при поддержке
{а}вижн

Лекция 10

Перевод и адаптация материалов Джона ДеНиро (John DeNero). Используется с разрешения автора.

Последовательности

Абстракция последовательности

красный, оранжевый, желтый, зелёный, голубой, синий, фиолетовый.

0 , 1 , 2 , 3 , 4 , 5 , 6 .

Не существует единого класса или единой абстракции данных для произвольной последовательности (и в Python’е, и в других языках).

Абстракция последовательности — это набор поведений:

Длина. Последовательность имеет конечную длину.

Выбор элемента. Элементам последовательности соответствуют индексы — порядковые номера (неотрицательные целые начинающиеся с 0).

Для выбора элементов можно применять встроенный синтаксис или использовать функции.

Список (list) — это встроенная разновидность абстракции последовательности.

Списки

```
['Пример']
```

Списки — это последовательности

```
>>> digits = [1, 8, 2, 8]
>>> len(digits)
4
>>> digits[3]
8
```

Длина. Последовательность имеет конечную длину.

Выбор элемента. Элементам последовательности соответствуют индексы — порядковые номера (неотрицательные целые начинающиеся с 0).

```
>>> [2, 7] + digits * 2
[2, 7, 1, 8, 2, 8, 1, 8, 2, 8]

>>> pairs = [[10, 20], [30, 40]]
>>> pairs[1]
[30, 40]
>>> pairs[1][0]
30
```

```
>>> 1 in digits
True
>>> 8 in digits
True
>>> 5 not in digits
True
>>> not(5 in digits)
True
```

Инструкция for

(Пример)

Итерация по последовательности

```
def count(s, value):
    total = 0
    for element in s:
        if element == value:
            total = total + 1
    return total
```

Связанное имя находится в первом фрейме текущего окружения (не в новом фрейме)

Исполнение инструкции for

```
for <имя> in <выражение>:
    <набор>
```

1. Вычислить заголовочное <выражение>, результат должен быть итерируемым значением, то есть последовательностью.

2. Для каждого элемента этой последовательности, по порядку:

   A. Связать <имя> с этим элементом в текущем фрейме.

   B. Выполнить <набор>.

Распаковка последовательностей инструкцией for

Последовательность последовательностей фиксированной длины

```
>>> pairs = [[1, 2], [2, 2], [3, 2], [4, 4]]

>>> same_count = 0
```

Имя для каждого элемента последовательности фиксированной длины

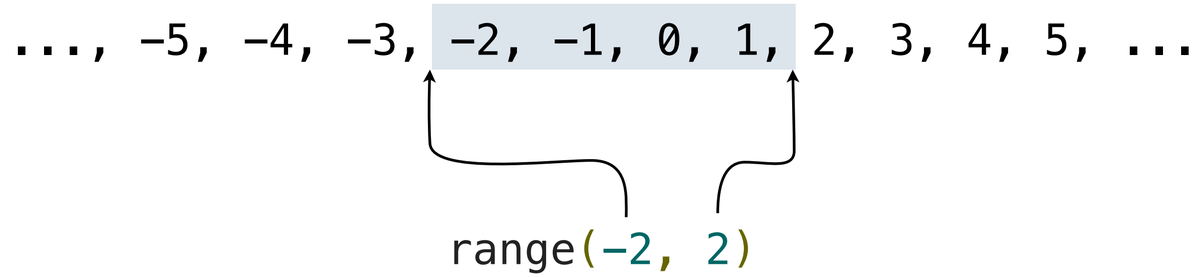
```
>>> for x, y in pairs:
...     if x == y:
...         same_count = same_count + 1

>>> same_count
2
```

Каждое имя связывается со значением — также как при множественном присвоении.

Диапазон (range)

Тип range (диапазон)

Диапазон — это последовательность последовательных целых чисел.*



Длина: конечное значение – начальное значение

Выбор элемента: начальное значение + индекс

(Пример)

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

Конструктор списка

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

По-умолчанию, диапазон начинается с нуля

* Диапазоны также могут представлять более сложные последовательности целых чисел.

