# Computational Intelligence

Laboratories

# Python

Language mostly used for data science.

- simple and powerful syntax
- flexible - many modules and system for their instalation
- used worldwide
- graphics
- online interpreter

## Version

Versions 2.x and 3.x are not fully compatible. Last is 3.9.6. Use version 3.x or above for this course.

https://www.python.org/

On Linux systems you can install packages according to the distribution you are using. For example on Ubuntu, you can use the `apt-get install python3` command.

On Windows 10, we recommend to install Anaconda containing Python and sophisticated packaging system for installing additional libraries. The Ananconda supports GPUs, if you have a NVIDIA graphic card on your computer.

Another alternative is to use WSL (Windows Subsystem on Linux) to install Ubuntu on your computer. Python3 is working well on WSL Ubuntu, but GPUs are not available yet.

## IDE

Many products are available. For serious coding we recomend PyCharm.

https://www.jetbrains.com/pycharm/

- separate environments for each project
- posibility to switch interpreter

An alternative is to use Visual Studio Code (From Microsoft). It supports Jupyter notebooks on contrary to Pycham community version.

## Jupyter

Using Jupyter notebooks is an easy way how to interactively write programs in Python3. It can be installed under Linux distributions. On Ubuntu, use `apt-get install jupyter` command.

Jupyter notebooks are also supported by Anaconda under Windows 10.

## Google Colab

Open environment for data processing in Python based on Jupyter notebooks.

Google Colab providing CPUs and also GPUs for data processing. You need Google account to access Colab. If you do not have such account, please, create it.

Google Colab is accesible by https://colab.research.google.com.

Specific settings - https://www.javatpoint.com/python-programming-with-google-colab

## The Python Programming Language

Introduction - https://docs.python.org/3/tutorial/index.html

Basic structures the same as in other languages.

# Example 1

Print the prime numbers from 1 to 100.

```
# Example 1
for i in range(101):
  prime=True
  j=2
  while prime and j<i:
    if (i % (j))==0:
      prime=False
    j+=1
  if prime:
    print(i)
```

```
0
1
2
3
5
7
11
13
17
19
23
29
31
37
41
43
```

```
47
53
59
61
67
71
73
79
83
89
97
```

# Task 1

Modify previous example and define function for testing if the number is a prime number.

```
#Write your code here
import math

def isprime(num):
  for num in range(num):
   if num >= 0:
       for i in range(2, num):
           if (num % i) == 0:
               break
       else:
           print(num)

isprime(101)
```

```
0
1
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
```

```
89
97
```

# Task 2

Modify the previous task and define class finding the prime numbers in selected range.

```
class IsPrime:
    def __init__(self, start, stop):
        self.start = start
        self.stop = stop
    def isPrimeNumber(self):
        # check for factors
        for num in range(self.start, self.stop):
            if num >= 0:
                for i in range(2,num):
                    if(num % i) == 0:
                        break
                else:
                    print(num)
isPrimeObj = IsPrime(0,101)
isPrimeObj.isPrimeNumber()
```

```
0
1
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
```

# Task 3

Load Iris dataset from https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

```
#Write your code here
import pandas as pd
df = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
df.head()
```

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

# Task 4

Print the content of the **iris_dataset** variable.

```
df.columns = ['slength','swidth','plength','pwidth','species']
df
```

| | slength | swidth | plength | pwidth | species |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| **...** | ... | ... | ... | ... | ... |
| **145** | 6.7 | 3.0 | 5.2 | 2.3 | Iris-virginica |
| **146** | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| **147** | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| **148** | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| **149** | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

