Звіт до лабораторної роботи №2

з предмету "Моделі інтелектуальних систем"

Дровольського Ярослава, ІПС-42

Варіант №2

Install numpy for Jupyter Notebook environment

In [2]:
```
!pip install numpy
```

```
Collecting numpy
  Downloading numpy-1.26.3-cp311-cp311-win_amd64.whl.metadata (61 kB)
     ---------------------------------------- 0.0/61.2 kB ? eta -:--:--
     ------ --------------------------------- 10.2/61.2 kB ? eta -:--:--
     ------------ ------------------------- 20.5/61.2 kB 131.3 kB/s eta 0:00:01
     ------------------ ------------------ 30.7/61.2 kB 163.8 kB/s eta 0:00:01
     ------------------------ ------------ 41.0/61.2 kB 178.6 kB/s eta 0:00:01
     ------------------------------ ------ 51.2/61.2 kB 217.9 kB/s eta 0:00:01
     ---------------------------------------- 61.2/61.2 kB 191.6 kB/s eta 0:00:00
Downloading numpy-1.26.3-cp311-cp311-win_amd64.whl (15.8 MB)
   ---------------------------------------- 0.0/15.8 MB ? eta -:--:--
   ---------------------------------------- 0.0/15.8 MB 991.0 kB/s eta 0:00:16
    --------------------------------------- 0.2/15.8 MB 2.5 MB/s eta 0:00:07
    --------------------------------------- 0.4/15.8 MB 3.5 MB/s eta 0:00:05
   - -------------------------------------- 0.5/15.8 MB 3.0 MB/s eta 0:00:06
   -- ------------------------------------- 1.1/15.8 MB 5.0 MB/s eta 0:00:03
   --- ------------------------------------ 1.3/15.8 MB 4.8 MB/s eta 0:00:04
   ---- ----------------------------------- 1.8/15.8 MB 5.8 MB/s eta 0:00:03
   ----- ---------------------------------- 2.0/15.8 MB 5.3 MB/s eta 0:00:03
   ------ --------------------------------- 2.5/15.8 MB 6.1 MB/s eta 0:00:03
   ------- -------------------------------- 2.8/15.8 MB 6.0 MB/s eta 0:00:03
   -------- ------------------------------- 3.2/15.8 MB 6.3 MB/s eta 0:00:02
   -------- ------------------------------- 3.5/15.8 MB 6.3 MB/s eta 0:00:02
   --------- ------------------------------ 3.8/15.8 MB 6.4 MB/s eta 0:00:02
   ---------- ----------------------------- 4.2/15.8 MB 6.4 MB/s eta 0:00:02
   ----------- ---------------------------- 4.6/15.8 MB 6.5 MB/s eta 0:00:02
   ------------ --------------------------- 4.9/15.8 MB 6.7 MB/s eta 0:00:02
   ------------- -------------------------- 5.3/15.8 MB 6.6 MB/s eta 0:00:02
   -------------- ------------------------- 5.6/15.8 MB 6.7 MB/s eta 0:00:02
   --------------- ------------------------ 6.0/15.8 MB 6.7 MB/s eta 0:00:02
   --------------- ------------------------ 6.3/15.8 MB 6.7 MB/s eta 0:00:02
   ---------------- ----------------------- 6.4/15.8 MB 6.7 MB/s eta 0:00:02
   ----------------- ---------------------- 7.0/15.8 MB 6.7 MB/s eta 0:00:02
   ----------------- ---------------------- 7.2/15.8 MB 6.7 MB/s eta 0:00:02
   ------------------ --------------------- 7.5/15.8 MB 6.7 MB/s eta 0:00:02
   ------------------- -------------------- 7.8/15.8 MB 6.6 MB/s eta 0:00:02
   -------------------- ------------------- 8.1/15.8 MB 6.6 MB/s eta 0:00:02
   -------------------- ------------------- 8.3/15.8 MB 6.6 MB/s eta 0:00:02
   --------------------- ------------------ 8.6/15.8 MB 6.6 MB/s eta 0:00:02
   ---------------------- ----------------- 9.0/15.8 MB 6.5 MB/s eta 0:00:02
   ---------------------- ----------------- 9.3/15.8 MB 6.6 MB/s eta 0:00:01
   ----------------------- ---------------- 9.6/15.8 MB 6.5 MB/s eta 0:00:01
   ----------------------- ---------------- 9.9/15.8 MB 6.5 MB/s eta 0:00:01
   ------------------------ --------------- 10.2/15.8 MB 6.5 MB/s eta 0:00:01
   ------------------------ --------------- 10.5/15.8 MB 6.8 MB/s eta 0:00:01
   ------------------------- -------------- 10.7/15.8 MB 6.8 MB/s eta 0:00:01
   -------------------------- ------------- 11.2/15.8 MB 6.7 MB/s eta 0:00:01
   -------------------------- ------------- 11.5/15.8 MB 6.8 MB/s eta 0:00:01
   --------------------------- ------------ 11.8/15.8 MB 6.8 MB/s eta 0:00:01
   ---------------------------- ----------- 12.1/15.8 MB 6.9 MB/s eta 0:00:01
   ---------------------------- ----------- 12.4/15.8 MB 6.7 MB/s eta 0:00:01
   ----------------------------- ---------- 12.7/15.8 MB 6.7 MB/s eta 0:00:01
   ------------------------------ --------- 13.0/15.8 MB 6.6 MB/s eta 0:00:01
   ------------------------------ --------- 13.4/15.8 MB 6.6 MB/s eta 0:00:01
   ------------------------------- -------- 13.7/15.8 MB 6.6 MB/s eta 0:00:01
   -------------------------------- ------- 13.9/15.8 MB 6.5 MB/s eta 0:00:01
   -------------------------------- ------- 14.2/15.8 MB 6.5 MB/s eta 0:00:01
   --------------------------------- ------ 14.6/15.8 MB 6.5 MB/s eta 0:00:01
   --------------------------------- ------ 14.9/15.8 MB 6.5 MB/s eta 0:00:01
   ---------------------------------- ----- 15.2/15.8 MB 6.4 MB/s eta 0:00:01
```

```
   ---------------------------------------  15.5/15.8 MB 6.5 MB/s eta 0:00:01
   ---------------------------------------  15.8/15.8 MB 6.5 MB/s eta 0:00:01
   ---------------------------------------  15.8/15.8 MB 6.5 MB/s eta 0:00:01
   ---------------------------------------- 15.8/15.8 MB 6.2 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-1.26.3
```

