UNIVERSITETET I BERGEN

KANDIDAT

568

PRØVE

INF100 0 Innføring i programmering

Emnekode	INF100
Vurderingsform	Skriftlig eksamen
Starttid	06.12.2021 08:00
Sluttid	06.12.2021 13:00
Sensurfrist	--
PDF opprettet	03.05.2024 10:55

Seksjon 1

Oppgave	Oppgavetype
ℹ	Informasjon eller ressurser
ℹ	Informasjon eller ressurser
ℹ	Informasjon eller ressurser

Seksjon 2

Oppgave	Oppgavetype
1	Paring

Seksjon 3

Oppgave	Oppgavetype
2	Nedtrekk
3	Fyll inn tekst
4	Nedtrekk
5	Nedtrekk
6	Nedtrekk
7	Nedtrekk
8	Nedtrekk
9	Nedtrekk
10	Nedtrekk
11	Nedtrekk

Seksjon 4

Oppgave	Oppgavetype
12	Programmering
13	Programmering
14	Programmering

1 Velg den passende datatype til de følgende uttrykk, der

- a = [1.1, 11]
- b = "11"
- c = 11
- d = b * a[1]

Finn de som passer sammen:

	str	bool	(-error-)	float	int	list
b*c	◉	○	○	○	○	○
len(a)	○	○	○	○	◉	○
a[1]	○	○	○	○	◉	○
c*a	○	○	○	○	○	◉
a*b	○	○	◉	○	○	○
f"{a}"	◉	○	○	○	○	○
a+b	○	○	◉	○	○	○
[d]	○	○	○	○	○	◉
c == "11"	○	◉	○	○	○	○
d	◉	○	○	○	○	○

Maks poeng: 5

2 Les inn hver linje fra en tekstfil og printe antallet bokstaver i linjen

filename = "foo.txt"

with (with, read, file, open) open(filename) (with(filename), read(filename),

filename, open(filename)) as f: (from f:, to f:, with f:, as f:)

- for line in f:
    - line = f.read() (line = line.strip(), line = line.split(), line = f.readline(), line = f.read()):
        - print(len(line))

Maks poeng: 2

3 Skriv inn navnet til en Exception-*type* slik at programmet printer "Unknown location". *Type* du velger må være så spesifikt som mulig:
f.eks. bruk "ZeroDivisionError", ikke bare "Exception", siden den kan også fange opp andre unntak.

```
location = {
  'Rick' : 'C-137',
  'Morty' : 'C500-A',
}
```

try:

- who = 'Summer'
- place = location[who]
- print(f"{who} is in {place}")

except KeyError :

- print("Unknown location")

except:

- print("Other error")

Maks poeng: 2

4 Velg slikt at alle sammenligninger er True. Dict xs ser sånn ut:

```
xs = {
'a' : 5,
'5' : 'hello',
7 : 3.1415,
5 : 7
}
```

xs['5'] (xs[a], xs[5], xs['a'], xs['5']) == 'hello'

'5' in xs.keys() (xs.keys(), xs.values(), xs.setdefault(), xs.items())

xs[5] + xs['a'] (len(xs['5']), xs[7] + xs[5], xs[12], xs[5] + xs['a'])== 12

len(xs['5']) (xs[5], xs[7], len(xs[5]), len(xs['5'])) == xs['a']

Maks poeng: 2

5 Returner True hvis **kun** det første elementet fra *input* listen er oddetall, ellers returner False

def only_first_is_odd(input):

- x = input[0]
- if `x % 2 == 0` (x // 2 == 1, x // 2 == 0, x % 2 == 1, x % 2 == 0) :
  - return `False` (True, False)
- for e in input[1:]:
  - if e % 2 == 1:
    - `return False` (continue, break, return False, return True)
- return `True` (True, False)

Maks poeng: 2

6 Spør om 7 ord og lage en string fra deres første bokstavene

initials = ""

`for _ in range(7):` (for initials in range(7):, while False:, while True:, for _ in range(7):)

- text = `input("Text: ")` (open("Text: "), input("Text: "), read("Text: "), print("Text: "))
- `initials += text[0]` (initials += text[0], initials = text[0], text += initials, initials + text[0])

print(f"The texts had `{ initials }`({ initials }, (initials), (text), { text }) as first letters.")

