Обработка ошибок и исключений

Конструкция try-except-finally

Теория 🕮

При программировании на Python мы можем столкнуться с двумя типами ошибок. Первый тип представляют синтаксические ошибки (syntax error). Они появляются в результате нарушения синтаксиса языка программирования при написании исходного кода. При наличии таких ошибок программа не может быть скомпилирована. При работе в какой-либо среде разработки, например, в PyCharm, IDE сама может отслеживать синтаксические ошибки и каким-либо образом их выделять.

Второй тип ошибок представляют ошибки выполнения (runtime error). Они появляются в уже скомпилированной программе в процессе ее выполнения.

Пример 1 (успешное выполнение)

```
string = input("Введите число: ") # Ввод числа "5"
number = int(string)
print(number)
# Данный скрипт успешно скомпилируется и выполнится, так как строка "5" вполне может быть конвертирована в число.
```

Пример 2 (компиляция с ошибкой)

```
string = input("Введите число: ")   # Ввод "MPT"
number = int(string)
print(number)

# При выполнении этого скрипта будет выброшено исключение ValueError, так как строку "hello" нельзя преобразовать в число: ValueError: invalid literal for int() with base 10: 'MPT'
```

При возникновении исключения работа программы прерывается, и чтобы избежать подобного поведения и обрабатывать исключения в Python есть конструкция try..except.

try..except finally

Как выглядит конструкция

```
try:
    инструкции
except [Тип_исключения]:
    инструкции
```

Весь основной код, в котором потенциально может возникнуть исключение, помещается после ключевого слова try. Если в этом коде генерируется исключение, то работа кода в блоке try прерывается, и выполнение переходит в блок except.

После ключевого слова except опционально можно указать, какое исключение будет обрабатываться (например, ValueError или KeyError). После слова except на следующей стоке идут инструкции блока except, выполняемые при возникновении исключения.

Рассмотрим обработку исключения на примере преобразовании строки в число:

Обработка исключения

```
try:
    number = int(input("Введите число: "))
    print("Введенное число:", number)
except:
    print("Преобразование прошло неудачно")
finally:
    print("Завершение программы")

# try - выполняется всегда
# except - тогда когда в try, что то пошло не так
# При обработке исключений также можно использовать необязательный блок finally.
Отличительной особенностью этого блока является то, что он выполняется вне
зависимости, было ли сгенерировано исключение:
```

except, обработка разных типов исключений

Базовые типы

В Python есть следующие базовые типы исключений:

```
BaseException: базовый тип для всех встроенных исключений

Exception: базовый тип, который обычно применяется для создания своих типов
исключений

ArithmeticError: базовый тип для исключений, связанных с арифметическими
операциями (OverflowError, ZeroDivisionError, FloatingPointError).

BufferError: тип исключения, которое возникает при невозможности выполнить
операцию с буффером

LookupError: базовый тип для исключений, которое возникают при обращении в
коллекциях по некорректному ключу или индексу (например, IndexError, KeyError)

От этих классов наследуются все конкретные типы исключений. В Python обладает
довольно большим списком встроенных исключений. Весь этот список можно посмотреть
в документации. Перечислю только некоторые наиболее часто встречающиеся:

IndexError: исключение возникает, если индекс при обращении к элементу коллекции
находится вне допустимого диапазона

KeyError: возникает, если в словаре отсутствует ключ, по которому происходит
обращение к элементу словаря.

OverflowError: возникает, если результат арифметической операции не может быть
представлен текущим числовым типом (обычно типом float).

RecursionError: возникает, если превышена допустимая глубина рекурсии.

TypeError: возникает, если операция или функция применяется к значению
недопустимого типа.

ValueError: возникает, если операция или функция получают объект корректного типа
с некорректным значением.

ZeroDivisionError: возникает при делении на ноль.

NotImplementedError: тип исключения для указания, что какие-то методы класса не
реализованы

ModuleNotFoundError: возникает при при невозможности найти модуль при его импорте
директивой import

OSError: тип исключений, которые генерируются при возникновении ошибок системы
(например, невозможно найти файл, память диска заполнена и т.д.)
```

