Работа с исключениями. Модули стандартной библиотеки Python.

Лавприт Сингх-Пальчевская
младший научный сотрудник МГУ им. Ломоносова,
кафедра биоинженерии

Проверка связи

Отправьте «+», если меня видно и слышно

Если у вас нет звука или изображения:

- перезагрузите страницу
- попробуйте зайти заново
- откройте трансляцию в другом браузере

О чём поговорим сегодня

МФТИ

1. Рассмотрим принципы обработки исключений в Python
2. Отработаем на практике использование конструкции `try-except`
3. Разберём, как устанавливать внешние библиотеки и создавать свои модули
4. Отработаем на практике специальные методы и создание собственных модулей

МФТИ

Давайте поразмышляем

Как часто возникают ошибки при обработке программ?

Ошибки возникают часто по разным причинам:

- ошибка программиста (недостаток понимания используемых инструментов, человеческий фактор);
- ошибка/баг в коде, который находится под ответственностью другого программиста;
- неизбежность (например, в некоторых случаях мы заранее ожидаем деление на ноль и/или другие ошибки).

Сообщения об ошибках в Python

```
ls= [24, 59, 1, 85, 3, 0, 42, 10]
i = int(input('Enter a number: '))
print('Your answer:', ls[i])
```

```
Enter a number: 100
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-1-9db7f6c17797> in <module>()
      1 ls= [24, 59, 1, 85, 3, 0, 42, 10]
      2 i = int(input('Enter a number: '))
----> 3 print('Your answer:', ls[i])

IndexError: list index out of range
```

Какая ошибка возникла?
В какой строчке и в каком месте программы возникла ошибка?

исключение (exception) - ещё один тип данных в python

- **исключения (exceptions)** сообщают программисту об ошибках
- каждое исключение содержит *краткую информацию* об ошибке с указанием *строки*, в которой она возникла

МФТИ

Обработка исключительных ситуаций в Python

Исключения в Python

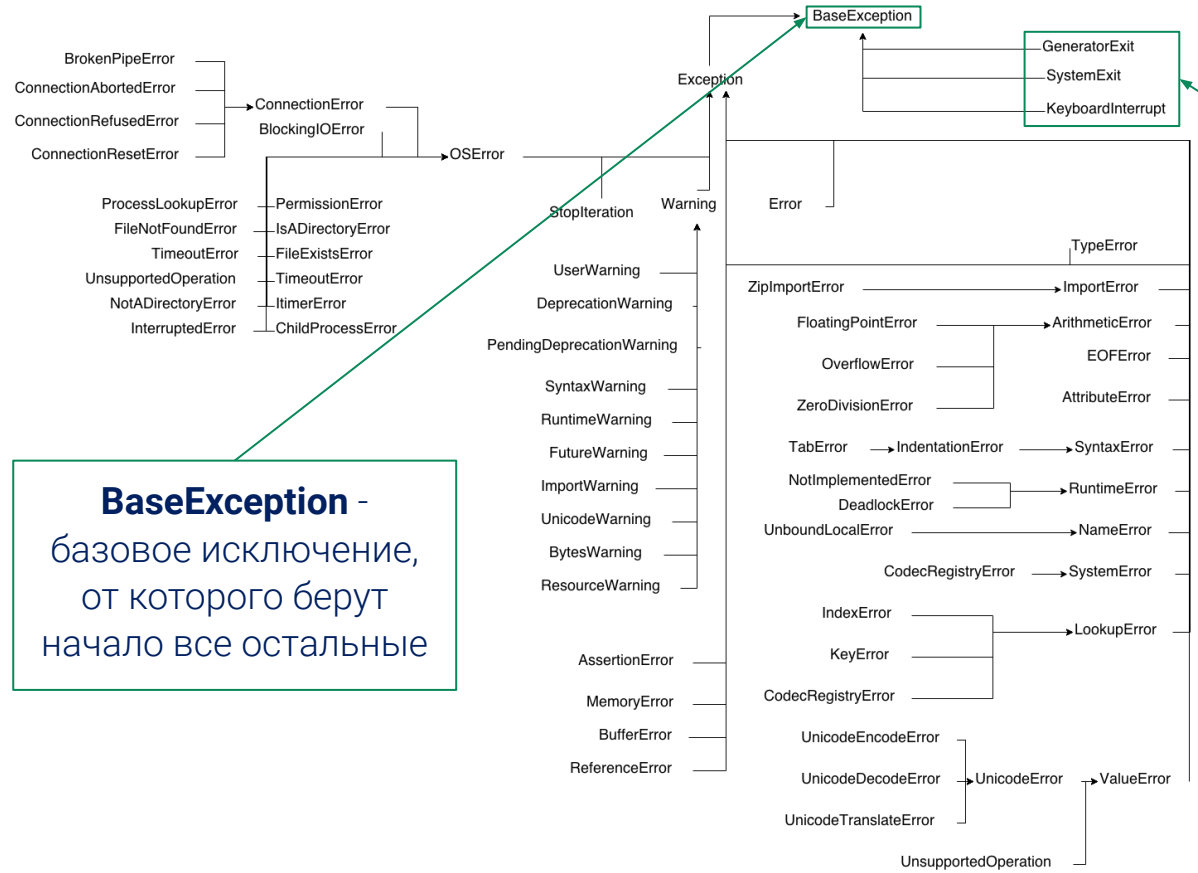
Исключение (exception) - ещё один тип данных в python.

