Исключения

№ урока: 4 **Курс:** Python Essential

Средства обучения: PyCharm

Обзор, цель и назначение урока

После завершения урока обучающиеся будут иметь представление об обработке ошибок и исключительных ситуаций и смогут пользоваться механизмом исключений в языке Python.

Изучив материал данного занятия, учащийся сможет:

- Иметь представление о разных видах ошибок и исключительных ситуаций
- Иметь представление о механизме исключений в языке Python
- Обрабатывать исключения
- Выбрасывать исключения
- Пользоваться стандартными классами исключений
- Создавать собственные исключения
- Пользоваться механизмом предупреждений

Содержание урока

1. Что такое исключительные ситуации?
2. Синтаксические ошибки в Python
3. Логические ошибки в Python
4. Обработка исключений
5. Создание собственных исключений

Резюме

Исключения в Python - это чрезвычайная ситуация, которая возникла в процессе выполнения кода. Программисты иногда совершают **ошибки**, которые приводят к исключениям.
Иногда программисты специально создают **исключительные ситуации**, чтобы использовать их, как часть логики программы. Рассмотрим детальнее.
Все **ошибки** (не исключения) в коде можно разделить на два типа:

- Синтаксические ошибки
- Логические ошибки (исключения)

Синтаксические ошибки в Python
Если программист не следует правильной структуре кода и синтаксиса, возникают **ошибки синтаксиса**. Рассмотрим пример:

```
>>> if 3 < 5
    print("Works fine!")

SyntaxError: invalid syntax
```

Во время запуска программы Python проверяет весь код на наличие синтаксических ошибок. После условия 3 < 5 пропущено двоеточие и Python не может обработать этот код корректно, что вызывает SyntaxError.

Логические ошибки в Python
Исключения (логические ошибки) - возникают в процессе выполнения программы. Например, попытка поделить на ноль (ZeroDivisionError), открыть не существующий файл (FileNotFoundError) или импортирование не существующего модуля (ImportError).

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Когда подобные ошибки возникают, Python создает **специальный объект исключения**. Если программист не написал алгоритм обработки данной ошибки, детали события выведутся в консоль, а программа остановится.
Вот несколько исключений:

```
>>> 1 / 0
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    1 / 0
ZeroDivisionError: division by zero
>>> open("itvdn.txt")
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    open("itvdn.txt")
FileNotFoundError: [Errno 2] No such file or directory: 'itvdn.txt'
```

Не правильные операции приводят к выбрасыванию (raise) исключения. В Python есть множество встроенных исключений, которые выбрасываются при соответствующих ситуациях в работе программы.
Мы можем посмотреть все встроенные исключения с помощью функции local():

```
print(dir(locals()['__builtins__']))
```

locals()['__builtins__'] вернет модуль со встроенными исключениями, а функция dir позволяет просмотреть все доступные атрибуты в виде строк.
Чтобы узнать, за что отвечает каждое исключение, обратитесь в официальной документации: https://docs.python.org/3.8/library/exceptions.html
Вы так же можете создавать собственные исключения, если вам недостаточно встроенных — это распространенная практика.

Обработка исключений
При возникновении исключения, программа останавливает работу.
try – except — механизм позволяющий предотвратить остановку программы и корректно обработать исключительную ситуацию.
Программа, в которой намеренно создана ошибка:

```
def divide(n_1, n_2):
    return n_1 / n_2

def print_divide(n_1, n_2):
    result = divide(n_1, n_2)
    print(f"Divide result: {result}")

def run_divide(number_1, number_2):
    print("Dividing numbers")
    print_divide(number_1, number_2)

if __name__ == "__main__":
    run_divide(10, 0)
```

Во время выполнения программы произойдет ошибка в функции divide(), которая не сможет произвести деление на ноль. При этом в момент ошибки будет следующая цепочка вывозов функций: **run_divide → print_divide→ divide.**
Обработаем ошибку:

```
def divide(n_1, n_2):
    return n_1 / n_2

def print_divide(n_1, n_2):
    result = divide(n_1, n_2)
    print(f"Divide result: {result}")

def run_divide(number_1, number_2):
    print("Dividing numbers")
    print_divide(number_1, number_2)
```


CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
if __name__ == "__main__":
    try:
        run_divide(10, 0)
    except:
        print("Some error happened...")
```

Блок try оборачивает код, в котором может случиться ошибка. Если произошла ошибка, выполнение блока try останавливается и выполняется блок except для ее обработки. Исключение имеют свойство всплытия: если ошибка не была обработана сразу, она будет всплывать до самой первой функции, из цепочки вызовов. Это значит, что если ошибка возникла в функции divide, она будет всплывать до самой главной функции run_divide, пока ее не обработают.
Рассмотрим пример с двумя вызовами функции run_divide:

```
def divide(n_1, n_2):
    return n_1 / n_2

def print_divide(n_1, n_2):
    result = divide(n_1, n_2)
    print(f"Divide result: {result}")

def run_divide(number_1, number_2):
    print("Dividing numbers")
    print_divide(number_1, number_2)

if __name__ == "__main__":
    try:
        run_divide(10, 0)
        run_divide(10, 2)
    except:
        print("Some error happened...")
Dividing numbers
Some error happened...
```

Хоть ошибка возникла в первом вызове функции run_divide, Python не вызвал второй раз функцию. Это приводит нас к первому совету: **обрабатывайте ошибки как можно локальнее**, например, так:

```
def divide(n_1, n_2):
    return n_1 / n_2

def print_divide(n_1, n_2):
    try:
        result = divide(n_1, n_2)
        print(f"Divide result: {result}")
    except:
        print("Error in numbers dividing!")

def run_divide(number_1, number_2):
    print("Dividing numbers")
    print_divide(number_1, number_2)

if __name__ == "__main__":
    run_divide(10, 0)
    run_divide(10, 2)
```

Конструкция обработки ошибок переместилась ближе к потенциальному месту исключений. Теперь мы можем вызывать функцию run_divide несколько раз, несмотря на возможные ошибки. Но ошибки могут быть разного рода, как например отличить TypeError от ZeroDivisionError? Просто:

```
def divide(n_1, n_2):
    return n_1 / n_2

def print_divide(n_1, n_2):
    try:
        result = divide(n_1, n_2)
```


CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
        print(f"Divide result: {result}")
    except ZeroDivisionError:
        print("You can't divide by zero")
    except TypeError:
        print("Some value was incorrect!")

def run_divide(number_1, number_2):
    print("Dividing numbers")
    print_divide(number_1, number_2)

if __name__ == "__main__":
    run_divide(10, 0)
    run_divide(10, '2')
```

Теперь мы можем прописать логику обработки для разных ошибок.
Обратите внимание, что, если произойдёт ошибка, которую вы не обработали, она будет всплывать выше, пока ее не обработает другая конструкция try - except или остановится программа.
Создание собственных исключений
Исключения созданы, чтобы уведомлять программиста или пользователя об ошибке. Такая ошибка имеет имя и некоторые детали (Traceback). Чтобы изменить имя исключения, надо создать собственный класс исключения.
Выберите имя для исключения, например, "UserInputError". Эту ошибку мы будем выбрасывать, когда пользователь вводит недопустимые символы: {'/', ';', '*', '%'}.

```
NOT_ALLOWED_SYMBOLS = ['/', ';', '*', '%']

class UserInputError(Exception):
    pass

def has_string_symbols(string, symbols):
    for symbol in symbols:
        if symbol in string:
            return True

    return False

def user_input_check_loop():
    while True:
        user_string = input("Enter a string to check: ")
        if has_string_symbols(user_string, NOT_ALLOWED_SYMBOLS):
            raise UserInputError

if __name__ == "__main__":
    user_input_check_loop()
```

Переменная NOT_ALLOWED_SYMBOLS хранит список запрещённых символов.
UserInputError - это пользовательское исключение. Все пользовательские исключения должны наследоваться от класса Exception (или его наследников). Зачастую, пользовательские исключения не описывают никакой логики и блок кода заполняется оператором pass. При выбрасывании, исключение будет иметь имя класса.
Функция has_string_symbols проверяет есть ли символы (symbols) в строке (string).
Функция user_input_check_loop в цикле спрашивает у пользователя строку. Если пользователь введет строку с запрещенным символом, будет выброшено исключение с помощью оператора raise.
Оператор raise принимает объект. Если вы передадите класс оператор raise сам создаст объект. Значит, raise UserInputError то же, что и raise UserInputError(). Для дополнительного описания ошибки в объект исключения можно передать строку: raise UserInputError("You used not allowed symbols!")
Ошибка выглядит так:

```
Enter a string to check: *
Traceback (most recent call last):
  File "temp.py", line 24, in <module>
    user_input_check_loop()
  File "temp.py", line 20, in user_input_check_loop
    raise UserInputError("Test")
__main__.UserInputError: You used not allowed symbols!
```


CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Закрепление материала

- Что такое исключительная ситуация (исключение)?
- Какой оператор в Python предназначен для обработки исключений?
- Какие его основные блоки? Какое их назначение?
- Какой оператор в Python предназначен для выброса исключений?
- Чем отличается создание объекта исключения от выброса исключения?
- Каким образом можно получить экземпляр обрабатываемого исключения?
- Каким образом задать одинаковый обработчик для нескольких разных исключений?
- От какого класса наследуются все исключения?
- Что такое сцепление исключений?
- Что такое синтаксическая ошибка?
- Что такое предупреждение и как его сгенерировать?

Дополнительное задание

Задание
Опишите свой класс исключения. Напишите функцию, которая будет выбрасывать данное исключение, если пользователь введёт определённое значение, и перехватите это исключение при вызове функции.

Самостоятельная деятельность учащегося

Задание 1
Выучите основные стандартные исключения, которые перечислены в данном уроке.

Задание 2
Напишите программу-калькулятор, которая поддерживает следующие операции: сложение, вычитание, умножение, деление и возведение в степень. Программа должна выдавать сообщения об ошибке и продолжать работу при вводе некорректных данных, делении на ноль и возведении нуля в отрицательную степень.

Задание 3
Опишите класс сотрудника, который включает в себя такие поля, как имя, фамилия, отдел и год поступления на работу. Конструктор должен генерировать исключение, если заданы неправильные данные. Введите список работников с клавиатуры. Выведите всех сотрудников, которые были приняты после заданного года.

Рекомендуемые ресурсы

Информация о механизме исключений
https://docs.python.org/3/tutorial/errors.html
https://docs.python.org/3/tutorial/classes.html#exceptions-are-classes-too
https://docs.python.org/3/reference/executionmodel.html#exceptions
Встроенные исключения
https://docs.python.org/3/library/exceptions.html
Модуль warnings
https://docs.python.org/3/library/warnings.html

Статьи в Википедии о ключевых понятиях, рассмотренных на этом уроке
https://ru.wikipedia.org/wiki/Обработка_исключений

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com