

i **Kopi av Forside**

Institutt for Datateknologi og Informatikk

Eksamensoppgave i TDT4110 Informasjonsteknologi Grunnkurs

Eksamensdato: 9. august 2023

Eksamenstid (fra-til): 15:00-19:00

Hjelpemiddelkode/Tillatte hjelpemidler: D

Ingen trykte eller håndskrevne hjelpemidler tillatt. Bestemt, enkel kalkulator tillatt.

Faglig kontakt under eksamen: Dag Olav Kjellemo, Guttorm Sindre

Tlf.: 47681639

Tlf.: 94430245

Faglig kontakt møter i eksamenslokalet: NEI

ANNEN INFORMASJON:

Skaff deg overblikk over oppgavesettet før du begynner på besvarelsen din.

Les oppgavene nøye, gjør dine egne antagelser og presiser i besvarelsen (på oppgaver der du skal skrive kode selv) eventuelle forutsetninger du har lagt til grunn i tolkning/avgrensning av oppgaven. Faglig kontaktperson skal kun kontaktes dersom det er direkte feil eller mangler i oppgavesettet. Henvend deg til en eksamensvakt hvis du ønsker å kontakte faglærer. Noter gjerne spørsmålet ditt på forhånd.

Vekting av oppgavene: Vekting i prosent er angitt for hver deloppgave.

Varslinger: Hvis det oppstår behov for å gi beskjeder til kandidatene underveis i eksamen (f.eks. ved feil i oppgavesettet), vil dette bli gjort via varslinger i Inspera. Et varsel vil dukke opp som en dialogboks på skjermen. Du kan finne igjen varselet ved å klikke på bjella øverst til høyre.

Trekk fra/avbrutt eksamen: Blir du syk under eksamen, eller av andre grunner ønsker å levere blankt/avbryte eksamen, gå til "hamburgermenyen" i øvre høyre hjørne og velg «Lever blankt». Dette kan ikke angres selv om prøven fremdeles er åpen.

Tilgang til besvarelse: Etter eksamen finner du besvarelsen din i arkivet i Inspera. Merk at det kan ta én virkedag før eventuelle håndtegninger vil være tilgjengelige i arkivet.

Useful Functions and Methods

These are listed in the following order:

- built-in (standard library)
 - often used functions and operators
 - exceptions
 - string methods
 - list operations
 - set operations
 - dictionary operations
 - files
 - pickle library (binary files)
- random library
- math library
- numpy library
- matplotlib.pyplot

Built-in:

f-string

Syntax: `f'.....{expression}...'` where expression can be variable or any expression. Can also use formatting characters such as : d=integer, f=floating-point, e=scientific notation, %=percentage, s=string. A number before the formatting character will specify field width. A number after the character "." will format the number of decimals. E.g. print PI with a field width of 5 with two decimals will be:
`print(f'{math.pi:5.2f}')`

%

Remainder (modulo operator): Divides one number by another and gives the remainder.

//

Floor/integer division: Returns the integral part of the quotient.

len(s)

Return the length (the number of items) of a string, tuple, list, dictionary or other data structure.

int(x)

Convert a string or number to a plain integer.

float(x)

Convert a string or a number to floating point number.

str([object])

Return a string containing a nicely printable representation of an object.

range(stop)

Returns a list with integers from 0, up to but not including stop. `range(3) = [0, 1, 2]`. Often used in combination with for loops: `for i in range(10)`

range([start], stop[, step])

start: Starting number of the sequence.

stop: Generate numbers up to, but not including, this number.

step: Difference between each number in the sequence.

chr(i)

Return a string of one character whose ASCII code is the integer i. For example, `chr(97)` returns the string 'a'. This is the inverse of `ord()`

ord()

Given a string of length one, return an integer representing the Unicode code point of the character when the argument is a unicode object, or the value of the byte when the argument is an 8-bit string. For example, `ord('a')` returns the integer 97.

tuple(iterable)

Accepts something iterable (list, range etc.) and returns the corresponding tuple. A tuple is immutable.

if x in iterable:

Returns True if x is an item in iterable.

for idx, x in enumerate(iterable)

Enters a loop with x being one and one item from iterable. idx is an integer containing the loop number.

Exceptions:

try:

Code to test

except:

If code fails. E.g exception types: IOError, ValueError, ZeroDivisionError.

Variant: except Exception as exc # Let's you print the exception.

else:

Runs if no exception occurs

finally:

Runs regardless of prior code having succeeded or failed.

String methods:

s.isalnum()

Returns true if the string contains only alphabetic letters or digits and is at least one character of length. Returns false otherwise.

s.isalpha()

Returns true if the string contains only alphabetic letters, and is at least one character in length. Returns false otherwise.

s.isdigit()

Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.

s.center(width)

Return the string center justified in a string of length width.

s.ljust(width)

Return the string left justified in a string of length width.

s.rjust(width)

Return the string right justified in a string of length width.

s.lower()

Returns a copy of the string with all alphabetic letters converted to lowercase.

s.upper()

Returns a copy of the string with all alphabetic letters converted to uppercase.

s.strip()

Returns a copy of the string with all leading and trailing white space characters removed.

s.strip(char)

Returns a copy of the string with all instances of *char* that appear at the beginning and the end of the string removed.

s.split(str)

Returns a list of all the words in the string, using str as the separator (splits on all whitespace if left unspecified).

s.splitlines([keepends])

Return a list of the lines in the string, breaking at line boundaries. This method uses the universal newlines approach to splitting lines. Line breaks are not included in the resulting list unless keepends is given and true.

Python recognizes "\r", "\n", and "\r\n" as line boundaries for 8-bit strings.

s.endswith(substring)

The substring argument is a string. The method returns true if the string ends with substring.

s.startswith(substring)

The substring argument is a string. The method returns true if the string starts with substring.

s.find(substring)

The substring argument is a string. The method returns the lowest index in the string where substring is found. If substring is not found the method returns -1.

s.replace(old, new)

The old and new arguments are both strings. The method returns a copy of the string with all instances of old replaced by new.

str.format(*args, **kwargs)

Perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces {}. Each replacement field contains either the numeric index of a positional argument, or the name of a keyword argument. Returns a copy of the string where each replacement field is replaced with the string value of the corresponding argument.

List operations:

s[i:j:k]

Return slice starting at position i extending to position j in k steps. Can also be used for strings.

item in s

Determine whether a specified item is contained in a list.

min(list)

Returns the item that has the lowest value in the sequence.

max(list)

Returns the item that has the highest value in the sequence.

s.append(x)

Append new element x to end of s. Works in_place. Returns None

s.insert(index,item)

Insert an item into a list at a specified position given by an index.

s.index(item)

Return the index of the first element in the list containing the specified item.

s.pop()

Return last element and remove it from the list.

s.pop(i)

Return element i and remove it from the list.

s.remove(item)

Removes the first element containing the item. Works in_place. Returns None

s.reverse()

Reverses the order of the items in a list. Works in_place. Returns None

s.sort()

Rearranges the elements of a list so they appear in ascending order. Works in_place.
Returns None

Sets operations:**len(s)**

Number of elements in set *s*

s.issubset(t)

Test whether every element in *s* is in *t*

s.issuperset(t)

Test whether every element in *t* is in *s*

s.union(t)

New set with elements from both *s* and *t*

s.intersection(t)

New set with elements common to *s* and *t*

s.difference(t)

New set with elements in *s* but not in *t*

s.symmetric_difference(t)

New set with elements in either *s* or *t* but not both

s.copy()

New set with a shallow copy of *s*

s.update(t)

Return set *s* with elements added from *t*

s.add(x)

Add element *x* to set *s*. Works in_place. Returns None

s.remove(x)

Remove *x* from set *s*; raises *KeyError* if not present. Works in_place. Returns None

s.clear()

Remove all elements from set *s*. Works in_place. Returns None

Dictionary operations:**d.clear()**

Clears the contents of a dictionary

d.get(key, default)

Gets the value associated with a specific key. If the key is not found, the method does not raise an exception. Instead, it returns a default value.

d.items()

Returns all the keys in a dictionary and their associated values as a sequence of tuples.

d.keys()

Returns all the keys in a dictionary as a sequence of tuples.

d.pop(key, default)

Returns the value associated with a specific key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.

d.popitem()

Returns a randomly selected key-value pair as a tuple from the dictionary and removes that key-value pair from the dictionary.

d.values()

Returns all the values in dictionary as a sequence of tuples.

del d[k]

Deletes element `k` in `d`.

d.copy()

Makes a copy of `d`.

Files:

open()

Returns a file object, and is most commonly used with two arguments:

`open(filename, mode)`. Mode can be 'r' (read only), 'w' (writing only), 'a' (appending), 'r+' (both reading and writing).

f.read(size)

Reads data from file and returns it as a string. Size is an optional and if left out the whole file will be read.

f.readline()

Reads a single line from the file (reads until newline character (`\n`) is found), and returns it as a string.

f.readlines()

Reads data from the file and returns it as a list of strings.

f.write(string)

Writes the contents of string to file.

f.writelines(list)

Writes the contents of a list to file

f.seek(offset, from_what)

Move the file pointer in the file and return the new position of the file pointer. The parameter

`from_what` can have three values: 0 – beginning of file, 1 – current position in file, and 2 – end of file. The offset parameter defines how much you will move from the `from_what` position.

f.tell()

Return the position of the file pointer.

f.close()

Close the file and free up any system resources taken up by the open file. If using **with open(filename) as ...** when you open the file, `close()` is not necessary, since it is implied by the ending of the with-block

Pickle Library:

pickle.dump(obj,file)

Write a pickled (serialized) representation of an obj to the open file object file.

pickle.load(file_object)

Read a string from the open file object file and interpret it as a pickle data stream, reconstructing and returning the original object hierarchy.

