Тема 14. Дополнительные материалы

14.1. Распаковка последовательностей

Распаковка позволяет разобрать последовательность на части и каждую часть присвоить отдельной переменной. Мы уже затрагивали распаковку, когда рассматривали множественное присваивание. В этом разделе рассмотрим ее более подробно.

Простая распаковка

Подразумевает, что заранее известно, сколько элементов в распаковываемой последовательности, и требует ровно такое же количество переменных для распределения этих элементов. Если же количество переменных и элементов не совпадает, будет выдана ошибка.

Пример:

```
s = '123' # строка
s0, s1, s2 = s
print(s0, s1, s2, sep=',')

s = [1, 2, 3] # список
s0, s1, s2 = s
print(s0, s1, s2, sep=',')

s = [1, 2, 3]
s0, s1 = s # количество переменных не совпадает с количеством элементов
print(s0, s1, s2, sep=',')
```

```
1,2,3
1,2,3
builtins.ValueError: too many values to unpack (expected 2)
```

Расширенная распаковка

Позволяет распаковать последовательность в случае, если количество переменных слева меньше, чем количество элементов в последовательности справа. Данный способ используется, как правило, тогда, когда количество элементов в последовательности не известно. Одна из переменных будет являться списком. Эта переменная должна быть объявленна с указанием слева от имени символа «*» (например: *a). Важно понимать, что список распаковывается в порядке следования элементов, т. е. результат зависит от расположения переменых слева от знака равенства. Внимательно рассмотрите пример ниже.

Пример:

```
s = [10, 2, 13, 23, 4, 7]
s0, s1, *s2 = s
print(s0, s1, s2, sep=', ')

s0, *s1, s2 = s
print(s0, s1, s2, sep=', ')
```

```
*s0, s1, s2 = s
print(s0, s1, s2, sep=', ')

s0, s1, *s2 = range(10)
print(s0, s1, s2, sep=', ')
```

Результат:

```
10, 2, [13, 23, 4, 7]
10, [2, 13, 23, 4], 7
[10, 2, 13, 23], 4, 7
0, 1, [2, 3, 4, 5, 6, 7, 8, 9]
```

Анонимные переменные в распаковке

При распаковке последовательностей возможна ситуация, когда не нужно использовать все ее элементы. В этом случае прибегают к помощи анонимных переменных. Анонимную переменную обозначают при помощи знака нижнее подчеркивание «_» (можно использовать и двойное подчеркивание). Одна анонимная переменная позволяет пропустить ровно одно значение. Если нужно пропустить несколько значений, идущих не по порядку, для каждого значения нужно использовать свою анонимную переменную. Если же нужно пропустить несколько подряд идущих значений, то перед анонимной переменной ставится знак «*». При использовании анонимной переменной без знака «*» нужно следить за тем, чтобы количество переменных слева от знака равенства совпадало с количеством элементов в последовательности справа.

Пример:

```
s = [10, 2, 7, 4, 5]
s0, _, s1, _, s2 = s
print(s0, s1, s2,  sep=', ')

s = [10, 2, 7, 4, 5]
s0, *_, s1 = s
print(s0, s1,  sep=', ')
```

```
10, 7, 5
10, 5
```

Распаковка последовательности при выводе

Вывод последовательности целиком с помощью функции **print** показывает ее как целую синтаксическую конструкцию, демонстрирующую тип последовательности. Для вывода всех элементов последовательности с нужным разделителем без использования цикла можно использовать распаковку.

Пример:

```
s = [10, 2, 7, 4, 5]
print(s)
print(*s)
print(*s, sep='-')
```

```
[10, 2, 7, 4, 5]
10 2 7 4 5
```

```
10-2-7-4-5
```

При распаковке словаря выводятся только ключи.

Пример:

```
a = dict(name='Катя', age=20, city='Новосибирск')
print(a)
print(*a)
```

```
{'city': 'Новосибирск', 'age': 20, 'name': 'Катя'}
city age name
```

Распаковка в цикле

Пример:

```
students = [['Вася', 18, 178], ['Петя', 22, 184], ['Катя', 20, 165]]
for name, age, height in students:
    print('Имя:', name, 'Возраст:', age, 'Рост:', height)
```

```
Имя: Вася Возраст: 18 Рост: 178
Имя: Петя Возраст: 22 Рост: 184
Имя: Катя Возраст: 20 Рост: 165
```

Примеры применения распаковки

1. Использование **enumerate** при распаковке в цикле.

```
students = [['Вася', 18, 178], ['Петя', 22, 184], ['Катя', 20, 165]]
for index, (name, age, height) in enumerate(students):
    print(index, 'Имя:', name, 'Возраст:', age, 'Рост:', height)
```

```
0 Имя: Вася Возраст: 18 Рост: 178
1 Имя: Петя Возраст: 22 Рост: 184
2 Имя: Катя Возраст: 20 Рост: 165
```

