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

```
import tensorflow as tf
```

W **TensorFlow** istnieją elementy, które są niezbędne do stworzenia dowolnego kodu TensorFlow. Są to tzw. **stałe** i **zmienne**. Służą one do przypisywania i przechowywania wartości.

**STAŁE**

Metoda **tf.constant()** tworzy **tensor** jako stały obiekt w **TensorFlow**. Przykłady:

```
import tensorflow as tf
a = tf.constant(4.0)
b = tf.constant([1, 2],dtype=tf.int32) # z określeniem typu
c = tf.constant([[1.0, 1.0, 4.0], [0.0, 1.0, -2.0]],dtype=tf.float32) # z określeniem typu
print(a)
print(b)
print(c)
```

```
tf.Tensor(4.0, shape=(), dtype=float32)
tf.Tensor([1 2], shape=(2,), dtype=int32)
tf.Tensor(
[[ 1.  1.  4.]
 [ 0.  1. -2.]], shape=(2, 3), dtype=float32)
```

```
print(a.numpy())
print(b.numpy())
print(c.numpy())
```

```
4.0
[1 2]
[[ 1.  1.  4.]
 [ 0.  1. -2.]]
```

Atrybuty **shape** (kształt) i **dtype** (datatype):

```
print(c.shape)
print(c.dtype)
```

```
(2, 3)
<dtype: 'float32'>
```

W **TensorFlow** mamy do dyspozycji następujące **typy**:

| Data type | Python type | Description |
|---|---|---|
| DT_FLOAT | tf.float32 | 32-bit floating point. |
| DT_DOUBLE | tf.float64 | 64-bit floating point. |
| DT_INT8 | tf.int8 | 8-bit signed integer. |
| DT_INT16 | tf.int16 | 16-bit signed integer. |
| DT_INT32 | tf.int32 | 32-bit signed integer. |
| DT_INT64 | tf.int64 | 64-bit signed integer. |
| DT_UINT8 | tf.uint8 | 8-bit unsigned integer. |
| DT_UINT16 | tf.uint16 | 16-bit unsigned integer. |
| DT_STRING | tf.string | Variable-length byte array. Each element of a Tensor is a byte array. |
| DT_BOOL | tf.bool | Boolean. |
| DT_COMPLEX64 | tf.complex64 | Complex number made of two 32-bit floating points: real and imaginary parts. |
| DT_COMPLEX128 | tf.complex128 | Complex number made of two 64-bit floating points: real and imaginary parts. |
| DT_QINT8 | tf.qint8 | 8-bit signed integer used in quantized ops. |
| DT_QINT32 | tf.qint32 | 32-bit signed integer used in quantized ops. |
| DT_QUINT8 | tf.quint8 | 8-bit unsigned integer used in quantized ops. |

Wartość atrybutu **dtype** można zmienić za pomocą metody **cast()**. Przykład:

```
x = tf.constant([1,2,3], dtype=tf.float32)
print(x.dtype)
x = tf.cast(x,tf.int64)
print(x.dtype)
```

```
<dtype: 'float32'>
<dtype: 'int64'>
```

Do fragmentów tensora możemy się odwoływać tak jak w przypadku tensorów **Numpy**:

```
A = tf.constant([[1.0, 1.0, 4.0, 3.0], [0.0, 1.0, -2.0, 7.0]])
print(A)
print(A[:,1:3]) # Drugi i trzecia kolumna z tensora A
print(A[1,:])   # Drugi wiersz z tensora A
```

```
tf.Tensor(
[[ 1.  1.  4.  3.]
 [ 0.  1. -2.  7.]], shape=(2, 4), dtype=float32)
tf.Tensor(
[[ 1.  4.]
 [ 1. -2.]], shape=(2, 2), dtype=float32)
tf.Tensor([ 0.  1. -2.  7.], shape=(4,), dtype=float32)
```

Operacja **tf.matmul** (odpowiednik mnożenia macierzy):

```
e = tf.constant([[2.0, -1.0], [-1.0, 5.0]])
d = tf.constant([[1.0, 1.0, 4.0], [0.0, 1.0, -2.0]])
f = tf.matmul(e,d)
f
```

```
<tf.Tensor: shape=(2, 3), dtype=float32, numpy=
array([[  2.,   1.,  10.],
       [ -1.,   4., -14.]], dtype=float32)>
```

