

**КАЛУЖСКИЙ ФИЛИАЛ
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО
ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ Н.Э. БАУМАНА
(национальный исследовательский университет)»**



Факультет «Информатика и управление»

Кафедра «Программное обеспечение ЭВМ, информационные технологии»

Высокоуровневое программирование

Лекция №12.

«Пользовательские модули в Python. Обработка исключительных ситуаций»

Калуга - 2020

Подключение модуля из стандартной библиотеки

```
1 import random
2 random.random()
```

0.04415253858714807

- `import module`
- `import module as`
- `from module import object`
- `from module import *`

- После импортирования модуля его название становится переменной, через которую можно получить доступ к атрибутам модуля. Например, можно обратиться к константе `e`, расположенной в модуле `math`:

```
1 import math
2 math.e
```

2.718281828459045

Использование псевдонимов

```
1 import math as m
2 m.e
```

2.718281828459045

- Теперь доступ ко всем атрибутам модуля **math** осуществляется только с помощью переменной **m**, а переменной **math** в этой программе уже не будет (если, конечно, вы после этого не напишете **import math**, тогда модуль будет доступен как под именем **m**, так и под именем **math**)

Инструкция from

- Подключить определенные атрибуты модуля можно с помощью инструкции from. Она имеет несколько форматов:

```
from <Название модуля> import <Атрибут 1> [ as <Псевдоним 1> ], [<Атрибу  
from <Название модуля> import *
```

- Первый формат позволяет подключить из модуля только указанные вами атрибуты. Для длинных имен также можно назначить псевдоним, указав его после ключевого слова as.

```
1 from random import random as rnd  
2 rnd()
```

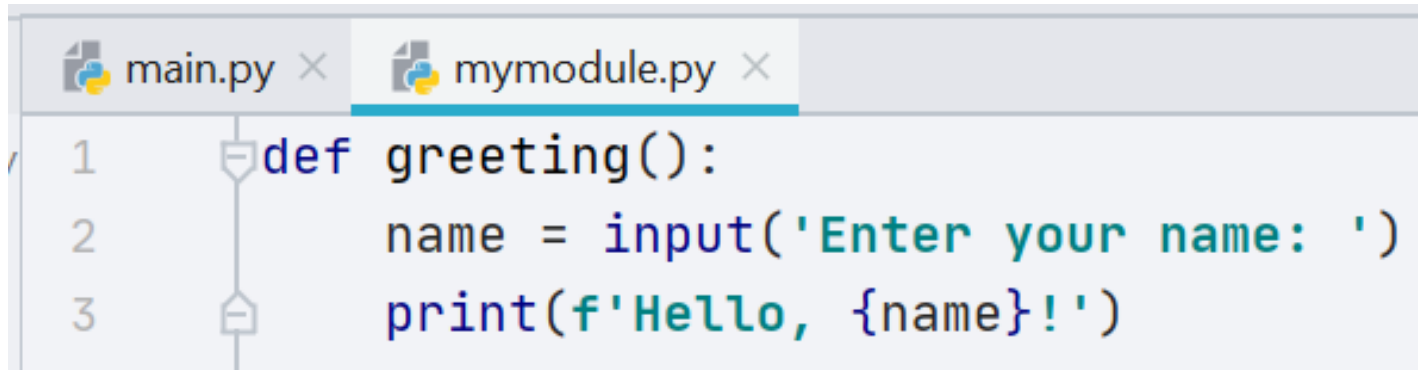
0.2142304392165213

Инструкция **from**

- Вторым формат инструкции **from** позволяет подключить все (точнее, почти все) переменные из модуля.
- Следует заметить, что не все атрибуты будут импортированы. Если в модуле определена переменная **__all__** (список атрибутов, которые могут быть подключены), то будут подключены только атрибуты из этого списка. Если переменная **__all__** не определена, то будут подключены все атрибуты, не начинающиеся с нижнего подчёркивания. Кроме того, необходимо учитывать, что импортирование всех атрибутов из модуля может нарушить пространство имен главной программы, так как переменные, имеющие одинаковые имена, будут перезаписаны.