Random Library:

random.random()

Return the next random floating-point number in the range [0.0, 1.0).

random.randint(a,b)

Return a random integer N such that $a \leq N \leq b$.

random.choice(seq)

Return a random element from the non-empty sequence seq. If seq is empty, raises `IndexError`.

random.randrange(start, stop [, step])

Return a randomly selected element from range(start, stop, step).

random.uniform(a, b)

Return a random floating-point number N such that $a \leq N \leq b$ for $a \leq b$ and $b \leq N \leq a$ for $b < a$.

math Library:

math.ceil(x)

Return the ceiling of x, the smallest integer greater than or equal to x.

math.floor(x)

Return the floor of x, the largest integer less than or equal to x.

math.exp(x)

Return e raised to the power x, where $e = 2.718281\dots$ is the base of natural logarithms.

math.cos(x)

Return the cosine of x radians.

math.sin(x)

Return the sine of x radians.

math.tan(x)

Return the tangent of x radians.

math.pi

The mathematical constant $\pi = 3.141592\dots$, to available precision.

math.e

The mathematical constant $e = 2.718281\dots$, to available precision.

numpy Library:

numpy.array(list)

Generates an array. If the list or tuple is a 2D-list it generates a matrix. With a 1D-list it generates a vector

ndarray.ndim

the number of axes (dimensions) of the array.

ndarray.shape

the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with n rows and m columns, shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim.

ndarray.size

the total number of elements of the array. This is equal to the product of the elements of shape.

ndarray.dtype

an object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. Additionally, NumPy provides types of its own. `numpy.int32`, `numpy.int16`, and `numpy.float64` are some examples.

indexing and slicing in arrays

for 1d arrays (vectors), indexing and slicing works the same way as for lists. For multi-dimensional arrays, there is a more compact index notation for arrays, e.g , if A is an array, `A[i, j]` is synonymous to `A[i][j]`. This more compact notation also allows more powerful slicing, for instance `A[:,j]` making a slice containing just column j of A (since colon `:` for the first index indicates all rows)

numpy.zeros(tuple)

creates a matrix with 0 using the tuple to define the number of lines and rows in the matrix

numpy.ones(tuple)

creates a matrix with 1 using the tuple to define the number of lines and rows in the matrix

numpy.abs(x)

returns the absolute value of *x*. Unlike the `abs()` function in the standard library, which only works on single values, `numpy.abs()` can process an entire array, returning an array with the corresponding absolute values

numpy.sum(x)

calculates the sum of the values in the array *x*

numpy.prod(x)

calculates the product of the values in the array *x*

numpy.sin(x), numpy.cos(x), numpy.sqrt(x)

calculates the sinus, cosinus and the square root, similar to the same functions in the math library. However, while the math functions work just on single values, the numpy functions can also work on entire arrays, then returning arrays of the corresponding sine, cosine or square root values.

numpy.arange(start,stop,step)

creates a vector starting at *start*, ending at (but not including) *stop* using *step* between the values

numpy.linspace(start, stop, num)

creates a vector starting at *start*, ending at (and including) *stop*, and with *num* evenly spaced values in total

numpy.ndarray.reshape(lines,rows)

reshapes an ndarray according to the values in *lines* and *rows*. The reshape method does not change the original array, instead returning a reshaped copy

numpy.ndarray.resize(lines,rows)

reshapes an ndarray (similar to reshape method above), but `resize` changes the original array in place, rather than making a copy. Hence, this method returns `None`.

numpy.ndarray.tolist()

returns a list (of lists) with the same dimensions and content as the numpy array. Unlike `list(A)` which will only convert the first dimension of a multi-dimensional array to a list, `A.tolist()` makes a deep conversion, making all dimensions into lists

numpy.loadtxt(filename)

Loads data from text file, returning them as a numpy array.

numpy.savetxt(filename, A)

Saves array *A* to text file.

matplotlib.pyplot:

subplots()

Creates a *figure* containing a single *axes* object for plotting.

plot(x, y)

Plots a graph of *y*-values against *x*-values

bar(x, height)

Plots bars in positions given by vector *x*, with heights given by vector *height*

show()

Shows the plot on the screen

1 Kopi av Operatorer og typer (4%)

Anta at vi har definert tre variable som følger:

`i = 4`

`x = 3.0`

`s = 'Python'`

`t = 'thon'`

I tabellen nedenfor står en del uttrykk, hvorav noen vil kunne beregnes og få en verdi, mens andre vil gi feilmelding.

For hver rad, se på uttrykket til venstre, og kryss av hvilken datatype resultatet vil bli - eller velg ERROR dersom uttrykket vil gi feilmelding. Ingen minuspoeng for feil svar, så du BØR svare noe også der du er usikker.

	int	float	ERROR	bool	string
<code>s[x]</code>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<code>i == x</code>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<code>s - t</code>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<code>i / i</code>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<code>i * x</code>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<code>7 // i</code>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<code>s + t</code>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<code>i % 0</code>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Maks poeng: 5.6

2 Kopi av Exact (3%)

Hvilke av disse tallene kan representeres eksakt som enten heltall eller flyttall i Python?

For hvert tall, sett kryss i rett kolonne. Ingen minuspoeng for feil, så du **BØR** svare noe også der du er usikker.

	IKKE eksakt	Eksakt
1.67	<input type="radio"/>	<input type="radio"/>
834257087666	<input type="radio"/>	<input type="radio"/>
1.5e-7	<input type="radio"/>	<input type="radio"/>
-3.75	<input type="radio"/>	<input type="radio"/>
0.8	<input type="radio"/>	<input type="radio"/>
-55	<input type="radio"/>	<input type="radio"/>

Maks poeng: 6

3 Kopi av Fix (5%)

Koden for funksjonen **fix()** er gitt som følger, sammen med fem print-setninger hvor **fix()** kalles (**NB:** Merk #kommentarnummer til høyre for print-setningene, viktig for å svare på riktig spørsmål i tabellen under)

```
def fix(strg, cond):
    new_strg = strg[0]
    for i in range(1, len(strg)):
        if (strg[i] > new_strg[-1]) == cond:
            new_strg += strg[i]
        elif strg[i] == new_strg[-1]:
            new_strg += strg[i]
    return new_strg
```

```
print(fix('riiga', True))    #1
print(fix('rigra', True))    #2
print(fix('agir', True))     #3
print(fix('rigar', False))   #4
print(fix('agir', False))    #5
```

For hver rad i tabellen, kryss av i den kolonna som viser hva som vil bli printa av den korresponderende print-setninga. Merk at radene refererer til kommentarnummer over, og kan ha blitt stokket om i forhold til nummerrekkefølge.

Finn de som passer sammen.

	rr	r	rigar	iiga	riiga	riga	a	agir
#4	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
#2	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
#1	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
#3	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
#5	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Maks poeng: 5

4 Kopi av files and exceptions (5%)

Figuren under viser koden for funksjonen **respons()**, samt definisjon av strengvariable **d** og **t**. Anta at tekstfila **datafil.txt** har innhold som vist i den innfelte boksen nede til høyre, og at dette er *den eneste* .txt-fila på samme katalog som programkoden vår.

```
def respons(fil, r, c):
    try:
        with open(fil) as f:
            line = f.readlines()[r].split(';')
            if c == 0:
                return line[3].strip()[int(line[c])]
            else:
                return float(line[c])
    except FileNotFoundError:
        return -1
    except:
        return -2

d = 'datafil.txt'
t = 'tallfil.txt'
```

```
3 ; 2.5 ; 3.5 ; Alta
1 ; 0.25 ; 1.0 ; Tana
2.0 ; 1; 1.0; Hamar
5 ; 7; 3; 9; Oslo
```

I tabellen viser hver rad ett kall av funksjonen og kolonnene viser mulige returverdier. Alle rader skal brukes, men ikke nødvendigvis alle kolonner.

For hver rad, sett kryss i kolonnen som viser riktig returverdi for kallet av funksjonen. Ingen minuspoeng for feil så du BØR svare noe også der du er usikker.

	-1	7.0	-2	5.0	'o'	'a'	't'	1.0
respons(d, 3, 0)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
respons(d, 0, 0)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
respons(d, 3, 1)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
respons(t, 0, 0)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
respons(d, 1, 2)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Maks poeng: 5

5 Kopi av Avvik, presisjon (5%)

Koden i den grå blokken definerer andregradsfunksjonen $f(x) = \frac{1}{3}x^3 - x$, samt en funksjon `sentral_deriv` som regner ut den sentralderiverte av f for x med intervallstørrelse h .

```
In [14]: 1 import numpy as np
          2
          3 def f(x):
          4     return (1/3)*x**3 - x
          5
          6 def sentral_deriv(f, x, h):
          7     # Regner ut f'(x) med intervallstørrelse h
          8     return (f(x+h) - f(x-h)) / (2*h)
```

```
In [15]: 1 sentral_deriv(f, 0.5, 1)
```

```
Out[15]: -0.41666666666666663
```

```
In [16]: 1 sentral_deriv(f, 0.5, 0.1)
```

```
Out[16]: -0.7466666666666666
```

```
In [21]: 1 sentral_deriv(f, 0.5, 1e-7)
```

```
Out[21]: -0.7499999998827889
```

```
In [23]: 1 sentral_deriv(f, 0.5, 1e-14)
```

```
Out[23]: -0.7494005416219807
```

```
In [24]: 1 sentral_deriv(f, 0.5, 1e-20)
```

```
Out[24]: 0.0
```

```
In [25]: 1 f'{0.5 + 1e-20:.30f}'
```

```
Out[25]: '0.500000000000000000000000000000'
```

Analytisk vet vi at det korrekte svaret på derivasjonen $f'(0.5)$ skal være **-0.75**. I de cellene 15, 16, 21, 23, 24 i bildet over blir funksjonen `sentral_deriv(f, x, h)` prøvd ut med $x=0.5$ og gradvis mindre verdier for h . Som vi kan se, er resultatet fra [15] langt unna riktig, mens de tre neste er ganske nærme korrekt svar. Den aller siste cella [25] regner ikke ut den deriverte, dette er en kontrollregning for å se hva $x+h$ blir med $h = 1e-20$. Spørsmålene nedenfor handler om hva slags slutninger vi kan trekke av koden og resultatene vi ser i bildet over.

