



# Prøveeksamen

Spørsmålene (uten fasit) finner du som [pdf](#).

## Løsningsforslag og sensorveiledning

- Automatisk rettet (multiple choice)
  - [Oppgave 1: typer](#)
  - [Oppgave 2: telling](#)
  - [Oppgave 3: logiske operasjoner](#)
  - [Oppgave 4: lister](#)
  - [Oppgave 5: oppslagsverk](#)
  - [Oppgave 6: 2d-liste](#)
  - [Oppgave 7: presedens](#)
- Korte kodesnutter
  - [Oppgave 8: alias](#)
  - [Oppgave 9: liste med liste med liste](#)
  - [Oppgave 10: prosentvis endring](#)
- Forklaringer
  - [Oppgave 11: destruktive funksjoner i snake](#)
  - [Oppgave 12: feil i funksjon som finner to like på rad](#)
- Kodeoppgaver
  - [Oppgave 13: tegne en stige](#)
  - [Oppgave 14: whac-a-mole](#)
  - [Oppgave 15: CSV-håndtering av fylker](#)

## Oppgave 1: typer

0.6 poeng per riktig svar, totalt 6 mulige poeng.

```
# Kjør denne koden for å se fasiten
a = '123'
b = 123
c = [1, 2, 3]
d = 1.23

# Oppgave: velg riktig datatype for hvert av uttrykkene:
expressions = [
    "a*b",
    "len(c)",
```

```
'f"{c}"',  
"c == 10.3",  
  
"a*d",  
"b*c",  
'a + "b"',  
"a+b",  
"a+a",  
"[a]",  
]  
  
def find_type(expression):  
    try:  
        return type(eval(expression))  
    except:  
        return "(-error-)"  
  
# Skriv ut løsning:  
for expression in expressions:  
    print(f"{expression:10} -> ", find_type(expression))
```

[Kopier](#)[Se steg](#)[Kjør](#)

## Oppgave 2: telling

1 poeng per riktig svar, men 4 poeng dersom begge svar var riktige.

Løsningsforslag:

```
def count(xs, x):  
    ct = 0  
    for i in xs:  
        if i == x:  
            ct += 1  
    return ct  
  
assert(3, count([1, 2, 3, 4, 3, 2, 3, 2, 1], 2))
```

[Kopier](#)[Se steg](#)[Kjør](#)

## Oppgave 3: logiske operasjoner

0.5 poeng per riktig svar, totalt 4 mulige poeng.

```
# Kjør koden for å se fasiten  
exp1 = "a and (b or c)"  
exp2 = "a or (not b)"
```

```
scenarios = [  
    # (a, b, c)  
  
    (True,  True,  True),  
    (False, False, True),  
    (False, True,  True),  
    (True,  False, False),  
]  
  
print(f"{'a':7}{ 'b':7}{ 'c':10}{exp1:20}{exp2}")  
for a, b, c in scenarios:  
    res1 = eval(exp1)  
    res2 = eval(exp2)  
    print(f"{'str(a):7'}{'str(b):7'}{'str(c):10'}{'str(res1):20'}{'str(res2)}")
```

[Kopier](#)[Se steg](#)[Kjør](#)

## Oppgave 4: lister

1 poeng per riktig svar, totalt 4 poeng

```
xs = ["hallo", [12, 13, 14], False, 3, 1.3]  
  
print(xs[0] == 'hallo')  
print(13 == xs[1][1])  
print(xs[-3] == False)  
print(len(xs[1]) == 3)
```

[Kopier](#)[Se steg](#)[Kjør](#)

## Oppgave 5: oppslagsverk

1 poeng per riktig svar, totalt 4 poeng

```
xs = {  
    'a': 5,  
    '5': 'hello',  
    'hello': 3.1415,  
    7: 'a',  
    '7': 0  
}  
  
print(xs['5'] == 'hello')  
print(7 in xs.keys())  
print(xs[xs['5']] == 3.1415)  
print(len(xs['5']) == xs['a'])
```

## Oppgave 6: 2d-liste

2 poeng for riktig svar

```
# Klikk "se steg" for å vise at dette er riktig bilde
# Kjør programmet for å se riktig svar.

a = [[1, 2], [3, 4]]
print(a[1])
```

