Laboratorium 1
Metody Numeryczne

¶

Biblioteki niezbędne do wykonania zadania:

(wykonanie tego bloku zapewnia do nich dostęp w kolejnych blokach)

In [1]:
```
#import main
import math
import numpy.linalg as linalg
import numpy as np
import sys
import scipy
```

Przydatne w trakcie zajęć mogą okazać się metody macierzy z pakietu Numpy, takie jak na przykład długość wektora - *len ()* czy rozmiar macierzy - *shape (https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.shape.html#numpy.nda* Poniższy kod ilustruje ich podstawowe działanie. Dodatkowe metody obiektu $ndarray$ można znaleźć w oficjalnej *dokumentacji (https://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html)*.

In [2]:
```
vector = np.array([1, 2, 3])
print('Wektor:')
print(vector)
print('Długość:', len(vector))
print('Rozmiar:', vector.shape, '\n')

matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print('Macierz:')
print(matrix)
print('Rozmiar:', matrix.shape)
```

```
Wektor:
[1 2 3]
Długość: 3
Rozmiar: (3,)

Macierz:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Rozmiar: (3, 3)
```

Poniżej znadujdą się funkcje niezbędne do zaimplementowania w ramach zadań na laboratorium:

In [3]:
```
def cylinder_area(r,h):
    return (2*math.pi*float(r)*float(h))+(2*math.pi*(float(r)**2))


def fib(n):
    lista_fib = []
    a, b = 0,1
    for _ in range(n):
        lista_fib.append(a)
        a, b = b, a + b
    return lista_fib


def matrix_calculations(a):
    matrix = np.array([[a, 1, -a], [0, 1, 1], [-a, a, 1]])
    inv_matrix = linalg.inv(matrix)
    transpose_matrix = np.transpose(matrix)
    Mdet = linalg.det(matrix)
    return (matrix, inv_matrix, transpose_matrix, Mdet)

def custom_matrix(m, n):
    if 3 <= m <= 7 and 3 <= n <= 7: # Sprawdzenie, czy m i n jest w przedzia
        result_matrix = np.zeros(m, n)  # Inicjalizacja macierzy wynikowej w
        for i in range(m):
            for j in range(n):
                if i > j:
                    result_matrix[i, j] = i
                else:
                    result_matrix[i, j] = j
        return result_matrix
    else:
        return None
```

Zadanie 1.

Zaimplementuj funkcję $cylinder\_area$ tak by zwracała pole powierzchni walca o promieniu podstawy r i wysokości h. Stała π jest zdefiniowana np. w bibliotece *math (https://docs.python.org/3/library/math.html#constants)*. Jeżeli nie da się policzyć pola funkcja powinna zwracać wartość *NaN*. Sprawdź działanie zaimplementowanej funkcji dla dowolnych wartości w tym notatniku.

In [4]:
```
try:
    r_value = 3
    h_value = k
    if r_value < 0 or h_value < 0:
        print('Wprowadzono wartość poniżej zera.')
    else:
        print('Pole walca to: ', cylinder_area(r_value, h_value))

except (ValueError, NameError):
    print("Wprowadzona wartość nie jest liczbą zmiennoprzecinkową.")
```

```
Wprowadzona wartość nie jest liczbą zmiennoprzecinkową.
```

Zadanie 2.

Wygeneruj dowolne ciągi arytmetyczny o kroku różnym od jeden i niebędącym liczbą całkowitą, używając w tym celu funkcji *arange*

(https://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html) oraz linspace (https://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.linspace.html). Wypisz

In [5]:
```
ciag_arange = np.arange(1.34,20,0.5)
print("Ciąg arytmetyczny generowany przez numpy.arange: ")
print(ciag_arange)
print("Długość ciągu:", len(ciag_arange))
print("Suma elementów ciągu:", np.sum(ciag_arange))
print()
ciag_linspace = np.linspace(1,10,20)
print("Ciąg arytmetyczny generowany przez numpy.linspace: ")
print(ciag_linspace)
print('Długość ciągu: ', len(ciag_linspace))
print('Suma elementów ciągu: ', np.sum(ciag_linspace))
```

```
Ciąg arytmetyczny generowany przez numpy.arange:
[ 1.34  1.84  2.34  2.84  3.34  3.84  4.34  4.84  5.34  5.84  6.34  6.84
  7.34  7.84  8.34  8.84  9.34  9.84 10.34 10.84 11.34 11.84 12.34 12.84
 13.34 13.84 14.34 14.84 15.34 15.84 16.34 16.84 17.34 17.84 18.34 18.84
 19.34 19.84]
Długość ciągu: 38
Suma elementów ciągu: 402.41999999999985

Ciąg arytmetyczny generowany przez numpy.linspace:
[ 1.          1.47368421  1.94736842  2.42105263  2.89473684  3.36842105
  3.84210526  4.31578947  4.78947368  5.26315789  5.73684211  6.21052632
  6.68421053  7.15789474  7.63157895  8.10526316  8.57894737  9.05263158
  9.52631579 10.        ]
Długość ciągu:  20
Suma elementów ciągu:  110.0
```

Zadanie 3.

