

Объектно-ориентированное программирование на Python

Осипов Никита Алексеевич

ЛЕКЦИЯ 9. ИСКЛЮЧЕНИЯ

Учебные вопросы:

1. Основы исключений.
2. Создание классов исключений.
3. Использование иерархии классов исключений.

Runtime Errors

Ошибка при
выполнении
программы

```
# Create a function that multiplies two provided variables and returns the result.  
# Provide the values using keyboard input.
```

```
def function4():  
    first = int(input("The first number is: "))  
    second = int(input("The second number is: "))  
    c = first * second  
    print("The result of ",first , " multiplied by ", second , " is: ", c)  
    return c
```

```
output = function4()  
print("The value of 'output' is: ",output)
```

The first number is: a

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-3-1657c53b5eb0> in <module>()  
      9  
     10  
----> 11 output = function4()  
      12 print("The value of 'output' is: ",output)  
  
<ipython-input-3-1657c53b5eb0> in function4()  
      2 # Provide the values using keyboard input.  
      3 def function4():  
----> 4     first = int(input("The first number is: "))  
      5     second = int(input("The second number is: "))  
      6     c = first * second
```

```
ValueError: invalid literal for int() with base 10: 'a'
```

Понятие исключений

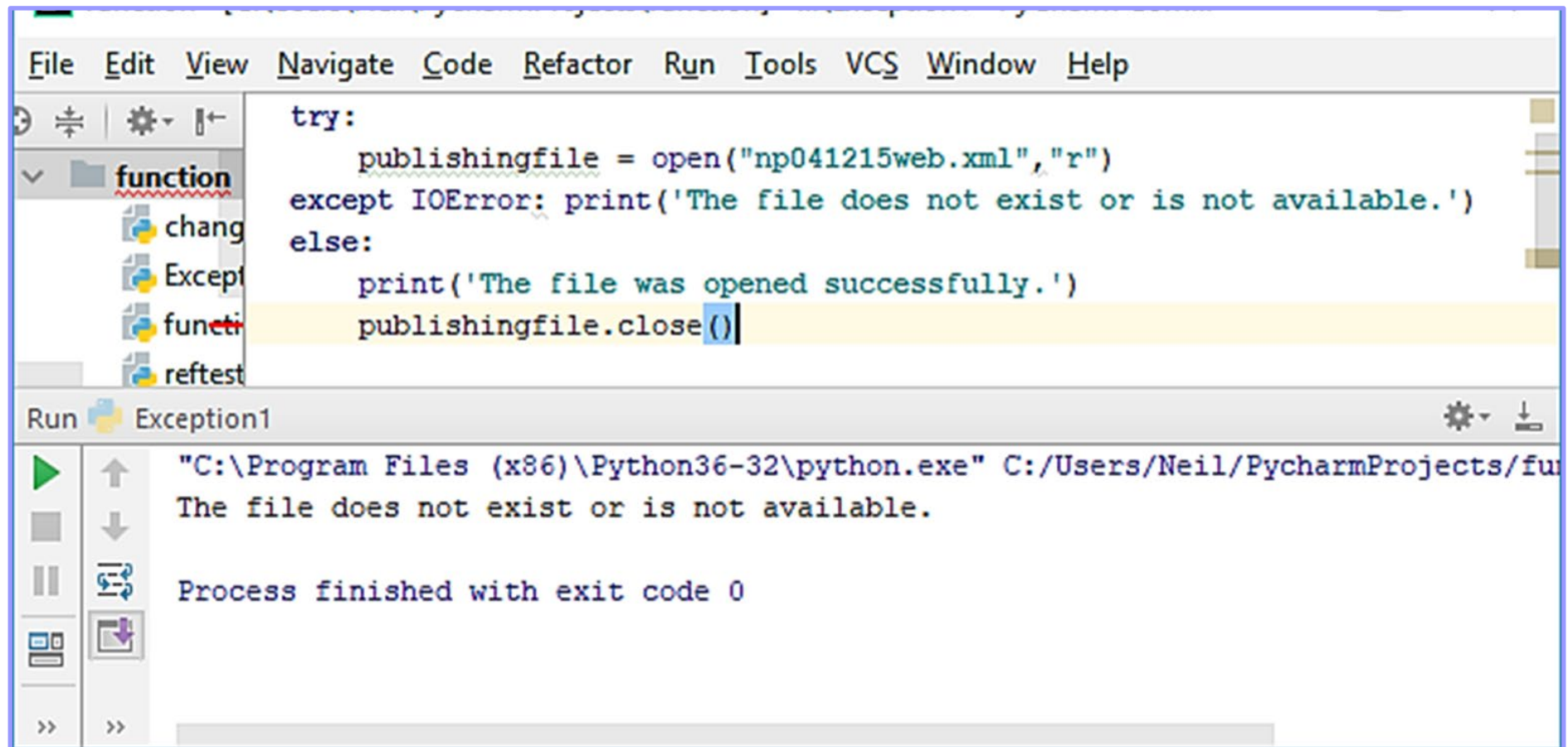
- Исключения – являются событиями, способными изменить ход выполнения программы

Обработчик исключений (инструкция try) ставит метку и выполняет некоторый программный код

Если затем где-нибудь в программе возникает исключение, интерпретатор немедленно возвращается к метке, отменяя все активные вызовы функций, которые были произведены после установки метки

Понятие исключений

- Исключения – являются событиями, способными изменить ход выполнения программы



The screenshot displays the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The left sidebar shows a project tree with a folder named 'function' containing files like 'chang', 'Except', 'functi', and 'reftest'. The main editor window shows a Python script with a try-except block. The code is as follows:

```
try:
    publishingfile = open("np041215web.xml", "r")
except IOError: print('The file does not exist or is not available.')
else:
    print('The file was opened successfully.')
    publishingfile.close()
```

The bottom panel shows the 'Run' tab with the title 'Exception1'. It displays the command used to run the program: `"C:\Program Files (x86)\Python36-32\python.exe" C:/Users/Neil/PycharmProjects/fu`. The output of the program is: `The file does not exist or is not available.` Below the output, it states: `Process finished with exit code 0`.

Области применения исключений

■ Обработка ошибок

- Интерпретатор возбуждает исключение каждый раз, когда обнаруживает ошибку во время выполнения программы.
- Программа может перехватывать такие ошибки и обрабатывать их или просто игнорировать

■ Уведомления о событиях

- Уведомление о наступлении некоторых условий, что устраняет необходимость передавать куда-либо флаги результата или явно проверять их

■ Обработка особых ситуаций

- Для редких событий проверку их наступления можно заменить обработчиками исключений

■ Заключительные операции

- Конструкция try/finally позволяет гарантировать выполнение завершающих операций независимо от наличия исключений

Обработка исключений

ErrorHandler.py

- Программы должны оставаться активными даже после появления внутренних ошибок:
 - Если требуется избежать реакции на исключение по умолчанию достаточно перехватить исключение, обернув опасный код инструкцией **try**

```
a = 10
try:
    b = int(input("Введите знаменатель: "))
    c = a/b
    print(c)
except ValueError:
    print("Преобразование прошло неудачно")
except ZeroDivisionError:
    print("Error - деление на ноль")
except:
    print("Error")
```

Обработка исключений

ErrorHandlerEx.py

- Программы должны оставаться активными даже после появления внутренних ошибок:
 - Получение информации об исключении: **as e**

```
a = 10
try:
    b = int(input("Введите знаменатель: "))
    c = a/b
    print(c)
except ValueError as e:          # Получение информации об исключении
    print("Преобразование прошло неудачно", e)
except ZeroDivisionError:
    print("Error - деление на ноль")
except:
    print("Error")
```


Обработка исключений

exепt01.py

- Программы должны оставаться активными даже после появления внутренних ошибок:
 - Если источник проблем – функция, то что оборачивается блоком **try**?

Функция
безобидна

```
def fun(obj, index):  
    return obj[index]
```

```
try:  
    k = int(input("Введите индекс: "))  
    f = fun(x, k)  
except ValueError as er:  
    print('Внимание! ', type(er), er)  
except IndexError:  
    print('\nИндекс вне диапазона')  
else:  
    print("\nВвод успешный, элемент: ", f)
```

Вызов функции опасен

Запускается, если в
блоке **try** **не возникло**
исключения

Обработка исключений

exept02finally.py

■ Заключительные операции:

- Комбинация try/finally определяет завершающие действия, которые **всегда** выполняются «на выходе», независимо от того, возникло исключение в блоке try или нет

```
while True:
    try:
        k = int(input("Введите индекс: "))
        f = fun(x, k)
        break
    except ValueError as er:
        print('Внимание! ', type(er), er)
    except IndexError:
        print('\nИндекс вне диапазона')
    finally:
        print('Отключите питание')
    print('End loop')
print(f)
```

```
def fun(obj, index):
    return obj[index]
```

Возбуждение (генерация) исключений

exept03raise.py

- Исключения могут возбуждаться интерпретатором или самой программой
 - Чтобы возбудить исключение вручную → выполнить инструкцию **raise**

```
while True:
    try:
        k = int(input("Введите индекс: "))
        if k == 0:
            raise Exception('Нулевой индекс зарезервирован')
        f = fun(x, k)
        break
    except Exception as e:
        print('\nОшибка!', e)
    finally:
        print('Отключите питание')
print('End loop')
print(f)
```

