Python

Урок 3. Функциональное программирование

Ввод данных пользователя: input()

Для ввода данных с клавиатуры испльзуется встроенная функция input(). Она также выводит подсказку, если вызвана с аргументом.

```
>>> input('Введите слово: ')
Введите слово: Привет!
'Привет!'
```

Интерактивные циклы

Допустим, вам необходимо написать цикл, который будет считывать одну или более строк, введенных пользователем с клавиатуры, и выводить их обратно на экран. Другими словами, вам нужно написать классический цикл, выполняющий операции чтения/ вычисления/вывода. Для реализации таких интерактивных циклов используется типичный шаблон, который выглядит так:

```
while True:
    reply = input('Enter text:')
    if reply == 'stop': break
    print(reply.upper())
```

Обработка ошибок проверкой ввода

Допустим, вы хотите поработать с введенными данными как с числом. В этом случае проверку введенных данных можно осуществить с помощью метода строки isdigit():

```
while True:
    reply = input('Enter text:')
    if reply == 'stop':
        break
    elif not reply.isdigit():
        print('Bad!' * 8)
    else:
        print(int(reply) ** 2)
print 'Bye'
```

Исключения

Более универсальный способ обработки введенной строки состоит в том, чтобы перехватывать и обрабатывать ошибки с помощью инструкции try:

```
while True:
    reply = input('Enter text:')
    if reply == 'stop':
        break
    try:
        num = int(reply)
    except:
        print('Bad!' * 8)
    else:
        print(int(reply) ** 2) print 'Bye'
```

Эта версия работает точно так же, как и предыдущая, только здесь мы заменили явную проверку наличия ошибки программным кодом, который предполагает, что преобразование будет выполнено и выполняет обработку исключения, если такое преобразование невозможно.

Эта инструкция try состоит из слова try, вслед за которым следует основной блок кода (действие, которые мы пытаемся выполнить), с последующей частью except, где располагается программный код обработки исключения. Далее следует часть else, программный код которой выполняется, если в части try исключение не возникло.

Интерпретатор сначала выполняет часть try, затем выполняет либо часть except (если возникло исключение), либо часть else (если исключение не возникло).

Анонимные функции (lambda)

В Python имеется возможность создавать объекты функций в форме выражений. Подобно инструкции def это выражение создает функцию, которая будет вызываться позднее, но в отличие от инструкции def, выражение возвращает функцию, а не связывает ее с именем. Именно поэтому lambda-выражения иногда называют анонимными (то есть безымянными) функциями. На практике они часто используются, как способ получить встроенную функцию или отложить выполнение фрагмента программного кода.

В общем виде lambda-выражение состоит из ключевого слова lambda, за которым следуют один или более аргументов и далее, вслед за двоеточием, находится выражение:

```
lambda argument1, argument2,... argumentN : выражение,
 использующее аргументы
```

```
>>> f = lambda x, y, z: x + y + z
>>> f(2, 3, 4)
9
```

В lambda-выражениях можно использовать аргументы со значениями по умолчанию:

```
>>> x = lambda a="fee", b="fie", c="foe": a + b + c
>>> x("wee")
'weefiefoe'
```

lambda-выражения очень удобны для создания очень маленьких функций. Они не являются предметом первой необходимости (вы всегда сможете вместо них использовать инструкции def), но они позволяют упростить сценарии, где требуется внедрять небольшие фрагменты программного кода.

Встроенные функции map, filter, reduce

map

Функция map выполняет отображение функции на последовательность.

Очень часто встречающейся задачей является применение некоторой операции к каждому элементу в списке или в другой последовательности и сборе полученных результатов.

Например, обновление всех счетчиков в списке может быть выполнено с помощью простого цикла for:

```
>>> counters = [1, 2, 3, 4]
>>> updated = []
>>> for x in counters:
...     updated.append(x + 10) # Прибавить 10 к каждому элементу
...
>>> updated
[11, 12, 13, 14]
```

Такие операции встречаются достаточно часто, язык Python предоставляет встроенную функцию, которая выполняет большую часть этой работы. Функция map применяет указанную функцию к каждому элементу последовательности и возвращает список, содержащий результаты всех вызовов функции. Например:

```
# Функция, которая должна быть вызвана
>>>  def inc(x): return x + 10
>>>  map(inc, counters) # Сбор результатов
[11, 12, 13, 14]
```

Функция map в Python 3.0 возвращает итерируемый объект, поэтому для вывода всех результатов в интерактивной оболочке мы используем функцию list; этого не требуется в Python 2.7.

Функция map ожидает получить в первом аргументе функцию, поэтому здесь часто можно встретить lambda-выражения:

```
>>> list(map((lambda x: x + 3), counters)) # Выражение-функция
[4, 5, 6, 7]
```

filter

Функция filter отфильтровывает элементы последовательности с помощью функции, выполняющей проверку.

Например, следующий вызов функции filter отбирает элементы последовательности больше нуля:

```
>>> list(range(-5, 5))
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]

>>> filter((lambda x: x > 0), range(-5, 5))
[1, 2, 3, 4]
```

Элементы последовательности, для которых применяемая функция возвращает истину, добавляются в список результатов. Как и map, функция filter является примерным эквивалентом цикла for, только она – встроенная функция и обладает высокой скоростью выполнения:

```
>>> res = []
>>> for x in range(-5, 5):
...     if x>0:
...         res.append(x)
...
>>> res
 [1, 2, 3, 4]
```

reduce

Функция reduce в Python 2.6 была простой встроенной функцией, но в версии 3.0 она была перемещена в модуль functools и стала более сложной. Она принимает итератор, но сама возвращает не итератор, а одиночный объект. Ниже приводятся два вызова функции reduce, которые вычисляют сумму и произведение элементов списка:

```
# В Python 3.0 требуется выполнить импортирование
>>> from functools import reduce
>>> reduce((lambda x, y: x + y), [1, 2, 3, 4])
10
 >>> reduce((lambda x, y: x * y), [1, 2, 3, 4])
24
```

На каждом шаге функция reduce передает текущую сумму или произведение вместе со следующим элементом списка lambda-функции. По умолчанию первый элемент последовательности принимается в качестве начального значения.

Ниже приводится цикл for, эквивалентный первому вызову, с жестко заданной операцией сложения внутри цикла:

```
>>> L = [1,2,3,4]
>>> res = L[0]
>>> for x in L[1:]:
...         res = res + x
...
>>> res
10
```