Существует две основные группы:

- Системные исключения и ошибки (обработку этих исключений ***лучше не делать***, а если и делать, то надо четко понимать для чего)
- Обыкновенные исключения (в полном распоряжении программиста)

Полный скрипт: https://colab.research.google.com/drive/1H0mjKGRY851r_84k-VuNjQMRhoOf4XvG?usp=sharing

Иерархия исключений



BaseException - базовое исключение, от которого берут начало все остальные

Системные исключения и ошибки:

SystemExit – исключение, порождаемое функцией sys.exit при выходе из программы

KeyboardInterrupt – возникает при прерывании программы пользователем (обычно сочетанием клавиш Ctrl+C)

GeneratorExit — возникает при вызове метода close объекта generator

Способы обработки исключений:

- описать условие `if ... else ...` (однако способ работает только в случае, если мы знаем какого рода ошибку можем получить, например, деление на ноль)
- использовать специальный способ обработки исключений `try ... except ...`

```
ls= [24, 59, 1, 85, 3, 0, 42, 10]
i = int(input('Enter a number: '))
print('Your answer:', ls[i])
```

```
Enter a number: 100
---------------------------------------------
IndexError                                Trace
<ipython-input-1-9db7f6c17797> in <module>()
      1 ls= [24, 59, 1, 85, 3, 0, 42, 10]
      2 i = int(input('Enter a number: '))
----> 3 print('Your answer:', ls[i])

IndexError: list index out of range
```

Простой способ обработки исключений:

```
ls= [24, 59, 1, 85, 3, 0, 42, 10]
i = int(input('Enter a number: '))
if i < len(ls):
 print('Your answer:', ls[i])
else:
 print('Index must be less than', len(ls))
```

Исключения в Python

МФТИ

The try / except structure

- *"опасный"* участок кода можно обернуть в конструкцию `try ... except ...`;
- если код внутри блока `try` сработает без ошибок - блок `except` пропускается;
- если код внутри блока `try` *"выбрасывает"* ошибку - выполняется блок `except`.

Полный скрипт: hhttps://colab.research.google.com/drive/1H0mjKGRY851r_84k-VuNjQMRhoOf4XvG?usp=sharing

Исключения в Python

МФТИ

The try / except structure

- *"опасный"* участок кода можно обернуть в конструкцию `try ... except ...`;
- если код внутри блока `try` сработает без ошибок - блок `except` пропускается;
- если код внутри блока `try` *"выбрасывает"* ошибку - выполняется блок `except`.

```
ls = [24, 59, 1, 85, 3, 0, 42, 10]
mydict = {'a': 6,
          'b': 10,
          'c': 0}
try:
  i = int(input('Enter a number: '))
  answer = ls[i] + mydict['d']
  print('Your answer:', answer)
except:
  print('Index must be less than', len(ls))
```

```
ls = [24, 59, 1, 85, 3, 0, 42, 10]
mydict = {'a': 6,
          'b': 10,
          'c': 0}
try:
  i = int(input('Enter a number: '))
  answer = ls[i] + mydict['d']
  print('Your answer:', answer)
except IndexError:
  print('Index must be less than', len(ls))
```

