


# Section VIII: Работа с файлами

# Section VIII: Работа с файлами

## Открытие файла

Путь (относительный  
или абсолютный)



```
1 f = open('text_file.txt', 'r')
2 print(type(f))
```

```
<class '_io.TextIOWrapper'>
```

‘r’ – модификатор: режим  
чтения

# Section VIII: Работа с файлами

## Режимы чтения

- r – открытие на чтение (является значением по умолчанию).
- w – открытие на запись, содержимое файла удаляется, если файла не существует, создается новый.
- x – открытие на запись, если файла не существует, иначе исключение.
- a – открытие на дозапись, информация добавляется в конец файла.
- b – открытие в двоичном режиме.
- t – открытие в текстовом режиме (является значением по умолчанию).
- + – открытие на чтение и запись

```
1 f = open('text_file.txt', 'rb')
```

# Section VIII: Работа с файлами

## Чтение из файлов

```
1  f = open('text_file.txt', 'r')
2  text = f.read()
3  print(text)
```

```
Hello there!
We are using Python.
And that is pretty awesome :)
```

```
text: 'Hello there!\nWe are using Python.\nAnd that is pretty awesome :)\n'
```

# Section VIII: Работа с файлами

## Чтение из файлов

```
1  f = open('text_file.txt', 'r')
2  text = f.read(2)
3  print(text)
4
5  text = f.read()
6  print(text)
```

```
He
llo there!
We are using Python.
And that is pretty awesome :)
```

# Section VIII: Работа с файлами

## Чтение из файлов

```
1 f = open('text_file.txt', 'r')
2 for line in f:
3     text = line
4     print(text)
```

Hello there!

We are using Python.

And that is pretty awesome :)

Пустые  
строки??

# Section VIII: Работа с файлами

## Чтение из файлов

```
1 f = open('text_file.txt', 'r')
2 text = f.read()
3 print(text)
```

```
text: 'Hello there!\nWe are using Python.\nAnd that is pretty awesome :)\n'
```

```
1 f = open('text_file.txt', 'r')
2 text = f.readlines()
3 print(text)
```

```
['Hello there!\n', 'We are using Python.\n', 'And that is pretty awesome :)\n']
```

```
1 f = open('text_file.txt', 'r')
2 text = f.readline()
```