Імпортовуємо бібліотеку NumPy

In [2]:
```
import numpy as np
```

In [2]:
```
# checks if given string str represents positive integer
def is_positive_integer(str):
    return str.isdecimal() and int(str) > 0
```

Завдання 1

Виконати набір вправ. Всі завдання виконати 2 способами:

- з використанням універсальних функцій бібліотеки NumPy
- за допомогою ітеративних конструкцій (з використанням циклів, спискових включень тощо).
- для обох випадків підрахувати час виконання скрипту та зробити висновки.овки.

Вправа 2

Вводиться число n. Розставити 1 та 0 у шаховому порядку, починаючи з 1 в матриці розміром n×n, використовуюч слайсінг.

In [33]:
```
import time

def create_chessboard_1(n):
    chessboard = np.zeros((n, n), dtype=int)

    # put 1 in definite places in order to get chessboard
    chessboard[::2, ::2] = 1
    chessboard[1::2, 1::2] = 1

    return chessboard

def create_chessboard_2(n):
    chessboard = np.zeros((n, n), dtype=int)

    # put 1 in definite places in order to get chessboard
    for i in range(0, n):
        for j in range(0,n):
            if (i+j) % 2 == 0:
                chessboard[i,j] = 1

    return chessboard


def task_1_2():
    error = False
```

```
    n_str = input("Enter n (positive integer): ")

    if is_positive_integer(n_str) == False :
            print("ERROR: n must be a positive integer")
            error = True

    if error == False :
        n = int(n_str)

        t = time.time()
        create_chessboard_1(n)
        print("Time using NumPy functions: " + str(time.time() - t) + " s")

        t = time.time()
        create_chessboard_2(n)
        print("Time using standard functions: " + str(time.time() - t) + " s")
```

In [34]:
```
task_1_2()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.002821207046508789 s
```

In [35]:
```
task_1_2()
```

```
Time using NumPy functions: 0.0020754337310791016 s
Time using standard functions: 0.24732279777526855 s
```

In [36]:
```
task_1_2()
```

```
ERROR: n must be a positive integer
```

In [37]:
```
task_1_2()
```

```
ERROR: n must be a positive integer
```

Висновок: код, що використовує універсальні функції бібліотеки NumPy, працює швидше, ніж код, що використовує ітеративні конструкції

Вправа 3

Вводяться 4 числа n, m, r, c. Вивести масив розміру n×m, в якому в кожному рядку з номером r і в кожному стовпчику
номером c стоять 0, а ін і елементи дорівнюють 1.

In [6]:
```
def task_1_3_impl_1(n, m, r, c):
    matrix = np.ones((n, m), dtype=int)

    matrix[::r, 0::] = 0
    matrix[0::, ::c] = 0

    return matrix


def task_1_3_impl_2(n, m, r, c):
    matrix = np.ones((n, m), dtype=int)

    for i in range(0, n):
        for j in range(0, m):
```

```
            if (i % r == 0) or (j % c == 0) :
                matrix[i,j] = 0

    return matrix

# print(task_1_3_impl_2(1,1,5,10))
```

In [9]:
```
import time

def task_1_3():
    error = False

    n_str = input("Enter n (positive integer): ")
    m_str = input("Enter m (positive integer): ")
    r_str = input("Enter r (positive integer): ")
    c_str = input("Enter c (positive integer): ")

    if is_positive_integer(n_str) == False :
            print("ERROR: n must be a positive integer")
            error = True
    if is_positive_integer(m_str) == False :
            print("ERROR: m must be a positive integer")
            error = True
    if is_positive_integer(r_str) == False :
            print("ERROR: r must be a positive integer")
            error = True
    if is_positive_integer(c_str) == False :
            print("ERROR: c must be a positive integer")
            error = True

    if error == False :
        n = int(n_str)
        m = int(m_str)
        r = int(r_str)
        c = int(c_str)

        t = time.time()
        matrix = task_1_3_impl_1(n, m, r, c)
        print("Time using NumPy functions: " + str(time.time() - t) + " s")

        t = time.time()
        task_1_3_impl_2(n, m, r, c)
        print("Time using standard functions: " + str(time.time() - t) + " s")

        print("Result matrix:")
        print(matrix)
```