Списковые включения

```
>>> letters = ['а', 'б', 'в', 'д', 'е', 'ж', 'м', 'и', 'о', ‘э']
>>> [letters[i] for i in [3, 4, 6, 8]]
```

```
['д', 'е', 'м', 'о']
```

Списковые включения

```
[<отображающее выраж.> for <имя> in <итерируемое выраж.> if <фильтрующее выраж.>]
```

Короткая версия: `[<отображающее выраж.> for <имя> in <итерируемое выраж.>]`

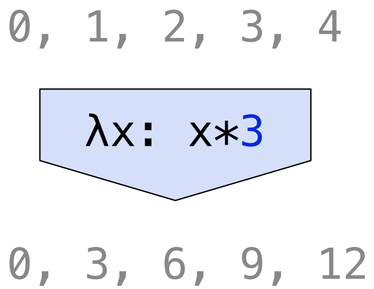
Сложное выражение, результат которого является списком и получается по правилам:

1. Добавить новый фрейм полагая текущий фрейм родительским.
2. Создать пустой *результирующий список* который будет значением выражения.
3. Для каждого элемента в итерируемом значении выражения `<итерируемое выраж.>`:
   A. Во фрейме из шага 1 связать `<имя>`
   B. Если `<фильтрующее выражение>` имеет истинное значение, тогда значение `<отображающего выражения>` заносится в *результирующий список*.

Функции высшего порядка для работы с последовательностями

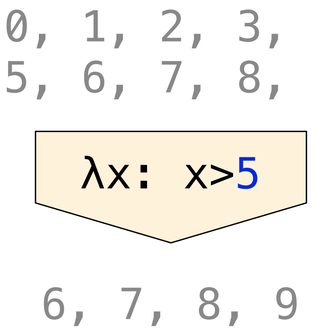
Функции, которые применяют генераторы списков

```
def apply_to_all(map_fn, s):
    """Применяет map_fn к каждому элементу s.


    """
    return [map_fn(x) for x in s]
```



То же количество других элементов

```
def keep_if(filter_fn, s):
    """Возвращает список элементов s для которых filter_fn(x) является истиной.


    """
    return [x for x in s if filter_fn(x)]
```



Меньшее количество тех же элементов

Свертка списка к значению

```
def reduce(reduce_fn, s, initial):
    """Попарно объединяет элементы s используя reduce_fn, начиная с initial.

    То есть, reduce(mul, [2, 4, 8], 1) эквивалентен mul(mul(mul(1, 2), 4), 8).

    >>> reduce(mul, [2, 4, 8], 1)
    64
    """
    reduced = initial
    for x in s:
        reduced = reduce_fn(reduced, x)
    return reduced
```

`reduce_fn` — это ...