For hver påstand nedenfor, kryss av om påstanden er riktig eller feil vedrørende koden over. Ingen minuspoeng for ukorrekt svar, så du **BØR** svare noe også der du er usikker.

	RIKTIG	FEIL
Vi får konsekvent mer nøyaktige resultat jo mindre intervallstørrelse h vi bruker i derivasjonen.	<input type="radio"/>	<input type="radio"/>
[23] gir mindre nøyaktig svar enn [21] på grunn av underflyt.	<input type="radio"/>	<input type="radio"/>
Celle [21] gir mer nøyaktig svar enn [16] som er mer nøyaktig enn [15] fordi mindre h gir mer nøyaktig resultat.	<input type="radio"/>	<input type="radio"/>
[23] gir mindre nøyaktig svar enn [21] på grunn av overflyt.	<input type="radio"/>	<input type="radio"/>
Celle [25] viser at liten h gir et ekstremt nøyaktig svar, siden vi får 30 nuller på rad	<input type="radio"/>	<input type="radio"/>
Når det blir for stor forskjell i størrelsesorden mellom x og h , vil det minste av disse to tallene bare forsvinne i addisjonen $x+h$	<input type="radio"/>	<input type="radio"/>

Maks poeng: 10

Useful Functions and Methods

These are listed in the following order:

- built-in (standard library)
 - often used functions and operators
 - exceptions
 - string methods
 - list operations
 - set operations
 - dictionary operations
 - files
 - pickle library (binary files)
- random library
- math library
- numpy library
- matplotlib.pyplot

Built-in:

f-string

Syntax: `f'.....{expression}...'` where expression can be variable or any expression. Can also use formatting characters such as : d=integer, f=floating-point, e=scientific notation, %=percentage, s=string. A number before the formatting character will specify field width. A number after the character "." will format the number of decimals. E.g. print PI with a field width of 5 with two decimals will be:
`print(f'{math.pi:5.2f}')`

%

Remainder (modulo operator): Divides one number by another and gives the remainder.

//

Floor/integer division: Returns the integral part of the quotient.

len(s)

Return the length (the number of items) of a string, tuple, list, dictionary or other data structure.

int(x)

Convert a string or number to a plain integer.

float(x)

Convert a string or a number to floating point number.

str([object])

Return a string containing a nicely printable representation of an object.

range(stop)

Returns a list with integers from 0, up to but not including stop. `range(3) = [0, 1, 2]`. Often used in combination with for loops: `for i in range(10)`

range([start], stop[, step])

start: Starting number of the sequence.

stop: Generate numbers up to, but not including, this number.

step: Difference between each number in the sequence.

chr(i)

Return a string of one character whose ASCII code is the integer i. For example, `chr(97)` returns the string 'a'. This is the inverse of `ord()`

ord()

Given a string of length one, return an integer representing the Unicode code point of the character when the argument is a unicode object, or the value of the byte when the argument is an 8-bit string. For example, `ord('a')` returns the integer 97.

tuple(iterable)

Accepts something iterable (list, range etc.) and returns the corresponding tuple. A tuple is immutable.

if x in iterable:

Returns True if x is an item in iterable.

for idx, x in enumerate(iterable)

Enters a loop with x being one and one item from iterable. idx is an integer containing the loop number.

Exceptions:

try:

Code to test

except:

If code fails. E.g exception types: IOError, ValueError, ZeroDivisionError.

Variant: except Exception as exc # Let's you print the exception.

else:

Runs if no exception occurs

finally:

Runs regardless of prior code having succeeded or failed.

String methods:

s.isalnum()

Returns true if the string contains only alphabetic letters or digits and is at least one character of length. Returns false otherwise.

s.isalpha()

Returns true if the string contains only alphabetic letters, and is at least one character in length. Returns false otherwise.

s.isdigit()

Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.

s.center(width)

Return the string center justified in a string of length width.

s.ljust(width)

Return the string left justified in a string of length width.

s.rjust(width)

Return the string right justified in a string of length width.

s.lower()

Returns a copy of the string with all alphabetic letters converted to lowercase.

s.upper()

Returns a copy of the string with all alphabetic letters converted to uppercase.

s.strip()

Returns a copy of the string with all leading and trailing white space characters removed.

s.strip(char)

Returns a copy of the string with all instances of *char* that appear at the beginning and the end of the string removed.

s.split(str)

Returns a list of all the words in the string, using str as the separator (splits on all whitespace if left unspecified).

s.splitlines([keepends])

Return a list of the lines in the string, breaking at line boundaries. This method uses the universal newlines approach to splitting lines. Line breaks are not included in the resulting list unless keepends is given and true.

Python recognizes "\r", "\n", and "\r\n" as line boundaries for 8-bit strings.

s.endswith(substring)

The substring argument is a string. The method returns true if the string ends with substring.

s.startswith(substring)

The substring argument is a string. The method returns true if the string starts with substring.

s.find(substring)

The substring argument is a string. The method returns the lowest index in the string where substring is found. If substring is not found the method returns -1.

s.replace(old, new)

The old and new arguments are both strings. The method returns a copy of the string with all instances of old replaced by new.

str.format(*args, **kwargs)

Perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces {}. Each replacement field contains either the numeric index of a positional argument, or the name of a keyword argument. Returns a copy of the string where each replacement field is replaced with the string value of the corresponding argument.

List operations:

s[i:j:k]

Return slice starting at position i extending to position j in k steps. Can also be used for strings.

item in s

Determine whether a specified item is contained in a list.

min(list)

Returns the item that has the lowest value in the sequence.

max(list)

Returns the item that has the highest value in the sequence.

s.append(x)

Append new element x to end of s. Works in_place. Returns None

s.insert(index,item)

Insert an item into a list at a specified position given by an index.

s.index(item)

Return the index of the first element in the list containing the specified item.

s.pop()

Return last element and remove it from the list.

s.pop(i)

Return element i and remove it from the list.

s.remove(item)

Removes the first element containing the item. Works in_place. Returns None

s.reverse()

Reverses the order of the items in a list. Works in_place. Returns None

s.sort()

Rearranges the elements of a list so they appear in ascending order. Works in_place.
Returns None

Sets operations:**len(s)**

Number of elements in set *s*

s.issubset(t)

Test whether every element in *s* is in *t*

s.issuperset(t)

Test whether every element in *t* is in *s*

s.union(t)

New set with elements from both *s* and *t*

s.intersection(t)

New set with elements common to *s* and *t*

s.difference(t)

New set with elements in *s* but not in *t*

s.symmetric_difference(t)

New set with elements in either *s* or *t* but not both

s.copy()

New set with a shallow copy of *s*

s.update(t)

Return set *s* with elements added from *t*

s.add(x)

Add element *x* to set *s*. Works in_place. Returns None

s.remove(x)

Remove *x* from set *s*; raises *KeyError* if not present. Works in_place. Returns None

s.clear()

Remove all elements from set *s*. Works in_place. Returns None

Dictionary operations:**d.clear()**

Clears the contents of a dictionary

d.get(key, default)

Gets the value associated with a specific key. If the key is not found, the method does not raise an exception. Instead, it returns a default value.

d.items()

Returns all the keys in a dictionary and their associated values as a sequence of tuples.

d.keys()

Returns all the keys in a dictionary as a sequence of tuples.

d.pop(key, default)

Returns the value associated with a specific key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.

d.popitem()

Returns a randomly selected key-value pair as a tuple from the dictionary and removes that key-value pair from the dictionary.

d.values()

Returns all the values in dictionary as a sequence of tuples.

del d[k]

Deletes element `k` in `d`.

d.copy()

Makes a copy of `d`.

Files:

open()

Returns a file object, and is most commonly used with two arguments:

`open(filename, mode)`. Mode can be 'r' (read only), 'w' (writing only), 'a' (appending), 'r+' (both reading and writing).

f.read(size)

Reads data from file and returns it as a string. Size is an optional and if left out the whole file will be read.

f.readline()

Reads a single line from the file (reads until newline character (`\n`) is found), and returns it as a string.

f.readlines()

Reads data from the file and returns it as a list of strings.

f.write(string)

Writes the contents of string to file.

f.writelines(list)

Writes the contents of a list to file

f.seek(offset, from_what)

Move the file pointer in the file and return the new position of the file pointer. The parameter

`from_what` can have three values: 0 – beginning of file, 1 – current position in file, and 2 – end of file. The offset parameter defines how much you will move from the `from_what` position.

f.tell()

Return the position of the file pointer.

f.close()

Close the file and free up any system resources taken up by the open file. If using **with open(filename) as ...** when you open the file, `close()` is not necessary, since it is implied by the ending of the with-block

Pickle Library:

pickle.dump(obj,file)

Write a pickled (serialized) representation of an obj to the open file object file.

pickle.load(file_object)

Read a string from the open file object file and interpret it as a pickle data stream, reconstructing and returning the original object hierarchy.