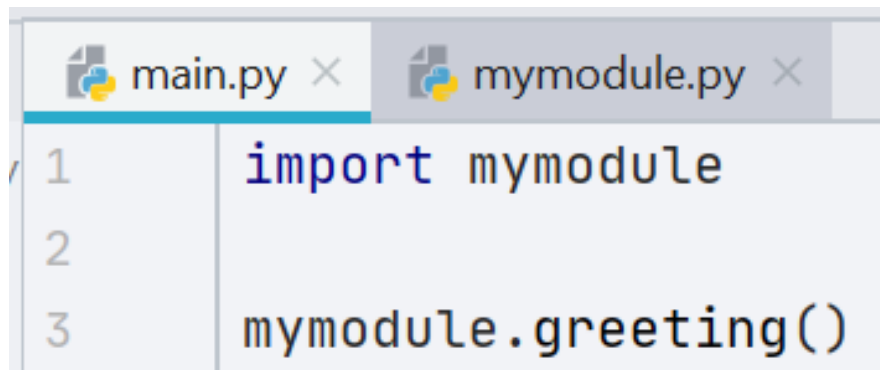
Создание пользовательского модуля

1. Создание файла *.py:



```
main.py x mymodule.py x
1 def greeting():
2     name = input('Enter your name: ')
3     print(f'Hello, {name}!')
```

2. В этой же папке создать другой файл, например, main.py:



```
main.py x mymodule.py x
1 import mymodule
2
3 mymodule.greeting()
```

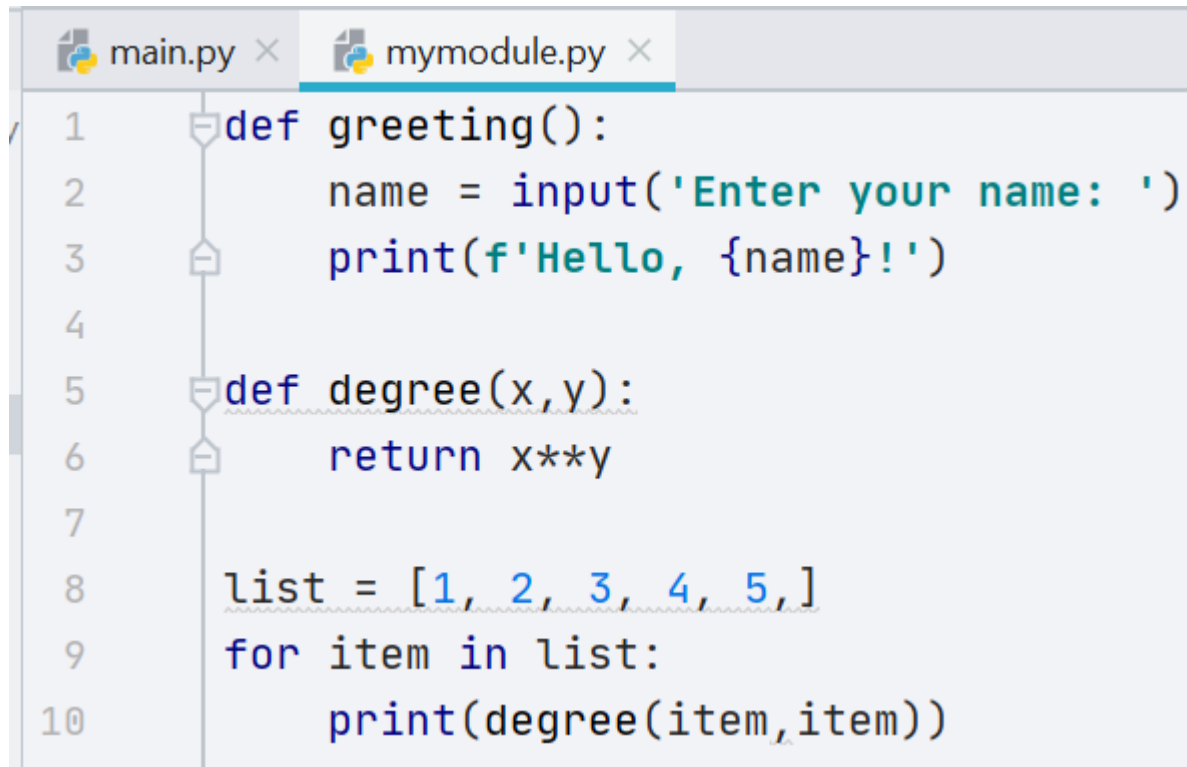
```
Enter your name: Helene
Hello, Helene!
```