```
def fun(obj, index):
    return obj[index]
```

Исключения, основанные на классах

- Могут быть организованы в категории
 - классы исключений поддерживают возможность изменения в будущем
- Могут нести в себе информацию о состоянии
 - включать как информацию о состоянии, так и методы, доступные через экземпляры класса
- Поддерживают наследование
 - участие в иерархиях наследования с целью обладания общим поведением
- Лучше поддерживают возможность развития программ и крупных систем

Иерархия исключений

exept03raise.py

- Исключения на основе классов идентифицируются отношением наследования
 - возбужденное исключение считается соответствующим предложению **except**, если в данном предложении указан класс исключения или любой из его суперклассов

```
while True:
    try:
        ...
    except Exception as e:
        print('\nОшибка!', e)
    finally:
        print('Отключите питание')
        print('End loop')
print(f)
```

Когда в инструкции try предложение except содержит суперкласс, оно будет перехватывать экземпляры этого суперкласса, а также экземпляры всех его подклассов, расположенных ниже в дереве наследования

Базовые классы исключений

Иерархия встроенных исключений

ClassExept00.py

■ Базовый класс BaseException

- Базовый класс для всех встроенных исключений. Не предназначен для прямого наследования определяемыми пользователем классам

■ Exception

- Все встроенные, не выходящие из системы исключения являются производными от этого класса. Все определяемые пользователем исключения также должны быть производными от этого класса.

■ ArithmeticError

- Базовый класс для трех встроенных исключений, которые поднимаются для различных арифметических ошибок:
 - OverflowError, ZeroDivisionError, FloatingPointError.

■ BufferError

- Когда операция, связанная с буфером, не может быть выполнена.

■ LookupError

- Базовый класс для исключений, которые возникают, когда ключ или индекс, используемые в коллекции, недопустимы: IndexError, KeyError.

Создание пользовательского исключения

exept03raiseMyException.py

- Исключения должны быть получены из класса Exception, прямо или косвенно

```
class MyError(Exception):  
  
    def __init__(self, value):  
        self.value = value  
  
    def __str__(self):  
        return(repr(self.value))
```

```
try:  
    raise(MyError(13*21))  
  
except MyError as error:  
    print('New Exception: ',error.value)
```

Значение исключения
хранится в ошибке

Создание классов исключений

ClassExept01.py

ClassExept02.py

- Исключения суперкласса создаются, когда необходимо обработать несколько различных ошибок
 - Подклассы определяют для создания типов исключений для различных ошибок

```
class General(Exception):  
    pass  
  
class Specific1(General):  
    pass  
  
class Specific2(General):  
    pass
```

Когда в инструкции try предложение except содержит суперкласс, оно будет перехватывать экземпляры этого суперкласса, а также экземпляры всех его подклассов, расположенных ниже в дереве наследования

Методы классов исключений

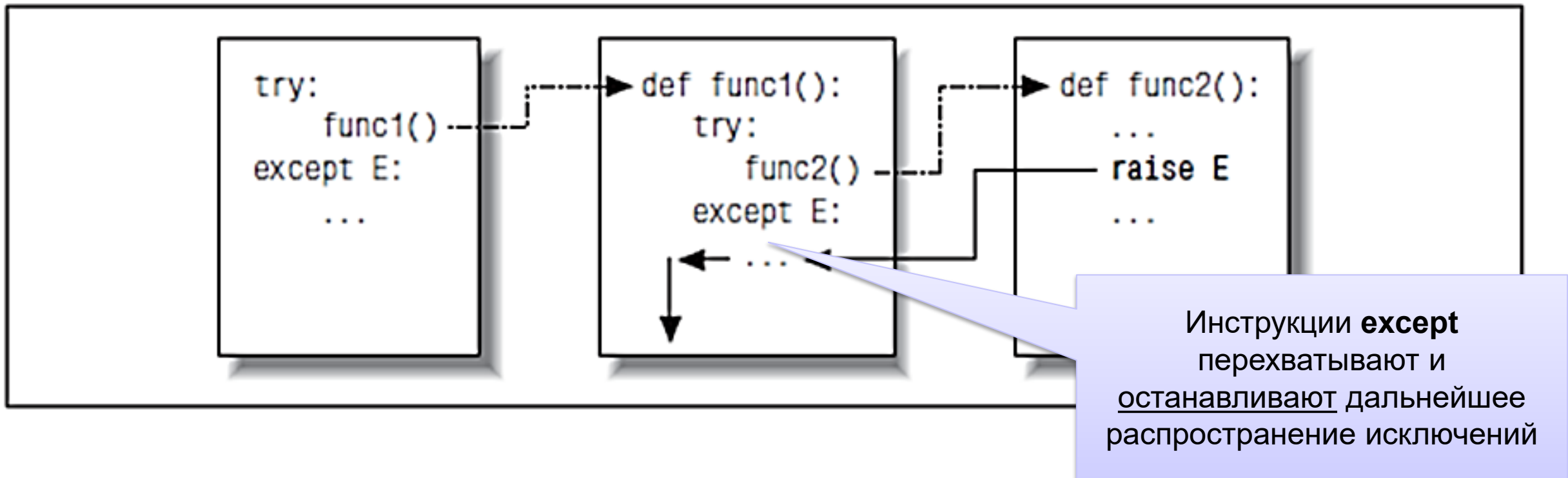
- Адаптированные классы могут использоваться для реализации специфического поведения объектов исключений
 - Класс исключения может определять дополнительные методы для использования в обработчиках