In [53]:
```
task_1_3()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.005087137222290039 s
Result matrix:
[[0 0 0 ... 0 0 0]
 [0 1 1 ... 1 1 1]
 [0 1 1 ... 1 1 1]
 ...
 [0 1 1 ... 1 1 1]
 [0 1 1 ... 1 1 1]
 [0 1 1 ... 1 1 1]]
```

In [13]:
```
task_1_3()
```

```
Time using NumPy functions: 0.0009958744049072266 s
Time using standard functions: 0.16396117210388184 s
Result matrix:
[[0 0 0 ... 0 0 0]
 [0 1 1 ... 1 1 1]
 [0 1 1 ... 1 1 1]
 ...
 [0 1 1 ... 1 1 1]
 [0 1 1 ... 1 1 1]
 [0 1 1 ... 1 1 1]]
```

In [14]:
```
task_1_3()
```

```
Time using NumPy functions: 0.005393266677856445 s
Time using standard functions: 0.705045223236084 s
Result matrix:
[[0 0 0 ... 0 0 0]
 [0 1 1 ... 1 1 1]
 [0 1 1 ... 1 1 1]
 ...
 [0 1 1 ... 1 1 1]
 [0 1 1 ... 1 1 1]
 [0 1 1 ... 1 1 1]]
```

In [15]:
```
task_1_3()
```

```
ERROR: n must be a positive integer
ERROR: m must be a positive integer
ERROR: r must be a positive integer
ERROR: c must be a positive integer
```

Висновок: код, що використовує універсальні функції бібліотеки NumPy, працює швидше, ніж код, що використовує ітеративні конструкції

Вправа 5

Вводиться число n. Вивести масив розміру n×n, в якому в рядках з парними індексами стоять 1, а в інших – 0.

In [32]:
```
def task_1_5_impl_1(n):
    matrix = np.zeros((n, n), dtype=int)

    matrix[0::2, 0::] = 1

    return matrix


def task_1_5_impl_2(n):
    matrix = np.zeros((n, n), dtype=int)

    for i in range(0, n):
        for j in range(0, n):
            if (i % 2 == 0) :
                matrix[i,j] = 1

    return matrix
```

In [27]:
```
def task_1_5():
    error = False

    n_str = input("Enter n (positive integer): ")

    if is_positive_integer(n_str) == False :
            print("ERROR: n must be a positive integer")
            error = True

    if error == False :
        n = int(n_str)

        t = time.time()
        matrix = task_1_5_impl_1(n)
        print("Time using NumPy functions: " + str(time.time() - t) + " s")

        t = time.time()
        task_1_5_impl_2(n)
        print("Time using standard functions: " + str(time.time() - t) + " s")

        print("Result matrix:")
        print(matrix)
```

In [29]:
```
task_1_5()
```

```
Time using NumPy functions: 0.0009741783142089844 s
Time using standard functions: 0.001993417739868164 s
Result matrix:
[[1 1 1 ... 1 1 1]
 [0 0 0 ... 0 0 0]
 [1 1 1 ... 1 1 1]
 ...
 [0 0 0 ... 0 0 0]
 [1 1 1 ... 1 1 1]
 [0 0 0 ... 0 0 0]]
```

In [31]:
```
task_1_5()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.02858257293701172 s
Result matrix:
[[1 1 1 ... 1 1 1]
 [0 0 0 ... 0 0 0]
 [1 1 1 ... 1 1 1]
 ...
 [0 0 0 ... 0 0 0]
 [1 1 1 ... 1 1 1]
 [0 0 0 ... 0 0 0]]
```

In [30]:
```
task_1_5()
```

```
Time using NumPy functions: 0.09466004371643066 s
Time using standard functions: 11.323092222213745 s
Result matrix:
[[1 1 1 ... 1 1 1]
 [0 0 0 ... 0 0 0]
 [1 1 1 ... 1 1 1]
 ...
 [0 0 0 ... 0 0 0]
 [1 1 1 ... 1 1 1]
 [0 0 0 ... 0 0 0]]
```

In [33]:
```
task_1_5()
```

```
ERROR: n must be a positive integer
```

Висновок: код, що використовує універсальні функції бібліотеки NumPy, працює швидше, ніж код, що використовує ітеративні конструкції

Вправа 9

Вводиться число n. Створити масив значень від n до 0.

