ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № _4_

Дисциплина: Backend-разработка

Тема: Исключения

Выполнил(а): студент(ка) группы __221-379__

_____Кодиров Жамшид Мурод угли_____
(Фамилия И.О.)

Проверил: _______ _____
(Фамилия И.О., степень, звание) **(Оценка)**

Дата, подпись _____ _____
(Дата) (Подпись)

Замечания:_____

_____

_____

Москва

2025

Задание

Цель работы: написать код, который демонстрирует работу исключений в Python.

Требуется написать следующие конструкции:

1. Минимум 2 разные функции, которые принимают на вход один или несколько параметров.
Функции ДОЛЖНЫ выбрасывать исключение при определённых значениях входных параметров.
Функции НЕ ДОЛЖНЫ содержать никаких обработчиков исключений.

2. Функция, которая принимает на вход один или несколько параметров.
Функция ДОЛЖНА выбрасывать исключение при определённых значениях входных параметров.
Функция ДОЛЖНА содержать ОДИН обработчик исключений общего типа (Exception). Внутри блока обработки исключения ДОЛЖНА быть какая-нибудь логика, связанная с обработкой исключения.
Обработчик НЕ ДОЛЖЕН содержать блок finally.

3. Функция, которая принимает на вход один или несколько параметров.
Функция ДОЛЖНА выбрасывать исключение при определённых значениях входных параметров.
Функция ДОЛЖНА содержать ОДИН обработчик исключений общего типа (Exception). Внутри блока обработки исключения ДОЛЖНА быть какая-нибудь логика, связанная с обработкой исключения.
Обработчик ДОЛЖЕН содержать блок finally. Логика внутри блока finally ДОЛЖНА способствовать нормальному завершению работы функции.

4. Минимум 3 разные функции, которые принимают на вход один или несколько параметров.
Функции ДОЛЖНЫ выбрасывать исключения при определённых значениях входных параметров.
Функции ДОЛЖНЫ содержать НЕСКОЛЬКО обработчиков РАЗНЫХ типов исключений (минимум 3 типа исключений). Внутри блоков обработки исключения ДОЛЖНА быть какая-нибудь логика, связанная с обработкой соответствующего типа исключения.
Каждый обработчик МОЖЕТ содержать блок finally. Логика внутри блока finally ДОЛЖНА способствовать нормальному завершению работы функции.

5. Функция, которая принимает на вход один или несколько параметров.
Функция ДОЛЖНА генерировать исключения при определённых условиях (в Python есть конструкция для генерации исключений).
Функция ДОЛЖНА содержать обрабопчики всех исключений, которые

генерируются внутри этой функции. Внутри блоков обработки исключения ДОЛЖНА быть какая-нибудь логика, связанная с обработкой соответствующего типа исключения.
Обработчик МОЖЕТ содержать блок finally. Логика внутри блока finally ДОЛЖНА способствовать нормальному завершению работы функции.

6. Минимум 3 разных пользовательских исключения и примеры их использования

7. Функция, которая принимает на вход один или несколько параметров. Функция ДОЛЖНА выбрасывать пользовательское исключение, созданное на шаге 6. при определённых значениях входных параметров.
Функция ДОЛЖНА содержать МИНИМУМ ОДИН обработчик исключений. Внутри блока обработки исключения ДОЛЖНА быть какая-нибудь логика, связанная с обработкой исключения.
Обработчик МОЖЕТ содержать блок finally.

8. Минимум 3 функции, демонстрирующие работу исключений. Алгоритм функций необходимо придумать самостоятельно

9. Функция, которая последовательно вызывает ВСЕ вышесозданные функции.
Функция ДОЛЖНА завершаться корректно и НЕ ДОЛЖНА иметь необработанных исключений

Дополнительные требования:

Функции, созданные в шагах 1-8 ДОЛЖНЫ быть размещены в одном или нескольких отдельных файлах.

Функция из шага 9 ДОЛЖНА быть размещена в файле main.py.
В файле main.py должна быть конструкция if __name__ == "__main__", внутри которой ДОЛЖНА вызываться функция из шага 9.

В комментариях к каждой функции из шагов 1-8 ДОЛЖНО быть отмечено, к какому шагу относится эта функция.

КРАЙНЕ ЖЕЛАТЕЛЬНО, чтобы реализуемые функции имитировали какую-то реальную логику, были как-нибудь связаны между собой и содержали как можно меньше искусственных примеров.

