PYTHON

Лекция 4

ПЛАН ЛЕКЦИИ

- Итерирование
- Списки

ФОРМЫ ИТЕРИРОВАНИЯ

- Циклы
- Списочные выражения
- Генераторы

ЦИКЛ FOR

- Перебираем элементы последовательности
- Последовательность - то, что может перебираться поэлементно
- Строка - простейший пример последовательности
- Перебор идет по элементам, не по индексам
- С индексами работать тоже можно - но это уже надстройка

ПРИМЕР

```
1 for c in 'hello':
2     print("code:", ord(c))
```

ЦИКЛ ПО ЧИСЛАМ

- Справа от in - вызываем range
- Это встроенная функция
- Она же - конструктор встроенного типа
- От 1 до 3 параметров
- Интерпретация параметров - почти как у слайсов
- Разница - в понимании минусов

ПРИМЕР

```
 1 for i in range(10): # 0, 1, 2, ...., 9
 2     print(i, i * i)
 3
 4 for i in range(5, 10): # 5, 6, 7, 8, 9
 5     print(i, i * i)
 6
 7 for i in range(1, 10, 2): # 1, 3, 5, 7, 9
 8     print(i, i * i)
 9
10 for i in range(10, -1, -2):
11     # 10, 8, 6, 4, 2, 0 - не проблемы с -1
12     print(i, i * i)
```

ТАК НЕ НАДО

```
1 s = 'hello'
2 for i in range(len(s)):
3     print(i, ord(s[c]))
```

ПРЕОБРАЗОВАНИЯ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

- Есть функции, которые преобразуют последовательности
- Берут на вход одну и возвращают другую
- Одна из них - enumerate
- Возвращает последовательности пар
- Номер элемента и элемент

ВОТ ТАК ЛУЧШЕ

```
1 s = 'hello'
2 for i, c in enumerate(len(s)):
3     print(i, ord(s[c]))
```

ГРУППОВОЕ ПРИСВАИВАНИЕ

- in означает присваивание в каждой итерации
- Той переменной, которая слева от in
- А у нас две переменных
- Это называется групповым присваиванием

ГРУППОВОЕ ПРИСВАИВАНИЕ

ГРУППОВОЕ ПРИСВАИВАНИЕ

```
1 a, b = '%^'
2 print(a)  # %
3 print(b)  # ^
4
5 a, b, c = enumerate(range(5, 10, 2))
6 print(a)  # (0, 5)
7 print(b)  # (1, 7)
8 print(c)  # (2, 9)
```

ЛЕНИВЫЕ ГЕНЕРАТОРЫ

- В первом приближении можно так думать, что enumerate формирует промужуточную последовательность
- Но это не совсем так
- Он при каждой итерации продвигается по исходной последовательности
- И создает новую пару
- Можно создавать свои ленивые генераторы

ВСПОМОГАТЕЛЬНЫЕ КОНСТРУКЦИИ

- break/continue - как везде
- Относятся к ближайшему по вложенности
- Из необычного - есть else - ветка у цикла
- Исполняется, когда цикл закончился без break

ПРИМЕР

```
1 cnt = 0
2 for i, c in enumerate(input()):
3     if c in 'qwerty':
4         cnt += 1
5     if cnt == 5:
6         print('Found')
7         break
8 else:
9     print('Not found')
```

ДРУГИЕ КОЛЛЕКЦИИ

- Списки
- Кортежи
- Множества
- Словари
- Многое другое

СПИСКИ

- Последовательность ссылок на объекты
- Нумерованная
- Материализованная
- Изменяемая

СОЗДАНИЕ

- Литерал - элементы через запятую
- В квадратных скобках
- Через операции: вырезки, сложение, умножение на число
- Функция list

ПРИМЕР

```
1 data_1 = []
2 data_2 = [123]
3 data_3 = ['vasya', 'dima', 'kolya']
4 data_4 = data_2 + data_3
5 print(data_4)
6 data_4 = data_4 * 3
7 print(data_4)
8 data_5 = [data_1, data_2]
9 print(data_5)
```

МНОГОМЕРНОСТЬ

- Явочным порядком
- Можно создать список списков
- Точнее - список ссылок на списки
- Встроенного механизма в языке нет
- Только в библиотеках (numpy)

ИНДЕКСЫ И ВЫРЕЗКИ

- Можно получить элемент по индексу
- Или по вырезке - все как со строками
- Вырезка как значение создает новый список
- Можно присваивать по индексу
- По вырезке - тоже, но про это попозже

ПРИМЕР

```
1 data_3 = ['vasya', 'dima', 'kolya']
2 data_4 = data_2 + data_3
3 data_4 = data_4 * 3
4 print(data_4[0])
5 print(data_4[2:-3])
6 print(data_4[:])
```

ПРИМЕР

```
 1 data_3 = ['vasya', 'dima', 'kolya']
 2 data_4 = data_2 + data_3
 3 data_4 = data_4 * 3
 4 data_4[5] = 'hello'
 5 print(data_4)
 6 data_5 = data_4[:]
 7 data_6 = data_4[:]
 8 data_5[1] = 'hello'
 9 data_6[1] = 'world'
10 print(data_4)
11 print(data_5)
12 print(data_6)
```

ФУНКЦИЯ LIST

```
1 print(list('hello'))
2 print(list(enumerate('hello')))
3 print(enumerate('hello')) # для сравнения
4 print(list(range(10)))
5 # print(list(range(1000000000000)))  - так не надо
```

ОПАСНОСТЬ

```
1 # Наивная инициализация двумерного списка 5 x 6
2 data = [[None] * 6] * 5
3 print(data)
4 data[0][1] = 10
5 print(data)  # !!!!!
```

