NTNU

KANDIDAT

10489

PRØVE

TDT4109 1 Informasjonsteknologi, grunnkurs

Emnekode	TDT4109
Vurderingsform	Skriftlig eksamen
Starttid	28.11.2024 14:00
Sluttid	28.11.2024 18:00
Sensurfrist	19.12.2024 22:59
PDF opprettet	14.05.2025 18:02

Seksjon 0 - Info

Oppgave	Tittel	Oppgavetype
ℹ	TDT4109 - H2024 - Forside	Informasjon eller ressurser
ℹ	Eksamensstruktur og råd	Informasjon eller ressurser

Seksjon 1 - Automatisk rettede oppgaver (65%)

Oppgave	Tittel	Oppgavetype
1	1a (2%) Alt er sant eller usant	Flervalg
2	1b (2%) Den stringente	Paring
3	1c (2%) Alle tall har verdi	Plasser i tekst
4	1d (2%) Kodeforståelse	Flervalg
5	1e (2%) Dobbeldobbelopp	Flervalg
6	1f (2%) Den stumme	Fyll inn tekst
7	1g (2%) Req	Flervalg
8	1h (2%) Lange uttrykk	Paring
9	1i (3%) Størst mulig areal	Plasser i tekst
10	1j (3%) Listesnop	Fyll inn tall
11	1k (3%) Alt blir grått	Plasser i tekst
12	1l (3%) Lost in Translation	Fyll inn tekst
13	1m (4%) Velg riktig	Paring
14	1n (4%) Rekursjon	Plasser i tekst
15	1o (5%) Command and Conquer	Fyll inn tekst
16	1p (6%) Filer og inntak	Paring
17	1q (6%) På skråplanet	Nedtrekk
18	1r (6%) Kan du lese dokumentasjon?	Plasser i tekst
19	1s (6%) Sudokustyr	Fyll inn tekst

Seksjon 2 - Programmering (35%)

Oppgave	Tittel	Oppgavetype
ℹ	Del 2 - Disse poenga...	Informasjon eller ressurser
20	2a (3%) get_number(...) - Finn seksjonsnummeret	Programmering
21	2b (3%) exam_score(...) - Totalpoeng	Programmering
22	2c (3%) update_scores(...) - Oppdater totalpoeng	Programmering
23	2d (4%) grade_points(...) - Karakterberegning	Programmering

24	2e (5%) - print_result(...) - Topp til bunn	Programmering
25	2f (4%) filter_section(...) - Oppgavefiltrering	Programmering
26	2g (7%) winner_per_section(...) - Seksjonsvinner(e)	Programmering
27	2h (6%) read_scores(...) - Filgreier	Programmering

Her kan du kladde

Husk at fullt cheat-sheet og dok. for Python, Numpy og Matplotlib finnes som ressurser.

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

Built-in
Often used functions and operators
`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([start,]stop[,step]):`

`start:` Starting number of the sequence.

`stop:` Generate numbers up to (from **`0`**), but not including, this number. Often used in combination with for loops: **`for i in range(10)`**.

`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.

s.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` <= *N* <= `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` <= N <= `b` for `a` <= `b` and `b` <= N <= `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… 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 π = 3.141592…, to available precision.

`math.e`: The mathematical constant `e` = 2.718281…, to available precision.

1 1a (2%) Alt er sant eller usant

I denne oppgaven skal vi finne ut om uttrykkene til `a` til `f` gir `True` eller `False`:

a = True or (False and True) or False

Verdien til a:

- [x] True
- [ ] False

```
b = 4 < 3
```

Verdien til b:

- [x] False
- [ ] True

```
c = (not a and not b) or b
```

Verdien til c:

- [ ] True
- [x] False

```
d = "a" in {"a":1, "b":2}
```

Verdien til d:

- [ ] False
- [x] True

```
e = bool([]) == bool({})
```

Verdien til e:

- [ ] False
- [x] True

```
f = 3 == 3.000
```

Verdien til f:

- [x] False
- [ ] True

Maks poeng: 2

2 1b (2%) Den stringente

```
test_string = "Hello world"
concat_string = "Hello" + " " + "Python"
```

Anta koden over. I tabellen nedenfor står en del uttrykk. For hver rad, se på uttrykket til venstre, og kryss av for hva som blir returnert for de ulike alternativene.

	world	World	WORLD	python	Python	PYTHON
test_string.upper()[6:]	○	○	◉	○	○	○
test_string.replace('World', 'Python')[6:]	◉	○	○	○	○	○
concat_string[6:]	○	○	○	○	◉	○
concat_string.lower()[6:]	○	○	○	◉	○	○

Maks poeng: 2

3 1c (2%) Alle tall har verdi

Funksjonen under skal summere alle tallene i strengen den får inn. Et tall kan bestå av flere siffer.

