19 сентября 2022 года, ВМ/462 (ул. Верхняя Масловка, д. 15)

Студенты: Поток УЦИ21-1, УЦИ-21-2.

Преподаватель: Смирнов Михаил Викторович,
доцент Департамента Анализа данных и машинного обучения
Финансового университета при Правительстве Российской Федерации

Лекция

Тема 2. Строки и коллекции

Оглавление

Оглавление|| =>

1. Строки

На этой лекции продолжим изучение свойств строковых переменных и приемов работы с ними.

Строковая величина в *Python* - это неизменяемый набор символов в формате *Unicode*.

Минимальная строка - пустая:

In [1]:

```
s = ''
print(s)
```

Допустимое количество элементов в строке зависит от технических характеристик компьютера. В Python можно обрабатывать строки, объемом в несколько десятков гигабайт.

Нам уже известно, что:

- строки можно складывать друг с другом.

In [2]:

```
'Строка 1' + ', ' + 'Строка 2'
```

Out[2]:

```
'Строка 1, Строка 2'
```

- можно умножать на число.

In [3]:

```
'@fa.ru, '*10
```

Out[3]:

```
'@fa.ru, @fa.ru, @fa.ru, @fa.ru, @fa.ru, @fa.ru, @fa.ru, @fa.ru, @fa.ru, @f
a.ru, '
```

- переводить в верхний и нижний регистры

In [4]:

```
'строка'.upper() + 'ПОДСТРОКА'.lower()
```

Out[4]:

```
'СТРОКАподстрока'
```

- функция `len()` позволяет узнать, сколько символов в строке.

In [5]:

```
len('Сколько символов в этой строке')
```

Out[5]:

```
30
```

Рассмотрим другие возможности для работы со строками.

Опреатор *in*

Позволяет ответить на вопрос, содержит ли строка подстроку: да или нет.

In [6]:

```
'просп.' in 'г. Москва, Ленинградский просп., 49'
```

Out[6]:

```
True
```

По символам строки можно пройти в цикле. Строка - итерируемый объект. Итерируемый объект - это объект, позволяющий поочередно перебирать составляющие его элементы.

In [7]:

```
s = 'г. Москва, Ленинградский просп., 49'
for symbol in s:
    print(symbol, end=',')
```

```
г,., ,М,о,с,к,в,а,,, ,Л,е,н,и,н,г,р,а,д,с,к,и,й, ,п,р,о,с,п,.,,, ,4,9,
```

Найдем, сколько знаков препинания в строке *s*.

In [8]:

```
s = 'г. Москва, Ленинградский просп., 49'
k = 0 # счетчик
for symbol in s:
    if symbol in '.,;':
        k+=1
print(k)
```

```
4
```

Метод *find()*

Иногда важно не только установить факт наличия подстроки в составе строки, но и определить, где находится подстрока. В этом случае можно воспользоваться методом *.find()*. Могут быть указаны необязательные параметры *start* и *end* - индексы символов начала и конца поиска. Если эти параметры не указаны, то поиск ведется по всей строке.

```
number=search_here.find(find_me, sart, end)
```

Метод *find* возвращает индекс позиции, с которой начинается искомая подстрока. Если подстрока не найдена, метод возвращает -1.

Найдем позицию первой и второй запятых в строке *s*.

In [1]:

```
s = 'г. Москва, Ленинградский просп., 49'
c1 = s.find(',')
c2 = s.find(',',c1+1)
print(c1, c2)
```

```
9 31
```

Метод *count()*

Для подсчёта количества вхождений подстроки в строку используется метод *count()*.

```
number=search_here.count(find_me, sart, end)
```

Сколько раз без учета регистра встречается подстрока "толк" в скороговорке "Толком толковать, да без толку расперетолковывать"?

In [10]:

```
"Толком толковать, да без толку расперетолковывать".lower().count("толк")
```

Out[10]:

```
4
```

Метод *strip()*

Удаляет начальные и конечные пробелы.

In [11]:

```
b = "   Здравствуй, мир!"
b.strip()
```

Out[11]:

```
'Здравствуй, мир!'
```

Метод *replace()*

Часто случается так, что числовые данные поступают в текстовом формате, и при этом у дробных чисел в качестве десятичного разделителя используется не точка, а запятая. Перевести такое «число» в формат *float* на Python невозможно: интерпретатор прервёт выполнение программы и выдаст сообщение об ошибке. Для корректного преобразования такой строки в вещественное число заменим использующуюся в качестве разделителя запятую на точку. Это можно сделать с помощью метода *replace()*.

