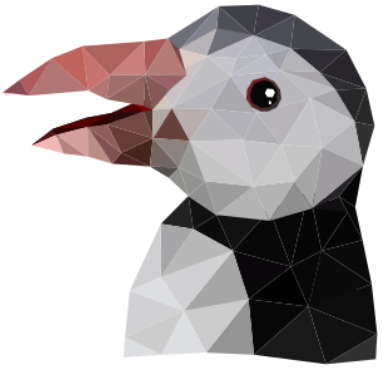
Eksamen INF100 Høsten 2018



10. desember 2018, 0900–1400

- Eksamen er på 5 timer
- Alle skrevne og trykte hjelpemidler er tillatt
- Prosentsatsene angir omtrentlig vekt ved sensur
- Les gjennom hele oppgavesettet før du begynner – se om noe er uklart, og noter deg evt. spørsmål du har til foreleser
- Du kan bruke resultater fra andre oppgaver selv om du ikke har løst dem
- Husk å disponere tiden godt – særlig den siste oppgaven vil gjerne ta mye tid

Lykke til!

– Anya Bagge

Del A – Teori og kodeforståelse (20%)

Tips:

- Hvis spørsmålet er komplisert er det ofte lurt å tegne opp hva som skjer
- På oppgavene med lister og referanser er det særlig viktig å tegne opp variablene og objektene, ellers går det fort surr i hva som skjer.

1) Navn og verdier

1a)

Hva vil denne koden skrive ut?

```
x = 2
x += 2
print(x)
```

1b)

Hva vil denne koden skrive ut?

```
x = 15
def f(x):
    print(x)
f(37)
```

1c)

Hva vil denne koden skrive ut?

```
x = 15
def f(x):
    x = x * 2
f(x)
print(x)
```

1d)

Hva vil denne koden skrive ut?

```
def g(b):
    return b + 1

def f(a):
    b = 10
    return g(a) + 1

print(f(1))
```

1e)

Hva vil denne koden skrive ut?

```
def f(a = 1, b = 2):
    return a * b

print(f(a=3))
```

2) If og løkker 1

2a)

Hva vil denne koden skrive ut?

```
x = 10
if x <= 0:
    svar = "ingen"
if x <= 5:
    svar = "noen"
if x <= 10:
    svar = "fler"
if x <= 100:
    svar = "mange"
print(svar)
```

2b)

Hva vil denne koden skrive ut?

```
l = [x*2 for x in [1, 2, 3, 4]]
print(l[1])
```

2c)

Hva vil denne koden skrive ut?

```
l = [x*2 for x in ['a', 'b', 'c', 'd']]
print(l[1])
```

3) If og løkker 2

3a)

Hva vil denne koden skrive ut?

```
s = '*'
for i in range(5):
    t = '-' * (5 - i)
    print(t + s*i + t)
```

4) Løkker og Referanser

4a)

Hva vil denne koden skrive ut?

```
x = [1,2]
def double(l):
    return [a*2 for a in l]
double(x)
print(x)
```

4b)

Hva vil denne koden skrive ut?

```
def f(x):
    x = x + 1
    return x
x = 3
f(x)
print(x)
```

4c)

Hva vil denne koden skrive ut?

```
def f(x):
    x[0] = x[0] + 1
    return x
x = [3]
f(x)
print(x)
```

5) Referanser

5a)

Hva vil denne koden skrive ut?

```
x = [1,2]
y = [x, x, x]
y[0] = [3,3]
x[0] = 3
print(y)
```

5b)

Forklar (evt. med ein illustrasjon).

Del B – Små programmeringsoppgaver (30%)

6) Tall (10%)

- For alle disse oppgavene er det bra om du gir en feilmelding (med `assert` eller `raise`) for ugyldige parametre.
- Hvis noe er uklart, skriv dokumentasjon eller kommentar som forklarer hva du har gjort.
- Om en oppgavene ber deg implementere funksjoner som allerede finnes i Python – da er det meningen du skal skrive ting på egenhånd, ikke bruke de innebygde.

a) Deling (2%)

Definér en funksjon `is_divisible(a, b)` som returnerer True hvis heltallet `a` er delbart på heltallet `b`.