Eksempel: `summer("1to3fire11")` skal returnere `15`.

Hjelp

`liste.append(int(variabel))` `variabel += x`

```
def summer(streng):
    liste = []
    variabel = ""
    for x in streng:
        if x.isdigit():
            variabel += x
            continue
        elif variabel:
            liste.append(int(variabel))
            variabel = ""
    if variabel:
        liste.append(int(variabel))
    return sum(liste)
```

Maks poeng: 2

4 1d (2%) Kodeforståelse

```
def dobbelopp(liste):
    for element in liste:
        element *= 2
    return liste

x = [1, 2, 3]
#1 print(dobbelopp(x))
#2 print(x)
```

Vi ønsker her å skrive kode som dobler hvert element i listen som hentes inn som parameter. Du kan fortsette at listen bare inneholder heltall.

Hvilken påstand under er sann?

- Bare #1 vil skrive ut [2, 4, 6]
- Bare #2 vil skrive ut [2, 4, 6]
- Både #1 og #2 vil skrive ut [2, 4, 6]
- **Ingen av #1 og #2 vil skrive ut [2, 4, 6], men koden vil ikke krasje.**
- Koden vil krasje på linjen med *=

Maks poeng: 2

5 1e (2%) Dobbeldobbelopp

```
def dobbelopp(liste):
    for element in liste:
        print(element*2)
```

Vi har gjort en liten endring i funksjonen `dobbelopp(liste)`, ved at den nå i stedet skriver ut resultatet. Alle kallene under, bortsett fra ett, resulterer i den samme utskriften. Hvilket kall er det som avviker fra resten?

- **dobbelopp({"123")**
- dobbelopp({1, 2, 3})
- dobbelopp((1, 2, 3))
- dobbelopp([1, 2, 3])
- dobbelopp({1:1, 2:2, 3:3})

Maks poeng: 2

6 1f (2%) Den stumme

```
def foo(liste):
    liste.append(4)
    liste[0] = 5
    return liste
liste = [1, 2, 3]
new_liste = foo(liste)
print(sum(liste) - sum(new_liste))
```

Gitt at koden over kjøres. Hva skrives ut? Her er det veldig viktig at du skriver inn bare det nøyaktige svaret. Ingenting mer! Ikke "Svaret er ..." Nei! Inspera vil ikke ha noe annet enn bare svaret.

Svar: 0

Maks poeng: 2

7 1g (2%) Req

```
def sumpin(stuff):
    return int(stuff[-1])*sumpin(stuff[:-1])
print(sumpin("1234"))
```

Gitt at koden over kjøres. Hva skrives ut?

- "24"
- 0
- 24
- Feilmelding

Maks poeng: 2

8 1h (2%) Lange uttrykk

```
R = {1, 2, 3, 3}
S = set([7, 8, 9, "ost"])
T = {2, 3, 4, 5, 6, 7}
```

Anta koden over. I tabellen nedenfor står en del uttrykk. For hver rad, se på uttrykket til venstre, og kryss av riktig lengde for de ulike alternativene.

	1	2	3	4	5	6	7	8	9
len(R.intersection(T))	○	●	○	○	○	○	○	○	○
len(R.union(S))	○	○	○	○	○	○	●	○	○
len(S.union(R).difference(T))	○	○	○	●	○	○	○	○	○
len(R.intersection(T).symmetric_difference(S))	○	○	○	○	○	●	○	○	○

Maks poeng: 2

9 1i (3%) Størst mulig areal

Funksjonen under skal finne det største arealet en får, gitt tallet verdi (heltall).

- Arealet beregnes ved å gange positive heltall opp til verdi med logaritmen til *resten* (verdi = tallet + resten)
- Funksjonen er ikke pinlig nøyaktig, og nøyer seg med å teste for alle heltallene opp til verdien.
- Funksjonen skal returnere verdien *x* som gir størst areal (heltall), samt dette arealet som flyttall.
- De to returverdiene skal returneres som et tuppel.

Hjelp

0 | 0, stuff | 0, verdi | 1, stuff | 1, verdi | i - verdi | i, verdi | stuff, 0 | verdi, 0

```
import math
def beregn(verdi):
    viktig = ( 0, 0 )
    for i in range( verdi ):
        stuff = ( verdi-i )*math.log(i)
        if stuff > viktig[ 1 ]:
            viktig = ( i, stuff )
    return viktig
```

Maks poeng: 3

10 1j (3%) Listesnop

```
liste = [1, 2, 5]
liste.append(4)
liste[-1] # Fill in the value: 4 liste.remove(2) sum(liste) # Fill in the value: 10 liste.pop(0) len(liste) #
```

Anta koden over. Fyll inn under hva verdiene vil være ved ulike tidspunkt.

