Технологии програм-ия. Python как второй язык.

Лекция 3/13

Типы коллекций (контейнерные последовательности):
кортежи (tuples), collections.namedtuple,
collections.deque (double-ended queue).
Типы коллекций (mappings, sets): словари, множества.
Плоские последовательности: строки (strings), bytearrays.
Про другие включения(comprehensions): не только listcomp -ы,
но и dict_comp -ы, и set_comp -ы (словарные и множественные).
Полезное: полезные модули-2. Вопросы для экз. (sic!).
(v. 0.6)

И.Ф.Травов

igor.travov@gmail.com

Саров 2017/09 - 2018/05

Из иерархии типов:[†]

5. Sequences (упорядоченная коллекция эл-тов с неотриц-ми индексами, с длиной и срезами.)
   - кортеж (2 метода класса tuple - кортеж)

[†] см слайд 10 лек 2

- *кортеж* – это упорядоченная последовательность из нуля или более ссылок на объекты, **неизменяемый**, контейнер.
- функционал кортежа как объекта (без спец методов):
  1. задав значение, узнать количество таких в кортеже
  2. задав значение, узнать его индекс (первый)

```
>>> a = (1)
>>> type(a)
<class 'int'>    # это не кортеж , а всего лишь интОвый объект
>>> a = (1, )    # а это одноэлементный кортеж
>>> type(a)
<class 'tuple'>  # у одиночного кортежа, запятая обязательна
```

```
>>> tuple() #неизменяемый, но м.б. вложенный, а там всякое
()
>>> date = ("year", 2017)
>>> date
('year', 2017)
>>> len(date)
2
>>> date[1] = 2018
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> tt = (1, 1, 1, 3, 6, 887)
>>> # Методы кортежа:
>>> tt.count(1)     # сколько указанных эл-тов в кортеже
3
>>> tt.index(887)   # индекс
5
>>> tuple(reversed(tt)) # исходный кортеж как-то поменяли
(887, 6, 3, 1, 1, 1)
```

Кортежи как: 1)неизменяемые списки, 2)записи для именованных кортежей

первое как в list только без добавления и удаления, а реверсирование через встр функ reversed()

```
>>> tt1 = (7, 5)
>>> tt2 = (1, 0)
>>> tt1 + tt2
(7, 5, 1, 0)
>>> tt1 * 3
(7, 5, 7, 5, 7, 5)
>>> 5 in tt1
True
>>> tt1
(7, 5)
>>> tt3 = tuple(reversed(tt1))
>>> tt3
(5, 7)
>>>
```

```
>>> from collections import namedtuple # см следующ слайд
>>> Result = namedtuple('Result', 'FIO math rus engl info')
>>> Ivanov = Result('Ivanov', 55, 77, 90, 56)
>>> Ivanov
Result(FIO='Ivanov', math=55, rus=77, engl=90, info=56)
>>> Ivanov.math, Ivanov.info, Ivanov[1] # обращение к полям по
('math', 'info', 'math') #имени или по номеру не хватило места
>>> sum(Ivanov[1:]) # еще слайсу можно дать имя
278
>>> sum(Ivanov[2:4]) # суммируем для, напр-р, усреднения
167
>>> Result._fields
('FIO', 'math', 'rus', 'engl', 'info')
>>> Ivanov._asdict()
OrderedDict([('FIO', 'Ivanov'), ('math', 55), ('rus', 77),
('engl', 90), ('info', 56)])
>>>
```

- Функция collections.namedtuple – это фабрика, порождающая подклассы tuple, плюс еще можно задавать имена полей (фабричная функция порождает объекты определенного типа)

```
>>> dir(Result)
['FIO', '__add__', '__class__', '__contains__', '__delattr_
'__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getnewargs__', '__gt_
 '__hash__', '__init__', '__init_subclass__', '__iter__',
'__le__', '__len__', '__lt__', '__module__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmul__', '__setattr__', '__sizeof__',
'__slots__', '__str__', '__subclasshook__', '_asdict',
'_fields', '_make', '_replace', '_source', 'count',
 'engl', 'index', 'info', 'math', 'rus']
>>>
```