Maks poeng: 2

7 Velg de riktige verdiene til hver rad

a	b	a and b	a or b
True	True	True (True, False)	True (False, True)
True	False	False (False, True)	True (False, True)
False	False	False (True, False)	False (True, False)
5 > 3	5 > 7	False (True, False)	True (False, True)

Maks poeng: 2

8 Hvor ofte finnes det *letter* i strengen *word?*

def count(word, letter):

- ct = 0
- for i in word:
  - if i == letter:
    - ct += 1 (ct = letter, ct += i, ct += 1, ct = 1)
- return ct (ct, i, letter, word)

Maks poeng: 2

9 Velg de riktige linjene slik at resultatet av dette programmet er
A
B
C
og det kjører uten feil.

a = 450

if a < 500: (if 'a' > 500:, if a > 500:, if a < 500:, if 'a' < 500:)

- print('A')

if a > 250: (else:, if a > 250:, elif a < 250:, elif a > 250:)

- print('B')

if a % 10 == 0: (elif a % 10 == 0:, if a % 10 == 0:, elif a % 10 != 0:, if a % 10 != 0:)

- print('C')

elif a < 500: (if 'a' < 500:, elif:, elif a < 500:, if a < 500:)

- print('D')

Maks poeng: 2

10 Skriv om denne løkken med *while* i stedet for *for*:

word = "blåbærsyltetøy"
sum = 0
for letter in word:

- if letter in "æøå":
  - sum += 1

---

word = "blåbærsyltetøy"

i = 0 (i = 0, i = len(word), i = word, i = None)
sum = 0

while i < len(word): (sum < word:, i < word:, i < len(word):, sum < len(word):)

- letter = word[i] (letter = word[i], i = len(word), letter = word[sum], sum = word[letter])
- if letter in "æøå":
  - sum += 1

- i += 1 (word += 1, i += 1, sum += 1, return sum)

Maks poeng: 2

11 Velg slik at alle sammenligninger er True. Listen xs ser slik ut:
xs = ["hei", [12, 13], False, 1.3]

xs[0] (xs[0:1], xs[-1], xs[1], xs[0]) == 'hei'

'e' == xs[0][1] (xs[0:1], xs[0 1], xs[0,1], xs[0][1])

xs[-2] (xs[-2], xs[-1], xs[-3], xs[0])== False

len(xs[1]) (len(xs[1]), len(xs[2]), len(xs), len(xs[0])) == 2

Maks poeng: 2

12 Stortinget har 169 mandater som må fordeles mellom 19 valgdistrikter. Distrikter med høyere befolkningstall og/eller større areal får flere mandater enn de som er mindre.

Metoden er beskrevet slik i valgloven §11-3:

(2) Hvert valgdistrikts fordelingstall fastsettes ved at antallet innbyggere i valgdistriktet [...] adderes med antall kvadratkilometer i valgdistriktet multiplisert med 1,8.

(3) Hvert valgdistrikts fordelingstall divideres med 1 – 3 – 5 – 7 osv. De kvotienter som fremkommer, nummereres fortløpende. Representantplassene fordeles på valgdistriktene på grunnlag av de fremkomne kvotientene. Representantplass nr. 1 tilfaller det valgdistriktet som har den største kvotienten. Representantplass nr. 2 tilfaller det valgdistriktet som har den nest største kvotienten, osv. [...]

Denne tildelingsmetoden ligner nesten den for valgresultater som vi brukte i uke 8. Her er fasiten til denne oppgaven: https://folk.uib.no/dgr061/uke_08_oppg_5.py som du kan tilpasse hvis du vil, men da må ubrukte deler fjernes for å få en oversiktlig løsning.

CSV-filen https://folk.uib.no/dgr061/valgdistrikt_2020-01-01.csv inneholder data om distriktene: navn, antallet innbyggere, areal i kvadratkilometer. Filen er i formatet UTF-8.

Oppgave

- Lag en *funksjon* **mandatfordeling(filnavn, antall_mandater)** som tar som argument et filnavn til en CSV-fil med data om distriktene og antallet mandater som skal fordeles. Funksjonen skal returnere et *dict* (eller collections.*Counter)* som viser distriktnavn som *key* og antallet mandater som distriktet fikk, som *value.*

- I *hovedprogrammet* skal selve funksjonen kalles **mandatfordeling("valgdistrikt_2020-01-01.csv", 169) .** Resultatet skal så printes fra høyest til færrest, med pen formatering, og med tallene justert til høyre:

  Distrikt Mandater
  ========== (denne understreken har lengden 22)
  Oslo 20
  Akershus 19
  ...
  ...
  ...

.