Maks poeng: 3

11 1k (3%) Alt blir grått

Innimellom er det greit å jevne ut ting. I denne oppgaven skal du utjevne sifrene i en liste:

- Sifferet på plass *x* skal være snittet av verdiene på plass *x*, samt verdien før og etter i listen.
- Finnes ikke disse stedene så skal man beregne gjennomsnittet for de verdiene man finner.

Hjelp

2	3	sum(liste[-2:])	sum(liste[0:2])	sum(liste[i+2:i-1])
sum(liste[i-1:i+2])	sum(liste[i-1:i+2])	sum(liste[i-2:i+2])	sum(liste[i:])	

```
def sandpapir(liste):
    ut = []
    for i in range(len(liste)):
        if i == 0:
            ut.append( sum(liste[0:2]) / 2 )
        elif i == len(liste) - 1:
            ut.append( sum(liste[-2:]) / 2 )
        else:
            ut.append( sum(liste[i-1:i+2]) / 3 )
    return ut
```

Maks poeng: 3

12 1l (3%) Lost in Translation

```
def transhleit(streng):
    ut = []
    for x in range(0, len(streng), 2):
        ut.append(chr(ord(streng[x+1])-1))
        ut.append(chr(ord(streng[::-1][5 - x])))
  return "".join(ut)
print(transhleit("ohikas"))
```

Anta at koden over kjøres. Hva skrives ut? Her er det veldig viktig at du skriver inn bare det nøyaktige svaret. Ingenting mer! Ikke "Svaret er ..." Nei! Inspera vil ikke ha noe annet enn bare svaret.

Svar: hojira

Maks poeng: 3

13 1m (4%) Velg riktig

```
a = 'velg'
b = 'riktig'
```

Anta koden over.

Hvilke(n) datatype(r) blir returnert? Eller krasjer den?

I tabellen nedenfor står en del uttrykk. 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 selv om du er usikker.

	int	float	bool	string	ERROR
40 * 0.5	○	●	○	○	○
11//2	●	○	○	○	○
1 % 0	○	○	○	○	●
6 - 1== 5.0	○	○	●	○	○
b - a	○	○	○	○	●
a[1.0]	○	○	○	○	●
7/11	○	●	○	○	○
a + b	○	○	○	●	○

Maks poeng: 4

14 1n (4%) Rekursjon

Funksjonen `kast_terninger(tall)` skal fungere slik:

- Tallet skal returnere et tuppel (jeg husket hva det het!) med så mange tilfeldige sekserterninger som tallet angir.
- *Eksempel:* `kast_terninger(4)` kan returnere for eksempel `(3, 1, 4, 4)`.

Oppgaven skal løses rekursivt.

Hjelp

randint | random | tall | tall +1 | tall - 1

```
import [ random ]
def kast_terninger(tall):
    if [ tall ] == 0:
        return []
    return [ [ random ] . [ randint ] (1,6)] + kast_terninger( [ tall - 1 ] )
```

Maks poeng: 4

15 1o (5%) Command and Conquer

Funksjonen `cnc(l)` tar inn en liste med strenger. Hver streng representerer en avlesning av målene på en kube. Det vil alltid være to heltall eller flyttall i strengen, fulgt av om målet er gjort i `m`, `dm` eller `cm`. Måleenheten kan etterfølges av punktum, komma eller mellomrom, men du kan forutsette at det alltid er en måleenhet etter tallet.

- Det finnes en hjelpefunksjon `convert_value(value, string)` som også skal lages.

Målet for funksjonen er å regne ut total areal disse kubene dekker, i kvadratmeter, og så returnere dette.
Eksempel:
>>> l = ["Høyde: 120 cm og bredde: 1.2 m", "130 cm, 13 dm"]
>>> cnc(l)
3.13 # 1.2*1.2 + 1.3*1.3

```
def convert_value(value, string):
    if string.[contains("cm")] :
        return [value*100]
    elif string.[contains] ("dm"):
        return [value*10]
    else:
        return value
def cnc(l): # cnc = calculate and convert
    total_area = 0
    for line in [l] :
        if "id:" [] :
            []
        side1 = 0
        side2 = 0
        parts = [line.split(" ")]
        for i, part in [enumerate(parts)] :
            try:
                val = [float(part)]
                if side1 == [0] :
                    side1 = [convert_value] (val, parts[i+1])
                else:
                    side2 = [convert_value] (val, parts[i+1])
                    [total_area] += side1*side2
            [except ValueError] :
                pass
    return total_area
```