- _fields – кортерж с именами полей
- _asdict возвращает collections.OrderedDict, построенный по именованному кортежу

collections.deque

- Вставка и удаление с левого конца списка – не эффективно (весь список надо двигать) и тогда для этого применяют дек: deque – double end queue (двусторонняя очередь) из модуля collections
- Как список, только с обоих концов и методы похожие
- методы смотрим и применяем (всего 15 и почти как у списка):

```
>>> [i for i in dir(dq) if '__' not in i]
['append', 'appendleft', 'clear', 'copy', 'count', 'extend',
'extendleft', 'index', 'insert', 'maxlen', 'pop',
'popleft', 'remove', 'reverse', 'rotate']
>>>
```

```
>>> from collections import deque
>>> dq = deque(range(10), maxlen=10)
>>> dq
deque([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], maxlen=10)
>>> dq.rotate(3)
>>> dq
deque([7, 8, 9, 0, 1, 2, 3, 4, 5, 6], maxlen=10)
>>> dq.rotate(-4)
>>> dq
deque([1, 2, 3, 4, 5, 6, 7, 8, 9, 0], maxlen=10)
>>> dq.appendleft(-1)
>>> dq
deque([-1, 1, 2, 3, 4, 5, 6, 7, 8, 9], maxlen=10)
>>> dq.extend([11, 22, 33])
>>> dq
deque([3, 4, 5, 6, 7, 8, 9, 11, 22, 33], maxlen=10)
>>> dq.extendleft([10, 20, 30, 40])
>>> dq
deque([40, 30, 20, 10, 3, 4, 5, 6, 7, 8], maxlen=10)
```

Из иерархии типов:

7. Mappings (индекс-ся ключами по произвол множ-ву эл-тов)
   - 11 (всего методов класса dict - словарь)

maps(отображения) – тип, к кот-му можно применить in, функцию len, возможность обхода эл-тов(items) в цикле

Тип dict (словарь) – неупорядоченная коллекция из нуля или более пар "ключ-значение", где в качестве ключей ссылки только на хэши, а значения – объекты любого типа.

приём объявления словаря через словарное включение (дикткомп)

```
>>> dd = {a: b for a, b in zip('aoeuiy', range(6))}
>>>"""ключи и значения (a, b) должны получать своё при
 итерировании, например от встр функ-и зип"""
>>> dd  #пары(items) ключи из англ гласных, значения из диапаз
{'a': 0, 'o': 1, 'e': 2, 'u': 3, 'i': 4, 'y': 5}
>>> from random import sample, random
>>> dd = {a: b for a, b in zip('aoeuiy', sample(range(6), 6))}
>>> dd   # и ключи(само-собой) и значения уникальны
{'a': 3, 'o': 1, 'e': 2, 'u': 5, 'i': 4, 'y': 0}
>>> dd = {a: randint(1,20) for a, b in zip('aoeuiy', range(6))}
>>> dd   # тут по классике - ключи уникальны, значения нет
{'a': 13, 'e': 13, 'i': 3, 'o': 17, 'u': 18, 'y': 12}
```

1. удалить все айтемы словаря до пустого
2. скопировать (поверхностное)
3. создать новый с указанными ключами из итерабельного и указ-ми значениями, иначе нан
4. возвратить значение по указ-му ключу или нан, если отсутствует
5. возвратить множество айтемов словаря
6. возвратить множество ключей словаря
7. возвратить указанный ключ и удалить
8. возвратить и удалить указанный эл-т
9. возвратить указанный ключ с указ элтом или с нан
10. модифицирует имеющуюся пару по ключу, либо добавляет новое
11. возвратить значения словаря

```
>>> dict() # hash map
{}
>>> date = {"year": 2017, "month": "Febrary"}
>>> len(date)
2
>>> date["year"]
2017
>>> date.get("day", 17)
17  # ошибки не будет вернет значение по несуществующему ключу
>>> date["year"] = 2015 # To the future!
>>> date
{'year': 2017, 'month': 'Febrary'}
>>> del date["year"]
>>> date
{'month': 'Febrary'}
```

Вопрос

Как проверить наличие элемента в словаре?

Ключи и значения в словаре

```
>>> date = {"year": 2017, "month": "Febrary"}
>>> date.keys()
dict_keys(['year', 'month'])
>>> date.values()
dict_values([2017, 'Febrary'])
>>> date.items()
dict_items([('year', 2017), ('month', 'Febrary')])
>>> other_date = {"month": "March", "day": 22}
>>> date.keys() + other_date.keys() #но множ-ва не складываютс
Traceback (most recent call last):
File "<input>", line 1, in <module>
TypeError:unsupported operand type(+):'dict_keys'and'dict_keys'
>>> date.keys() | other_date.keys() # но объединение множеств
{'day', 'year', 'month'} # "ключевых" работает
>>> date.items() | other_date.items()  # также множеств "айтем
{('year', 2017),('month', 'March'),('month', 'Febrary'),('day',
>>>
```

