

## Въведение в Машинното обучение

### Лабораторно упражнение №3

#### Работа с множества, модули, функции, файлове и графики

1. **Множества (set)** – всеки елемент се съдържа точно един път. Елементите може да не са подредени. Създават се със служебната дума **set** ( [ списък / комплект ] ) или {елемент<sub>1</sub>, елемент<sub>2</sub>, ..., елемент<sub>n</sub>}. Множеството трябва да съдържа поне един елемент.

##### Пример 1:

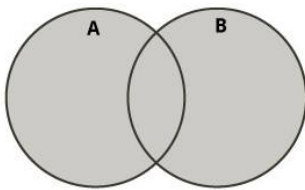
$x = \{3, 1, 2, 5, 5\}$

x

##### Резултат:

{1, 2, 3, 5}

#### 1.1. Обединение на множества ( дизюнкция ).



##### Пример 2:

$A = \{1, 2, 3\}$

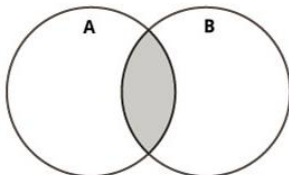
$B = \{2, 1, 4\}$

$A \cup B$

##### Резултат:

{1, 2, 3, 4}

#### 1.2. Сечение на множества ( конюнкция ).



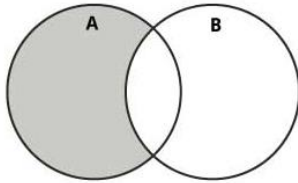
##### Пример 3:

$A \cap B$

**Резултат:**

$\{1, 2\}$

**1.3. Разлика на множества.**



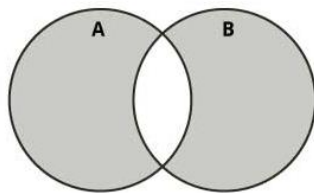
**Пример 4:**

$A - B$

**Резултат:**

$\{3\}$

**1.4. Симетрична разлика на множества.**



**Пример 5:**

$A \cap B$

**Резултат:**

$\{3, 4\}$

**1.5. Подмножество.**

**Пример 6:**

$C = \{1, 2\}$

$A \supset C$

**Резултат:**

True

Множеството C е подмножество на множеството A.

### **Самостоятелна задача 1.1:**

Разгледайте самостоятелно методите на множество, като използвате `help(set)`. Тествайте със собствени примери.

**2. Речници.** – представляват неподредено множество от двойка ключ(с уникална стойност) и стойност. Ключовете могат да бъдат число, низ, комплект. Стойностите могат да бъдат всеки тип. Можете да дефинирате, променяте, търсите и изтривате двойки ключ-стойност в речника.

**2.1. Създаване на речник, чрез използване на фигурални скоби {}.**

**Пример 7:**

```
lang={'USA':'English','Bulgaria':'Bulgarian'}
```

**2.2. Проверка дали дадена дума(key) я има в речника**

```
lang['USA']
```

**Резултат: ?**

**2.3. Проверка дали дадена дума(key) я има в речника чрез оператора in**

**Пример 8:**

```
'USA' in lang
```

**Резултат: ?**

**Пример 9:**

```
'English' in lang
```

**Резултат: ?**

**2.4. Извеждане на ключовете и стойностите на речника, поотделно и по двойки**

**Пример 10:**

```
lang.keys()
```

**Резултат: ?**

**Пример 11:**

```
lang.values()
```

**Резултат: ?**

**Пример 12 :**

```
lang.items()
```

**Резултат: ?**

**2.5. Добавяне на нова двойка ключ-стойност в речника**

**Пример 13 :**

```
lang['Russia']='Russian'  
lang
```

**Резултат: ?**

**2.6. Изтриване на двойка ключ-стойност в речника**

**Пример 14 :**

```
del lang['Russia']  
lang
```

**Резултат: ?**

**2.7. Сортиране**

**Пример 15 :**

```
sorted(lang)
```

**2.8. Създаване на речник, чрез използване на ключова дума dict и именувани параметри.**

**Пример 16 :**

```
dict(USA='English',Bulgaria='Bulgarian')
```

**Резултат: ?**

**2.9. Създаване на речник, чрез използване на ключова дума dict и списък от двойки.**

**Пример 17 :**

```
dict([('USA','English'),('Bulgaria','Bulgarian')])
```

**Резултат: ?**

```
{'USA': 'English', 'Bulgaria': 'Bulgarian'}
```

### **Самостоятелна задача 1.2:**

Разгледайте самостоятелно методите на речника, като използвате `help(dict)`. Тествайте със собствени примери.