Skriv ditt svar her

```
1
2
3  def mandatfordeling(filnavn, antall_mandater):
4
5      output = ""
6      with open(filnavn, encoding='utf-8') as f:
7          output = f.readlines()
8
9      mandat_antall = {}
10     valgdistrikter = []
```

```
11
12      for row in output:
13          row = row.split(",")
14
15          #hopper over første raden
16          if row[0] == "Valgdistrikt":
17              continue
18
19          fordelingstall = int(row[1]) + (int(row[2]) * 1.8)
20
21          fordelt_tall_liste = []
22          #deler fordelingstallet med oddetall og legger dem i en liste
23          for i in range(1, antall_mandater, 2):
24              fordelt_tall_liste.append(fordelingstall / i)
25
26          row.append(fordelt_tall_liste)
27          valgdistrikter.append(row)
28
29          #legger alle valgdistrikter inn i en dict med 0 mandater til å starte med.
30          mandat_antall[row[0]] = 0
31
32      #sjekker gjennom alle mandater
33      for n in range(antall_mandater):
34          mandat_mottaker = []
35          for distrikt in valgdistrikter:
36              #finner distriktet med størst kvotient
37              if len(mandat_mottaker) == 0 or distrikt[3] > mandat_mottaker[1]:
38                  mandat_mottaker = (distrikt[0], distrikt[3])
39          #gir distriktet med størst kvotient 1 mandat
40          mandat_antall[mandat_mottaker[0]] += 1
41
42          for distrikt in valgdistrikter:
43              #finner distriktet som vant mandatet
44              if distrikt[0] == mandat_mottaker[0]:
45                  #fjærner den største kvotienten til distriktet
46                  distrikt[3].pop(0)
47                  break
48      return mandat_antall
49
50
51  def main():
52      mandat_antall = mandatfordeling("valgdistrikt_2020-01-01.csv", 169)
53
54      #lager en liste med alle keys og values i dictionaryen mandat_antall
55      mandat_antall_liste = [(distrikt, mandat_antall.get(distrikt)) for distrikt in
56      mandat_antall_liste.sort(key=lambda x: x[1])
57      mandat_antall_liste.reverse()
58
59      print(printer(mandat_antall_liste))
60
61
62  def printer(liste):
63      result = ""
64      headline = "Distrikt                    Mandater\n"
65      result += headline + "=" * len(headline) + "\n"
66
67      for distrikt in liste:
68          result += distrikt[0] + str(distrikt[1]).rjust(len(headline) - len(distrikt
69
70      return result
71
72
73  if __name__ == '__main__':
74      main()
75
```

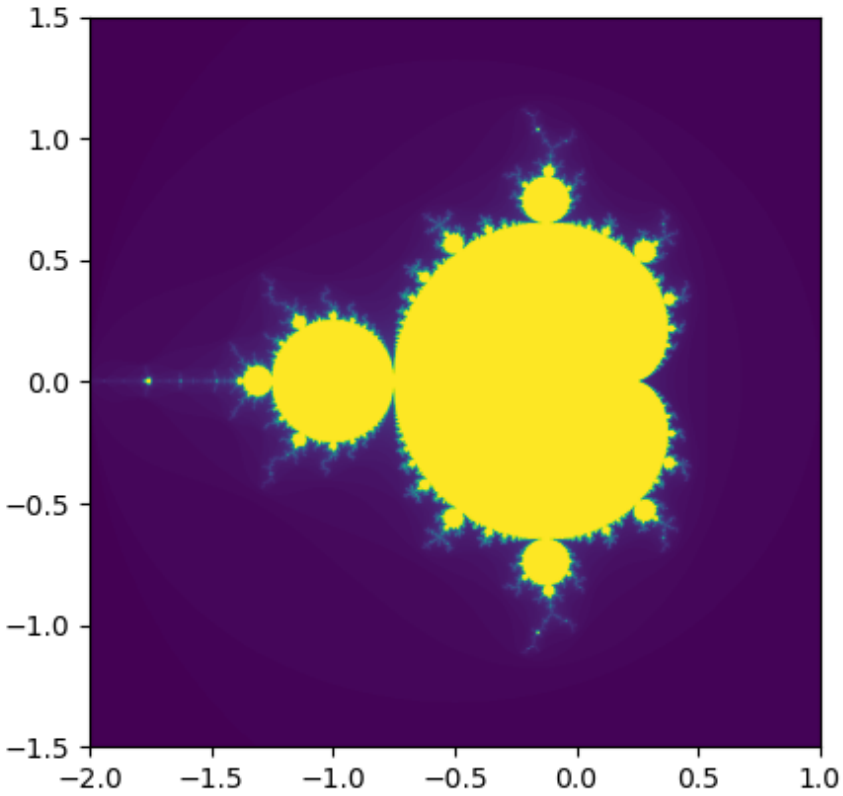
(2) Hvert valgdistrikts fordelingstall fastsettes ved at antallet innbyggere i valgdistriktet [...] adderes med antall kvadratkilometer i valgdistriktet multiplisert med 1,8.

(3) Hvert valgdistrikts fordelingstall divideres med 1 – 3 – 5 – 7 osv. De kvotienter som fremkommer, nummereres fortløpende. Representantplassene fordeles på valgdistriktene på grunnlag av de fremkomne kvotientene. Representantplass nr. 1 tilfaller det valgdistriktet som har den største kvotienten. Representantplass nr. 2 tilfaller det valgdistriktet som har den nest største kvotienten, osv. [...]

Maks poeng: 20

13 Last ned filen https://folk.uib.no/dgr061/mandelbrot.py . Du skal **tilpasse filen** for å løse de etterfølgende oppgavene (det er 3 oppgaver).