In [1]:
```
def task_1_9_impl_1(n):
    return np.flip(np.arange(n+1))

def task_1_9_impl_2(n):
    array = np.zeros(n+1, dtype=int)

    for i in range(0, n+1):
        array[i] = n-i

    return array
```

In [2]:
```
def task_1_9():
    error = False

    n_str = input("Enter n (positive integer): ")

    if is_positive_integer(n_str) == False :
            print("ERROR: n must be a positive integer")
            error = True

    if error == False :
        n = int(n_str)

        t = time.time()
        array = task_1_9_impl_1(n)
        print("Time using NumPy functions: " + str(time.time() - t) + " s")

        t = time.time()
        task_1_9_impl_2(n)
        print("Time using standard functions: " + str(time.time() - t) + " s")

        print("Result array:")
        print(array)
```

In [53]:
```
task_1_9()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.0 s
Result array:
[100  99  98  97  96  95  94  93  92  91  90  89  88  87  86  85  84  83
  82  81  80  79  78  77  76  75  74  73  72  71  70  69  68  67  66  65
  64  63  62  61  60  59  58  57  56  55  54  53  52  51  50  49  48  47
  46  45  44  43  42  41  40  39  38  37  36  35  34  33  32  31  30  29
  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11
  10   9   8   7   6   5   4   3   2   1   0]
```

In [54]:
```
task_1_9()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.0015587806701660156 s
Result array:
[10000  9999  9998 ...     2     1     0]
```

In [59]:
```
task_1_9()
```

```
Time using NumPy functions: 0.0010058879852294922 s
Time using standard functions: 0.10160398483276367 s
Result array:
[500000 499999 499998 ...      2      1      0]
```

In [55]:
```
task_1_9()
```

```
Time using NumPy functions: 0.2253127098083496 s
Time using standard functions: 14.958905220031738 s
Result array:
[100000000  99999999  99999998 ...         2         1         0]
```

In [58]:
```
task_1_9()
```

```
ERROR: n must be a positive integer
```

In [56]:
```
task_1_9()
```

```
ERROR: n must be a positive integer
```

In [57]:
```
task_1_9()
```

```
ERROR: n must be a positive integer
```

Висновок: код, що використовує універсальні функції бібліотеки NumPy, працює швидше, ніж код, що використовує ітеративні конструкції

Вправа 13

Розмістити на полі 8×8 нулі та одиниці в шахматному порядку, використовуючи функцію повторення (*).

In [3]:
```
# returned row starts from 1
def make_chessboard_row(n):
    row = [1,0] * int(n/2)

    if n % 2 != 0:
        row.append(1)
    return row

def task_1_13_impl_1(n):
    chessboard = []
```

```
    if n % 2 == 0 :
        row = make_chessboard_row(n)
        chessboard.append(row)
        chessboard.append(np.flip(row))
        chessboard = chessboard * int(n/2)
    else:
        chessboard = [1,0] * int(n*n / 2)
        chessboard.append(1)
    return np.reshape(chessboard, (n,n))

def task_1_13_impl_2(n):
    chessboard = np.zeros((n, n), dtype=int)

    # put 1 in definite places in order to get chessboard
    for i in range(0, n):
        for j in range(0,n):
            if (i+j) % 2 == 0:
                chessboard[i,j] = 1

    return chessboard
```

In [55]:
```
import time

def task_1_13():
    error = False

    n_str = input("Enter n (positive integer): ")

    if is_positive_integer(n_str) == False :
            print("ERROR: n must be a positive integer")
            error = True

    if error == False :
        n = int(n_str)

        t = time.time()
        matrix = task_1_13_impl_1(n)
        print("Time using NumPy functions: " + str(time.time() - t) + " s")

        t = time.time()
        task_1_13_impl_2(n)
        print("Time using standard functions: " + str(time.time() - t) + " s")

        print("Result matrix:")
        print(matrix)
```

In [13]:
```
task_1_13()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.0 s
Result matrix:
[[1 0 1 0 1]
 [0 1 0 1 0]
 [1 0 1 0 1]
 [0 1 0 1 0]
 [1 0 1 0 1]]
```

In [12]:
```
task_1_13()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.0 s
Result matrix:
[[1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]]
```

In [122…
```
task_1_13()
```

```
Time using NumPy functions: 0.0009655952453613281 s
Time using standard functions: 0.0014400482177734375 s
Result matrix:
[[1 0 1 ... 0 1 0]
 [0 1 0 ... 1 0 1]
 [1 0 1 ... 0 1 0]
 ...
 [0 1 0 ... 1 0 1]
 [1 0 1 ... 0 1 0]
 [0 1 0 ... 1 0 1]]
```

In [123…
```
task_1_13()
```

```
Time using NumPy functions: 2.244391679763794 s
Time using standard functions: 13.14900279045105 s
Result matrix:
[[1 0 1 ... 0 1 0]
 [0 1 0 ... 1 0 1]
 [1 0 1 ... 0 1 0]
 ...
 [0 1 0 ... 1 0 1]
 [1 0 1 ... 0 1 0]
 [0 1 0 ... 1 0 1]]
```

In [5]:
```
task_1_13()
```

```
ERROR: n must be a positive integer
```

In [6]:
```
task_1_13()
```

```
ERROR: n must be a positive integer
```

In [7]:
```
task_1_13()
```

```
ERROR: n must be a positive integer
```

Висновок: код, що використовує універсальні функції бібліотеки NumPy, працює швидше, ніж код, що використовує ітеративні конструкції

Вправа 14

Розмістити на полі 8×8 нулі та одиниці в шахматному порядку, використовуючи функцію numpy `tile()`.