Operacje **tf.multiply** i **tf.add**:

```
a = tf.constant([3.0,4.0])
b = tf.constant([-2.0,5.0])
#c = tf.multiply(a,b)
c = a*b
print(c)
#
d = tf.add(a,b)
d = a+b
print(d)
```

```
tf.Tensor([-6. 20.], shape=(2,), dtype=float32)
tf.Tensor([1. 9.], shape=(2,), dtype=float32)
```

**TensorFlow** jest ściśle zintegrowany z biblioteką **NumPy**. **Tensory w tensorflow można tworzyć z tensorów w Numpy**:

```
import numpy as np

#tensor Numpy
t_numpy = np.array([[[3],[1]],[[2],[4]]])

#wykorzystując t_numpy definiujemy tensor TensorFlow:
t_tf = tf.constant(t_numpy)

t_tf
```

```
<tf.Tensor: shape=(2, 2, 1), dtype=int64, numpy=
array([[[3],
        [1]],

       [[2],
        [4]]])>
```

Możliwa jest też konwersja w drugą stonę (**TensorFlow** --> **Numpy**):

```
#TensorFlow
t_tf = tf.constant([[1.0, 1.0, 4.0], [0.0, 1.0, -2.0]])
```

```
#Numpy
t_numpy = np.array(t_tf)
t_numpy
```

```
array([[ 1.,  1.,  4.],
       [ 0.,  1., -2.]], dtype=float32)
```

Typy **TensorFlow** są zgodne z odpowiadającymi im typami **Numpy**. Przykład:

```
print(tf.int64==np.int64)
```

```
True
```

**ZMIENNE**

W **TensorFlow** zmienne są obiektami tensorowymi przechowującymi **wartości, które można modyfikować** podczas wykonywania programu.

*Podczas trenowania modelu zmiennych używamy do przechowywania i aktualizowania parametrów modelu. Zmienne to bufory w pamięci zawierające tensory.*

Przykładowe definicje **zmiennych**:

```
a = tf.Variable(4.0)
b = tf.Variable([1.0, 2.0])
c = tf.Variable([[1.0, 1.0, 4.0], [0.0, 1.0, -2.0]],dtype=tf.float32) # z określeniem typu
print(a)
print(b)
print(c)
```

```
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=4.0>
<tf.Variable 'Variable:0' shape=(2,) dtype=float32, numpy=array([1., 2.], dtype=float32)>
<tf.Variable 'Variable:0' shape=(2, 3) dtype=float32, numpy=
array([[ 1.,  1.,  4.],
       [ 0.,  1., -2.]], dtype=float32)>
```

Zwróćmy uwagę, że "wartość zmiennej" jest powyżej wartością atrybutu **numpy**. Dostęp do wartości tego atrybutu daje metoda **numpy()**. Porównajmy:

```
print(a)
print(a.numpy())
```

```
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=4.0>
4.0
```

Na zmiennych można wykonywać operacje tak jak na stałych np. **tf.matmul**:

```
c = tf.Variable([[1.0, 2.0], [3.0, 4.0]])
d = tf.Variable([[1.0, 1.0, 3.0], [0.0, 1.0, -2.0]])
e = tf.matmul(c, d)
e
```

```
<tf.Tensor: shape=(2, 3), dtype=float32, numpy=
array([[ 1.,  3., -1.],
       [ 3.,  7.,  1.]], dtype=float32)>
```

Za pomocą metody **assign** możemy do zmiennej przypisać nową wartość:

```
A = tf.Variable(4.0)
A.assign(3.0)
A.numpy()
```

```
3.0
```

Za pomocą metody **assign_add** możemy do wartości zmiennej **dodać** pewną wartość:

```
A = tf.Variable(4.0)
A.assign_add(3.0)
A.numpy()
```

```
7.0
```

Za pomocą metody **assign_sub** możemy od wartości zmiennej **odjąć** pewną wartość:

```
A = tf.Variable(4.0)
A.assign_sub(3.0)
A.numpy()
```

```
1.0
```

Sprawdzamy czy zmienna **x** zmienia wartość:

```
x = tf.Variable(0.0)
for i in range(20):
  x.assign(i)
  print(x.numpy())
```

```
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
```