Задать словарь можно(5 вариан-в: 4 через *dict()*, 1явно:)

```
>>> d1 = dict({"id": 1948, "name": "Washer", "size": 3})
>>> d2 = dict(id=1948, name="Washer", size=3)
>>> d3 = dict([("id", 1948), ("name", "Washer"), ("size", 3)])
>>> d4 = dict(zip(("id", "name", "size"),(1948, "Washer", 3)))
>>> d5 = {"id": 1948, "name": "Washer", "size": 3}
>>> d1  # прописываем словарь как большой аргумент функции
{"id": 1948, "name": "Washer", "size": 3}
>>> d2  #
{'size': 3, 'id': 1948, 'name': 'Washer'}
>>> d3  # список кортежей
{'name': 'Washer', 'id': 1948, 'size': 3}
>>> d4  #парал-ное итерирование по нескольким последоват-ям
{'name': 'Washer', 'id': 1948, 'size': 3}
>>> d5  # явный вызов
{"id": 1948, "name": "Washer", "size": 3}
```

```
>>> [d for d in dir(dict) if "__" not in d]
['clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop',
'popitem', 'setdefault', 'update', 'values']
>>> dd = {}
>>> dd.fromkeys([1, 2, 3])  #новый словарь с ключами из итрбл
{1: None, 2: None, 3: None}
>>> dd.fromkeys([1, 2, 3], ('a', 'v', 'c'))
{1: ('a', 'v', 'c'), 2: ('a', 'v', 'c'), 3: ('a', 'v', 'c')}
>>> date.setdefault("week", 22) #вставить пару или ключ с нан
22
>>> date
{'week': 22, 'year': 2017, 'month': 'Febrary'}
>>> date.pop('week')  # удалить этот айтем
22
>>> date
{'year': 2017, 'month': 'Febrary'}
>>>
```

dict comprehension дикткомп, задать словарь (создать) через включение словаря (словарное включ-е)

```
#как-то должны прописать уникальные ключи и какие-то значения,
>>> dc = {k : None for k in random.sample(range(10), 5) }
>>> dc
{6: None, 0: None, 8: None, 3: None, 4: None}
>>>""" если импортировать стринговую константу и
  рандомный метод, то можно получить словарь с лат буквами
  как ключами и рандомными числами как значениями"""
>>> dd = {a: b for a in ascii_lowercase for b
          in sample(range(1, 27), 1)}
>>> dd
{'a': 6, 'b': 23, 'c': 22, 'd': 24, 'e': 6, 'f': 10, 'g': 18,
'h': 18, 'i': 9, 'j': 7, 'k': 25, 'l': 24, 'm': 6, 'n': 11,
'o': 19, 'p': 20, 'q': 8, 'r': 3, 's': 13, 't': 12, 'u': 6,
'v': 19, 'w': 23, 'x': 26, 'y': 16, 'z': 21}
>>>
```

- Из встроенных функций: enumerate(iterable, start=0) возвращает 2-х элт-ные кортежи: счетчик (по дефолту 0) и значение, полученное из Итерабл объекта.

```
class enumerate(object)                # Help on class enumerate
| enumerate(iterable[, start]) -> iterator for index, value of
| iterable. Return an enumerate object. iterable must be anothe
| object that supports iteration. The enumerate object yields
| pairs containing acount and a value yielded by the itrblarg.
| enumerate is useful for obtaining an indexed list:
|       (0, seq[0]), (1, seq[1]), (2, seq[2]), ...
>>>
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
>>> list(enumerate(seasons, start=1))
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
>>> enumerate(seasons, start=1)          # энумерейт объект как
<enumerate object at 0x7fe99128da20>   # генератор пар
>>>
```

- enumerate(iterable, start=0) – полезная функция (вместо range()), если нужен еще и индекс
- теперь – как не надо (моветон). Пример получение рейтинга сортов мороженного.

```
>>>flavor_list =['vanilla', 'chocolate', 'pecan', 'strawberry']
>>>for i in range(len(flavor_list)):
...   flavor = flavor_list[i]
...   print('%d: %s' % (i + 1, flavor))
...
1: vanilla
2: chocolate
3: pecan
4: strawberry
>>> #специально печатаем номер элемента списка и его значение
```

