ⁱ Informasjon om eksamen

UNIVERSITY OF OSLO

Faculty of mathematics and natural sciences

Written exam in IN1900 - fall 2022

Exam date: December 2, 2022
Exam time: 0900 - 1300 (four hours)

Permitted aids: None.
A calculator is available in Inspera.

It is important to read this information carefully before you start.

The exam contains multiple choice questions, and text questions where you shall write short programs or read programs and write the output from the program. There are 15 questions that should be answered in total, and in total 75 points available on these questions. Question 16 shall not be answered, but is used by the examiners during the grading of the exam, to include the points from the mid term exams.

The number of points available is specified for each question. For questions with multiple sub-questions, each sub-question has the same number of points. On multiple choice questions you get the same score (0) for a wrong answer and for not answering at all, so you should always mark an answer.

If you are missing information you can make your own reasonable assumptions, as long as they are in line with the "nature" of the question. In text questions you should then specify the assumptions you made, for instance in comments to the code.

All code in the question texts is written in Python 3.

Most of the questions lead to short code with little need for comments, unless you do something complicated or non-standard (which is not recommended; but in this case the comments should explain the ideas behind the program to make it easier to evaluate the code).

A question may ask you to write a function. A main program which calls the function is in this case not needed, unless it is specifically asked for in the question text.

1 Finne verdi i en liste

We want to find all positions in a list **L** that contain a certain value **x**. The positions are to be stored in a list called **pos**. Below are two alternative suggestions for code to solve this task.

Alternative 1:
```
pos = [ ]
for i in range(0, len(L)):
  if L[i] == x:
    pos.append(x)
```

Alternative 2:
```
pos = [k for k in range(len(L)) if L[k]==x]
```

Which of the alternatives is/are correct?

Select one alternative:

○ Alternative 1 is correct

○ Alternative 2 is correct

○ Both are correct

○ Both are wrong

Maximum marks: 2

2 Hvilke hører sammen?

We have the following three lists: **x = [-1,0,1],   y = [1,2,3,4],   liste = [1, 2, [1], [2], [[1]], [[2]]].** To the left of the table below are seven expressions, and at the top of the table are six possible results of evaluating these expressions. Match the correct result to each expression.

Please match the values:

	1	2	[1]	[2]	[[1]]	[[2]]
liste[3]	○	○	○	○	○	○
liste.index([1])	○	○	○	○	○	○
liste[x[-1]]	○	○	○	○	○	○
liste[x[2]+y[2]]	○	○	○	○	○	○
liste[y[x[0]]]	○	○	○	○	○	○
liste[y[1]:y[2]]	○	○	○	○	○	○
liste[y[3]][0]	○	○	○	○	○	○

Maximum marks: 3.5

3 Hvilken linje mangler?

We want to write a program that asks the user for a mathematical expression and then computes and prints the answer. The expression should be a valid Python expression and can only include numbers, blanks, parentheses and the four basic operations +, -, *, /. Examples of valid expressions are:

4/3
2*(1+6) - 7/18 + 93

Here is a program sketch:

```
expression = input("Expression: ")
# A line is missing here
print(f"{expression} = {answer}")
```

One line is missing in the code. Which of the following lines must be added?
Select one alternative:

○ answer = double(expression)

○ answer = eval(expression)

○ answer = exec(expression)

○ answer = exec(lambda x: expression)

○ answer = expression

Maximum marks: 2

4 Funksjoner med parametre

We want to write a class that implements this function:

$$f(x; a, b, c) = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The function takes one argument (x) and has three parameters (a,b,c). It should be possible to set the values of the three parameters first, and then call the function by simply writing f(x). Here's a suggested solution:

```
import sqrt from math

class Quadratic:
    def __init__(self, a, b, c):
        self.a, self.b, self.c = a, b, c

    def __call__(x):
        r = sqrt(b*b - 4*a*c)
        solution1 = (-b-r) / (2*a)
        solution2 = (-b+r) / (2*a)
        return solution1, solution2

f = Quadratic(2, 5, 3)
print(f(2))
```

Unfortunately, the program does not work as it should. What error(s) are present in the program?