In [3]:
```
# returned row starts from 1
def make_chessboard_row(n):
```

```
    row = [1,0] * int(n/2)

    if n % 2 != 0:
        row.append(1)
    return np.array(row)


def task_1_14_impl_1(n):
    row = make_chessboard_row(n)

    if n == 1 :
        return np.array([row])

    stripe = np.append(row, (1-row)).reshape(2, n) # it is correct (chess-ordere
    chessboard = np.tile(stripe, (n//2, 1)) # repeat one two-striped block along

    if n % 2 == 0 :
        return chessboard
    else:
        return np.vstack([chessboard, row])


def task_1_14_impl_2(n):
    chessboard = np.zeros((n, n), dtype=int)

    # put 1 in definite places in order to get chessboard
    for i in range(0, n):
        for j in range(0,n):
            if (i+j) % 2 == 0:
                chessboard[i,j] = 1

    return chessboard
```

In [4]:
```
import time

def task_1_14():
    error = False

    n_str = input("Enter n (positive integer): ")

    if is_positive_integer(n_str) == False :
            print("ERROR: n must be a positive integer")
            error = True

    if error == False :
        n = int(n_str)

        t = time.time()
        matrix = task_1_14_impl_1(n)
        print("Time using NumPy functions: " + str(time.time() - t) + " s")

        t = time.time()
        task_1_14_impl_2(n)
        print("Time using standard functions: " + str(time.time() - t) + " s")

        print("Result matrix:")
        print(matrix)
```

In [73]:
```
task_1_14()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.0 s
Result matrix:
[[1]]
```

In [74]:
```
task_1_14()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.0 s
Result matrix:
[[1 0 1 0 1]
 [0 1 0 1 0]
 [1 0 1 0 1]
 [0 1 0 1 0]
 [1 0 1 0 1]]
```

In [75]:
```
task_1_14()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.0 s
Result matrix:
[[1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]]
```

In [78]:
```
task_1_14()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.003131866455078125 s
Result matrix:
[[1 0 1 ... 0 1 0]
 [0 1 0 ... 1 0 1]
 [1 0 1 ... 0 1 0]
 ...
 [0 1 0 ... 1 0 1]
 [1 0 1 ... 0 1 0]
 [0 1 0 ... 1 0 1]]
```

In [79]:
```
task_1_14()
```

```
Time using NumPy functions: 0.1300508975982666 s
Time using standard functions: 16.742042064666748 s
Result matrix:
[[1 0 1 ... 0 1 0]
 [0 1 0 ... 1 0 1]
 [1 0 1 ... 0 1 0]
 ...
 [0 1 0 ... 1 0 1]
 [1 0 1 ... 0 1 0]
 [0 1 0 ... 1 0 1]]
```

In [5]:
```
task_1_14()
```

```
ERROR: n must be a positive integer
```

In [6]:
```
task_1_14()
```

```
ERROR: n must be a positive integer
```

In [7]:
```
task_1_14()
```

```
ERROR: n must be a positive integer
```

Висновок: код, що використовує універсальні функції бібліотеки NumPy, працює швидше, ніж код, що використовує ітеративні конструкції

Вправа 16

Заповнити вектор значеннями від 0 до n. Всі значення, що більші за n/4 та менші за 3n/4 замінити нулями.

In [7]:
```
import numpy as np

def task_1_16_impl_1(n):
    array = np.arange(n+1)
    array[(array > n/4) & (array < 3*n/4)] = 0
    return array

def task_1_16_impl_2(n):
    array = np.arange(n+1)

    for i in range(0, n+1):
        if (array[i] > n/4) and (array[i] < 3*n/4) :
            array[i] = 0

    return array
```

In [5]:
```
import time

def task_1_16():
    error = False

    n_str = input("Enter n (positive integer): ")

    if is_positive_integer(n_str) == False :
            print("ERROR: n must be a positive integer")
            error = True

    if error == False :
        n = int(n_str)

        t = time.time()
        array = task_1_16_impl_1(n)
        print("Time using NumPy functions: " + str(time.time() - t) + " s")

        t = time.time()
        task_1_16_impl_2(n)
        print("Time using standard functions: " + str(time.time() - t) + " s")

        print("Result array:")
        print(array)
```

In [16]:
```
task_1_16()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.0 s
Result array:
[ 0  1  2  0  0  0  0  0  8  9 10]
```

In [17]:
```
task_1_16()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.05921792984008789 s
Result array:
[    0     1     2 ...  9998  9999 10000]
```

In [9]:
```
task_1_16()
```

```
Time using NumPy functions: 0.008514642715454102 s
Time using standard functions: 7.8941895961761475 s
Result array:
[      0       1       2 ...  999998  999999 1000000]
```

In [10]:
```
task_1_16()
```

```
ERROR: n must be a positive integer
```

In [11]:
```
task_1_16()
```

```
ERROR: n must be a positive integer
```

In [12]:
```
task_1_16()
```

```
ERROR: n must be a positive integer
```

Висновок: код, що використовує універсальні функції бібліотеки NumPy, працює швидше, ніж код, що використовує ітеративні конструкції

Вправа 17

Заповнити вектор значеннями від 0 до n. Замінити знаки для всіх значень, що менші за n/2 та більші за 3n/4 на протилежні.

