

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

**Факультет Информационных технологий
Кафедра Информатики и информационных технологий**

**направление подготовки
09.03.02 «Информационные системы и технологии»**

ЛАБОРАТОРНАЯ (ПРАКТИЧЕСКАЯ РАБОТА) № 4

Дисциплина: Backend-разработка

Тема: Исключения

Выполнил(а): студент(ка) группы 221-373

Хрей Кирилл Андреевич
(Фамилия И.О.)

Дата, подпись 17.04.2024 _____
(Дата) (Подпись)

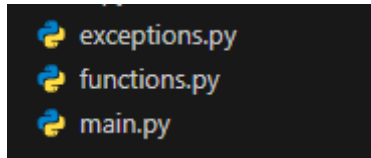
Проверил: _____
(Фамилия И.О., степень, звание) (Оценка)

Дата, подпись _____
(Дата) (Подпись)

**Москва
2024**

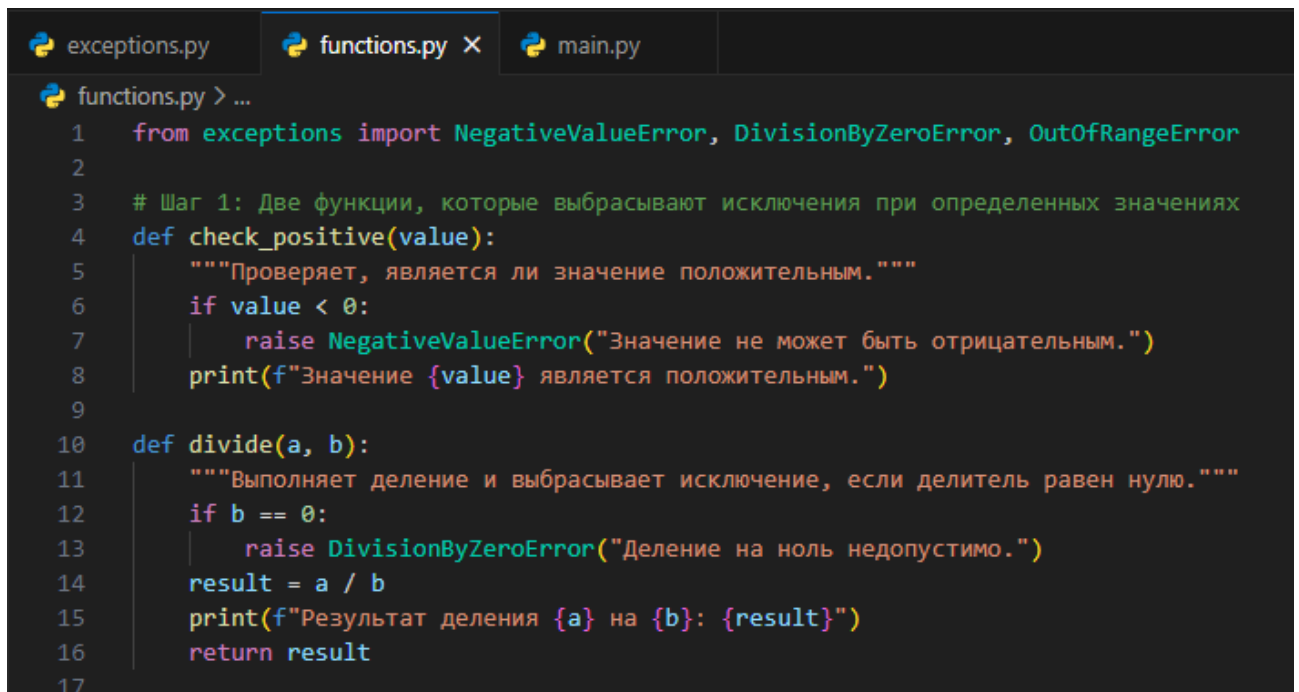
Ссылка на github - <https://github.com/Mospolytech-IIT/lab4-KirillKhrey>

Создаем файлы exceptions.py, funcrions.py, main.py.