150 rows × 5 columns

# Task 5

Rencode labels to three numerical colums according to the table:

| Label | Output[0] | Output[1] | Output[2] |
|---|---|---|---|
| Iris-versicolor | 1 | 0 | 0 |
| Iris-virginica | 0 | 1 | 0 |
| Iris-setosa | 0 | 0 | 1 |

so that the first row of the Iris dataset will look like `5.1 3.5 1.4 0.2 1 0 0`

```
import numpy as np
col0 = df['slength']
col1 = df['swidth']
col2 = df['plength']
col3 = df['pwidth']
col4 = pd.get_dummies(df['species'])
#print(col4)
irisDfEncoded = pd.concat([col0,col1,col2,col3,col4],axis=1).values
print(irisDfEncoded)
```

```
[[5.1 3.5 1.4 ... 1.  0.  0. ]
 [4.9 3.  1.4 ... 1.  0.  0. ]
 [4.7 3.2 1.3 ... 1.  0.  0. ]
 ...
 [6.5 3.  5.2 ... 0.  0.  1. ]
 [6.2 3.4 5.4 ... 0.  0.  1. ]
 [5.9 3.  5.1 ... 0.  0.  1. ]]
```

# Task 6

Split the `iris_dataset_enc` into two numpy arrays. The `iris_features` will contain four numerical columns containing features and the `iris_desired` will contain three numerical outputs containing desired outputs.

```
#Write your code here
iris_features = pd.concat([col0,col1,col2,col3],axis=1).values
iris_desired = col4
print(iris_features)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
```

```
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.1 1.5 0.1]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5.  3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]
 [5.7 2.8 4.5 1.3]
 [6.3 3.3 4.7 1.6]
```

```
print(iris_desired)
```

```
     Iris-setosa  Iris-versicolor  Iris-virginica
0              1                0               0
1              1                0               0
2              1                0               0
3              1                0               0
4              1                0               0
..           ...              ...             ...
145            0                0               1
146            0                0               1
147            0                0               1
148            0                0               1
149            0                0               1
```

```
[150 rows x 3 columns]
```

## Task 7

Normalize features in the `iris_dataset` into the range of $< 0, 1 >$ using min-max norm according to the following formula:

$$y = \frac{y_{max} - y_{min}}{x_{max} - x_{min}}(x - x_{min}) + y_{min}$$

where $x$ is raw value and $y$ is the normalised value. The $y_{min}$ and $y_{max}$ determine the output range. The $x_{max}$ and $x_{min}$ denote the input range.

For this task $y_{max} = 1, y_{min} = 0$. The $x_{max}$ and $x_{min}$ must be obtained from the dataset, for each column separately.

The result prints with limited precision to two decimal digits (0.00).

```
#Write your code here
x_min = np.min(iris_features,0)
x_max = np.max(iris_features,0)
y_min = np.zeros(4)
y_max = np.ones(4)
print(x_min)
print(x_max)
print(y_min)
print(y_max)
```

```
[4.3 2.  1.  0.1]
[7.9 4.4 6.9 2.5]
[0. 0. 0. 0.]
[1. 1. 1. 1.]
```