Maks poeng: 5

16 1p (6%) Filer og inntak

```
def respons(fil, siffer):
    try:
        fil.append(".txt")
    except:
        return "Én"
    if not siffer and fil[-1] == ".txt":
        return "To"
    try:
        total = 0
        with open(fil[0]+fil[1]) as f:
            lines = f.readlines()
            for i in range(siffer):
                total += lines[i].index(str(i+1))
        return total
    except FileNotFoundError:
        return "FeilFil"
    except ValueError:
        return "FeilVerdi"
    except:
        return "GenerellFeil"
    return "Nah"
```

Funksjonen `respons(...)` har to parametre - `fil` og `siffer`.

- Funksjonen skal telle hvor på en linje denne linjens linjenummer opptrer for alle linjer i filen opp til nummeret angitt ved `siffer`.
- Med linjenummer tenker vi her at i tekstfilen starter vi på linje nummer *1*, og ikke *0*.
- Noen unntak kan oppstå, gitt argumenter og innholdet i filen.
- Du kan forutsette at filen `"kunst.txt"` befinner seg i samme mappe, og den inneholder 4 linjer:

Linje 1 er den første, eller erst
Linje 2 kommer dernest! (nødrim)
Bronsen går til linje tre,
4 plass - akk oh ve.

Din oppgave er å finne ut hva funksjonen *respons* returnerer for de ulike argumentene gitt under.

	Én	To	FeilFil	FeilVerdi	GenerellFeil	Nah	6	12
respons("kunst.txt", 0)	●	○	○	○	○	○	○	○
respons(["kunst"], 0)	○	●	○	○	○	○	○	○
respons(["kunst"], 1)	○	○	○	○	○	○	●	○
respons(["lala"], 1)	○	○	●	○	○	○	○	○
respons(["kunst"], 3)	○	○	○	●	○	○	○	○
respons(["kunst"], 2)	○	○	○	○	○	○	○	●

Maks poeng: 6

17 1q (6%) På skråplanet

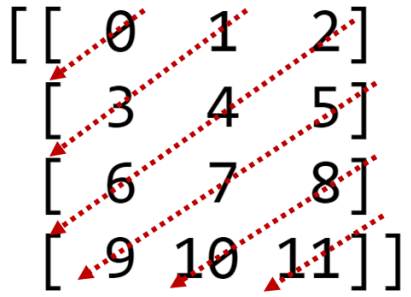
```
# Se oppgavebeskrivelsen under
def diagonala(a):
     result = list()
     rows, cols = len(a), len(a[0])

     diags = [ row+cols-1 ](row+cols, row+cols+1, row+cols-1)     for d in range(diags):          if [ d < cols
             [ i = rows - 1 ](i = rows - 1, i = d, i = 0, i = rows - 1 - d)
             [ j = cols - 1 - d ](j = cols - 1, j = d, j = cols - 1 - d, j = 0)                  else: [ i = rows
             [ j = 0 ](j = cols - (d - rows) - 2, j = cols - 1, j = 0, j = d - i)                 while [ i
             [ j += 1 ](j += 1, j -= 1, j -= i, j -= d)          return result
```

Funksjonen `diagonal(a)` får inn et todimensjonalt liste med tall og skal returnere en endimensjonal liste med de samme tallene, men der rekkefølgen på elementene fremkommer ved å iterere den todimensjonale listen diagonalt. Koden skal kunne virke på lister av vilkårlig størrelse, dvs. ikke bare på eksemplet under.

Eksempel på kjøring:
>>> a = [[0,1,2], [3,4,5], [6,7,8], [9,10,11]]
>>> print(diagonal(a))
[11, 10, 8, 9, 7, 5, 6, 4, 2, 3, 1, 0]

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



Velg riktig i feltene slik at koden virker som den skal.

Maks poeng: 6

18 1r (6%) Kan du lese dokumentasjon?

I denne oppgaven skal du gjøre noe litt merkelig. Du skal lese dokumentasjonen til en modul i Python som ikke er en del av pensum. Hvorfor i alle dager er det relevant? Jo, det er klart relevant å kunne lese en enkel spesifikasjon, og så bruke koden slik dokumentasjonen beskriver.