**3. Преобразуване между списък, комплект и множество.** – чрез използване на ключовите думи: `list`, `tuple` и `set`.

**3.1. Проверка на типа на променливата**

**Пример 18:**

```
x=[1, 2, 3]
type(x)
```

**Резултат: ?**

**Пример 19:**

```
y=1
type(y)
```

**Резултат: ?**

**3.2. Преобразуване от set в list**

**Пример 20:**

```
list({1,2,3})
```

**Резултат: ?**

**3.3. Преобразуване от tuple в list**

**Пример 21:**

```
list((1,2,3))
```

**Резултат: ?**

**3.4. Преобразуване от list в tuple**

**Пример 22:**

```
tuple([1,2,3])
```

**Резултат: ?**

**3.5. Преобразуване от set в tuple**

**Пример 23:**

```
tuple({1,2,3})
```

**Резултат: ?**

### 3.6. Преобразуване от tuple в set

**Пример 24:**

```
set((1,2,3))
```

**Резултат: ?**

### 3.7. Преобразуване от list в set

**Пример 25:**

```
set([1,2,3])
```

**Резултат: ?**

## 4. Допълнителни сведения за функции

### 4.1. Стойности на аргументите по подразбиране

Пример за функция, която може да бъде извиквана с по-малко аргументи, отколкото е дефинирана:

**Пример 26:**

```
def function(answer, count=5, text='Желаете ли затворите приложението? Отговорете с да или не!'):
```

```
    while 1:
```

```
        yesno = input(answer)
```

```
        if yesno in ('Да', 'да'):return 1
```

```
        if yesno in ('Не', 'не'):return 0
```

```
        count = count - 1
```

```
        if count < 0: raise IOError('Довиждане!')
```

```
        print (text)
```

Тази функция може да бъде извиквана например така:

```
function ('Наистина ли желаете да затворите приложението?')
```

или така:

```
function ('Наистина ли желаете да затворите приложението?', 2)
```

Стойностите по подразбиране се изчисляват по време на дефинирането на функцията в обсега на самата дефиниция, така че например:

**Пример 27 :**

```
x = 111

def function2(parameter=x):

    print(parameter)

parameter = 200

function2( )
```

ще отпечата 111.

**Важно предупреждение:** Стойностите по подразбиране се изчисляват само веднъж. Това поражда разлика, когато стойността по подразбиране е променлив обект като списък или речник (dictionary). Например, следващата функция събира аргументите, които са и подадени в последователни извиквания:

**Пример 28 :**

```
def function(x, y = []):

    y.append(x)

    return y

print(function(1))

print(function(2))

print(function(3))
```

**Резултат:**

```
[1]
[1, 2]
[1, 2, 3]
```

Ако не желаете стойността по подразбиране да бъде споделяна между последователните извиквания, можете вместо това да напишете функцията така:

**Пример 29 :**

```
def function(x, y = None):

    if y is None:

        y=[ ]

    y.append(x)
```

return y

### Самостоятелна задача 2:

Извиквайте функцията function последователно с параметри 1, 2, 3.