For eksempel:

```
>>> is_divisible(-3, 2)
False

>>> is_divisible(4, 3)
False
```

b) Primtall (2%)

Definér en funksjon `is_prime(n)` som tar et heltall `n >= 2` og sjekker om `n` er et primtall. Et primtall er delelig bare på seg selv og 1.

Det finnes veldig smarte måter å gjøre dette på, men det er lettest å bare sjekke `n` mot alle tall fra 2 og opp til `n` (eller, enda bedre, opp til `int(math.sqrt(n)+1)`).

For eksempel:

```
>>> is_prime(7)
True

>>> is_prime(9)
False
```

c) Likebehandling (3%)

Definér en funksjon `evenize(l)` som tar en liste av heltall, og returnerer en ny liste der alle oddetall er økt med 1.

For eksempel:

```
>>> evenize([-3, 4, 5, 9, 0])
[-2, 4, 6, 10, 0]
```

d) Positiv sum (3%)

Definér en funksjon `pos_sum(l)` som tar en liste av tall, og returnerer summen av alle de *positive* tallene (alle `l[i] > 0`).

For eksempel:

```
>>> pos_sum([-4, 2, 0, -5, 9])
11
```

Matte-repetisjon:

- I Python: `a` er *delelig* på `b` hvis `a % b == 0`.
- Full forklaring: Et heltall `a` er delelig på et annet heltall `b` hvis *resten* som er igjen etter deling er 0; dvs. at `(a // b) * b == a`. F.eks. `(10 // 5) == 2` og `2 * 5 == 10`, så 10 er delbart på 2, mens `(10 // 3) == 3` og `3 * 3 != 10`, så 10 er ikke delbart på 3. (Operatoren `//` er heltallsdivisjon, mens `/` gir et flyttall.)
- `a % b` gir deg resten fra å dele `a` på `b`. Så, `(a // b) * b + (a % b) == a`. I eksempelet på forrige linje vil `10 % 5 == 0` og `2 * 5 + 0 == 10`, mens `10 % 3 == 1` som passer med at `3 * 3 + 1 == 10`.
- *(`a % b` gir deg forøvrig mellom 0 og `b`, som ofte kan være ofte nyttig – når du ikke har eksamen)*

---

7) Ord (10%)

- For alle disse oppgavene er det bra om du gir en feilmelding (med `assert` eller `raise`) for ugyldige parametre.
- Hvis noe er uklart, skriv dokumentasjon eller kommentar som forklarer hva du har gjort.
- Om en oppgavene ber deg implementere funksjoner som allerede finnes i Python – da er det meningen du skal skrive ting på egenhånd, ikke bruke de innebygde.

a) Entall og flertall (3%)

Lag en funksjon `plural(word)` som returnerer ordet `word` (en streng) i flertall. Vi antar at `word` er et passende entallsord, og at vi kan gjøre det om til flertall ved å legge til en endelse / suffiks. Vi bruker en *veldig* forenklet utgave av engelsk grammatikk, med følgende regler:

- Ord som `'s'`, `'sh'` eller `'ch'` får endelsen *-es* i flertall: *witch* → *witches*
- Ordet som ender på *-y* mister *y*-endelsen og får *-ies* i stedet: *cherry* → *cherries*
- Alle andre ord får *-s* endelse i flertall: *banana* → *bananas*

For eksempel:

```
>>> plural('daisy')
'daisies'

>>> plural('present')
'presents'

>>> plural('dish')
'dishes'

>>> plural('day')
'daies'

>>> plural('foot')
'foots'
```

(Siden vi bruker forenklede regler, blir det feil på de to siste.)

b) Telling (4%)

Vi har en kurv full av frukt, og har lyst til å vite hvor mange vi har av hver frukt. La oss anta at vi representerer fruktene som strenger i en liste, f.eks. `['apple', 'apple', 'grape', 'banana', 'pear', 'kiwi', 'kiwi', 'kiwi']`.