- теперь – как надо, как правильно по Дзену Питона

```
>>> for i, flavor in enumerate(flavor_list, start=1):
...     print('%d: %s' % (i, flavor))
...                     # или print(i, ':', ' ', flavor, sep='')
1: vanilla
2: chocolate
3: pecan
4: strawberry
>>> # теперь номер элемента получается автоматом
>>> # либо еще вариант через генераторное выражение
>>>list((i,flav) for i,flav in enumerate(flav_list, start=1))
[(1, 'vanilla'), (2, 'chocolate'), (3, 'pecan'), (4, 'strawberr
>>>  # сократил имена для слайда (далее знакомый пример)
```

- Группировка элементов в каких-то хранилищах.
  Задана произвольная строка и надо найти все позиции каждого уникального элемента этой строки (2 варианта). †

† из сообщения М.Усачева "Красота и изящность стандартной библиотеки Python" на Python Meetup 2013

```
>>> import random
>>> import string
>>> choices = string.ascii_letters + string.digits
>>> choices  #строка из разнорегистровых букв и плюс цифр 0-9
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
>>> r_str = [random.choice(choices) for i in range(1, 999)]
>>> res = {}  #в результате будут 998айтемов(элементов словаря)
>>> for indx, elem in enumerate(r_str):
...     res.setdefault(elem, []).append(indx)
...
>>> res
{'I': [0, 220, 285, 348, 365, 395, 408, 441, 469,
505, 679, 908, 939], 'a': [1, 23, 44, 156, 212, 262, 297, 311,
490, 518, 688, 711, 760, 767, 790, 824, 899, 907, 937, 955, 961
'o': [2,.....
....]}
>>> #Отлично (7-8 стр), но есть еще способ и, говорят, лучше!!
```

- другой способ через метод defaultdict из модуля collections.

```
>>> # пример defaultdict
>>>#можно использовать список (или др) как default_factory как
>>>#последовательность key-value пар для финального словаря:
>>> s = [('yellow', 1), ('blue', 2), ('yellow', 3),
('blue', 4), ('red', 1)]
>>> d = defaultdict(list)  #фабричная функция возвращает list
>>> for k, v in s:
...     d[k].append(v)
...
>>> sorted(d.items())
[('blue', [2, 4]), ('red', [1]), ('yellow', [1, 3])]
>>> d = defaultdict(int)   #фабричная функция возвращает int
>>> d['v']                  >>> d['vo'] += 5
0                           >>> d
>>> d['v'] += 2             defaultdict(<class'int'>,{'v':2,'vo':5})
>>> d                       >>>
defaultdict(<class 'int'>, {'v': 2})
>>>
```

```
>>> # окочательный вариант
>>> import random
>>> import string
>>> from collections import defaultdict
>>>
>>> choices = string.ascii_letters + string.digits
>>> r_str = [random.choice(choices) for i in range(1, 999)]
>>> res = defaultdict(list)
>>> for indx, elem in enumerate(r_str):
...     res[elem].append(indx)
...
>>> sorted(res.items())  # отсортированный результат
[('0', [7, 17, 96, 252, 265, 306, 648, 668, 733, 804, 850, 968]
('1'...
...
('y', [79, 223, 278, 327, 486, 518, 543, 769, 805, 835, 837,
854, 994]), ('z', [75, 148, 225, 305, 393, 457, 470, 565,
576, 654, 744, 760, 833, 834, 914])]
>>>
```

Из иерархии типов:

6. Set types (неупоряд коллекция уник-ых безиндексных объектов, изм/неизменяемых, Sets/Frozen sets)
   - 17 всего методов класса set - множество

set, frozenset – неупорядоченная коллекция из нуля или более ссылок на объекты, указывающих на хешируемые.

```
# !!! Они итерируемы и поддерживают in и len
>>> ss = {1, 4, 55, 77}
>>> ss
{1, 4, 77, 55}  # неупорядоченность
>>> ls = [i for i in dir(ss) if "__" not in i]
>>> ls
['add', 'clear', 'copy', 'difference', 'difference_update',
'discard', 'intersection', 'intersection_update', '
isdisjoint', 'issubset', 'issuperset', 'pop', 'remove',
'symmetric_difference', 'symmetric_difference_update',
'union', 'update']
>>> len(ls)
17   # у множества 17 методов
>>>
```

```
>>> xs = {1, 2, 3, 4}
>>> ys = {4, 5}
>>> xs.intersection(ys)
{4}
>>> xs & ys    # пересеченеиме множеств
{4}
>>> xs.union(ys)  # объединение
{1, 2, 3, 4, 5}
>>> xs | ys
{1, 2, 3, 4, 5}
>>> xs.difference(ys)  # разность  xs - ys
{1, 2, 3}
>>> xs.symmetric_difference(ys)  # симетрическая разность
{1, 2, 3, 5}                     # (xs - ys) U (ys - xs)
>>>
```

```
>>> aa = set([2, 4, 6])
>>> hash(aa)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'set'
>>> aa = frozenset([2, 4, 6])
>>> aa
frozenset({2, 4, 6})
>>> hash(aa)
3252282146219220 8
>>> # так кто хешируемый (какой объект)?
```