#### 4.2. Аргументи с ключови думи

Функциите могат да бъдат извиквани и с използването на аргументи с ключови думи във формата *ключова\_дума* = '*стойност*'. Например, следващата функция:

##### Пример 30 :

```
def sendMessage(sender, recipient='World!', greeting='Hello', question='How have you been?', sendoff='Best Wishes'):
```

```
    print (greeting, recipient, question, sendoff, sender)
```

може да бъде извиквана по следните начини:

```
sendMessage("Skynet")
```

**Резултат:** ?

##### Пример 31 :

```
sendMessage(greeting = 'Dear Mr/Ms', sender = 'Random Internet Person')
```

**Резултат:** ?

##### Пример 32 :

```
sendMessage('Me', sendoff = 'Regards')
```

**Резултат:** ?

##### Пример 33 :

```
sendMessage('Your mom', 'Ron', 'What did you do with the car')
```

**Резултат:** ?

, но следващите извиквания биха били невалидни:

##### Примери 34 :

```
sendMessage() # липсва задължителен аргумент
```

```
sendMessage('Me', sender='You') # дублирана стойност за аргумент
```

```
sendMessage(city='Sofia') # непозната ключова дума
```

Списъкът с аргументите може да съдържа всякакви позиционни аргументи, следвани от всякакви аргументи с ключови думи, където ключовите думи са избрани от



имената на формалните параметри. Не е важно дали даден формален параметър има стойност по подразбиране или не. Не може обаче един аргумент да получи стойност повече от веднъж. Имената на формалните параметри, съответни на позиционните аргументи, не могат да бъдат използвани като ключови думи в едно и също извикване. Ето един пример, който пропада заради това ограничение:

**Пример 35 :**

```
def fun(a):  
...     print("Hello World")  
...  
fun(0, a=0)
```

Traceback (innermost last):

File "<stdin>", line 1, in ?

**TypeError:** keyword parameter redefined

Когато последният формален параметър има формата **\*\*име**, той получава речник, съдържащ всички аргументи с ключови думи, които не съответстват на формален параметър. Това може да бъде комбинирано с формален параметър с формата **\*име**, който получава комплект (tuple), съдържащ позиционните аргументи извън списъка на формалните параметри. (**\*име** трябва да се яви преди **\*\*име**.) Например, ако дефинираме функция, подобна на тази:

**Пример 36 :**

```
def showRanking (*arguments, **keywords):  
    print (arguments)  
    print (keywords)
```

Тя може да бъде извикана например така:

```
showRanking('Skiing', 'Bobsled', 'Triathlon', first='Bulgaria', second='Romania',  
third='Macedonia')
```

**Резултат: ?**

**Задача:** Тествайте последната функция с различни параметри.

## 5. Работа с файлове

### 5.1. Отваряне на файл

open() - връща файлов обект object и най-често се използва с два аргумента:

Формат на функцията:

‘open(файлово\_име, режим)’.

- Първи аргумент – име на файла

- **Режими:**

'r' – файлът се отваря само за четене

'w' – създава се нов файл и се отваря само за запис (съществуващ файл със същото име ще бъде изтрит)

'a' – файлът се отваря за запис, като добавя съдържанието в края на файла, ако той съществува

't' – текстов режим (по подразбиране)

'r+' – файлът се отваря за четене и запис

'a+' – файлът се отваря за четене и запис, като добавя съдържанието в края на файла, ако той съществува

Аргументът режим е незадължителен; ако 'r' е пропуснат, той се подразбира.

За повече информация: `help(open)`

## **5.2. Четене на файл**

Функция `read()` - без параметри прочита целия текстов файл (всички символи)

Създайте текстов файл с произволно съдържание с име: `testfile`

Метод `read(char_numbers)` - прочита `char_numbers` символи от файла

### **Пример 37 :**

```
file = open("testfile.txt", "r")  
print(file.read())
```

Метод `readline()` - прочита ред от файла

При първото извикване прочита първия ред, при второто прочита втория и т.н.

### **Пример 38 :**

```
file = open("testfile.txt", "r")  
print(file.readline())
```

Метод `readlines()` - прочита всички редове от файла, разделени в подходящ формат

### **Пример 39 :** за четене на файл ред по ред:

```
file = open("testfile.txt", "r")  
print(file.readlines())
```

Или:

### **Пример 40 :**

```
file = open("testfile.txt", "r")  
  
for line in file:  
    print(line)
```

### 5.3. Запис във файл

Метод `write(string)` - запис на символния низ `string` във файла

Пример за създаване на текстов файл и запис на текст в него.