Random Library:

random.random()

Return the next random floating-point number in the range [0.0, 1.0).

random.randint(a,b)

Return a random integer N such that $a \leq N \leq b$.

random.choice(seq)

Return a random element from the non-empty sequence seq. If seq is empty, raises `IndexError`.

random.randrange(start, stop [, step])

Return a randomly selected element from range(start, stop, step).

random.uniform(a, b)

Return a random floating-point number N such that $a \leq N \leq b$ for $a \leq b$ and $b \leq N \leq a$ for $b < a$.

math Library:

math.ceil(x)

Return the ceiling of x, the smallest integer greater than or equal to x.

math.floor(x)

Return the floor of x, the largest integer less than or equal to x.

math.exp(x)

Return e raised to the power x, where $e = 2.718281\dots$ is the base of natural logarithms.

math.cos(x)

Return the cosine of x radians.

math.sin(x)

Return the sine of x radians.

math.tan(x)

Return the tangent of x radians.

math.pi

The mathematical constant $\pi = 3.141592\dots$, to available precision.

math.e

The mathematical constant $e = 2.718281\dots$, to available precision.

numpy Library:

numpy.array(list)

Generates an array. If the list or tuple is a 2D-list it generates a matrix. With a 1D-list it generates a vector

ndarray.ndim

the number of axes (dimensions) of the array.

ndarray.shape

the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with n rows and m columns, shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim.

ndarray.size

the total number of elements of the array. This is equal to the product of the elements of shape.

ndarray.dtype

an object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. Additionally, NumPy provides types of its own. `numpy.int32`, `numpy.int16`, and `numpy.float64` are some examples.

indexing and slicing in arrays

for 1d arrays (vectors), indexing and slicing works the same way as for lists. For multi-dimensional arrays, there is a more compact index notation for arrays, e.g , if A is an array, `A[i, j]` is synonymous to `A[i][j]`. This more compact notation also allows more powerful slicing, for instance `A[:,j]` making a slice containing just column j of A (since colon `:` for the first index indicates all rows)

numpy.zeros(tuple)

creates a matrix with 0 using the tuple to define the number of lines and rows in the matrix

numpy.ones(tuple)

creates a matrix with 1 using the tuple to define the number of lines and rows in the matrix

numpy.abs(x)

returns the absolute value of *x*. Unlike the `abs()` function in the standard library, which only works on single values, `numpy.abs()` can process an entire array, returning an array with the corresponding absolute values

numpy.sin(x), numpy.cos(x), numpy.sqrt(x)

calculates the sinus, cosinus and the square root, similar to the same functions in the math library. However, while the math functions work just on single values, the numpy functions can also work on entire arrays, then returning arrays of the corresponding sine, cosine or square root values.

numpy.arange(start,stop,step)

creates a vector starting at *start*, ending at (but not including) *stop* using *step* between the values

numpy.linspace(start, stop, num)

creates a vector starting at *start*, ending at (and including) *stop*, and with *num* evenly spaced values in total

numpy.ndarray.reshape(lines,rows)

reshapes an ndarray according to the values in *lines* and *rows*. The reshape method does not change the original array, instead returning a reshaped copy

numpy.ndarray.resize(lines,rows)

reshapes an ndarray (similar to reshape method above), but `resize` changes the original array in place, rather than making a copy. Hence, this method returns `None`.

numpy.ndarray.tolist()

returns a list (of lists) with the same dimensions and content as the numpy array. Unlike `list(A)` which will only convert the first dimension of a multi-dimensional array to a list, `A.tolist()` makes a deep conversion, making all dimensions into lists

numpy.loadtxt(filename)

Loads data from text file, returning them as a numpy array.

numpy.savetxt(filename, A)

Saves array *A* to text file.

matplotlib.pyplot:

subplots()

Creates a *figure* containing a single *axes* object for plotting.

plot(x, y)

Plots a graph of *y*-values against *x*-values

bar(x, height)

Plots bars in positions given by vector *x*, with heights given by vector *height*

show()

Shows the plot on the screen

6 Kopi av Counting words (6%)

Vi ønsker en funksjon **count_start_end_words(word_list, letter)**

Funksjonen har to parametre, **word_list** som er ei liste av tekststrenger, og **letter**, som er en enkelt bokstav. Funksjonen skal returnere antall strenger i **word_list** som er slik at de både begynner og slutter på bokstaven **letter**.

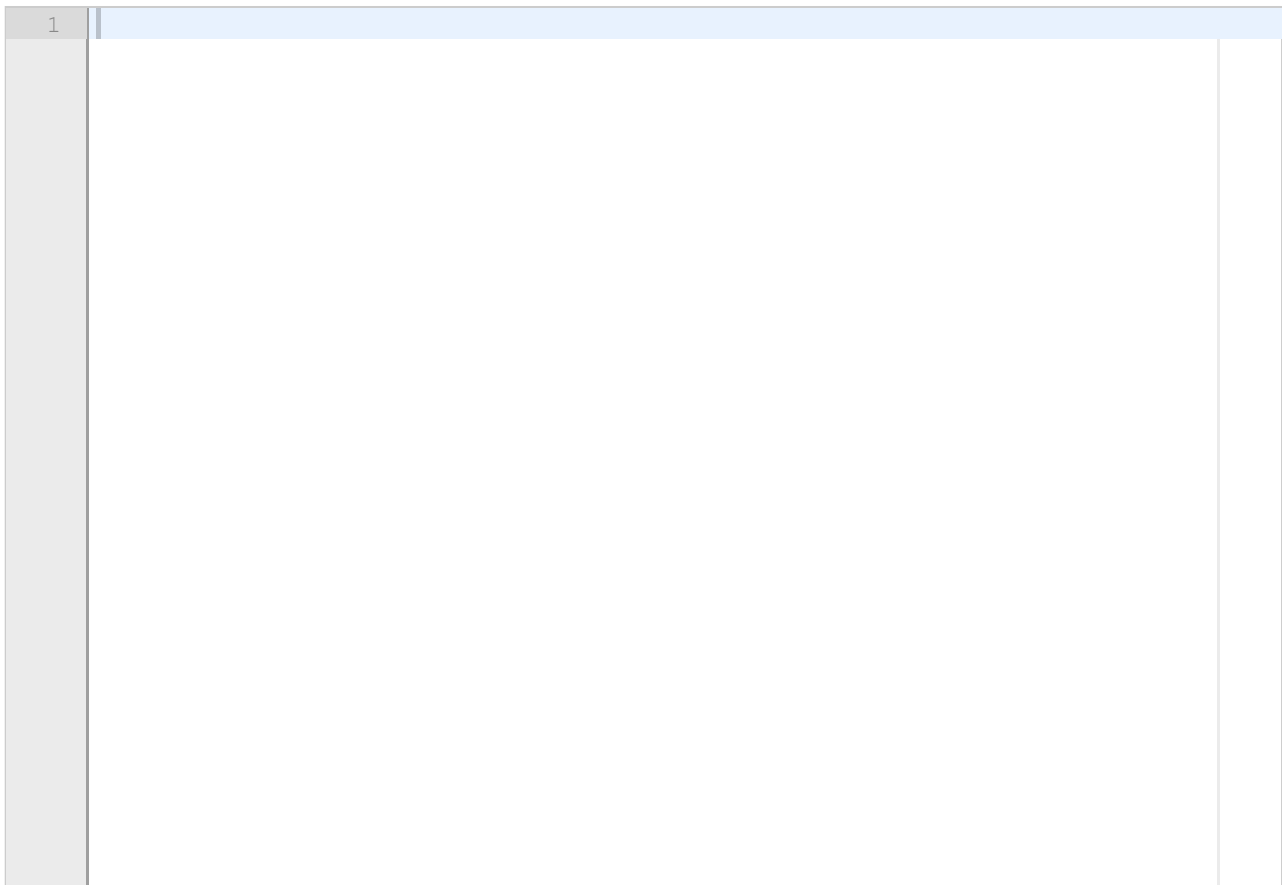
Eksempel på ønsket virkemåte av funksjonen, hvis vi har

word_list = ["ada", "ida", "alta", "ana", "y"]

så skal

- **count_start_end_words(word_list, "a")** returnere 3, fordi 3 strenger, "ada", "alta", "ana" har "a" i begge ender.
- **count_start_end_words(wordlist, "i")** returnere 0 (ingen streng har "i" i begge ender)
- **count_start_end_words(wordlist, "y")** returnere 1 (dvs. en streng på bare 1 bokstav skal telle med, gitt at bokstaven er riktig)

Skriv koden for funksjonen count_start_end_words()



Maks poeng: 6

7 Kopi av Taxfree (7%)

Følgende kvoter gjelder for hvor mye alkoholholdig drikke en person tollfritt kan innføre til Norge:

- 1 liter brennevin
- 1,5 liter vin
- 2 liter øl

Brennevinskvoten kan byttes i 1,5 liter vin eller øl (men det omvendte er ikke mulig, dvs. man får ikke ta med mer brennevin om man dropper vin eller øl), dvs. 3 liter vin + 2 liter øl vil også være innenfor. Likeledes kan vinkvoten byttes samme volum øl (men ikke omvendt), så kun 5 liter øl vil også være innenfor. Videre gjelder at man må være minst 18 år gammel for å ha noen kvote overhodet, og minst 20 år gammel for å ha med brennevin.

Vi ønsker en funksjon **legal_alcohol(age, booze, wine, beer)** som har fire parametre:

- age - et heltall for personens alder i antall hele år
- booze, wine, beer - flyttall som viser antall liter av henholdsvis brennevin, vin og øl. Alle tall som gis inn kan antas å være ≥ 0 .