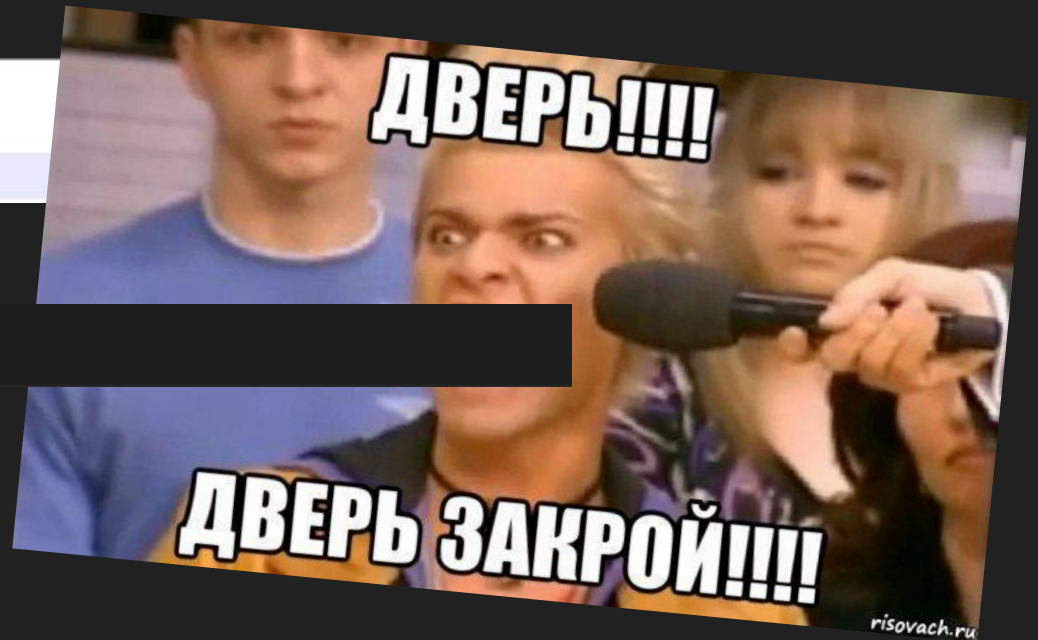
# Section VIII: Работа с файлами

## Запись в файл

```
1 f = open('text_file.txt', 'w')
2 text = 'We are ready\nto work with files!'
3 f.write(text + '\n')
```

```
1 We are ready
2 to work with files!
3
```

```
5 f.close()
```





# Section VIII: Работа с файлами

With ... as – менеджеры контекста

Синтаксис:

```
"with" expression ["as" target] ("," expression ["as" target])* ":"  
suite
```

```
1  with open('newfile.txt', 'w', encoding='utf-8') as g:  
2      d = int(input())  
3      print('1 / {} = {} '.format(d, 1 / d), file=g)  
  
5  g.close()
```

# Section VIII: Работа с файлами

## With ... as – менеджеры контекста

```
"with" expression ["as" target] ("," expression ["as" target])* ":"  
suite
```

1. Выполняется выражение в конструкции with ... as.
2. Загружается специальный метод `__exit__` для дальнейшего использования.
3. Выполняется метод `__enter__`. Если конструкция with включает в себя слово as, то возвращаемое методом `__enter__` значение записывается в переменную.
4. Выполняется suite.
5. Вызывается метод `__exit__`, причём неважно, выполнилось ли suite или произошло исключение. В этот метод передаются параметры исключения, если оно произошло, или во всех аргументах значение None, если исключения не было.

# Section VIII: Работа с файлами

## Названия файлов и пути

```
1  f = open('text_file.txt', 'r')

3  import os
4  cwd = os.getcwd()  # current working directory
5  print(cwd)
```

C:\Users\Nikita Nikolaev\Desktop

```
1  relative_path = 'python\\text_file.txt'
2  f = open(relative_path, 'r')
3
4  import os
5  absolute_path = os.path.abspath(relative_path)
6  print(absolute_path)
```

C:\Users\Nikita Nikolaev\Desktop\python\text\_file.txt

# Section VIII: Работа с файлами

## Названия файлов и пути

```
1  import os
2  relative_path = 'python\\text_file.txt'
4  print(os.path.exists(relative_path))
```

True

```
5  print(os.path.isdir(relative_path))
```

False

```
6  print(os.listdir('python'))
```

```
['another_text_file.txt', 'secret', 'text_file.txt']
```

# Section VIII: Работа с файлами

`walk` – рекурсивный обход директории

```
1  import os
2
3  for root, dirs, files in os.walk(directory):
4      for filename in files:
5          full_path = os.path.join(root, filename)
6          name, extension = os.path.splitext(filename)
7          ...
```

# Section VIII: Работа с файлами

`try ... except` – обработка исключений

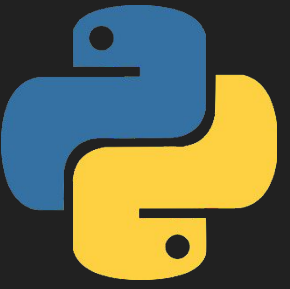
```
1  relative_path = 'bad_file.asdf'
2  f = open(relative_path, 'r')
```

`FileNotFoundError: [Errno 2] No such file or directory: 'bad_file.asdf'`

```
1  relative_path = 'python\\text_file.txt'
2  f = open(relative_path, 'w')
```