ЗАПРЕЩАЕТСЯ копировать примеры из лекции.

РАЗРЕШАЕТСЯ использовать примеры из лекции за основу.

Решение

1: Две функции без обработчиков исключений

Описание: Функция divide_numbers делит первое число на второе и выбрасывает исключение, если делитель равен 0.
Код в exceptions.py:

```
def divide_numbers(a, b):
    if b == 0:
        raise ZeroDivisionError("Деление на ноль!")
    return a / b
```

Вызов в main.py:

```
print("Деление:", divide_numbers(10, 2))
```

Результат: "Деление: 5.0"

Описание: Функция check_positive проверяет, положительное ли число, и выбрасывает исключение, если число отрицательное.
Код в exceptions.py:

```
def check_positive(number):
    if number < 0:
        raise ValueError("Число должно быть
положительным!")
    return number
```

Вызов в main.py:

```
print("Проверка положительного:", check_positive(5))
```

Результат: "Проверка положительного: 5"

2: Функция с одним общим обработчиком исключений

Описание: Функция process_string проверяет длину строки и возвращает 0, если строка пустая или возникла ошибка.
Код в exceptions.py:

```
def process_string(text):
    if not isinstance(text, str):
        raise TypeError("Нужна строка!")
    try:
```

```
        length = len(text)
        if length == 0:
            raise ValueError("Строка пустая!")
    except Exception as e:
        print(f"Ошибка: {e}")
        return 0
    return length
```

Вызов в main.py:

```
print("Обработка строки:", process_string("hello"))
```

Результат: "Обработка строки: 5"

3: Функция с обработчиком и блоком finally

Описание: Функция calculate_square вычисляет квадрат числа, возвращает -1 при ошибке и всегда выводит сообщение о завершении.
Код в exceptions.py:

```
def calculate_square(number):
    if not isinstance(number, int):
        raise TypeError("Нужно целое число!")
    try:
        if number < 0:
            raise ValueError("Число отрицательное!")
        result = number * number
    except Exception as e:
        print(f"Поймано исключение: {e}")
        return -1
    finally:
        print("Завершение вычисления квадрата")
    return result
```

Вызов в main.py:

```
print("Квадрат числа:", calculate_square(4))
```

Результат:

```
Завершение вычисления квадрата
Квадрат числа: 16
```

4: Три функции с несколькими обработчиками разных типов исключений

Описание: Функция process_list проверяет список и возвращает его или значение по умолчанию при разных ошибках.
Код в exceptions.py:

```
def process_list(my_list):
    try:
        if not isinstance(my_list, list):
            raise TypeError("Нужен список!")
        if len(my_list) == 0:
            raise ValueError("Список пуст!")
        if "error" in my_list:
            raise KeyError("Найдена ошибка в списке!")
    except TypeError as e:
        print(f"Ошибка типа: {e}")
        return []
    except ValueError as e:
        print(f"Ошибка значения: {e}")
        return [0]
    except KeyError as e:
        print(f"Ошибка ключа: {e}")
        return None
    return my_list
```

Вызов в main.py:

```
print("Обработка списка:", process_list([1, 2, 3]))
```

Результат: "Обработка списка: [1, 2, 3]"

Описание: Функция check_file_name проверяет имя файла и возвращает его или значение по умолчанию при ошибках.
Код в exceptions.py:

```
def check_file_name(filename):
    try:
        if not isinstance(filename, str):
            raise TypeError("Имя файла должно быть
строкой!")
        if "." not in filename:
            raise ValueError("Нет расширения файла!")
```

```
        if len(filename) > 20:
            raise NameError("Слишком длинное имя!")
    except TypeError as e:
        print(f"Тип ошибки: {e}")
        return "default.txt"
    except ValueError as e:
        print(f"Значение ошибки: {e}")
        return "file.txt"
    except NameError as e:
        print(f"Имя ошибки: {e}")
        return "short.txt"
    finally:
        print("Проверка имени завершена")
    return filename
```

Вызов в main.py:

```
print("Проверка имени файла:",
check_file_name("test.txt"))
```