Filen genererer dette Mandelbrot-fraktalet som et png-bilde. x-verdiene går fra -2 til 1, y-verdiene fra -1.5 til 1.5, og bildet har 1000x1000 piksler:



Oppgaver (lim inn **ett program** som gjennomfører alle deler**)**

(1) Istedenfor at alt skjer i hovedprogrammet, legg inn alt som er relevant i en funksjon **mandel(x, y, size, pixels, filename)** som tar inn 5 argumenter og skal bruke plt.savefig til å lagre en bildefil. Funksjonen skal ikke returnere noe.
Argumentene **x** og **y** er koordinater til det nedre venstre hjørnet,
size er avstanden fra x/ymin til x/ymax,
pixels er antallet piksler i hver retning, og
filename er navnet til filen som skal lagres.

(bildet ovenfor tilsvarer **mandel(-2., -1.5, 3., 1000, "mandelbrot.png")**)

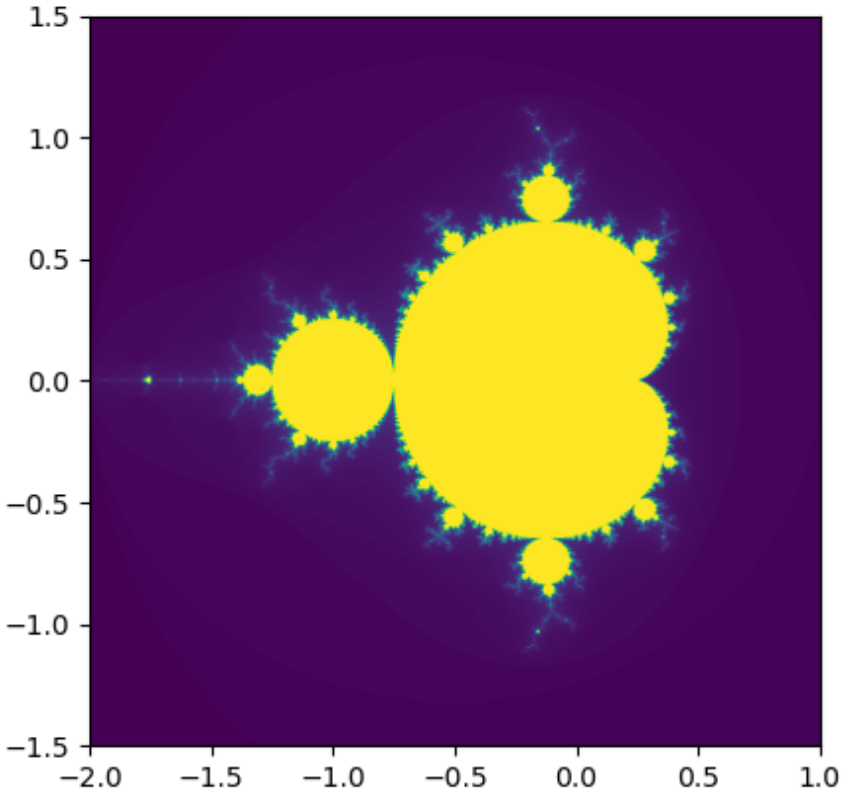
(2) Lag en funksjon
mandel_zoom(old_x, new_x, old_y, new_y, old_size, new_size, pixels, num_steps)
som ikke returnerer noe, men lagrer en sekvens av **num_steps** png-bilder med navn fra zoom_01.png frem til zoom_NN.png, der NN tilsvarer num_steps. Du skal kalle mandel()-funksjonen fra del 1 for hvert bilde:

Det første bildet skal tilsvare **mandel(old_x, old_y, old_size, pixels, "zoom_01.png")**

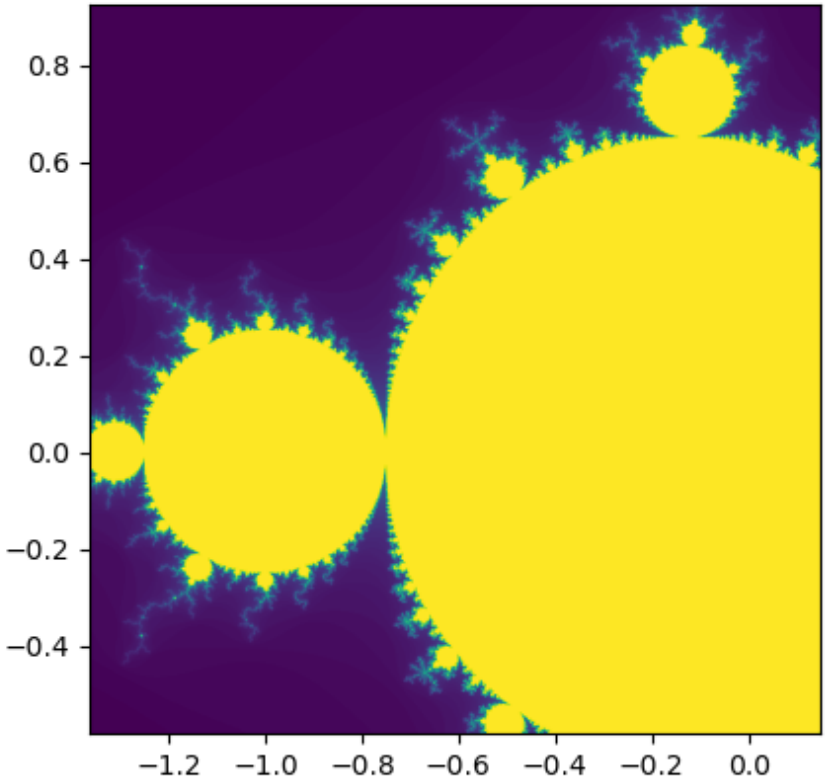
Det siste bildet skal tilsvare **mandel(new_x, new_y, new_size, pixels, "zoom_NN.png")**
Bildene imellom skal ta **x, y** og **size** fra en lineær interpolasjon mellom old_x og new_x, old_y og new_y, old_size og new_size. (Her kan du godt bruke np.linspace())