Fill in your answer here

Maximum marks: 3

5 Finne antall like verdier

We have two lists **x** and **y** containing numbers, and we want to find out how many common values there are in the lists. The answer should be stored in a variable named **count**. If, for instance, **x = [-5, -2, -6, -3, -8]** and **y = [-2, -3, 6, 14,-3]**, the variable **count** should get the value 2 since there are two common values (-2 and 3). Which of the code alternatives to the left will work? (Answer "Yes" for the code alternatives you think will work, and "No" for the alternatives that in your opinion do not give the correct result).

Please match the values:

	Yes	No
Alternative A	○	○
Alternative B	○	○
Alternative C	○	○
Alternative D	○	○
Alternative E	○	○

Maximum marks: 2.5

6 Terminerer løkken?

We have the following program:

```
import numpy as np
x = np.array([1,2])
y = np.array([1,2])
while <expression>:
    x = x * y
    y = x + y
```

where <expression> is one of the logical expressions below. Some of the logical expressions will make the loop terminate (i.e., the loop criterion will become False after a finite number of iterations), while others will result in an infinite loop. We will in this question disregard other events that can make the loop terminate, such as variable overflow or someone turning off the computer.

For each of the alternatives below, mark whether the loop terminates or gives an infinite loop. Recall that x[0] % 2 == 0 is a test that is True if x[0] is an even integer and False if x[0] is an odd integer.

Please match the values:

	Terminates	Infinite loop
x[0] < 8	○	○
x[0] < 8	○	○
x[0] <= y[0]	○	○
False	○	○
x[1] <= y[1]	○	○
x[0] < 1	○	○
x[0] % 2 == 0	○	○

Maximum marks: 3.5

7 Feilhåndtering (exceptions)

We have the following program:

```
import sys
x = [0, 1, 2, 3]
try:
    v1 = int(sys.argv[1])
    v2 = int(sys.argv[2])
    answer = x[v1] / x[v2]
except ValueError:
    print("Error A")
    sys.exit()
except IndexError:
    print("Error B")
    sys.exit()
except ZeroDivisionError:
    print("Error C")
    sys.exit()
```

We try to run the program from the command line with each of the alternatives shown below (on the left side of the table). Decide for each row in the table what is printed by the program (where "No output" means that nothing is printed).

Please match the values:

	Error A	Error B	Error C	No output
python divide.py	○	○	○	○
python divide.py verdi1 verdi2	○	○	○	○
python divide.py 0 0	○	○	○	○
python divide.py 1 1 0 3	○	○	○	○
python divide.py 0 -3	○	○	○	○

Maximum marks: 2.5
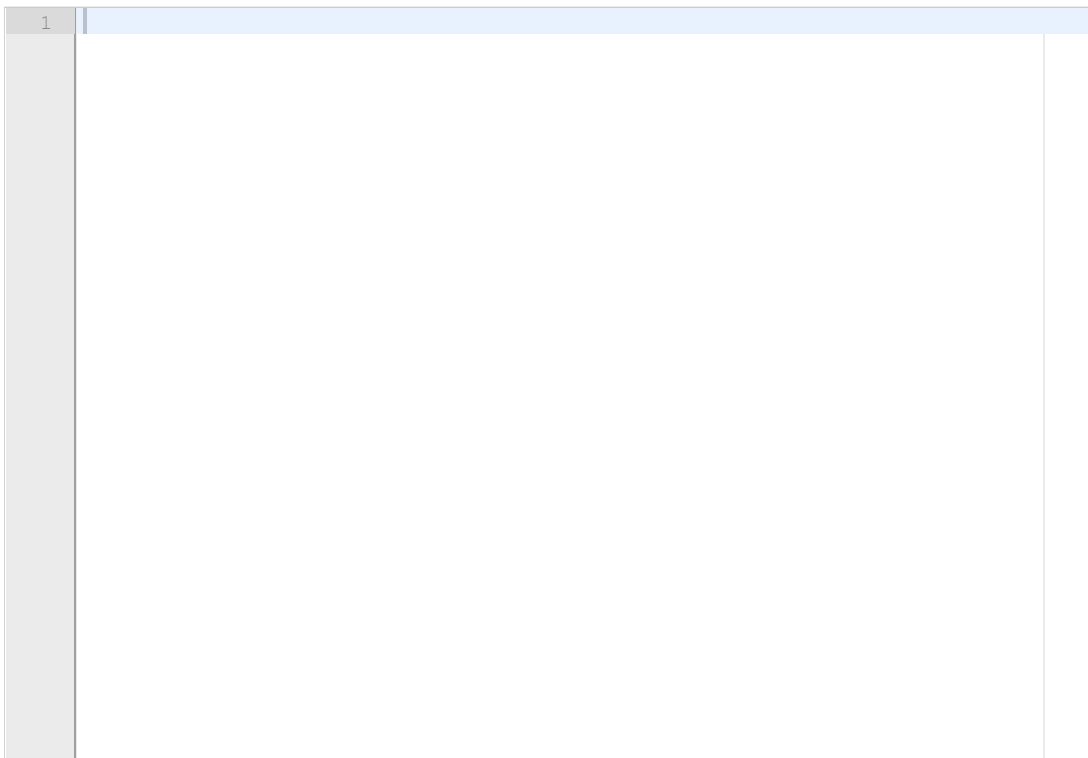
8 Summer og testfunksjoner

