МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Институт цифрового развития

ОТЧЁТ

по лабораторной работе

Дисциплина: «Объектно – ориентированное программирование»

Выполнил: студент 3 курса
группы ИВТ-б-о-21-1
Криворот Владимир Геннадьевич

Ставрополь 2023

Работа с исключениями в языке Python

Цель работы: приобретение навыков по работе с исключениями при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

1.Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.

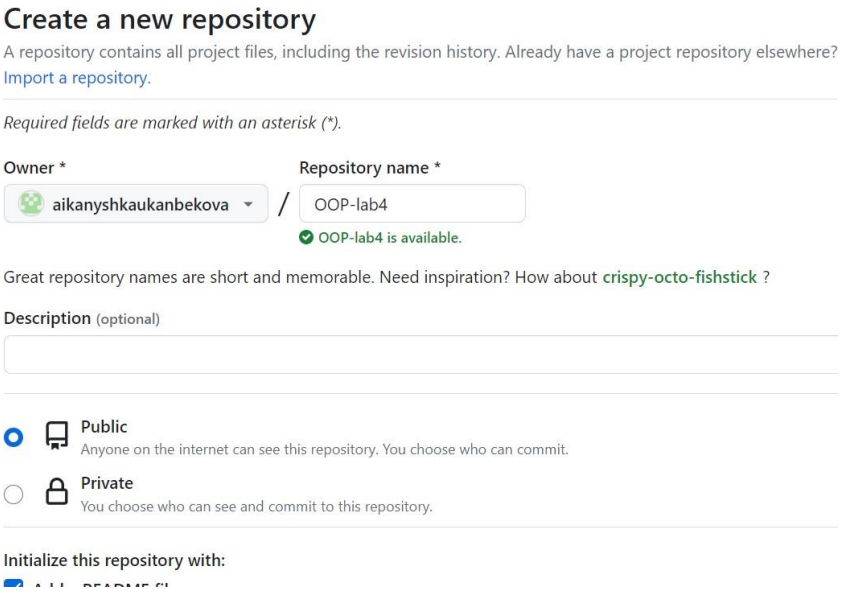

Рисунок 1. Создание репозитория

2.Выполните клонирование созданного репозитория.

```
C:\Users\User>cd C:\Users\User\Documents\3,1 курс\Обьектно-ориентированное\лаб 4

C:\Users\User\Documents\3,1 курс\Обьектно-ориентированное\лаб 4>git clone https://github.com/aikanyshkaukanbekova/OOP-la
b4.git
Cloning into 'OOP-lab4'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.

C:\Users\User\Documents\3,1 курс\Обьектно-ориентированное\лаб 4>
```

Рисунок 2. Клонирование репозитория

3.Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

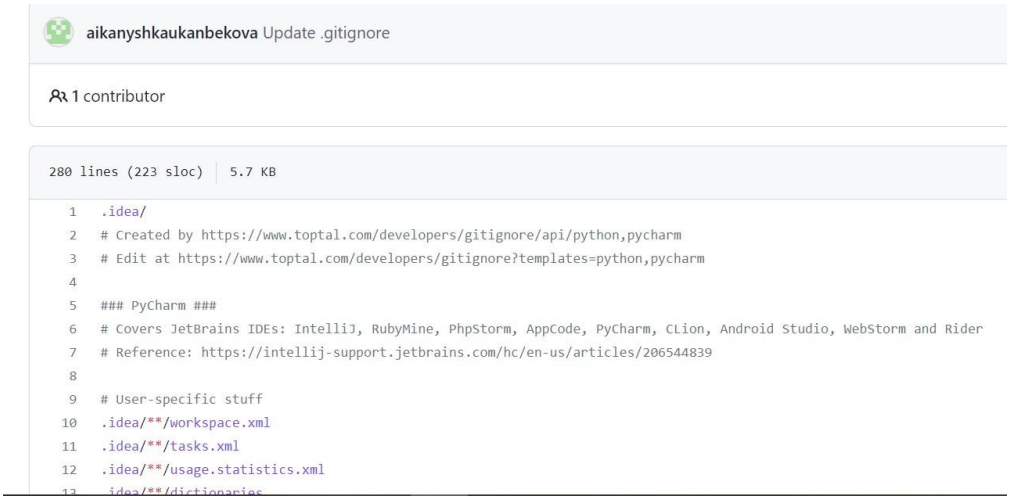

Рисунок 3. Дополнение файла .gitignore

Практическая часть:

Задние 1. Решите следующую задачу: напишите программу, которая запрашивает ввод двух значений. Если хотя бы одно из них не является числом, то должна выполняться конкатенация, т. е. соединение, строк. В остальных случаях введенные числа суммируются.

