Лицей Академии Яндекса

АВРОРА
центр развития талантов

Основы программирования на языке Python

ПОЛУПАНОВ Дмитрий Васильевич
к.т.н., доцент

А на прошлом уроке было…

Разбор ошибок и задание на закрепление прошедшего материала

Ещё одна коллекция - кортеж

Кортеж (tuple) – индексируемая неизменяемая коллекция, содержит элементы произвольного типа

```
a = (1,2)
b =()
c = tuple()
d =(1,)
```

? Что можно сказать про каждый из этих кортежей?

a - кортеж из двух целочисленных элементов
b – пустой кортеж
c – ещё один пустой кортеж
d – кортеж из одного элемента

! Чаще всего кортежи применяют для объединения разнородных сущностей, имеющих общий смысл.

Кортежи можно сравнивать друг с другом

```
a = ('green', 0.1)
b = ('red', 0.15)
print(a == b)
print (a != b)
print (a < b)
print (a <= b)
```

```
False
True
True
True
```

Что будет выведено?

В начале сравниваются первые элементы кортежей, если они равны, то вторые и т.п.
С помощью операций == и != можно сравнивать кортежи любых типов.
Операции <, <=, >, >= применимы только для кортежей одинаковых

Кортежи можно присваивать друг другу

```
a, b, c = 1, 2, 3
```

Сначала вычисляются все значения справа, и лишь затем они кладутся в левую часть оператора присваивания

```
a, b = b, a
```

Сортировка

Сортировка – упорядочивание элементов в выборке в соответствии с некоторым критерием

? К каким коллекциям Python можно применять алгоритмы сортировки?

Из известных нам коллекций – только к спискам.
Множество – неиндексируемая неупорядоченная коллекция.
Строка и кортеж – неиз[illegible]

? Тогда какое отношение кортежи имеют к алгоритмам сортировки?

Их удобно использовать при перестановке элементов.

Будем рассматривать сортировку списка по возрастанию.

Сортировка обменами (пузырьковая)

Последовательно просматриваются элементы списка, если встречается неупорядоченная пара s[k] < s[k-1], то элементы этой пары меняются местами; просмотр продолжается до s[n]. В результате первого просмотра наибольший элемент оказывается на первом месте. Второй просмотр начинается с первого элемента и заканчивается элементом s[n-1]. Алгоритм заканчивается, когда в просмотре участвуют только два первых элемента

Сортировка выбором

Из элементов списка выбирается максимальный и меняется местами с первым элементом. В результате первого выбора максимальный элемент оказывается на первом месте. Второй выбор максимума начинается со второго элемента и так до тех пор, пока среди выбираемых не останутся два последних элемента

А если надо сортировать по убыванию?

Преобразование коллекций

Применение методов `list()`, `set()` и `tuple()` к итерируемым объектам позволяет превратить эти объекты в список, строку и кортеж соответственно

```
s = 'корова'
a = list(s)
b = set(s)
c = tuple(s)
print(a)
print(b)
print(c)
```

? Что будет выведено?

```
['к', 'о', 'р', 'о', 'в', 'а']
{'о', 'а', 'в', 'р', 'к'}
('к', 'о', 'р', 'о', 'в', 'а')
```

```
s = list()
for i in range(int(input())):
    s.append(input())
m = set(s)
print(*m)
```

? Что выведет эта программа?

В одну строчку будут выведены уникальные элементы списка в произвольном порядке

Финал и не финал - разбор задачи, анализ ошибок

Вводятся данный о N, командах, каждая из которых характеризуется названием и количеством очков.
Половина команд (с округлением вверх) проходит в финал.
Соответственно, нам нужно вывести список команд прошедших в финал (по алфавиту) и список команд не прошедших в финал (по алфавиту)

Простой сортировкой по убыванию не обойдемся

Первый этап – ввод исходных данных

Вполне можно сделать два списка – для имен команд и набранных командами очков соответственно. Но тема урока была «кортежи». Давайте воспользуемся этой коллекцией

```
n = int(input())
commands = list()
for i in range(n):
    name = input()
    result = int(input())
    commands.append((name, result))
```

Второй этап – сортировка команд по убыванию очков

Тут главное не запутаться в индексах

`commands[j][0]` - название j-ой команды

`commands[j][1]` - количество очков j-ой команды

Используем алгоритм «пузырьковой» сортировки

```
for i in range(n - 1):
    for j in range(n - 1 - i):
        if commands[j][1] < commands[j + 1][1]:
            commands[j], commands[j + 1]
          = commands[j + 1], commands[j]
```

Третий этап – сортировка имен команд по алфавиту в каждой из групп (финалистов и нефиналистов)

? А собственно сколько у нас финалистов?

