АКАДЕМИЯ АЙТИ

День 8

academyit.ru

# Типовые задачи на обработку текста.

## Практические примеры составления блоксхем и псевдокода.
## Простейшие алгоритмические задачи.
## Перевод алгоритма в код.
## Подпрограммы (функции) как основные блоки кода.

Турашова Анна Николаевна
Преподаватель
anna1turashova@gmail.com
Telegram: @anna1tur

## Обобщение свойств встроенных коллекций в сводной таблице:

| Тип коллекции | Мутабельность | Индексированность | Уникальность | Как создаём |
|---|---|---|---|---|
| **Список** (list) | + | + | - | *[]*<br>*list()* |
| **Кортеж** (tuple) | - | + | - | *0 ,*<br>*tuple()* |
| **Строка** (string) | - | + | - | *''*<br>*''''* |
| **Множество** (set) | + | - | + | *{}*<br>*set()* |
| **Неизменное множество** (frozenset) | - | - | + | *frozenset()* |
| **Словарь** (dict) | + элементы<br>- ключи<br>+ значения | - | + элементы<br>+ ключи<br>- значения | *{key: value,}*<br>*dict()* |

# Словари

# Коллекции

## Словари:

Словари являются важнейшей структурой данных в Python-е. Они позволяют хранить данные в формате ключ-значение. Чтобы определить словарь, нужно использовать литерал фигурные скобки или просто вызвать dict. Если мы хотим, определяя словарь, сразу добавить в него данные, пишем ключ-значение через двоеточие.

```
empty_dict = {}
empty_dict = dict()
collections_map = {
                        'mutable': ['list', 'set', 'dict'],
                        'immutable': ['tuple', 'frozenset']}
```

Доступ к значению по ключу осуществляется за константное время, то есть не зависит от размера словаря. Это достигается с помощью алгоритма хеширования. Если пытаться получить доступ по ключу, которого не существует, Python выдаст ошибку KeyError. Однако, часто бывает полезно попытаться достать значение по ключу из словаря, а в случае отсутствия ключа вернуть какое-то стандартное значение. Для этого есть встроенный метод get.

```
print(collections_map['immutable'])
```

```
['tuple', 'frozenset']
```

# Коллекции

## Словари:

```
print(collections_map['irresistible'])
```

```
KeyError                                  Traceback (most recent call last) in ()
----> 1 print(collections_map['irresistible'])
KeyError: 'irresistible'
```

```
print(collections_map.get('irresistible', 'not found'))
```

```
not found
```

Проверка на вхождения ключа в словарь так же осуществляется за константное время и выполняется с помощью ключевого слова in:

```
'mutable' in collections_map
```

```
True
```

# Коллекции

## Словари:

Так как словарь является изменяемой структурой данных, мы можем добавлять и удалять элементы из него. Например, мы можем определить словарь beatles_map, который содержит знаменитых музыкантов и их инструменты, и добавить в него Ринго с 11 ударными, просто используя доступ по ключу. Чтобы удалить ключ и значение из словаря, можно использовать уже знакомый вам оператор del.

```
beatles_map = {
                    'Paul': 'Bass',
                    'John': 'Guitar',
                    'George': 'Guitar', }
print(beatles_map)
```

```
{'Paul': 'Bass', 'John': 'Guitar', 'George': 'Guitar'}
```

```
beatles_map['Ringo'] = 'Drums'
print(beatles_map)
```

```
{'Paul': 'Bass', 'John': 'Guitar', 'George': 'Guitar', 'Ringo': 'Drums'}
```

```
del beatles_map['John']
print(beatles_map)
```

```
{'Paul': 'Bass', 'George': 'Guitar', 'Ringo': 'Drums'}
```

# Коллекции

## Словари:

Также, чтобы добавить какой-то ключ-значение в словарь, можно использовать встроенный метод update, который принимает словарь и дополняет им (а также обновляет в случае одинаковых ключей) исходный словарь.

```
beatles_map.update({ 'John': 'Guitar' })
print(beatles_map)
```

```
{'Paul': 'Bass', 'George': 'Guitar', 'Ringo': 'Drums', 'John': 'Guitar'}
```

Чтобы удалить ключ-значение из словаря и одновременно вернуть значение, используют метод pop:

```
# удаляем Ринго, нам возвращаются его ударные
print(beatles_map.pop('Ringo'))
print(beatles_map)
```

```
Drums
{'Paul': 'Bass', 'George': 'Guitar', 'John': 'Guitar'}
```