## Oppgave 7: presedens

4 poeng for riktig svar.

```
def question(a, b, c, y, z):
    return a or b and y < z or c

def answer(a, b, c, y, z):
    # Her er fasit
    return (a or (b and (y < z))) or c

# Sjekk at dette funker
print("Sjekker at question og answer er like... ", end="")
for a in range(-5, 5):
    for b in range(-5, 5):
        for c in range(-5, 5):
            for y in range(-5, 5):
                for z in range(-5, 5):
                    expected = question(a, b, c, y, z)
                    actual = answer(a, b, c, y, z)
                    assert(expected == actual)

print("OK")
```

## Oppgave 8: alias

- 4 poeng for riktig svar.
  - 2 poeng dersom a og b er aliaser
  - 2 poeng hvis a er en liste med elementene 1, 2, og 3.

```
# Klikk på 'se steg' for å verifisere at vi får riktig bilde.
```

```
a = [1, 2, 3]
b = a
```

Kopier

Se steg

Kjør

## Oppgave 9: liste med liste med liste

- 4 poeng for riktig svar.
  - 1 poeng hvis a er en liste med 1, 2, 3
  - 1 poeng hvis a er et element i b
  - 2 poeng hvis c er riktig

*# Klikk på 'se steg' for å verifisere at vi får riktig bilde.*

```
a = [1, 2, 3]
b = [a, 2, 3]
c = [a, b, 3]
```

Kopier

Se steg

Kjør

## Oppgave 10: prosentvis endring

- 10 poeng for riktig svar.
  - Det er ikke nødvendig å bruke løkker for å få full uttelling
  - Det er ikke nødvendig å skrive ut svaret for å få full uttelling, så lenge svaret er lagre i x eller skrives ut, er det godkjent.
  - 3 poeng for å gange med 1.5 (eller  $(100 + 50) / 100$ ) (1 poeng for hver gang)
  - 3 poeng for å gange med 0.5 (eller  $(100 - 50) / 100$ ) (1 poeng for hver gang)
  - 4 poeng for å binde alt sammen

```
x = 100

for _ in range(3):
    x = x * 1.5
    x = x * 0.5

print(x)
```

Kopier

Se steg

Kjør

## Oppgave 11: destruktive funksjoner i snake

- 10 poeng
  - 2 poeng: en destruktiv funksjon muterer/endrer på en verdi gitt som argument til funksjonen. Denne endringen vil påvirke variablene hos den som kaller på funksjonen også.
  - 2 poeng: Korrekt eksempel på minst én destruktiv funksjon i snake

- 2 poeng: Korrekt eksempel på minst en destruktiv funksjon i snake
- 2 poeng: Korrekt eksempel på minst én ikke-destruktiv funksjon i snake
- 4 poeng: Skjønnsmessig vurdering. Bruker kandidaten gode faguttrykk? Er det sammenheng mellom resonnementer?

Eksempelbesvarelse:

En **destruktiv** funksjon er en funksjon med sideeffekter: den muterer minst ett av argumentene som blir gitt til funksjonen. I snake.py er for eksempel funksjonen **subtract\_one\_from\_all\_positives** en destruktiv funksjon, siden den muterer listen den blir gitt som argument. Selv om denne funksjonen ikke har noen retur-verdi, har den en sideeffekt: den trekker fra 1 på mange posisjoner i en 2d-liste. Denne endringen kan observeres etter at funksjonskallet er ferdig, og funksjonen er derfor destruktiv. Andre destruktive funksjoner i snake.py som er gode eksempler: **add\_apple\_at\_random\_location**, **move\_snake**, **key\_pressed**, **timer\_fired**, **init**, og **app\_started**

En **ikke-destruktiv** funksjon er en funksjon uten sideeffekter: den eneste måten man kan ha utbytte av å kalle på en ikke-destruktiv funksjon er ved å benytte seg av retur-verdien til funksjonen. Ikke-destruktive funksjoner har den fordel at de er lettere å feilsøke og ressonere rundt. Gode eksempler på destruktive funksjoner i snake.py er **new\_default\_board**, **get\_max\_value\_in\_2dlist**, **position\_of\_value\_in\_2dlist**, **get\_next\_head\_position**, **is\_legal\_move**, og **get\_color**.

