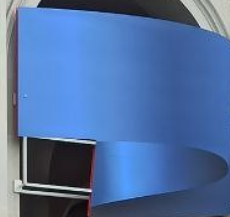


# Обработка ИСКЛЮЧЕНИЯ

try.. except..





# Обработка исключений в Python

Программа, написанная на языке Python, останавливается сразу как обнаружит ошибку. Ошибки могут быть:

**Синтаксические ошибки** — возникают, когда написанное выражение не соответствует правилам языка (например, написана лишняя скобка);

**Логические ошибки** — это ошибки, когда синтаксис действительно правильный, но логика не та, какую вы предполагали. Программа работает успешно, но даёт неверные результаты.

**Исключения** — возникают во время выполнения программы (например, при делении на ноль).








# Перехват ошибок во время выполнения

Не всегда при написании программы можно сказать возникнет или нет в данном месте исключение. Чтобы приложение продолжило работу при возникновении проблем, такие ошибки нужно перехватывать и обрабатывать с помощью ловуши **try/except**.



## В каких случаях нужно предусматривать обработку ошибок ?

- **Работа с файлами** ( нет прав доступа , диск переполнен и т.д)
- **Работа с СУБД** ( сервер не доступен, ошибка выполнения запроса и т.д)
- **Работа с сетью** (сеть недоступна, ошибка соединения, обработка кода возврата и т.д)
- **Работа с различными библиотеками** которые бросают exception в случае обнаружения ошибки.



## Как устроен механизм исключений ?

В Python есть встроенные исключения, которые появляются после того как приложение находит ошибку. В этом случае текущий процесс временно приостанавливается и передает ошибку на уровень вверх до тех пор, пока она не будет обработана. В случае когда ошибка не обрабатывается, программа прекратит свою работу и в консоли мы увидим Traceback с подробным описанием ошибки.




# Иерархия классов исключений Python

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

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
    +-- StopIteration
    +-- StopAsyncIteration
    +-- ArithmeticError
        |   +-- FloatingPointError
        |   +-- OverflowError
        |   +-- ZeroDivisionError
    +-- AssertionError
    +-- AttributeError
    +-- BufferError
    +-- EOFError
    +-- ImportError
        |   +-- ModuleNotFoundError
    +-- LookupError
        |   +-- IndexError
        |   +-- KeyError
    +-- MemoryError
    +-- NameError
        |   +-- UnboundLocalError
    +-- OSError
```





**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

**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.

**ZeroDivisionError** – деление на ноль.



# Генерация исключения

Напишем код, который будет создавать исключительную ситуацию. К примеру, попробуем поделить число на 0:

```
>>> print(1 / 0)
```

В командной оболочке получим следующее:

```
Traceback (most recent call last):
```

```
  File "", line 1, in
```

```
ZeroDivisionError: division by zero
```

Разберём это сообщение подробнее:

Интерпретатор нам сообщает о том, что он поймал исключение и напечатал информацию: `Traceback (most recent call last)`.

Далее имя файла `File ""`. Имя пустое, потому что этот код был запущен в интерактивном интерпретаторе, строка в файле `line 1`;

Название исключения **`ZeroDivisionError`** и краткое описание исключения `division by zero`.

При выбросе исключения программа закрывается и не выполняет код, который следует за строкой, в которой произошло исключение!

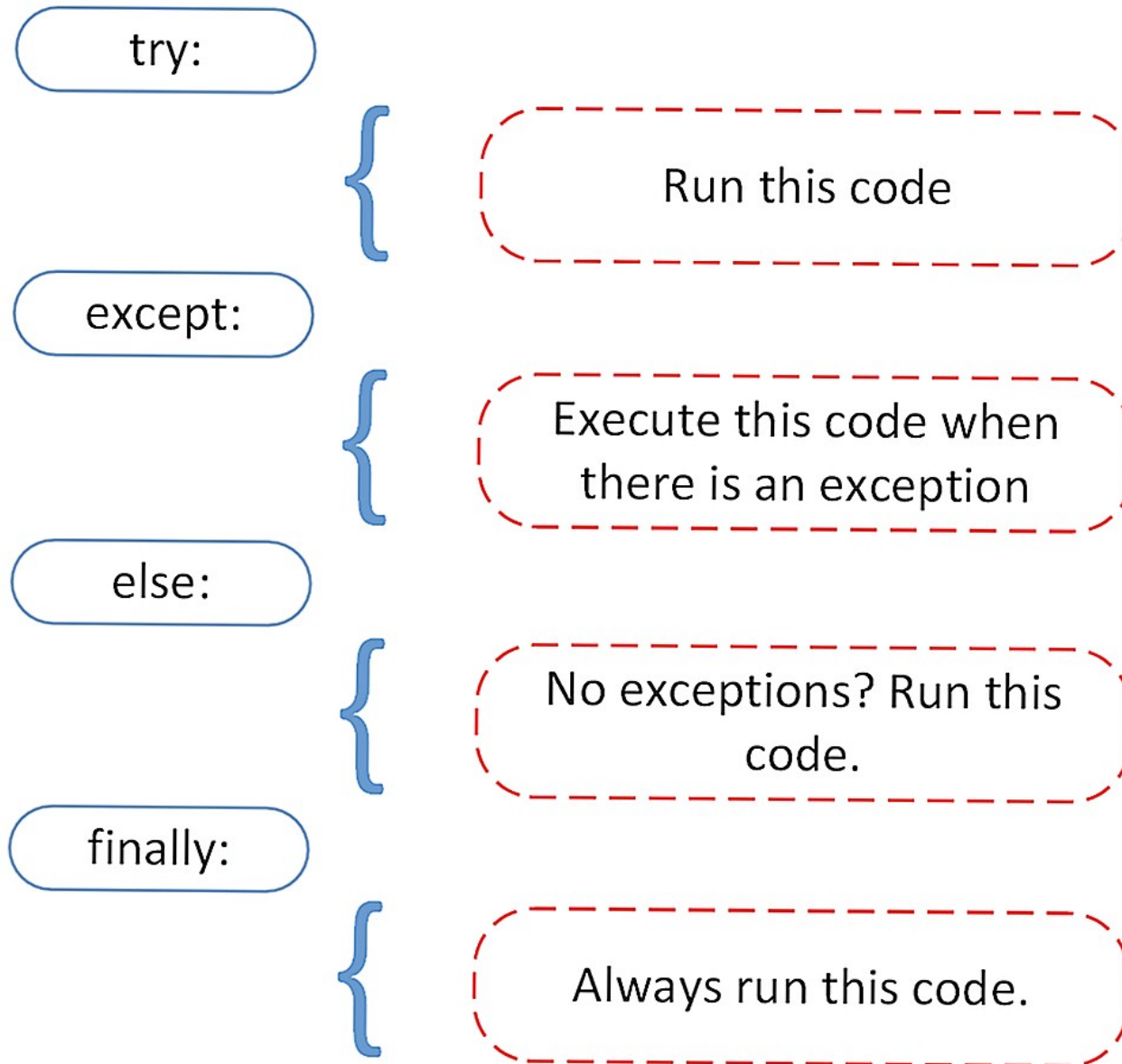
Windows

A fatal exception 0E has occurred at 0028:C0011E36 in UXD VMM(01) + 00010E36. The current application will be terminated.

- \* Press any key to terminate the current application.
- \* Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all your applications.

Press any key to continue \_

# Синтаксис конструкции





# Пример

```
try:
    a = 1/0

except ZeroDivisionError:
    print("Возникло исключение:ошибка деления на ноль! ")
    a = 0

else:
    print("Ветка else вызывается если не возникло
исключения")

finally:
    print("Аккуратно обрабатываем ошибку и идем дальше...")
```





## Ловушка для двух исключений

```
from calc import div
```

```
try:
```

```
    result = div(100, 0)
```

```
    print("Расчёт проведён успешно")
```

```
except (ZeroDivisionError, KeyError) as e:
```

```
    print("Ошибка деления или ошибка обращения по  
        ключу. Вот она:", e)
```

```
print(result)
```



# Несколько ловушек

```
from calc import div
try:
    result = div(100, 0)
    print("Расчёт проведён успешно")
except ZeroDivisionError as e:
    print("Ошибка деления произошла", e)
except KeyError as e:
    print("Ошибка обращения по ключу произошла:", e)

print(result)
```



# Перехват ошибки чтения

```
try:  
    file = open('ok123.txt', 'r')  
except FileNotFoundError as e:  
    print(e)
```