Название модуля

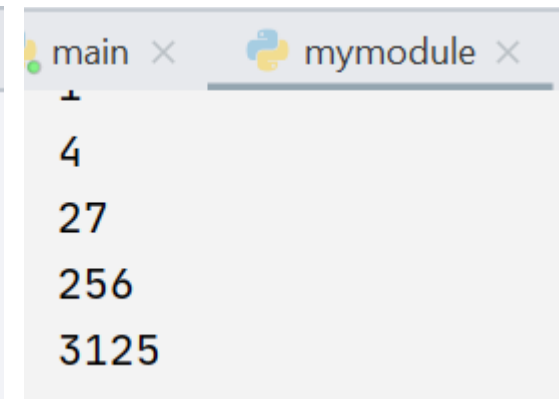
1. Модуль нельзя именовать также, как и ключевое слово.
2. Имена модулей нельзя начинать с цифры.
3. Не стоит называть модуль также, как какую-либо из встроенных функций:
 - это создаст большие неудобства при его последующем использовании.

Можно ли использовать модуль как самостоятельную программу?

- Можно. Однако надо помнить, что при импортировании модуля его код выполняется полностью, то есть, если программа что-то печатает, то при её импортировании это будет напечатано.



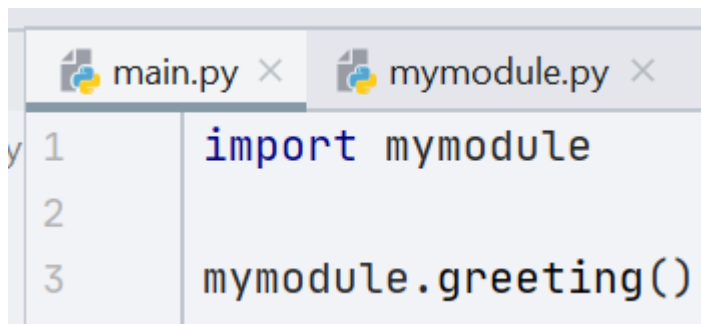
```
1 def greeting():
2     name = input('Enter your name: ')
3     print(f'Hello, {name}!')
4
5 def degree(x,y):
6     return x**y
7
8 list = [1, 2, 3, 4, 5,]
9 for item in list:
10     print(degree(item,item))
```



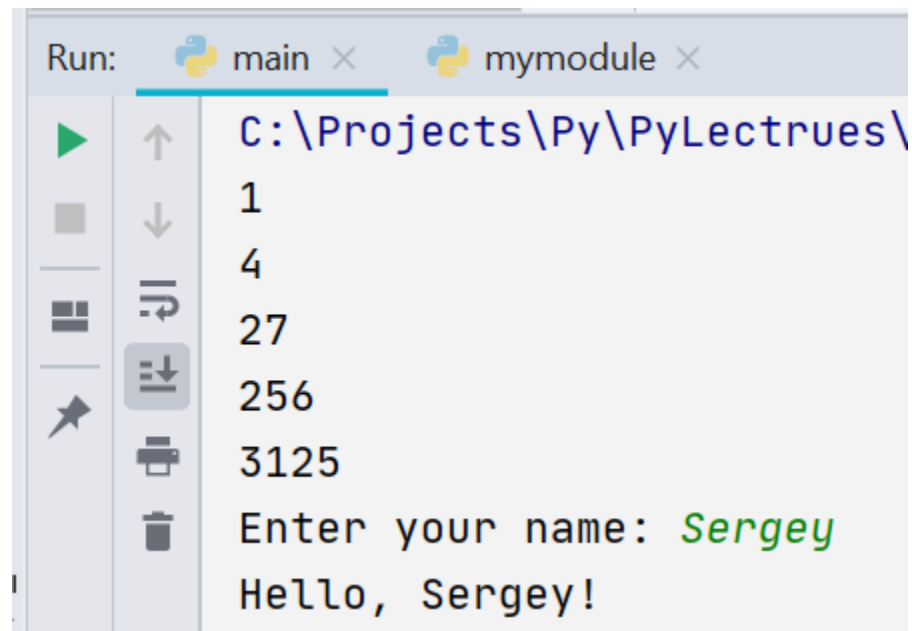
```
main × mymodule ×
+
4
27
256
3125
```


Можно ли использовать модуль как самостоятельную программу?