Почему множество – не последовательность? (см.определение)

```
>>> se = set([1, 2, 33, 2]) #
>>> se
{1, 2, 33}
>>> len(se)  # проверяем длину
3
>>> 1 in se  # проверяем вхождение элемента
True
>>> se[:2]   # проверяем срез (слайс) от второго до конца
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'set' object is not subscriptable#подиндекс нельзя
>>> se[0]    # проверяем на индекс
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support indexing
>>>                              # не индексируется
>>>
```

Set comprehension сеткомп, задать множество (создать) через включение множества (множественное включ-е)

```
>>> # множество scomp будет включать только русские согласные
>>> # гласные фильтруем, первую и три последних отсекаем явно
>>> scomp = {chr(x) for x in range(ord('б'),ord('ъ'))
if chr(x) != 'е' and  chr(x) != 'и' and chr(x) != 'о'
and chr(x) != 'у' and chr(x) != 'ы'}
>>> scomp
{'м', 'л', 'р', 'ж', 'н', 'т', 'п', 'ч', 'с', 'щ', 'ш', 'в',
'б', 'з', 'к', 'ц', 'г', 'ф', 'й', 'д', 'х'}
>>> # вопрос
>>> # почему не подряд?
>>>
```

Из иерархии типов: †

5. Sequences
   (упоряд-енная коллекция эл-тов с неотриц-ми индексами, длиной и срезами.)
   - строка (44 метода класса str - строка)

†см слайд 10 лек 2

Строки в Python (не байты). Строковое представление объекта. Три варианта через 3 встр-ые функции.

- последовательность символов Юникода типа *str* (неизменяемый)
- 1)*str()* –> string version of object. Если объекта нет, то пустая
  2)*repr(obj)* –> string containing a printable representation of an object, репрезентативная форма представления для возможного восстановления через встр функцию *eval()*
  3)*ascii(obj)* –> ASCII-only representation, но экранируют не-аски символы через \\x, \\u, \\U.

```
>>> ss = "строка"
>>> str(ss)
'строка'
>>> repr(ss)
"'строка'"
>>> ascii(ss)
"'\\u0441\\u0442\\u0440\\u043e\\u043a\\u0430'"
>>> # не-аски символы идут как 4 хекс цифры и как юникод (u)
```

Роль и количество кавычек

```
>>> "Vasya" == 'Vasya'
True
>>> "Вася" == 'Вася'
True
>>> "I'll be back"
"I'll be back"
>>> """ I'll      #тройными могут быть как двойные,
... be back"""    #так и одинарные кавычки
>>> " I'll \      #если на двух строках, то заэкранировать
...be back"       #перевод строки, но лучше через тройные
>>> " I'll" "be " "back"
" I'llbe back"    #склеивание строк или через +, но избыточно
>>>>>> ('mama'   #выр-е в любых скобках не требует экрана
... 'myla'
... 'ramu')
'mamamylaramu'
>>>
```

Экранированные последовательности и сырые (row) строки

```
\'
\"
\t
\n
\xhh     # символ с HEX кодом hh
\N{название}
>>> print("\ttell me")
      tell me    # с отступом в табуляцию (от 1 до 8 пробелов)
>>> print(r"\tsay A")  # row strings экраны не работают
\tsay A  # не обработанная строка -- ни экрана, ни табуляции
>>> print('\tскажи \u0410\u0410\u0410')
        скажи AAA      #русская AAA и отступ
>>>
```

От ASCII (1963) до Unicode (1991-сег) (см таб.симв-в)