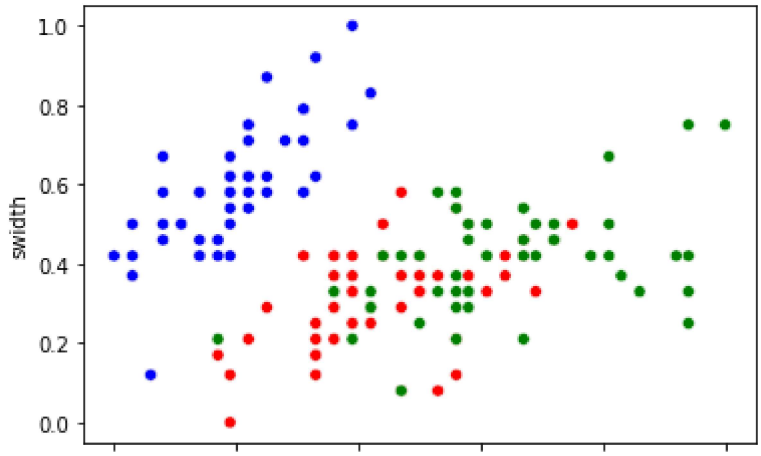
```
#Write your code here
y=((y_max-y_min)*(iris_features-x_min)+y_min)/(x_max-x_min)
y = np.around(y, decimals=2)
print(y)
```

```
[0.42 0.25 0.51 0.46]
[0.19 0.12 0.39 0.38]
[0.36 0.29 0.54 0.5 ]
[0.39 0.42 0.54 0.46]
[0.39 0.37 0.54 0.5 ]
[0.53 0.37 0.56 0.5 ]
[0.22 0.21 0.34 0.42]
[0.39 0.33 0.53 0.5 ]
[0.56 0.54 0.85 1.  ]
[0.42 0.29 0.69 0.75]
[0.78 0.42 0.83 0.83]
[0.56 0.37 0.78 0.71]
[0.61 0.42 0.81 0.88]
[0.92 0.42 0.95 0.83]
[0.17 0.21 0.59 0.67]
```

```
[0.17 0.21 0.59 0.67]
[0.83 0.37 0.9  0.71]
[0.67 0.21 0.81 0.71]
[0.81 0.67 0.86 1.  ]
[0.61 0.5  0.69 0.79]
[0.58 0.29 0.73 0.75]
[0.69 0.42 0.76 0.83]
[0.39 0.21 0.68 0.79]
[0.42 0.33 0.69 0.96]
[0.58 0.5  0.73 0.92]
[0.61 0.42 0.76 0.71]
[0.94 0.75 0.97 0.88]
[0.94 0.25 1.   0.92]
[0.47 0.08 0.68 0.58]
[0.72 0.5  0.8  0.92]
[0.36 0.33 0.66 0.79]
[0.94 0.33 0.97 0.79]
[0.56 0.29 0.66 0.71]
[0.67 0.54 0.8  0.83]
[0.81 0.5  0.85 0.71]
[0.53 0.33 0.64 0.71]
[0.5  0.42 0.66 0.71]
[0.58 0.33 0.78 0.83]
[0.81 0.42 0.81 0.62]
[0.86 0.33 0.86 0.75]
[1.   0.75 0.92 0.79]
[0.58 0.33 0.78 0.88]
[0.56 0.33 0.69 0.58]
[0.5  0.25 0.78 0.54]
[0.94 0.42 0.86 0.92]
[0.56 0.58 0.78 0.96]
[0.58 0.46 0.76 0.71]
[0.47 0.42 0.64 0.71]
[0.72 0.46 0.75 0.83]
[0.67 0.46 0.78 0.96]
[0.72 0.46 0.69 0.92]
[0.42 0.29 0.69 0.75]
[0.69 0.5  0.83 0.92]
[0.67 0.54 0.8  1.  ]
[0.67 0.42 0.71 0.92]
[0.56 0.21 0.68 0.75]
[0.61 0.42 0.71 0.79]
[0.53 0.58 0.75 0.92]
[0.44 0.42 0.69 0.71]]
```

```
#Write your code here
import seaborn as sns
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

df_scatter = pd.DataFrame(y, columns = ['slength','swidth','plength','pwidth'])
df_scatter['species'] = df['species']

sns.scatterplot(df_scatter['slength'],df_scatter['swidth'],hue=df_scatter['species'],
                palette=['blue','red','green'],legend=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f77d9dfee10>
```



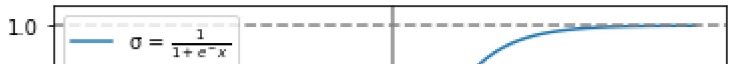

# Task 8

The sigmoid function is frequently used activation of a neuron. It is defined by

$$y = \sigma(x) = \frac{1}{1 + e^{-x}}$$

Draw a graf of this function for $x \in < -8, 8 >$.

```
#Write your code here
import math
import numpy as np
import matplotlib.pyplot as plt

sigmoid = lambda x: 1 / (1 + np.exp(-x))
x = np.arange(-8., 8., 0.2)
sig = sigmoid(x)
plt.plot(x,sig)
plt.axhline(0, color='grey', ls='--')
plt.axhline(1, color='grey', ls='--')
plt.axhline(0.5, color='grey', ls='--')
plt.axvline(0, color='grey')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(['\u03C3 = ' +r'$\frac{1}{1+e^-x}$'])
plt.show()
```