# Коллекции

## Словари:

Часто бывает необходимо не только попробовать проверить, существует ли ключ в словаре, но и в случае неудачи добавить эту новую пару ключ-значение. Для этого есть метод setdefault:

```
unknown_dict = {}
print(unknown_dict.setdefault('key', 'default'))
print(unknown_dict)
```

```
default
{'key': 'default'}
```

Если вызвать setdefault и в качестве дефолтного значения передать new_default, вернётся значение, которое уже лежит в словаре — значение default:

```
print(unknown_dict.setdefault('key', 'new_default'))
```

```
default
```

# Коллекции

## Словари:

Словари, как и все коллекции, поддерживают протокол итерации. С помощью цикла for можно итерироваться по ключам словаря:

```
print(collections_map)
for key in collections_map:
    print(key)
```

```
{'mutable': ['list', 'set', 'dict'], 'immutable': ['tuple', 'frozenset']}
mutable immutable
```

Если нам нужно итерироваться не по ключам, а по ключам и значениям сразу, можно использовать метод словаря items, который возвращает ключи и значения.

```
for key, value in collections_map.items():
    print('{} — {}'.format(key, value))
```

```
mutable — ['list', 'set', 'dict']
immutable — ['tuple', 'frozenset']
```

# Коллекции

## Словари:

Если нужно итерироваться по значениям, используйте логично метод values, который возвращает именно значения. Также существует симметричный метод keys, который возвращает итератор ключей.

```
for value in collections_map.values():
    print(value)
```

```
['list', 'set', 'dict']
['tuple', 'frozenset']
```

Важная особенность словарей в Python-е: они содержат ключи и значения в неупорядоченном виде. Однако, в Python-е существует тип OrderedDict ( содержится в модуле collections), который гарантирует вам, что ключи хранятся именно в том порядке, в каком вы их добавили в словарь.

```
from collections import OrderedDict

ordered = OrderedDict()
for number in range(10):
   ordered[number] = str(number)
for key in ordered:
  print(key)
```

# Множества

# Коллекции

## Множества:

Множество в питоне — это неупорядоченный набор уникальных объектов. Множества изменяемы и чаще всего используются для удаления дубликатов и всевозможных проверок на вхождение.

Чтобы объявить пустое множество, можно воспользоваться литералом set или использовать фигурные скобки, чтобы объявить множество и одновременно добавить туда какие-то элементы.

```
empty_set = set()
number_set = {1, 2, 3, 3, 4, 5}

print(number_set)
```

```
{1, 2, 3, 4, 5}
```

# Коллекции

## Множества:

Чтобы проверить, содержится ли объект в множестве, используется уже знакомое нам ключевое слово in. Проверка выполняется за константное время, время выполнения операции не зависит от размера множества. Это достигается за счёт хэширования каждого элемента структуры по аналогии со словарями. По полученному от хэш-функции ключу и происходит поиск объекта. Таким образом, во множествах могут содержаться только хэшируемые объекты.

```
print(2 in number_set)
```

```
True
```

# Коллекции

## Множества:

Чтобы добавить элемент в множество, используется метод add. Также множества в Python поддерживают стандартные операции над множествами --- такие как объединение, разность, пересечение и симметрическая разность.

Создадим два множества с чётными и нечётными числами до десяти:

```
even_set = set()
odd_set = set()

for number in range(10):
    if number % 2:
        odd_set.add(number)
    else:
        even_set.add(number)

print(even_set)
print(odd_set)
```

```
{1, 3, 5, 7, 9}
{0, 2, 4, 6, 8}
```

# Коллекции

## Множества:

Теперь найдём объединение и пересечение этих множеств:

```
# обьединение
union_set = odd_set | even_set
other_union_set= odd_set.union(even_set)
print(union_set)
print(other_union_set)
```

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

```
# пересечение
intersection_set = odd_set & even_set
other_intersection_set= odd_set.intersection(even_set)
print(intersection_set)
print(other_intersection_set)
```

set()
set()

# Коллекции

## Множества:

Найдём разность и симметрическую разность двух множеств:

```
# разность
difference_set = even_set - odd_set
other_difference_set= odd_set.difference(even_set)
print(difference_set)
print(other_difference_set)
```