#### **Пример 41 :**

```
file = open("testfile.txt", "w")  
file.write("This is our new text file\n")  
file.write("and this is another line.")  
file.write("\nWhy? Because we can.")  
file.close()
```

### 5.4. Затваряне на файл

Метод `close()`

### 5.5. Достъп до файл с оператор `with`

Операторът `with` предоставя по-ясен синтаксис за обработване на файл. Негово предимство е, че автоматично затваря всеки отворен файл след приключване на файловите операции.

Формат:

`with open("filename") as file:`

#### **Пример 42 :**

```
with open("testfile.txt") as f:  
    for line in f:  
        print(line)
```

#### **Резултат: ?**

Липсва извикване на метода `close()`, който се извиква автоматично в края на оператора `with`.

**Пример 43 :** за създаване на файл и запис в него:

```
with open("123.txt", "w") as f:  
    f.write("hello\n")
```

```
f.write("1234567\n")  
f.write("abcdefgh\n")
```

**Пример 44 :** за прочитане на съдържанието на файла в променлива:

```
with open("123.txt") as f:  
    data = f.readlines()  
    print(data)
```

**Резултат: ?**

## 5.6. Разделяне на думи, прочетени от текстов файл

Метод `split()` - разделя думи в символен низ с разделител интервал

**Пример 45 :**

```
with open("123.txt", "r") as f:  
    data = f.readlines()  
for line in data:  
    words = line.split()  
    print(words)
```

**Резултат: ?**

Методът `split()` може да се извика с параметър символ. В този случай думите се разделят с разделител този символ.

**Пример 46 :**

```
str=" I love banana, orange and kiwi smoothie."  
print(str.split(","))
```

**Резултат: ?**

За повече информация: `help(str.split)`

### Самостоятелна задача 3:

Даден е символен низ "I love chocolate very much". Да се запише всяка дума от низа на отделен ред в текстов файл.

## 6. Визуализация на данни

### 6.1. Въведение в `matplotlib`

`matplotlib` - мултиплатформена библиотека за визуализация на данни

`matplotlib.pyplot` - функции за изчертаване на графики и фигури

За повече информация:

```
import matplotlib.pyplot
```

```
help(matplotlib.pyplot)
```

или [https://matplotlib.org/api/\\_as\\_gen/matplotlib.pyplot.plot.html](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html)

#### 6.1.1. Функция `plot`

Функцията `plot` се използва за изчертаване на графики на функции.

Примери:

```
plot(x, y)      # plot x and y using default line style and color
```

```
plot(x, y, 'bo') # plot x and y using blue circle markers
```

```
plot(y)         # plot y using x as index array 0..N-1
```

```
plot(y, 'r+')    # ditto, but with red plusses
```

За повече информация:

```
import matplotlib.pyplot as plt
```

```
help(plt.plot)
```

**Пример 47 :**

```
import matplotlib.pyplot as plt
```

```
plt.plot([1,2,1,2,3,2,4])
```

```
plt.ylabel('some numbers')
```

```
plt.show()
```

**Резултат: ?**

Параметрите на функцията `plot()` са стойности за `y` в графиката. По подразбиране `python` създава същите данни за `x`, като започва от 0.

**Пример 48 :**

```
plt.plot([1, 2, 3, 4], [1, 4, 1, 16])
```

```
plt.show()
```

**Резултат: ?**

За всяка двойка параметри е възможен трети параметър. Това е символен низ, който задава цвят и тип на графиката. Например:

‘b-’ - непрекъснатата синя линия (по подразбиране)

‘r-’ - непрекъснатата червена линия

‘o-’ - данни с точки

#### 6.1.2. Функция `axis()`

axis() - функция, задаваща [xmin, xmax, ymin, ymax]

**Пример 49 :**

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,1,16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```

**Резултат: ?**

Практически matplotlib се използва често с numpy масиви. Numpy е библиотека за представяне и операции с множества от данни.