Обработка конкретного исключения

Полный скрипт: hhttps://colab.research.google.com/drive/1H0mjKGRY851r_84k-VuNjQMRhoOf4XvG?usp=sharing

Исключения в Python

Обработка нескольких исключений

```
ls = [24, 59, 1, 85, 3, 0, 42, 10]
mydict = {'a': 6,
          'b': 10,
          'c': 0}
try:
  i = int(input('Enter a number:
'))
  d = input('Enter a letter: ')
  answer = ls[i] + mydict[d] +  4
  print('Your answer:', answer)
except (IndexError, KeyError):
  print('Enter correct data.')
```

```
ls = [24, 59, 1, 85, 3, 0, 42, 10]
mydict = {'a': 6,
          'b': 10,
          'c': 0}
try:
  i = int(input('Enter a number: '))
  d = input('Enter a letter: ')
  answer = ls[i] + mydict[d] +  4
  print('Your answer:', answer)
except IndexError:
  print('Index must be less than', len(ls))
except KeyError:
  print('Letter must be one of the symbols:',
','.join(mydict.keys()))
```

Полный скрипт: hhttps://colab.research.google.com/drive/1H0mjKGRY851r_84k-VuNjQMRhoOf4XvG?usp=sharing

Исключения в Python

Генерация исключений и их вызов

```
a, b = int(input()), int(input())
if b==0:
  raise ZeroDivisionError
else:
  # большой блок программы ...
  print(a/b)
  #...
```

```
a, b = int(input()), int(input())
if b==0:
  raise ZeroDivisionError('Деление на 0')
else:
  # большой блок программы ...
  print(a/b)
  #...
```

Полный скрипт: hhttps://colab.research.google.com/drive/1H0mjKGRY851r_84k-VuNjQMRhoOf4XvG?usp=sharing

Исключения в Python

Генерация исключений и их вызов

```
a, b = int(input()), int(input())
if b==0:
  raise ZeroDivisionError
else:
  # большой блок программы ...
  print(a/b)
  #...
```

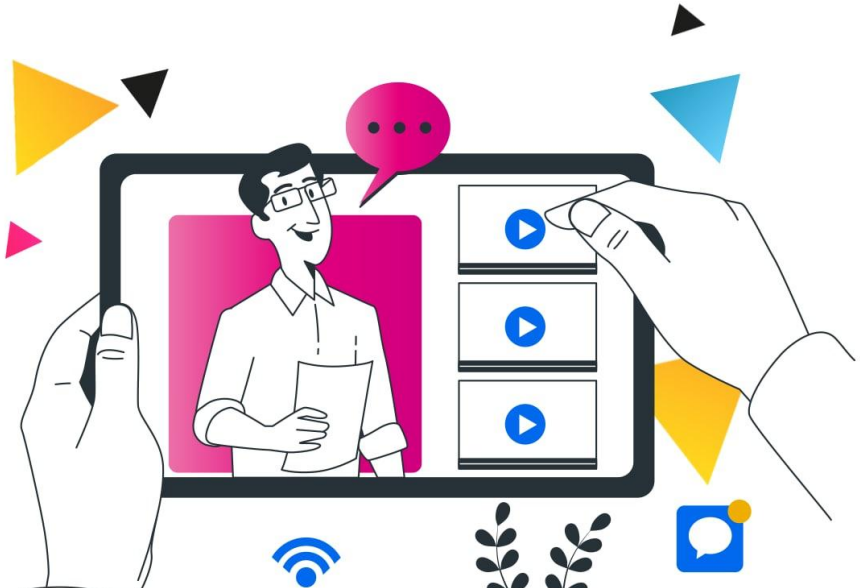
```
a, b = int(input()), int(input())
if b==0:
  raise ZeroDivisionError('Деление на 0')
else:
  # большой блок программы ...
  print(a/b)
  #...
```

```
a, b = int(input()), int(input())
assert(b!=0)
# большой блок программы ...
print(a/b)
#...
```

Подходит для *“отлавливания”* собственных ошибок в коде

Полный скрипт: hhttps://colab.research.google.com/drive/1H0mjKGRY851r_84k-VuNjQMRhoOf4XvG?usp=sharing

Ваши вопросы?