Пропишем функции по заданию:

1. Минимум 2 функции, выбрасывающие исключения без обработчиков

A screenshot of a code editor with three tabs: exceptions.py, functions.py (active), and main.py. The functions.py file contains the following code:

```
1  from exceptions import NegativeValueError, DivisionByZeroError, OutOfRangeError
2
3  # Шаг 1: Две функции, которые выбрасывают исключения при определенных значениях
4  def check_positive(value):
5      """Проверяет, является ли значение положительным."""
6      if value < 0:
7          raise NegativeValueError("Значение не может быть отрицательным.")
8      print(f"Значение {value} является положительным.")
9
10 def divide(a, b):
11     """Выполняет деление и выбрасывает исключение, если делитель равен нулю."""
12     if b == 0:
13         raise DivisionByZeroError("Деление на ноль недопустимо.")
14     result = a / b
15     print(f"Результат деления {a} на {b}: {result}")
16     return result
17
```

- `check_positive(value)` — проверяет, является ли значение положительным, и выбрасывает исключение `NegativeValueError`, если значение отрицательное.
- `divide(a, b)` — выполняет деление и выбрасывает исключение `DivisionByZeroError`, если делитель равен нулю.

2. Функция с одним обработчиком общего типа Exception, без блока finally

```
19 # Шаг 2: Функция с одним обработчиком исключений общего типа
20 def safe_divide(a, b):
21     """Выполняет деление с обработкой исключений общего типа."""
22     try:
23         result = divide(a, b)
24         print(f"Результат safe_divide: {result}")
25     except Exception as e:
26         print(f"Произошла ошибка: {e}")
27
```

- `safe_divide(a, b)` — выполняет деление с обработкой исключений общего типа, т.е. перехватывает все исключения и выводит их сообщение, не останавливая выполнение программы.

3. Функция с обработчиком исключений и блоком finally

```
29 # Шаг 3: Функция с обработчиком исключений и блоком finally
30 def process_value(value):
31     """Проверяет положительность значения и использует блок finally для завершения."""
32     try:
33         check_positive(value)
34     except Exception as e:
35         print(f"Произошла ошибка: {e}")
36     finally:
37         print("Завершение проверки положительности значения.")
38
```

- `process_value(value)` — вызывает функцию `check_positive` для проверки положительности значения и использует блок `finally`, который всегда выполняется, даже если возникла ошибка.

4. Три функции с несколькими обработчиками исключений, используя пользовательские исключения

```
# Шаг 4: Три функции с несколькими обработчиками исключений, используя пользовательские исключения
def calculate(a, b):
    """Выполняет деление и проверку диапазона результата."""
    try:
        result = divide(a, b)
        if not (1 <= result <= 100):
            raise OutOfRangeError("Результат вне допустимого диапазона (1-100).")
        print(f"Результат операции calculate: {result}")
        return result
    except DivisionByZeroError as e:
        print(f"Ошибка деления: {e}")
    except OutOfRangeError as e:
        print(f"Ошибка диапазона: {e}")
    finally:
        print("Завершение операции деления с проверкой диапазона.")

def check_range(value):
    """Проверяет, что значение в диапазоне 1-100."""
    try:
        if value < 1 or value > 100:
            raise OutOfRangeError("Значение вне допустимого диапазона (1-100).")
        print(f"Значение {value} в допустимом диапазоне.")
    except OutOfRangeError as e:
        print(f"Ошибка диапазона: {e}")
    finally:
        print("Завершение проверки диапазона.")
```

```
67 def evaluate_conditions(value):
68     """Проверяет разные условия на значение и выбрасывает исключения."""
69     try:
70         check_positive(value)
71         check_range(value)
72     except NegativeValueError as e:
73         print(f"Ошибка отрицательного значения: {e}")
74     except OutOfRangeError as e:
75         print(f"Ошибка диапазона: {e}")
76     finally:
77         print("Завершение проверки условий.")
78
```

- `calculate(a, b)` — выполняет деление и проверяет, находится ли результат в диапазоне от 1 до 100. В случае деления на ноль выбрасывает `DivisionByZeroError`, если результат вне диапазона — выбрасывает `OutOfRangeError`.

- `check_range(value)` — проверяет, что значение в пределах диапазона от 1 до 100, выбрасывает исключение `OutOfRangeError`, если оно вне диапазона.

- `evaluate_conditions(value)` — вызывает функции `check_positive` и `check_range` для проверки условий и выбрасывает исключения `NegativeValueError` или `OutOfRangeError`, если условия не выполнены.