The inverse hyperbolic tangent function $\tanh^{-1} x$ has the following series approximation:

$$\tanh^{-1} x = x + \tfrac{1}{3}x^3 + \tfrac{1}{5}x^5 + \tfrac{1}{7}x^7 + \cdots$$

a) Write a Python function **inv_tanh(x,n)** that for a given x and n computes and returns the sum above, and which includes all terms with degree $\leq$ n.

b) Write a test function for **inv_tanh(x,n)**. The test function shall compare the answer from **inv_tanh(x,50)** with the answer from the function **atanh(x)** in the math module. We assume that the latter function gives the correct answer. The test is to be performed for three values of x in the open interval (-1,1). You can choose the x values yourself. For n=50 you can assume that the series approximation is accurate to at least 10 decimal places.

Fill in your answer here

```
1
```

Maximum marks: 6

value1 = 0, value2 = 98

10 Implementere et folkeregister

The government of the country Ruritania has a population register which for every citizen contains four pieces of information: social security number (ID number), name, occupation, and address. The population register is stored in a text file *register.txt* which has four columns and a header, and where the first three lines may look like this (the whole file is much longer, however):

Fnr	Navn	Yrke	Adresse
10125564233	Ole Olsen	Lege	Gåsemyrgaten 14, 0323 Gåseby
30069912345	Signe Nes	Lærer	Gufsealleen 5, 6233 Olsby

All entries in a column start in exactly the same character position as the header. Once per month the file is to be updated so that people who should no longer be in the register are removed (for instance those who died during the last month), and new citizens are added (for instance all newborn children and all new immigrants).

Your task is to make a Python program that performs this file update. The update is performed by first reading the file into a dictionary, with the ID number as key and the rest of the information as the associated value. All required changes (removing and adding persons) are then performed on this dictionary, and after this has been done the entire dictionary is written back to file.

People to be added to the register are listed in a separate file *newpersons.txt,* which has exactly the same format as the *register.txt* file described above. People to me removed are in a separate file *remove.txt*, which contains a single column with the ID numbers of the people to be removed (this file has no header).

a) Write the function **read_register()**, which reads the entire file *register.txt* to a dictionary, as explained above. The function should return the dictionary.

b) Write the function **add_to(register)** where the argument **register** is a dictionary made by the function in question a). The function should read the file *newpersons.txt* and make the necessary changes to the dictionary. The function should return the updated dictionary.

c) Write the function **remove_from(register)** where the argument **register** is a dictionary made by the function in question a). The function should read the file *remove.txt* and make the necessary changes to the dictionary. The function should then return the updated dictionary.

d) Write the function **write_register(register)** which writes the dictionary **register** to the file *register.txt*. You can assume that the existing file with the same name is not write protected, so your program will simply overwrite this file.

Fill in your answer here

```
1
```
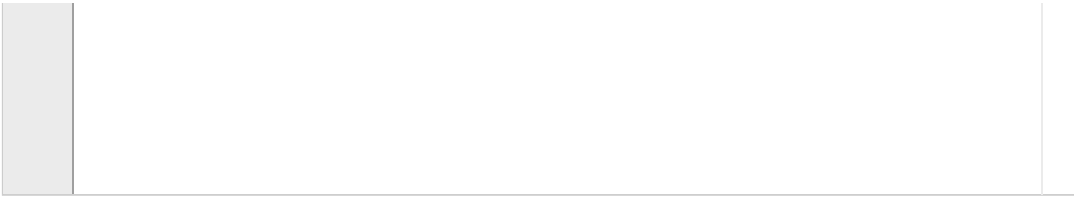
Maximum marks: 8