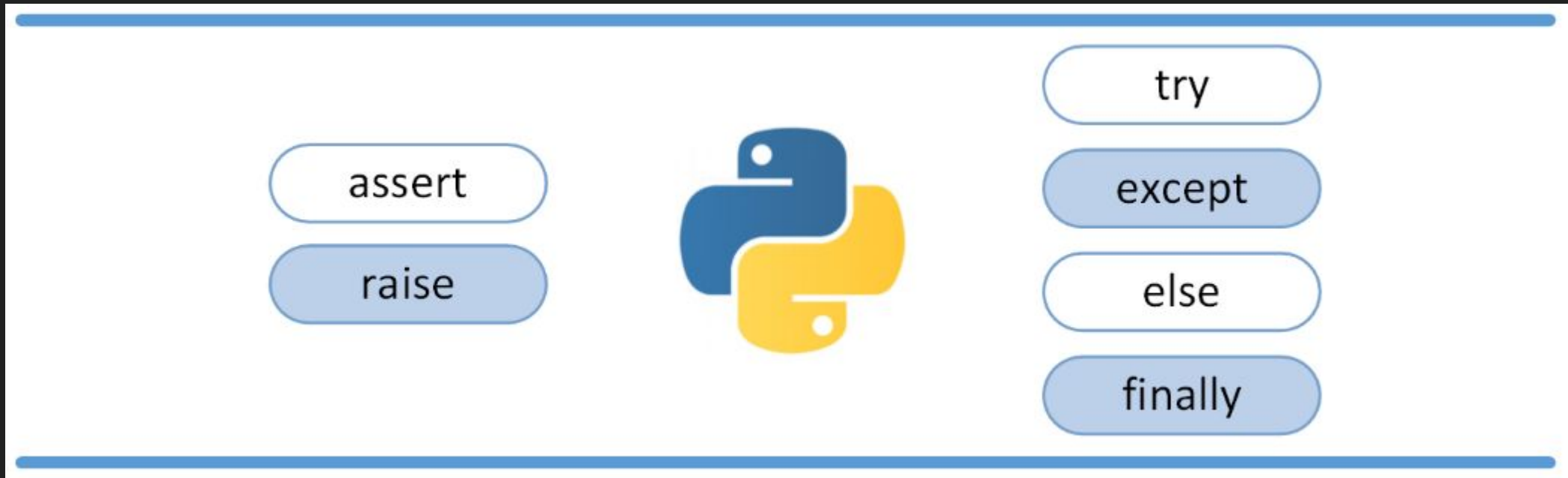
`PermissionError: [Errno 13] Permission denied: 'python\\text_file.txt'`

```
1  try:
2      relative_path = 'python\\text_file.txt'
3      f = open(relative_path, 'w')
4  except:
5      print('Unable to open file {}'.format(relative_path))
```



# Section IX: Обработка исключений (exceptions)

# Section IX: Обработка исключений





# Section IX: Обработка исключений

## Exceptions vs. Syntax Errors

```
>>> print( 0 / 0 ))  
File "<stdin>", line 1  
    print( 0 / 0 ))  
            ^  
SyntaxError: invalid syntax
```

```
>>> print( 0 / 0)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ZeroDivisionError: integer division or modulo by zero
```

# Section IX: Обработка исключений

## Исключения

- **BaseException** - базовое исключение, от которого берут начало все остальные.
  - **SystemExit** - исключение, порождаемое функцией `sys.exit` при выходе из программы.
  - **KeyboardInterrupt** - порождается при прерывании программы пользователем (обычно сочетанием клавиш Ctrl+C).
  - **GeneratorExit** - порождается при вызове метода `close` объекта `generator`.
  - **Exception** - а вот тут уже заканчиваются полностью системные исключения (которые лучше не трогать) и начинаются обыкновенные, с которыми можно работать.
    - **StopIteration** - порождается встроенной функцией `next`, если в итераторе больше нет элементов.
    - **ArithmeticError** - арифметическая ошибка.
      - **FloatingPointError** - порождается при неудачном выполнении операции с плавающей запятой. На практике встречается нечасто.
      - **OverflowError** - возникает, когда результат арифметической операции слишком велик для представления. Не появляется при обычной работе с целыми числами (так как python поддерживает длинные числа), но может возникать в некоторых других случаях.
      - **ZeroDivisionError** - деление на ноль.