Lag en funksjon `count(l)` som tar en liste `l` og teller hvor mange ganger hvert element forekommer og returnerer en `dict` med antall forekomster av hvert unike element.

Tips: Start med en tom ordbok/dictionary, `{}`. Når du teller trenger du å slå opp hvor mange ganger du har sett et element (denne frukten) før, og det kan gi trøbbel hvis den ikke finnes i ordboken fra før. Da er det smart å bruke `get`-metoden: `d.get(k, 0)` slår opp nøkkelen `k` i `d` og gir `0` om `k` ikke finnes.

For eksempel:

```
>>> count(['apple', 'apple', 'grape', 'banana', 'pear',
'kiwi', 'kiwi', 'kiwi'])
{'apple': 2, 'grape': 1, 'banana': 1, 'pear': 1, 'kiwi':
 3}
```

c) Fruktkurv-rapport (3%)

Skriv en liten kodesnutt som printer en liten oversikt over innholdet i listen `fruit_basket`, en linje per frukt med antallet og navnet på frukten. Bruk `count` for å telle, og `plural` for å skrive navnet på frukten i entall eller flertall avhengig av om du har en eller flere av den.

For eksempel, med

```
fruit_basket = ['apple', 'apple', 'grape', 'banana', 'pe
ar', 'kiwi', 'kiwi', 'kiwi']
```

skal utskriften bli:

```
2 apples
1 grape
1 banana
1 pear
3 kiwis
```

---

8) Tekst (10%)

a) Sentrale prinsipper (4%)

Lag en funksjon `center(s,width)` som sentrerer strengen `s` på en linje som er `width` tegn lang. Dvs. fyller med mellomrom på hver side inntil lengden av `s` er større enn eller lik `width`.

Fjern eventuelle mellomrom på begynnelsen og slutten av `s` først.

For eksempel (med mellomrom vist som «␣»):

```
>>> center('***', 30)
'␣␣␣␣␣␣␣␣␣␣␣␣␣***␣␣␣␣␣␣␣␣␣␣␣'

>>> center('␣␣␣␣A Christmas Carol', 30)
'␣␣␣␣␣␣␣A Christmas Carol␣␣␣␣␣␣␣'
```

b) Juletre (6%)

Lag en funksjon `print_tree(width, height)` som printer et juletre. Du kan «tegne» et tre med vanlige symboler/tegn, f.eks. `*` på den første linjen, `***` på neste, `*****` på neste osv.; avslutt med én på siste linje for å lage en «fot»:

```
   *
  ***
 *****
*******
   *
```

Sentrer linjene med `center` før du printer dem, så får du et tre som vokser fint utover.

- For ekstra fin grantre-effekt, reduser antall tegn med fire for hver fjerde linje.
- Du kan justere på tegnene du bruker underveis hvis du vil «pynte» treet. (F.eks. 🎁, 🌟, ❄, 🕯)

Definer funksjonen, og **legg ved** et eksempel på hvordan du tenker utskriften for *din* funksjon blir for `print_tree(10, 7)`.

For eksempel:

```
>>> print_tree(10, 7)
    ☆
   ###
  #####
   ###
  #####
 #######
    #
```

Del C – Data Claus 🎅 (50%)

Bakgrunn 🎁

«Santa Claus is comin' to town»

Julenissen driver en av verdens største dataoperasjoner – større enn datainnsamlingen som amerikanske NSA og CIA driver med. Etter et skandale-år hvor hundretusenvis av uskyldige barn fikk kull under treet i stedet for presanger, har nissen bestemt seg for å investere tungt i Big Data og moderne IT-systemer. Som ung alv med tung IT-ekspertise fra INF100 har du fått i oppdrag å implementere et moderne system som skal gi rettferdig fordeling av julepresanger.

Systemet skal:

- Registrere observasjoner av potensielle presang-mottakere
- Kalkulere «naughty» eller «nice» status for hver mottaker
- Fordele gaver på alle mottakerne
- Generere en rapport med navn, adresse og gave-id som julenissen kan ha med seg i sleden
- Dobbeltsjekke rapporten (i tråd med legenden)

Du skal løse dette i deloppgavene under.

a) Registrere observasjoner (10%)

«He's making a list...»