Funksjonen skal returnere **True** hvis man er innenfor kvoten, **False** hvis ikke. Noen eksempler:

- legal_alcohol(17, 0.0, 0.0, 3.6) -> False (personen var for ung, kunne ikke ha med øl)
- legal_alcohol(19, 1.0, 0.0, 0.0) -> False (for ung til å ha med brennevin)
- legal_alcohol(19, 0.0, 3.0, 2.0) -> True (alder 19 er ok her, bare vin og øl)
- legal_alcohol(20, 1.0, 1.5, 2.0) -> True (akkurat innenfor på alt)
- legal_alcohol(20, 1.5, 0.0, 0.0) -> False (for mye brennevin)
- legal_alcohol(20, 0.5, 3.0, 0.0) -> False (mer enn 1.5 vin kun ok helt UTEN brennevin)
- legal_alcohol(20, 0.0, 0.0, 5.0) -> True (både brennevin og vin byttet til øl)
- legal_alcohol(20, 0.5, 1.5, 4.0) -> False (1.5 vin + 4 øl er for mye til sammen)

Skriv koden for funksjonen legal_alcohol()

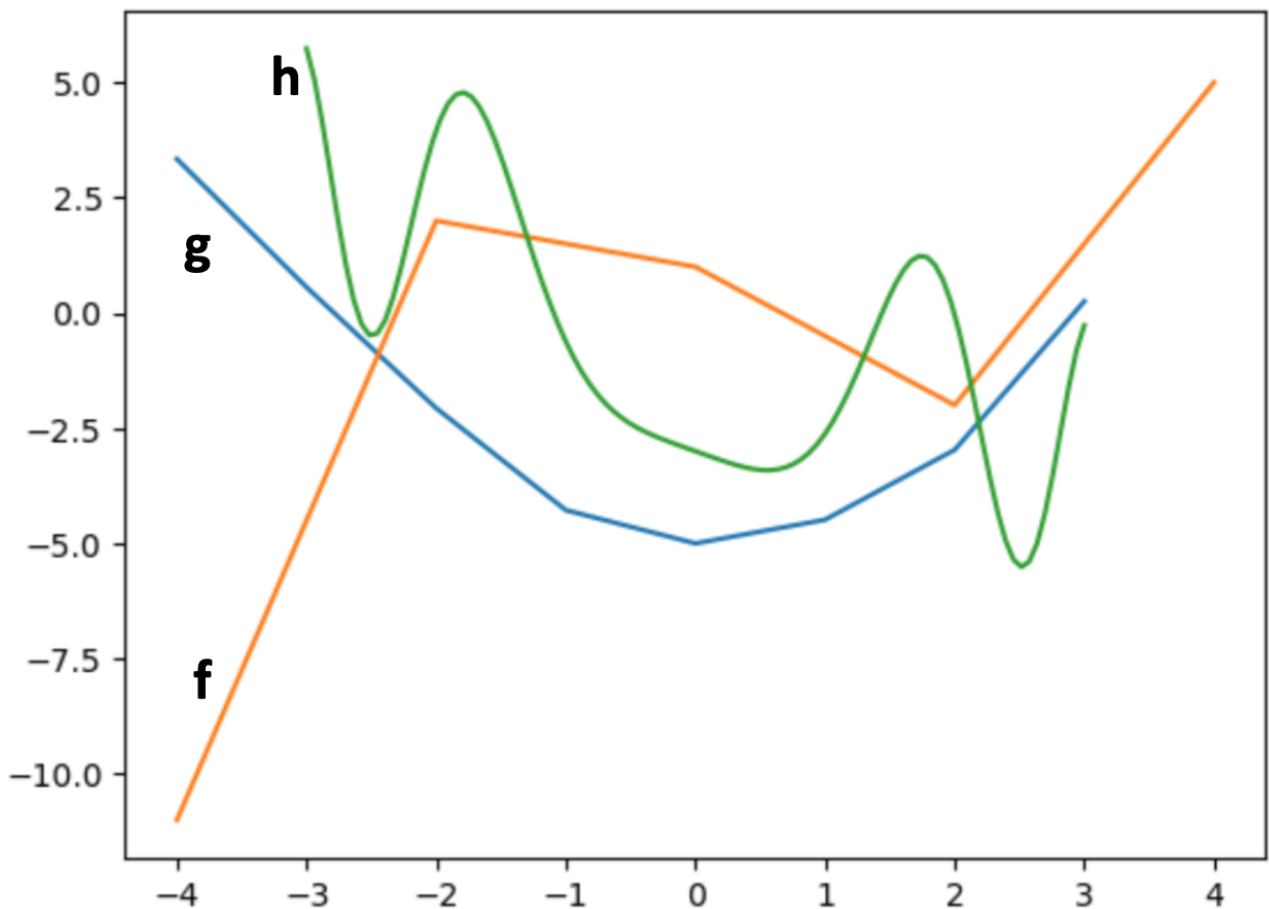
1	
---	--



Maks poeng: 7

8 Kopi av Plot (5%)

Koden vår skal plote de tre grafene vist under i samme skjermbilde. (NB: utskrift av de tre bokstavene f, g, h er ikke med i koden, disse er tatt med i bildet bare for å vise tydelig hva som er hvilken funksjon)



OPPGAVE: Trekk kodelinjer til riktig plass så programmet virker som det skal. Ingen minuspoeng for feil svar, så du **BØR** svare noe også der du er usikker. Tre av kodelinjene skal ikke brukes.

 [Hjelp](#)

`x = np.arange(-4, 4)`

`np.linspace(-4, 3, 3)`

`as plt`

`x = np.linspace(-4,4,5)`

`x = np.linspace(-3,3,100)`

`import numpy as np`

`plt.show()`

`ax.plot(x, h(x))`

`np.arange(-4.0, 4.1, 0.1)`

`plt.subplots()`

`import numpy`

`import matplotlib.pyplot`

`def f(x):`

`return 0.25*x**3 - 0.25*x**2 - 2*x + 1`

`def g(x):`

`return 3*x**2/(np.sin(x)+5) - 5`

```
def h(x):  
    return -3*np.cos(x**2)-x
```

```
fig, ax =
```

```
ax.plot(x, g(x))
```

```
ax.plot(x, f(x))
```

Maks poeng: 5

9 Kopi av Lottery (4%)

I et lotteri hvor brukerne skraper fram kombinasjoner av bokstaver, er reglene slik:

- 1.premie hvis det er 3 eller flere A'er: 100
- 2.premie hvis det er 3 eller flere B'er: 60
- 3.premie hvis det er 3 eller flere C'er: 20
- Ingen premie hvis det ikke er 3 eller flere av noen bokstav, eller hvis det bare er 3 eller flere av andre bokstaver enn A, B, C.

Man får kun én premie (dvs. hvis det f.eks. er 6 A'er, får man likevel bare 100), og kun den beste mulige premien (hvis det er både 3 A'er og 3 B'er, får man 100, ikke 60).

Eksempel på ønsket resultat ved kjøring av funksjonen:

- `calc_prize('ABABBCCCCA')` skal returnere 100
- `calc_prize('ABACCBCCCCC')` skal returnere 20
- `calc_prize('DDDXABDDDDXX')` skal returnere 0

Fyll inn det som mangler slik at funksjonen virker som den skal. Ingen minuspoeng for feil, så du BØR svare noe også der du er usikker. Merk at det allerede står apostrofer inni parentesene i kallene av funksjonen `count()` og i `[]`-parentesene for prizes, slik at du selv IKKE skal skrive apostrofer der.

```
def calc_prize(string):
    prizes = {'A': 100, 'B': 60, 'C': 20}
    reward = 
    if string.count('  ')  3:
        reward = prizes['  ']
    elif string.count('B')  3:
        reward = prizes['B']
     string.count('  ')  3:
        reward = prizes['  ']
    return 
```

Maks poeng: 4

10 Kopi av Names (5%)

Gitt lister av navn, f.eks.

```
navn = ['Logan Paul', 'Ada Wong', 'Andy Warhol', 'Paul Anka',
        'Grace Hopper', 'Ada Lovelace', 'Kalle Anka', 'Andy Anka']
```

For enkelhets skyld antar vi at alle navn består av bare to ord, fornavn og etternavn, uten noe mellomnavn eller andre tillegg. Vi ønsker å lage to funksjoner:

- **name_sets(name_list)** skal få inn ei liste av navn som f.eks. ovenstående, og returnere tre mengder (set), nemlig ei mengde som inneholder fornavnene, ei med etternavnene, og ei med initialene som fantes i lista.
- **dual_use_names(first_names, last_names)** skal returnere mengden av navn som er i bruk både som fornavn og etternavn, dvs. som forekommer i begge mengdene som gis inn som parametre

Eksempel på kode for å kalle funksjonene:

```
first, last, inits = name_sets(navn)
print(first)
print(last)
print(inits)
print(dual_use_names(first, last))
```

Ønsket resultat av kjøring, hvis variabelen navn har innhold som gitt øverst:

```
{'Andy', 'Logan', 'Ada', 'Kalle', 'Grace', 'Paul'}
{'Lovelace', 'Wong', 'Paul', 'Hopper', 'Anka', 'Warhol'}
{'GH', 'LP', 'KA', 'AL', 'AA', 'AW', 'PA'}
{'Paul'}
```

OPPGAVE: Trekk kodelinjer til riktig plass så de to funksjonene virker som de skal. To av kodefragmentene skal ikke brukes. Ingen minuspoeng for feil svar, så du **BØR** svare noe også der du er usikker.