# Task 9

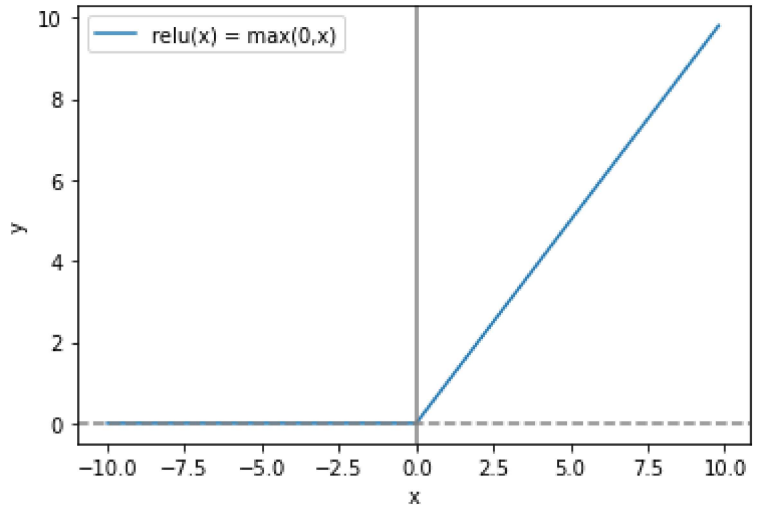
The RELU function is frequently used activation function of neurons in deep neural networks. It is defined by

$$y = \text{relu}(x) = \max(0, x)$$

Draw a graf of this function for $x \in < -10, 10 >$.

−8 −6 −4 −2 0 2 4 6 8

```
#Write your code here
relu = lambda x: np.maximum(0,x)
x = np.arange(-10., 10., 0.2)
y = relu(x)

plt.plot(x,y)
plt.axhline(0, color='grey', ls='--')
plt.axvline(0, color='grey')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(['relu(x) = max(0,x)'])
plt.show()
```



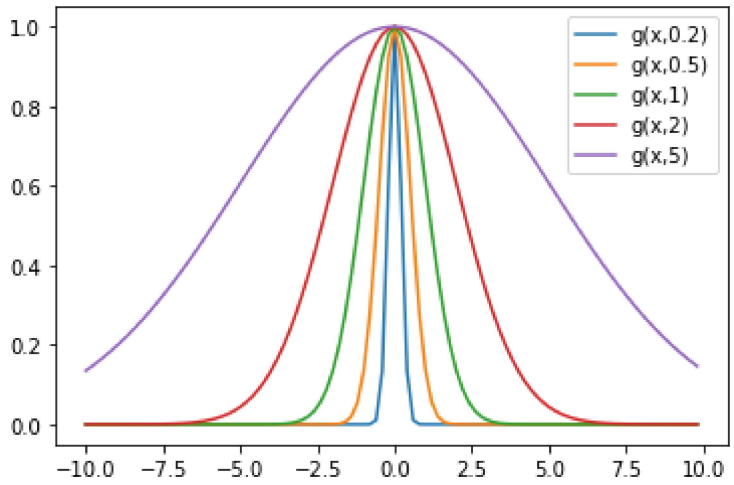

# Task 11

The Gaussian function is frequently used as an activation function of RBF neurons. It is defined by the equation

$$y = g(x) = e^{\frac{-x^2}{2\sigma^2}}$$

Draw a graf of this function for $x \in < -10, 10 >$ and $\sigma = \{0.2, 0.5, 1, 2, 5\}$.

```
#Write your code here
import numpy as np
def gaussian(x,sig):
  return np.exp(-np.power(x, 2.) / (2 * np.power(sig, 2.)))

x = np.arange(-10.,10.,0.2)
sig = np.array([0.2,0.5,1,2,5])
y = gaussian(x,0.2)
y2 = gaussian(x,0.5)
y3 = gaussian(x,1)
y4 = gaussian(x,2)
y5 = gaussian(x,5)
fig, ax = plt.subplots()

ax.plot(x, y,label = 'g(x,0.2)')
ax.plot(x, y2, label = 'g(x,0.5)')
ax.plot(x,y3, label = 'g(x,1)')
ax.plot(x,y4, label ='g(x,2)')
ax.plot(x,y5,label ='g(x,5)')
#plt.plot(x,y)
#ax.legend(custom_lines, ['Cold', 'Medium', 'Hot'])
ax.legend(loc='upper right')
plt.show()
```