Результат:

```
Проверка имени завершена
Проверка имени файла: test.txt
```

Описание: Функция process_number проверяет число и возвращает его или значение по умолчанию при ошибках.
Код в exceptions.py:

```
def process_number(num):
    try:
        if not isinstance(num, (int, float)):
            raise TypeError("Нужно число!")
        if num < 0:
            raise ValueError("Число отрицательное!")
        if num > 1000:
            raise OverflowError("Число слишком
большое!")
    except TypeError as e:
        print(f"Ошибка типа: {e}")
        return 0
```

```
    except ValueError as e:
        print(f"Ошибка значения: {e}")
        return 1
    except OverflowError as e:
        print(f"Переполнение: {e}")
        return 1000
    return num
```

Вызов в main.py:

```
print("Обработка числа:", process_number(50))
```

Результат: "Обработка числа: 50"

5: Функция с генерацией исключений и их обработкой

Описание: Функция analyze_input анализирует входные данные и возвращает их или значение по умолчанию при ошибках.
Код в exceptions.py:

```
def analyze_input(data):
    try:
        if not data:
            raise ValueError("Данные пустые!")
        if isinstance(data, str) and len(data) > 10:
            raise TooBigError("Строка слишком
длинная!")
        if isinstance(data, int) and data < 0:
            raise TooSmallError("Число слишком
маленькое!")
    except ValueError as e:
        print(f"Ошибка значения: {e}")
        return "default"
    except TooBigError as e:
        print(f"Слишком большое: {e}")
        return "short"
    except TooSmallError as e:
        print(f"Слишком маленькое: {e}")
        return 0
    finally:
        print("Анализ завершен")
    return data
```

Вызов в main.py:

```
print("Анализ ввода:", analyze_input("test"))
```

Результат:

```
Анализ завершен
Анализ ввода: test
```

6: Пользовательские исключения

Описание: Созданы три пользовательских исключения: TooSmallError, TooBigError, WrongTypeError.
Код в exceptions.py:

```
class TooSmallError(Exception):
    pass

class TooBigError(Exception):
    pass

class WrongTypeError(Exception):
    pass
```

Вызов: Используются в других функциях, например, в analyze_input и check_value.
Результат: Эти классы сами по себе ничего не выводят, но используются для генерации исключений.

7: Функция с пользовательским исключением

Описание: Функция check_value проверяет значение и возвращает его или значение по умолчанию при пользовательских исключениях.
Код в exceptions.py:

python
CollapseWrapCopy

```
def check_value(value):
    try:
        if value < 10:
            raise TooSmallError("Значение слишком
маленькое!")
        if value > 100:
```

```
            raise TooBigError("Значение слишком
большое!")
    except TooSmallError as e:
        print(f"Ошибка: {e}")
        return 10
    except TooBigError as e:
        print(f"Ошибка: {e}")
        return 100
    finally:
        print("Проверка значения завершена")
    return value
```

Вызов в main.py:

```
print("Проверка значения:", check_value(50))
```

Результат:

```
Проверка значения завершена
Проверка значения: 50
```

8: Три дополнительные функции

Описание: Функция count_chars считает символы в тексте и возвращает 0 или 50 при ошибках.
Код в exceptions.py:

```
def count_chars(text):
    try:
        if not text:
            raise ValueError("Текст пустой!")
        if len(text) > 50:
            raise OverflowError("Текст слишком
длинный!")
    except ValueError as e:
        print(f"Ошибка: {e}")
        return 0
    except OverflowError as e:
        print(f"Ошибка: {e}")
        return 50
    return len(text)
```

Вызов в main.py:

```
print("Подсчет символов:", count_chars("hello"))
```

Результат: "Подсчет символов: 5"

Описание: Функция multiply_numbers умножает два числа и возвращает 0 или 10000 при ошибках.
Код в exceptions.py:

```
def multiply_numbers(a, b):
    try:
        if a == 0 or b == 0:
            raise ValueError("Одно из чисел равно
нулю!")
        result = a * b
        if result > 10000:
            raise OverflowError("Результат слишком
большой!")
    except ValueError as e:
        print(f"Ошибка: {e}")
        return 0
    except OverflowError as e:
        print(f"Ошибка: {e}")
        return 10000
    return result
```

Вызов в main.py:

```
print("Умножение:", multiply_numbers(5, 6))
```

Результат: "Умножение: 30"