Modulen som skal brukes heter *json*. Alt du trenger å vite er at *json*-modulen gjør deg i stand til å lagre en variabel direkte til en fil, og lese rett i fra en fil og inn i en variabel. Ikke noe styr med parsing av tekstfiler, eller sånt *pickles*-tull.

Dokumentasjonen du trenger for å bruke *json* finner du vedlagt som en PDF-fil, se under ressurser.

Oppgaven er som følger:

- Det ligger en dictionary lagret i filen *d.json* i samme mappe som du 'står i'.
- Du skal, ved å bruke *json*-biblioteket, lese og så lagre en oppdatert versjon der alle nøkler er doblet.
- Du skal også skrive ut hvor mange tegn hele dictionarien er, som i at *{1:2}* totalt er *5* tegn langt. Det holder med at tallet skrives ut.

Så skal du skrive all koden som trengs for å gjøre nettopp det, da.

Hjelp

d.json | dump | dumps | json | json.dumps | json.load | json.loads | load | loads

```
import [ json ]
with open(" [ d.json ] ", "r") as f:
    data = [ json.loads ] (f)
updated_data = {key*2: value for key, value in data.items()}
print(len( [ json.dumps ] (updated_data)))
with open(" [ d.json ] ", "w") as f:
    [ dumps ] .(updated_data, f)
```

Maks poeng: 6

19 1s (6%) Sudokustyr

```
# SE UNDER KODEN FOR BESKRIVELSE AV OPPGAVEN!
# SCROLL BELOW THE CODE!
# Check if a list contains only unique digits of appropriate value
def is_valid_block(block):
    return len(block) == len(set(block)) and max(block) [==len(block)] len(block)def is_valid(board):

    dim = len(board) # Ytre dimensjoner av brettet
    sub = int(dim ** 0.5) # Størrelsen (rader og kolonner) for en blokk (dim = 9 -> sub = 3)

    # Check rows
    for row in board:
        if not is_valid_block(row):
            return [False]        # Check columns    for col in [range(dim)] :        (
```

```
board_9x9 = [
    [5, 3, 3, 6, 7, 8, 9, 1, 2],
    [6, 7, 2, 1, 9, 5, 3, 4, 8],
    [1, 9, 8, 3, 4, 2, 5, 6, 7],
    [8, 5, 9, 7, 6, 1, 4, 2, 3],
    [4, 2, 6, 8, 5, 3, 7, 9, 1],
    [7, 1, 3, 9, 2, 4, 8, 5, 6],
    [9, 6, 1, 5, 3, 7, 2, 8, 4],
    [2, 8, 7, 4, 1, 9, 6, 3, 5],
    [3, 4, 5, 2, 8, 6, 1, 7, 9]
]
```

Vi skal lage en funksjon `is_valid(board)` som får inn en todimensjonal liste - et innfylt sudokubrett i dimensjoner 4x4, 9x9, 16x16 eller 25x25.

- Funksjonen skal returnere `True` hvis brettet er gyldig, ellers `False`.
- Reglene for sudokubrett, som `is_valid(board)` må sjekke, er for standard 9x9-versjonen slik:
  - Hver rad og hver kolonne må inneholde nøyaktig ett eksemplar av hvert av tallene 1-9 - se for eksempel rad merket med grønn stiplet boks og kolonne merket med heltrukket boks i figuren over.
  - Hver 3x3 boks må også inneholde nøyaktig ett eksemplar av hvert av tallene 1-9 - se bokser merket opp med grønt og hvitt (sjakkmønster) i figuren.
  - For ordens skyld, rader går 'vannrett' <----> mens kolonner går opp og ned |||
- Hvilke tall som skal være med i rader, kolonner og 'bokser' avhenger av hvor stort brettet er. I figuren over er brettet på 9x9, og har da 9 unike tall.

For mindre og større varianter er reglene tilsvarende, bare at det blir færre eller flere ulike tall, og småboksene får annen størrelse (2x2 i en 4x4-sudoku, 4x4 i en 16x16 sudoku, osv.)

Din oppgave:
Fyll inn hullene i koden under. Det er to funksjoner.

IKKE LEGG INN MELLOMROM MELLOM ARGUMENTER: BRUK `x-y` OG IKKE `x - y`

Maks poeng: 6

Her kan du kladde
Husk at fullt cheat-sheet og dokumentasjon for Python finnes som ressurser.