<https://docs.python.org/3/library/exceptions.html>

...

# Section IX: Обработка исключений

## raise – ВЫЗОВ ИСКЛЮЧЕНИЯ

Use raise to force an exception:



```
x = 10
if x > 5:
    raise Exception('x should not exceed 5. The value of x was: {}'.format(x))
```

Traceback (most recent call last):

File "<input>", line 4, in <module>

Exception: x should not exceed 5. The value of x was: 10

# Section IX: Обработка исключений

`assert` – проверка на наличие ошибок. `AssertionError`

Assert that a condition is met:

`assert:`

{

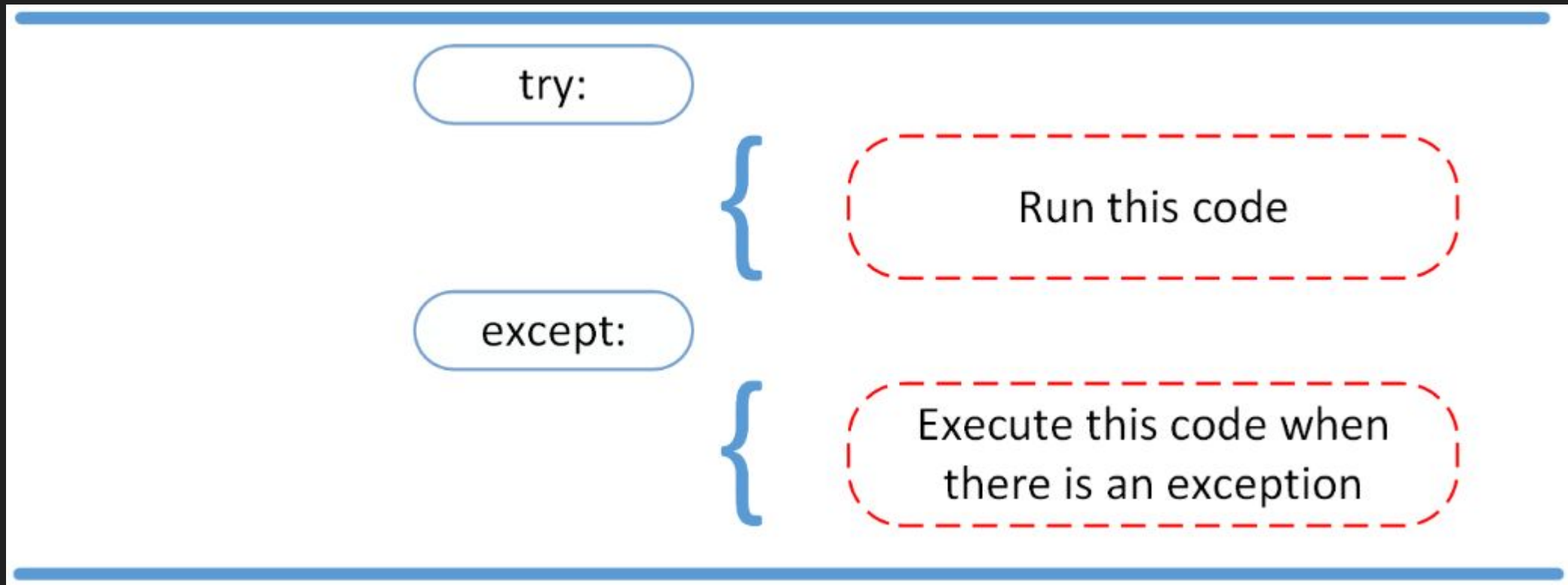
Test if condition is True

```
import sys
assert ('linux' in sys.platform), "This code runs on Linux only."
```

```
Traceback (most recent call last):
  File "<input>", line 2, in <module>
AssertionError: This code runs on Linux only.
```

