Python XIV:
Обработка исключений

Исключения (exceptions) в Python - это специальные объекты, которые возникают при выполнении программы и обозначают ошибки или неожиданные ситуации. Когда возникает исключение, выполнение программы прерывается и Python пытается найти обработчик (handler) для этого исключения. Если обработчик не найден, программа аварийно завершается и выводится сообщение об ошибке.

В Python есть несколько встроенных типов исключений, таких как ValueError, TypeError, IndexError, KeyError и другие. Кроме того, вы можете создавать свои собственные исключения.

Для того, чтобы обработать исключение, нужно написать соответствующий обработчик, используя блок try-except. В блоке try указывается код, который может породить исключение, а в блоке except указывается код, который будет выполнен в случае, если исключение произошло. Также можно добавить блок finally, который будет выполнен независимо от того, произошло исключение или нет.

Пример обработки исключения ValueError:

```
try:
    x = int(input("Введите число: "))
    print("Введенное число:", x)
except ValueError:
    print("Ошибка! Введите число.")
```

В этом примере мы пытаемся преобразовать введенную пользователем строку в целое число с помощью функции int(). Если пользователь вводит строку, которую невозможно преобразовать в число, возникает исключение ValueError. В блоке except мы обрабатываем это исключение и выводим соответствующее сообщение.

Также можно указывать несколько блоков except для разных типов исключений:

```
try:
    some_code()
except ValueError:
    handle_value_error()
except TypeError:
    handle_type_error()
except:
    handle_other_exceptions()
```

Кроме того, можно использовать оператор else в блоке try-except. Код, который находится в блоке else, будет выполнен только в том случае, если исключение не возникло:

```
try:
    some_code()
except ValueError:
    handle_value_error()
else:
    handle_no_exceptions()
```

Блок finally будет выполнен в любом случае, даже если возникло исключение, и даже если было применено оператор return, break или continue в блоке try или except.

Синтаксис оператора finally следующий:

```
try:
    # выполнение кода
except Exception:
    # обработка исключения
finally:
    # код, который будет выполнен в любом случае
```

Пример использования оператора finally:

```
try:
    f = open("example.txt", "r")
    f.read()
except:
    print("Произошла ошибка при чтении файла")
finally:
    f.close()
```

В этом примере, если произойдет ошибка при чтении файла, то программа напечатает сообщение об ошибке, а затем закроет файл с помощью метода close(). Если ошибки не произойдет, то файл все равно будет закрыт в блоке finally.

Использование оператора finally особенно полезно, когда нужно освободить ресурсы, такие как файлы или соединения с базой данных, независимо от того, произошла ошибка или нет.

Оператор raise в Python используется для явного возбуждения исключения в программе.

Синтаксис оператора raise следующий:

```
raise [Exception [, args [, traceback]]]
```

- Exception - класс исключения, которое нужно возбудить. Если не указывать этот аргумент, будет использовано базовое исключение BaseException.
- args - необязательный аргумент, содержащий аргументы для исключения, которое возбуждается.
- traceback - необязательный аргумент, содержащий объект traceback.

Примеры использования raise:

```
# Возбуждение исключения ValueError
raise ValueError("Не корректное значение!")

# Возбуждение исключения с базовым классом Exception
raise Exception("Ошибка")

# Возбуждение исключения с аргументами
x = 10
if x > 5:
     raise ValueError("x должно быть меньше или равно 5")

# Возбуждение исключения с базовым классом Exception и
traceback
import traceback
try:
     x = 1 / 0
except ZeroDivisionError:
     raise Exception("Ошибка деления на ноль") from None
```

Как правило, оператор raise используется в блоках try-except для явного возбуждения исключения при возникновении определенных условий, например, при некорректном вводе пользователем данных или при наличии ошибок в программе.