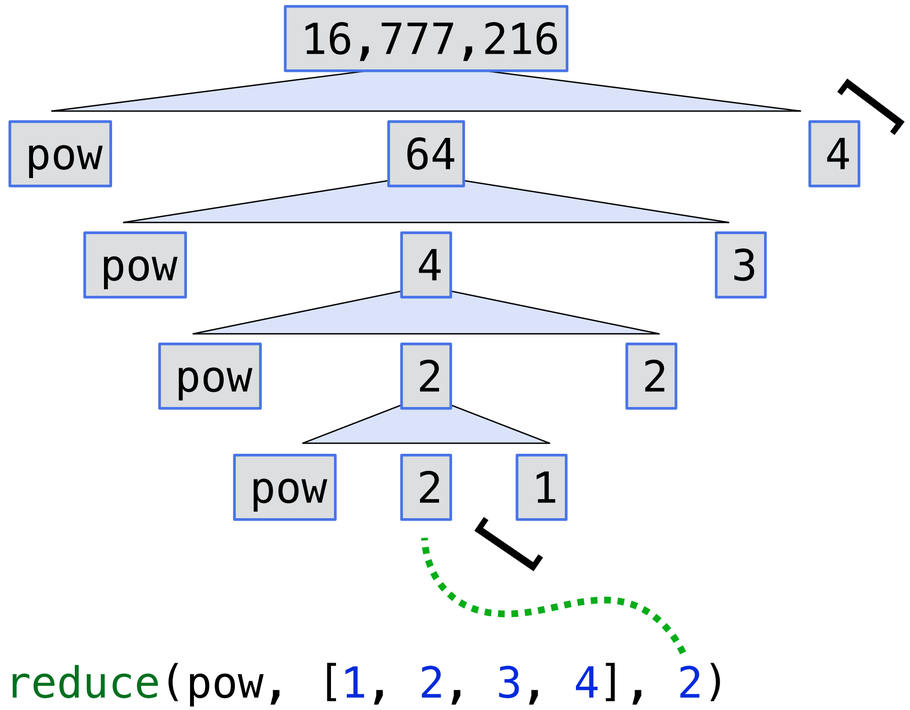
функция двух аргументов

`s` — это ...

последовательность значений (второй аргумент)

`initial` — это ...

начальное значение (первый аргумент)



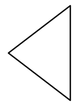
`reduce(pow, [1, 2, 3, 4], 2)`

(Пример)

Принятые имена функций высшего порядка

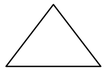
`apply_to_all` обычно называют `map`

`keep_if` обычно называют `filter`



`map` и `filter` встроены в Python, но они не возвращают списки

`reduce` обычно называют `reduce` (но иногда `fold` о `accumulate`)



`reduce` – встроен в стандартную библиотеку, в модуль `functools`.

Большинство программистов на Python просто используют генераторы

Строки

Строки — это абстракция

Представление данных:

```
'200'      '1.2e-5'      'False'      '(1, 2)'
```

Представление языка:

```
"""Ученый, сверстник Галилея,
был Галилея не глупее.
Он знал, что вертится земля,
но у него была семья.
"""
```

Представление программ:

```
'curry = lambda f: lambda x: lambda y: f(x, y)'
```

(Пример)

Три формы строк

```
>>> 'Я строка!'
'Я строка!'

>>> "А я — об'явление"
"А я — об'явление"

>>> ‘您好'
'您好'


>>> """Дзен пайтона утверждает —
читаемость имеет значение.
Подробнее: import this."""
'Дзен пайтона утверждает —\nчитаемость имеет значение.\nПодробнее: import this.'
```

Одинарные и двойные кавычки эквивалентны

Заставляет игнорировать следующий символ

Приводит к переводу строку

Строки — это последовательности

Длинна и выбор элемента одинаковы для всех последовательностей

```
>>> city = 'Ярославль'
>>> len(city)
9
>>> city[3]
'с'
```

Аккуратно: элемент строки — это тоже строка, но только с одним элементом!

Однако операторы «in» и «not in» работают с подстроками.

```
>>> 'но выпил' in "и немедленно выпил"
True
>>> 234 in [1, 2, 3, 4, 5]
False
>>> [2, 3, 4] in [1, 2, 3, 4, 5]
False
```

При работе со строками обычно большее внимание уделяется словам, чем буквам.

Словари

```
{'Дем': 0}
```

Словари

Словарь — это **неупорядоченная** коллекция пар ключ–значение.

Для ключей действуют два ограничения:

- Ключ словаря **не может быть** ни списком, ни словарем (ни любым *изменяемым типом*).
- Два ключа **не могут быть равны;** должно быть не менее одного значения на ключ.

Первое ограничение связано с реализацией словарей в Python.

Второе ограничение следует из абстракции словаря.

Если с одним ключом нужно связать множество значений, то их надо хранить в виде последовательности.