# Task 12

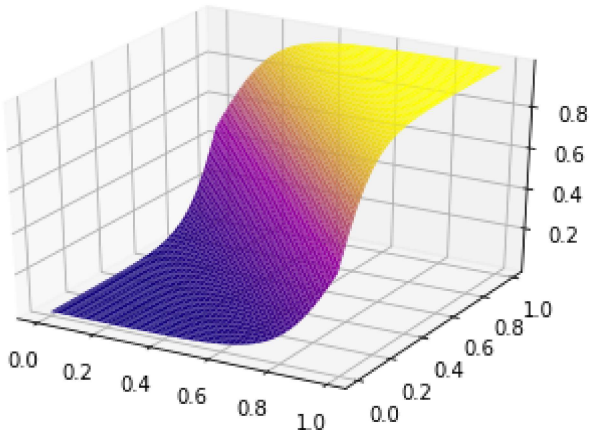
Draw response of the two input perceptron neuron defined by

$$y = \sigma(\sum_{i=0}^{1} w_i x_i + b)$$

where $w_0 = 10, w_1 = 10, b = -10$.

```
#Write your code here
import matplotlib.pyplot as plt
import numpy as np
```

```
import matplotlib.colors
from mpl_toolkits.mplot3d import Axes3D

sigmoid = lambda x: 1 / (1 + np.exp(-x))


def perceptron(x,w,b):
  return np.dot(x, w) + b

brycmap = matplotlib.colors.LinearSegmentedColormap.from_list("", ["#14037e","#94019e","ye
w=np.array([10,10])
b = -10
x = np.arange(0,1,0.01)
y = np.arange(0,1,0.01)
x0,x1 = np.meshgrid(x,y)
perceptronGrid=np.zeros(x0.shape)
for i in range(x.size):
    for j in range(y.size):
      perceptronGrid[i,j]= sigmoid(perceptron(np.asarray([x[j],y[i]]),w,b))

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(x0,x1,perceptronGrid,cmap = brycmap)
plt.show()
```




# Task 13

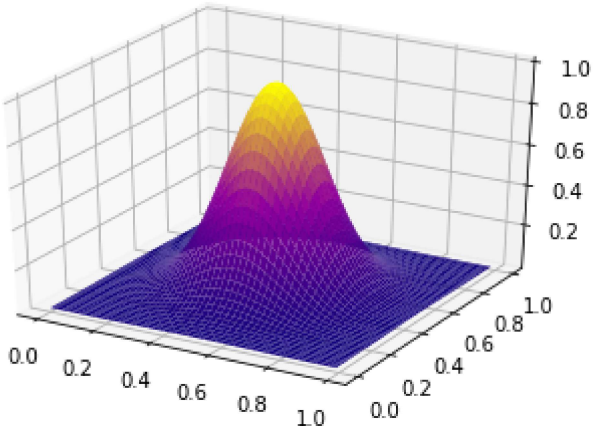
Draw response of the two input RBF neuron defined by