Поскольку берется половина от общего числа с округлением вверх, то число финалистов считается как `(n + 1) // 2`

Для разнообразия и закрепления алгоритма будем использовать сортировку выбором.

В отличие от предыдущего этапа, будем сортировать по первому компоненту кортежа, т.е. по `commands[j][0]`

Сортируем финалистов

```
for i in range((n + 1) // 2 - 1):
    m = i
    for j in range(i, (n + 1) // 2):
        if commands[j][0] < commands[m][0]:
            commands[j], commands[m] =
        commands[m], commands[j]
```

А теперь аутсайдеров

```
for i in range((n + 1) // 2, n - 1):
    m = i
    for j in range(i, n):
        if commands[j][0] < commands[m][0]:
            commands[j], commands[m] =
        commands[m], commands[j]
```

? Почему цикл по i именно в этих диапазонах?

И завершение программы – выводим результаты на печать

Не забудем, что выводить нужно только название команд

```
for command in commands:
    print(command[0])
```

Методы `split` и `join`. Списочные выражения

Популярные методы строк. Правила генерации списков. Считывание значений, введённых одной строкой.

Методы коллекций

Методы – это функции, «приклеенные» к объекту (например, списку или множеству) и изменяющими его содержимое.

Какие методы вы знаете для множеств? Для списков?

Множества

- `add()`
- `discard()`
- `remove()`
- `pop()`
- `clear()`
- `copy()`
- `union()`
- `intersection()`
- `difference()`
- `symmetric_difference()`
- `issubset()`
- `issuperset()`

Списки

- `append()`
- `extend()`
- `insert()`

Это далеко не все методы…
Постепенно будем их изучать

? Каким образом вызывается метод?

```
имя_коллекции.имя_метода(аргументы_метода)
```

? А есть ли методы у других коллекций?

Конечно же есть. Их не может не быть

Едва ли не самые популярные методы строк, широко используемых питонистами – `split()` и `join()`

`split()` разбивает строку по произвольному разделителю на список «слов»

`join()` собирает из списка слов единую строку через заданный разделитель

`split()` и `join()`, в отличие, например, от метода списков `append()` или метода множеств `add()`, не изменяют объект, которому принадлежат, а создают новый (список или строку, соответственно) и возвращают его, как это делают обычные функции типа `len()`.

Почему так?

Строка – неизменяемый объект

Вызов метода `split()` или `join()` можно подставить куда угодно — в любое выражение, в правую часть оператора присваивания, в цикл for и т.д.

Метод `split()`

Служит для преобразования строки в список

```
my_string = 'Иван Иванович Иванов'
my_list = my_string.split()
print(my_list)
```

? Что выведет эта программа?

```
['Иван', 'Иванович', 'Иванов']
```

! Рассмотрели пример метода split() без аргумента.

Строка разбивается на части, разделённые любыми символами пустого пространства (набором пробелов, символом табуляции и т.д.).

```
my_string = 'Иван...Иванович...Иванов'
my_list = my_string.split('...')
print(my_list)
```

Что выведет эта программа?

```
['Иван', 'Иванович', 'Иванов']
```

Рассмотрели пример метода `split()` с аргументом.

Строка разбивается на части, разделителем считается строка аргумент

А в чем разница между `split()` и `split(' ')`?

Давайте сравним

Один пробел между словами в строке

```
my_string = 'Иван Иванович Иванов'
my_list = my_string.split()
print(my_list)
my_list = my_string.split(' ')
print(my_list)
```

```
['Иван', 'Иванович', 'Иванов']
['Иван', 'Иванович', 'Иванов']
```

Два пробела между словами в строке

```
my_string = 'Иван  Иванович  Иванов'
my_list = my_string.split()
print(my_list)
my_list = my_string.split(' ')
print(my_list)
```

```
['Иван', 'Иванович', 'Иванов']
['Иван', '', 'Иванович', '', 'Иванов']
```

Пример

```
s = 'раз два три'
print(s.split() == ['раз', 'два', 'три'])
print('     one two  three  '.split() == ['one', 'two', 'three'])
print('192.168.1.1'.split('.') == ['192', '168', '1', '1'])
print(s.split('а') == ['р', 'з дв', ' три'])
print('A##B##C'.split('##') == ['A', 'B', 'C'])
```

? Что выведет эта программа?

```
True
True
True
True
True
```

Метод `join()`

Служит для преобразования списка в строку

```
my_list = ['1', '2', '3', '4', '5']
my_string =' '.join(my_list)
print(my_string)
```

? Что выведет эта программа?

```
1 2 3 4 5
```

! Метод `join()` же всегда принимает один аргумент — список слов, которые нужно склеить. Разделителем (точнее, «соединителем») служит та самая строка, чей метод `join` вызывается. Это может быть и пустая строка, и пробел, и символ новой строки, и что угодно ещё.

