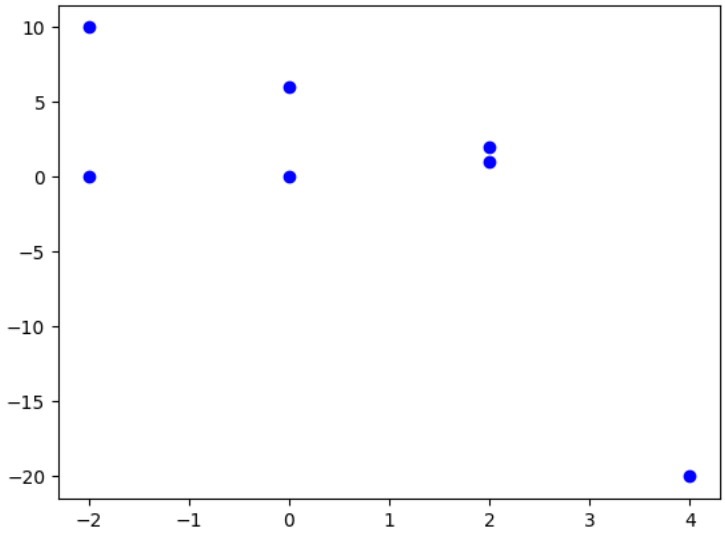
Import biblioteki **TensorFlow** (https://www.tensorflow.org/) z której będziemy korzystali w **uczeniu maszynowym**:

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
```

```
x_point = []
y_point = []
```

```
x_point = [2, 2, 0, -2, -2, 0, 4]
y_point= [1, 2, 6, 10, 0, 0, -20]
d = [1, 1, 1, -1, -1, -1, -1]
```

```
plt.scatter(x_point,y_point,c='b')
plt.show()
```




```
real_x = np.array(x_point)
real_y = np.array(y_point)
```

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

Definiujemy model:

```
model = Sequential()
```

Dodajemy **jedną warstwę** (Dense) z **jednym neuronem** (units=1) z **biasem** (use_bias=True) i **liniową funkcją aktywacji** (activation="linear"):

```
model.add(Dense(units = 1, use_bias=True, input_dim=1, activation = "linear"))
```

Definiujemy **optymalizator** i **błąd** (średni błąd kwadratowy - MSE). **Współczynnik uczenia = 0.1**

```
opt = tf.keras.optimizers.SGD(learning_rate=0.001)
```

```
model.compile(loss='MSE',optimizer=opt)
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 dense_2 (Dense)             (None, 1)                 2