```
> [Errno 2] No such file or directory: 'ok123.txt'
```



# Порядок следования обработки

```
try:  
    file = open('ok123.txt', 'r')  
except Exception as e:  
    Print(Exception ,e)  
except FileNotFoundError as e:  
    Print(FileNotFoundError, e)
```

Какой блок поймает исключение если файл не обнаружен ?



## Блок `finally` выполняется всегда

```
try:
    file = open('ok.txt', 'r')
    lines = file.readlines()
    print(lines[5]) ← Обращение к несуществ. строке
except IndexError as e:
    print(e)
finally:
    file.close()
    if file.closed:
        print("файл закрыт!")
> файл закрыт!
```





# Каскадное включение перехватчиков

```
try:
```

```
.....
```

```
    try:
```


```
        .....
```

```
    except E:
```

```
        .....
```

```
except E:
```

```
.....
```



```
import numpy as np
```

```
def divide(x, y):
```

```
    try:
```

```
        out = x/y
```

```
    except:
```

```
        try:
```

```
            out = np.inf * x / abs(x)
```

```
        except:
```

```
            out = np.nan
```

```
    finally:
```

```
        return out
```

```
divide(15, 3)    # 5.0
```

```
divide(15, 0)    # inf
```

```
divide(-15, 0)   # -inf
```

```
divide(0, 0)     # nan
```



# Генерация исключений в Python

Для принудительной генерации исключения используется инструкция **raise**.

```
try:  
    raise Exception("Что то пошло не так")  
except Exception as e:  
    print("Message:" + str(e))
```



Валидатор входного строкового значения на имя человека.

```
def validate(name) :  
    if len(name) < 10:  
        raise ValueError  
  
try:  
    name = input("Введите имя:")  
    validate(name)  
except ValueError:  
    print("Имя слишком короткое:")
```



# Пользовательские исключения

В Python можно создавать собственные исключения. Такая практика позволяет увеличить гибкость процесса обработки ошибок в рамках той предметной области, для которой написана ваша программа.

```
class NameTooShortError(ValueError):  
    pass  
  
def validate(name):  
    if len(name) < 10:  
        raise NameTooShortError
```





# Пользовательские исключения в Python

Для реализации собственного типа исключения необходимо создать класс, являющийся наследником от одного из классов исключений.

```
class NegValException(Exception):  
    pass  
  
try:  
    val = int(input("input positive number: "))  
    if val < 0:  
        raise NegValException("Neg val: " + str(val))  
    print(val + 10)  
except NegValException as e:  
    print(e)
```



# Вызов конструктора базового класса

```
class NegValException(Exception):  
    def __init__(self, number):  
        super().__init__(f"Neg val: {number}")  
        self.number = number  
  
try:  
    val = int(input("input positive number: "))  
    if val < 0:  
        raise NegValException(val)  
  
except NegValException as e:  
    print(e)
```




Продолжение следует...



# Handle psycopg2 exceptions that occur while connecting to PostgreSQL

```
# declare a new PostgreSQL connection object
try:
    conn = connect(
        dbname = "python_test",
        user = "WRONG_USER",
        host = "localhost",
        password = "mypass"
    )
except OperationalError as err:
    # pass exception to function
    print_psycopg2_exception(err)

    # set the connection to 'None' in case of error
    conn = None
```



```
def print_psycopg2_exception(err):
    # get details about the exception
    err_type, err_obj, traceback = sys.exc_info()

    # get the line number when exception occurred
    line_num = traceback.tb_lineno

    # print the connect() error
    print ("\npsycopg2 ERROR:", err, "on line
number:", line_num)
    print ("psycopg2 traceback:", traceback, "--
type:", err_type)

    # psycopg2 extensions.Diagnostics object attribute
    print ("\nextensions.Diagnostics:", err.diag)

    # print the pgcode and pgerror exceptions
    print ("pgerror:", err.pgerror)
    print ("pgcode:", err.pgcode, "\n")
```





# Pecypc

<https://kb.objectrocket.com/postgresql/python-error-handling-with-the-psycopg2-postgresql-adapter-645>