Zaimplementuj funkcję fib zwracającą wektor pierszych n elementów ciągu Fibonnaciego (https://pl.wikipedia.org/wiki/Ci%C4%85g_Fibonacciego), jeżeli nie jest to możliwe funkcja powinna zwrócić wartość *None*.

In [6]:
```
#Podaj numer n elementów ciagu Fibonnaciego
n_elementow = 3

print(fib(n_elementow))
```

```
[0, 1, 1]
```

Zadanie 4.

Napisz funkję $matrix\_calculations$ która jako argument przyjmuje dowolną wartość liczbową *a* i tworzy macierz:

$$\mathbf{M} = \begin{bmatrix} a & 1 & -a \\ 0 & 1 & 1 \\ -a & a & 1 \end{bmatrix}$$

Dla zadeklarowanej macierzy wyznacz numerycznie macierz odwrotną $\mathbf{Minv}$ (jeżeli nie istnieje taka macierz wartość wynosi NaN), macierz transponowaną $\mathbf{Mt}$ i wyznacznik macierzy $\mathbf{Mdet}$. Zwróć otrzymane wartości w postaci krotki postaci $(\mathbf{Minv}, \mathbf{Mt}, \mathbf{Mdet})$. Wypisz otrzymane wyniki.

Wskazówki: Do tworzenia obiektów mających własności macierzy w języku Python używa się klasy *array (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.array.html)* z pakietu *numpy (http://www.numpy.org/)*,

In [7]:
```
#Podaj wartość argumentu a
a = 5
matrix, inv_matrix, transpose_matrix, Mdet = matrix_calculations(a)

print(f"Macierz wynosi: \n{matrix}\n")
print(f"Macierz odwrotna wynosi: \n{inv_matrix}\n")
print(f"Macierz transponowana wynosi: \n{transpose_matrix}\n")
print(f"Wyznacznik macierzy wynosi: \n{Mdet}\n")
```

```
Macierz wynosi:
[[ 5  1 -5]
 [ 0  1  1]
 [-5  5  1]]

Macierz odwrotna wynosi:
[[ 0.08  0.52 -0.12]
 [ 0.1   0.4   0.1 ]
 [-0.1   0.6  -0.1 ]]

Macierz transponowana wynosi:
[[ 5  0 -5]
 [ 1  1  5]
 [-5  1  1]]

Wyznacznik macierzy wynosi:
-49.99999999999997
```

In [8]:
```
import numpy as np # słowo kluczowe "as" oznacza przesłania nazwę numpy i po
a = np.array([1, 2, 3])
b = np.array([[1], [2], [3]])
A = np.array([[1,2],[3,4]])
print("Wektor poziomy:\n {0}".format(a))
print("Wektor pionowy:\n {0}".format(b))
print("Macierz:\n {0}".format(A))
```

```
Wektor poziomy:
 [1 2 3]
Wektor pionowy:
 [[1]
 [2]
 [3]]
Macierz:
 [[1 2]
 [3 4]]
```

Do wykonania operacji odwracania macierzy należy użyć funkcji *inv (https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.inv.html)*, do obliczenia wyznacznika macierzy stosuje się funkcję *det (https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.linalg.det.html)* z bliblioteki *linalg (https://docs.scipy.org/doc/numpy/reference/routines.linalg.html)*. Natomiast transpozycję

macierzy wykonujemy przez funkcję *transpose (https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.transpose.html)* której skrucona wersja to $M.T$ z pakietu *numpy*.

Ciekawostka: Python natywnie nie zawiera struktury danych typu tablica, na poziomie języka jest to rozwiązane poprzez strukture listy list do której elementów odwołuje sie poprzez

Zadanie 5.
Stwórz w noataniku macierz:

$$\mathbf{M} = \begin{bmatrix} 3 & 1 & -2 & 4 \\ 0 & 1 & 1 & 5 \\ -2 & 1 & 1 & 6 \\ 4 & 3 & 0 & 1 \end{bmatrix}$$

Wypisz przy pomocy funkcji *print* następujące elementy macierzy $\mathbf{M}$: $\mathbf{M}_{1,1}, \mathbf{M}_{3,3}, \mathbf{M}_{3,2}$. Zdefiniuj wektor $\mathbf{w1}$ którego elementy to trzecia kolumna macierzy $\mathbf{M}$ oraz wektor $\mathbf{w2}$ który składa się z drugiego wiersza tej macierzy.