- Можно. Однако надо помнить, что при импортировании модуля его код выполняется полностью, то есть, если программа что-то печатает, то при её импортировании это будет напечатано.



```
1 import mymodule
2
3 mymodule.greeting()
```

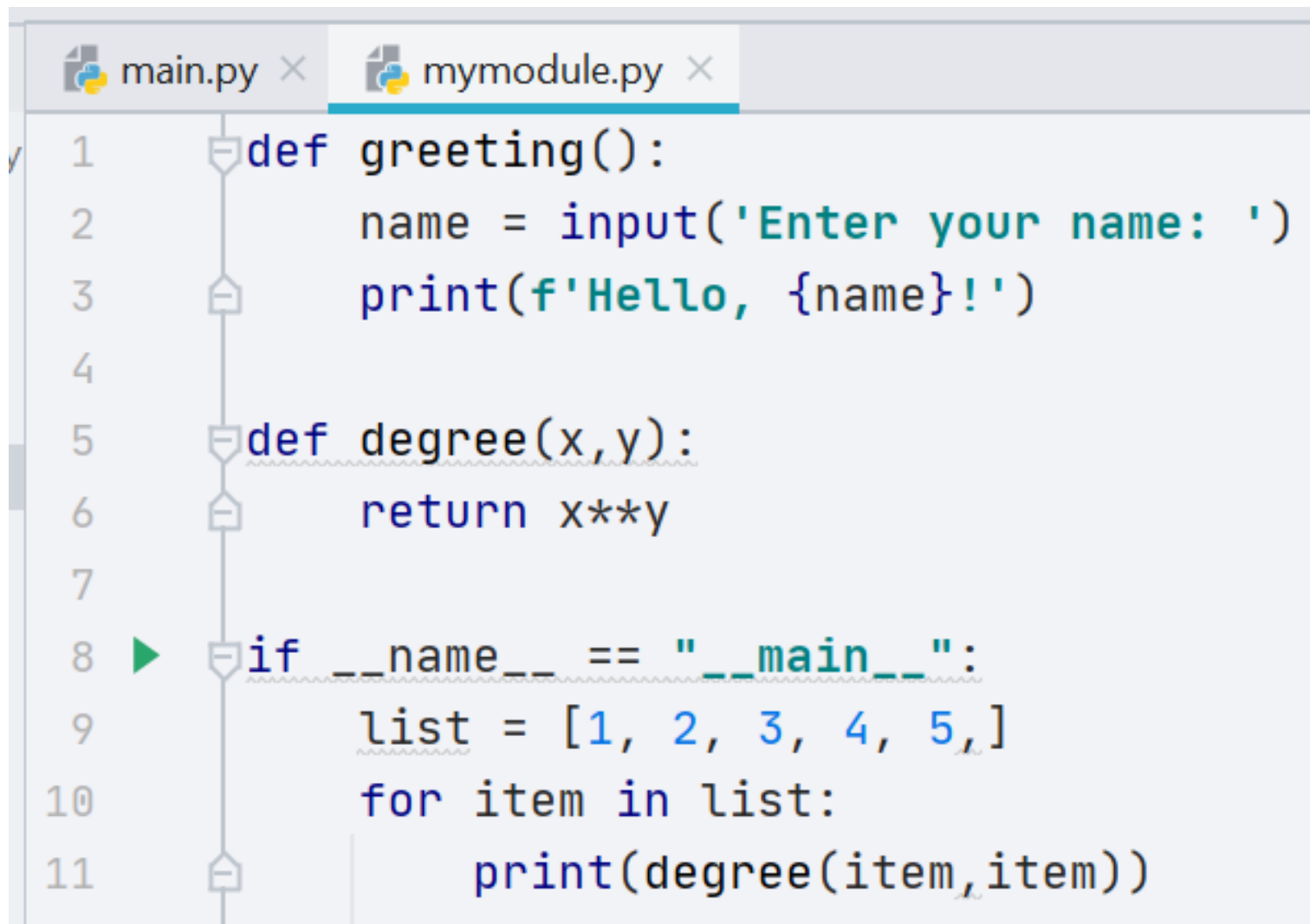


```
Run: C:\Projects\Py\PyLectrues\
1
4
27
256
3125
Enter your name: Sergey
Hello, Sergey!
```

Можно ли использовать модуль как самостоятельную программу?