Julenissen registrerer en «observasjon» hver gang noen gjør noe snilt eller slemt (en såkalt «hendelse», på fagspråket). Hver hendelse fører til at status justeres opp eller ned med en viss poengsum. For å være «nice» og få fine presanger, må man ha positiv status.

Vi trenger å lagre følgende informasjon:

- Nåværende status for hver enkelt person. Vi antar at personer er identifisert ved navn (`str`), og status er et tall («nice-poeng», `int` eller `float`), der negative tall betyr at personen har vært (jevnt over) slem i løpet av året, og positive tall betyr at personen har vært snill.
- En logg av alle observerte hendelser, slik at det går an å kontrollere i ettertid at man har regnet riktig. Observasjonslisten skal lagre navn (f.eks. `'Anya'`), en beskrivelse (f.eks. `'Rappet pepperkake fra student på eksamen'`), antall positive eller negative poeng denne hendelsen gir (f.eks. `-0.5`), og nåværende poengsum etter at hendelsen er registrert (dvs. akkumulert poeng inkludert denne hendelsen – f.eks. `-200.37`).

Gjør følgende:

- *(3%)* Finn ut hvordan du vil lagre informasjonen. Det passer antakelig greit med en `dict` for poeng-status, og en liste av lister eller liste av `dict` for å lagre hendelsesloggen. (Du kan pakke alt i en klasse om du vil.)
- *(2%)* Skriv ned et lite eksempel på de to datastrukturene
- *(5%)* Implementer en funksjon eller metode `reg_incident` tar imot navn, beskrivelse og poeng på en observert hendelse, oppdater status for personen, og legger til hendelsen i loggen.

Tips:

- Oppgaveteksen antar at du lagrer tilstanden til systemet i globale variabler, men du står helt fritt til å gjøre dette slik du selv synes er best, inkludert å bruke klasser og objekter.
- Forklar evt. slike designvalg du tar.
- Skriv dokumentasjon (docstring) til koden din.

---

b) (5%)

«Gonna find out who's naughty and nice»

- *(2%)* Lag en funksjon `is_nice` som tar navnet på en person og returnerer True om personen har positiv antall poeng (har vært mer snill enn slem i løpet av året). For eksempel:

  ```
  >>> is_nice('Anya')
  False
  ```

- *(3%)* Lag en funksjon `status_all` som returner en `dict` der hver person mappes til True eller False avhengig av om personen har vært snill eller slem. For eksempel:

  ```
  >>> status_all()
  {'Albin':True, 'Anya':False, 'Colin':False, ...
  ```

---

c) (15%)

«So be good for goodness sake!»

Gavefabrikken har sitt eget datasystem, som produserer en liste over tilgjengelige julepresanger på denne formen:

Gåvefabrikken har sitt eige datasystem, som produserar ei liste over tilgjengelege julepresangar på denne formen:

```
[(gave_id, beskrivelse, vekt, hylleplass), ...]
```

For eksempel:

```
presents = [(23791, 'Dinosaurfigur', 0.3, 'AD-339'), (23
792, 'Stripete slips', 0.2, 'ZR-201'),
 (23793, 'Lekekjøkken', 5.3, 'UB-799'), (23794, 'PuffinB
oard v0.1', 0.01, 'ZZ-237')]
```

Lag en funksjon `assign_presents` som, gitt en slike gaveliste, fordeler presanger på personer:

- Julenissens sekretær vil kjøre denne funksjonen etter at alle hendelser er registrert, så du kan bruke `status_all` eller `is_nice` til å finne ut hvem som fortjener å presang og ikke
- *(10%)* Presangfordeling:
  - De som er «snille» skal bli tildelt en presang – det er det samme hvilken presang du plukker fra listen, men du må passe på å bare bruke hver presang én gang, og at hver person bare får én.
  - Du kan anta at det alltid er flere presanger i listen, uansett hvor mange du plukker.
- *(5%)* I henhold til tradisjonen skal de som har vært «slemme» få kull i stedet. Grunnet nye forskrifter om bruk av fossilt brensel har julenissen en begrenset mengde kull tilgjengelig – antall kilo kull blir oppgitt i et parameter `available_coal`. Du kan håndtere dette på en av disse måtene:
  - *(Enklest, 0/5)* se bort fra begrensningen, og håp at Julenissen ordner ting likevel
  - *(Enkel, 2/5)* del ut presanger i stedet når det er tomt for kull (denne gir deg sikkert «snill»-poeng neste år...)
  - *(Mer avansert, 5/5)* gå gjennom alle og sjekk hvor mange som skal ha kull, og fordel kullet likt

