Исключительные ситуации

Программный код не всегда ведет себя так, как задумано, и при этом в этом не всегда виноват программист. Современные программы могут запускаться в большом количестве различных окружений и не всегда оно похоже на окружение, в котором программа разрабатывалась. В том случае, когда код по тем или иным причинам не может выполнить запрошенную операцию, вместо того, чтобы просто завершить программу с ошибкой, может быть сгенерирована *исключительная ситуация*. Исключительная ситуация это ошибочное состояние программы, которое может быть обработано или проигнорировано. В последнем случае обычно программа действительно завершается с ошибкой.

Обработка исключительных ситуаций производится при помощи однотипной конструкции языка программирования, которая может объединять набор инструкций, которые могут порождать однотипные исключительные ситуации. Игнорирование исключительных ситуаций не может пройти незамеченным: проигнорированное исключение “уронит” программу. Исключительные ситуации разделяются по типу, что позволяет программисту понять, что именно пошло не так.

Например, если обратиться к недопустимому индексу у списка, будет сгенерирована исключительная ситуация IndexError.

```
arr = []
print(arr[0]) # IndexError
```

Несмотря на то, что базовым классом для всех исключительных ситуаций в Python является BaseException, порождать и перехватывать имеет смысл только наследников класса Exception.

Обработка исключительных ситуаций

Для обработки исключительных ситуаций используется конструкция try-except-else-finally:

```
try:
    # тут код, который может порождать
    # исключительные ситуации
except ТипИсключения1:
    # действия, если ТипИсключения1 произошло,
    # переменная не нужна
except ТипИсключения2 as переменная:
    # действия, если ТипИсключения2 произошло,
    # переменная нужна
except:
```

```
    # действия, если сработало любая другая исключительная ситуация
else:
    # действия, которые нужно выполнить, если исключительной ситуации не произошло
finally:
    # действия, которые нужно выполнить вне зависимости от того, произошло исключение или нет
```

Может использоваться минимум один блок catch, если не указан блок finally. Если блок finally указан, блоков catch может не быть. Блок finally может быть указан максимум один раз. Блок else может быть указан максимум один раз.

В случае возникновения исключительной ситуации блоки catch просматриваются в том порядке, в котором они объявлены. Срабатывает первый подходящий блок, исключительная ситуация считается обработанной, остальные блоки catch не рассматриваются.

Если указан блок finally, вне зависимости от того, завершился ли блок try корректно, или сработал ли какой-нибудь catch или исключительная ситуация произошла, но ее не смог обработать ни один блок catch, сработает блок finally. Он хорошо подходит для освобождения ресурсов, которые захватываются внутри блока try, т.к. их надо освободить вне зависимости от того, произошла ли исключительная ситуация и обработана ли она.

Если не подошел ни один блок catch, исключительная ситуация считается необработанной, прерывается текущая функция и исключение “всплывает” в функцию, вызвавшую прерванную функцию. Так продолжается до тех пор, пока исключение не будет обработано, либо не будет прервана основная программа. В этом случае исполнение программы завершается с ошибкой.

Если исключительной ситуации не произошло, срабатывает блок else, если он объявлен. Блок else срабатывает до блока finally, если тот объявлен.

Пример обработки исключительной ситуации на месте:

```
arr = []
try:
    print(arr[0]);
except IndexError as e:
    print(f"Некорректный индекс: {e}")
```

Как видно из этого примера, получить текстовое описание исключительной ситуации можно, приведя ее экземпляр к строке.

Пример обработки “всплывающей” исключительной ситуации:

```
def failing():
    a = []
    x = a[0] # IndexError
    print(x) # никогда не выполнится

def recovering():
    try:
        failing() # всплывет IndexError
    except IndexError as e:
        print(e)
```

Порождение исключительных ситуаций

Прежде чем породить исключительную ситуацию, необходимо определиться с ее типом. Не рекомендуется использовать класс Exception напрямую, т.к. он не несет смысловой нагрузки о том, что именно произошло, и затрудняет использование других исключительных ситуаций в том же блоке try, что и он.

Можно использовать стандартные классы-наследники Exception, либо создать собственный.

Стандартных исключительных ситуаций в Python довольно много и полный актуальный список можно найти в документации: https://docs.python.org/3/library/exceptions.html.

Большинство стандартных исключительных ситуаций имеют конструктор, принимающий строку с описанием того, что произошло. Именно это значение и будет возвращено при преобразовании экземпляра исключительной ситуации к строке.

Для порождения исключительной ситуации используют оператор raise.

```
raise RuntimeError(“Что-то пошло не так”)
```

Зачастую в своем коде может не хватать стандартных исключительных ситуаций и тогда можно создавать свои собственные:

```
class InvalidTimeException(Exception):
    pass # нам нужно только имя, все остальное унаследовано
```

Пример использования собственного исключения и его перехвата:

```
def failing():
    raise InvalidTimeException("Неверный формат времени")

def recovering():
    try:
        failing()
    except InvalidTimeException as e:
        print(e)
        # ^^^ напечатает "Неверный формат времени"
```