Oppgavens datastrukturer (beklager at den er et bilde, men Inspera nekter å lagre teksten på en godt organisert og fargelagt måte...):

```
candidates = [{'name':'Børge', '1a':4, '1b':6, '1c':3, '2a': 5, \
                  '2b': 7, '2c': 8, '3a': 5, '3b': 9},
              {'name':'Vetle', '1a':5, '1b':7, '1c':4, '2a': 6, \
                  '2b': 8, '2c': 9, '3a': 9, '3b': 9},
              {'name':'Mathea', '1a':6, '1b':8, '1c':5, '2a': 7,\
                '2b': 9, '2c': 10, '3a': 8, '3b': 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

Built-in
Often used functions and operators
`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([start,]stop[,step]):`
`start:` Starting number of the sequence.
`stop:` Generate numbers up to (from `0`), but not including, this number. Often used in combination with for loops: **`for i in range(10)`**.
`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.

s.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` <= *N* <= `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` <= N <= `b` for `a` <= `b` and `b` <= N <= `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… 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 π = 3.141592…, to available precision.

`math.e`: The mathematical constant `e` = 2.718281…, to available precision.

20

2a (3%) get_number(...) - Finn seksjonsnummeret

En oppgave kan ha nøkkelen `'1a'`, eller kanskje `'10c'` (altså seksjon pluss deloppgave). Som ledd i en senere oppgave (i din eksamen) kan det være greit å hente ut sifrene først i denne strengen. Du skal altså skrive funksjonen `get_number(string)` som returnerer sifrene i oppgaven. Du skal altså finne *seksjonen* basert på *oppgaven*!

- Gitt `'1a'` inn så skal funksjonen returnere strengen `'1'`.
- Gitt `'13w'` inn så skal funksjonen returnere strengen `'13'`.
- Det vil kun være én bokstav (deloppgaven) bak sifferet (seksjonen) i navnet på en oppgave.
- Du trenger ikke å dobbeltsjekke at inputen er på riktig format, du kan gå ut i fra at den er det.

Eksempel:
>>> print(get_number('1a))
'1'

Skriv ditt svar her

```
1 def get_number(string: str) -> str:
2     return string[:-1]
3
```

Maks poeng: 3

21

2b (3%) exam_score(...) - Totalpoeng

Først må vi få på plass hvor mange poeng en kandidat har fått totalt på eksamen. Det betyr summen av alle poeng som finnes i dictionarien til denne kandidaten. Lag funksjonen `candidate_score(...)`.

- Funksjonen får inn som parameter dictionarien til kandidaten som skal gis totalpoeng.
- Funksjonen skal returnere totalpoengene til kandidaten som et heltall.
- To nøkler må filtreres ut av beregningen: `'name'` og `'total'` (som brukes senere i oppgavene).

Eksempel:

>>> kandidat1 = {'name':'Børge', '1a':4, '1b':6, '1c':3, '2a': 5, '2b': 7, '2c': 8, '3a': 5, '3b': 9}

>>> print(candidate_score(kandidat1))

47

Skriv ditt svar her

```python
def candidate_score(candidate: dict) -> int:
    score = 0
    for (key, value) in candidate.items():
        if key == "name" or key == "total": continue
        score += value

    return score
```

Maks poeng: 3

22

2c (3%) update_scores(...) - Oppdater totalpoeng

Nå som vi har etablert hvordan man regner ut totalpoeng for én kandidat, da ønsker vi å beregne dette for alle. I tillegg skal totalpoengene lagres i hver kandidat sin dictionary. Lag funksjonen `update_scores(...)`.

- Funksjonen tar inn som parameter listen med kandidater, som definert øverst i oppgaven på din eksamen.
- Funksjonen beregner hver kandidat sine totalpoeng, og legger dette inn i dennes dictionary under nøkkelen `'total'`.
- Funksjonen skal ikke returnere noe spesifikt, men mutere listen den får inn.

Eksempel:

>>> update_scores(candidates)

>>> candidates[0]['total'] # Børge sine totalpoeng.

47

Skriv ditt svar her

```python
def update_scores(candidates: list[dict]) -> None:
    for candidate in candidates:
        score = candidate_score(candidate)
        candidate["total"] = score
```

Maks poeng: 3

23 2d (4%) grade_points(...) - Karakterberegning

Karakterer gis basert på kandidatens totalpoeng, sammenliknet med en oversikt over karaktergrensene. Disse er satt i en dictionary `grade_limits`. Skriv funksjonen `grade_points(...)`.

- Funksjonen har to parametre:
  - `grade_limits` - En dictionary med karakterskalaen, som vist i eksempelet under.
  - `points` - Poengene som skal gis karakter.
- Totalpoeng under grensen for `'E'` gis karakteren `'F'`.
- Funksjonen tar inn et heltall, som tilsvarer total poengskår på eksamen.
- Karaktergrensene er fra og med tallet som nevnes. `24` gir `'F'` mens `25` gir `'E'`.
- Funksjonen returnerer karakteren denne totalskåren gir, altså bokstavene `'A'` - `'F'`.
- Husk at dictionary husker rekkefølgen på nøklene.

Eksempel:

>>> grade_limits = {25: 'E', 32: 'D', 40: 'C', 50: 'B', 60: 'A'}

>>> print(grade_points(grade_limits, 45))

'C' # symbolene ' er ikke en del av strengen, det er kun tallbokstaven som skal returneres.

Skriv ditt svar her