Генерация различных типов исключения

```
try:
    number1 = int(input("Введите первое число: "))
```

```
    number2 = int(input("Введите второе число: "))
    print("Результат деления:", number1/number2)
except ValueError:
    print("Преобразование прошло неудачно")
except BaseException:
    print("Общее исключение")
except ZeroDivisionError:
    print("Попытка деления числа на ноль")
finally:
    print("Завершение программы")

# Если возникнет исключение в результате преобразования строки в число, то оно
будет обработано блоком except ValueError. Если же второе число будет равно нулю,
то есть будет деление на ноль, тогда возникнет исключение ZeroDivisionError, и оно
будет обработано блоком except ZeroDivisionError.
```

Обработка нескольких типов исключения

Python позволяет в одном блоке except обрабатывать сразу несколько типов исключений. В этом случае все типы исключения передаются в скобках:

```
try:
    number1 = int(input("Введите первое число: "))
    number2 = int(input("Введите второе число: "))
    print("Результат деления:", number1/number2)
except (ZeroDivisionError, ValueError):    #  обработка двух типов исключений -
ZeroDivisionError и ValueError
    print("Попытка деления числа на ноль или некорректный ввод")
finally:
    print("Завершение программы")
```

Получение информации об исключении

С помощью оператора as мы можем передать всю информацию об исключении в переменную, которую затем можно использовать в блоке except:

Пример

```
try:
    number1 = int(input("Введите первое число: "))
    number2 = int(input("Введите второе число: "))
```

```
    print("Результат деления:", number1/number2)
except ValueError as e:
    print("Преобразование прошло неудачно","Вывод информации о ошибке", e)
except BaseException:
    print("Общее исключение")
finally:
    print("Завершение программы")
```

Генерация исключений и создание своих типов исключений 🧁

Генерация исключений и оператор raise

Для создания своего исключения нам необходимо использовать оператор

Оператору raise передается объект BaseException - в данном случае объект Exception. В конструктор этого типа можно ему передать сообщение, которое затем можно вывести пользователю. В итоге, если number2 будет равно 0, то сработает оператор raise, который сгенерирует исключение. В итоге управление программой перейдет к блоку except, который обрабатывает исключения типа Exception:

```
try:
    number1 = int(input("Введите первое число: "))
    number2 = int(input("Введите второе число: "))
    if number2 <= 5:
        raise Exception("Второе число не должно быть меньше или равно 5")
    print("Результат деления двух чисел:", number1/number2)
except ValueError:
    print("Введены некорректные данные")
except Exception as e:
    print(e)
finally:
    print("Завершение программы")
```

Создание своего исключения:

Для создания своего исключения перейдем в наш уже созданный Class и проведем там некоторые коректировки

Пример

```
# В начале здесь определен класс PersonAgeException, который наследуется от класса
Exception. Как правило, собственные классы исключений наследуются от класса
Exception. Класс PersonAgeException предназначен для исключений, связанных с
возрастом пользователя.

# В конструкторе PersonAgeException получаем три значения - собственное
некорректное значение, которое послужило причиной исключения, а также минимальное
и максимальное значения возраста.
class PersonAgeException(Exception):
    def __init__(self, age, minA, maxA):
        self.age = age
        self.minAge = minA
        self.maxAge = maxA

    def __str__(self):
        return f"Недопустимое значение: {self.age}. " \
               f"Возраст должен быть в диапазоне от {self.minAge} до
{self.maxAge}"

class Person:

    def __init__(self, name, age):
        self.__name = name   # имя человека
        maxAge, minAge = 150,1
        if minAge < age < maxAge:
            self.__age = age        # устанавливаем возраст, если передано
корректное значение
        else:
            # В конструкторе класса Persoon проверяем переданное для возраста
пользователя значение. И если это значение не соответствует определенному
диапазону, то генерируем исключение типа PersonAgeException:
            raise PersonAgeException(age, minAge, maxAge)   # иначе генерируем
исключение
    def info(self):
        print(f"Вас зовут {self.__name} вам {self.__age} лет",)

# Вызов класса в другом файле .py
try:
    TestExc = classTest.Person("Victor", 11)
    TestExc.info()  # Имя: Victor  Возраст: 11

    TestExc2 = classTest.Person("Alex", -3)  # генерируется исключение типа
PersonAgeException
    TestExc2.info()
except classTest.PersonAgeException as e:
    print(e)
```

Задание

5

```
# Сделать два исключения с классами для калькулятора
```

4

```
# Создать два исключения бкз классов для калькулятора
```

3

```
# Повторить все как в файле"
```