$$y = g(\sum_{i=0}^{1}(x_i - c_i)^2)$$

where $c_0 = 0.5, c_1 = 0.5, \sigma = 0.15$.

```
#Write your code here
import matplotlib.pyplot as plt
import numpy as np
```

```
import matplotlib.colors
from mpl_toolkits.mplot3d import Axes3D
import math

def gaussian(x,sig):
  return np.exp(-np.power(x, 2.) / (2 * np.power(sig, 2.)))


def rbf(x1,x2):
  sum = 0
  for i in range(len(x1)):
    sum += (x1[i]-x2[i]) ** 2
  return np.sqrt(sum)

brycmap = matplotlib.colors.LinearSegmentedColormap.from_list("", ["#14037e","#94019e","ye

sig = 0.15
c = np.array([0.5,0.5])
x = np.arange(0,1,0.01)
y = np.arange(0,1,0.01)
x0,x1 = np.meshgrid(x,y)
rbfGrid=np.zeros(x0.shape)
for i in range(x.size):
    for j in range(y.size):
      rbfGrid[i,j]= gaussian(rbf(np.asarray([x[j],y[i]]),c),sig)

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(x0,x1,rbfGrid,cmap = brycmap)
plt.show()
```




# Task 14

Load MNIST dataset using tensorflow from [http://yann.lecun.com/exdb/mnist/](http://yann.lecun.com/exdb/mnist/) including additional information.

```
from keras.datasets import mnist

(train_X, train_y), (test_X, test_y) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mn
11490434/11490434 [==============================] - 0s 0us/step
```

## Subtask 1

print the size of the input image at sample 0,

```
# train_X[0].shape
train_X[0].shape
```

```
(28, 28)
```

## Subtask 2

print the input image at sample 0 as a picture composed from dot and hash characters in text mode.

```
five = np.array(train_X[0])

#five[0]

for i in range(five[0].size):
  for j in range(five[i].size):
    if five[i][j] == 0:
      print('.',end='')
    else:
      print('#',end='')
  print()
```

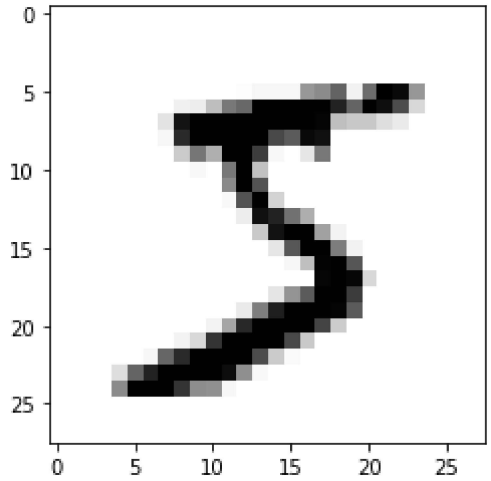
```
............................
............................
............................
............................
............................
...........############....
........################....
.......################.....
.......###########..........
........#######.##..........
.........#####..............
...........####.............
...........####.............
............######..........
.............######.........
..............######........
...............#####........
```

```
.................###.......
..............#######.......
............########........
..........#########.........
........##########..........
......##########............
....##########..............
....########................
..........................
..........................
..........................
```

## Subtask 3

print the input image at sample 0 as a picture using matplotlib library.

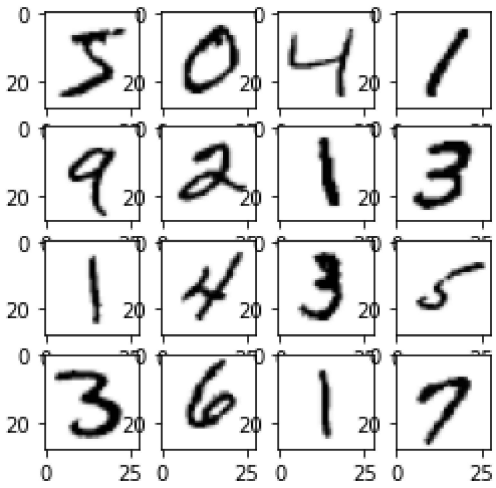
```
from matplotlib import pyplot

plt.imshow(train_X[0],cmap='Greys')
plt.show()
```




## Subtask 4

Display input images of the first 16 samples using the matplotlib library.

```
fig = plt.figure(figsize=(4, 4))

noOfrows = 4
noOfcolumns = 4

for i in range(16):
  fig.add_subplot(noOfrows, noOfcolumns, i+1)
  plt.imshow(train_X[i],cmap='Greys')
```





✓ 0s completed at 7:43 PM