```
# Antar at grade_limits er sortert i stigende rekkefølge som i eksempelet
def grade_points(grade_limits: dict[int, str], points: int) -> str:
    last_grade = "F"
    for (grade, limit) in grade_limits.items():
        if score < limit:
            return last_grade
        last_grade = grade
    return last_grade
```

Maks poeng: 4

24 2e (5%) - print_result(...) - Topp til bunn

Lag funksjonen `print_result(candidates_with_total, grade_limits)`.

- Funksjonen har to parametre:
  - `candidates_with_total` - Lista med kandidater, **med** totalpoeng ferdig utregnet.
  - `grade_limits` - Karakterskalaen.
- Funksjonen skal ikke returnere noe, men skrive ut resultatlisten.
- Funksjonen skal skrive ut karakterene `A` - `F` på hver sin linje, slik: '`A:\<eventuelle navn\>`'
- Alle karakterer skal skrives ut, selv om ingen kandidater får denne karakteren.
- Rekkefølgen skal være `A` som første linje, `F` som siste.
- Etter hver karakter skal alle kandidater som har fått denne karakteren listes opp, kommaseparert. Rekkefølge uviktig.
- Det er et krav at dictionarien `candidates_with_total`, som hentes inn som første parameter, **ikke** skal endres på.

Eksempel:

*>>> candidates_with_total = [{'name': 'Børge', 'total': 47}, {'name': 'Vetle', 'total': 57}, *

{'name': 'Mathea', 'total': 63}, {'name':'Marte', 'total':66}]

>>> print_results(candidates_with_total, grade_limits)

A: Vetle, Marte

B: Mathea

C: Børge

D:

E:

F:

Skriv ditt svar her

```python
def print_result(candidates: list[dict], grade_limits: dict[int, str]) -> None:
    # Create results dict
    results = {grade : [] for grade in grade_limits.values()}

    # Populate results dict
    for candidate in candidates:
        score = candidate["total"]
        name = candidate["name"]

        grade = grade_points(grade_limits, score)
        results[grade].append(name)

    # Print results dict
    grades = results.keys()
    for grade in grades.sort(): # Sorting just in case
        print(grade, end=": ")

        people = results[grade]
        for person in people[:-1]:
            print(person, end=", ")

        # Print last person seperatly for comma reasons
        if len(people) > 0:
            print(people[-1], end="")

        print()
```

Maks poeng: 5

25 2f (4%) filter_section(...) - Oppgavefiltrering

Skriv funksjonen `filter_section()`, som henter ut kandidatens oppgave/poeng-kombinasjon fra én seksjon.

- Denne funksjonen tar utgangspunkt i en dictionary av én kandidat slik den er beskrevet tidligere, altså den **uten** beregnet totalpoeng.
- Funksjonen tar inn to parametre:
  - `dict` - Dictionarien med kandidatens poeng per oppgave (uten totalpoeng).
  - `section` - En streng (seksjon) som hver oppgave må være i for at oppgaven skal bli med.
- Bare nøklene (oppgavene) som inneholder seksjonsnummeret gitt som andre parameter skal være med.
- Funksjonen skal **ikke** endre på den eksisterende dictionarien til kandidaten, men returnere en ny dictionary som inneholder de ønskede nøkkel/verdi-kombinasjonene.
- Rekkefølgen på innslagene i dictionarien er ikke viktig.
- Tips: Husk at hvis `'1'` angis som seksjon, så skal ikke oppgaver i seksjon 10 og slikt med.

Eksempel: (gitt `candidate` som én kandidat-dictionary fra listen `candidates`)

>>> candidate6 = {'name':'Børge', '1a': 4, '1b': 6, '1c': 3, '2a': 5, '2b': 7, '2c': 8, '3a': 5, '3b': 9, '10a': 5}

>>> print(filter_section(candidate6, '1'))

{'1a': 4, '1b': 6, '1c': 3}

>>> print(filter_section(candidate6, '10'))

{'10a': 5}

Skriv ditt svar her