In [12]:

```
b = "Здравствуй, мир!"
b.replace("Здравствуй", "Привет")
```

Out[12]:

```
'Привет, мир!'
```

Задание.
Имеется строка "На берегу моря был камень". С помощью метода *replace()* напишите код, который преобразует ее в строку "На берегу океана был дом".

Метод *zfill()*

Синтаксис: `string.zfill(len)`

Метод *zfill()* добавляет нули к началу строки до достижения указанной длины строки.

In [4]:

```
a = "Привет!"
b = "Добро пожаловать!"
c = "10.000"

print(a.zfill(10))
print(b.zfill(10))
print(c.zfill(10))
```

```
000Привет!
Добро пожаловать!
000010.000
```

Индексы и срезки

- Первый элемент строки имеет индекс 0.
- Возможно использование отрицательных индексов.
- Элементы строки можно отбирать с помощью срезки: `string[start:end:step]`
- Шаг `step` может быть отрицательным.

In [13]:

```
b = "Здравствуй, мир!"
b[2:5]
```

Out[13]:

```
'рав'
```

In [14]:

```
b = 'abcdefghij'
b[1::2]
```

Out[14]:

```
'bdfhj'
```

In [15]:

```
b[-1:3:-1]
```

Out[15]:

```
'jihgfe'
```

Построчное чтение файла

In [1]:

```
f=open('../Data/Студенты.txt', encoding='utf-8')
#f=open('../Data/Студенты.txt', encoding='cp1251') # Если кодировка Windows

for k, line in enumerate(f):
    print(line, end='')

print('\nВсего', k, 'студентов')
```

```
Авдошкин Даниил Дмитриевич
Акимов Макар Витальевич
Бураго Владислав Вячеславович
Веденеев Никита Олегович
Грицаев Андрей Игоревич
Давидов Моисей Изьяевич
Ерин Егор Вадимович
Ерохин Андрей Владимирович
Иванова Вероника Александровна
Кременский Кирилл Вадимович
Круговых Роман Георгиевич
Назаркин Николай Борисович
Никитин Георгий Андреевич
Орищин Андрей Сергеевич
Петренко Александра Николаевна
Подвальный Виктор Андреевич
Рябцев Максим Николаевич
Талачёв Павел Денисович
Федулов Даниил Романович
Хандожко Андрей Дмитриевич
Хмелевская Екатерина Андреевна
Шабарьков Иван Евгеньевич
Шелкоплясов Иван Алексеевич
Айдамиров Магомед
Аванесов Артём Арменович
Артищев Роман Сергеевич
Артюхович Иван Александрович
Аюпов Арсен Альбертович
Балаян Даниил Арсенович
Васенков Никита Михайлович
Дородников Леонид Павлович
Евдокимов Артем Алексеевич
Зубова Маргарита Павловна
Иванова Наталья Игоревна
Игошин Алексей Владиславович
Меняйлов Иван Дмитриевич
Наумов Марк Алексеевич
Парнюгин Никита Олегович
Писарева Карина Игоревна
Погост Александр Дмитриевич
Проявко Екатерина Евгеньевна
Свириденко Илья Вадимович
Семенова Екатерина Александровна
Столбов Александр Максимович
Тимошенко Василий Геннадьевич
Фомин Иван Сергеевич
Хафизова Элина Маратовна
Шляхтин Владислав Олегович

Всего 47 студентов
```

Поиск с помощью регулярных выражений

Документация по модулю *re:* https://docs.python.org/3.7/library/re.html (https://docs.python.org/3.7/library/re.html)
Дополнительно: статья о регулярных выражениях https://habr.com/ru/post/349860/ (https://habr.com/ru/post/349860/)

Импорт модуля регулярных выражений

In [27]:

```
import re
```

Создание шаблона

In [28]:

```
pattern=re.compile('[А-Я][а-я]*на')
```

Методы регулярных выражений

```
match - ищет символы по шаблону в начале строки, возвращает объект или None
search - ищет символы по шаблону во всей строке, возвращает объект или None
findall - ищет символы по шаблону во всей строке, возвращает список значений
```

Применим шаблон

In [29]:

```
s = 'Арина Родионовна'
```

In [30]:

```
pattern.match(s)
```

Out[30]:

```
<re.Match object; span=(0, 5), match='Арина'>
```

In [31]:

```
pattern.search(s)
```

Out[31]:

```
<re.Match object; span=(0, 5), match='Арина'>
```

In [32]:

```
pattern.findall(s)
```

Out[32]:

```
['Арина', 'Родионовна']
```

In [33]:

```
s = 'Александр Сергеевич'
print(pattern.findall(s))
print(len(pattern.findall(s)))
```

```
[]
0
```

Распечатаем только имена девушек, заодно посчитаем сколько их.