Описание: Функция check_age проверяет возраст и возвращает 25 при ошибках.
Код в exceptions.py:

```
def check_age(age):
    try:
        if age < 0:
```

```
            raise ValueError("Возраст не может быть
отрицательным!")
        if age > 150:
            raise ValueError("Слишком большой
возраст!")
    except ValueError as e:
        print(f"Ошибка возраста: {e}")
        return 25
    return age
```

Вызов в main.py:

```
print("Проверка возраста:", check_age(25))
```

Результат: "Проверка возраста: 25"

9: Главная функция

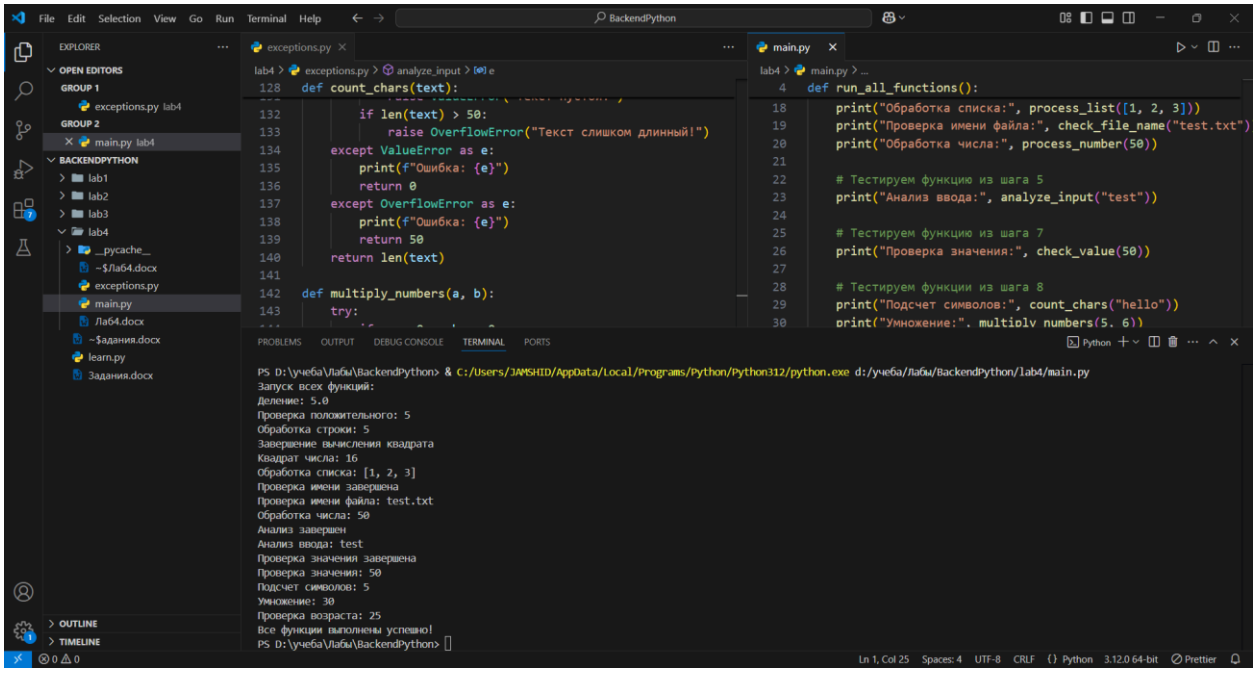
Описание: Функция run_all_functions вызывает все предыдущие функции и завершает работу программы.
Код в main.py:

```
def run_all_functions():
    print("Запуск всех функций:")
    print("Деление:", divide_numbers(10, 2))
    print("Проверка положительного:",
check_positive(5))
    print("Обработка строки:", process_string("hello"))
    print("Квадрат числа:", calculate_square(4))
    print("Обработка списка:", process_list([1, 2, 3]))
    print("Проверка имени файла:",
check_file_name("test.txt"))
    print("Обработка числа:", process_number(50))
    print("Анализ ввода:", analyze_input("test"))
    print("Проверка значения:", check_value(50))
    print("Подсчет символов:", count_chars("hello"))
    print("Умножение:", multiply_numbers(5, 6))
    print("Проверка возраста:", check_age(25))
    print("Все функции выполнены успешно!")
```

Вызов в main.py:

```python
if __name__ == "__main__":
    run_all_functions()
```

Результат: Вывод всех результатов функций в консоль, заканчивается строкой "Все функции выполнены успешно!" без необработанных исключений.



Ссылка на Github: https://github.com/QodirovJM/BackendPython