Eksempel: mandel_zoom(-2, -0.725, -1.5, 0.335, 3, 0.02, 1000, 3) lagrer 3 filer

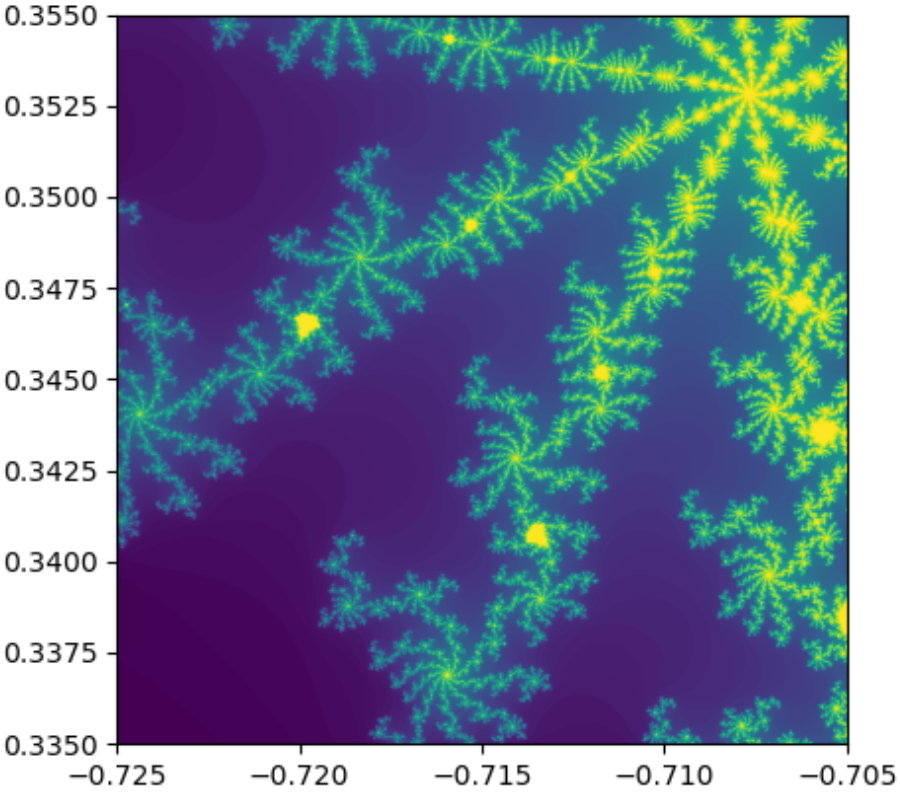
zoom_01.png



zoom_02.png


0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−1.2
−1.0
−0.8
−0.6
−0.4
−0.2
0.0

zoom_03.png


0.3550
0.3525
0.3500
0.3475
0.3450
0.3425
0.3400
0.3375
0.3350
−0.725
−0.720
−0.715
−0.710
−0.705

(3) Til slutt, skriv et nytt hovedprogram som spør brukeren med input() om antallet piksler og antallet zoom-bilder. Bruk exception-håndtering for å sjekke at brukeren skriver to heltall (integers). Om brukeren skriver noe annet enn tall, skal spørsmålet gjentas.

Så skal hovedprogrammet kalle funksjonen mandel_zoom(-2, -0.725, -1.5, 0.335, 3, 0.02, pixels, num_images) der **pixels** og **num_images** er de to tallene fra brukeren.