In [53]:

```
f = open('../Data/Студенты.txt', encoding='utf-8')
n, fe = 0, 0 # Счетчики: n - всех студентов, fe - девушек

pattern=re.compile('[А-Я][а-я]*\bна')

for line in f:
    n += 1
    if len(pattern.findall(line)) != 0:
        fe += 1
        print(line, end='')
print('\n---')
print('Всего студентов: ', n)
print('Юношей: ', n-fe)
print('Девушек: ', fe)
```

```
---
Всего студентов:  26
Юношей:  26
Девушек:  0
```

In [39]:

```
pattern.findall('Антонов Владислав')
```

Out[39]:

```
['Владисла']
```

Метод *split()*

Разделяет строку на элементы списка по разделителю. Если разделитель не указан, то в качестве разделителя используется символ пробела.

In [14]:

```
s = "Я студент Финансового университета"
list1 = s.split()
print(list1)
```

```
['Я', 'студент', 'Финансового', 'университета']
```

В следующем примере укажем, что разделителем является дефис.

In [15]:

```
s = "Рахат-лукум"
list1 = s.split("-")
print(list1)
```

```
['Рахат', 'лукум']
```

Метод *join()*

Соединяет элементы итерируемого объекта.

In [16]:

```
list1 = ["Рахат", "лукум"]
s = "-".join(list1)
print(s)
```

```
Рахат-лукум
```

Методы работы со строками

In [2]:

```
with open("../Data/Strings methods.csv") as f:
    content=f.readlines()
for item in content:
    print(item.replace(";", "  \t\t"), end = "")
```

```
Method                  Description
capitalize()            Converts the first character to upper case
casefold()              Converts string into lower case
center()                Returns a centered string
count()                 Returns the number of times a specified value occurs
in a string
encode()                Returns an encoded version of the string
endswith()              Returns true if the string ends with the specified v
alue
expandtabs()            Sets the tab size of the string
find()                  Searches the string for a specified value and return
s the position of where it was found
format()                Formats specified values in a string
format_map()            Formats specified values in a string
index()                 Searches the string for a specified value and return
s the position of where it was found
isalnum()               Returns True if all characters in the string are alp
hanumeric
isalpha()               Returns True if all characters in the string are in
the alphabet
isdecimal()             Returns True if all characters in the string are dec
imals
isdigit()               Returns True if all characters in the string are dig
its
isidentifier()                  Returns True if the string is an identifier
islower()               Returns True if all characters in the string are low
er case
isnumeric()             Returns True if all characters in the string are num
eric
isprintable()           Returns True if all characters in the string are pri
ntable
isspace()               Returns True if all characters in the string are whi
tespaces
istitle()               Returns True if the string follows the rules of a ti
tle
isupper()               Returns True if all characters in the string are upp
er case
join()                  Joins the elements of an iterable to the end of the
string
ljust()                 Returns a left justified version of the string
lower()                 Converts a string into lower case
lstrip()                Returns a left trim version of the string
maketrans()             Returns a translation table to be used in translatio
ns
partition()             Returns a tuple where the string is parted into thre
e parts
replace()               Returns a string where a specified value is replaced
with a specified value
rfind()                 Searches the string for a specified value and return
s the last position of where it was found
rindex()                Searches the string for a specified value and return
s the last position of where it was found
rjust()                 Returns a right justified version of the string
rpartition()            Returns a tuple where the string is parted into thre
```

```
e parts
rsplit()                Splits the string at the specified separator, and re
turns a list
rstrip()                Returns a right trim version of the string
split()                 Splits the string at the specified separator, and re
turns a list
splitlines()            Splits the string at line breaks and returns a list
startswith()            Returns true if the string starts with the specified
value
strip()                 Returns a trimmed version of the string
swapcase()              Swaps cases, lower case becomes upper case and vice
versa
title()                 Converts the first character of each word to upper c
ase
translate()             Returns a translated string
upper()                 Converts a string into upper case
zfill()                 Fills the string with a specified number of 0 values
at the beginning
```

<= || Оглавление || =>

2. Списки

Список в Python - это упорядоченная изменяемая коллекция объектов произвольных типов

In [17]:

```
list1 = ["яблоко", "банан", "вишня"]
print(list1)
```

```
['яблоко', 'банан', 'вишня']
```

Элементы списка упорядочены, изменяемы и допускают повторяющиеся значения. Элементы списка проиндексированы, первый элемент имеет индекс [0], второй элемент - индекс [1] и т. д. При добавлении новых элементов, они помещаются в конец списка.