Код программы:

```
#!/usr/bin/env python3
# _*_ coding: utf-8 _*_

if __name__ == '__main__':
    a = input("Введите первое значение: ")
    b = input("Введите второе значение: ")

    if a.isnumeric() and b.isnumeric():
        a = int(a)
        b = int(b)

    print(a + b)
```

```
Введите первое значение: 20
Введите второе значение: 30
50
```

```
Введите первое значение: Привет
Введите второе значение: 10
Привет10
```

Рисунок 1. Результат задания 1

Задание 2. Решите следующую задачу: напишите программу, которая будет генерировать матрицу из случайных целых чисел. Пользователь может

указать число строк и столбцов, а также диапазон целых чисел. Произведите обработку ошибок ввода пользователя.

Код программы:

```
#!/usr/bin/env python3
# _*_ coding: utf-8 _*_

import random

if __name__ == '__main__':
    try:
        n = int(input("Введите высоту матрицы: "))
        m = int(input("Введите ширину матрицы: "))
        a = int(input("Введите левый диапазон чисел: "))
        b = int(input("Введите правый диапазон чисел: "))
    except Exception as e:
        print("Ошибка: Необходимо ввести числа")
        exit(1)

    matrix = []

    for i in range(0, n):
        r = []
        for j in range(0, m):
            r.append(random.randint(a, b))
        matrix.append(r)

    print(matrix)
```

```
Введите высоту матрицы: 3
Введите ширину матрицы: 3
Введите левый диапазон чисел: 10
Введите правый диапазон чисел: 100
[[29, 92, 10], [20, 86, 94], [72, 34, 100]]
```

Рисунок 2. Результат задания 2

Индивидуальное задание

Вариант 19

Задание 1. Выполнить индивидуальное задание 1 лабораторной работы 2.19, добавив возможность работы с исключениями и логгирование. Изучить возможности модуля logging. Добавить для предыдущего задания вывод в файлы лога даты и времени выполнения пользовательской команды с точностью до миллисекунды.

Код программы:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import argparse
import json
import os.path
import pathlib
import sys
import logging
from datetime import datetime


def add_route(routes, start, finish, number):
    """
    Добавить данные о маршруте
    """

    routes.append(
        {
            'start': start,
            'finish': finish,
            'number': number
        }
    )
    return routes


def display_route(routes):
    """
    Отобразить списко маршрутов
    """
    if routes:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Начальный пункт",
                "Конечный пункт",
                "Номер маршрута"
            )
        )
        print(line)

        # Вывести данные о всех рейсах.
        for idx, worker in enumerate(routes, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('start', ''),
                    worker.get('finish', ''),
                    worker.get('number', 0)
                )
            )
        print(line)
    else:
        print("Список маршрутов пуст.")
```

```
def select_route(routes, period):
    """
    Выбрать маршрут
    """
    result = []
    for employee in routes:
        if employee.get('number') == period:
            result.append(employee)

    return result


def save_routes(file_name, routes):
    """
    Сохранить всех работников в файл JSON.
    """
    try:
        with open(file_name, "w", encoding="utf-8") as fout:
            json.dump(routes, fout, ensure_ascii=False, indent=4)
            directory = pathlib.Path.cwd().joinpath(file_name)
            directory.replace(pathlib.Path.home().joinpath(file_name))
    except Exception as e:
        logging.error(str(e))


def load_routes(file_name):
    """
    Загрузить всех работников из файла JSON.
    """
    try:
        with open(file_name, "r", encoding="utf-8") as fin:
            return json.load(fin)
    except Exception as e:
        logging.error(str(e))


def main(command_line=None):
    logging.basicConfig(
        filename='./commands.log',
        level=logging.INFO
    )
    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "-d",
        "--data",
        action="store",
        required=False,
        help="The data file name"
    )
    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("routes")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )
    subparsers = parser.add_subparsers(dest="command")
    # Создать субпарсер для добавления маршрута.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new route"
```

```
    )
    add.add_argument(
        "-s",
        "--start",
        action="store",
        required=True,
        help="The start of the route"
    )
    add.add_argument(
        "-f",
        "--finish",
        action="store",
        help="The finish of the route"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        type=int,
        required=True,
        help="The number of the route"
    )
    # Создать субпарсер для отображения всех маршрутов.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all routes"
    )
    # Создать субпарсер для выбора маршрута.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the route"
    )
    select.add_argument(
        "-N",
        "--numb",
        action="store",
        type=int,
        required=True,
        help="The route"
    )
    # Выполнить разбор аргументов командной строки.
    args = parser.parse_args(command_line)
    logging.info(f"[{datetime.now()}]: {str(args)}")
    # Загрузить все маршруты из файла, если файл существует.
    data_file = args.data
    if not data_file:
        data_file = os.environ.get("WORKERS_DATA")
    if not data_file:
        print("The data file name is absent", file=sys.stderr)
        sys.exit(1)
    # Загрузить всех работников из файла, если файл существует.
    is_dirty = False
    if os.path.exists(data_file):
        routes = load_routes(data_file)
    else:
        routes = []
    # Добавить маршрут.
    if args.command == "add":
        routes = add_route(
            routes,
            args.start,
            args.finish,
```

