Обработка исключений

Иногда во время работы программы возникают ошибки. Такие ошибки в программировании называют исключениями. Мы уже сталкивались с исключением KeyError, которое вызывается, если мы обратились к словарю по несуществующему ключу.

Приведем пример. Предположим, мы пишем программу, которая бесконечно считывает два числа и

1 a = int(input('a = '))  
2 b = int(input('b = '))  
3  
4 result = a / b  
5 print('a / b = ', result)

Вывод на консоль:

a = **1**

b = **0**

Traceback (most recent call last):

File "examle.py", line 4, in <module>

result = a / b

ZeroDivisionError: division by zero

Работа программы завершилась из-за возникновения исключения («ZeroDivisionError» – ошибка деления на ноль) в 4 строке (result = a / b) нашего файла ("examle.py").

В данном случае ошибку можно предотвратить простой проверкой делителя, тогда при делении на 0 выведется сообщение 'Ошибка деления на 0!' и программа продолжит работу:

1 a = int(input('a = '))  
2 b = int(input('b = '))  
3

4 if b != 0:  
5 result = a / b  
6 print('a / b = ', result)

7 else:

8 print('Ошибка деления на 0!')

Вывод на консоль:

a = **1**

b = **0**

Ошибка деления на 0!

Иногда очень сложно предугадать все ошибки в программе, или условия становятся слишком сложными. Python предоставляет возможности для обработки исключений. Для этого существует конструкция try/except:

try:

# код

except [ИмяИсключения]:

# код

«ИмяИсключения» указывает, какое исключение обрабатывает данный блок except.

Рассмотрим код:

try:  
 a = int(input('a = '))  
 b = int(input('b = '))  
  
 result = a / b  
 print('a / b = ', result)  
  
except ZeroDivisionError:  
 print('Нельзя делить на 0!')  
  
print('Все отлично!')

В данном случае блок except обрабатывает ошибку «ZeroDivisionError».

Работа начинается с блока try, затем:

* Если код в блоке try вызывает исключение, которое указано в «ИмяИсключения», то выполняется код, который указан после выражения except;

Вывод на консоль:

a = **1**

b = **0**

Нельзя делить на 0!

Конец

* Если код в блоке try вызывает исключение, которое не указано в «ИмяИсключения», то программа выдаст сообщение об ошибке и завершит работу;

Вывод на консоль:

a = **1**

b = **пять**

…

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

# в данном примере произошла ошибка значения – строка 'пять' не смогла привестись к типу int

* Если код в блоке try не вызывает исключение, то код в блоке except пропускается, и программа продолжает работу.

Вывод на консоль:

a = **1**

b = **5**

a / b = 0.2

Конец

Иногда есть необходимость обработать все исключения сразу. Если «ИмяИсключения» не указано, то блок except будет обрабатывать исключение любого типа, однако, мы не узнаем исключение какого типа было вызвано и не сможем грамотно обработать исключение.

Пример с try/except без «ИмяИсключения»:

while True:  
 try:  
 a = int(input('a = '))  
 b = int(input('b = '))  
  
 result = a / b  
 print('a / b = ', result)  
  
 except:  
 print('Произошла ошибка! Не знаю, какая именно.')

Вывод на консоль:

a = **девять**

Произошла ошибка! Не знаю, какая именно.

a = **2**

b = **0**

Произошла ошибка! Не знаю, какая именно.

Также, можно обрабатывать сразу несколько типов исключений, с помощью конструкции:

try:

# код

except ИмяИсключения1:

# код

except ИмяИсключения2:

# код

…

while True:  
 try:  
 a = int(input('a = '))  
 b = int(input('b = '))  
  
 result = a / b  
 print('a / b = ', result)  
  
 except ValueError:  
 print('Вы ввели значение, которое нельзя привести к числу.')  
  
 except ZeroDivisionError:  
 print('Нельзя делить на 0!')

Вывод на консоль:

a = **девять**

Вы ввели значение, которое нельзя привести к числу.

a = **2**

b = **0**

Нельзя делить на 0!

При написании программы не всегда можно обойтись стандартными исключениями. Например, мы пишем программу, которая спрашивает у пользователя длину и ширину прямоугольника и возвращает его площадь:

a = int(input('Длина прямоугольника = '))  
b = int(input('Ширина прямоугольника = '))  
  
s = a \* b  
print('Площадь прямоугольника =', s)

Как мы знаем длина и ширина прямоугольника должны быть больше 0. Если пользователь введёт некорректные данные, то программа посчитает результат, но ее работа будет некорректной. Поэтому мы можем возбудить собственное исключение с помощью ключевого слова raise:

a = int(input('Длина прямоугольника = '))  
if a <= 0:  
 raise Exception('Длина не может быть отрицательной')  
b = int(input('Ширина прямоугольника = '))  
if b <= 0:  
 raise Exception('Ширина не может быть отрицательной')  
  
s = a \* b  
print('Площадь прямоугольника =', s)

Вывод на консоль:

Длина прямоугольника = **10**

Ширина прямоугольника = **-10**

…

Exception: Ширина не может быть отрицательной

**Функции**

В этой главе рассматриваются способы создания пользовательских функций в языке Python.