## Oppgave 12: feil i funksjon som finner to like på rad

- 10 poeng
  - 3 poeng: Koden returnerer for tidlig
  - 2 poeng: Blanding av indekser og elementer
  - 1 poeng: For mange iterasjoner
  - 1 poeng: Syntaks-feil (men trenger ikke nevne dette dersom man nevner at koden returnerer for tidlig, da tildeles dette poenget automatisk i tillegg)
  - 3 poeng: Skjønnsmessig vurdering. Bruker kandidaten gode faguttrykk? Er det sammenheng mellom resonnementer?

```
def has_consecutive_elements(a):
    for i in a:
        if a[i] == a[i+1]:
            return True
        else:
            return False

assert(has_consecutive_elements([1, 3, 3, 4]))
assert(not has_consecutive_elements([1, 3, 4, 3]))
assert(not has_consecutive_elements([3, 1, 4, 3]))
```

```
assert(has_consecutive_elements([3, 3, 1, 4]))
assert(has_consecutive_elements([1, 4, 3, 3]))
```

Kopier

Se steg

Kjør

## Eksempelbesvarelse:

*Koden over har flere feil.*

- *Det er en syntaksfeil på linje 5. Det mangler kolon etter else.*
- *Koden returnerer for tidlig. Dersom det ikke stemmer at de to første elementene er like, vil funksjonen alltid returnere False. Så fort koden utfører return-setningen, avbrytes nemlig løkken og funksjonen er ferdig. Vi kan derfor ikke returnere False inne i løkken, siden vi ikke vet om svaret er False før vi har sjekket alle posisjoner. For å fikse problemet, kan vi flytte `return False` ut av løkken og heller ha den som siste setning i funksjonen etter at løkken er ferdig.*
- *Koden blander indekser og elementer i løkken over `a`. Iteranden `i` brukes som om det er en indeks (f. eks. `a[i]` og `a[i+1]`), men i denne typen for-løkke er iteranden et element i listen `a`. For å bruke en løkke over indekser, endre løkken slik at den er over `range(len(a))` i stedet for å være over `a`.*
- *Koden krasjer fordi man forsøker å sammenligne `a[i]` og `a[i+1]` også når `i` er den siste posisjonen i listen. For å reparere dette, kan vi endre løkken slik at den slutter når `i` peker på nest siste posisjon. Løkken blir da over `range(len(a) - 1)`.*

## Oppgave 13: tegne en stige

- 10 poeng
  - 3 poeng for å bruke en løkke for å tegne stigetrinnene.
    - Disse poengene deles ut uavhengig av om resten fungerer eller ikke.
    - Dersom løkke brukes, men det er problemer med selve løkken eller utregninger viktige i løkken, reduseres poengene for dette punktet basert på en skjønnsmessig vurdering.
  - 1 poeng for å ha lignende formattering (strekene har tykkelse og farge – hvilken farge er ikke viktig)
  - 6 poeng dersom tegningen fungerer
    - Dersom tegningen ikke fungerer, deles det ut maksimalt 4 poeng basert på hvor nært koden er til å fungere. Dersom koden er svært nært å fungere deles det altså ut 4 poeng. Dersom koden ikke demonstrerer å være i nærheten av funksjonell, deles det ut 0-3 poeng basert på en skjønnsmessig vurdering.
- Plassering av stigen i bildet betyr ingenting
- Avstanden mellom stigetrinnene bør være lik
- Dersom det benyttes løkke, er det to logiske måter å gjøre det på som begge blir fullt ut godkjent. Den ene er (som under) å fordele stigetrinnene jevnt over et område med en fast størrelse; det andre er at stigen blir større hvis den har flere trinn

størrelse, det andre er at stigen blir større hvis den har flere trinn.