Функция *len()*

Чтобы определить, сколько элементов в списке.

In [18]:

```
list1 = ["яблоко", "банан", "вишня"]
print(len(list1))
```

```
3
```

Элементы списка могут быть любого типа данных

In [19]:

```
list1 = ["яблоко", "банан", "вишня"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

Список может содержать разные типы данных

In [20]:

```
list1=["abc", 34, True, 40, "студент"]
list1
```

Out[20]:

```
['abc', 34, True, 40, 'студент']
```

type()

Если *lst1* - список, то об этом можно узнать с помощью функции *type()*

In [21]:

```
type(list1)
```

Out[21]:

```
list
```

Конструктор *list()*

Помещает данные в список, если они хранятся в другой структуре, например, создает список из кортежа.

In [22]:

```
list1 = list(("яблоко", "банан", "вишня")) # двойные круглые скобки
print(list1)
```

```
['яблоко', 'банан', 'вишня']
```

Индексы и срезы

Доступ к элементам списка осуществляется по индексу.

In [23]:

```
list1 = ["яблоко", "банан", "вишня", "апельсин", "киви", "дыня", "манго"]
print(list1[1])
print(list1[-1])
print(list1[::2])
print(list1[1::2])
print(list1[2:5])
print(list1[-3:-6:-1])
```

```
банан
манго
['яблоко', 'вишня', 'киви', 'манго']
['банан', 'апельсин', 'дыня']
['вишня', 'апельсин', 'киви']
['киви', 'апельсин', 'вишня']
```

Проверка на наличие в списке

In [24]:

```
list1 = ["яблоко", "банан", "вишня"]
if "яблоко" in list1:
  print("Да, 'яблоко' в списке фруктов")
```

```
Да, 'яблоко' в списке фруктов
```

Замена элементов списка

In [25]:

```
list1 = ["яблоко", "банан", "вишня"]
list1[1] = "черная смородина"
list1
```

Out[25]:

```
['яблоко', 'черная смородина', 'вишня']
```

Замена диапазона значений

In [26]:

```
list1 = ["яблоко", "банан", "вишня", "апельсин", "киви", "дыня", "манго"]
list1[2:6] = ["виноград", "груша"]
list1
```

Out[26]:

```
['яблоко', 'банан', 'виноград', 'груша', 'манго']
```

Метод *insert()*

In [27]:

```
list1 = ["яблоко", "банан", "вишня"]
list1.insert(2,"грейпфрут")
list1
```

Out[27]:

```
['яблоко', 'банан', 'грейпфрут', 'вишня']
```

Метод *append()*

In [28]:

```
list1 = ["яблоко", "банан", "вишня"]
list1.append("черешня")
list1
```

Out[28]:

```
['яблоко', 'банан', 'вишня', 'черешня']
```

Метод *extend()*

Чтобы добавить элементы из другого списка в текущий список.

In [29]:

```
list1 = ["яблоко", "банан", "вишня"]
list2 = ["калина", "рябина", "облепиха"]
list1.extend(list2)
list1
```

Out[29]:

```
['яблоко', 'банан', 'вишня', 'калина', 'рябина', 'облепиха']
```

Методы *remove(), pop(), clear()*

In [30]:

```
list1 = ["яблоко", "банан", "вишня", "апельсин", "киви", "дыня", "манго"]
list1.remove("вишня")
print(list1)
```

```
['яблоко', 'банан', 'апельсин', 'киви', 'дыня', 'манго']
```

In [31]:

```
list1 = ["яблоко", "банан", "вишня", "апельсин", "киви", "дыня", "манго"]
list1.pop(2)
print(list1)
```

```
['яблоко', 'банан', 'апельсин', 'киви', 'дыня', 'манго']
```

In [32]:

```
list1 = ["яблоко", "банан", "вишня", "апельсин", "киви", "дыня", "манго"]
list1.pop()
print(list1)
```

```
['яблоко', 'банан', 'вишня', 'апельсин', 'киви', 'дыня']
```

In [33]:

```
list1.clear()
print(list1)
```

```
[]
```

Метод *sort()*

In [34]:

```
list1 = ["яблоко", "банан", "вишня", "апельсин", "киви", "дыня", "манго"]
list1.sort()
print(list1)
```

```
['апельсин', 'банан', 'вишня', 'дыня', 'киви', 'манго', 'яблоко']
```

In [35]:

```
list1 = ["яблоко", "банан", "вишня", "апельсин", "киви", "дыня", "манго"]
list1.sort(reverse = True)
print(list1)
```

```
['яблоко', 'манго', 'киви', 'дыня', 'вишня', 'банан', 'апельсин']
```

<= || [Оглавление] || =>

3. Словари

Словарь - упорядоченная изменяемая коллекция. Каждым элементом словаря является пара ключ - значение.

In [36]:

```
dict1 = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

К значениям словаря можно обращаться по ключу.

In [37]:

```
dict1['brand']
```

Out[37]:

```
'Ford'
```

В словарях не допускаются дупликаты

In [38]:

```
dict1 = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
dict1
```

Out[38]:

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

Значениями могут быть данные любых типов

In [39]:

```
dict1 = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["красный", "белый", "голубой"]
}
```

Метод *keys()*

In [40]:

```
dict1.keys()
```

Out[40]:

```
dict_keys(['brand', 'electric', 'year', 'colors'])
```

Метод *values()*

In [41]:

```
dict1.values()
```

Out[41]:

```
dict_values(['Ford', False, 1964, ['красный', 'белый', 'голубой']])
```

Метод *items()*

In [42]:

```
dict1.items()
```

Out[42]:

```
dict_items([('brand', 'Ford'), ('electric', False), ('year', 1964), ('color
s', ['красный', 'белый', 'голубой'])])
```

Проверка на наличие ключа в словаре

In [43]:

```
if 'year' in dict1:
    print(dict1['year'])
else:
    print('Такого ключа нет')
```

```
1964
```

Изменение значений словаря

In [44]:

```
dict1 = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
dict1["year"] = 2018

print(dict1)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

Метод *pop()*

Метод *pop()* удаляет элемент словаря по имени ключа

In [45]:

```
dict1 = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
dict1.pop("brand")
print(dict1)
```

```
{'model': 'Mustang', 'year': 1964}
```

Метод *clear()* опустошает словарь

In [46]:

```
dict1.clear()
print(dict1)
```

```
{}
```

Проход по словарю в цикле

Распечатать имена всех ключей

In [47]:

```
dict1 = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in dict1:
    print(x)
```

```
brand
model
year
```

Распечатать все значения

In [48]:

```
for x in dict1:
    print(dict1[x])
```

```
Ford
Mustang
1964
```

Методы *keys()* и *values()*

Для доступа к именам ключей и значениям можно также использовать методы *keys()* и *values()*

In [49]:

```
for x in dict1.keys():
    print(x)
```

```
brand
model
year
```

In [50]:

```
for x in dict1.values():
    print(x)
```

```
Ford
Mustang
1964
```

Метод *items()*

Метод *items()* используется для доступа как к именам ключей, так и значениям

In [51]:

```
for x, y in dict1.items():
    print(x, y)
```

```
brand Ford
model Mustang
year 1964
```

Словари можно вкладывать друг в друга

In [52]:

```
park = {
    'car1' : {
        'model' : 'Ford',
        'brand' : 'Mustang',
        'year' : 1996
    },
    'car2' : {
        'model' : 'Lada',
        'brand' : 'Granta',
        'year' : 2020
    }
}
print(park)
```

```
{'car1': {'model': 'Ford', 'brand': 'Mustang', 'year': 1996}, 'car2': {'mode
l': 'Lada', 'brand': 'Granta', 'year': 2020}}
```

<= || Оглавление || =>

4. Кортежи

Кортеж - это упорядоченная неизменяемая коллекция. Для создания кортежа используются круглые скобки.

In [53]:

```
t1 = ("рябина", "шиповник", "облепиха")
print(t1)
```

```
('рябина', 'шиповник', 'облепиха')
```

Элементы кортежа упорядочены, неизменяемы и допускают повторяющиеся значения. Элементы кортежа индексируются, первый элемент имеет индекс [0], второй элемент имеет индекс [1] и т. д.

Когда говорят, что кортежи упорядочены, это означает, что элементы имеют определенный порядок, и этот порядок не изменяется.

Кортежи неизменяемы, то есть мы не можем изменять, добавлять или удалять элементы после создания кортежа.

Поскольку кортежи индексируются, они могут иметь элементы с одинаковыми значениями.

In [54]:

```
t1 = ("рябина", "шиповник", "облепиха", "рябина", "шиповник")
print(t1)
```

```
('рябина', 'шиповник', 'облепиха', 'рябина', 'шиповник')
```

Чтобы создать кортеж из одного элемента, необходимо добавить запятую, иначе Python не распознает его как кортеж.

In [55]:

```
t1 = ("рябина",)
print(type(t1))

#Не кортеж
t2 = ("рябина")
print(type(t2))
```

```
<class 'tuple'>
<class 'str'>
```

Кортеж может содержать разные данные разных типов

In [56]:

```
t1 = ("ху", 43, True, "40")
```

Конструктор *tuple()*

Конструктор *tuple()* используется для создания кортежей

In [57]:

```
t1 = tuple(["рябина", "шиповник", "облепих"])
print(t1)
```

```
('рябина', 'шиповник', 'облепих')
```

Доступ к элементам кортежа

In [58]:

```
t1 = tuple(["рябина", "шиповник", "облепиха", "смородина"])
print(t1[1])
print(t1[-1])
print(t1[-1:1:-1])
```

```
шиповник
смородина
('смородина', 'облепиха')
```

<= || Оглавление || =>

5. Множества

Множество - это неупорядоченная и неиндексированная коллекция. Элементы множества неупорядочены, неизменяемы и не допускают повторяющихся значений. На элемент множества нельзя сослаться по индексу. Мы не можем изменять значения элементов множества, но можем добавлять новые элементы. В множестве не может быть двух элементов с одинаковыми значениями. Для множества используются фигурные скобки.

Элементы множества могут быть любого типа

In [59]:

```
set1={"a","b","c","u",3,6,9,True}
set1
```

Out[59]:

```
{3, 6, 9, True, 'a', 'b', 'c', 'u'}
```

Дупликаты в множествах не допускаются

In [60]:

```
set1={"a","v","a"}
set1
```

Out[60]:

```
{'a', 'v'}
```

Тип данных - множество

In [61]:

```
print(type(set1))
```

```
<class 'set'>
```

Конструктор *set()*

Конструктор *set()* используется для создания множества. Создадим множество из кортежа.

In [62]:

```
my_set=set(("яблоко","банан","черешня")) # ! Двойные круглые скобки
print(my_set)
```

```
{'банан', 'яблоко', 'черешня'}
```

Используем конструктор для создания множества из строки

In [63]:

```
set1=set('статистика')
print(set1)
```

```
{'и', 'к', 'т', 'с', 'а'}
```

Метод *add()*

Метод *add()* используется для добавления элементов множества

In [64]:

```
set2={"человек","лев","орел"}
set2.add("куропатка")
print(set2)
```

```
{'орел', 'куропатка', 'лев', 'человек'}
```

Метод *update()*

In [66]:

```
set1={"человек","лев","орел","куропатка"}
set2={"рогатый олень", "морская звезда"}
set1.update(set2)
print(set1)
```

```
{'орел', 'куропатка', 'лев', 'морская звезда', 'рогатый олень', 'человек'}
```

Удаление элементов множества с помощью *remove()* и *discard()*

In [42]:

```
set1={"человек","лев","орел","куропатка"}
set1.remove("орел")
print(set1)
```

```
{'куропатка', 'лев', 'человек'}
```

In [43]:

```
set1={"человек","лев","орел","куропатка"}
set1.discard("лев")
print(set1)
```

```
{'куропатка', 'орел', 'человек'}
```

Если удаляемый элемент отсутствует, метод *remove()* приводит к ошибке. Метод *discard()* не приводит к ошибке.

In [44]:

```
set1={"человек","лев","орел","куропатка"}
set1.discard("утка")
print(set1)
```

```
{'куропатка', 'лев', 'орел', 'человек'}
```

Метод *pop()*

Удаляет последний элемент множества. Возвращает удаленный элелемент. Так как элементы множества неупорядочены, не известно, какой элемент будет удален.

In [46]:

```
set1={"человек","лев","орел","куропатка"}
element=set1.pop()
print(element)
```

```
куропатка
```

Метод *clear()*

Метод *clear()* очищает множество

In [47]:

```
set1={"человек","лев","орел","куропатка"}
set1.clear()
print(set1)
```

```
set()
```

Команда *del*

Команда *del* удаляет множество

In [ ]:

```
set1={"человек","лев","орел","куропатка"}
del set1
print(set1) # Приводит к ошибке, так как множество set1 не существует
```

Цикл по элементам множества

In [49]:

```
set1={"человек","лев","орел","куропатка"}
for x in set1:
    print(x)
```

```
куропатка
лев
орел
человек
```

Объединение двух множеств

Для объединения двух множеств используются методы *update()* и *union()*. Метод *update()* добавляет к множеству элементы другого множества. Метод *union()* создает новое множество из двух объединяемых множеств.

In [50]:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

```
{1, 2, 3, 'c', 'a', 'b'}
```

Метод *intersection_update()*

Возвращает элементы, присутствующие в обоих множествах

In [51]:

```
x = {"бериллий", "бор", "углерод"}
y = {"лес", "бор", "роща"}

x.intersection_update(y)

print(x)
```

```
{'бор'}
```

Метод *symmetric_difference_update()*

Находит неповторяющиеся элементы в обоих множествах

In [52]:

```
x = {"бериллий", "бор", "углерод"}
y = {"лес", "бор", "роща"}

x.symmetric_difference_update(y)

print(x)
```

```
{'роща', 'бериллий', 'углерод', 'лес'}
```

<= || Оглавление || =>

Задания

1. Распечатать второе значение списка.

In [64]:

```
fruits = ["яблоко", "банан", "вишня"]
# Ваш код здесь
```

2. Поменять значение "яблоко" на "киви".

In [65]:

```
fruits = ["яблоко", "банан", "вишня"]
# Ваш код здесь
```

3. Добавить к списку "апельсин".

In [66]:

```
fruits = ["яблоко", "банан", "вишня"]
# Ваш код здесь
```

4. В имеющийся список *fruits* вставить элемент "лимон" на вторую позицию.

In [67]:

```
fruits = ["яблоко", "банан", "вишня"]
# Ваш код здесь
```

5. Из списка *fruits* удалить элемент "банан".

In [68]:

```
fruits = ["яблоко", "банан", "вишня"]
# Ваш код здесь
```

6. Имеется строка `'SIX, SEVEN, EIGHT, NINE, TEN'` . Инвертировать последовательность слов, разделенных запятыми. Ожидается результат: `'TEN, NINE, EIGHT, SEVEN, SIX'` .

In [69]:

```
s = "SIX, SEVEN, EIGHT, NINE, TEN"
# Ваш код здесь
```

7. В строке, содержащей последовательность слов, обозначающих числа

   `'НОЛЬ,ОДИН,ДВА,ТРИ,ЧЕТЫРЕ,ПЯТЬ,ШЕСТЬ,СЕМЬ,ВОСЕМЬ,ДЕВЯТЬ'`

   удалить слова, обозначающие нечетные числа. Ответ представить в виде строки.

In [70]:

```
s='НОЛЬ,ОДИН,ДВА,ТРИ,ЧЕТЫРЕ,ПЯТЬ,ШЕСТЬ,СЕМЬ,ВОСЕМЬ,ДЕВЯТЬ'
# Ваш код здесь
```

8. Дана статистика расходов на рекламную кампанию по каналу ('source'), расходам ('cost') и прибыли от рекламы ('profit').

```
ads = [
 {'cost':98, 'source':'vk', 'profit':150},
 {'cost':153, 'source':'yandex', 'profit':270},
 {'cost':110, 'source':'facebook', 'profit':201}
]
```

   Найти канал, обеспечивший максимальную эффективность = прибыль/расходы. Распечатать название канала и значение эффективности, округленное до двух десятичных знаков.

In [71]:

```
ads = [
    {'cost':98, 'source':'vk', 'profit':150},
    {'cost':153, 'source':'yandex', 'profit':270},
    {'cost':110, 'source':'facebook', 'profit':201}
]

# Ваш код здесь
```

9. Представить данные Задачи 8 в табличном виде, вывести эффективность в отдельный столбец и найти максимально эффективную организацию, оперируя данными таблицы.

10. На языке Markdown написать формулу y = Интеграл от a до b от x квадрат dx

Оглавление