Модули и инструкция import

Модули и инструкция import

Чтобы код был хорошо организован, рекомендуется начать с группировки связанного кода. **Модуль** — это набор связанного кода, сохраненного в файле с расширением `.py`. В модуле можно определить функции, классы и/или переменные. Также можно включать исполняемый код в модули.

Пример модуля (my_module.py):

```
x = 10
def print_lyrics():
  print("I'm a lumberjack, and I'm okay.")
  print('I sleep all night and I work all day.')
```

Полный скрипт: https://colab.research.google.com/drive/1KdNE7t5K7_IpIzcC2q3SKOMFOo-M6ZPw?usp=sharing

Модули и инструкция import

Для того, чтобы подключить модуль (библиотеку), используется ключевое слово `import`.

```
import my_module

my_module.print_lyrics()
print(f'x is {my_module.x}')
```

```
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
x is 10
```

```
from my_module import print_lyrics

print_lyrics()
print(f'x is {x}')
```

```
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-733bee46c7ff> in <module>()
      1 from my_module import print_lyrics
      2 print_lyrics()
----> 3 print(f'x is {my_module.x}')

NameError: name 'my_module' is not defined
```

Полный скрипт: https://colab.research.google.com/drive/1KdNE7t5K7_IpIzcC2q3SKOMFOo-M6ZPw?usp=sharing

Установка внешних библиотек

Установка внешних библиотек

Для установки внешних библиотек в Python существует команда `pip`, обычно она ставится автоматически при установке Python.

```
pip install pandas

pip install pandas==1.3.2

pip install --upgrade pandas

pip uninstall pandas
```

Установка внешних библиотек

МФТИ

```
pip list
pip freeze
```

просмотр всех установленных пакетов в рабочем окружении

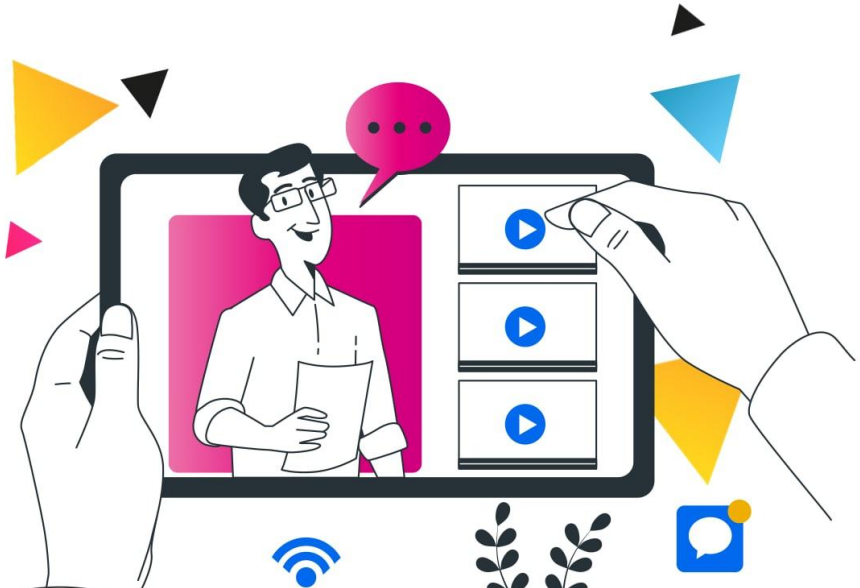
```
pip freeze > requirements.txt
```

запись списка установленных пакетов в файл

```
pip install -r requirements.txt
```

установка всех пакетов разом из файла

Ваши вопросы?