*За повече информация:*

```
import numpy
help(numpy)
```

Или <http://www.numpy.org/>

**6.1.3.      Функция arange (...)**

*Синтаксис:*

```
arange([start,] stop[, step,], dtype=None)
```

*За повече информация:*

```
import numpy as np
help(np.arange)
```

**Пример 50 :**

```
import numpy as np
import matplotlib.pyplot as plt
t = np.arange(0., 5., 0.2)
print(t)
```

**Резултат: ?**

**Пример 51 :**

```
# равномерно разпределя на интервали от 200 ms
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
# червени тирета, сини квадратчета, зелени триъгълници
plt.show()
```

**Резултат: ?**

#### **Самостоятелна задача 4:**

Визуализирайте графиката на функцията  $y=x^2$ , за стойности на  $x=[-2;+2]$  със стъпка 0.01.

### **6.2. Работа с множество графики**

#### **6.2.1. Функция figure – функция, създаваща фигура**

*Синтаксис:*

```
figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None,  
frameon=True, FigureClass=<class 'matplotlib.figure.Figure'>, **kwargs)
```

*За повече информация:*

```
import numpy as np
```

```
help(plt.figure)
```

#### **6.2.2. Функция subplot**

*Синтаксис:*

```
subplot(nrows, ncols, plot_number)
```

Субдиаграмите позволяват няколко графики да бъдат поставени на една и съща фигура. Аргументите са: брой редове, брой колони и в коя клетка да се постави графиката. Извикване на субдиаграмата трябва да бъде последвано от диаграмна функция, която може да е plot.

Ако числата указващи nrows, ncols и plot\_number са едноцифрени може да се пропуска разделителя запетая.

#### **Пример 52 :**

```
import matplotlib.pyplot as plt
```

```
plt.figure(1)      # първа фигура
```

```
plt.subplot(211)   # първа графика в първата фигура, като фигурата е на два реда и 1 колона
```

```
plt.plot([1, 4, 2])
```

```
plt.subplot(212)   # втора графика в първата фигура
```

```
plt.plot([3, 1, 7])
```

```
plt.figure(2)      # втора фигура
```

```
plt.plot([4, 2, 8]) # създава subplot(111) по подразбиране
```

```
plt.figure(1)      # задава фигура 1 текуща; subplot(212) също е текуща
```

```
plt.subplot(211)          # задава subplot(211) във фигура 1 текуща
plt.title('Easy as 1, 2, 3') # заглавие на subplot 211
plt.show()
```

**Резултат: ?**

### 6.3. Добавяне на текст в графика

**Пример 53 :**

```
import numpy as np
import matplotlib.pyplot as plt
x = np.random.randn(10000) # връща извадка от нормално разпределение
# хистограма
plt.hist(x)
plt.title("Gaussian Histogram")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()
```

**Резултат: ?**

### 6.4. Добавяне на анотации в графика

`annotate(s, xy, *args, **kwargs)`

анотация на точка **xy** с координати (x,y) с текст **s**

**xytext(x,y)** – позиция (x,y) за поставяне на текста. По подразбиране е позицията на **xy**

**arrowprops** – стрелка между позициите **xy** и **xytext**

За повече информация: `help(plt.annotate)`

**Пример 54 :**

```
import numpy as np
import matplotlib.pyplot as plt
ax = plt.subplot(111)
t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line = plt.plot(t, s)
```



```
plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),arrowprops=dict(facecolor='black'))  
plt.ylim(-2,2) # задаване на ограничения по y - оста  
plt.show()
```

**Резултат: ?**

## 7. Анализ на данни с pandas

Pandas (<http://pandas.pydata.org/>) е библиотека за обработка и анализ на данни, която се използва от множество програмни среди за машинно обучение (machine learning framework).