**Пользовательские функции**

Программа – это последовательность инструкций. Некоторые блоки инструкций неоднократно повторяются, что усложняет “читаемость” программного кода. Чтобы устранить подобного рода избыточность, используют специальные наименованные блоки инструкций, называемые функциями (подпрограммами). В простейшем случаи синтаксис создания функции в языке Python имеет следующий вид:

def functionName():

набор инструкций

Ключевое слово def указывает на начало описания функции. Далее следует название функциии, круглые скобки, двоеточие и набор инструкций - тело функции. Тело функции пишется с отступом со следующей строки и не может быть пустым.

Примеры. Функции без параметров:

# Пример 1  
def myFunc():  
 print('Hello world')  
  
  
# Пример 2  
def sumFunc():  
 a = 10  
 b = 20  
 c = a + b  
 if c > 29:  
 print(c)  
 else:  
 print(-c)  
  
  
# Пример 3  
# Неправильное объявление функции  
def someFunc() # Отсутствует двоеточие  
 # Пустое тело функции  
f = 10  
  
# Выведет на консоль: Hello world  
myFunc()  
# Выведет на консоль: 30  
sumFunc()

В функции можно передавать значения из основной программы. Для этого при объявлении функции в круглых скобках требуется перечислить через запятую список параметров, которые будет принимать функция. При вызове функции нужно указать значения всех параметров. Значения, с которыми будет вызвана функция называются аргументами функции. Тогда, синтаксис объявления функции примет вид:

def functionName(параметры):

набор инструкций

Примеры:

# Пример 1  
# a и b - параметры функции  
def add(a, b):  
 print(a + b)  
  
  
# Пример 2  
# number - параметр функции  
def square(number):  
 sq = number \*\* 2  
 print(sq)  
  
  
r = 2  
m = 10  
# Выведет на консоль сумму 12  
# r и m - аргументы функции add()

# a = r, b = m  
add(r, m)  
  
# Выведет на консоль 4  
# 2 - аргумент функции square()

# number = 2  
square(2)

Вызывая функции из примера 1 и 2, говорят, что используются позиционные аргументы, т.е. каждый переданный параметр становится значением параметра в соответствующей позиции. Некоторые функции могут иметь параметры для которых определены значения по умолчанию, т.е. в случаи, когда аргумент для этого параметра отсутствует, параметр принимает значение, заданное при объявлении функции. Объявление значения по умолчанию для параметра функции имеет вид: param = default\_value. Такие параметры обязательно должны быть описаны в конце списка параметров функции и называются необязательными параметрами.

Примеры:

# Пример 1  
# Функция имеет 3 параметра: a,b,l.  
# Параметр l = по умолчанию имеет значение None  
def func1(a, b, l=None):  
 if l is None:  
 l = []  
 l += [a, b]  
 print(l \* 2)  
  
  
# Пример 2  
# Неправильное объявление списка параметров  
# Параметры имеющие значения по умолчанию   
# должны быть описаны в конце списка  
  
def func2(a,b = 10,c,d):  
 print(b)  
  
# Выведет на консоль: [3, 4, 1, 2, 3, 4, 1, 2]  
func1(1, 2, [3, 4])  
# Выведет на консоль: [4, 6, 4, 6]  
func1(4, 6)

При вызове функции с параметрами не обязательно передавать аргументы в том порядке, в каком они указаны при определении функции. Можно использовать именованные аргументы, т.е. передавать аргументы в виде: name\_param = value.

Примеры:

# Пример 1  
# Функция проверяет длину строки  
# Если длина строки превышает значение параметра length, то на консоль выводится  
# срез строки от начало и до разницы между длиной  
# строки и длиной indicator с добавление значения параметра indicator в конец.  
# В противном случаи, строка полностью выводится на консоль  
def shorten(text, length=25, indicator="..."):  
 if len(text) > length:  
 print(text[:length - len(indicator)] + indicator)  
 else:  
 print(text)  
  
  
# Выведет на консоль: Мама мыла раму  
shorten('Мама мыла раму')  
# Выведет на консоль: Ма~~~  
shorten(indicator='~~~', length=5, text='Мама мыла раму')  
# Выведет на консоль: Мама мыла раму  
shorten(length=15, text='Мама мыла раму')

Функции в языке Python могут не только принимать, но и возвращать значения. Для того чтобы вернуть значения из функции, в конце тела функции прописывается оператор return, после которого следует список возвращаемых значений, перечисленных через запятую.