11 Beregne verdien til et polynom

Chebychev polynomials of the first kind are a sequence of polynomials $T_0(x), T_1(x), T_2(x), \ldots$ defined as follows:

$T_0(x) = 1$
$T_1(x) = x$
$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$ for $n = 1, 2, \ldots$

The first four polynomials in the sequence are:

$T_0(x) = 1, \ T_1(x) = x, \ T_2(x) = 2x^2 - 1, \ T_3(x) = 4x^3 - 3x.$

In general, $T_n(x)$ is a polynomial of degree n.

Write a Python function **chebychev(n, x)** that computes and returns the value of $T_n(x)$ in a given point x.

Hint: If you have computed the values $T_n(x)$ and $T_{n-1}(x)$, then it is easy to compute the value $T_{n+1}(x)$ from the formula above. Write a for loop which either stores all the values $T_0(x), T_1(x), T_2(x), \ldots$ in a list, or that just stores the two last values and updates these for each iteration.

Remark: The question is to be solved with a loop. It is also possible to solve the task with recursion, but this is <u>not</u> part of the course and it is <u>not</u> the solution we are seeking here.

Fill in your answer here

```
1
```

Maximum marks: 5

12 Finne polynomkoeffisienter

You should now write a program that, for a given value of n, finds the coefficients of the n+1 first Chebychev polynomials $T_0(x), T_1(x), \ldots, T_n(x)$. Recall that these are defined as $T_0(x) = 1$, $T_1(x) = x$, og $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$ for $n > 0$. In general, $T_0(x)$ is a polynomial of degree 0, $T_1(x)$ a polynomial of degree 1, $T_2(x)$ a polynomial of degree 2, etc.

Since all the n+1 polynomials have degree less than or equal to n, we can for $i = 0, 1, 2, \ldots, n$ write:

$$T_i(x) = B_{i,0} + B_{i,1}x + B_{i,2}x^2 + \cdots + B_{i,n}x^n$$

where $B_{i,0}, B_{i,1}, \ldots, B_{i,n}$ are the coefficients of the polynomial. That is, $B_{i,j}$ is the j'th coefficient of the i'th polynomial. Some of these coefficients will be equal to 0. For example, for the first polynomial $T_0(x) = 1$ (corresponding to $i = 0$) we have $B_{0,0} = 1$ while $B_{0,1} = B_{0,2} = \cdots = B_{0,n} = 0$. We want to compute all the polynomial coefficients $B_{i,j}$ and store them in a Numpy array. To do this we first create a 2-dimensional array of the correct size and with zeros in all entries:

B = np.zeros((n+1,n+1))

and we subsequently fill in the correct values $B_{i,j}$ into the array such that the array element **B[i,j]** becomes equal to the j'th coefficient of the i'th polynomial $T_i(x)$. When we are finished, the coefficients of the first polynomial $T_0(x)$ should be stored in **B[0,0], B[0,1], ..., B[0,n]**, and the coefficients of the next polynomial $T_1(x)$ should be stored in **B[1,0], B[1,1], ..., B[1,n]**, and so on.

The procedure to fill in values in the array **B** is as follows. We first fill in values in the first two rows of B which correspond to respectively $T_0(x)$ and $T_1(x)$. After that, we can use the following formulas to fill in values in the remaining rows of **B**, one row at a time ($i = 2, 3, \ldots, n$):

$$B_{i,0} = -B_{i-2,0}$$
$$B_{i,k} = 2B_{i-1,k-1} - B_{i-2,k} \text{ when } k > 0$$

Write a Python function **chebychev_coef(n)** that finds the coefficients of the n+1 first Chebychev polynomials with the method described above. The function should return the coefficients as a 2-dimensional numpy array with n+1 rows and n+1 columns. Take care that the special cases n=0 and n=1 are also handled correctly.