{0, 2, 4, 6, 8}
{1, 3, 5, 7, 9}

```
# симметрическая разность
symmetric_difference_set = even_set ^ odd_set
other_symm_difference_set= odd_set.symmetric_difference(even_set)
print(symmetric_difference_set)
print(other_symm_difference_set)
```

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

# Коллекции

## Множества:

Множества --- изменяемая структура данных, поэтому можно как добавлять туда элементы, так и удалять. Для удаления конкретного элемента существует метод remove, для удаления любого элемента можно использовать pop. Остальные методы можно посмотреть в help или документации.

```
even_set.remove(2)
print(even_set)
```

```
{0, 4, 6, 8}
```

```
print(even_set.pop())
```

```
0
```

Также в питоне существует неизменяемый аналог типа set --- тип frozenset.

```
frozen = frozenset(['Anna', 'Elsa', 'Kristoff'])
frozen.add('Olaf')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-64-962f221e1321>  in ()  <module>
1 frozen = frozenset(['Anna', 'Elsa', 'Kristoff'])
2
----> 3 frozen.add('Olaf')
 AttributeError: 'frozenset' object has no attribute 'add'
```

# Коллекции

## Множества. Пример программы:

Задача на множества: через сколько итераций функция random.randint(1, 10) выдаст повтор? Будем добавлять неповторяющиеся случайные числа в множество random_set. Если очередное число уже есть в random_set --- выйдем из цикла. Затем посчитаем длину множества (и прибавим 1, тк не учли последнее число).

```
import random
random_set = set()

while True:
    new_number = random.randint(1,10)
    if new_number in random_set:
        break
    random_set.add(new_number)

print(len(random_set)+1)
```

4

Таким образом, мы получили повтор через 4 итераций.

# Поверка домашнего задания

# Домашняя работа:

Задача 1:

Испорченный гугл: Создать программу, которая будет обрабатывать русский текст, заменяя в нём символы русского языка на схожие по написанию символы английского языка(a,c,x,o,p,e), при этом сохраняя регистр.

Задача 2:

Буквоежка level-up: Отсортировать по алфавиту три(можно больше) отдельных строки убирая общие(между всеми тремя строками) дубли

(итог выводить для каждой отдельной строки)

Задача 3:

Склад: Создать список словарей для инвентаризации чего-либо

Задача 4:

Цензор: создать программу, которая будет цензурировать вводимый текст по заданному шаблону

(не обязательно цензурировать мат! Шаблон может быть любым!(например слова-паразиты))

# Домашняя работа. Задача 1:

Для этого нам необходимо рассмотреть момент создания слова и дополнить программу словарями с заменой букв, где ключом будет символ для поиска, а значением - символ для замены. Создадим словари для всех возможных случаев.(совпадения во всех и конкретных регистрах)

```
text = input()
text_map = []
word = ''
repl_all = {'e':'e', 'o':'o', 'p':'p', 'a':'a', 'x':'x', 'c':'c'}
repl_up = {'T': 'T', 'K': 'K', 'B': 'B', 'M': 'M'}
repl_low = {'y': 'y'}
```

# Домашняя работа. Задача 1:

```
for symbol in text:
    if symbol in ' ,.!?':
        if word != '':
            text_map.append(word)
            word = ''
        if symbol in ',.!?' and symbol != ' ':
            text_map.append(symbol)
    else:
```

# Домашняя работа. Задача 1:

Сформируем "испорченное" слово с помощью условий и "поиска".

```
if symbol in repl_up:
    word += repl_up[symbol]
elif symbol in repl_low:
    word += repl_low[symbol]
elif symbol.lower() in repl_all:
    if symbol in repl_all:
        word += repl_all[symbol]
    else:
        word += repl_all[symbol.lower()].upper()
else:
    word += symbol
```

# Домашняя работа. Задача 1:

```
if word != '':
    text_map.append(word)
    word = ''
normalize_text = text_map[0].capitalize()
prev = ''
for ele in text_map[1:]:
    if ele in ',.!?':
        normalize_text += ele
    else:
        if prev in '.!?' and prev:
            normalize_text += ' ' + ele.capitalize()
            print(prev)
        else:
            normalize_text += ' ' + ele
    prev = ele
print(normalize_text)
```

