA:
        if row['train_series'] == ARGS['series']:
            if row['need'] not in result:
                result[row['need']] = []
            result[row['need']].append((int(row['order']), row['action']))
            result[row['need']].sort()
    return [
        {
            'need': key,
            'actions': [x[1] for x in value]
        }
        for key, value in sorted(result.items())
    ]


def solve(args, file_data):
    ARGS.update(args)
    FILE_DATA[:] = file_data
    return data()


ARGS = {}
```

```
FILE_DATA = []
app = Flask(__name__)


@app.route(ROUTE)
def app_route():
    return json.dumps(data(), indent=4, ensure_ascii=False)


def main():
    ARGS.update(parse_args(sys.argv[1:]))
    with open(ARGS[FILE_ARG], 'r', encoding='utf-8') as stream:
        FILE_DATA[:] = csv.DictReader(stream, delimiter=DELIMITER)
    app.run(ARGS[HOST_ARG], ARGS[PORT_ARG])


if __name__ == '__main__':
    main()
```

Задача №6. Нет дороги

```
import csv
import json
import sys

INPUT_DELIMITER = '='
INPUT_HEADERS = 'id=city=distance=railway'.split(INPUT_DELIMITER)
OUTPUT_FILENAME = 'roads.json'


def solve(stdin, file_data):
    result = []
    for row in file_data:
        if row['railway'] == '1':
            result.append({
                'city': row['city'],
                'distance': int(row['distance'])
            })
    result.sort(key=lambda x: x['city'], reverse=True)
    return result


def main():
    stdin = list(map(str.strip, sys.stdin))

    with open(stdin[0], 'r', encoding='utf-8') as stream:
        file_data = list(csv.DictReader(stream, delimiter=INPUT_DELIMITER))

    output_data = solve(stdin, file_data)

    with open(OUTPUT_FILENAME, 'w', encoding='utf-8') as stream:
        json.dump(output_data, stream, ensure_ascii=False, indent=4)


if __name__ == '__main__':
    main()
```

Задача №7. Дедлайн близко

```
from collections import deque

first, second = deque(), deque()

for _ in range(int(input())):
    data = input().split()
    if data[0] == '-':
        print(first.popleft())
    elif data[0] == '+':
        second.append(data[1])
    else:
        second.appendleft(data[1])
    if len(first) < len(second):
        first.append(second.popleft())
```