Fill in your answer here

Maximum marks: 5

13 Representere polynom som klasse

You will in this question write a class for representing the Chebychev polynomial $T_n(x)$. The class should have the following name and structure:

```
class Chebychev:
    def __init__(self, n):
        # Code missing here

    def __call__(self, x):
        # Code missing here

    def __str__(self):
        # Code missing here
```

The class should be possible to use in this way:

```
T = Chebychev(3)     # Instance of the class representing  T3(x)
print(T)             # OUTPUT:    4x^3-3x^1
print(T(2))          # Evaluate T4(x) for x = 2, OUTPUT:    26
```

a) Complete the method **__init__(self,n)**. You can assume that the function **chebychev_coef(n)** from the previous question is already present in your program and that it can be used without import. You can also assume that the latter function works as specified, even if you did not answer the previous question.

b) Complete the method **__call__(self,x)**. This method should compute and return the value of the Chebychev polynomial $T_n(x)$ for the given value of x.

c) Complete the method **__str__(self)**. This method should return a text representation of the Chebychev polynomial $T_n(x)$, as shown in the example above. The output should only include the non-zero terms and should look like normal mathematical formulae. We want the sign of each term to be handled correctly, so that negative terms are written as, for instance, "-3x" and not "+ -3x". For simplicity you can disregard other special cases. For instance, it is not necessary to simplify x^1 and x^0 to x and 1, respectively.

Fill in your answer here

```
1
```

Maximum marks: 9

14 Differenslikninger

We have the following system of difference equations for $n \geq 0$:

$$x_{n+1} = x_n + y_{n+1}$$
$$y_{n+1} = y_n + k\,x_n(x_n - 1)$$

The system is called the *Bogdanov map* after the Russian mathematician Rifkat Bogdanov.

a) Write a function **bogdanov(N, k, x0, y0)** that computes the solution (x_n, y_n) to the difference equations above, for $n = 1, 2, \ldots, N$. The three last arguments to the function are the parameter k and the initial conditions x_0 and y_0. The function should return the solution as a tuple (x,y), where x is a list (or array) with the solution values $x_0, x_1, x_2, \ldots, x_N$ and y is a list (or array) with the solution values $y_0, y_1, y_2, \ldots, y_N$.

b) We can not decide numerically whether an infinite sequence of numbers $x_0, x_1, x_2, \ldots$ converges or not. However, we can get an indication of convergence by computing the first N+1 values of the sequence and checking whether the last two numbers are approximately equal (which might indicate that the sequence is approaching convergence). Write a Python function **convergence(x, eps)** that uses this criterion to indicate whether a sequence converges. Here, x is a list containing the first values of a sequence and eps is a tolerance. The function shall return True if the two last values are closer than the tolerance, and otherwise False.

c) Explain briefly how the function you wrote in question b) can be used to indicate whether the solution returned from the function **bogdanov** in point a) converges.

d) Some difference equations have a solution that does not converge, but instead cycles through (i.e. repeats indefinitely) a finite number of values. For instance, the sequence 0,1,0,1,0,1,0,1, ... cycles through the values 0,1 while the sequence 0,1,2,0,1,2,0,1,2,... cycles through the values 0,1,2. The length of the cycle is 2 and 3, respectively, in these two cases. In general, a cycle can have length k > 0. This means that the first k values of the sequence, $x_0, x_1, x_2, \ldots, x_{k-1}$, are identical to the next k values $x_k, x_{k+1}, x_{k+2}, \ldots, x_{2k-1}$ (i.e., $x_0 = x_k$, $x_1 = x_{k+1}$, ..., $x_{k-1} = x_{2k-1}$), which in turn are equal to the next k values $x_{2k}, x_{2k+1}, x_{2k+2}, \ldots, x_{3k-1}$, etc. Assume that we have already written a function **isCycle(x,k)** which checks if the sequence x (a list) has a cycle of length k (a positive integer) and which in that case returns True (and otherwise False). Use this function to write a new function **findCycle(x, kmax)** that finds the smallest value of k in the interval $1, 2, \ldots, kmax$ such that x cycles through k values. If the function finds such a k it should return this value. If it does not find such a k, it should return 0.