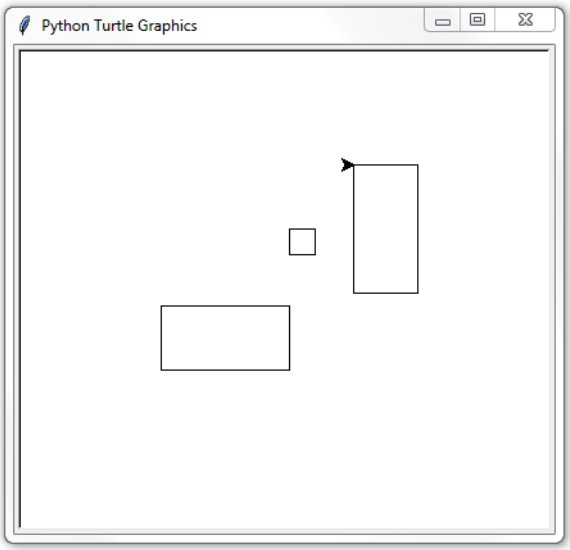
Стоит обратить внимание, что переменные для распаковки объединены в группу с помощью круглых скобок. Такое объединение, в данном случае, является обязательным.

2. Рисование прямоугольников по параметрам, взятым из списка, с помощью Черепахи из модуля **turtle**.

```
import turtle

def rectangle(x, y, w, h):
    turtle.up()
    turtle.goto(x, y)
    turtle.down()
    for i in range(2):
        turtle.forward(w)
        turtle.right(90)
        turtle.forward(h)
        turtle.right(90)
    turtle.up()

rects = [[-100, -10, 100, 50], [0, 50, 20, 20], [50, 100, 50, 100]]
for x, y, w, h in rects:
    rectangle(x, y, w, h)
```



3. Транспонирование матрицы с помощью **map**, **zip** и распаковки.

```
m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(list(map(list, zip(*m))))
```

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

14.2. Обработка исключений

Исключения – это извещения интерпретатора при возникновении ошибки в программе или при наступлении какого-либо события. Если в коде не реализована обработка исключений, то при возникновении ошибки выполнение программы прерывается. Исключения позволяют не прерывать работу программы, а продолжать ее работу исходя из результата обработки ошибки.

Существует три типа ошибок в программе:

- **Синтаксические** – это ошибки в имени оператора или функции, отсутствие открывающей или закрывающей кавычки и т. д., т. е. ошибки в синтаксисе языка. При возникновении ошибки данного типа программа вообще не запустится, а интерпретатор сообщит о возникшей проблеме.

Пример:

```
Print('Привет Python')
```

Получим ошибку: ***NameError: name 'Print' is not defined***.

- **Логические** – ошибки в логике написания программы. Данные ошибки не определяются интерпретатором. Найти такую ошибку можно только по результатам выполнения программы, т. е. программа будет интерпретироваться, но результат ее выполнения будет неверный.

Пример:

Нужно написать функцию нахождения среднего арифметического двух чисел. Был написан следующий код:

```
def sred(a, b):
    return a + b / 2

print(sred(3, 5))
```

Программа запустится и выполнится, но результат в общем случае получится неверный, так как не учтен приоритет выполнения операций и деление выполнится раньше сложения.

- **Ошибки времени выполнения** – возникают во время работы программы при получении ей данных. Причиной являются события, непредусмотренные программистом.

Пример:

Ошибка деления на ноль.

```
a = int(input())
print(1 / a)
```

Программа запустится, но при введении пользователем нуля возникнет ошибка.

Виды исключений

Исключений в Python много, и они меняются от версии к версии. Более того, пользователь сам может создавать исключения.

Этот курс не подразумевает полное изучение исключений. Здесь мы рассматриваем только базовые, наиболее часто встречающиеся исключения при изучении языка программирования и решении простых задач.

Список некоторых исключений

Список приведен с соблюдением иерархии ошибок.

- **ArithmeticError** – арифметическая ошибка.
  - **FloatingPointError** – появляется при неудачном выполнении операции с плавающей запятой. На практике встречается нечасто.
  - **OverflowError** – возникает, когда результат арифметической операции слишком велик для представления.
  - **ZeroDivisionError** – деление на ноль.
- **EOFError** – функция встретила конец файла и не смогла прочитать то, что должна была.
- **ImportError** – не выполнено импортирование модуля или его атрибута.
- **LookupError** – некорректный индекс или ключ.
  - **IndexError** – индекс не входит в диапазон элементов.
  - **KeyError** – несуществующий ключ, например, в словаре.
- **NameError** – не найдено переменной с таким именем.
- **UnboundLocalError** – сделана ссылка на локальную переменную в функции, но переменная ранее не определена.
- **OSError** – ошибка, связанная с системой.