# Section IX: Обработка исключений

`try ... except` – обработка исключений



# Section IX: Обработка исключений

`try ... except` – обработка исключений

```
def linux_interaction():  
    assert ('linux' in sys.platform), "Function can only run on Linux systems."  
    print('Doing something.')
```

```
try:  
    linux_interaction()  
except:  
    print('Linux function was not executed')
```

Shell

```
Linux function was not executed
```

# Section IX: Обработка исключений

`try ... except` – отслеживание исключений

```
def linux_interaction():  
    assert ('linux' in sys.platform), "Function can only run on Linux systems."  
    print('Doing something.')
```

```
try:  
    linux_interaction()  
except AssertionError as error:  
    print(error)  
    print('The linux_interaction() function was not executed')
```

## Shell

```
Function can only run on Linux systems.  
The linux_interaction() function was not executed
```



# Section IX: Обработка исключений

`try ... except` – отслеживание исключений

```
def linux_interaction():  
    assert ('linux' in sys.platform), "Function can only run on Linux systems."  
    print('Doing something.')
```

```
try:  
    linux_interaction()  
except AttributeError as error:  
    print(error)  
    print('The linux_interaction() function was not executed')
```

```
Traceback (most recent call last):  
  File "<input>", line 2, in <module>  
AssertionError: Function can only run on Linux systems.
```



# Section IX: Обработка исключений

`try ... except` – обработка файлов

```
try:
    with open('file.log') as file:
        read_data = file.read()
except:
    print('Could not open file.log')
```

Shell

```
Could not open file.log
```

```
try:
    with open('file.log') as file:
        read_data = file.read()
except FileNotFoundError as fnf_error:
    print(fnf_error)
```

# Section IX: Обработка исключений

`try ... except` – обработка нескольких исключений

```
try:
    linux_interaction()
    with open('file.log') as file:
        read_data = file.read()
except FileNotFoundError as fnf_error:
    print(fnf_error)
except AssertionError as error:
    print(error)
    print('Linux linux_interaction() function was not executed')
```

Какое исключение будет  
отловлено?