- Det gjøres ingen forskjell på om man bruker `create_line` eller `create_rectangle` for å tegne med.
- Det gjøres ingen forskjell på om avstanden fra starten av stigen til første trappetrinn er ulik avstanden mellom trinnene, så lenge den ikke begynner nøyaktig samme sted (i så fall -1 poeng).

```
from uib_inf100_graphics import *

def draw_ladder(canvas, x_left, y_top, x_right, y_bottom, steps, col):
    # Strekene på sidene
    canvas.create_line(x_left, y_top, x_left, y_bottom, fill=col, width=5)
    canvas.create_line(x_right, y_top, x_right, y_bottom, fill=col, width=5)

    # Trinnene på stigen
    y_dist = (y_bottom - y_top)/(steps+1)
    for i in range(steps):
        y = y_top + (i + 1)*y_dist
        canvas.create_line(x_left, y, x_right, y, fill=col, width=5)

def redraw_all(app, canvas):
    draw_ladder(canvas, 100, 50, 200, 350, 6, "red")

run_app(width=300, height=400)
```

[Kopier](#)

## Oppgave 14: whac-a-mole

- 14 poeng
  - 10 poeng for standard utførsel.
    - 1 poeng: variabel for muldvarpen initialiseres i `app_started`
    - 1 poeng: variabel for musens posisjon initialiseres i `app_started`
    - 1 poeng: poeng initialiseres i `app_started`
    - 1 poeng: finner tilfeldig posisjon for muldvarpen i `app_started`
    - 1 poeng: finner tilfeldig posisjon for muldvarpen i `mouse_pressed` under riktige betingelser.
    - 1 poeng: `mouse_moved` kaller `get_cell`-funksjonen og oppdaterer musens posisjon.
    - 1 poeng: muldvarpen tegnes
    - 1 poeng: museposisjonen tegnes
    - 1 poeng: poengene tegnes
    - 1 poeng: skjønnsmessig vurdering
  - 4 poeng for ekstra utfordring: timer og nedtelling. 4 poeng hvis det fungerer fullt ut. Skjønnsmessig reduksjon hvis timeren fungerer delvis.



## Løsningsforslag (10 poeng)

```
# Her vises kun de funksjonene som måtte endres ift.
# kursnotatene sitt eksempel om klikking i rutenett.
# Med andre ord, i tillegg til denne koden; inkluder
# funksjonene point_in_grid, get_cell, get_cell_bounds
# fra kursnotatene.

def app_started(app):
    app.rows = 5
    app.cols = 8
    app.margin = 50 # margin rundt rutenettet

    app.mole_position = (-1, -1) # (row, col), men (-1,-1) betyr "ingen rute"
    app.mouse_is_over = (-1, -1) # (row, col), men (-1,-1) betyr "ingen rute"
    app.points = 0
    place_mole_randomly(app)

def place_mole_randomly(app):
    # velg en tilfeldig rute
    row = random.choice(range(app.rows))
    col = random.choice(range(app.cols))
    app.mole_position = (row, col)

def mouse_pressed(app, event):
    (row, col) = get_cell(app, event.x, event.y)
    if (row, col) == app.mole_position:
        app.points += 1
        place_mole_randomly(app)

def mouse_moved(app, event):
    app.mouse_is_over = get_cell(app, event.x, event.y)

def redraw_all(app, canvas):
    # poengsum
    canvas.create_text(app.width/2, 25,
                       text=f'Poeng: {app.points}',
                       font='Arial 20 bold')

    # tegn alle rutene
    for row in range(app.rows):
        for col in range(app.cols):
            (x0, y0, x1, y1) = get_cell_bounds(app, row, col)

            is_mole = (row, col) == app.mole_position
            is_mouse = (row, col) == app.mouse_is_over
            fill = get_color(is_mole, is_mouse)
```

```
        canvas.create_rectangle(x0, y0, x1, y1, fill=fill)

def get_color(is_mole, is_mouse):
    if is_mole and is_mouse: return 'light yellow'
    elif is_mole: return 'yellow'
    elif is_mouse: return 'gray60'
    else: return 'gray50'

run_app(width=400, height=300)
```

[Kopier](#)