Wskazówki: Tablice z pakietu *numpy* są indeksowane od zera a do każdego elementu można odwołać się poprzez jego indeks. Przykład użycie

In [9]:
```
# Inicjalizacja macierzy do przykładu
P = np.array([[1,3,2],[3,4, 6],[7,8,9]])
print("Macierz P=\n{0}".format(P))
# wyciągnięcie trzeciej kolumny
wektor1 = P[:,2]
# wyciągnięcie trzeciego wiersza
wektor2 = P[2,:]
print("Elementy trzeciej kolumny:\n {0}".format(wektor1))
print("Elementy trzeciego wiersza:\n {0}".format(wektor2))
```

```
Macierz P=
[[1 3 2]
 [3 4 6]
 [7 8 9]]
Elementy trzeciej kolumny:
 [2 6 9]
Elementy trzeciego wiersza:
 [7 8 9]
```

In [10]:
```
M = np.array([[3, 1, -2, 4],
              [0, 1, 1, 5],
              [-2, 1, 1, 6],
              [4, 3, 0, 1]])


print("Element M[1,1]:", M[0, 0])
print("Element M[3,3]:", M[2, 2])
print("Element M[3,2]:", M[2, 1])


w1 = M[:, 2] # Stworzenie wektora w1
print("Wektor w1 (trzecia kolumna macierzy M):")
print(w1)


w2 = M[1, :] # Stworzenie wektora w2
print("Wektor w2 (drugi wiersz macierzy M):")
print(w2)
```

```
Element M[1,1]: 3
Element M[3,3]: 1
Element M[3,2]: 1
Wektor w1 (trzecia kolumna macierzy M):
[-2  1  1  0]
Wektor w2 (drugi wiersz macierzy M):
[0 1 1 5]
```

Dodatkowo twórcy biblioteki umożliwiają użytkownikowi na manipulację elementami tablicy poprzez operator ':', więcej szczegółów na temat jego użycia w artykule NumPy for Matlab users (https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html#numpy-for-matlab-users).

Zadanie 6.
Uzupełnij funkcję $custom\_matrix$, tak by parametry m, n określały wymiary macierzy wynikowej, która będzie wypełniona w/g algorytmu: jeśli indeks wiersza jest większy od indeksu kolumny wartością komórki jest indeks wiersza, w przeciwnym wypadku wartością komórki jest indeks kolumny. Funkcja jako wynik powinna zwracać uzupełnioną macierz, jeżeli nie jest to możliwe to powinna zwrócić *None*. Na koniec wyświetlić wynikową macierz dla dowolnych argumentów m, n z przedziału $\langle 3, 7 \rangle$.

Wskazówka: Inicjalizacja pustej macierz wykonywana jest w pakiecie Numpy przy pomocy funkcji *zeros (https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.zeros.html)*, zaś macierzy składającej się z jedynek *ones (https://docs.scipy.org/doc/numpy/reference/generated/numpy.ones.html)*.

In [11]:
```
zero_matrix = np.zeros((2, 2))
ones_matrix = np.ones((3,1))

print('zero_matrix: \n{}'.format(zero_matrix))
print('ones_matrix: \n{}'.format(ones_matrix))
```

```
zero_matrix:
[[0. 0.]
 [0. 0.]]
ones_matrix:
[[1.]
 [1.]
 [1.]]
```

In [12]:
```
# Argumenty dla funkcji
m = 6
n = 5
result = custom_matrix(m, n)
if result is not None:
    print("Wynikowa macierz:")
    print(result)
else:
    print("Podane wymiary macierzy są nieodpowiednie.")
```

```
Wynikowa macierz:
[[0 1 2 3 4]
 [1 1 2 3 4]
 [2 2 2 3 4]
 [3 3 3 3 4]
 [4 4 4 4 4]
 [5 5 5 5 5]]
```

Zadanie 7.

Biblioteka Numpy posiada własne metody, pozwalające na szybsze i wygotniejsze wykonywanie operacji na wektorach i macierzach. Kolejne zadania mają na celu przećwiczenie tych metod. Zamiast samemu implementować mnożenie macierzy, należy skorzystać właśnie z wbudowanych funkcji Numpy. Najbardzie podstawowe z nich to *np.multiply (https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.multiply.html)*, *np.dot (https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.dot.html)* oraz *np.matmul (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.matmul.html)*. Przed wykonaniem zadania należy zapoznać się z ich dokumentacją, aby stosować poprawną funkcję do danego typu mnożenia. Dodatkowo ciekawą i użyteczną funkcjonalnością Numpy, wykorzystywaną niekiedy przy dodawaniu macierzy jest *broadcasting*, proszę o zapoznaie się z opisem: *[1] (https://docs.scipy.org/doc/numpy-1.15.0/user/basics.broadcasting.html) [2] (https://www.tutorialspoint.com/numpy/numpy_broadcasting.htm)*.

