# Вложенные структуры данных

# Типы данных: обобщение

## Примеры вложенных структур

# «Естественные» типы данных

«Неделимые» типы данных.

Целое число / integer

```
int
3
```

Вещественное число / floating-point number

```
float
3.5
```

Логическая переменная / logical

```
bool
False
```

# Упорядоченные типы данных

Последовательности элементов.

Строка / string

```
str
'молоко'
```

Список / list

```
list
['молоко', 'хлеб']
```

Кортеж / tuple

```
tuple
('молоко', 'хлеб')
```

# Неупорядоченные типы данных

Коллекции элементов.

Множество / set

```
set
{'молоко', 'хлеб'}
```

Словарь / dictionary

```
dict
['молоко':69.90, 'хлеб':35.0]
```

# Изменяемые типы данных

*Можем изменять, удалять и добавлять элементы.*

***НЕ** могут входить во множества и быть ключами словарей.*

Множество / set

Словарь / dictionary

Список / list

# Неизменяемые типы данных

***НЕ** можем изменять, удалять и добавлять элементы.*

*Могут входить во множества и быть ключами словарей.*

Целое число / integer

Вещественное число / floating-point number

Логическая переменная / logical

Строка / string

Кортеж / tuple

## Типы данных: обобщение

# Примеры вложенных структур

# Примеры вложенных структур данных

Список списков

```
[['хлеб','сыр'], ['порошок','мыло']]
```

Список кортежей

```
[(10, 8, 9.4), (7, 5, 7.8)]
```

Список множеств

```
[{'ужик','кот'}, {'собака'}]
```

Список словарей

```
[{'Петя':True, 'Аня':True},
 {'Вася':True, 'Алена':False}]
```

# Примеры вложенных структур данных

Словарь списков

```
{'181 группа':['Маша','Семен','Олег'],
 '182 группа':['Ольга','Алена']}
```

Словарь множеств

```
{'Вася':{'ужик','кошка','хомячок'},
 'Олег':{'шиншилла','кошка'}}
```

Словарь словарей списков

```
{'1 курс':
      {'181 группа':['Маша','Семен','Олег'],
       '182 группа':['Ольга','Алена']},
'2 курс':
      {'171 группа':['Таня','Женя','Саша'],
       '172 группа':['Аня','Валя']}}
```

# Обращение к вложенным элементам

Список списков

```
sh_list = [['хлеб','сыр'], ['порошок']]

sh_list[0]
>   ['хлеб','сыр']

sh_list[0][2]
>   'сыр'

sh_list[0][2][1]
>   'ы'
```

# Обращение к вложенным элементам

Словарь словарей списков

```
students =
{'1 курс':
{'181 гр.':['Маша','Семен','Олег'],
 '182 гр.':['Ольга','Алена']},
'2 курс':
{'171 гр.':['Таня','Женя','Саша'],
 '172 гр.':['Аня','Валя']}}


students['1 курс']['181 гр.'][1]
> 'Семен'

students['1 курс']['181 гр.'][1][0]
> 'С'
```

# Вложенные структуры данных

# Сортировка с sorted()

## Сортировка строк

## Параметр key

## Нахождение минимума и максимума

# Когда делать сортировку?

На самом деле мы часто сталкиваемся с задачами сортировки:

- Напечатать даты *в хронологическом порядке*.
- Занести имена студентов в ведомость *по алфавиту*.
- Упорядочить категории расходов *по убыванию суммы трат*.
- Отсортировать товары в интернет-магазине *от дешевых к самым дорогим*.