### 7.1. Основни понятия в pandas

Включване на библиотеката и извеждане на нейната версия:

```
from __future__ import print_function  
import pandas as pd  
pd.__version__
```

Основните структури данни в pandas са имплементирани в два класа:

- DataFrame – представлява таблица с редове и именувани колони
- Series – представлява една колона; един DataFrame съдържа една или повече Series с име за всяка серия

Един начин за създаване на серия е чрез обект Series:

```
pd.Series(['San Francisco', 'San Jose', 'Sacramento'])
```

DataFrame обекти се създават чрез подаване на параметри имена на колони и съответните серии от данни. Ако подадена серия няма необходимата дължина, липсващите стойности се запълват със специални стойности NA/NaN. Например:

**Пример 55 :**

```
city_names = pd.Series(['San Francisco', 'San Jose', 'Sacramento'])  
population = pd.Series([852469, 1015785, 485199])  
pd.DataFrame({ 'City name': city_names, 'Population': population })
```

**Резултат: ?**

Най-често DataFrame обект се зарежда от файл с данни. В следващия пример DataFrame се зарежда с данни за домакинства. Методът describe извежда статистически данни:

**Пример 56 :**

```
california_housing_dataframe = pd.read_csv("https://download.mlcc.google.com/mledu-  
datasets/california_housing_train.csv", sep=",")  
  
california_housing_dataframe.describe()
```

**Резултат: ?**

Често се използва функцията `DataFrame.head`, която извежда първите редове:

**Пример 57 :**

```
california_housing_dataframe.head()
```

**Резултат: ?**

Друга полезна възможност в `pandas` е графично изобразяване на данни. Например `DataFrame.hist` изобразява разпределението на данните в колона:

**Пример 58 :**

```
california_housing_dataframe.hist('housing_median_age')
```

**Резултат: ?**

## 7.2. Достъп до данни в `DataFrame`

Достъпът до `DataFrame` се извършва чрез речници (dictionaries) и списъци (lists):

**Пример 59 :**

```
cities = pd.DataFrame({ 'City name': city_names, 'Population': population })  
  
print(type(cities['City name']))  
  
cities['City name']
```

**Резултат: ?**

**Пример 60 :**

```
print(type(cities['City name'][1]))  
  
cities['City name'][1]
```

**Резултат: ?**

**Пример 61 :**

```
print(type(cities[0:2]))  
  
cities[0:2]
```

**Резултат: ?**

Освен изброените средства pandas предлага множество допълнителни възможности за индексване и достъп до данни в DataFrame ([http://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html](http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html)).

### 7.3. Обработване на данни

С данните в серия могат да бъдат изпълнявани основните аритметични операции в Python.

#### Пример 62 :

```
population / 1000  
pd.DataFrame({ 'City name': city_names, 'Population': population/1000 })
```

#### Резултат : ?

Серия може да се използва като аргумент в повечето функции от NumPy.

#### Пример 63 :

```
import numpy as np  
np.log(population)
```

#### Резултат : ?

За по-сложни обработки с данните от една колона се използва метод Series.apply. Той приема като аргумент lambda функция (<https://docs.python.org/2/tutorial/controlflow.html#lambda-expressions>), която се изпълнява с всяка стойност. Следващият пример създава нова серия, която показва дали населението на града е над един милион жители:

#### Пример 64 :

```
population.apply(lambda val: val > 1000000)
```

#### Резултат : ?

Модифицирането на DataFrame също става лесно. Пример за добавяне на две серии в съществуващ DataFrame:

#### Пример 65 :

```
cities['Area square miles'] = pd.Series([46.87, 176.53, 97.92])  
cities['Population density'] = cities['Population'] / cities['Area square miles']  
cities
```

#### Резултат : ?

#### 7.4. Индексиране

Обектите Series и DataFrame имат член-променлива index, която представлява уникален идентификатор на всеки елемент от серията и на всеки DataFrame ред. При създаване на обекта индексът съответства на реда елементите. Стойностите на индексите не се променят при размяна на елементи в серия или DataFrame.

Методът DataFrame.reindex се използва за ръчно пренареждане на редове.

##### **Пример 66 :**

```
cities.reindex([2, 0, 1])
```

##### **Резултат : ?**

Реиндексирането се използва за случайно пренареждане (разбъркване) на редовете в DataFrame. Следващият пример използва функцията random.permutation за пермутация на индексите на редовете в DataFrame. Изпълнете следващия ред последователно няколко пъти:

##### **Пример 67 :**

```
cities.reindex(np.random.permutation(cities.index))
```

##### **Резултат : ?**

Повече информация за индексирането - [http://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html](http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html).