Lim inn den endelige koden som fullfører alle deler

```
import numpy as np
import matplotlib.pyplot as plt


def main():
    #sjekker at input er int:
    while True:
        num_pixels = input("Hvor mange pixeler vil du ha?: ")
        try:
            num_pixels = int(num_pixels)
        except ValueError:
            print("input må være en int.")
            continue
        break

    while True:
        num_images = input("Hvor mange zoom bilder vil du ha?: ")
        try:
            num_images = int(num_images)
        except ValueError:
            print("input må være en int.")
            continue
        break

    mandel_zoom(-2, -0.725, -1.5, 0.335, 3, 0.02, num_pixels, num_images)


def mandel(x, y, size, pixels, filename):
    # initial setup of calculation constant C for each pixel
    X = np.linspace(x, x + size, pixels)[None, :]
    Y = np.linspace(y, y + size, pixels)[:, None]
    C = X + 1j * Y
    # start value of Z is always 0
    Z = np.zeros_like(C)
    # P counts iterations, this is what we plot at the end
    P = np.zeros_like(C, dtype='uint8')  # unsigned int 0..255

    # iteration of Z <- Z*Z + C
    for i in range(120):
        # print(f"Iteration {i}")
        # which elements are still "live"?
        live = np.abs(Z) < 2.
        # update live pixels with current iteration number
        P[live] = i
        # iterate
        Z[live] = Z[live] * Z[live] + C[live]

    plt.imshow(
        P,
        origin='lower',
        extent=(X.min(), X.max(), Y.min(), Y.max())
    )
    plt.savefig(filename)
```

```
def mandel_zoom(old_x, new_x, old_y, new_y, old_size, new_size, pixels, num_steps):
    x = np.linspace(old_x, new_x, num_steps)
    y = np.linspace(old_y, new_y, num_steps)
    size = np.linspace(old_size, new_size, num_steps)

    for i in range(num_steps):
        filename = f"zoom_{i + 1}.png"
        mandel(x[i], y[i], size[i], pixels, filename)


if __name__ == '__main__':
    main()
```

Maks poeng: 25

14 Last ned filen https://folk.uib.no/dgr061/football.py . Du skal **tilpasse kun de markerte delene i filen** for å løse de etterfølgende deloppgavene. Denne oppgaven har 2 deler som kan løses hver for seg.

Vi skal se på noen resultater fra gruppefasen fra ulike fotball-VM. I hver gruppe finnes det 4 landslag som spiller 6 kamper mot hverandre. Kampresultatene er gitt som en liste av 4-tuples, med 2 strings og 2 int:

```
[
  ("Brazil", "Scotland", 2, 1),
  ("Morocco", "Norway", 2, 2),
  ("Scotland", "Norway", 1, 1),
  ("Brazil", "Morocco", 3, 0),
  ("Brazil", "Norway", 1, 2),
  ("Scotland", "Morocco", 0, 3),
]
```

For hvert lag kan vi nå lage en statistikk med 4 tall:

- **GF** (goals for) - antallet mål som laget har skåret
- **GA** (goals against) - antallet mål som ble skåret **mot** laget
- **GD** (goal difference) - differansen mellom GF og GA, alltid lik GF-GA
- **PT** (points) - poengsummen: 3 poeng til kampvinneren, 1 poeng til hvert lag om kampen er uavgjort

Denne statistikken skal lagres i en datastruktur som er et dict av dicts. Til eksemplet ovenfor ser det sånn ut:

```
{
  'Brazil': {'GF': 6, 'GA': 3, 'GD': 3, 'PT': 6},
  'Scotland': {'GF': 2, 'GA': 6, 'GD': -4, 'PT': 1},
  'Morocco': {'GF': 5, 'GA': 5, 'GD': 0, 'PT': 4},
  'Norway': {'GF': 5, 'GA': 4, 'GD': 1, 'PT': 5},
}
```

Denne strukturen er **ikke sortert** ennå, en tilfeldig rekkefølge er greit!

Deloppgave 1 (make_stats)

Lag en funksjon **make_stats(matches) ,** der **matches** er en liste med kampresultater i det formatet vist ovenfor. Funksjonen skal returnere et dict med statistikken i det nevnte dict-formatet. Det finnes et skjelett til make_stats() allerede i filen, med en liten if-test som sjekker funksjonen din.

Selv om du ikke klarer deloppgave 1, kan du fjerne 'exit()' på linje 44, og prøve å løse deloppgave 2.

Deloppgave 2 (compare)

Her skal vi sortere lagene i en tabell for gruppen. Print-funksjoner osv. er allerede klare, det eneste som mangler er en **compare-**funksjon som avgjør rekkefølgen til lagene. Avgjørelsen skjer med flere regler, der vi bruker den neste når alle tidligere regler ikke gir én vinner:

(1) Større poengsum PT
(2) Større måldifferanse GD

(3) Flere skårete mål GF
(4) Vinneren av den direkte kampen mellom de to lagene
(vi ignorer en situasjon hvor det er flere enn to lag som er helt like når vi kommer til denne regelen)
(5) Trekking av lodd

Lag en funksjon **compare(matches, data, a, b)** der
matches er listen til kampresultatene i formatet ovenfor,
data er et ferdig dict med statistikk i formatet ovenfor,
a og **b** er to strenger med **navnene** til 2 landslag

(vi tar med *data* som argument slik at vi kan ignorere funksjonen i del 1.
Vi kan alltid anta at *data* == make_stats(*matches*)

Funksjonen skal returnere tallet **1** om lag **a** ligger høyere i tabellen enn lag **b**
Funksjonen skal returnere tallet **-1** om lag **b** ligger høyere i tabellen enn lag **a**
Funksjonen skal printe **f"LOTTERY {a} {b}"** og returnere tallet **0** om situasjonen er uavgjort mellom **a** og **b**

Det finnes et skjelett til compare() allerede i filen, med en rekke tester som sjekker funksjonen din.