МФТИ

Практика

Задача 1

Напишите **модуль**, содержащий один **класс**, который позволяет хранить информацию о результатах FRET-эксперимента. Класс должен обладать методом, который читает данные из файла (пример).

1 и 3 колонки - это время (одинаковые).

2 и 4 колонки - это интенсивность молекулы-*акцептора* и молекулы-*донора* соответственно.

Метод должен возвращать **объект класса**.

Данные в файле могут быть "грязные" (например, может оказаться меньше или больше столбцов, или вместо чисел может встретиться строка).

По необходимости обработайте ожидаемые исключения.

Задача 1

```
	Cy5	Cy3
Measurement Time (s)	Count rate (kHz)	Measurement Time (s)	Count rate (kHz)
0.00000000	0.000	0.00000000	1.333
0.00300000	1.000	0.00300000	1.000
0.00600000	1.667	0.00600000	4.000
0.00900000	1.333	0.00900000	3.667
0.01200000	3.333	0.01200000	2.333
0.01500000	2.333	0.01500000	3.667
0.01800000	0.000	0.01800000	1.000
0.02100000	0.000	0.02100000	2.000
0.02400000	2.000	0.02400000	2.667
0.02700000	1.000	0.02700000	2.667
0.03000000	1.333	0.03000000	2.333
0.03300000	0.667	0.03300000	1.333
0.03600000	2.000	0.03600000	2.667
0.03900000	3.000	0.03900000	3.667
0.04200000	4.000	0.04200000	2.667
0.04500000	2.667	0.04500000	2.000
```

Пример содержимого файла:
https://drive.google.com/file/d/1n6FSVVMs8i7sM6utXUe7ajcnTt0PqSTR/view?usp=sharing

Задача 2

Создайте переменные с именами `zero`, `one`, `two`, `three`, `four`, `five`, `six`, `seven`, `eight`, `nine`. При этом должен работать код, описанный в примере работы программы.

Программа должна поддерживать следующие действия:

- `plus` — сложение,
- `minus` — вычитание,
- `times` — умножение.

Вы можете реализовывать любые классы и функции, проверяться будет только работоспособность кода, подобного тому, что написано в примере.

Пример работы программы:

```
five                                ⇒ 5

five.minus.three                    ⇒ 2

two.plus.two                        ⇒ 4

five.plus.two.times.four            ⇒ 28

# порядок операций - слева направо,

# независимо от порядка сложения и умножения

nine.times.nine.times.nine          ⇒ 729
```

Скрипт с задачей: https://colab.research.google.com/drive/1ctf_HJRUXhvAaiyHFb5Im3ZrKKOvCgXq?usp=sharing

Задача 1

содержимое модуля fret.py

```
class IntensityData:
  def __init__(self):
    self.time, self.donor, self.acceptor = [], [], []
  def read(self, filename):
    with open(filename,'r') as f:
      for line in f.readlines()[1:]:
        line_list = line.split()
        if len(line_list) == 4:
          try:
            time, acceptor, _, donor =  tuple(map(float,
line_list))
            self.time.append(time)
            self.donor.append(donor)
            self.acceptor.append(acceptor)
          except ValueError:
            pass
```

чтение данных из файла

```
from fret import IntensityData
data =
IntensityData().read("prox-pure.txt")
```

Полный скрипт: https://colab.research.google.com/drive/11idIbm2hwICATo-X_sF85-CVAyI6FKDJ?usp=sharing

Задача 2

Создайте переменные с именами `zero`, `one`, `two`, `three`, `four`, `five`, `six`, `seven`, `eight`, `nine`. При этом должен работать код, описанный в примере работы программы.