ОПАСНОСТЬ

```
1 # Правильная, но громоздкая инициализация
2 # двумерного списка 5 x 6
3 data = [None] * 5
4 for i, _ in enumerate(data):
5     data[i] = [None] * 6
```

ВАЖНЫЕ МЕТОДЫ

- append - добавляет элемент в список
- extend - расширяет список другим списком
- Эффективные способы расширять список
- += - тоже (для списка)

ВСПОМНИМ СТРОКИ

- += для строк - не "extend"
- Возможны оптимизации
- Я бы не полагался

POP

- Без параметров - убирает элемент из конца. Недорого
- Комбинация append(v)/pop() - дает стек
- С параметром - удаляет элемент из середины
- В среднем - дорого

INSERT/REMOVE/CLEAR

- insert - вставить элемент в позицию. Дорого
- remove - удалить первый с данным значением. Дорого
- Если не нашли - исключение
- clear - очистка текущего списка

INDEX

- Напоминает строку, но есть отличия
- В строке ищем подстроку, здесь - элемент
- Многое упрощает
- Нет find

ОПАСНОСТЬ

```
 1 print('123'.index('23'))  # 1
 2 #print(list('123').index(list('23'))) # исключение
 3 #print(list('123').index('23')) # исключение
 4 #print(list('123').index(list('2'))) # исключение
 5 print(list('123').index('2')) # 1
 6
 7 print('123'.index('', 2))  # 2
 8 print('123'.index('', 3))  # 3
 9 #print('123'.index('', 4))  # исключение
10
11 #print(list('123').index([], 2))  # исключение
12 #print(list('123').index([], 3))  # исключение
13 #print(list('123').index([], 4))  # исключение
```

SORT

- Сортирует in-place
- Важно, чтобы сравнения были определены
- Можно задать функцию-ключ
- Например, str.lower - если сортируем строчки без учета регистра

ОПАСНОСТЬ

```python
import random

data = ['123', 'qwerty', 'Hello', 'abc', 'ABC']
data.sort()
print(data)
# ['123', 'ABC', 'Hello', 'abc', 'qwerty']
data.sort(key=str.lower)
print(data)
# ['123', 'ABC', 'abc', 'Hello', 'qwerty']

# ............
```

ОПАСНОСТЬ

```
1 # ..............
2
3 data.sort(key=lambda v: random.random())
4 print(data)
5 # плохой ключ - непредсказуемый порядок
6 data.sort(key=lambda v: random.random())
7 print(data)
8 # для перемешивания есть random.shuffle - он эффективнее
```

ПОДЕТАЛЬНЕЕ

- Есть (в принципе) два способа настройки сортировки - ключи и компараторы
- Ключ - функция с одним параметром, возвращающая значения для сравнения
- Компаратор - функция с двумя параметрами, возвращающая их статус сравнения
- При использовании ключа - его можно посчитать один раз для каждого элемента
- Компаратор - используется на каждом сравнении

ПОДЕТАЛЬНЕЕ

- Python2 использовал компараторы
- Python3 перешел на ключи
- Из ключа легко сделать компаратор
- Из компаратора ключ - посложнее в общем случае

ПОДЕТАЛЬНЕЕ

- Python предлагает преобразователь - functools.cmp_to_key()
- Но если компаратор дорогой - это не сэкономит времени
- Все равно будут вызовы компаратора на каждом сравнении
- Классический пример - строки с длинным общим префиксом

ПОДЕТАЛЬНЕЕ

- Еще можно заказать в обратном порядке (reversed=True)
- Есть стандартная функция sorted
- Первый параметр - любая коллекция, дальше - ключ и reversed
- Возвращает отсортированный список

ПРОЧЕЕ

- copy() - неглубокое копирование
- reverse() - переворачивание in-place
- count(x) - поэлементный подсчет, без start, end
- clear() - in-place опустошение

СПИСОЧНЫЕ ВЫРАЖЕНИЯ

- "List comprehension"
- Короткая запись преобразования коллекции
- Базовая идея: многие задачи можно выразить в виде цепочки типовых преобразований
- Базис: фильтрация, отображение, свертка

ФИЛЬТРАЦИЯ

- Пройдя по коллекции, отбросим ненужное
- Ненужность определим по значению функции вызванной над элементом
- Например, по списку пользователей отберем тех, кто зарегестрирован в этом году

ОТОБРАЖЕНИЕ

- Для каждого элемента определим новое значение
- Логика определения нового значения задается функцией-параметром
- Пройдем по коллекции, породив новую коллекцию, преобразовав каждый элемент
- Например по списку пользователей получим пару (id, количество постов)

СВЕРТКА

- Возьмем начальное значение
- Поместим в условную переменную-аккумулятор
- Будем перебирать элементы коллекции, используя функцию с двумя параметрами
- На каждом элементе передадим в функцию аккумулятор и очередной элемент

СВЕРТКА

- Возвращенное функцией значение сделаем новым аккумулятором
- Результат свертки - финальное значение аккумулятора
- Примеры: count, sum, max, min

СПИСОЧНОЕ ВЫРЕЖЕНИЕ

- Списочное выражение позволяет компактно записать отображение с фильтрацией
- Описывает проход по коллекции
- И для каждого элемента позволяет задать отображение и фильтр

ПРИМЕР

```
1 print([len(v) for v  in input().split()])
2 print([v[::-1] for v  in input().split() if len(v) > 5])
3 print(
4   [v for v  in input().split() if v and v[0] in 'aeouiy']
5 )
```

ИДИОМА

- Хороший стиль - писать цепочки преобразований
- Плохой стиль - выписывать цикл, в котором делается append под if
- Особенно если не делается больше ничего

ДВУМЕРНЫЙ МАССИВ

```
1 data = [[None] * 5 for _ in range(6)]
2 # Инициализация здорового человека
```