Eksempel output

Om alt er bra i begge deler, ser outputtet slik ut (husk å fjerne exit() i linje 44). *Formateringen i kolonner kommer ikke frem pent i Inspera:*

---

make_stats looks good, you can remove the exit() line now!

```
            GF  GA  GD   PT
Brazil       6   3   3    6
Norway       5   4   1    5
Morocco      5   5   0    4
Scotland     2   6  -4    1
--------------------
This looks good!
```

```
            GF  GA  GD   PT
Nigeria      6   2   4    6
Bulgaria     6   3   3    6
Argentina    6   3   3    6
Greece       0  10 -10    0
--------------------
This looks good!
```

```
            GF  GA  GD   PT
Mexico       3   3   0    4
```

```
Ireland        2  2  0    4
Italy          2  2  0    4
Norway         1  1  0    4
--------------------
This looks good!


LOTTERY Ireland Netherlands
             GF  GA  GD   PT
England        2   1   1    5
Ireland        2   2   0    3
Netherlands    2   2   0    3
Egypt          1   2  -1    2
--------------------
This looks good!
```

Legg inn kode her, ikke output

```python
def make_stats(matches):
    teams = {}  # dict of dicts: name -> {GF, GA, GD, PT}

    teams[matches[0][0]] = {'GF': 0, 'GA': 0, 'GD': 0, 'PT': 0}
    teams[matches[0][1]] = {'GF': 0, 'GA': 0, 'GD': 0, 'PT': 0}
    teams[matches[1][0]] = {'GF': 0, 'GA': 0, 'GD': 0, 'PT': 0}
    teams[matches[1][1]] = {'GF': 0, 'GA': 0, 'GD': 0, 'PT': 0}

    for match in matches:
        #First team
        teams[match[0]]["GF"] += match[2]
        teams[match[0]]["GA"] += match[3]
        teams[match[0]]["GD"] = teams[match[0]]["GF"] - teams[match[0]]["GA"]

        #Second team
        teams[match[1]]["GF"] += match[3]
        teams[match[1]]["GA"] += match[2]
        teams[match[1]]["GD"] = teams[match[1]]["GF"] - teams[match[1]]["GA"]

        #Awarding points
        if match[2] > match[3]:
            teams[match[0]]["PT"] += 3
        elif match[2] < match[3]:
            teams[match[1]]["PT"] += 3
        else:
            teams[match[0]]["PT"] += 1
            teams[match[1]]["PT"] += 1
    return teams


# an example for the "matches" datastructure
FIFA_1998_A = [
```

```
    ("Brazil", "Scotland", 2, 1),
    ("Morocco", "Norway", 2, 2),
    ("Scotland", "Norway", 1, 1),
    ("Brazil", "Morocco", 3, 0),
    ("Brazil", "Norway", 1, 2),
    ("Scotland", "Morocco", 0, 3),
]

# make_stats(FIFA_1998_A) takes this ^^^^
# and the return value should be

FIFA_1998_A_data = {
    'Brazil': {'GF': 6, 'GA': 3, 'GD': 3, 'PT': 6},
    'Scotland': {'GF': 2, 'GA': 6, 'GD': -4, 'PT': 1},
    'Morocco': {'GF': 5, 'GA': 5, 'GD': 0, 'PT': 4},
    'Norway': {'GF': 5, 'GA': 4, 'GD': 1, 'PT': 5}
}

# checking the return value matches what we expect
if make_stats(FIFA_1998_A) == FIFA_1998_A_data:
    print("make_stats looks good, you can remove the exit() line now!\n\n\n")
else:
    print("Something is wrong in make_stats")


def compare(matches, data, a, b):
    """
    Compare two team _names_ a and b.
    matches is a list of match results.
    data is the result of make_stats(matches).

    If a is ranked higher, return 1
    If b is ranked higher, return -1
    If a and b are undecided, print f"LOTTERY {a} {b}" and return 0

    Sequence for ranking (go to the next rule if the two teams are equal)
    (1) The higher points score PT wins
    (2) The higher goal difference GD wins
    (3) The higher "goals for" GF wins
    (4) The winner of the direct match between the two teams
        (we will ignore the situation where 3 teams are equal)
    (5) Lottery draw
    """
    #finding the match a and b played together
    ab_match = [match for match in matches if match[0] == a and match[1] == b or r

    #Sjekker PT