Fill in your answer here

```
1
```

Maximum marks: 12

15 Planeters bane rundt sola

In this question we study the motion of a planet around the sun. Assume that the planet at time t has the position $(x(t), y(t), z(t))$ and the velocity vector $(u(t), v(t), w(t))$. Then, from Newton's laws, one can show that the motion is governed by these six differential equations:

$$
\begin{aligned}
x'(t) &= u(t) \\
y'(t) &= v(t) \\
z'(t) &= w(t) \\
u'(t) &= -x(t)/(x(t)^2 + y(t)^2 + z(t)^2)^{3/2} \\
v'(t) &= -y(t)/(x(t)^2 + y(t)^2 + z(t)^2)^{3/2} \\
w'(t) &= -z(t)/(x(t)^2 + y(t)^2 + z(t)^2)^{3/2}
\end{aligned}
$$

Write a Python program that solves this ODE system, using the ODESolver module (see attached file), when the initial conditions are given by $x(0) = y(0) = z(0) = u(0) = v(0) = z(0) = 1$. The program should use the RungeKutta4 method with time points given by

    time_points = np.linspace(0,50,5001)

where we assume that numpy has been imported with **import numpy as np**. The solution should be written to the screen as a table with four columns and with one row for each time point (see the example below). Time points should be written with two decimals, while the other values should be written with four decimals. Make sure that the table is nicely formatted so that numbers in the same column start in the same position on the line. Note that the table should only include time and position, while the velocity vector is not included in the output.

Time	x(t)	y(t)	z(t)
0.00	1.0000	1.0000	1.0000
0.01	...	...	...

You do not have to write code to plot the solution as curves. The relevant parts of the ODEsolver module are in the attached pdf file.

Fill in your answer here

```
1
```

Maximum marks: 6

16 Skal ikke besvares, til bruk under sensuren

This question is not to be answered, but is used by the examiners during the marking of the exam, to add the points from the midterm exam.

Not to be answered.

| Format ▾ | **B** *I* U x₂ x² | Iₓ | 🗇 📋 | ↶ → ⟲ | ⅟≣ ⋮≣ | Ω ⊞ | ✎ |
| Σ | ⤢ |

Words: 0

Maximum marks: 25

```python
import numpy as np

class ODESolver:
    def __init__(self, f):
        self.f = lambda u, t: np.asarray(f(u, t), float)

    def set_initial_condition(self, U0):
        if isinstance(U0, (float,int)):  # scalar ODE
            self.neq = 1                      # no of equations
            U0 = float(U0)
        else:                                 # system of ODEs
            U0 = np.asarray(U0)
            self.neq = U0.size            # no of equations
        self.U0 = U0

    def solve(self, time_points):
        self.t = np.asarray(time_points)
        N = len(self.t)
        if self.neq == 1:  # scalar ODEs
            self.u = np.zeros(N)
        else:              # systems of ODEs
            self.u = np.zeros((N,self.neq))

        self.u[0] = self.U0

        for n in range(N-1):
            self.n = n
            self.u[n+1] = self.advance()
        return self.u, self.t

class ForwardEuler(ODESolver):
    def advance(self):
        u, f, n, t = self.u, self.f, self.n, self.t
        dt = t[n+1] - t[n]
        unew = u[n] + dt*f(u[n], t[n])
        return unew

class RungeKutta4(ODESolver):
    def advance(self):
        u, f, n, t = self.u, self.f, self.n, self.t
        dt = t[n+1] - t[n]
        dt2 = dt/2.0
        k1 = f(u[n], t[n])
        k2 = f(u[n] + dt2*k1, t[n] + dt2)
        k3 = f(u[n] + dt2*k2, t[n] + dt2)
        k4 = f(u[n] + dt*k3, t[n] + dt)
        unew = u[n] + (dt/6.0)*(k1 + 2*k2 + 2*k3 + k4)
        return unew
```
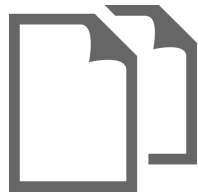
Question 5

Attached

```python
# Alternative A

count = 0
for e1 in x:
    for e2 in y:
        if e1==e2:
            count += 1

# Alternative B

count = 0
for e in x:
    if e in y:
        count += 1

# Alternative C

x.sort()  # Sort values in x from smallest to largest
y.sort()  # Sort values in y
count = 0
for e in x:
    k = 0
    while k < len(y) and y[k] < e:
        k = k + 1
    if y[k] == e:
        count +=1

# Alternative D

import numpy as np
x = np.array(x)
y = np.array(y)
count = sum(x==y)

# Alternative E

x = np.array(x)
y = np.array(y)
distance = [min(abs(k-y)) for k in x]
distance = np.array(distance)
count = sum(distance==0)
```