Gavefordelingen returners som en `dict` der navnet på personen er nøkkelen og gaveinformasjonen er verdien: `{'Martha' : (4387, 'Zelda brettspill', 2.0, 'NT-279'), ...}`

Kull registreres som en «gave» med `gave_id == 0`: `(0, 'Kull', 1.0, '')`.

For eksempel:

```
>>> assign_presents(presents, available_coal=1.0)
{'Albin' : (23791, 'Dinosaurfigur', 0.3, 'AD-339'),
 'Anya' : (0, 'Kull', 1.0, ''),
 'Colin' : (23792, 'Stripete slips', 0.2, 'ZR-201'), ...
}
```

Tips

- Du kan gjøre fordelingen ved å gå gjennom alle personer i en løkke og plukke presanger til dem, eller omvendt, ved å gå gjennom listen av presanger og plukke personen som skal få den.
- For å plukke et element fra en liste kan du bruke metoden `list.pop()` – den fjerner et element og returnerer det

---

d) (10%)

«He sees you when you're sleeping; He knows when you're awake; He knows if you've been bad or good»

Ettersom Julenissen også opererer i EU/EØS-området er han nødt til å tilfredsstille de nye GDPR reglene, som blant annet gir «kundene» kontroll over sine egne persondata. Med mengden sensitive personopplysninger som er involvert, vil dette antakeligvis lage arbeid for Julenissens advokater i årevis fremover. På datasiden skal vi nøye oss med å implementere to små tiltak:

- Vi må gjøre det mulig å samle sammen all informasjon om én enkelt person, i tilfelle personen krever innsyn i sine data
- Det må være mulig å slette alle data tilhørende en person.

Det er lett å finne nåværende status for en person, så for å håndtere innsyn trenger vi bare en funksjon `person_log` som returnerer alle elementene i hendelsesloggen som omhandler en person. F.eks.:

```
>>> person_log('Anya')   # evt. person_log('Anya', incid
ents) eller db.person_log('Anya') e.l.
[('Anya', 'Rappet pepperkake fra student på eksamen', -
0.5, -0.5), ('Anya', 'Rappet nøtt fra student på eksame
n', -0.5, -1.0), ...]
```

(Selve gavelisten er unntatt innsyn, ihht. GDPR Artikkel 15(1)(x).)

For å slette en person skal vi ha en funksjon `delete_person`, som fjerner personen fra både hendelsesloggen og statusoversikten.

- Implementer `person_log` og `delete_person`. *(5% hver)*

Tips

- For å slette fra liste eller dictionary, kan du bruke `del`: f.eks. `del status_dict[name]`.
- For å slette fra loggen er det kanskje best å sette elementene du skal fjerne til `None`, for det kan fort skje rare ting hvis du fjerner elementer fra en liste mens du holder på å gå gjennom den. Evt. kan du bygge en ny liste med `[e for e in lst if ...]`

e) (10%)

«...And checking it twice»

I henhold til tradisjonen skal listen dobbeltsjekkes. Lag en funksjon `check_assignment` som tar imot gavelisten og dobbeltsjekker at alle har fått det de fortjener.

