# МИНИCTEPCTBO 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:

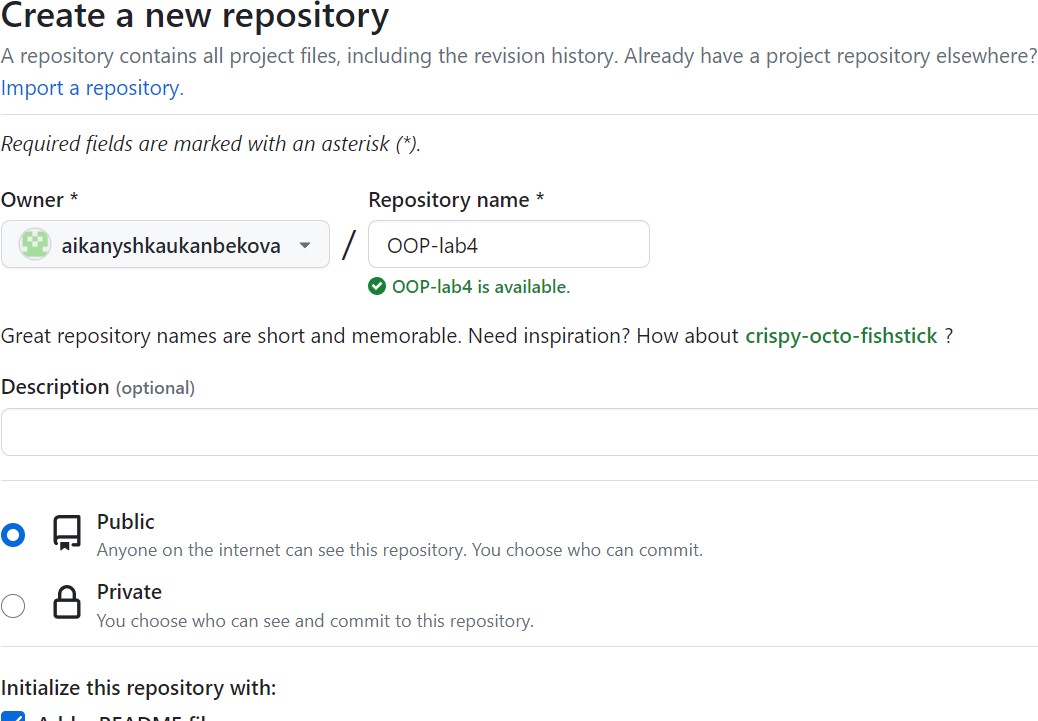
1.Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.

Рисунок 1. Создание репозитория 2.Выполните клонирование созданного репозитория.

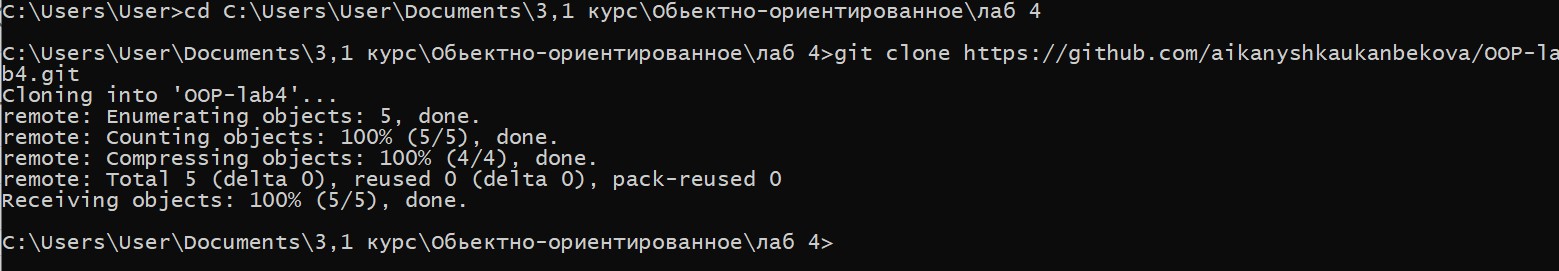


Рисунок 2. Клонирование репозитория

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

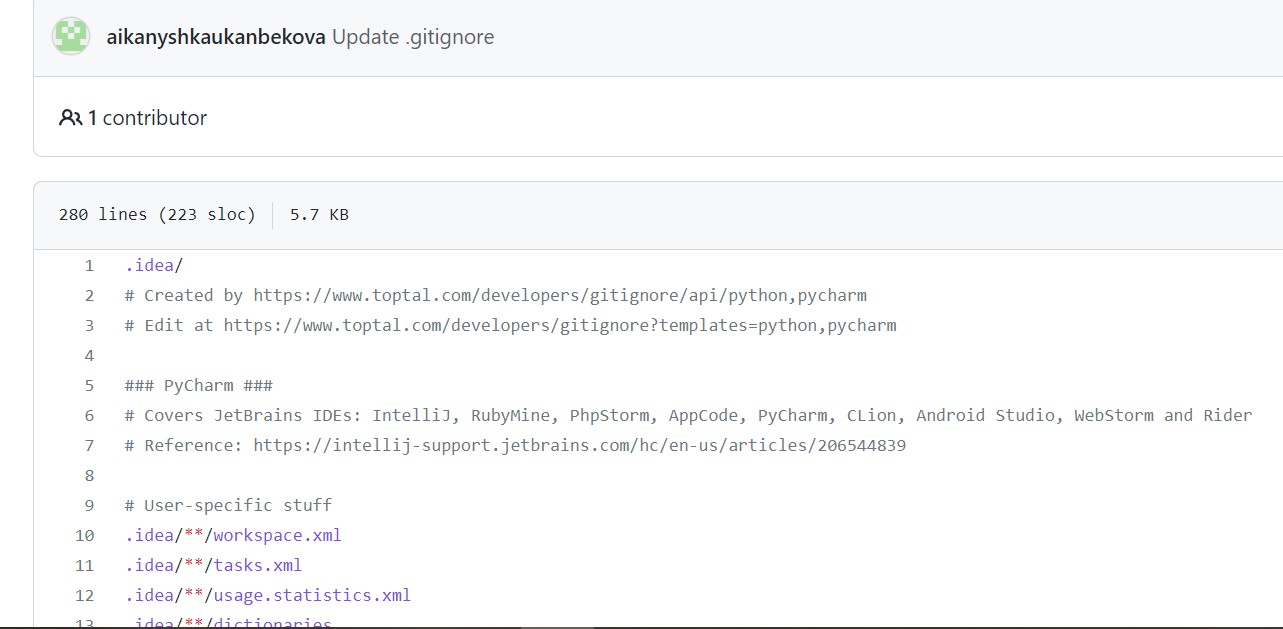


Рисунок 3. Дополнение файла .gitignore

# 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)

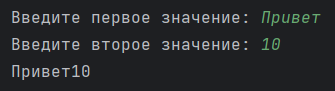
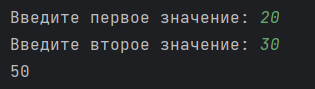


Рисунок 1. Результат задания 1

**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)

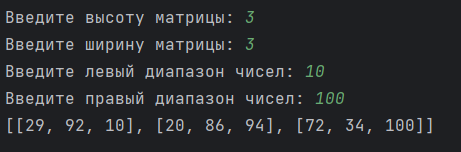


Рисунок 2. Результат задания 2

# 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:

#!/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.

# 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.

# 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.

# 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.

# 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".

# 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.

# 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.