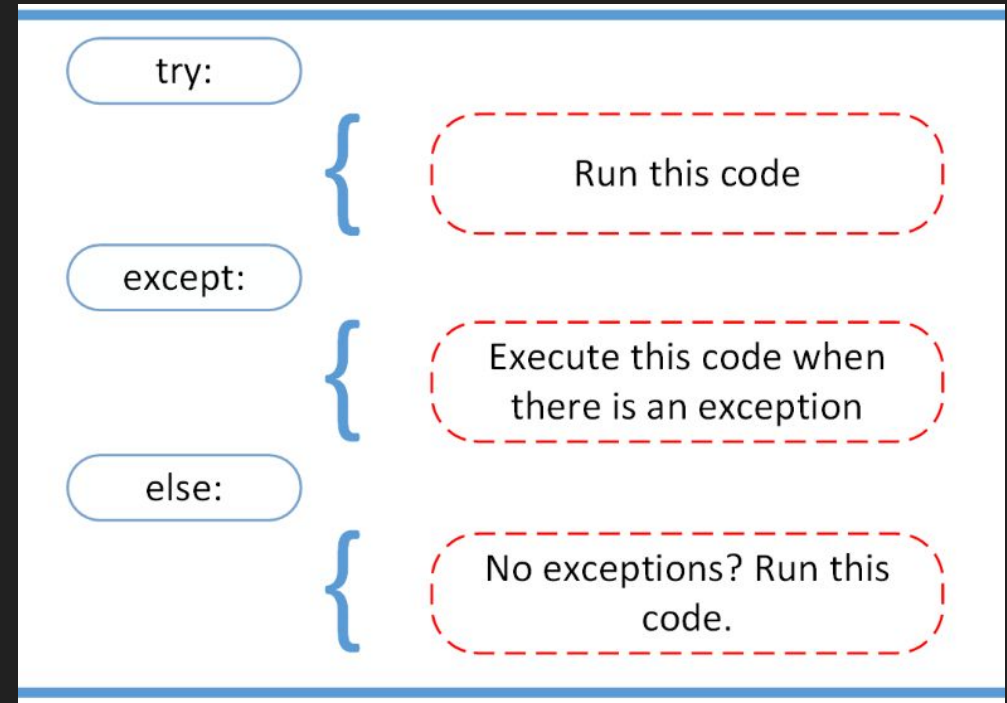
# Section IX: Обработка исключений

## Волшебное слово `else`

```
try:
    linux_interaction()
except AssertionError as error:
    print(error)
else:
    try:
        with open('file.log') as file:
            read_data = file.read()
    except FileNotFoundError as fnf_error:
        print(fnf_error)
```

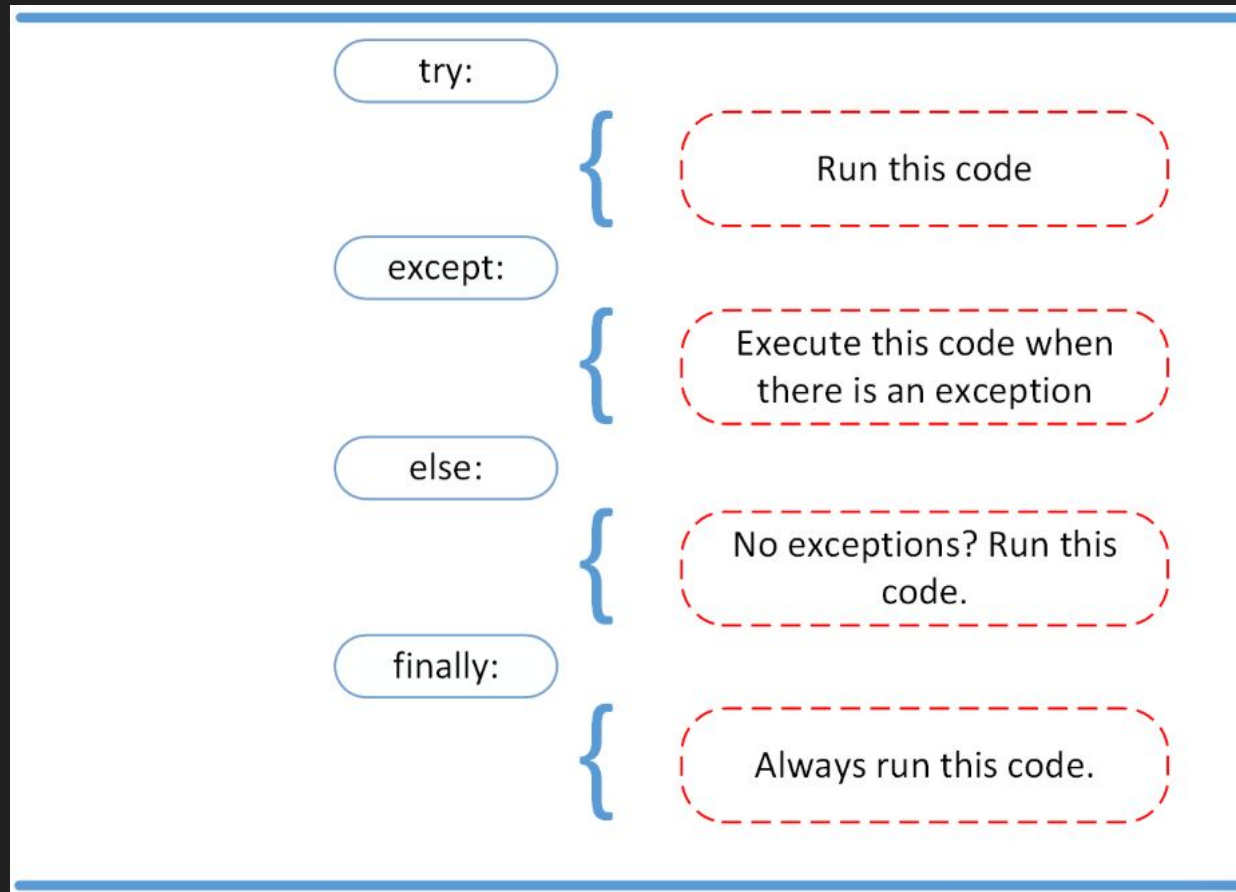
### Shell

```
Doing something.
[Errno 2] No such file or directory: 'file.log'
```