Два основных раздела в юникодовой таблице:
U+0000 до U+007F содержит символы набора ASCII
Далее области знаков различных письменностей.
Кириллице выделены области знаков с кодами от
U+0400 до U+052F, от U+2DE0 до U+2DFF, от U+A640 до U+A69F

```
>>> ord('A')          >>> chr(1040)      >>> chr(ord('A'))
1040                  'A'                'A'
>>> hex(ord('A'))          # русская A
'0x410'
```

byte order mark(BOM) макер последовательности (U+FEFF)

```
>>> (0x0410).to_bytes(2, byteorder='big') # у нас порядок big
b'\x04\x10'
>>> (0x0410).to_bytes(2, byteorder='little')
b'\x10\x04'
```

```
>>> str("я строка")
'я строка'
>>> str("мама мыла раму")
'мама мыла раму'
>>> l = list(str("мама мыла раму"))
>>> l      #список с элементами
['м', 'а', 'м', 'а', ' ', 'м', 'ы', 'л', 'а', ' ', 'р', 'а', 'м
>>> l[0]                    >>> l[0], type(l[0])
'м'                         ('м', <class 'str'>)
# как представляются в памяти на примере спец значков?
>>> chr(0x20BD), chr(0x262D) #это рубль и серп и молот
(' ', ' ') # но ЛаТеХ их не берет, а питон оболочки берут
>>> [ord(i) for i in "hello"]  #через списковое включение
[104, 101, 108, 108, 111]      # получили список кодов
>>> [ord(i) for i in "привет"]
[1087, 1088, 1080, 1074, 1077, 1090]
>>> [hex(ord(i)) for i in "привет"]
['0x43f', '0x440', '0x438', '0x432', '0x435', '0x442']
>>> "\N{RUBLE SIGN}" # этот шрифт не дает прописать
```

Слайсы (вырезы) такие же как у списков и кортежей

```
>>> ss = '7531Dvorak 7531Dvorak'
>>> ss2 = ss[:5]
>>> ss2
'7531D'     # от нулевого до пятого символа
>>> ss3 = ss[4:]
>>> ss3
'Dvorak 7531Dvorak'    # от четвёртого до конца коллекции
>>> ss4 = ss[::-1]
>>> ss4
'karovD1357 karovD1357' # реверс последов-ти, с шагом 1
>>> ss5 = ss[4:10]
>>> ss5
'Dvorak'
>>> ss6 = ss[(len(ss)-1):3:-2]
>>> ss6      # от последнего через один до третьего не включая,
'krv15 aoD' # граница слайса остановилась на D
>>>
```

Методы класса str
(самая богатая функциональность – 44 метода)

- Смена регистров букв в т.ч. в указанных позициях, в т.ч. для удобства сравнения, в т.ч. в заголовках, словах в заголовках, в т.ч. свап буквомест, всех букв (6 методов)
- Выравнивание к границам, центру (3 метода)
- Удаление указанных символов справа, слева (2 метода)
- Делать из строк разное число списков по разделителям, элементы выходных списков группируются по разному (4 метода)
- Объединение строк в итерабл (2 метода)
- Нахождение (первых, непервых), индексов и подсчет количества подстрок (5 методов)
- Проверки (isчто-то) на: букву, цифру, то и другое, большую, малую, заголовок, печатность, пробел, имя (чтоб не кИвёрд был), ту или иную концовку/начало (13 методов)
- Переформатирование, перекодировка в байты (3 метода)
- Расщепление справа слева строки, нескольких строк (3 метода)
- Обрезание/заполнение либо указ-х символов, либо вайтспейсов справа слева (пробелов, таб-ий, лайнфидов, ритёрнов, формфидов, и верттабов), нулями слева (3 метода)
- Замена старой подстроки на новую (1 метод)

```
>>> # задание строки явно и через фабричную функцию
>>> aa = 'строковый объект'
>>> aa
'строковый объект'
>>> aa = str("строковый объект") #кавычки либо ", либо '
>>> aa
'строковый объект'
>>>
>>> # делаем список из 44 имён методов строк
>>> [i for i in dir(str) if "__" not in i]
['capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'maketrans', 'partition', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
'split', 'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
```

```
>>> "asd asd".capitalize()  #далее всевозмож капитализации
'Asd asd'
>>> "asd asd".upper()
'ASD ASD'    # капитализируем все буквы
>>> "asd asd".title()
'Asd Asd'
>>> "ASD ASAD".lower()
'asd asad'
>>> "asd asd2".title().swapcase()
'aSD aSD2'
>>> "asd asd2".ljust(33) # выравнивание всякие
'asd asd2
>>> "asd asd2".rjust(33)
'                         asd asd2'
>>> "asd asd2".center(33, '+')
'++++++++++++asd asd2++++++++++++'
```

```
>>> "<<asd asd2>>".lstrip("<<")  # теперь удаление с одной
# или с обеих сторон
'asd asd2>>'
>>> "<<asd asd2>>".rstrip(">>")
'<<asd asd2'
>>> "   <<asd asd2>>   ".strip()
'<<asd asd2>>'
>>> "asd asd2".split()  # делать спск с указанным разделителем
['asd', 'asd2']         # или пробел, по дефолту
>>> "asd,asd2".split(',')
['asd', 'asd2']
>>> "asd,asd2,asd,asd2".split(',',1)
['asd', 'asd2,asd,asd2'] # все разделять не нужно дай только
# первое разделение (rsplit() работает справа)
>>> "asd,asd2,asd,asd2".rsplit(',',1)
['asd,asd2,asd', 'asd2']
>>>
```

```
>>> "asd,asd2,asd,asd2".partition(',')
('asd', ',', 'asd2,asd,asd2')
>>> "asd,asd2,asd,asd2".rpartition(',')
('asd,asd2,asd', ',', 'asd2')
# partition всегда возвращает кортеж из трех эл-тов, каких?
# join соединяет строки через указанный разделитель
>>> ' '.join(["asd", "asd2", "asd", "asd2"])
'asd asd2 asd asd2'
>>> ','.join(["asd", "asd2", "asd", "asd2"])
'asd,asd2,asd,asd2'
>>> проверка на вхождение
>>> "as" in "asd asd2 asd asd2"
True
>>> "bs" in "asd asd2 asd asd2"
False
>>>
```

```
>>> "asd asd2 asd asd2".find("sd")
1
>>> "asd asd2 asd asd2".find("sd", 2)
5
>>> "asd asd2 asd asd2".find("bd")
-1    # по Си- шному
>>> "asd asd2 asd asd2".index("sd")
1
>>> "asd asd2 asd asd2".index("bd")
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ValueError: substring not found
>>> find и index возвращают по-разному
>>> "asd asd2 asd asd2".replace("asd", "aassdd")
'aassdd aassdd2 aassdd aassdd2'
>>> "asd asd2 asd asd2".replace("asd", "aassdd", 1)
'aassdd asd2 asd asd2'
>>>
```