```python
def filter_section(candidate: dict, section: str) -> dict[str, int]:
    section_results = {}
    for (key, value) in candidate.items():
        # Checking for total just in case
        if key == "name" or key == "total": continue

        if get_number(key) != section: continue

        section_results[key] = value

    return section_results
```

Maks poeng: 4

26 2g (7%) winner_per_section(...) - Seksjonsvinner(e)

Skriv funksjonen `winner_per_section(...)`. I denne funksjonen skal du gå igjennom datastrukturen og skrive ut hvilken kandidat som har skåret flest poeng per eksamensseksjon. En seksjon er definert ved at oppgavens nøkkel starter på samme tall, for eksempel er `1a` og `1b` oppgaver i samme seksjon mens `1a` og `2b`er oppgaver i ulike seksjoner.

- Funksjonen skal returnere resultatet som en todimensjonal liste.
- De indre listene representerer seksjonsdelen og skal ha formatet `[seksjonsnummer, vinnerpoeng, vinnernavn]`.
- Rekkefølgen på disse indre listene kan være vilkårlig.
- Alle nøkler i `candidates` består av oppgaver (som `'1a'`) samt `'name'` og `'total'`. De to siste skal ikke være med i beregningene!
- Løsningen skal håndtere et vilkårlig antall eksamensoppgaver fra ulike seksjoner.
- Du kan forutsette at alle kandidater har besvart i det minste én oppgave i hver del.
- Det er minst én kandidat som har besvart eksamen.
- Hvis det er flere vinnere i en seksjon skal alle legges til på slutten av den seksjonens indre liste, rekkefølge på kandidatene ikke viktig (se første indre liste i eksempelet under).

Eksempel:

>>> print(winner_per_section(candidates))

[['3', 18, 'Vetle', 'Mathea'], ['2', 26, 'Mathea'], ['1', 19, 'Mathea']]

Skriv ditt svar her

```python
def winner_per_section(candidates: list[dict]) -> list[list]:
    # Get all sections
    sections = []
    for (key, value) in candidates[0].items():
        if key == "total" or key == "name": continue

        section = get_number(key)
        if section not in sections:
            sections.append(section)

    # Calculate scores for each section
    results = []
    for section in sections:
        winners = []
        high_score = 0

        for candidate in candidates:
            scores = filter_section(candidate, section)
            score = sum(scores.values())
            if score > high_score:
                winners.append(candidate["name"])
                high_score = score

        results.append([section, high_score, *winners])

    return results
```

Maks poeng: 7

27 2h (6%) read_scores(...) - Filgreier

Skriv funksjonen `read_scores(file)`.

- Funksjonen skal returnere en datastruktur lik den som er oppgitt i starten av oppgaven:
  - En liste med dictionaries, der hver dictionary beskriver én kandidat.
  - Navnet på kandidaten skal være knyttet til nøkkelen `'name'`.
  - For oppgavene knyttet til en kandidat skal de ha nøkkel oppgavenummer (`'1a'`) og verdi poeng som heltall.
- Det eneste unntaket som kan oppstå under kjøring er tolking av poeng.
  - Hvis et poengfelt ikke lar seg gjøre om til et heltall skal verdien til denne oppgaven settes til `0`.
- Ellers kan du forutsette at filen er der og at andre feil unngås.
- Eksempel på filen 'grades.txt' som vil gi poengene som er brukt i oppgavene. (Merk at det kan finnes tomme linjer mellom hver kandidat):

<...> tom linje
name:Børge
1a:4
<...> tommer linjer
3a:5
3b:9
<...> Flere oppgave/poeng-linjer

name:Vetle
1a:5
<...> Flere oppgave/poeng-linjer
name:Mathea
1a:6
<...> Flere oppgave/poeng-linjer
<end of file>

Skriv ditt svar her

```python
def read_scores(filename: str) -> list[dict]:
    # Antar utf-8 encoding
    with open(filename, mode="r", encoding="utf-8") as file:
        lines = file.readlines()

    if not lines:
        print(f"Could not read file '{filename}'")
        return

    candidates = []
    candidate = {} # Just for readability, can be removed
    for line in lines:
        line = line.strip()

        if not line: continue # Ignore empty lines

        key, value = line.split(":")

        # New candidate
        if key == "name"
            candidates.append(candidate)
            candidate = {}
            candidate[key] = value
            continue

        # Calculate the score
        score = 0
        try:
            score = int(value)
        except ValueError:
            print(f"Could not create int from '{value}' for '{key}'")

        candidate[key] = value

    candidates.append(candidate)

    return candidates
```

Maks poeng: 6