# Домашняя работа. Задача 2:

Сформируем три списка и подготовим три множества.

```
text_1 = input().split()
text_2 = input().split()
text_3 = input().split()
mn_1 = set()
mn_2 = set()
mn_3 = set()
```

# Домашняя работа. Задача 2:

Очистим элементы списка и заполним множества.

```
for ele in text_1:
    mn_1.add(ele.strip(',.!?'))
for ele in text_2:
    mn_2.add(ele.strip(',.!?'))
for ele in text_3:
    mn_3.add(ele.strip(',.!?'))
```

# Домашняя работа. Задача 2:

Создадим множество дублей на основе повторений среди всех и каждой строки

```
mn = mn_1 & mn_2
for ele in mn_1 & mn_3:
    mn.add(ele)
for ele in mn_2 & mn_3:
    mn.add(ele)
```

Выведем результат(разность исходного множества каждой стоики и множества пересечений), не забыв отсортировать его.

```
print(" ".join(sorted(mn_1-mn)))
print(" ".join(sorted(mn_2-mn)))
print(" ".join(sorted(mn_3-mn)))
```

# Домашняя работа. Задача 4:

Для этого нам необходимо рассмотреть момент записи слова и дополнить программу множеством, которое будет содержать цензурируемые слова.

```
text = input()
text_map = []
word = ''
ban_word = {'привет', 'пока'}
for symbol in text:
    if symbol in ' ,.!?':
        if word != '':
```

# Домашняя работа. Задача 4:

Проверив слово на наличие в множестве цензурируемых. И продолжим старым кодом до момента контрольной записи

```
            if word in ban_word:
                text_map.append('***')
             else:
                text_map.append(word)
             word = ''
        if symbol in ',.!?' and symbol != ' ':
            text_map.append(symbol)
    else:
        word += symbol

if word != '':
```

# Домашняя работа. Задача 4:

Повторим проверку и закончим старым кодом.

```
    if word in ban_word:
        text_map.append('***')
    else:
        text_map.append(word)
    word = ''
normalize_text = text_map[0].capitalize()
prev = ''
```

# Домашняя работа. Задача 4:

```
for ele in text_map[1:]:
    if ele in ',.!?':
        normalize_text += ele
    else:
        if prev in '.!?' and prev:
            normalize_text += ' ' + ele.capitalize()
            print(prev)
        else:
            normalize_text += ' ' + ele
    prev = ele
print(normalize_text)
```

# Домашнее задание:

## Задача 1

Отсортировать слова в тексте по алфавиту, сохраняя регистр букв, но удаляя знаки препинания.

Ввод:

Добрый день, дорогие друзья! Сегодня ХОРОШАЯ погода!

Вывод:

день Добрый дорогие друзья погода Сегодня ХОРОШАЯ

## Задача 2

Необходимо отредактировать текст следующим образом:

1. Заменить неприличные слова на ***
2. Первая буква в предложении должна начинаться с заглавной буквы, а остальные – с маленькой.

Ввод:

Добрый мат день, дорогие блин друзья. сегодня ХОРОШАЯ мат погода!

Вывод:

Добрый *** день, дорогие *** друзья. Сегодня хорошая *** погода!

АПЛАНА
ГРУППА КОМПАНИЙ

Входит в ГК Аплана

АКАДЕМИЯ АЙТИ

Основана в 1995 г.

E-learning и очное обучение

## Направления обучения:

- Информационные технологии
- Информационная безопасность
- ИТ-менеджмент и управление проектами
- Разработка и тестирование ПО
- Гос. и муниципальное управление

## Филиалы:

Санкт-Петербург, Казань, Уфа, Челябинск, Хабаровск, Красноярск, Тюмень, Нижний Новгород, Краснодар, Волгоград, Ростов-на-Дону

## Головной офис в Москве

Разработка программного обеспечения и информационных систем



Ежегодные награды Microsoft, Huawei, Cisco и другие

## Программы по импортозамещению

Ресурсы более 400 высококлассных экспертов и преподавателей

Сеть региональных учебных центров по всей России

## Крупные заказчики

СБЕР

РЖД

Билайн

TATNEFT

НОРНИКЕЛЬ

**100+**
сотрудников

АКАДЕМИЯ АЙТИ

# Спасибо за внимание!

Центральный офис:
Москва, Варшавское шоссе 47, корп. 4, 7 этаж
Тел: +7 (495) 150-96-00
academy@it.ru
academyit.ru