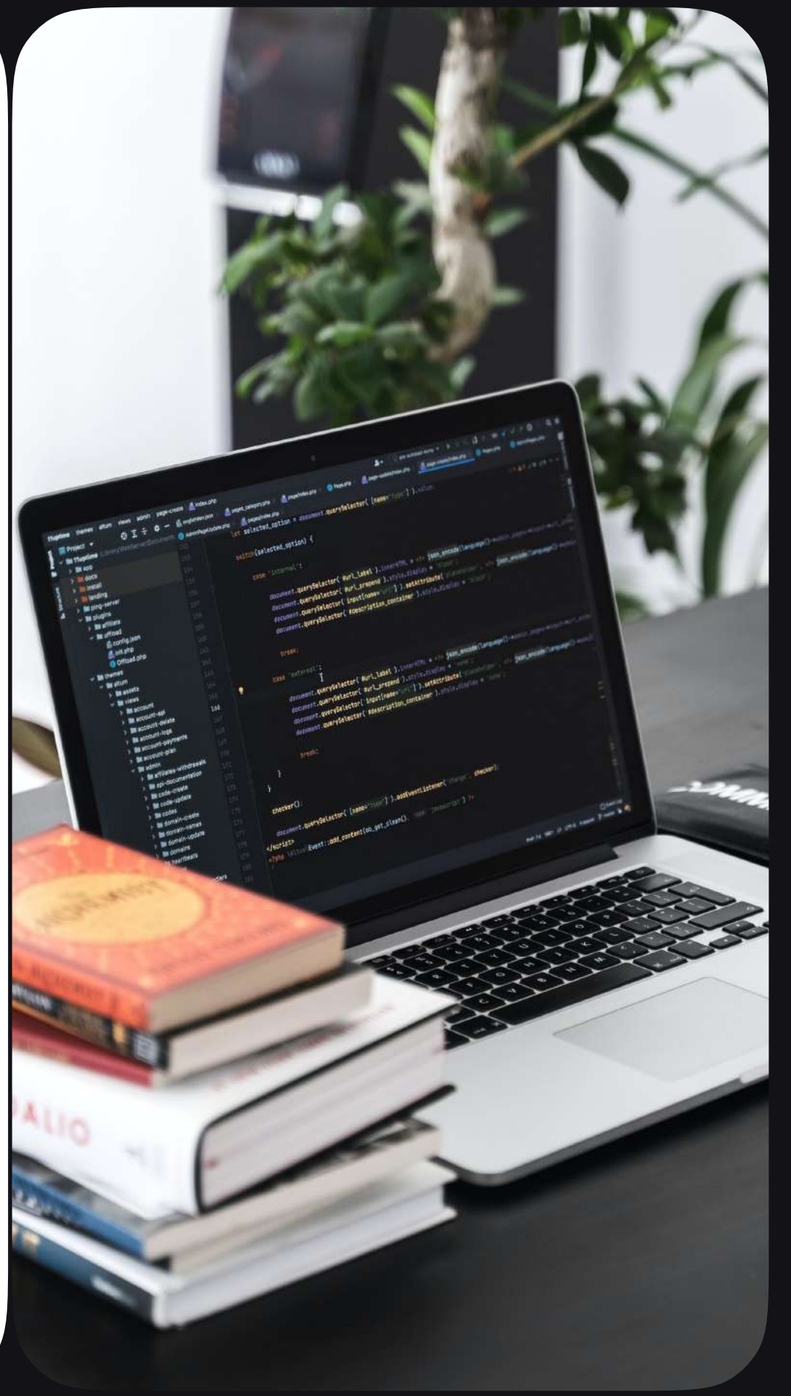
Модуль 2 Занятие 9

Обработка исключений



Исключения

Обработка исключений

try и except

else и finally

Оператор raise

Создание собственных исключений

Ошибки и исключения

В процессе написания кода программы мы сталкиваемся с ошибками. Можно выделить два вида ошибок ошибок: синтаксические ошибки и исключения.

- Синтаксические ошибки обнаруживаются до выполнения программы и легко исправляются.

- Исключения — это ошибки, возникающие в процессе выполнения программы. Такие ошибки могут возникать даже если программа написана верно, и поэтому важно научиться отлавливать такие ошибки, чтобы они не приводили к завершению программы.

Пример

Напишем простую программу:
Необходимо считать два целых числа на одной строке через пробел и найти результат их деления.

Что может пойти не так?
Какие исключения могут возникнуть в процессе выполнения программы?

Решение:

```
a, b = map(int, input().split())
print(a / b)
```

Ввод и вывод:

```
1 2
0.5
```

Примеры исключений

```
>>> 1 / 0
Traceback (most recent call last):
...
ZeroDivisionError: division by zero
```

```
>>> '1' + 2
Traceback (most recent call last):
...
TypeError: can only concatenate str (not "int") to str
```

```
>>> int('abc')
Traceback (most recent call last):
...
ValueError: invalid literal for int() with base 10: 'abc'
```

Исключения бывают разных типов, например, это типы ZeroDivisionError, TypeError, ValueError.