 [Hjelp](#)

name.split()	intersection(last_names)	names[1][0]
union(last_names)	names[0]	set(), set(), set()
first_names	names[0][0]	names[1]
fi_n + la_n		

```
def name_sets(name_list):
    fi_n, la_n, init = 
    for name in name_list:
        names = 
        fi_n.add( )
        la_n.add( )
        init.add( + )
    return fi_n, la_n, init

def dual_use_names(first_names, last_names):
    return .
```

Maks poeng: 5

11 Kopi av convert file (5%)

Vi har tekstfiler med flyttall separert med mellomrom. Hver rad i fila kan antas å ha like mange elementer. Et eksempel på mulig innhold kan være:

```
3.22242 4.50020 5.67230 6.33232
5.04001 6.66666 4.48341 7.93939
3.00001 5.72992 4.40001 0.77077
```

Vi ønsker en funksjon **convert_file(filename)** som får navnet på ei slik fil inn som parameter, og som kan skrive ei ny fil med tilsvarende data, men hvor hvert tall har blitt multiplisert med **1.25** (dvs. økt 25%) og hvert tall skal også være **avrundet til to desimaler**. Navnet på den nye fila skal tilsvare navnet på den gamle fila, men med tillegg av **_conv** bakerst i filnavnet (men foran filtypen). Eksempel: Hvis fila vi fikk inn, het **data.txt** skal fila som skrives av funksjonen hete **data_conv.txt** og hvis data.txt hadde innhold som vist over, ville kjøring av funksjonen **convert_file** på denne fila gjøre at data_conv.txt ble skrevet med følgende innhold:

```
4.03 5.63 7.09 7.92
6.30 8.33 5.60 9.92
3.75 7.16 5.50 0.96
```

OPPGAVE: Trekk kodelinjer til riktig sted så funksjonen edit_file() virker som den skal. 5 av kodelinjene skal ikke brukes. Ingen minuspoeng for feil svar, så du BØR svare noe også der du er usikker.

 [Hjelp](#)

names[0] + '_conv.' + names[1]

a = np.savetxt(filename)

return a

new_name = filename + '_conv'

np.loadtxt(a, filename)

b.round(2)

np.savetxt(new_name, b)

a = np.loadtxt(filename)

b = a * 1.25

filename.split('.')

a = []

```
import numpy as np
```

```
def convert_file(filename):
```

```
    
```

```
    
```

```
    
```

```
    names = 
```

```
    
```

new_name =

Maks poeng: 5

12 Kopi av dictionary, mutering (5%)

Vi ønsker å justere en ordbok (dictionary) hvor verdiene er lister av heltall slik at alle partall fjernes, mens oddetall blir værende igjen. Hvis ei liste som følge av fjerning av partall blir tom, skal hele den tilhørende nøkkelen fjernes fra ordboka. Vi ønsker to funksjoner for dette:

- **remove_even_nos(d)** som muterer ordboka (dictionary) og listene inni
- **even_nos_removed(d)**, som returnerer ny versjon av ordboka og listene inni, uten å endre originaldataene.

For den muterende **remove_even_nos** skal endringene skje i samme ordbok og mutere listene. For **even_nos_removed** skal det lages en ny ordbok med nye lister som verdier.

Eksempel på kjøring av funksjonene:

```
d = {'A': [3, 2, 4, 5], 'B': [5, 4, 1, 1, 8, 3], 'C': [1, 7, 7], 'D': [2, 4, 6]}
print(even_nos_removed(d)) # viser justert ordbok, endrer ikke original
print(d)                  # printer original ordbok
remove_even_nos(d)        # endrer original ordbok ved mutering
print(d)                  # printer original ordbok på nytt
```

Gjør at følgende printes:

```
{'A': [3,5], 'B': [5,1,1,3], 'C': [1, 7, 7]}
{'A': [3,2,4,5], 'B': [5,4,1,1,8,3], 'C': [1, 7, 7], 'D': [2, 4, 6]}
{'A': [3,5], 'B': [5,1,1,3], 'C': [1, 7, 7]}
```

Som vi ser forsvinner nøkkelen 'D' helt fra ordboka ved justering i dette eksemplet fordi alle tallene i lista til den nøkkelen var partall, slik at lista endte opp med å bli tom.

OPPGAVE: Trekk kodelinjer til rett posisjon slik at de to funksjonene virker som de skal. Ingen minuspoeng for feil valg, så du BØR svare noe også der du er usikker.

 [Hjelp](#)

<code>if len(d[key]) == 0:</code>	<code>return new_d</code>
<code>new_d = { }</code>	<code>new_val.append(num)</code>
<code>for num in d[key]:</code>	<code>if num % 2 != 0:</code>
<code>if d[key][i] % 2 == 0:</code>	<code>for i in range(len(d[key])-1,-1,-1):</code>
<code>del d[key]</code>	<code>new_val = []</code>
<code>new_d[key] = new_val</code>	<code>if len(new_val) > 0:</code>
<code>del d[key][i]</code>	

```
def remove_even_nos(d): # Mutating the dictionary
    for key in list(d): #list(d) avoids iterating d itself while changing
```



```
def even_nos_removed(d): # not changing the original
```

```
for key in d:
```

Maks poeng: 6.5

13 Kopi av Diagonal (6%)

Funksjonen **diagonal(a)** får inn et todimensjonalt numpy array (matrise) og skal returnere et endimensjonalt array (vektor) med samme innhold, men hvor rekkefølgen på elementene i vektoren fremkommer ved å iterere det todimensjonale arrayet diagonalt.

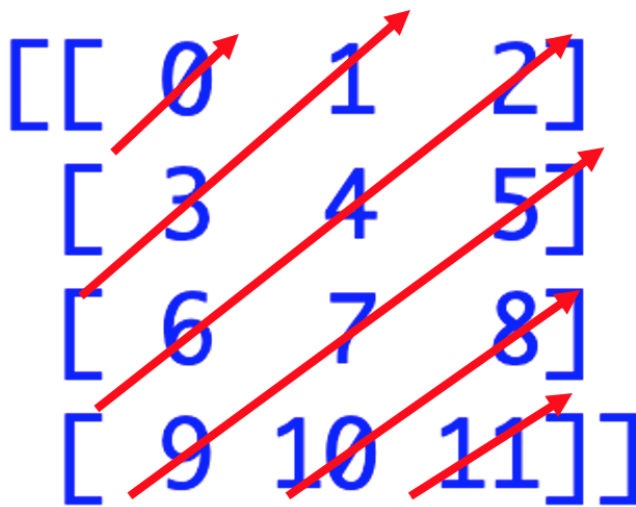
Eksempel på kjøring:

Hvis **a = numpy.array([[0,1,2], [3,4,5], [6,7,8], [9,10,11]])**

skal kallet **print(diagonal(a))** resultere i utskrift

[0 3 1 6 4 2 9 7 5 10 8 11]

Dvs., den diagonale iterasjonen av matrisa skal starte i øvre venstre hjørne, og så ta en og en stigende diagonal helt til den ender i nedre høyre hjørne, jfr. figur:



OPPGAVE: Fyll inn det som mangler i funksjonen så den virker som den skal. Vær nøye med små og store bokstaver, siden det er semantisk signifikant i Python.

```
import numpy as np
```

```
def diagonal(a):
```

```
    result = 
```

```
    n_rows, n_cols = a.shape
```

```
    n_diags = n_rows + n_cols 
```

```
    for d in range(n_diags):
```

```
        if d < n_rows:
```

```
            i = 
```

```
            j = 
```

```
        else:
```

```
            i = 
```

```
j =  - i
while  and j < n_cols:
    result.append(a[i,j])
    i  1
    j  1
return np.array(result)
```

Maks poeng: 6

14 Kopi av Dice game (6%)

Terningspillet Titusen spilles med seks terninger og har (litt forenklet) følgende poengregler:

- Tre spesialkombinasjoner gir ekstra mye poeng:
 - to ganger tre like, f.eks. [2,2,2,5,5,5], gir 2500 poeng
 - straight [1,2,3,4,5,6] gir 2000 poeng
 - tre par, f.eks. [1,1,4,4,6,6] gir 1500 poeng
- Hvis man ikke har noen spesialkombinasjon, gjelder følgende:
 - tre enere oppnådd i samme kast gir 1000 poeng
 - tre like av annet tall, gir tallet x 100 (f.eks. tre 6'ere, 600, tre 2'ere, 200)
 - fire like gir det dobbelte av tre like, fem like det dobbelte av fire like, og seks like igjen dobbelt av dette (f.eks. 4 enere gir 2000, 5 enere 4000, 6 enere 8000).
 - enkeltstående 1 gir 100 poeng per stykk, og enkeltstående 5 gir 50 poeng per stykk. Andre tall gir ikke noen poeng enkeltvis.