Программа должна поддерживать следующие действия:

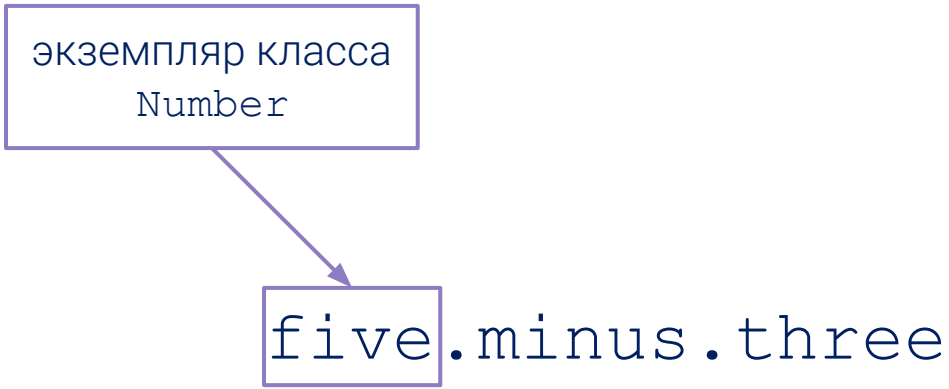
- `plus` — сложение,
- `minus` — вычитание,
- `times` — умножение.

Вы можете реализовывать любые классы и функции, проверяться будет только работоспособность кода, подобного тому, что написано в примере.

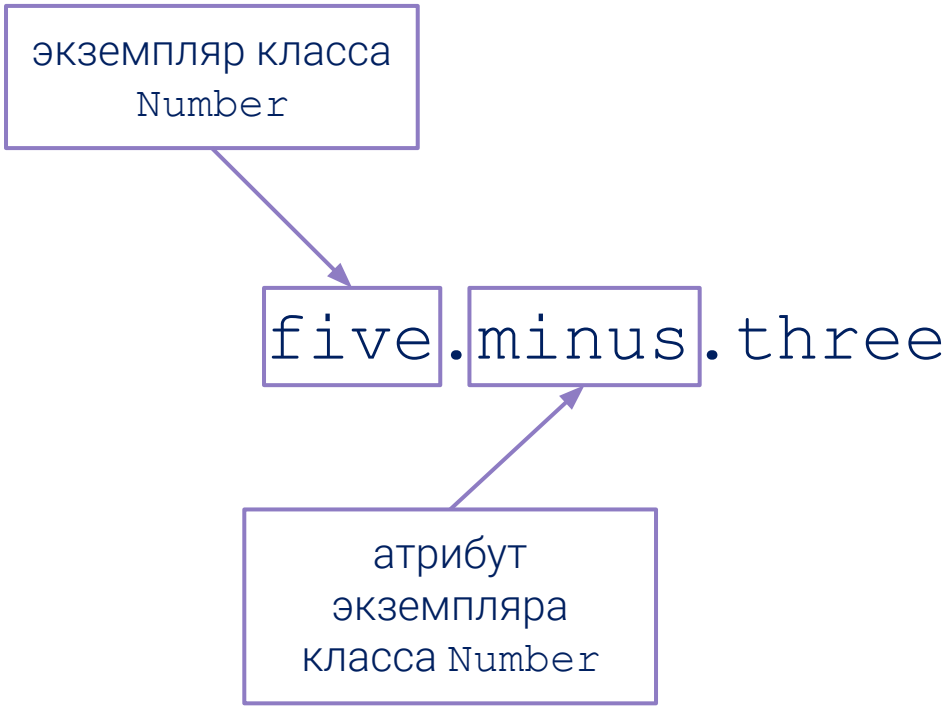
Пример работы программы:

```
five                                ⇒ 5

five.minus.three                    ⇒ 2

two.plus.two                        ⇒ 4

five.plus.two.times.four            ⇒ 28

# порядок операций - слева направо,

# независимо от порядка сложения и умножения

nine.times.nine.times.nine          ⇒ 729
```