метод translate для множественной замены через таблицу транслитерации

```
>>> translation_map = {ord("a"): "*", ord("s"): "&"}
>>> "asd asd2 asd asd2".translate(translation_map)
'*&d *&d2 *&d *&d2'
>>> "1040".isdigit()    # предикаты
True
>>> "0x1040".isalnum()
True
>>> "x".isalpha()
True
>>> "asd asd2 asd asd2".islower()
True
>>> "ASD ASD2".isupper()
True
>>> "Asd Asd2".istitle()
True
```

метод format (2 способа форматирования строк)

```
>>> "{} , {}, how are you?".format("hello", "John")
'hello , John, how are you?'
>>> "На дворе {} год".format(2017)
'На дворе 2017 год'
```

{} обозначает место, в кот будет подставлен позиционный аргуме
внутри {} можно опционально указать способ преобразования в стр
Три способа сделать из объекта строку\\

```
>>> str("я строка")
'я строка'
>>> repr("я строка")
"'я строка'"
>>> ascii("я строка")
"'\\u044f \\u0441\\u0442\\u0440\\u043e\\u043a\\u0430'"
>>> # где пробел?
```

форматирование строк {}

```
>>> "{!s}".format("я строка") # str
'я строка'
>>> "{!r}".format("я строка") # repr
"'я строка'"
>>> "{!a}".format("я строка") # ascii
"'\\u044f \\u0441\\u0442\\u0440\\u043e\\u043a\\u0430'"
>>>
```

модуль string и его методы

умножение и конкатенация строк

```
>>> a = str('*')
>>> a
'*'
>>> a *= 4
>>> a
'****'
>>> import string
>>> string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.digits
'0123456789'
>>> string.punctuation
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
>>> string.octdigits
'01234567'
>>>
```

Читаем аргументы командной строки

sys.argv[] список подстрок, которые мы должны ввести в командной строке при запуске программы

```
1 import math
2 import sys
3 a = int(sys.argv[1])
4 b = int(sys.argv[2])
5 c = int(sys.argv[3])
6 print("a = ", a, "b = ", b, "c = ", c)
7 P = (a + b + c) / 2
8 Geron = math.sqrt(P * (P - a) * (P - b) * (P - c))
9 print("Площадь треугольника по Герону будет: ", Geron)
```

Запускаем файл geron.py с нужными a b c в виде аргументов

```
01702$ python geron.py 4 2 5
a =  4 b =  2 c =  5
Площадь треугольника по Герону будет:  3.799671038392666
01702$
```