# Section IX: Обработка исключений

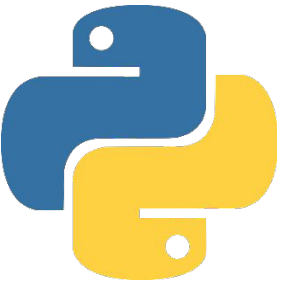
## Волшебное слово `finally`



# Section IX: Обработка исключений

## Волшебное слово `finally`

```
try:
    linux_interaction()
except AssertionError as error:
    print(error)
else:
    try:
        with open('file.log') as file:
            read_data = file.read()
    except FileNotFoundError as fnf_error:
        print(fnf_error)
finally:
    print('Cleaning up, irrespective of any exceptions.')
```



# Section: Детали с прошлых лекций

# Section: Детали с прошлых лекций

Вопрос: зачем нужны кортежи, если есть списки?

1. Защита от изменений, как хороших, так и

плохих

2. Меньший размер

```
>>> a = (1, 2, 3, 4, 5, 6)
>>> b = [1, 2, 3, 4, 5, 6]
>>> a.__sizeof__()
36
>>> b.__sizeof__()
44
```

3. Возможность использовать кортежи в качестве ключей словаря

```
>>> d = {(1, 1, 1) : 1}
>>> d
{(1, 1, 1): 1}
>>> d = {[1, 1, 1] : 1}
Traceback (most recent call last):
  File "", line 1, in
    d = {[1, 1, 1] : 1}
TypeError: unhashable type: 'list'
```

# Section: Детали с прошлых лекций

Задача: найти самый часто встречающийся элемент из списка

Задача стандартная, Google нам в помощь 😊

<https://www.geeksforgeeks.org/python-find-most-frequent-element-in-a-list/>



# Section: Детали с прошлых лекций

## Вопросы по множествам

- Как инициализировать пустое множество?
- А с помощью конструктора?
- А что с порядком элементов?
- Где еще такой порядок?
- frozenset?

```
>>> a = set('qwerty')
>>> b = frozenset('qwerty')
>>> a == b
True
>>> True
True
>>> type(a - b)
<class 'set'>
>>> type(a | b)
<class 'set'>
>>> a.add(1)
>>> b.add(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
```

# Section: Детали с прошлых лекций

Вопрос: разница между \*args и \*\*kwargs

```
>>> def func(*args):  
...     return args  
...  
>>> func(1, 2, 3, 'abc')  
(1, 2, 3, 'abc')  
>>> func()  
(  
>>> func(1)  
(1,)
```

```
>>> def func(**kwargs):  
...     return kwargs  
...  
>>> func(a=1, b=2, c=3)  
{'a': 1, 'c': 3, 'b': 2}  
>>> func()  
{  
>>> func(a='python')  
{'a': 'python'}
```

# Вопросы?