Пример

```
lst = input().split()
if len(lst) == 2 and lst[0].isdigit() and lst[1].isdigit():
    a = int(lst[0])
    b = int(lst[1])
    if b != 0:
        print(a / b)
else:
    print('Второе число не может быть равно 0')
```

Можно предусмотреть все возможные ситуации, которые могут привести к ошибкам во время выполнения программы.

LBYL vs EAFP

Обычно используют одну из двух стратегий работы с ошибками:

- Предотвращение возникновения ошибок — такая стратегия называется Look before you leap (LBYL), в переводе: «Посмотри, прежде чем прыгнуть»
- Обработка ошибок после их возникновения — стратегия Easier to ask forgiveness than permission (EAFP), в переводе: «Легче попросить прощения, чем разрешения»

Try и except

В блоке try находится код, который может вызвать исключение. Если исключения не возникает, то блок except пропускается и выполнение try завершается.

Если во время выполнения в блоке try возникает исключение, то выполняется код из блока except, который обрабатывает данное исключение.

```
try:
    a, b = map(int, input().split())
    print(a / b)
except ValueError:
    print('Необходимо ввести 2 числа через пробел')
except ZeroDivisionError:
    print('Второе число не может быть равно 0')
```

Пример

```
try:
    a, b = map(int, input().split())
    print(a / b)
except (ZeroDivisionError, ValueError):
    print('Ошибка при выполнении')
```

В блоке except можно указать сразу несколько типов исключений.

Пример

```
try:
    a, b = map(int, input().split())
    result = a / b
except Exception as e:
    print(e)
```

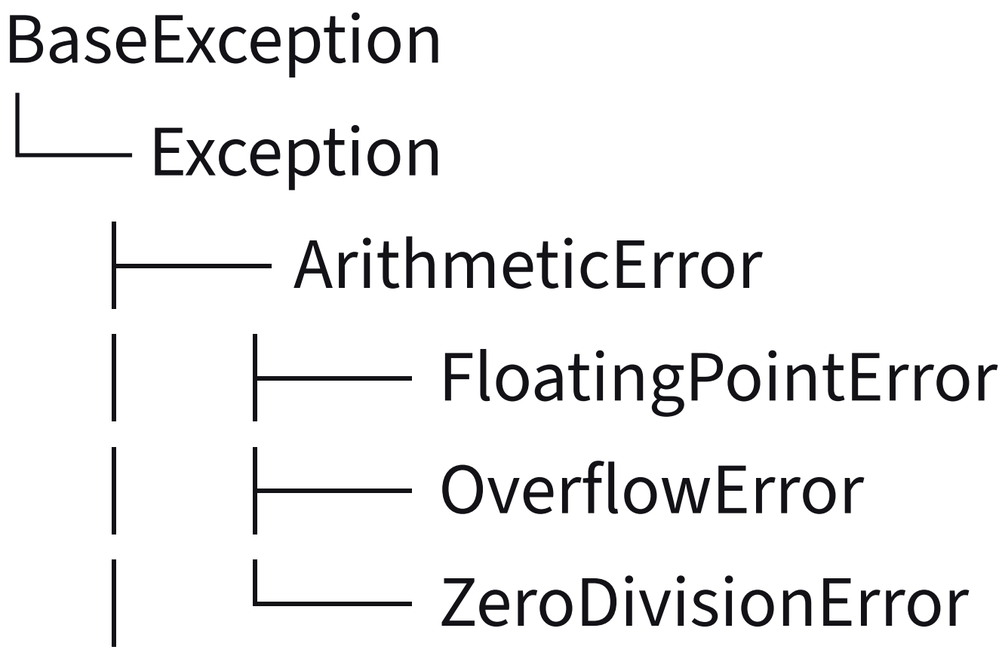
Если необходимо перехватывать все исключения, то можно использовать исключение Exception.
В этом случае полезно иметь доступ к объекту ошибки, для этого используется ключевое слово as и дальше переменная, которая будет ссылаться на этот объект.

Ввод и вывод:

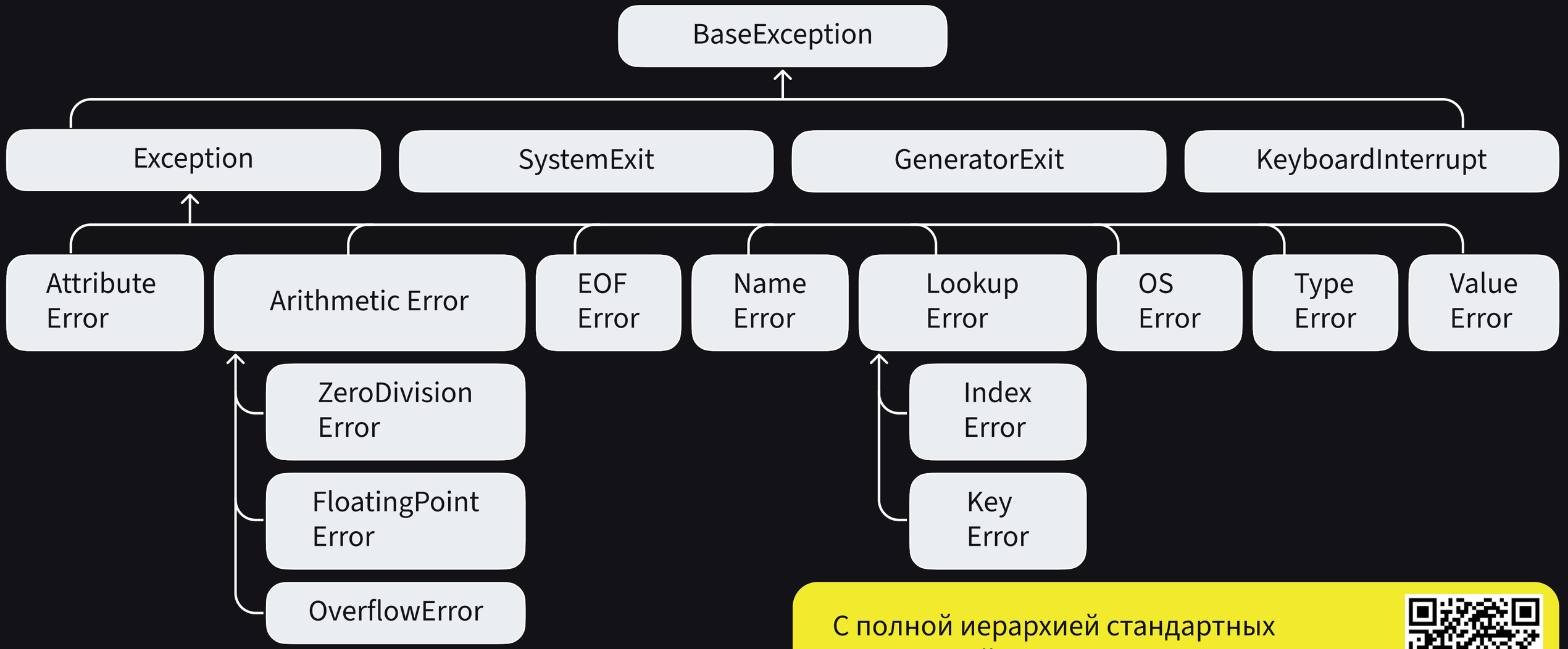
```
1 0
division by zero
```

Исключения это классы

Все исключения в Python — это классы, которые наследуются от базового класса BaseException. BaseException — это базовый класс для всех стандартных исключений.



Иерархия исключений



С полной иерархией стандартных исключений можно ознакомиться на странице официальной документации

else и finally

```
try:
    a, b = map(int, input().split())
    result = a / b
except ZeroDivisionError:
    print('Второе число не может быть равно 0')
except ValueError:
    print('Необходимо ввести 2 числа через пробел')
else:
    print(result)
finally:
    print('Конец программы')
```

Блок else выполняется в случае, если в блоке try не было вызвано исключение.

Блок finally выполняется всегда, вне зависимости от того, было вызвано исключение или нет.

Ввод и вывод:

```
1 2
0.5
Конец программы
```

raise

```
a, b = map(int, input().split())
if b == 0:
    raise ZeroDivisionError
else:
    result = a / b
```

Ввод:

```
1 0
```

Ошибка:

```
ZeroDivisionError
```

Исключения можно вызывать самостоятельно, с помощью оператора raise.

Также можно дополнительно указать в качестве параметра строку с сообщением об ошибке.

```
raise ZeroDivisionError('Нельзя делить на 0')
```

Создание собственных исключений

```
class MyError(Exception):
    pass


try:
    raise MyError('Ой, ошибка')
except MyError as e:
    print(e)
    print(e.__class__.__name__)
```

При создании собственного исключения создаем класс, обязательно выполнив наследование от стандартного исключения.

Имена для классов исключений обычно заканчиваются на «Error», аналогично именам стандартных исключений.

Ввод:

```
Ой, ошибка
MyError
```

Пример

```
class MinLengthError(Exception):
    pass


def check_password(password):
    if len(password) < 8:
        raise MinLengthError('Пароль должен быть не менее 8 символов')


password = input()
try:
    check_password(password)
except MinLengthError as e:
    print(e)
else:
    print('OK')
```

Создадим исключение MinLengthError, которое будем вызывать при проверке длины пароля.

Итоги

Общий синтаксис обработки исключений:

```
try:
    код, который может вызвать исключение
except Исключение1:
    код обработки Исключения1
except Исключение2:
    код обработки Исключения2
и так далее…
else:
    код, который выполняется если не было вызвано исключение в блоке try
finally:
    код, который выполняется всегда
```

Итоги

- Все исключения в Python — это классы, которые наследуются от базового класса BaseException.
- Исключения можно вызывать самостоятельно, с помощью оператора raise.
- При создании собственного исключения необходимо создать класс, обязательно выполнив наследование от стандартного исключения. Имена для классов исключений обычно заканчиваются на «Error», аналогично именам стандартных исключений.