Det som skal sjekkes er *(2% for hver + 2% om du har gjort alle de tre første)*:

1. *(minimum)* At antall personer på presanglisten er lik antall personer registrert.
2. At alle registrerte personer er på presanglisten.
3. At «gaven» (presang eller kull) stemmer overens med status
4. *(ekstra)* At «gaven» (presang eller kull) og status stemmer overens med hendelsesloggen

Du kan f.eks. bruke denne funksjonen til å sjekke om en gave er en ordentlig gave eller kull:

```
def is_present(gift):
    """True if the gift is a real present, False if it's
 coal"""
    return gift[0] != 0
```

Hvis du finner feil, skal du rapportere det slik du synes er mest praktisk. F.eks. at du sjekker ting med `assert`, melder fra om feil med `raise ValueError('...')`, eller evt. bruker `print` eller returnerer en liste av feilmeldinger.

God jul og godt nyttår! 🎄 ⭐ ❄ 🎁 🎅 🦌

(og lykke til med evt. gjenværende eksamener!)

Python Referanse

Innebygde funksjoner

`a = len("foo")` – *Finn lengden til en streng, liste eller sekvens*

`len("foo")` → `3`

`type("5")` – *Finn typen til en verdi*

`type("5")` → `'str'`

`print(a, b, c)` – *Skriv ut til skjermen: `1 0 [1, 2, 3]`*

`print(a, b, c)` → `None`

`s = input()` – *Les inn ein streng ifrå tastaturet*

`input()` → `'Hei!'`

`a = ord("0")` – *Finn teiknkoden (code point) til eit teikn*

`ord("0")` → `128039`

`s = chr(128039)` – *Oversett teiknkode til streng*

`chr(128039)` → `'0'`

`d = locals()` – *Ordbok med lokale variabler*

`locals()` → `{'a': 42, 'x': 4.2, 's': '  pUffinS aRe c00l'}`

`d = globals()` – *Ordbok med globale variabler*

`globals()` →

```
{'f': <funcion f>, '__name__': '__main__', '__builtins__': {'len':
                                                          ...}}
```

`l = dir("5")` – *Se hvilke navn/metoder/felter som finnes i et objekt/scope*

`dir("5")` → `['__add__',...,'capitalize', 'casefold',...]`

`s = help(str.split)` – *Se på hjelpeteksten til et objekt (funksjon/type vanligvis)*

`help(str.split)` → `'...Returns a list of...'`

`x = abs(-2.4)` – *Finn absoluttverdien til et tall*

`abs(-2.4)` → `2.4`

`x = min(-2.4,5)` – *Finn den minste av en mengde verdier*

`min(-2.4,5)` → `-2.4`

`a = max(-2.4,5)` – *Finn den største av en mengde verdier*

`max(-2.4,5)` → `5`

```
r = range(10)  – Lag et intervall [0, 1, ..., 9]
```
range(10) → range(0, 10)

```
r = range(2, 10)  – Lag et intervall [2, 3, ... 9]
```
range(2, 10) → range(2, 10)

```
r = range(2, 10, 3)  – Lag et intervall [2, 5, 8]
```
range(2, 10, 3) → range(2, 10, 3)

`int` – Heltall

```
a = 42  – Lag heltall ved å skrive vanlige tall
```
a → 42 : int

```
a = 0x2a  – Du kan også bruke 16-tallssystemet (heksadesimalt)...
```
a → 42 : int

```
a = 0b101010  – ...eller 2-tallssystemet (binært)
```
a → 42 : int

```
a = a + 3  – Bruk +, -, *
```
a + 3 → 45

```
x = a / 3  – Deling gir flyttall (med desimaler)
```
a / 3 → 14.0 : float

```
a = a // 3  – Heltallsdeling gir heltall (uten desimaler)
```
a // 3, (a+1) // 3 → (4, 5)

```
a = a % 3  – Modulus / divisions-rest
```
a % 3, (a+1) % 3 → (0, 1)

```
a // 0  – Deling på null gir feil
```
ZeroDivisionError: integer division or modulo by zero

`float` – Flyttall

```
x = 4.2  – Lag flyttal ved å skrive desimaltall
```
x → 4.2 : float

```
x = 42e-1  – Du kan også bruke vitenskaplig notasjon
```
x → 4.2 : float

```
x = x + 3.0  – Bruk +, -, *
```
x + 3.0 → 7.2

```
x = 4.2 / 3.0  – Deling – merk at flyttall kan være litt upresise
```
4.2 / 3.0 → 1.4000000000000001