    if data[a]["PT"] > data[b]["PT"]:
        return 1
    elif data[a]["PT"] < data[b]["PT"]:
        return -1

    #Sjekker GD
    elif data[a]["GD"] > data[b]["GD"]:
        return 1
    elif data[a]["GD"] < data[b]["GD"]:
        return -1

    #sjekker GF
    elif data[a]["GF"] > data[b]["GF"]:
        return 1
    elif data[a]["GF"] < data[b]["GF"]:
        return -1

    #Sjekker Direkte kamper:
    #If team A is the first name in the match and team a won.     or      team B i
    elif ab_match[0][0] == a and ab_match[0][2] > ab_match[0][3] or ab_match[0][0]
        ab_match[0][2]:
```

```
        return 1

    #If team A is the first name in the match and team b won.     or      team B i
    elif ab_match[0][0] == a and ab_match[0][2] < ab_match[0][3] or ab_match[0][0]
        ab_match[0][2]:
        #and B won
        return -1

    #Lottery
    else:
        print(f"LOTTERY {a} {b}")
        return 0

    ########## ^^^^^^ edit this part ^^^^^^^ ###############


#########################################################
############# nothing to edit below here #############
############ some more examples for testing ##########
#########################################################

FIFA_1994_D = [
    ("Argentina", "Greece", 4, 0),
    ("Nigeria", "Bulgaria", 3, 0),
    ("Argentina", "Nigeria", 2, 1),
    ("Bulgaria", "Greece", 4, 0),
    ("Argentina", "Bulgaria", 0, 2),
    ("Greece", "Nigeria", 0, 2),
]

FIFA_1994_D_data = {
    'Argentina': {'GF': 6, 'GA': 3, 'GD': 3, 'PT': 6},
    'Greece': {'GF': 0, 'GA': 10, 'GD': -10, 'PT': 0},
    'Nigeria': {'GF': 6, 'GA': 2, 'GD': 4, 'PT': 6},
    'Bulgaria': {'GF': 6, 'GA': 3, 'GD': 3, 'PT': 6},
}

FIFA_1994_E = [
    ("Italy", "Ireland", 0, 1),
    ("Norway", "Mexico", 1, 0),
    ("Italy", "Norway", 1, 0),
    ("Mexico", "Ireland", 2, 1),
    ("Italy", "Mexico", 1, 1),
    ("Ireland", "Norway", 0, 0),
]

FIFA_1994_E_data = {
    'Italy': {'GF': 2, 'GA': 2, 'GD': 0, 'PT': 4},
    'Ireland': {'GF': 2, 'GA': 2, 'GD': 0, 'PT': 4},
    'Norway': {'GF': 1, 'GA': 1, 'GD': 0, 'PT': 4},
    'Mexico': {'GF': 3, 'GA': 3, 'GD': 0, 'PT': 4},
}


FIFA_1990_F = [
    ("England", "Ireland", 1, 1),
    ("Netherlands", "Egypt", 1, 1),
    ("England", "Netherlands", 0, 0),
    ("Ireland", "Egypt", 0, 0),
    ("England", "Egypt", 1, 0),
    ("Ireland", "Netherlands", 1, 1),
]

FIFA_1990_F_data = {
    'England': {'GF': 2, 'GA': 1, 'GD': 1, 'PT': 5},
    'Ireland': {'GF': 2, 'GA': 2, 'GD': 0, 'PT': 3},
    'Netherlands': {'GF': 2, 'GA': 2, 'GD': 0, 'PT': 3},
    'Egypt': {'GF': 1, 'GA': 2, 'GD': -1, 'PT': 2},
}
```

```
168
169 from functools import cmp_to_key, partial
170
171
172 def pretty(order, data):
173     print(f"{' ':14} {'GF':>3} {'GA':>3} {'GD':>3}   {'PT':>3}")
174     for team in order:
175         k = team
176         v = data[team]
177         print(f"{k:14} {v['GF']:3d} {v['GA']:3d} {v['GD']:3d}   {v['PT']:3d}")
178     print("-" * 20)
179
180
181 all_groups = [FIFA_1998_A, FIFA_1994_D, FIFA_1994_E, FIFA_1990_F]
182 all_data = [FIFA_1998_A_data, FIFA_1994_D_data, FIFA_1994_E_data, FIFA_1990_F_data
183
184 expected = [
185     ['Brazil', 'Norway', 'Morocco', 'Scotland'],
186     ['Nigeria', 'Bulgaria', 'Argentina', 'Greece'],
187     ['Mexico', 'Ireland', 'Italy', 'Norway'],
188     ['England', 'Ireland', 'Netherlands', 'Egypt'],
189 ]
190
191 for matches, data, ex in zip(all_groups, all_data, expected):
192     order = sorted(data, key=cmp_to_key(partial(compare, matches, data)), reverse=
193     pretty(order, data)
194     if order == ex:
195         print("This looks good!\n\n\n")
196     else:
197         print(f"!!! expected {ex}, but got {order}")
198
```

Maks poeng: 30