5. Функция с генерацией и обработкой исключений для разных условий

```
79
80 # Шаг 5: Функция с генерацией и обработкой исключений
81 def generate_exceptions(value):
82     """Выбрасывает и обрабатывает исключения для различных условий."""
83     try:
84         if value < 0:
85             raise NegativeValueError("Отрицательное значение.")
86         elif value == 0:
87             raise DivisionByZeroError("Значение не должно быть нулем.")
88         elif value > 100:
89             raise OutOfRangeError("Значение слишком большое.")
90         print(f"Значение {value} прошло все проверки.")
91     except NegativeValueError as e:
92         print(f"Обработка отрицательного значения: {e}")
93     except DivisionByZeroError as e:
94         print(f"Обработка деления на ноль: {e}")
95     except OutOfRangeError as e:
96         print(f"Обработка значения вне диапазона: {e}")
97     finally:
98         print("Завершение генерации и обработки исключений.")
99
```

- `generate_exceptions(value)` — выбрасывает различные исключения в зависимости от значения: `NegativeValueError` для отрицательных значений, `DivisionByZeroError` для значения ноль, и `OutOfRangeError` для значений, превышающих 100.

6. Определение пользовательских исключений

```
exceptions.py X functions.py main.py
exceptions.py > ...
1  # Шаг 6: Определение пользовательских исключений
2
3  class NegativeValueError(Exception):
4      """Исключение для отрицательных значений."""
5      pass
6
7  class DivisionByZeroError(Exception):
8      """Исключение для деления на ноль."""
9      pass
10
11 class OutOfRangeError(Exception):
12     """Исключение для значений вне допустимого диапазона."""
13     pass
14
```

- NegativeValueError, DivisionByZeroError, OutOfRangeError — все эти исключения определены в файле exceptions.py.

7. Функция с пользовательским исключением

```
101 # Шаг 7: Функция с пользовательским исключением
102 def custom_exception_function(value):
103     """Выбрасывает NegativeValueError для отрицательных значений."""
104     try:
105         check_positive(value)
106     except NegativeValueError as e:
107         print(f"Обработка пользовательского исключения: {e}")
108     finally:
109         print("Завершение проверки пользовательского исключения.")
110
```

- custom_exception_function(value) — выбрасывает исключение NegativeValueError, если значение отрицательное.

8. Функции, демонстрирующие работу исключений.

```

112 # Шаг 8: Функции, демонстрирующие работу исключений
113
114 def demo_check_positive():
115     """Демонстрация выбрасывания исключения NegativeValueError."""
116     try:
117         check_positive(-1)
118     except NegativeValueError as e:
119         print(f"Демонстрация: {e}")
120
121 def demo_divide():
122     """Демонстрация выбрасывания исключения DivisionByZeroError."""
123     try:
124         divide(10, 0)
125     except DivisionByZeroError as e:
126         print(f"Демонстрация: {e}")
127
128 def demo_check_range():
129     """Демонстрация выбрасывания исключения OutOfRangeError."""
130     try:
131         check_range(150)
132     except OutOfRangeError as e:
133         print(f"Демонстрация: {e}")
134

```

- demo_check_positive(), demo_divide(), demo_check_range() — демонстрируют выбрасывание исключений для различных условий, таких как отрицательное значение, деление на ноль, значение вне диапазона.

9. Функция, которая последовательно вызывает все функции в main.py

```

from functions import (
    check_positive, divide, safe_divide, process_value,
    calculate, check_range, evaluate_conditions, generate_exceptions,
    custom_exception_function
)
from functions import demo_check_positive, demo_divide, demo_check_range

# Шаг 9: Функция, которая последовательно вызывает все функции
def run_all_functions():
    """Запускает все функции последовательно для демонстрации работы исключений."""
    try:
        print("Тестирование check_positive(5):")
        check_positive(5) # Положительное значение
        print("\nТестирование divide(10, 1):")
        divide(10, 1) # Деление без ошибок
        print("\nТестирование safe_divide(10, 1):")
        safe_divide(10, 1) # Без ошибки
        print("\nТестирование process_value(-3):")
    except:
        pass

```

```

process_value(-3) # Отрицательное значение
print("\nТестирование calculate(10, 1):")
calculate(10, 1) # Должен вернуться корректный результат
print("\nТестирование check_range(99):")
check_range(99) # В пределах диапазона
print("\nТестирование evaluate_conditions(200):")
evaluate_conditions(200) # Выбросит ошибку диапазона
print("\nТестирование generate_exceptions(1):")
generate_exceptions(1) # Не должно выбрасывать ошибку
print("\nТестирование custom_exception_function(10):")
custom_exception_function(10) # Не должно выбрасывать исключение
print("\nТестирование demo_check_positive():")
demo_check_positive() # Демонстрация ошибки при отрицательном значении
print("\nТестирование demo_divide():")
demo_divide() # Демонстрация ошибки деления на ноль
print("\nТестирование demo_check_range():")
demo_check_range() # Демонстрация ошибки диапазона
except Exception as e:
    print(f"Ошибка в основной функции: {e}")
finally:
    print("Все функции завершены корректно.")

if __name__ == "__main__":
    run_all_functions()