`x = x // 3` – *Heltallsdeling kutter desimalene fra resultatet*

`6.0 // 3.0, 7.0 // 3.0 → (2.0, 2.0)`

`x = x % 3.0` – *Modulus / divisions-rest fra heltallsdivisjon*

`6.0 % 3.0, 7.0 % 3.0 → (0.0, 1.0)`

~~`x / 0`~~ – *Deling på null gir feil*

`ZeroDivisionError: float division by zero`

`str` – Strenger og tekst

`s = "  pUffinS aRe c00l"` – *Lag strenger med '...' eller "..."*

`s → '  pUffinS aRe c00l' : str`

`s = f"{a} + 2 = {a + 2}"` – *Streng-formattering med {Python-uttrykk} inni strengen*

`f"{a} + 2 = {a + 2}" → '3 + 2 = 5'`

`s = "a" + "b"` – *Lim sammen (konkatenering) med streng + streng*

`"a" + "b" → 'ab'`

`s = "a" * 3` – *Multiplisering med streng * int*

`"a" * 3 → 'aaa'`

`s = s[3]` – *Indeksering, finn teikn på posisjon i*

`s[3] → 'U'`

`s = s[-1]` – *Indeksering, finn teikn på posisjon len(s)-i*

`s[-1] → 'l'`

`s = s[1:3]` – *slicing – «skjer av» ein substreng*

`s[1:3] → ' p'`

`if s.startswith("p"): ...` – *Se om strenger starter på gitt prefix (denne starter på " ")*

`s.startswith("p") → False`

`if s.endswith("00l"): ...` – *Se om strenger ender på gitt suffix*

`s.endswith("00l") → True`

`s = s.strip()` – *Fjerner mellomrom før og etter*

`s.strip() → 'pUffinS aRe c00l'`

`s = s.lower()` – *Gjør alle bokstaver små*

`s.lower() → '  puffins are c00l'`

`s = s.upper()` – *Gjør alle bokstaver store*

```
s.upper() → '  PUFFINS ARE C00L'
```

`s = s.capitalize()` – *Gjør første bokstav stor, resten små*

```
s.capitalize() → '  puffins are c00l'
```

`s = s.title()` – *Gjør første bokstav stor i hvert ord stor, resten små*

```
s.title() → '  Puffins Are C00L'
```

`s = s.replace("00", "oo")` – *Søk og erstatt*

```
s.replace("00", "oo") → '  pUffinS aRe cool'
```

`s = " og ".join(["a","b","c"])` – *Slå sammen liste av strenger*

```
" og ".join(["a","b","c"]) → 'a og b og c'
```

`l = s.split()` – *Splitt til liste av strenger (ved mellomrom eller argument)*

```
s.split() → ['pUffinS', 'aRe', 'c00l']
```

`s.upper()` – *Strengen blir ikke endret; metodene returnerer ny verdi*

```
s → '  pUffinS aRe c00l'
```

~~`s[1] = "X"`~~ – *Du kan ikkje endre/oppdatera teikna i ein streng*

```
TypeError: 'str' object does not support item assignment
```

`list` – Lister

`l = [4, 2, 4, 9, ["a","b"]]` – *Lag lister med `[e1, e2, ...]`*

```
l → [4, 2, 4, 9, ['a', 'b']] : list
```

`a = l[0]` – *Hent element*

```
l[0] → 4
```

`l = l[-1]` – *Hent siste element*

```
l[-1] → ['a', 'b']
```

`del l[0]` – *Slett element*

```
l → [2, 4, 9, ['a', 'b']]
```

`l[1] = 3` – *Oppdater element på ein gitt indeks*

```
l → [2, 3, 9, ['a', 'b']]
```

`l[3][0] = "c"` – *Oppdater element i ei nøsta liste*

```
l → [2, 3, 9, ['c', 'b']]
```

`l = l[1:3]` – *slicing – «skjer av» ei subliste*

```
l[1:3] → [3, 9]
```

`l[2:] = [0,1,2]` – *slicing – erstatt ei avskoren subliste*

```
l → [2, 3, 0, 1, 2]
```