## Oppgave 15: CSV-håndtering av fylker

- 10 poeng
  - 2 poeng: les inn filen og behandle den som en 2d-liste
  - 1 poeng: deler opp data om fylker og kommuner
  - 1 poeng: en metode for å finne hvilke kommuner som tilhører et gitt fylke
  - 0.5 poeng: finner største kommune i fylket
  - 0.5 poeng: finner minste kommune i fylket
  - 0.5 poeng: finner nordligste kommune i fylket
  - 0.5 poeng: finner sydligste kommune i fylket
  - 2 poeng: pen utskrift (0.5 poeng for hver av linjene med informasjon)
  - 2 poeng: skjønnsmessig vurdering.

```
import csv

def read_csv_file(path, encoding="utf-8", **kwargs):
    r''' Reads a csv file from the provided path, and returns its
    content as a 2D list. The default encoding is utf-8, the default
    column delimitier is comma and the default quote character is the
    double quote character ("), though this can be overridden with
    named parameters "delimiter" and "quotechar".'''
    with open(path, "rt", encoding=encoding, newline='') as f:
        return list(csv.reader(f, **kwargs))

adm_table = read_csv_file("NO_ADM12.csv", delimiter=";")

# Alternativ uten å bruke csv-modulen:
# with open("NO_ADM12.csv", "rt", encoding='utf-8') as f:
#     adm_table = [line.strip().split(";") for line in f.readlines()]

# Liste med oppslagsverk, ett oppslagsverk per kommune
municipalities = []
# Oppslagsverk, nøkkelen er fylkeskode, verdiene er navn på fylket
counties = {}
```

```
for row in adm_table[1:]:
    name = row[1]
    lat = float(row[3])

    lon = float(row[4])
    county_code = row[7]
    population = int(row[9])

    if row[5] == "ADM1":
        # Fylke
        counties[county_code] = name
    elif row[5] == "ADM2":
        # Kommune
        municipalities.append({
            "name": name,
            "county_code": county_code,
            "lat": lat,
            "lon": lon,
            "population": population,
        })

def get_municipalities_in(county_name, municipalities, counties):
    result = []
    for mun in municipalities:
        this_county_code = mun["county_code"]
        this_county_name = counties[this_county_code]

        if county_name == this_county_name:
            # Alternativ if-setning som ikke krever at man staver nøyaktig:
            # if this_county_name.lower().startswith(county_name.lower()):
            result.append(mun)
    return result

def get_largest(munlist):
    largest_size = -1
    largest = {}
    for mun in munlist:
        if mun["population"] > largest_size:
            largest_size = mun["population"]
            largest = mun
    return largest

def get_smallest(munlist):
    smallest_size = float("inf")
    smallest = {}
    for mun in munlist:
        if mun["population"] < smallest_size:
            smallest_size = mun["population"]
            smallest = mun
    return smallest
```

```
def get_northernmost(munlist):
    largest_lat = -float("inf")


    largest = {}
    for mun in munlist:
        if mun["lat"] > largest_lat:
            largest_lat = mun["lat"]
            largest = mun
    return largest

def get_southernmost(munlist):
    smallest_lat = float("inf")
    smallest = {}
    for mun in munlist:
        if mun["lat"] < smallest_lat:
            smallest_lat = mun["lat"]
            smallest = mun
    return smallest

def print_county(county_name, municipalities, counties):
    munlist = get_municipalities_in(county_name, municipalities, counties)
    if len(munlist) == 0:
        return False

    print("="*38)
    print(county_name)
    print("="*38)
    bigcity = get_largest(munlist)
    print(f'{bigcity["name"]:25} {bigcity["population"]:12}')
    smallcity = get_smallest(munlist)
    print(f'{smallcity["name"]:25} {smallcity["population"]:12}')
    northern = get_northernmost(munlist)
    print(f'{northern["name"]:20} {round(northern["lat"], 1):6}°N '
          + f'{round(northern["lon"], 1):6}°Ø')
    southern = get_southernmost(munlist)
    print(f'{southern["name"]:20} {round(southern["lat"], 1):6}°N '
          + f'{round(southern["lon"], 1):6}°Ø')
    print("="*38)
    return True

while True:
    print("Which county (q to quit)?", end=" ")
    county_name = input()
    if county_name == "q":
        break
    if not print_county(county_name, municipalities, counties):
        print("No matching county found. Try again.")
```

Universitetet i Bergen  Om siden.