  - **FileExistsError** – попытка создания файла или директории, которая уже существует.
  - **FileNotFoundError** – файл или директория не существует.
- **SyntaxError** – синтаксическая ошибка.
- **TypeError** – операция применена к объекту несоответствующего типа.
- **ValueError** – функция получает аргумент правильного типа, но некорректного значения.
- **Warning** – предупреждение.
- **RuntimeError** – возникает, когда исключение не попадает ни под одну из других категорий.

Обработка исключений

Для обработки исключений предназначена инструкция **try…except…else…finaly**.

Формат инструкции

try:
<блок, в котором перехватываются исключения>
*[**except** [<исключение 1>[**as** <объект исключения>]]:*
<блок, выполняемый при возникновении исключения>
[. . .
***except** [<исключение>[**as** <объект исключения>]]:*
<блок, выполняемый при возникновении исключения]]>
*[**else**:*
<блок, выполняемый, если исключение не возникло]>
*[**finally**:*
<блок, выполняемый в любом случае]>

Пример:

Решение задачи получения обратного числа для введенного с обработкой исключения деления на ноль будет выглядеть следующим образом:

```
a = int(input('Введите целое число: '))
try:
    b = 1 / a
except ZeroDivisionError:
    print('Попытка деления на ноль!')
else:
    print(b)
```

При вводе целого числа, отличного от нуля, получим обратное число, а при вводе нуля получим сообщение «Попытка деления на ноль!».

Но эта ошибка не единственная, которая может возникнуть в данной задаче. Пользователь может вместо числа ввести букву или другой символ. Тогда обработчик будет уже немного сложнее.

В данном случае ошибка возникнет при передаче функции **int** неподходящих по типу данных. Вызовется исключение **ValueError**.

```
a = input('Введите целое число: ')
try:
    a = int(a)
except ValueError:
    print('Вы ввели неподходящие данные, можно вводить только числа.')
else:
    try:
        b = 1 / a
    except ZeroDivisionError:
        print('Попытка деления на ноль!')
    else:
        print(b)
```

Обратите внимание, в данном случае мы использовали два блока для обработки исключений.

Эту же задачу можно решить, используя только один блок.

```
a = input('Введите целое число: ')

try:
    b = int(a)
    c = 1 / b
except ValueError:
    print('Вы ввели неподходящие данные, можно вводить только числа.')
except ZeroDivisionError:
    print('Попытка деления на ноль!')
else:
    print(c)
```

Здесь, если введем число, то получим обратное число, если введем ноль, то получим сообщение «*Попытка деления на ноль!*», а если введем символы, не являющиеся числовыми, то получим сообщение «*Вы ввели неподходящие данные, можно вводить только числа.*».

Если нам не нужно на каждое исключение выполнять различные действия, а в любом случае при возникновении исключения выполнить что-то одно, то исключения можно указывать через запятую в скобках в одном блоке.

```
a = input('Введите целое число: ')
try:
    b = int(a)
    c = 1 / b
except (ValueError, ZeroDivisionError):
    print('Вы ввели неподходящие данные.')
else:
    print(c)
```

Получить информацию о том, какое исключение сработало, можно при помощи параметра, указываемого оператором **as** в конструкции **except**.

```
a = input('Введите целое число: ')
try:
    b = int(a)
    c = 1 / b
except (ValueError, ZeroDivisionError) as err:
    print('Вы ввели неподходящие данные.')
    print(err)
else:
    print(c)
```

Если введем число, отличное от нуля, получим обратное число.

Если введем ноль, в результате получим:

```
Вы ввели неподходящие данные.
division by zero
```

Если введем нечисловой символ, допустим «*k*», в результате получим:

```
Вы ввели неподходящие данные.
invalid literal for int() with base 10: 'k'
```

Если в конструкции **except** не указать тип исключения, то такой блок перехватит все исключения. Но в программе лучше это не использовать, так как можно перехватить исключение, которое является просто сигналом для системы, а не ошибкой.

Как уже было сказано выше, инструкции в блоке **else** сработают только в том случае, если не было вызвано исключение. Но если возникает необходимость выполнить какие-либо завершающие действия в независимости от того, есть ошибка или нет, используется блок **finally**.

Пример:

```
c = int(input())
try:
    a = 5 / c
except ZeroDivisionError:
    print('Попытка деления на ноль')
else:
    print(a)
finally:
    print('Выполняется независимо от того, есть исключение или нет')
```

Результат выполнения при отсутствии исключения:

1. выведется значение переменной *a*;
2. выведется сообщение: «*Выполняется независимо от того, есть исключение или нет*».

Результат выполнения при возникновении исключения:

1. выведется сообщение: «*Попытка деления на ноль*»;
2. выведется сообщение: «*Выполняется независимо от того, есть исключение или нет*».