Vi ønsker en funksjon **points(throw)** som kan regne ut poengverdien av ett kast med 6 terninger, hvor inn-parameteren **throw** er ei liste av heltall som var i kastet, og hvor vi kan anta at tallene alltid kommer sortert i stigende rekkefølge. Eksempel på ønsket resultat av kjøring:

points([1,2,3,4,5,6]) gir 2000 poeng (straight)

points([1,1,1,2,2,2]) gir 2500 poeng (to tripler)

points([1,1,1,2,5,5]) gir 1100 poeng (1000 for 3 enere + 2 x 50 for to femmere)

points([1,2,2,2,4,4]) gir 300 poeng (200 for 3 toere, pluss 100 for den ene eneren)

points([2,3,4,4,4,4]) gir 800 poeng (for 4 firere, dvs. dobbelt av poeng for tre firere)

points([1,2,3,3,4,5]) gir 150 poeng (100 for eneren, 50 for femmeren)

points([2,3,3,4,4,6]) gir 0 poeng

Fyll inn det som mangler slik at funksjonen virker som den skal. Ingen minuspoeng for feil, så du BØR svare noe også der du er usikker.

```
def points(throw):
```

```
    if throw ==  (sorted(throw), set(throw), [1,2,3,4,5,6]):
```

```
        score = 2000 # straight
```

```
    elif len(set(throw)) == 2 and  (throw[3] < throw[4], throw[2] != throw[3],
```

```
        throw[1] == throw[2]):
```

```
            score = 2500 # two triplets
```

```
    elif len(set(throw)) == 3 and  (throw[0::2] == throw[1::2], throw[0] ==
```

```
        throw[1], throw[0,2,4] == throw[1,3,5]):
```

```
            score = 1500 # three pairs
```

```
    else: # other cases
```

Velg alternativ (score = 0, throw.sort(), result = [])

for n in set(throw):

ct = throw.count(n)

Velg alternativ (if ct == 3, if ct > 3, if ct >= 3):

multi = Velg alternativ ((4 - ct) ** 2, 2 ** (ct-3), 4 - ct)

if n == 1:

score += Velg alternativ (1000 * multi, 1000 * multi + 50 * ct, (1000 + 50 *

ct) * multi)

else:

score += Velg alternativ (n*100 * multi, (n - 1) * multi, n * multi)

Velg alternativ (elif ct < 3:, elif n == 1:, elif ct <= 3:)

score += ct * 100

Velg alternativ (elif n > 1:, else:, elif n == 5:)

score += ct * 50

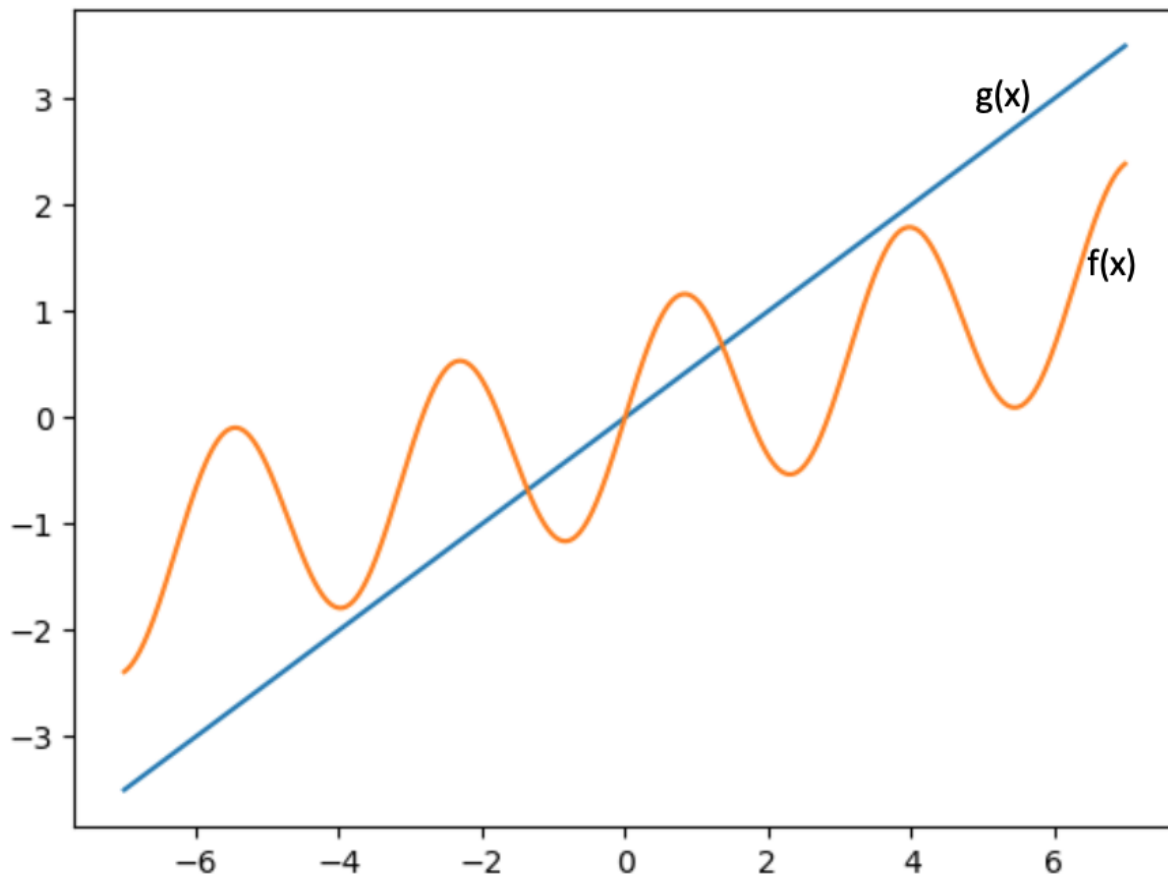
return score

Maks poeng: 9

15 Kopi av func as param, p1 (6%)

Vi ønsker å lage en funksjon `n_crossings(f, g, x_values)` som får inn som parametre to andre funksjoner `f` og `g`, samt et array `x_values` som inneholder en serie x-verdier. Funksjonen `n_crossings()` skal returnere antall ganger `f(x)` og `g(x)` krysser hverandre over den gitte serien av x-verdier, dvs. . ganger hvor det skifter hvilken funksjon som er størst. NB: funksjonene må krysse hverandre for at det skal gi telling, det er ikke nok at de tangerer hverandre. (Dvs., hvis f.eks. `f` og `g` var samme funksjon, slik at de var like hele tiden, skulle dette gi returverdi 0 fra `n_crossings`)

Eksempel: Hvis `f(x)` er $\sin(2x) + 0.2x$ (oransje kurve) og `g(x)` er $0.5x$ (blå linje), som vist i figuren nedenfor, så skal `n_crossings(f, g, x_values)` returnere **3** hvis `x_values` er `np.arange(-2, 2, 0.1)` siden de to funksjonene krysser hverandre tre steder i intervallet -2 til +2 og inkrementverdien 0.1 er fingranulær nok til å fange opp alle disse kryssingene. Hvis derimot `x_values` er `np.arange(4, 6, 0.1)` skal den returnere 0, siden det ikke er noen kryssinger i intervallet mellom `x=4` og `x=6`.



OPPGAVE: Trekk kodelinjer til riktig sted så funksjonen virker som den skal. Tre av kodelinjene skal IKKE brukes. Ingen minuspoeng for feil, så du BØR svare noe også der du er usikker.

 [Hjelp](#)

if small(x) > big(x):

crossings = 0

return result

elif f(x) == g(x):

if f(x) <= g(x):

small, big

if f(x) == g(x):

g, f

crossings += 1

elif f(x) < g(x):

for x in x_values:

f, f

f, g

def n_crossings(f, g, x_values):

big, small =

big, small =

elif f(x) > g(x):