Примеры:

# Пример 1  
# Функция возвращает сумму a и b  
def add(left, right):  
 return left + right  
  
  
# Пример 2  
# Функция возвращает True - если треугольник со сторонами a, b, c существует  
# В противном случаи функция вернет False  
# Обратите внимание, что из одной функции можно вызвать другу функцию  
def IsExistsTriangle(a, b, c):  
 result = add(a, b) > c and add(a, c) > b and add(c, b) > a  
 part\_message = str(a) + ', ' + str(b) + ', ' + str(c)  
 if result:  
 print("Треугольник существует со сторонами: " + part\_message + ' - существует')  
 else:  
 print("Треугольник существует со сторонами: " + part\_message + ' - не существует')  
 return result  
  
  
# Пример 3  
# Функция возвращает два значения: остаток и целую часть от деления number на base  
def DivMod(number, base):  
 return number % base, number // base  
  
  
# Выведет на консоль: True  
print(IsExistsTriangle(3, 4, 5))  
# Записываем результат функции в переменную res  
res = IsExistsTriangle(1, 1, 3)  
# Выведет на консоль: False  
print(res)  
# При вызове функции DvMod в переменную dm будет записан кортеж из двух элементов: dm = (1,2)  
dm = DivMod(7, 3)

Все функции в языке Python возвращают какое-либо значение. Если оператор return указывается без списка значений или вообще не используется в теле функции, тогда функция вернет значение None. С помощью оператора return можно покинуть функцию в любой момент.

Примеры:

# Пример 1  
# Функция принимает list и добавляет в него значения от 0 до 4  
# Т.к. не указан оператор return, функция возвращает None  
def func1(lst):  
 for i in range(5):  
 lst.append(i)  
  
  
# Пример 2  
# Функция проверяет длину списка.  
# Если длина списка меньше максимальной длины, то функция вернет строку: "Длина списка меньше максимальной длины"  
# В противном случаи: "Длина списка больше либо равна максимальной длины"  
def func2(lst, max\_len):  
 if len(lst) < max\_len:  
 return "Длина списка меньше максимальной длины"  
 else:  
 return "Длина списка больше либо равна максимальной длины"  
  
  
short\_list = [8, 3, 0]  
# Выведет на консоль: None  
print(func1(short\_list))  
# Выведет на консоль: [8, 3, 0, 0, 1, 2, 3, 4]  
print(short\_list)  
# Выведет на консоль: Длина списка меньше максимальной длины  
print(func2(short\_list, 4))

В языке Python существует два типа переменных: глобальные и локальные. Глобальные переменные – это переменные, которые объявлены в основной программе и доступны для всех её подпрограмм только для чтения. Чтобы изменить значение глобальной переменной в функции, нужно в теле функции объявить ключевое слово global и через запятую перечислить глобальные переменные. Если не перечислить глобальные переменные с помощью ключевого слова global, то внутри функции такие переменные доступны только для чтения, т.е. их значение внутри функции менять нельзя. Локальные переменные – это переменные, которые объявлены внутри функций. Основная программа не имеет доступа к локальным переменным.

Примеры:

A = 10  
  
  
# Пример 1  
# Функция вернет значение увеличенное на A  
def addA(v):  
 return v + A  
  
  
# Выведет на консоль: 20  
print(addA(10))  
  
  
# Пример 2  
# Функция вернет площадь круга  
# Объявление глобальной переменной может следовать после объявления функции в которой она используется  
  
def areaCircle(r):  
 return r \*\* 2 \* Pi  
  
  
# Вызов функции до объявления глобальной Pi вызовет ошибку,  
# т.к. переменная Pi должна быть описана раньше вызова функции areaCircle  
areaCircle(2)  
  
Pi = 3.14  
  
# Выведет на консоль: 28.26  
print(areaCircle(3))  
  
I = 0  
  
  
# Пример 3  
# Функция выводит на экран переданный текст и подсчитывает число вызовов  
# Для того, что бы считать число вызовов используется глобальная переменная I  
# Ключевое слово global указывает на то, что в функции изменяется глобальная переменная  
def funcCount(text):  
 global I  
 print(text)  
 I += 1  
  
  
funcCount("Text 1")  
funcCount("Text 2")  
funcCount("Text 3")  
# Выведет на консоль: 3  
print(I)  
  
# Пример 3  
# Глобальная переменная K  
K = 0  
  
  
# Функция возводит переменную a в степень c и присваивает результат в локальную переменную K  
# Глобальная переменная K не изменится  
def funcExample(a, c):  
 # локальная переменна K  
 K = a \*\* c  
  
  
funcExample(3, 4)  
# Выведет на консоль: 0  
print(K)  
  
# Пример 4  
# Суммирование в локальную переменную, которая не объявлена  
def funcExample2(a,c):  
 K += a\*\*c