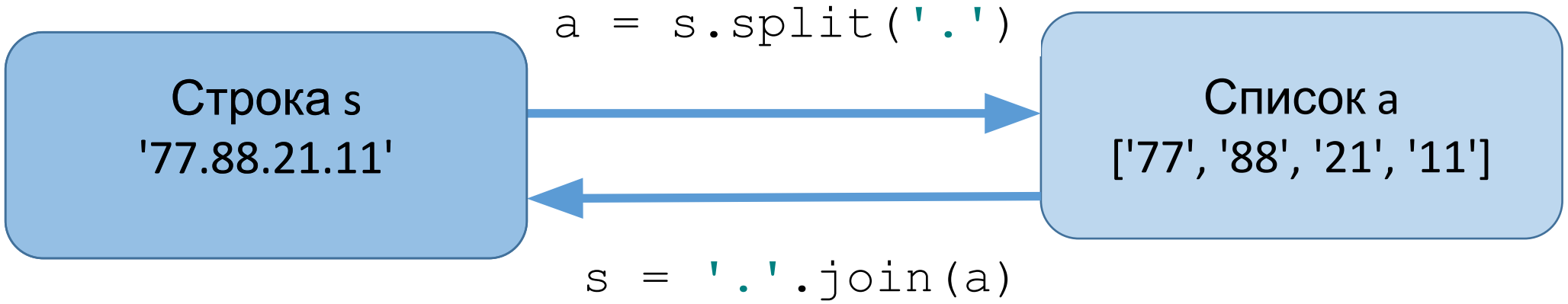
Примеры

```
s = ['Тот', 'Кого', 'Нельзя', 'Называть']
print(''.join(s) == 'ТотКогоНельзяНазывать')
print(' '.join(s) == 'Тот Кого Нельзя Называть')
print('-'.join(s) == 'Тот-Кого-Нельзя-Называть')
print('! '.join(s) == 'Тот! Кого! Нельзя! Называть')
```

? Что выведет эта программа?

```
True
True
True
True
```

Подытожим



`split()` или `join()` — это методы строк. Попытка вызвать такой метод у объекта, не являющегося строкой, вызовет ошибку!

Давайте проверим

```
[1, 2, 3].join([4, 5, 6])
```

```
AttributeError: 'list' object has no attribute 'join'
```

Задача 1. «Глория Скотт»

Ограничение времени 1 секунда
Ограничение памяти 64Mb
Ввод стандартный ввод или input.txt
Вывод стандартный вывод или output.txt

В рассказе Артура Конан Дойля «Глория Скотт» юный Шерлок Холмс сумел прочитать шифрованное письмо, текст которого в переводе Г. Любимова приведён в примере: в письме нужно читать только каждое третье слово (как будто непонятно, что может означать слово «берегитесь»!). Напишите программу, которая вычленяет из введённой строки каждое третье слово. Знаки препинания считать частью слова, даже если это приводит к неверной пунктуации (как, например, в примере, где не должны стоять запятые после подлежащего «дело» и после подлежащего «Хадсон»).

Формат ввода. Одна строка, состоящая из нескольких слов.

Формат вывода. Одна строка, включающая каждое третье слово из введённой строки; слова разделены пробелами.

Пример

Ввод

С дичью дело, мы полагаем, закончено. Глава предприятия Хадсон, по сведениям, рассказал о мухобойках всё. Фазаньих курочек берегитесь.

Вывод

дело, закончено. Хадсон, рассказал всё. берегитесь.

Списочные выражения

Списочные выражение – ещё одна фишка Python, позволяющая упростить код

Эта конструкция позволяет создавать элементы списка в цикле for, не записывая цикл целиком.

В Python не принято создавать пустые списки, чтобы потом заполнять их значениями

Для инициализации списков используем списочные выражения

Необходимо создать список квадратов целых чисел от 0 до 9

```
squares = []
for i in range(10):
    squares.append(i**2)
print(squares)
```

То же самое, но со списочными выражениями

```
squares = [i**2 for i in range(10)]
print(squares)
```

Результат – список: `[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]`

Необходимо создать список квадратов четных целых чисел от 0 до 9 включительно

```
even_squares = []
for i in range(10):
    if i % 2 == 0:
        even_squares.append(i**2)
print(even_squares)
```

То же самое, но со списочными выражениями

```
even_squares = [i**2 for i in range(10) if i % 2 == 0]
print(even_squares)
```

? А можно ли ещё

```
even_squares = [i**2 for i in range(0,10,2)]
print(even_squares)
```

Вместо условия в списочном выражении использовали шаг в `range()`

Результат – список: `[0, 4, 16, 36, 64]`

Можно ставить несколько циклов в списочном выражении

```
print([i * j for i in range(3) for j in range(3)])
```

? Что будет выведено?

```
[0, 0, 0, 0, 1, 2, 0, 2, 4]
```

Задание 3. Списочная квадратура

Ограничение времени 1 секунда
Ограничение памяти 64Mb
Ввод стандартный ввод или input.txt
Вывод стандартный вывод или output.txt

Используя списочное выражение, создайте список, включающий числа от 0 до введённого пользователем, возведённые в квадрат. Выведите элементы этого списка, каждый на отдельной строке, с помощью прохода циклом for непосредственно по элементам списка.