`if 4 in l: ...` – *Sjå om elementet finnast*

```
4 in l → False
```

`if 4 not in l: ...` – *Sjå om elementet ikkje finnast*

```
4 not in l → True
```

`l = [f(e) for e in seq]` – *Liste [f(e1), f(e2), ...], for e1, e2, ... i en sekvens*

```
[str(e*2) for e in l] → ['4', '6', '0', '2', '4']
```

`l = [f(e) for e in seq if p(e)]` – *List comprehension med filtrering*

```
[e for e in range(10) if e%2 == 0] → [0, 2, 4, 6, 8]
```

`l.append(5)` – *Legg til element på slutten*

```
l → [2, 3, 0, 1, 2, 5]
```

`l.insert(3,7)` – *Legg til element foran element i*

```
l → [2, 3, 0, 7, 1, 2, 5]
```

`t = l.pop(), l` – *Fjern og returner siste element*

```
l.pop(), l → (5, [2, 3, 0, 7, 1, 2])
```

`l = l.copy()` – *Lag kopi, `l.copy() == l`, `not l.copy() is l`*

```
l.copy() → [2, 3, 0, 7, 1, 2]
```

`l = l + [9]` – *Slå sammen (konkatenér) to lister til ei ny liste*

```
l + [9] → [2, 3, 0, 7, 1, 2, 9]
```

~~`l + 9`~~ – *+ verkar berre med lister, ikkje elementer*

```
TypeError: can only concatenate list (not "int") to list
```

`tuple` – Tupler

`t = (2, 4, 9)` – *Lag tupler med `(e1, e2, ...)`*

```
t → (2, 4, 9) : tuple
```

`t = 1, 2, 3` – *Vi kan utelate parentesar når det ikkje kan mistolkes*

```
t → (1, 2, 3) : tuple
```

`t = (2,)` – *Lag 1-tupler med `(e1,)`*

```
t → (2,) : tuple
```

```
a = t[1]
```
– *Bruk tupler som lister*

```
t[1] → 4
```

```
t[1] = 3
```
– *Du kan ikkje endre/oppdatere elementa i tupler*

```
TypeError: 'tuple' object does not support item assignment
```

---

dict – Ordbøker/dictionaries

```
d = {"mango":"grønnsak","gulrot":"grønnsak"}
```
– *Lag ordbok med k1→v1, k2→v2*

```
d → {'mango': 'grønnsak', 'gulrot': 'grønnsak'} : dict
```

```
s = d["mango"]
```
– *Slå opp nøkkel*

```
d["mango"] → 'grønnsak'
```

```
del d["mango"]
```
– *Slett oppføring*

```
d → {'gulrot': 'grønnsak'}
```

```
d["mango"] = "frukt"
```
– *Oppdater / legg til oppføring*

```
d → {'gulrot': 'grønnsak', 'mango': 'frukt'}
```

```
s = d.get("banan", "ukjent")
```
– *Slå opp nøkkel, med default verdi*

```
d.get("banan", "ukjent") → 'ukjent'
```

```
d.keys()
```
– *Liste av nøklar*

```
d.keys() → dict_keys(['gulrot', 'mango'])
```

```
l = [k for k in d]
```
– *« for » itererer over / henter nøklane*

```
[k for k in d] → ['gulrot', 'mango']
```

```
d = {k:d[k].upper() for k in d}
```
– *Comprehension: Bygg dict basert på sekvens*

```
{k:d[k].upper() for k in d} →
{'gulrot': 'GRØNNSAK', 'mango': 'FRUKT'}
```

```
if "mango" in d: ...
```
– *Sjekk om nøkkel finnes*

```
"mango" in d → True
```

```
if "mango" not in d: ...
```
– *Sjekk om nøkkel ikke finnes*

```
"mango" not in d → False
```

```
d.update({"banan":"frukt"})
```
– *Legg til alle nøkkel/verdi-par frå ein anna dict*

```
d.update({"banan":"frukt"}) → None
```