In [34]:
```
import numpy as np

def task_1_17_impl_1(n):
    array = np.arange(n+1)
    array[(array < n/2) | (array > 3*n/4)] = (-1) * array[(array < n/2) | (array
    return array

def task_1_17_impl_2(n):
    array = np.arange(n+1)

    for i in range(0, n+1):
        if (array[i] < n/2) or (array[i] > 3*n/4) :
            array[i] = (-1) * array[i]

    return array
```

In [35]:
```
import time

def task_1_17():
    error = False
```

```
    n_str = input("Enter n (positive integer): ")

    if is_positive_integer(n_str) == False :
            print("ERROR: n must be a positive integer")
            error = True

    if error == False :
        n = int(n_str)

        t = time.time()
        array = task_1_17_impl_1(n)
        print("Time using NumPy functions: " + str(time.time() - t) + " s")

        t = time.time()
        task_1_17_impl_2(n)
        print("Time using standard functions: " + str(time.time() - t) + " s")

        print("Result array:")
        print(array)
```

In [36]:
```
task_1_17()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.0 s
Result array:
[  0  -1  -2  -3  -4   5   6   7  -8  -9 -10]
```

In [37]:
```
task_1_17()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.0653533935546875 s
Result array:
[     0     -1     -2 ...  -9998  -9999 -10000]
```

In [38]:
```
task_1_17()
```

```
Time using NumPy functions: 0.02066946029663086 s
Time using standard functions: 7.16433572769165 s
Result array:
[       0       -1       -2 ...  -999998  -999999 -1000000]
```

In [39]:
```
task_1_17()
```

```
ERROR: n must be a positive integer
```

In [40]:
```
task_1_17()
```

```
ERROR: n must be a positive integer
```

In [41]:
```
task_1_17()
```

```
ERROR: n must be a positive integer
```

Висновок: код, що використовує універсальні функції бібліотеки NumPy, працює швидше, ніж код, що використовує ітеративні конструкції

Вправа 20

Згенерувати вектор із n випадкових елементів, що лежать в інтервалі (0,1). Замінити максимальний елемент на 0.

In [72]:
```
import numpy as np
import math

def task_1_20_impl_1(array):
    array_copy = np.copy(array)
    max_indices = np.where(array_copy == np.max(array_copy))

    array_copy[max_indices] = 0
    return array_copy


def task_1_20_impl_2(array):
    array_copy = np.copy(array)

    max_item = max(array_copy)

    for i in range(0, array_copy.size):
        if math.isclose(max_item, array_copy[i]) == True :
            array_copy[i] = 0
    return array_copy
```

In [80]:
```
import time

def task_1_20():
    error = False

    n_str = input("Enter n (positive integer): ")

    if is_positive_integer(n_str) == False :
            print("ERROR: n must be a positive integer")
            error = True

    if error == False :
        n = int(n_str)

        array = np.random.random(n)

        t = time.time()
        result_array = task_1_20_impl_1(array)
        print("Time using NumPy functions: " + str(time.time() - t) + " s")

        t = time.time()
        task_1_20_impl_2(array)
        print("Time using standard functions: " + str(time.time() - t) + " s")

        print("\nOriginal array:")
        print(array)
        print("Result array:")
        print(result_array)
```

In [81]:
```
task_1_20()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.0 s

Original array:
[0.79004229 0.76705957 0.35479101 0.76934704 0.71067404 0.94366045
 0.86651532 0.11568448 0.91891809 0.61431413]
Result array:
[0.79004229 0.76705957 0.35479101 0.76934704 0.71067404 0.
 0.86651532 0.11568448 0.91891809 0.61431413]
```

In [82]:
```
task_1_20()
```

```
Time using NumPy functions: 0.0 s
Time using standard functions: 0.0 s

Original array:
[0.17609779 0.14764824 0.15769118 0.15676041 0.99407655 0.37357729
 0.45786722 0.45896932 0.91408672 0.78792058 0.65241608 0.93998265
 0.69140192 0.12549802 0.64401781 0.34618314 0.12844803 0.31170786
 0.1485834  0.69465253 0.04971429 0.38840722 0.0745326  0.22858046
 0.41378789 0.40503801 0.1759694  0.10631506 0.96941349 0.85400736
 0.59487385 0.67511991 0.33079881 0.86932108 0.28376377 0.35055741
 0.01297232 0.70314644 0.64784861 0.90626852 0.85949104 0.79344768
 0.5726976  0.42077766 0.64239937 0.95244609 0.98891825 0.64442743
 0.73522154 0.03624768 0.76114877 0.50605134 0.40333098 0.91135833
 0.08111944 0.13189308 0.77092098 0.92528661 0.11003949 0.65704571
 0.0396414  0.84520205 0.64804751 0.15240263 0.93099178 0.45023428
 0.0010138  0.98729173 0.57212515 0.34079594 0.0032709  0.12617724
 0.84711602 0.21657855 0.83690948 0.67204205 0.48852605 0.61313153
 0.16566463 0.55535782 0.15740694 0.49591223 0.53418484 0.41146402
 0.02361802 0.1971754  0.81389172 0.50222372 0.24532766 0.71674789
 0.80677633 0.04900325 0.67871186 0.97534797 0.70825887 0.73676891
 0.61072779 0.39650081 0.03641125 0.13807249]
Result array:
[0.17609779 0.14764824 0.15769118 0.15676041 0.         0.37357729
 0.45786722 0.45896932 0.91408672 0.78792058 0.65241608 0.93998265
 0.69140192 0.12549802 0.64401781 0.34618314 0.12844803 0.31170786
 0.1485834  0.69465253 0.04971429 0.38840722 0.0745326  0.22858046
 0.41378789 0.40503801 0.1759694  0.10631506 0.96941349 0.85400736
 0.59487385 0.67511991 0.33079881 0.86932108 0.28376377 0.35055741
 0.01297232 0.70314644 0.64784861 0.90626852 0.85949104 0.79344768
 0.5726976  0.42077766 0.64239937 0.95244609 0.98891825 0.64442743
 0.73522154 0.03624768 0.76114877 0.50605134 0.40333098 0.91135833
 0.08111944 0.13189308 0.77092098 0.92528661 0.11003949 0.65704571
 0.0396414  0.84520205 0.64804751 0.15240263 0.93099178 0.45023428
 0.0010138  0.98729173 0.57212515 0.34079594 0.0032709  0.12617724
 0.84711602 0.21657855 0.83690948 0.67204205 0.48852605 0.61313153
 0.16566463 0.55535782 0.15740694 0.49591223 0.53418484 0.41146402
 0.02361802 0.1971754  0.81389172 0.50222372 0.24532766 0.71674789
 0.80677633 0.04900325 0.67871186 0.97534797 0.70825887 0.73676891
 0.61072779 0.39650081 0.03641125 0.13807249]
```