# Сортировка с sorted()

Сортировку последовательностей и коллекций можно сделать с помощью встроенной функции ***sorted()***

```
marks = [8.4, 4, 8, 9.4]
sorted(marks)
> [4, 8, 8.4, 9.4]
```

Сортировка происходит от меньшего к большему, чтобы сделать наоборот, нужно установить значение именованного параметра ***reverse*** равным `True`

```
marks = [8.4, 4, 8, 9.4]
sorted(marks, reverse=True)
> [9.4, 8.4, 8, 4]
```

# Сортировка с sorted()

Функция ***sorted()*** *не меняет исходный объект*.
Если мы хотим продолжать работу с отсортированным списком, то нужно сохранить результат сортировки в переменную.

```
marks = [8.4, 4, 8, 9.4]
marks_sorted = sorted(marks)
print(marks)
print(marks_sorted)
```

```
> [8.4, 4, 8, 9.4]
  [4, 8, 8.4, 9.4]
```

# Сортировка с sorted()

Функция ***sorted()*** *возвращает отсортированный список*, какой бы тип данных мы бы ни сортировали.

```
    marks_t = (8.4, 4, 8, 9.4) # кортеж
    sorted(marks)
>   [8.4, 4, 8, 9.4]

    marks_s = {9, 10, 7} # множество
    sorted(marks_s)
>   [7, 9, 10]
```

# Сортировка с sorted()

Отсортировать можно только структуру данных, содержащую данные *одного типа* (вещественные и целые числа считаются за числа).

```
marks = [8.4, 'хорошо', 8, 9.4]
sorted(marks)
> TypeError:
'<' not supported between instances
of 'int' and 'str'
```

Сортировка с sorted()

**Сортировка строк**

Параметр key

Нахождение минимума и максимума

# Сортировка строк

*Строки сортируются в соответствии с таблицей кодировки*, в которой записаны все-все символы в определенном порядке

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 00 □ | 01 □ | 02 □ | 03 □ | 04 □ | 05 □ | 06 □ | 07 □ | 08 □ | 09 □ | 0A □ | 0B □ | 0C □ | 0D □ | 0E □ | 0F □ |
| 10 | 10 □ | 11 □ | 12 □ | 13 □ | 14 □ | 15 □ | 16 □ | 17 □ | 18 □ | 19 □ | 1A □ | 1B □ | 1C □ | 1D □ | 1E □ | 1F □ |
| 20 | 20 | 21 ! | 22 “ | 23 # | 24 $ | 25 % | 26 & | 27 ( | 28 ) | 29 ' | 2A * | 2B + | 2C , | 2D - | 2E . | 2F / |
| 30 | 30 0 | 31 1 | 32 2 | 33 3 | 34 4 | 35 5 | 36 6 | 37 7 | 38 8 | 39 9 | 3A : | 3B ; | 3C < | 3D = | 3E > | 3F ? |
| 40 | 40 @ | 41 A | 42 B | 43 C | 44 D | 45 E | 46 F | 47 G | 48 H | 49 I | 4A J | 4B K | 4C L | 4D M | 4E N | 4F O |
| 50 | 50 P | 51 Q | 52 R | 53 S | 54 T | 55 U | 56 V | 57 W | 58 X | 59 Y | 5A Z | 5B [ | 5C \ | 5D ] | 5E ^ | 5F _ |
| 60 | 60 ` | 61 a | 62 b | 63 c | 64 d | 65 e | 66 f | 67 g | 68 h | 69 i | 6A j | 6B k | 6C l | 6D m | 6E o | 6F p |
| 70 | 70 p | 71 q | 72 r | 73 s | 74 t | 75 u | 76 v | 77 w | 78 x | 79 y | 7A z | 7B { | 7C \| | 7D } | 7E ~ | 7F □ |

# Сортировка строк

*Строки сортируются посимвольно.* Если первый символ строки идет в таблице кодировки раньше символов других строк, то строка сортируется в начало списка.

```
words = ['bee', 'apple', 'cat']
sorted(words)
> ['apple', 'bee', 'cat']
```

Если первые символы совпадают, то сортировка идет по второму, третьему и т.д.

```
words = ['apple', 'artist', 'app']
sorted(words)
> ['app', 'apple', 'artist']
```

# Сортировка строк

Если в вашем списке числа представлены строками, то *Python будет их сравнивать не как числа, а как строки*! И результат такой сортировки будет неожиданным.

```
dates = [8, 10, 14, 1] # числа float
sorted(dates)
>   [1, 8, 10, 14]


dates = ['8','10','14','1'] # числа str
sorted(dates)
>   ['1','10','14','8']
```