- Этого можно избежать, если проверять, запущен ли скрипт как программа, или импортирован. Это можно сделать с помощью переменной `__name__`, которая определена в любой программе, и равна `"__main__"`, если скрипт запущен в качестве главной программы, и имя, если он импортирован. Например, `mymodule.py` может выглядеть вот так:

Можно ли использовать модуль как самостоятельную программу?



```
main.py x mymodule.py x
1 def greeting():
2     name = input('Enter your name: ')
3     print(f'Hello, {name}!')
4
5 def degree(x, y):
6     return x**y
7
8 if __name__ == "__main__":
9     list = [1, 2, 3, 4, 5,]
10    for item in list:
11        print(degree(item, item))
```

Что такое `__all__` в Python?

- `__all__` в Python – это список публичных объектов данного модуля.

```
.py × mymodule.py ×  
  
def greeting():  
    name = input('Enter your name: ')  
    print(f'Hello, {name}!')  
  
def degree(x, y):  
    return x**y  
  
__all__ = ["greeting"]
```

```
main.py × mymodule.py ×  
  
1 from mymodule import *  
2  
3 greeting()  
4 print(degree(2, 3))
```

Обработка исключительных ситуаций (ОИС)

- Виды ошибок:
 - синтаксические;
 - Исключения (exceptions).

```
1 2/0
```

```
-----  
ZeroDivisionError                                Traceback (most recent call last)  
<ipython-input-1-e8326a161779> in <module>  
----> 1 2/0
```

```
ZeroDivisionError: division by zero
```

```
1 'test' + 2
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-2-f4af361180a2> in <module>  
----> 1 'test' + 2
```

```
TypeError: can only concatenate str (not "int") to str
```

ОИС

- Для работы с исключениями используется конструкция **try/except**:

```
1 try:
2     2/0
3     'test' + 2
4 except ZeroDivisionError:
5     print('Ошибка деления на ноль!')
6 except TypeError:
7     print('Операция применена к объекту несоответствующего типа!')
8 except Exception:
9     print('Видимо, что-то случилось...')
10
11
```

Ошибка деления на ноль!

ОИС

- Для работы с исключениями используется конструкция **try/except**:

```
1 try:
2     2/0
3     'test' + 2
4 except Exception:
5     print('Видимо, что-то случилось...')
6 except ZeroDivisionError:
7     print('Ошибка деления на ноль!')
8 except TypeError:
9     print('Операция применена к объекту несоответствующего типа!')
10
11
12
```

Видимо, что-то случилось...

Когда в программе возникает исключение, она сразу завершает свою работу.

ОИС

Конструкция **try** работает таким образом:

- сначала выполняются выражения, которые записаны в блоке **try**;
- если при выполнении блока **try** не возникло никаких исключений, блок **except** пропускается, и выполняется дальнейший код;
- если во время выполнения блока **try** в каком-то месте возникло исключение, оставшаяся часть блока **try** пропускается;
- если в блоке **except** указано исключение, которое возникло, выполняется код в блоке **except**;
- если исключение, которое возникло, не указано в блоке **except**, выполнение программы прерывается и выдается ошибка.

ОИС

```
1 try:
2     print("Let's divide some numbers")
3     2/0
4     print('Cool!')
5 except ZeroDivisionError:
6     print("You can't divide by zero")
```

Let's divide some numbers
You can't divide by zero

```
1 try:
2     a = input("Введите первое число: ")
3     b = input("Введите второе число: ")
4     print("Результат: ", int(a)/int(b))
5 except ValueError:
6     print("Пожалуйста, вводите только числа")
7 except ZeroDivisionError:
8     print("На ноль делить нельзя")
```

