Глава 3

Обработка исключений: проект EXCEP

Проект EXCEP знакомит с приемами обработки исключительных ситуаций (исключений). Описывается структура try-блоков и демонстрируется осо­бенности, связанные с использованием вложенных try-блоков. Дается об­зор исключений, связанных с арифметическими операциями. Описывается варианта try-блока, использующий раздел finally.

3.1. Обработка конкретного исключения и групп исключений

В этом примере, как и в предыдущем, будет разрабатываться консольное приложение. Создайте файл EXCEP.py и добавьте в него следующий код (листинг 3.1).

**Листинг 3.1. Описание файла EXCEP.py**

**def M1(x, y, z):**

**try:**

**a = pow(x, y)**

**print('x ^ y / z = %d' % (a / z))**

**except ZeroDivisionError:**

**print('ZeroDivision Exception')**

**print('M1 finished')**

**def M2(x, y, z):**

**try:**

**M1(x, y, z)**

**except ArithmeticError:**

**print('Arithmetic Exception')**

**print('M2 finished')**

**def main():**

**x = int(input('x = '))**

**y = int(input('y = '))**

**z = int(input('z = '))**

**M2(x, y, z)**

**if \_\_name\_\_ == '\_\_main\_\_':**

**main()**

**Результат.** Для трех введенных целых чисел *x, y, z* программа вычисляет выражение *xy/z*, обрабатывая возникающие при этом *исключительные ситу­ации* (исключения, exceptions). Для выхода из программы надо нажать кла­вишу <Enter>. Опишем различные варианты работы программы.

**Вариант А. Обработка допустимых значений** (листинг 3.2). Вычисления выполняются успешно; ни один обработчик исключений не активизиру­ется.

**Листинг 3.2. Содержимое консольного окна при обработке допустимых**

**значений**

x = 9

y = 2

z = 3

x ^ y / z = 27

M1 finished

M2 finished

**Вариант B. Деление на 0** (листинг 3.3). Активизируется обработчик try-блока метода M1, обрабатывающий исключение типа ZeroDivisionError, после чего выполнение программы продолжается с оператора, следующего за данным try-блоком.

**Листинг 3.3. Содержимое консольного окна при делении на 0**

x = 1

y = 1

z = 0

ZeroDivision Exception

M1 finished

M2 finished

**Вариант C. Целочисленное переполнение** (листинг 3.4). При попытке воз­ведения числа 1000 в степень 1000 возникает исключение OverflowError. Поскольку в разделе except try-блока метода M1 обработка исключения OverflowError не предусмотрена, происходит немедленный переход в об­работчик try-блока следующего уровня (т.е. в раздел except try-блока метода M2). Здесь исключение OverflowError обрабатывается, так как оно является *потомком* исключения ArithmeticError – предка всех ис­ключений, порожденных ошибками при выполнении арифметических опе­раций. После этой обработки выполнение программы продолжается с опе­ратора, следующего за try-блоком метода M2.

**Листинг 3.4. Содержимое окна при целочисленном переполнении**

x = 1000

y = 1000

z = 1

Arithmetic Exception

M2 finished

**Вариант D. Ввод недопустимого символа.** После ввода недопустимого символа (например, звездочки \*) выполнение программы немедленно пре­рывается, в консоль выводится текст ошибки (листинг 3.5). Такое поведе­ние программы объясняется тем, что в методе main не предусмотрена об­работка возникшего исключения ValueError.

**Листинг 3.5. Вывод текста ошибки**

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

3.2. Обработка любого исключения

Измените метод main в файле EXCEP.py (листинг 3.6). Теперь программа содержит три вложенных try-блока.

**Листинг 3.6. Новый вариант метода main**

def main():

**try:**

x = int(input('x = '))

y = int(input('y = '))

z = int(input('z = '))

M2(x, y, z)

**except:**

**print('Other exception')**

input()

**Результат.** В любом из вариантов A, B, C, рассмотренных в *разд. 3.1,* ре­зультат работы программы будет прежним. В варианте D (ввод недопусти­мого символа) в консольном окне будет выведено:

x = \*

Other exception

и программа будет ожидать нажатия клавиши <Enter> для своего заверше­ния. Таким образом, никакое исключение теперь не приведет к аварийному завершению программы.

**Недочет.** Информация, выводимая на экран, не позволяет определить, ка­кое именно исключение возникло в ходе выполнения программы.

**Исправление.** Подключите в начале программы модуль logging, а также измените раздел except в методе main:

except **Exception as e**:

**logging.Logger('catch\_all').exception(e, exc\_info=False)**

**Результат.** Теперь в случае ввода недопустимого символа будет выведена более содержательная информация:

x = \*

invalid literal for int() with base 10: '\*'

Комментарий

В последнем варианте программы был определен обработчик для исключе­ния Exception – общего предка всех не полностью системных исключе­ний. Поэтому он активизируется при возникновении любого исключения, не обработанного в предыдущих try-блоках. Использование переменной e типа Exception позволяет получить доступ к методам и свойствам воз­никшего исключения. Вывод текста ошибки зависит от параметра exc\_info, эта информация выводится по умолчанию, однако в данном примере эти сведения не требуются, поэтому параметр равен False.

3.3. Повторное возбуждение обработанного исключения

Дополните раздел except для try-блока метода M2:

except ArithmeticError:

print('Arithmetic Exception')

**raise**

**Результат.** В любом из вариантов A, B, D результат работы программы будет прежним. В варианте С (целочисленное переполнение) в окне будет более содержательная информация (листинг 3.7). Это связано с тем, что до­бавленный в раздел except метода M2 оператор raise выполняет *по­вторное возбуждение* только что созданного исключения. Повторно воз­бужденное исключение окончательно обрабатывается в разделе except метода main *(см. разд. 3.2).*

**Листинг 3.7. Содержимое окна при целочисленном переполнении**

x = 1000

y = 1000

z = 1

Arithmetic Exception

integer division result too large for a float

Комментарий

1. Оператор raise используется также для явного возбуждения ис­ключения в программе. Например, если в некотором методе M пара­метр n целого типа может принимать только значения от 1 до nMax, то при нарушении этого условия следует возбудить в методе M ис­ключение.

if n < 1 or n > nMax:

raise Exception('n')

1. В try-блоке может содержаться несколько разделов except, каж­дый из которых будет обрабатывать исключения определенного типа. Предусмотрены также дополнительные разделы try-блока c именами else и finally, которые располагаются после всех раз­делов except. В блоке else содержится код, выполняемый в слу­чае, если исключение не было возбуждено; в блоке finally нахо­дится код, выполняющий очистку ресурсов, выделенных в про­грамме, и другие завершающие действия. Раздел finally выпол­няются всегда: и при нормальной работе операторов в блоке try, и при возбуждении исключения, причем даже в ситуации, если это ис­ключение не было перехвачено в предыдущих разделах except.