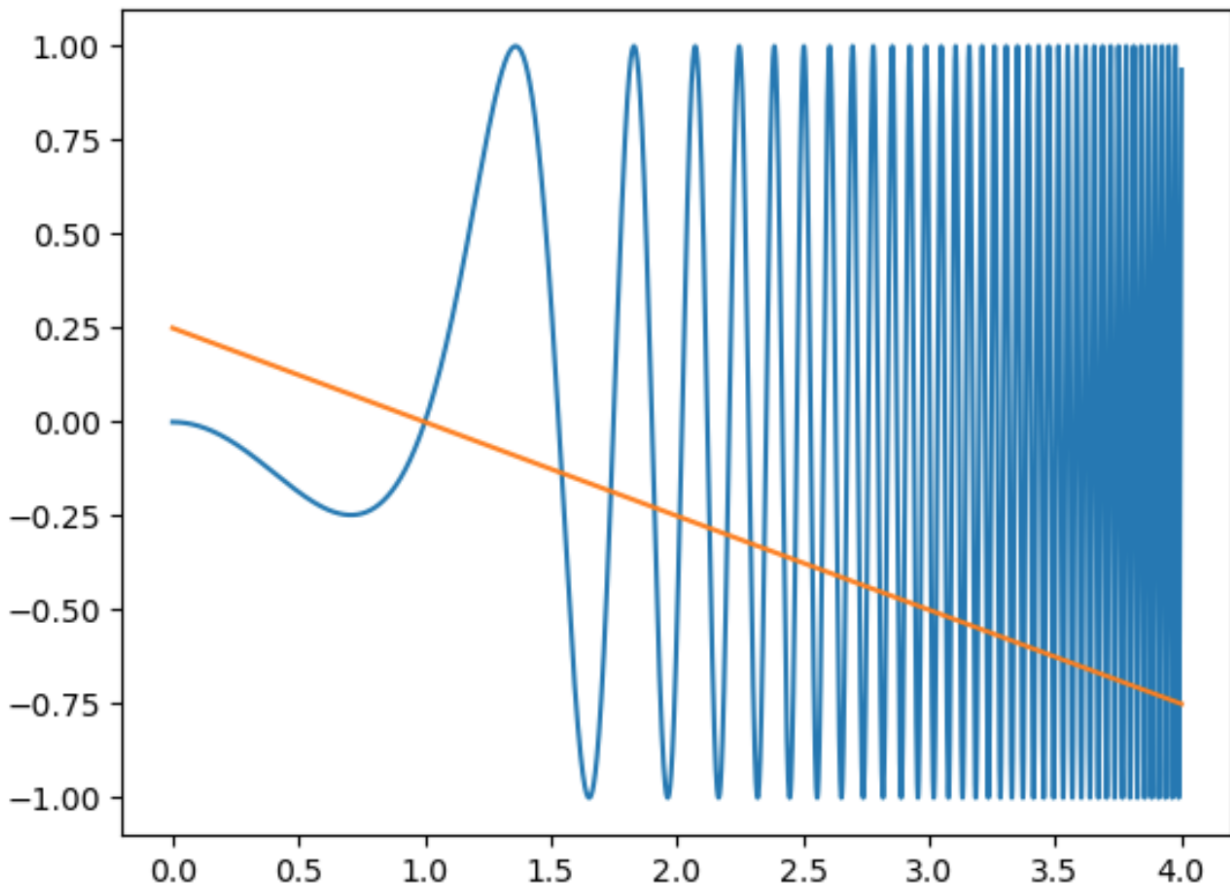
big, small =

big, small =

Maks poeng: 6

16 Kopi av func as param, p2 (6%)

Hvis to funksjoner krysser hverandre svært hyppig, kan det lett skje at funksjonen `n_crossings()` som vi laget i forrige oppgave, ikke finner alle kryssningene. Dette skjer typisk hvis kryssningene kommer tettere enn x-verdiene i arrayet som vi får inn. Et eksempel på en slik situasjon er gitt i bildet, med funksjonene $f(x) = 0.25 - 0.25x$ (oransje linje) og $g(x) = \sin(x^4 - x^2)$ (blå). Hvis det er 0.1 intervall mellom x-verdiene, vil `n_crossings`-funksjonen virke fint for intervallet 0.0-1.2, hvor det bare er én kryssing, men f.eks. for intervallet 3.5-4.0 vil den returnere et altfor lite tall fordi kryssningene kommer hyppigere enn x-verdiene.



Vi ønsker derfor å lage en ny funksjon som skal kalle funksjonen `n_crossings()`. Den nye funksjonen skal hete `test_n_crossings(f, g, a, b)` og skal få inn fire parametre: to funksjoner `f` og `g`, samt tallene `a` og `b` som er henholdsvis startverdi og sluttverdi for intervallet hvor vi skal telle kryssninger. Test-funksjonen skal starte med avstand 1 mellom x-verdiene og så gå gjentatte runder, hvor avstanden deles på 10 for hver runde, inntil tre påfølgende runder har gitt identisk resultat. Funksjonen skal returnere ei liste med antall kryssninger den fant på hver runde. Eksempel på kjøring (hvis funksjonene `f` og `g` var som i bildet over):

```
>>>test_n_crossings(f, g, 0.0, 2.0)
[0, 2, 2, 2]
>>>test_n_crossings(f, g, 3.5, 4.0)
[0, 2, 20, 32, 32, 32]
```

Som vi ser, returnerer det første kallet feilaktig 0 når x-avstand er 1 (finner ingen kryssinger fordi blå kurve er lavere enn oransje linje både for $x = 0.0$, 1.0 og 2.0), men kommer så til riktig svar allerede med x-avstand 0.1 og lavere. Det neste kallet returnerer feilaktig få kryssninger både med x-avstand 1.0, 0.1 og 0.01, før den stabilt kommer til svaret 32 fra 0.001 og utover.

OPPGAVE: Trekk kodelinjer til riktig posisjon slik at funksjonen `test_n_crossings()` virker som den skal. Tre av kodefragmentene skal ikke brukes. Ingen minuspoeng for feil svar,

så du bør svare noe også der du er usikker.

 [Hjelp](#)

results.append(n_crossings(f, g, x_values))

if len(results) >= 3

results = [n_crossings(f, g, x_values)]

x_values = np.arange(a, b + h/2, h)

h /= 10

while len(results) < 3

results[-3:] != [results[-3]]*3

x_values = np.arange(a, b + 0.5)

results += n_crossings(f, g, x_values)

results = n_crossings(f, g, x_values)

return results

def test_n_crossings(f, g, a, b):

h = 1

or

:

Maks poeng: 6

17 Kopi av Long division (5%)

Lærer "Leif" ønsker å lage et program som kan illustrere manuelt utført divisjon for elevene. Som del av dette programmet ønskes en funksjon **show_division(n, m)** som skal returnere en streng over flere linjer som viser hvordan man går fram for å dele n på m. Eksempel på kjøring:

show_division(127, 4) skal returnere en streng på 7 linjer som vist under. Linje 2 viser -12 fordi første siffer i løsningen er 3, og 3×4 (divisoren) er 12. Linje 4 viser 7 fordi det er det man har igjen av dividenden når man trekker fra $12 - 12$. Linje 5 viser -4 fordi neste siffer i løsningen er 1, som multiplisert med divisoren gir 4. Til slutt får vi 3 i rest på linja nederst.

```
127 : 4 = 31
-12
----
  7
 -4
 ----
  3 i rest
```

Hvis divisjonen går opp, f.eks. **show_division(124, 4)** skal det stå **0 i rest** på siste linje av resultatstrengen. Merk at det ikke nødvendigvis blir 7 linjer i strengen som returneres. Hvis svaret bare blir ensifret, vil resultatet ha færre linjer, og hvis det blir mer enn to sifret, flere linjer.

OPPGAVE: Velg rett alternativ i hvert tomrom slik at funksjonen show_division() virker som den skal. Ingen minuspoeng for feil valg, så du BØR svare noe også der du er usikker. Funksjonen len_diff() er en hjelpefunksjon som brukes for å finne differansen i lengde mellom to tall (i antall siffer). Tekststrengen '\n' betyr linjeskift.

```
def len_diff(num1, num2):
    return len(str(num1)) - len(str(num2))
```

```
def show_division(n, m):
```

Velg alternativ (result = str(n // m), result = n / m, result = n // m)

```
division_string = f' {n} : {m} = {result}'
```

```
indent = " # to enkle apostrofer helt inntil hverandre, dvs tom streng
```

```
num = n
```

```
for digit in result:
```

Velg alternativ (subtr = n - digit, subtr = digit * m, subtr = int(digit) * m)

```
division_string += Velg alternativ (f'{m*indent}-{subtr}\n', f'\n{indent}-{subtr}', f'{indent}-{subtr}\n')
```

division_string += Velg alternativ (f'{m*indent}----\n', f'\n{indent}----', f'{indent}----\n')

Velg alternativ (new_num = num - subtr * 10 *, new_num = subtr - num * 10 *,
new_num = num - subtr * 10 **) len_diff(num, subtr)

indent += ' ' * Velg alternativ (len_diff(subtr, num), len_diff(num, subtr), len_diff(num,
new_num)) # en blank mellom apostrofene

Velg alternativ (num = new_num - subtr, new_num = num - subtr, num = new_num)

division_string += Velg alternativ (f'\n {indent}{num}', f' {indent}{new_num}\n', f'\n
{indent}{new_num-subtr}')
division_string += ' i rest\n'
return division_string

Maks poeng: 7.5

18 Kopi av 2D arrays (12%)

Funksjonen **same_row_col(a, b, n)** skal få inn to todimensjonale numpy-array (matriser) a og b som inneholder tall, samt opsjonelt en tredje parameter n som vil inneholde et heltall ≥ 0 . Det skal også gå an å kalle funksjonen uten å gi inn noe tall for n, i så fall skal den være null. Funksjonen skal returnere ei liste av tupler, hvor hvert tuppel består av to heltall (r, k) - hvor r er indeks til ei rad i arrayet a, og k er indeks til ei kolonne i arrayet b. Kriteriet for at et tuppel (r, k) skal være med i løsningen er at det forekommer en identisk tallserie i aktuell rad og kolonne, dvs. samme tall i samme rekkefølge. Mer detaljert er kravene:

- hvis det ikke gis noe tredje argument til funksjonen, eller tredje argument n er 0, må rad r og kolonne k være identiske i sin helhet (dvs. hele rada lik hele kolonna)
- hvis n er et heltall > 0 , må rad r og kolonne k ha en identisk tallserie på minst n elementer.

a [[1 2 3 4]	b [[5 0 6 6]
[5 6 7 8]	[6 0 6 7]
[1 1 1 1]]	[7 0 8 8]
	[8 1 1 1]]

Eksempel, gitt at a og b har innhold som vist over:

- **same_row_column(a, b)** skal returnere [(1, 0)]. Eneste tilfelle hvor ei hel rad i a er identisk med ei hel kolonne i b, er tallserien [5 6 7 8] som fins i rad i a med indeks 1, kolonne i b med indeks 0 - illustrert med fet skrift i a og b ovenfor.
- **same_row_column(a, b, 3)** skal returnere [(1, 0), (1, 3)]. Tallserien [5 6 7 8] fra forrige eksempel tilfredsstiller kravet også her (siden 4 tall er ok når det trengs minst 3 tall). Dessuten får vi med (1, 3) fordi tallserien 6, 7, 8 fins i både rad 1 i a og kolonne 3 i b (vist med kursiv i b)
- **same_row_column(a, b, 5)** skal returnere tom liste [], fordi det ikke fins noen serie på minst 5 tall som er identisk mellom de to matrisene.

Merk at resultatlista kun skal inneholde én forekomst av hvert tuppel som er med. I eksemplet `same_row_column(a,b,3)` vil tuppelet (1,0) kvalifisere på tre ulike måter: hele tallserien 5,6,7,8 matcher, 5,6,7 matcher, og 6,7,8 matcher, men (1,0) skal likevel bare være en gang i resultatet, ikke 3. For å få full score på oppgaven må løsningen ikke bare returnere korrekt svar, men også unngå å lete videre etter flere matcher for en gitt (r, k) hvis det allerede er fastslått at tuppelet skal være med i løsningen.

Skriv koden for funksjonen `same_row_column()`.



Maks poeng: 12