Введите первое число: 1
Введите второе число: 2й
Пожалуйста, вводите только числа

try/except/else

В конструкции **try/except** есть опциональный блок **else**. Он выполняется в том случае, если не было исключения.

```
1  try:
2      a = input("Введите первое число: ")
3      b = input("Введите второе число: ")
4      result = int(a)/int(b)
5  except (ValueError, ZeroDivisionError):
6      print("Что-то пошло не так...")
7  else:
8      print("Результат в квадрате: ", result**2)
```

```
Введите первое число: 1
Введите второе число: 2
Результат в квадрате:  0.25
```

try/except/finally

- Блок **finally** - это еще один опциональный блок в конструкции **try**. Он выполняется всегда, независимо от того, было ли исключение или нет.
- Сюда ставятся действия, которые надо выполнить в любом случае. Например, это может быть закрытие файла.

try/except/finally

```
1  try:
2      a = input("Введите первое число: ")
3      b = input("Введите второе число: ")
4      result = int(a)/int(b)
5  except (ValueError, ZeroDivisionError):
6      print("Что-то пошло не так...")
7  else:
8      print("Результат в квадрате: ", result**2)
9  finally:
10     print("Эта строка будет выведена всегда.")
```

Введите первое число: 1

Введите второе число: 0

Что-то пошло не так...

Эта строка будет выведена всегда.

Когда использовать исключения?

```
1 while True:
2     a = input("Введите число: ")
3     b = input("Введите второе число: ")
4     try:
5         result = int(a)/int(b)
6     except ValueError:
7         print("Поддерживаются только числа")
8     except ZeroDivisionError:
9         print("На ноль делить нельзя")
10    else:
11        print(result)
12        break
```

```
Введите число: 1
Введите второе число: 0
На ноль делить нельзя
Введите число: 1
Введите второе число: 2й
Поддерживаются только числа
Введите число: 1
Введите второе число: 3
0.3333333333333333
```

Когда использовать исключения?

```
1 while True:
2     a = input("Введите число: ")
3     b = input("Введите второе число: ")
4     if a.isdigit() and b.isdigit():
5         if int(b) == 0:
6             print("На ноль делить нельзя")
7         else:
8             print(int(a)/int(b))
9             break
10    else:
11        print("Поддерживаются только числа")
```

```
Введите число: 1
Введите второе число: 0
На ноль делить нельзя
Введите число: 1
Введите второе число: 2й
Поддерживаются только числа
Введите число: 1
Введите второе число: 3
0.3333333333333333
```

Когда использовать исключения?

```
1 while True:
2     a = input("Введите число: ")
3     b = input("Введите второе число: ")
4     try:
5         result = int(a)/int(b)
6     except ValueError:
7         print("Поддерживаются только числа")
8     except ZeroDivisionError:
9         print("На ноль делить нельзя")
10    else:
11        print(result)
12        break
```

```
1 while True:
2     a = input("Введите число: ")
3     b = input("Введите второе число: ")
4     if a.isdigit() and b.isdigit():
5         if int(b) == 0:
6             print("На ноль делить нельзя")
7         else:
8             print(int(a)/int(b))
9             break
10    else:
11        print("Поддерживаются только числа")
```

Когда использовать исключения?

- Решайте сами!
- Важно в каждой конкретной ситуации оценивать, какой вариант кода более понятный, компактный и универсальный — с исключениями или без.

Задачи для самостоятельного решения

- Даны n предложений. Определите, сколько из них содержат хотя бы одну цифру.
- Дана строка s и символ k . Реализуйте функцию, рисующую рамку из символа k вокруг данной строки, например:

```
*****  
*Текст в рамке*  
*****
```
- Для введенного предложения выведите статистику *символ=количество*. Регистр букв не учитывается.
- Дата характеризуется тремя натуральными числами: день, месяц и год. Учитывая, что год может быть високосным, реализуйте две функции, которые определяют вчерашнюю и завтрашнюю дату.

Задачи для самостоятельного решения

- Напишите функцию, которая принимает неограниченное количество числовых аргументов и возвращает кортеж из двух списков:
 - отрицательных значений (отсортирован по убыванию);
 - неотрицательных значений (отсортирован по возрастанию).
- Составьте две функции для возведения числа в степень: один из вариантов реализуйте в рекурсивном стиле.
- Дано натуральное число. Напишите рекурсивные функции для определения:
 - суммы цифр числа;
 - количества цифр в числе.