# Сортировка строк

Так как в любых таблицах кодировки сначала идут все заглавные буквы по алфавиту, а затем все строчные, то список строк в разных регистрах будет отсортирован сначала *по заглавным буквам, а затем по строчным.*

```
marks = ['Хор.', 'отл.', 'Удов.']
sorted(marks)
> ['Удов.', 'Хор.', 'отл.']
```

Сортировка с sorted()

Сортировка строк

**Параметр key**

Нахождение минимума и максимума

# Параметр key

1. У функции ***sorted()*** есть еще один именованный параметр – **key**.
2. **key** можно присвоить название функции и эта функция применится ко всем элементам последовательности или коллекции перед сортировкой.

   Например, с помощью этого параметра можно отсортировать строки без учета регистра.

```
marks = ['Хор.', 'отл.']
sorted(marks, key=str.lower)
> ['отл.' , 'Хор.']
```

# Параметр key

Логика работы ***key*** чем-то похожа на то,
как работает функция *map()*.

Но при сортировке с ***key***, мы применяем функцию
к элементам контейнера только для этапа сортировки,
мы не изменяем оригинальные элементы.

```
marks = map(str.lower, ['Хор.','отл.'])
# в marks теперь ['хор.','отл.']
sorted(marks)
>   ['отл.' , 'хор.']

marks = ['Хор.','отл.']
sorted(marks, key=str.lower)
>   ['отл.' , 'Хор.']
```

Сортировка с sorted()

Сортировка строк

Параметр key

**Нахождение минимума и максимума**

# Минимум и максимум

Функции **min()** и **max()** для последовательностей и коллекций с числами возвращают минимум и максимум соответственно.

```
marks = [8.4, 4, 8, 9.4]
min(marks)
> 4
marks = {8.4, 4, 8, 9.4} # множество
max(marks)
> 9.4
```

# Минимум и максимум

Функции **min()** и **max()** для последовательностей и коллекций со строками работают по аналогии с сортировкой (Python смотрит в таблицу символов, чтобы решить, что идет раньше, а что позже).

```
words = ['bee', 'apple', 'cat']
max(words)
> 'cat'

words = ['bee', 'apple', 'cat']
min(words)
> 'apple'
```

# Вложенные структуры данных

# Повторение

# Словари

```
sh_d = {‘хлеб’: 30.0, ‘молоко’: 86.4}


print(sh_d[‘хлеб’])
# 30.0


print(sh_d.keys())
# dict_keys(['хлеб', 'молоко’])
print(sh_d.values())
# dict_values([30.0, 86.4])
print(sh_d.items())
# dict_items([('хлеб', 30.0), ('молоко',
86.4)])
```

# Вложенные структуры данных

Списки списков:

```
students = [
    ['Ольга Ларина', 2003, 'Политология'],
    ['Вениамин Ерофеев', 2002, 'Химия']
]


print(students[0])
# ['Ольга Ларина', 2003, 'Политология']


print(students[0][2])
# Политология
```

# Вложенные структуры данных

Словари списков:

```
books = {
    'Ася Казанцева': ['В интернете кто-то не прав!', 'Мозг материален'],
    'Лев Выготский': ['Мышление и речь']
}


print(books['Ася Казанцева'])
# ['В интернете кто-то не прав!', 'Мозг материален']


print(books['Ася Казанцева'][1])
# Мозг материален
```

# Вложенные структуры данных

Словари множеств:

```
exam = {
     '205': {'Аня', 'Дима'},
     '404': set()
}


print(exam['205'])
# {'Дима', 'Аня'}
```

# Вложенные структуры данных

Словари словарей:

```
books = {
    'Норберт Винер': {
        'title' : 'Кибернетика или
управление и связь в животном и машине’,
        'year' : 1958,
        'publisher' : 'Советское радио’
    }
}

print(books[‘Норберт Винер’][‘title’])
# Кибернетика или управление и связь в
животном и машине
```