Генераторные выражения.

genexp экономит пямять, отдает элементы по одному

```
>>> vowel_symbols = 'ауоыиэяюёе' # гласные в русском
>>> tuple(ord(symbol) for symbol in vowel_symbols)
(1072, 1091, 1086, 1099, 1080, 1101, 1103, 1102, 1105, 1077)
>>># теперь согласные в русском
>>> consonant_symbols = 'бвгджзйклмнпрстфхцчшщ'
>>> tuple(ord(symbol) for symbol in consonant_symbols)
(1073, 1074, 1075, 1076, 1078, 1079, 1081, 1082, 1083, 1084,
1085, 1087, 1088, 1089, 1090, 1092, 1093, 1094, 1095,1096,1097)
>>>
>>> import array  # теперь массив из гласных
array.array('I', (ord(symbol) for symbol in vowel_symbols))
array('I', [1072, 1091, 1086, 1099, 1080, 1101, 1103, 1102,
 1105, 1077])
>>> # genexp идет вторым аргументом и поэтому стоит в скобках
>>>
```

Итак! (выводы по Лек. 3)

Финиш Лек 3

the End по Лекции 3, вопросы?

Далее

Вопросы(для экзамена)/упражнения(для лабораторных)

28. тип у словарей называется mappings – отображения. Что отображается и в чего? и как называется то, что отображается и то, куда отображается?
29. если оператором in (not in) мы проверяем входит ли элемент в коллекцию, то для словарей через in проверяем что?
30. кортеж, как нам говорили на лекции – неизменяемая коллекция - контейнер, но как проверить, покажите? (через встр функцию на русскую букву 'х'. Еще, что будет, если в кортеж вставить изменяемый объект, сохраниться ли его неизменяемость?
31. я слышал о пяти ролях, применения кортежа где и зачем, назовите хотя бы две?
32. что общего между множеством и словарем, кроме того, что оба пишутся в фигурных скобках?
33. можно ли проитерироваться по множеству или словарю? либо скажите почему нельзя, либо продемонстрируйте?
34. Известно, что слияние словарей делается методом update(). Но что будет если у обоих есть одинаковые ключи. Пусть dd1 = {1: 10, что-то_там_еще}, а dd2 = {1: 20, что-то_там_еще2}. Как сделать слияние и чего с чем, чтобы осталось значение 20?

34. как узнать принадлежит ли данный элемент данному множеству?
35. три главных "магических" слова из чего состоит словарь?
36. если составлять записи в базе дынных где одна таблица показывает успеваемость всей группы по одному предмету, то как представить запись в виде словаря, с какими обязательными составляющими, типа ФИО студента, оценка, кто принимал экзамен и т.п. (профантазируйте, чтобы не было противоречий и недопоняток)?
37. в множество я могу с помощью метода add() добавить не только одиночные новые элементы, но и какие-то коллекции, вопрос какие? или может всякие?
38. как называется операция, когда нужно проверить, что у меня совпадают сразу несколько значений данных из множества?
39. прокомментируйте, каким объектам что и как присваивается и почему айди получается разный?, слева и справа (от '=') что?

```
>>> ass2 = 1, 4, 'q'
>>> id(ass2)
140234758406288
>>>
```

```
>>> ass2 = ass2, 1, 4, 'q'
>>> id(ass2)
140234758917880
>>>
```

40. ограничение по типу на ключ и на значение есть в словаре? а, если есть, то какие?
41. является ли множество подмножеством самого себя и как проверить? (м.б. из help видно?)
42. оператор [] (квдр скобки) в Питоне делает много чего, в частности с помощью него берутся индексы в коллекциях, но не во всех, в каких не берутся?, а в какой он очень редко когда по циферкам?
43. почему у множества не работают слайсы?, хочу поместить в пустое множество frozenset() интОвую 1, как?
44. если строка это плоская последовательность, а кортеж – контейнерная, то что это означает в смысле содержимого по каждому внутреннему элементу для той и другой коллекции?
45. хочу объявить одноэлементный кортеж и написал ss = ('a'), но он говорит, что это строка, где ошибка?
46. Назовите 2 определения хешируемого, хешбл (hashable) объекта: одно однословное на букву из конца первой половины русского алфавита, другое вытекает из функциональных возможностей Питоновских объектов?

47. Из кортежа
t = ('привет', 'от', 'васи', 'пупкина', '!')
сделать в одну строку кортеж
(['привет'], ['от'], ['васи'], ['пупкина'], ['!'])