In [83]:
```
task_1_20()
```

```
Time using NumPy functions: 0.0012173652648925781 s
Time using standard functions: 0.023110389709472656 s

Original array:
[0.95397985 0.32436108 0.72255348 ... 0.43442206 0.23689587 0.56486067]
Result array:
[0.95397985 0.32436108 0.72255348 ... 0.43442206 0.23689587 0.56486067]
```

In [86]:
```
task_1_20()
```

```
Time using NumPy functions: 0.04534173011779785 s
Time using standard functions: 2.1437437534332275 s

Original array:
[0.62770264 0.72401226 0.89313227 ... 0.31906153 0.18861307 0.3495027 ]
Result array:
[0.62770264 0.72401226 0.89313227 ... 0.31906153 0.18861307 0.3495027 ]
```

Висновок: код, що використовує універсальні функції бібліотеки NumPy, працює швидше, ніж код, що використовує ітеративні конструкції

Завдання 2

Розв'язати систему алгебраїчних рівнянь за допомогою формул Крамера і виконати перевірку за допомогою:

1. матричного множення;
2. оберненої матриці;
3. функції `numpy.linalg.solve()`

Порівняти всі рішення за допомогою функції `numpy.allclose()`

Дана така СЛАР:

$$\begin{cases} x_1 + 2x_2 + 3x_3 - 2x_4 = 6 \\ x_1 - x_2 - 2x_3 - 3x_4 = 8 \\ 3x_1 + 2x_2 - x_3 + 2x_4 = 4 \\ 2x_1 - 3x_2 + 2x_3 + x_4 = -8 \end{cases}$$

In [29]:
```
# about numpy: https://numpy.org/doc/stable/user/absolute_beginners.html#can-you

import numpy as np

# solves SLAE Ax=b
# A is square matrix; Det(A) must be != 0
# b is free vector (vector of right part of SLAE)
def solve_slae_cramer(A, b):
    # check for incorrect input
    if A.ndim != 2 :
        print("A must be a matrix")
        return
    if A.shape[0] != A.shape[1] :
        print("A must be a square matrix")
        return
    if b.ndim != 1 :
        print("b must be a vector")
```

```
        return
    if A.shape[0] != b.shape[0] :
        print("A and b must have same number of rows")
        return

    # solving SLAE using Cramer's rule
    x = np.zeros(A.shape[1])
    det_A = np.linalg.det(A)

    for i in range(A.shape[1]):
        temp = np.copy(A)
        temp[0:,i] = b[0:]
        x[i] = np.linalg.det(temp) / det_A

    return x
```

In [28]:
```
A = np.array([[1, 2, 3, -2],
              [1, -1, -2, -3],
              [3, 2, -1, 2],
              [2, -3, 2, 1]])

b = np.array([6, 8, 4, -8])


# solve
x = solve_slae_cramer(A, b)
print("Solution using Cramer's method: x = " + str(x))


# check by matrix multiplication
b1 = A @ x
print ("\nCheck #1:")
print("Does Ax = b? : " + str(np.allclose(b, b1)))

# check by inverse matrix (using fact that x = A^(-1) * b)
x1 = np.linalg.inv(A) @ b
print ("\nCheck #2:")
print("Does x = A^(-1) * b? : " + str(np.allclose(x, x1)))

# check by np.linalg.solve()
x2 = np.linalg.solve(A,b)
print ("\nCheck #3:")
print("Is x a solution? (compare with library solution)? : " + str(np.allclose(x
```

```
Solution using Cramer's method: x = [ 1.125   1.9375 -1.125  -2.1875]

Check #1:
Does Ax = b? : True

Check #2:
Does x = A^(-1) * b? : True

Check #3:
Is x a solution? (compare with library solution)? : True
```

Всі перевірки пройдено!