Формат ввода. Одно натуральное число.

Формат вывода. Числа на отдельных строках.

Использование списочных выражений в аргументах методов `split()` и `join()`

? Когда это применяется?

Ещё один «питонский» прием при написании кода – считывание значений, введённых одной строкой

! Комбинация метода `split()` и списочного выражения позволяет удобно считать числа, записанные в одну строку

```
print([int(x) for x in '976 929 289 809 7677'.split()])
```

Что будет выведено?

```
[976, 929, 289, 809, 7677]
```

А как это работает?

Строка разделяется на отдельные слова с помощью split. Затем списочное выражение пропускает каждый элемент получившегося списка через функцию int, превращая строку '976' в число 976.

Обычно строка не задаётся прямо в списочном выражении, а получается из `input()`

```
evil, good = [int(x) for x in '666 777'.split()]
print(evil, good, sep='\n')
```

Что будет выведено?

```
666
777
```

Какая структура использовалась в этом примере?

Комбинация метода `join()` и списочного выражения позволяет «склеивать» в одну строку список

```
print(', '.join(str(i) + '**2=' + str(i**2) for i in range(1, 10)))
```

Что будет выведено?

```
1**2=1, 2**2=4, 3**2=9, 4**2=16, 5**2=25, 6**2=36, 7**2=49, 8**2=64, 9**2=81
```

В аргументе функции `join()` стоит списочное выражение, не обёрнутое в квадратные скобки (это можно сделать, но не обязательно)

И финальный аккорд – решаем задачу в одну строку

Ввести несколько целых чисел в строку и вывести строку, в которой каждое из введенных чисел возводится в степень, равную самому числу

```
print(' '.join([str(int(i)**int(i)) for i in input().split()]))
```

```
1 2 3 4 5
1 4 27 256 3125
```

Если вы кодите таким образом – можете считать себя настоящим питонистом

На закрепление

Вводится строка произвольной длины. Каждое слово этой строки напечатать с новой строки столько раз, сколько в данном слове букв. Между словами в строке вывода должны быть пробелы.

```
зима лето попугай
зима зима зима зима
лето лето лето лето
попугай попугай попугай попугай попугай попугай попугай
```

«Соединителем» в методе `join()` выступает `'\n'`—символ начала новой строки

```
print('\n'.join((x  + ' ') * len(x) for x in input().split()))
```

Задание 2. Вертикальная диаграмма

Ограничение времени 1 секунда
Ограничение памяти 64Mb
Ввод стандартный ввод или input.txt
Вывод стандартный вывод или output.txt

Напишите программу для построения горизонтальных столбчатых диаграмм с помощью символа звёздочки.

Подсказка: выводя строку, не используйте для этого цикл.

Формат ввода. Вводится несколько натуральных чисел, разделённых пробелами.

Формат вывода. Для каждого введённого числа выводится строка, состоящая из звёздочек *, длина которой равна этому числу.

Приме

Ввод	Вывод
3 7 1 10 8	*** ******* * ********** ********

Задание 4. Списочная квадратура — 2

Ограничение времени 1 секунда
Ограничение памяти 64Mb
Ввод стандартный ввод или input.txt
Вывод стандартный вывод или output.txt

Используя списочное выражение, создайте список, состоящий из чисел от 0 до введенного пользователем (не включая это число), возведённые в квадрат. Выведите элементы этого списка на одной строке, разделяя их пробелами, с помощью метода join.

Формат ввода. Одно натуральное число.

Формат вывода. Несколько чисел на одной строке, разделённые пробелами.

Приме

Ввод	Вывод
4	0 1 4 9

Задание 5. Списочная квадратура - 3

Ограничение времени 1 секунда
Ограничение памяти 64Mb
Ввод стандартный ввод или input.txt
Вывод стандартный вывод или output.txt

Используя списочное выражение и метод split, составьте список из введённых чисел, записанных на одной строке без указания заранее их количества; затем выведите их квадраты также на одной строке.

Формат ввода. Несколько натуральных чисел на одной строке.
Формат вывода. Несколько чисел на одной строке.

Приме

р

Ввод	Вывод
7 1 6	49 1 36