Zainicjalizować dwa wektory v_1 i v_2:

$$v_1 = \begin{bmatrix} 1 \\ 3 \\ 13 \end{bmatrix} \quad v_2 = \begin{bmatrix} 8 \\ 5 \\ -2 \end{bmatrix}$$

Następnie wykonać operacje i wypisać ich wynik:

- $4 * v_1$
- $-v_2 + \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$
- $v_1 \circ v_2$ (w sensie mnożenia macierzy, tzw. mnożenie macierzy w sensie Cauchy'ego)
- $v_1 * v_2$ (w sensie mnożenia Hadamarda (element-wise))

Wskazówki: Warto wiedzieć o [np.dot (https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.dot.html)](https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.dot.html) i [np.multiply (https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.multiply.html)](https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.multiply.html). Zbadać jak zachowuje się przeciążony operator mnożenia $*$ oraz $@$ dla macierzy $ndarray$.

In [13]:
```
v1 = np.array([1, 3, 13])
v2 = np.array([8, 5, -2])

wynik1 = 4*v1
wynik2 = -v2 + np.array([2, 2, 2])
wynikCauchy = np.dot(v1, v2)
WynikHadamarda = np.multiply(v1,v2)

print(f'4 * v1 = \n{wynik1}\n')
print(f'-v2 + [2 2 2] = \n{wynik2}\n')
print(f"Mnożenie v1 v2 metodą Cauchy'ego = \n{wynikCauchy}\n")
print(f"Mnożenie v1 v2 metodą Hadamarda = \n{WynikHadamarda}\n")
```

```
4 * v1 =
[ 4 12 52]

-v2 + [2 2 2] =
[-6 -3  4]

Mnożenie v1 v2 metodą Cauchy'ego =
-3

Mnożenie v1 v2 metodą Hadamarda =
[  8  15 -26]
```

Zadanie 8. Zainicjalizować macierz M_1:

$$M_1 = \begin{bmatrix} 1 & -7 & 3 \\ -12 & 3 & 4 \\ 5 & 13 & -3 \end{bmatrix}$$

Następnie wykonać operacje i wypisać ich wynik:

- $3M_1$
- $3M_1 + \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
- M_1^T
- $M_1 \circ v_1$
- $v_2^T \circ M_1$

In [14]:
```
M1 = np.array([[1, -7, 3],
               [-12, 3, 4],
               [5, 13, -3]])

wynik1 = 3*M1
wynik2 = wynik1 * np.array([[1,1,1], [1,1,1], [1,1,1]])
wynik3 = M1.T
wynik4 = np.multiply(M1, v1)
wynik5 = v2.T * M1

print(f'Wynik 1: \n{wynik1}\nWynik 2:\n{wynik2}\nWynik 3:\n{wynik3}\nWynik 4
```

```
Wynik 1:
[[  3 -21   9]
 [-36   9  12]
 [ 15  39  -9]]
Wynik 2:
[[  3 -21   9]
 [-36   9  12]
 [ 15  39  -9]]
Wynik 3:
[[  1 -12   5]
 [ -7   3  13]
 [  3   4  -3]]
Wynik 4:
[[  1 -21  39]
 [-12   9  52]
 [  5  39 -39]]
Wynik 4:
[[  8 -35  -6]
 [-96  15  -8]
 [ 40  65   6]]
```

Materiały uzupełniające:

- Scipy Lecture Notes (http://www.scipy-lectures.org/index.html)
- NumPy for Matlab users (https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html#numpy-for-matlab-users)
- Python Tutorial - W3Schools (https://www.w3schools.com/python/default.asp)
- NumPy (https://www.numpy.org)
- Matplotlib (https://matplotlib.org/)
- Anaconda (https://www.anaconda.com/)
- Learn Python for Data Science (https://www.datacamp.com/learn-python-with-anaconda?utm_source=Anaconda_download&utm_campaign=datacamp_training&utm_medium=bar
- Learn Python (https://www.learnpython.org/)
- Wujek Google (https://google.pl) i Ciocia Wikipedia (https://pl.wikipedia.org/wiki/Wikipedia:Strona_g%C5%82%C3%B3wna)