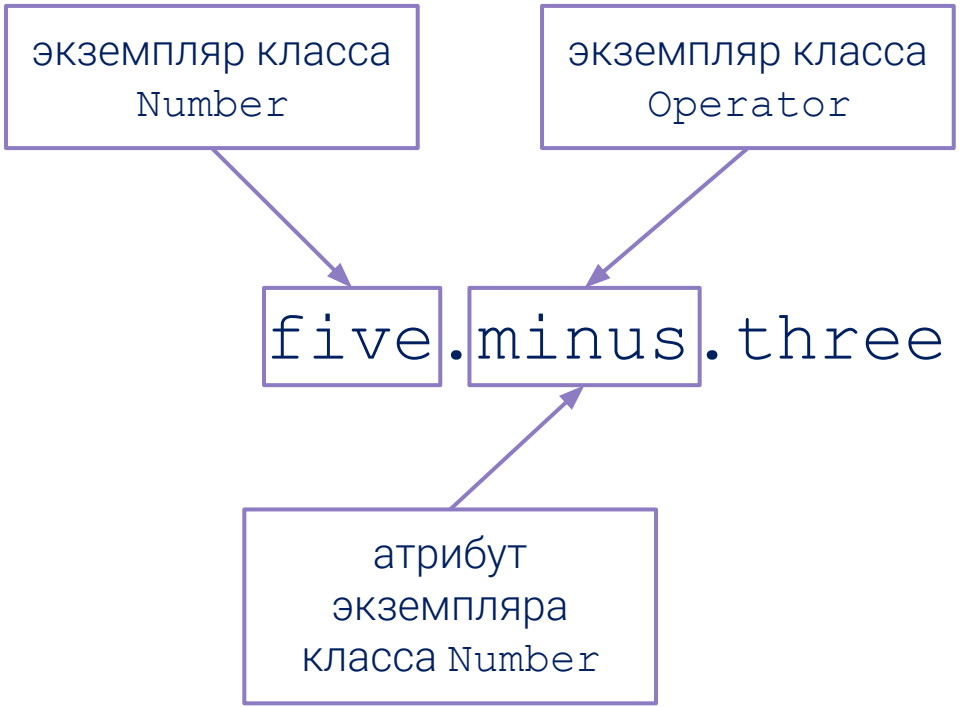
Задача 2



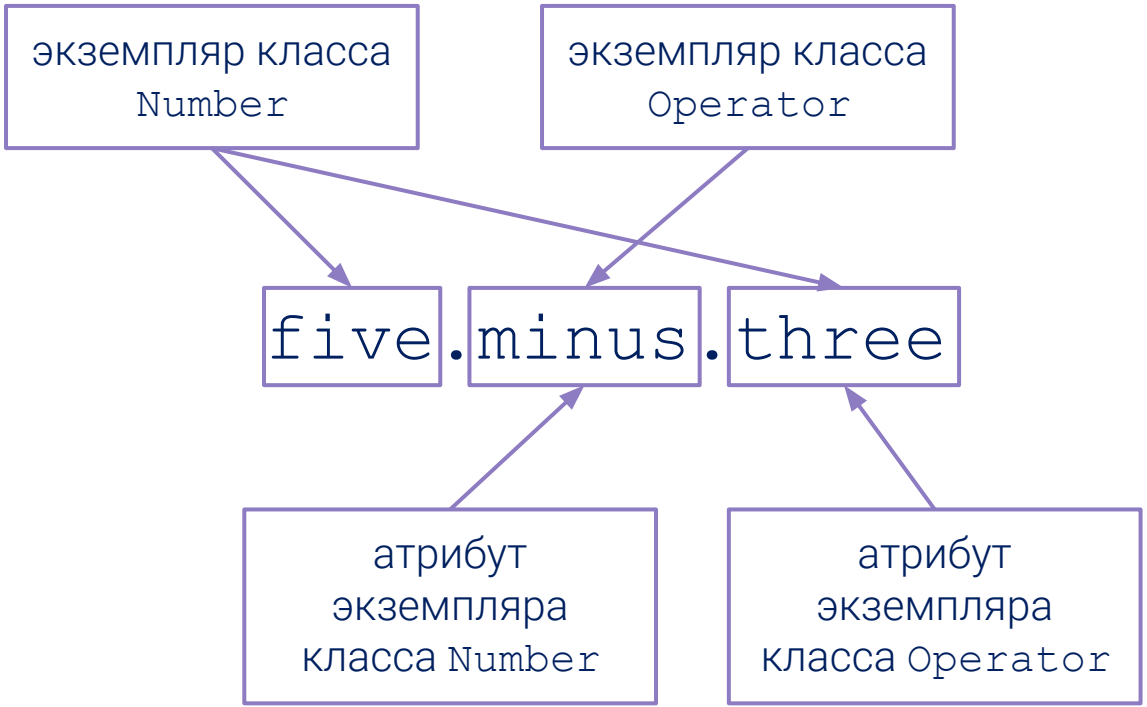
Задача 2


экземпляр класса
Number
five.minus.three
атрибут
экземпляра
класса Number

Задача 2


экземпляр класса Number
экземпляр класса Operator
five.minus.three
атрибут экземпляра класса Number

Задача 2


экземпляр класса Number
экземпляр класса Operator
five.minus.three
атрибут экземпляра класса Number
атрибут экземпляра класса Operator

Задача 2

наследуемся от `int`

```
class Number(int):
    ops = {
        'plus': lambda x, y: x + y,
        'minus': lambda x, y: x - y,
        'times': lambda x, y: x * y,
    }

    def __getattr__(self, attr):
        try:
            return Operator(self.ops[attr], self)
        except KeyError:
            raise AttributeError(f"int object has no attribute '{attr}'")
```

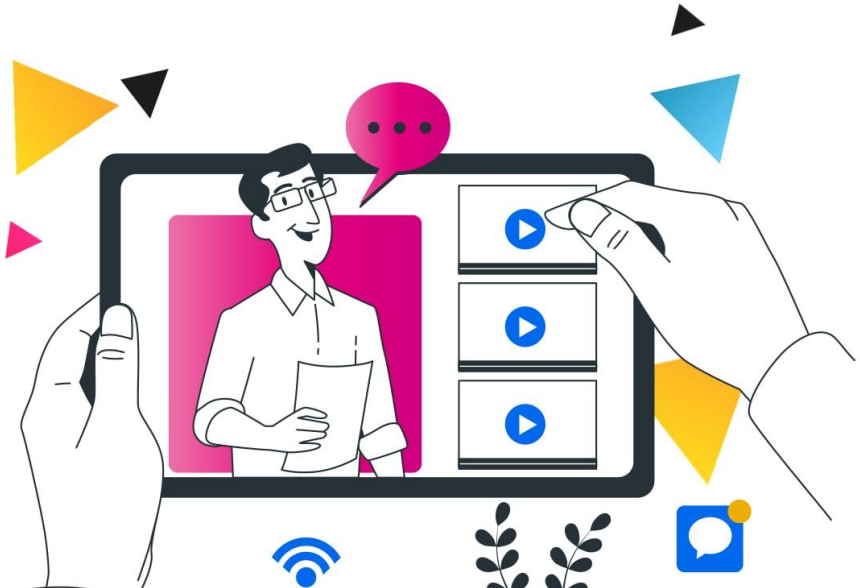
Полный скрипт: https://colab.research.google.com/drive/11idIbm2hwICATo-X_sF85-CVAyI6FKDJ?usp=sharing

Задача 2

```
class Operator:
    numbers = {
        'zero': 0, 'one': 1, 'two': 2, 'three': 3, 'four': 4,
        'five': 5, 'six': 6, 'seven': 7, 'eight': 8, 'nine': 9,
        }

    def __init__(self, op, n):
        self.operator = op
        self.n = n

    def __getattr__(self, attr):
        try:
            return Number(self.operator(self.n, self.numbers[attr]))
        except KeyError:
            raise ValueError(f'unknown number: {attr}')
```

Полный скрипт: https://colab.research.google.com/drive/11idIbm2hwICATo-X_sF85-CVAyI6FKDJ?usp=sharing

Ваши вопросы?



Итоги занятия

Итоги занятия

МФТИ

1. Рассмотрели принципы обработки исключений в Python
2. Рассмотрели, как устанавливать внешние библиотеки и создавать свои модули
3. Отработаем на практике специальные методы, создание собственных модулей и использование конструкции `try-except`

Ваши вопросы?