=================================================================
Total params: 2 (8.00 Byte)
Trainable params: 2 (8.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Proces **uczenia**:

```
epochs = 1000
h = model.fit(real_x,real_y, verbose=1, epochs=epochs, batch_size=100)
```

```
Epoch 836/1000
1/1 [==============================] - 0s 8ms/step - loss: 35.5284
Epoch 837/1000
1/1 [==============================] - 0s 7ms/step - loss: 35.5276
Epoch 838/1000
1/1 [==============================] - 0s 7ms/step - loss: 35.5267
Epoch 839/1000
1/1 [==============================] - 0s 8ms/step - loss: 35.5259
Epoch 840/1000
1/1 [==============================] - 0s 7ms/step - loss: 35.5251
Epoch 841/1000
1/1 [==============================] - 0s 9ms/step - loss: 35.5243
Epoch 842/1000
1/1 [==============================] - 0s 9ms/step - loss: 35.5234
Epoch 843/1000
1/1 [==============================] - 0s 11ms/step - loss: 35.5226
Epoch 844/1000
1/1 [==============================] - 0s 9ms/step - loss: 35.5218
Epoch 845/1000
1/1 [==============================] - 0s 8ms/step - loss: 35.5210
Epoch 846/1000
1/1 [==============================] - 0s 9ms/step - loss: 35.5202
Epoch 847/1000
1/1 [==============================] - 0s 8ms/step - loss: 35.5194
Epoch 848/1000
1/1 [==============================] - 0s 9ms/step - loss: 35.5186
Epoch 849/1000
1/1 [==============================] - 0s 9ms/step - loss: 35.5178
Epoch 850/1000
1/1 [==============================] - 0s 8ms/step - loss: 35.5170
Epoch 851/1000
1/1 [==============================] - 0s 8ms/step - loss: 35.5162
Epoch 852/1000
1/1 [==============================] - 0s 8ms/step - loss: 35.5154
Epoch 853/1000
1/1 [==============================] - 0s 9ms/step - loss: 35.5146
Epoch 854/1000
1/1 [==============================] - 0s 7ms/step - loss: 35.5138
Epoch 855/1000
1/1 [==============================] - 0s 9ms/step - loss: 35.5131
Epoch 856/1000
1/1 [==============================] - 0s 10ms/step - loss: 35.5123
Epoch 857/1000
1/1 [==============================] - 0s 9ms/step - loss: 35.5115
Epoch 858/1000
1/1 [==============================] - 0s 8ms/step - loss: 35.5107
Epoch 859/1000
1/1 [==============================] - 0s 9ms/step - loss: 35.5100
Epoch 860/1000
1/1 [==============================] - 0s 9ms/step - loss: 35.5092
Epoch 861/1000
1/1 [==============================] - 0s 8ms/step - loss: 35.5084
Epoch 862/1000
1/1 [==============================] - 0s 8ms/step - loss: 35.5077
Epoch 863/1000
1/1 [==============================] - 0s 10ms/step - loss: 35.5069
Epoch 864/1000
1/1 [==============================] - 0s 9ms/step - loss: 35.5062
Epoch 865/1000
```

```
Loss = h.history['loss']
Loss
```

```
35.44778823852539,
35.44724655151367,
35.44670867919922,
35.446170806884766,
35.445640563964844,
35.445106506347656,
35.444576263427734,
35.44404983520508,
35.443519592285156,
35.442996978759766,
35.44247055053711,
35.441951751708984,
35.44143295288086,
35.44091796875,
35.44040298461914,
35.43988800048828,
35.43937683105469,
35.43886947631836,
35.43836212158203,
35.4378547668457,
35.43735122680664,
35.436851501464844,
35.43634796142578,
35.43585205078125,
35.43535614013672,
35.43486404418945,
35.434364318847656,
35.433876037597656,
35.433387756347656,
35.432899475097656,
35.43241500854492,
35.43193435668945,
35.43144989013672,
35.430965423583984,
35.43048858642578,
35.43001174926758,
35.42953872680664,
35.4290657043457,
35.428592681884766,
35.428123474121094,
35.427650451660156,
35.42718505859375,
35.42672348022461,
35.42626190185547,
35.4257926940918,
35.42533874511719]
```

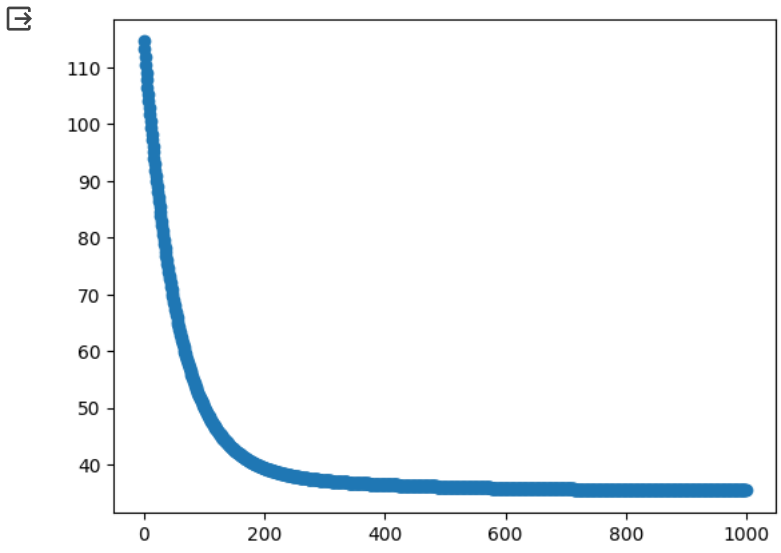
Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

print(weights[0][0][0])
print(weights[1][0])    #bias
```

```
-3.0866568
1.2872447
```

```
plt.scatter(np.arange(epochs),Loss)
plt.show()
```



Sprawdzenie **modelu**:

```
#model.predict([0.6])
```