```
            args.number
        )
        is_dirty = True
    # Отобразить все маршруты.
    elif args.command == "display":
        display_route(routes)
    # Выбрать требуемые маршруты.
    elif args.command == "select":
        selected = select_route(routes, args.numb)
        display_route(selected)
    # Сохранить данные в файл, если список маршрутов был изменен.
    if is_dirty:
        save_routes(data_file, routes)


if_name_== '_main__':
    main()
```

Контрольные вопросы:

1. Какие существуют виды ошибок в языке программирования Python?

1. Синтаксические ошибки: возникают из-за неправильного использования ключевых слов, операторов или скобок.

2. Логические ошибки: возникают из-за неправильной логики программы, что приводит к неправильным результатам.

3. Ошибки времени выполнения: возникают во время выполнения программы из-за неправильных входных данных или других непредвиденных ситуаций.

4. Ошибки исключений: возникают при попытке выполнения недопустимой операции, например, деления на ноль или обращения к несуществующему элементу.

5. Ошибки импорта: возникают при попытке импортировать несуществующий модуль или библиотеку.

6. Ошибки типов данных: возникают из-за неправильного использования типов данных, например, попытка выполнить операцию с разными типами данных.

7. Ошибки алгоритма: возникают из-за неправильного выбора или реализации алгоритма, что приводит к неправильным результатам.

2. Как осуществляется обработка исключений в языке программирования Python?

Обработка исключений в Python осуществляется с помощью конструкции try-except. Код, который может вызвать исключение, помещается в блок try, а обработка исключения - в блок except. Если исключение произошло в блоке try, то выполнение программы переходит к соответствующему блоку except, где можно обработать ошибку или вывести сообщение об ошибке.

3. Для чего нужны блоки finally и else при обработке исключений?

Блок finally используется для выполнения кода независимо от того, произошло исключение или нет. Это может быть полезно, например, для освобождения ресурсов, закрытия файлов или соединений с базой данных.

Блок else используется для выполнения кода, если исключение не произошло в блоке try. Таким образом, код в блоке else будет выполнен только в том случае, если не было исключения. Это может быть полезно, например, для выполнения каких-то действий, если операция в блоке try завершилась успешно.

4. Как осуществляется генерация исключений в языке Python?

Исключения в Python генерируются с помощью ключевого слова raise, за которым следует объект исключения. Например:

```
Python
raise ValueError("Invalid value")
```

Этот код генерирует исключение типа ValueError с сообщением "Invalid value". Также исключения могут быть сгенерированы автоматически в случае ошибок выполнения кода, например, при делении на ноль или обращении к несуществующему индексу списка.

5. Как создаются классы пользовательский исключений в языке Python?

В Python пользовательские исключения создаются путем создания нового класса, который наследуется от встроенного класса Exception или его подклассов. Например:

```
Python
class CustomError(Exception):
    def__init__(self, message):
        self.message = message
        super().__init__(self.message)
```

В этом примере мы создаем класс CustomError, который является подклассом Exception. Мы также определяем метод__init__, который принимает сообщение об ошибке и передает его в конструктор родительского класса. Теперь мы можем использовать наш пользовательский класс исключений для генерации и обработки исключений в нашем коде:

```
Python
raise CustomError("Something went wrong")
```

Этот код генерирует исключение типа CustomError с сообщением "Something went wrong".

6. Каково назначение модуля logging?

Модуль logging в Python предназначен для записи сообщений, ошибок, предупреждений и другой информации о работе программы в специальные файлы или на консоль. Он позволяет логировать различные события в приложении, что помогает разработчикам отслеживать и анализировать работу программы в процессе ее выполнения. Модуль logging также предоставляет возможность настройки уровня логирования, форматирования сообщений и выбора целей записи (файл, консоль и т. д.), что делает его очень гибким и удобным инструментом для отладки и мониторинга приложений.

7. Какие уровни логгирования поддерживаются модулем logging? Приведите примеры, в которых могут быть использованы сообщения с этим уровнем журналирования.

Модуль logging поддерживает следующие уровни логирования:

1. DEBUG - используется для записи детальной информации о работе программы, например, значения переменных, результаты вычислений и т.д. Этот уровень полезен при отладке приложения.

2. INFO - используется для записи информационных сообщений о ходе выполнения программы, например, о начале или завершении определенной операции.

3. WARNING - используется для записи предупреждений о потенциальных проблемах в программе, которые не являются критическими, но требуют внимания разработчика.

4. ERROR - используется для записи сообщений об ошибках, которые привели к некорректной работе программы.

5. CRITICAL - используется для записи критических ошибок, которые привели к невозможности продолжения работы программы.

Примеры использования:

- DEBUG: Запись значений переменных и результатов вычислений во время отладки приложения.

- INFO: Запись сообщения о начале выполнения важной операции в приложении.

- WARNING: Запись предупреждения о возможной утечке памяти в программе.

- ERROR: Запись сообщения об ошибке при обращении к базе данных.

- CRITICAL: Запись сообщения о критической ошибке, приводящей к аварийному завершению работы приложения.