Завдання 3

Обчислити значення матричного виразу:

$$3A - (A - 2B)B \tag{1}$$

1. з використанням універсальних функцій бібліотеки NumPy
2. за допомогою ітеративних конструкцій (з використанням циклів, спискових включень тощо).
3. для обох випадків підрахувати час виконання скрипту та зробити висновок.

Порівняти всі рішення за допомогою функції `numpy.allclose()`.

де

$$A = \begin{pmatrix} 4 & 5 & -2 \\ 3 & -1 & 0 \\ 4 & 2 & 7 \end{pmatrix} \tag{2}$$

$$B = \begin{pmatrix} 2 & 1 & -1 \\ 0 & 1 & 3 \\ 5 & 7 & 3 \end{pmatrix} \tag{3}$$

In [1]:
```python
import numpy as np

# matrix is NumPy two-dimensional array
# returns matrix cpy multiplied by number
def multiply_by_number(matrix, number) :
    if matrix.ndim != 2 :
        print("A must be a matrix")
        return

    matrix_copy = np.copy(matrix)
    for i in range(matrix.shape[0]):
        for j in range(matrix.shape[1]):
            matrix_copy[i,j] = matrix[i,j] * number

    return matrix_copy

# returns new matrix that is result of A-B
def subtract_matrices(A, B):
    if A.ndim != 2 :
        print("A must be a matrix")
        return
    if B.ndim != 2 :
        print("B must be a matrix")
        return
    if (A.shape[0] != B.shape[0]) or (A.shape[1] != B.shape[1]):
        print("A and B must have the same shape")
        return

    n = A.shape[0]
    m = A.shape[1]

    result = np.zeros((n, m), dtype=A.dtype)

    for i in range(n) :
        for j in range(m) :
```

```
            result[i,j] = A[i,j] - B[i,j]

    return result

def multiply_matrices(A, B):
    if A.ndim != 2 :
        print("A must be a matrix")
        return
    if B.ndim != 2 :
        print("B must be a matrix")
        return
    if A.shape[1] != B.shape[0]:
        print("A.cols must be equal to B.rows")
        return

    n = A.shape[0]
    m = B.shape[1]
    p = A.shape[1]

    result = np.zeros((n, m), dtype=A.dtype)

    for i in range(n):
        for j in range(m):
            sum = 0
            for k in range(p):
                sum += A[i,k] * B[k,j]
            result[i,j] = sum

    return result
```

In [2]:
```
def task_3_impl_1(A, B):
    a1 = multiply_by_number(A, 3)
    a2 = multiply_by_number(B, 2)
    a3 = subtract_matrices(A, a2)
    a4 = multiply_matrices(a3, B)
    a5 = subtract_matrices(a1, a4)

    return a5


def task_3_impl_2(A, B):
    return 3*A - (A - 2*B)@B
```

In [3]:
```
import time

def task_3(A, B):
    # using NumPy funtions
    t_start = time.time()
    result_1 = task_3_impl_1(A,B)
    t_end = time.time()
    print("Using standard functions:")
    print(result_1)
    print("Time: " + str(t_end - t_start) + " s")

    # using standard functions
    t_start = time.time()
    result_2 = task_3_impl_2(A,B)
    t_end = time.time()
    print("\nUsing NumPy functions:")
```

```
    print(result_2)
    print("Time: " + str(t_end - t_start) + " s")

    print("\nAre the results the same?: " + str(np.allclose(result_1, result_2))
```

Виконання для матриць з умови:

In [4]:
```
A = np.array([[4, 5, -2],
              [3, -1, 0],
              [4, 2, 7]])

B = np.array([[2, 1, -1],
          [0, 1, 3],
          [5, 7, 3]])


task_3(A, B)
```

```
Using standard functions:
[[ 12  12 -15]
 [ 33  39  30]
 [ 19  17  48]]
Time: 0.0 s

Using NumPy functions:
[[ 12  12 -15]
 [ 33  39  30]
 [ 19  17  48]]
Time: 0.0 s

Are the results the same?: True
```

Виконання для того, щоб отримати порівнюваний час двох способів:

In [5]:
```
n = 100
A = np.ones((n, n), dtype=int)
B = np.ones((n, n), dtype=int)

task_3(A,B)
```

```
Using standard functions:
[[103 103 103 ... 103 103 103]
 [103 103 103 ... 103 103 103]
 [103 103 103 ... 103 103 103]
 ...
 [103 103 103 ... 103 103 103]
 [103 103 103 ... 103 103 103]
 [103 103 103 ... 103 103 103]]
Time: 0.36458420753479004 s

Using NumPy functions:
[[103 103 103 ... 103 103 103]
 [103 103 103 ... 103 103 103]
 [103 103 103 ... 103 103 103]
 ...
 [103 103 103 ... 103 103 103]
 [103 103 103 ... 103 103 103]
 [103 103 103 ... 103 103 103]]
Time: 0.0 s

Are the results the same?: True
```

Висновок: код, що використовує універсальні функції бібліотеки NumPy, працює швидше, ніж код, що використовує ітеративні конструкції. Окрім цього, результати обчислень за домогою цих двох способів однакові.