```

Запустим программу и проверим:

```

Тестирование check_positive(5):
Значение 5 является положительным.

Тестирование divide(10, 1):
Результат деления 10 на 1: 10.0

Тестирование safe_divide(10, 1):
Результат деления 10 на 1: 10.0
Результат safe_divide: 10.0

Тестирование process_value(-3):
Произошла ошибка: Значение не может быть отрицательным.
Завершение проверки положительности значения.

Тестирование calculate(10, 1):
Результат деления 10 на 1: 10.0
Результат операции calculate: 10.0
Завершение операции деления с проверкой диапазона.

```


Тестирование `check_range(99)`:

Значение 99 в допустимом диапазоне.

Завершение проверки диапазона.

Тестирование `evaluate_conditions(200)`:

Значение 200 является положительным.

Ошибка диапазона: Значение вне допустимого диапазона (1-100).

Завершение проверки диапазона.

Завершение проверки условий.

Тестирование `generate_exceptions(1)`:

Значение 1 прошло все проверки.

Завершение генерации и обработки исключений.

Тестирование `custom_exception_function(10)`:

Значение 10 является положительным.

Завершение проверки пользовательского исключения.

Тестирование `demo_check_positive()`:

Демонстрация: Значение не может быть отрицательным.

Тестирование `demo_divide()`:

Демонстрация: Деление на ноль недопустимо.

Тестирование `demo_check_range()`:

Ошибка диапазона: Значение вне допустимого диапазона (1-100).

Завершение проверки диапазона.

Все функции завершены корректно.

Листинг кода:

exceptions.py

```
# Шаг 6: Определение пользовательских исключений

class NegativeValueError(Exception):
    """Исключение для отрицательных значений."""
    pass

class DivisionByZeroError(Exception):
    """Исключение для деления на ноль."""
    pass

class OutOfRangeError(Exception):
    """Исключение для значений вне допустимого диапазона."""
    pass
```

functions.py

```
from exceptions import NegativeValueError, DivisionByZeroError, OutOfRangeError

# Шаг 1: Две функции, которые выбрасывают исключения при определенных значениях
def check_positive(value):
    """Проверяет, является ли значение положительным."""
    if value < 0:
        raise NegativeValueError("Значение не может быть отрицательным.")
    print(f"Значение {value} является положительным.")

def divide(a, b):
    """Выполняет деление и выбрасывает исключение, если делитель равен нулю."""
    if b == 0:
        raise DivisionByZeroError("Деление на ноль недопустимо.")
    result = a / b
    print(f"Результат деления {a} на {b}: {result}")
    return result

# Шаг 2: Функция с одним обработчиком исключений общего типа
def safe_divide(a, b):
    """Выполняет деление с обработкой исключений общего типа."""
    try:
        result = divide(a, b)
        print(f"Результат safe_divide: {result}")
    except Exception as e:
        print(f"Произошла ошибка: {e}")
```

```

# Шаг 3: Функция с обработчиком исключений и блоком finally
def process_value(value):
    """Проверяет положительность значения и использует блок finally для
    завершения."""
    try:
        check_positive(value)
    except Exception as e:
        print(f"Произошла ошибка: {e}")
    finally:
        print("Завершение проверки положительности значения.")

# Шаг 4: Три функции с несколькими обработчиками исключений, используя
пользовательские исключения
def calculate(a, b):
    """Выполняет деление и проверку диапазона результата."""
    try:
        result = divide(a, b)
        if not (1 <= result <= 100):
            raise OutOfRangeError("Результат вне допустимого диапазона (1-100).")
        print(f"Результат операции calculate: {result}")
        return result
    except DivisionByZeroError as e:
        print(f"Ошибка деления: {e}")
    except OutOfRangeError as e:
        print(f"Ошибка диапазона: {e}")
    finally:
        print("Завершение операции деления с проверкой диапазона.")

def check_range(value):
    """Проверяет, что значение в диапазоне 1-100."""
    try:
        if value < 1 or value > 100:
            raise OutOfRangeError("Значение вне допустимого диапазона (1-100).")
        print(f"Значение {value} в допустимом диапазоне.")
    except OutOfRangeError as e:
        print(f"Ошибка диапазона: {e}")
    finally:
        print("Завершение проверки диапазона.")

def evaluate_conditions(value):
    """Проверяет разные условия на значение и выбрасывает исключения."""
    try:
        check_positive(value)
        check_range(value)
    except NegativeValueError as e:
        print(f"Ошибка отрицательного значения: {e}")
    except OutOfRangeError as e:
        print(f"Ошибка диапазона: {e}")
    finally:

```

```

        print("Завершение проверки условий.")

# Шаг 5: Функция с генерацией и обработкой исключений
def generate_exceptions(value):
    """Выбрасывает и обрабатывает исключения для различных условий."""
    try:
        if value < 0:
            raise NegativeValueError("Отрицательное значение.")
        elif value == 0:
            raise DivisionByZeroError("Значение не должно быть нулем.")
        elif value > 100:
            raise OutOfRangeError("Значение слишком большое.")
        print(f"Значение {value} прошло все проверки.")
    except NegativeValueError as e:
        print(f"Обработка отрицательного значения: {e}")
    except DivisionByZeroError as e:
        print(f"Обработка деления на ноль: {e}")
    except OutOfRangeError as e:
        print(f"Обработка значения вне диапазона: {e}")
    finally:
        print("Завершение генерации и обработки исключений.")

# Шаг 7: Функция с пользовательским исключением
def custom_exception_function(value):
    """Выбрасывает NegativeValueError для отрицательных значений."""
    try:
        check_positive(value)
    except NegativeValueError as e:
        print(f"Обработка пользовательского исключения: {e}")
    finally:
        print("Завершение проверки пользовательского исключения.")

# Шаг 8: Функции, демонстрирующие работу исключений
def demo_check_positive():
    """Демонстрация выбрасывания исключения NegativeValueError."""
    try:
        check_positive(-1)
    except NegativeValueError as e:
        print(f"Демонстрация: {e}")

def demo_divide():
    """Демонстрация выбрасывания исключения DivisionByZeroError."""
    try:
        divide(10, 0)
    except DivisionByZeroError as e:
        print(f"Демонстрация: {e}")

```

```
def demo_check_range():
    """Демонстрация выбрасывания исключения OutOfRangeError."""
    try:
        check_range(150)
    except OutOfRangeError as e:
        print(f"Демонстрация: {e}")
```

main.py

```
from functions import (
    check_positive, divide, safe_divide, process_value,
    calculate, check_range, evaluate_conditions, generate_exceptions,
    custom_exception_function
)
from functions import demo_check_positive, demo_divide, demo_check_range

# Шаг 9: Функция, которая последовательно вызывает все функции
def run_all_functions():
    """Запускает все функции последовательно для демонстрации работы исключений."""
    try:
        print("Тестирование check_positive(5):")
        check_positive(5) # Положительное значение
        print("\nТестирование divide(10, 1):")
        divide(10, 1) # Деление без ошибок
        print("\nТестирование safe_divide(10, 1):")
        safe_divide(10, 1) # Без ошибки
        print("\nТестирование process_value(-3):")
        process_value(-3) # Отрицательное значение
        print("\nТестирование calculate(10, 1):")
        calculate(10, 1) # Должен вернуться корректный результат
        print("\nТестирование check_range(99):")
        check_range(99) # В пределах диапазона
        print("\nТестирование evaluate_conditions(200):")
        evaluate_conditions(200) # Выбросит ошибку диапазона
        print("\nТестирование generate_exceptions(1):")
        generate_exceptions(1) # Не должно выбрасывать ошибку
        print("\nТестирование custom_exception_function(10):")
        custom_exception_function(10) # Не должно выбрасывать исключение
        print("\nТестирование demo_check_positive():")
        demo_check_positive() # Демонстрация ошибки при отрицательном значении
        print("\nТестирование demo_divide():")
        demo_divide() # Демонстрация ошибки деления на ноль
        print("\nТестирование demo_check_range():")
        demo_check_range() # Демонстрация ошибки диапазона
    except Exception as e:
        print(f"Ошибка в основной функции: {e}")
    finally:
        print("Все функции завершены корректно.")
```

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