```
class Error(Exception):  
  
    def __init__(self, value, file):  
        self.value = value  
        self.file = file  
  
    def __str__(self):  
        return repr(self.value)  
  
    def logerror(self):  
        log = open(self.file, 'a')  
        print('Error!', 'Не допустимое значение:', self.value, file=log)
```

При использовании подобных классов методы (такие, как `logger`) могут наследоваться подклассами

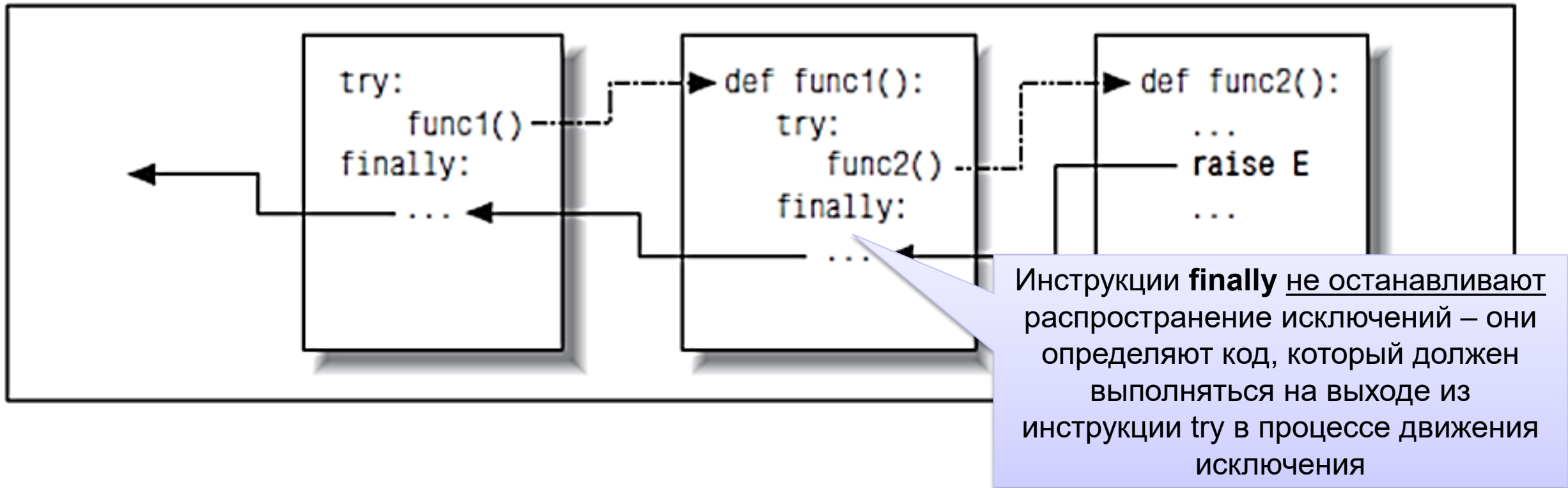
Вложенные обработчики исключений

- Что произойдет, если внутри инструкции **try** вызывается функция, которая выполняет другую инструкцию **try**?
 - Интерпретатор складывает инструкции try стопкой во время выполнения
 - Вложенные инструкции **try/except**



Вложенные обработчики исключений

- Что произойдет, если внутри инструкции **try** вызывается функция, которая выполняет другую инструкцию **try**?
 - Интерпретатор складывает инструкции try стопкой во время выполнения
 - Вложенные инструкции **try/finally**



Рекомендации по применению

- Исключения, определяемые программой могут служить сигналами об условиях, которые не являются ошибками
 - Например, процедура поиска может предусматривать возбуждение исключения в случае нахождения соответствия вместо того, чтобы возвращать флаг состояния, который должен интерпретироваться вызывающей программой

```
class Found(Exception): pass
def searcher():
    if ...успех...:
        raise Found()
    else:
        return
try:
    searcher()
except Found: # Исключение, если элемент найден
    ...успех...
else: # иначе: элемент не найден
    ...неудача...
```

Инструкция **try/except/else** играет роль инструкции if/else, предназначенной для проверки возвращаемого значения