

# Praktisk Informasjon

- **Kommunikasjon**

- Jeg ønsker en dialog. Still spørsmål når som helst.
- Tilbakemeldinger er velkomne. Er det noe som er uklart?
- Gi beskjed hvis tempoet er for raskt eller for sakte.
- Informer meg om lydnivået er passende, og om teksten er lesbar.

NB: Nytt med lukket eksamen

- **Kursstruktur**

- Vi skal gjennom mye pensum, men ikke alt. Noe innhold kan være repetitivt, men kanskje lærer du noe nytt?
- Det blir en 10-minutters pause hver 50 min, f.eks., fra 15:20 til 15:29:59.

- **Engasjement**

- Bli aktiv i læringen. Noter på den måten som fungerer best for deg.
- Presentasjonen vil bli tilgjengelig etter kurset. (Discord?)
- Målet er å avslutte senest kl. 18:30. Hvis vi er ferdige tidligere, avslutter vi da.
- Om det er et sterkt ønske om å fortsette etter planlagt sluttid, kan vi forlenge.
- Føl deg fri til å gå når du må.

Mål: Gøy og lærerikt



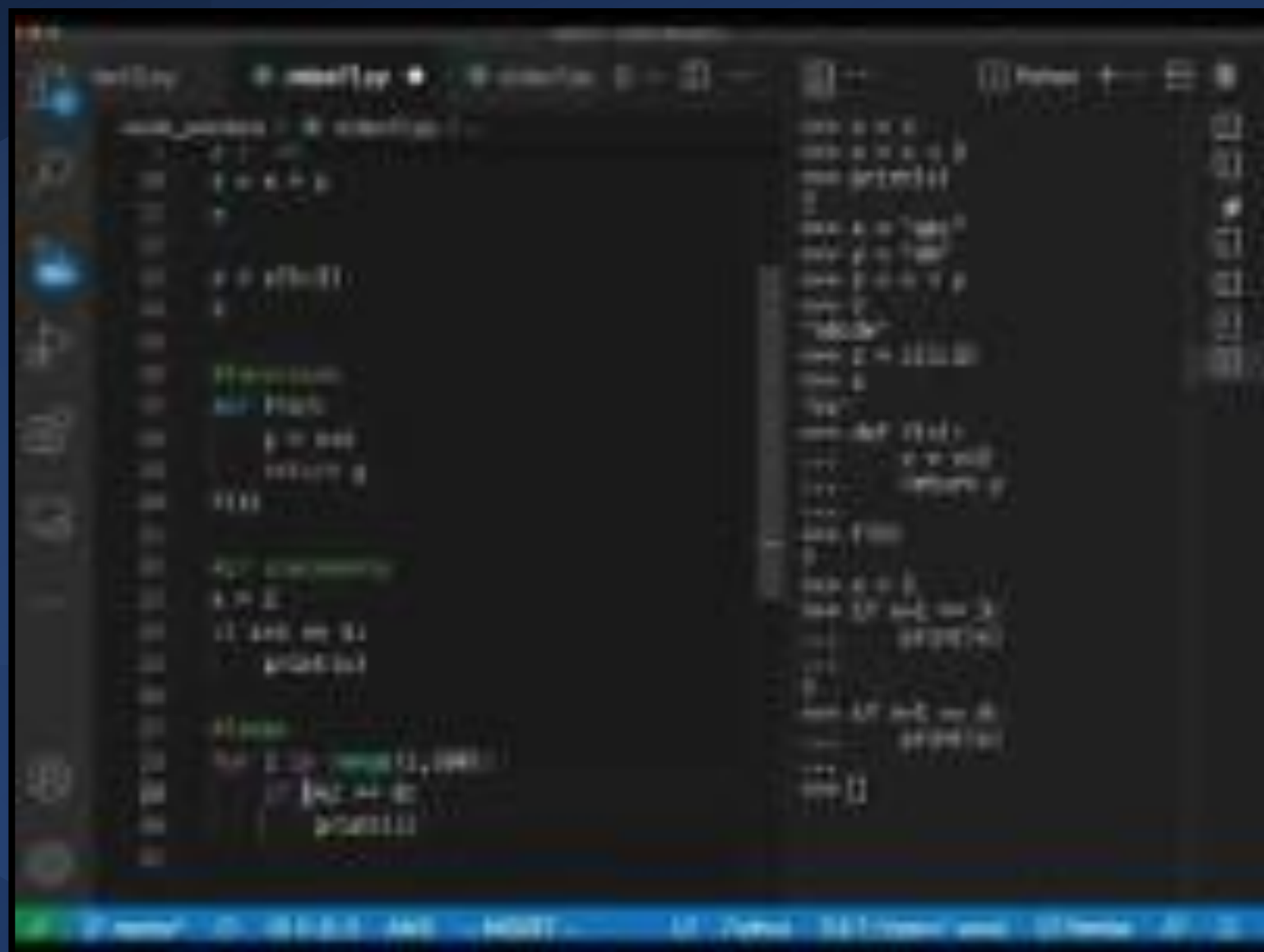
# INF100 KRÆSJKURS H23

m/Gnist

Av Aleksander L. Fedøy

Join at  
**slido.com**  
**#3921 939**





# Plan for kurset

- **Python-pensum:**
  - Typer/datatyper/uttrykk (med print)
  - Betingelser, boolske uttrykk og presedens
  - Lister, oppslagsverk, mengder med løkker
  - Funksjoner
  - Feilhåndtering (try/except)
  - Lese/skrive til fil
- «Oppgaver»/øving:
  - Kodesporing
  - Forklaringer
  - Kodeskriving (uten editor?)
- **Tips og triks frem mot eksamen**



# Datatyper i Python (teori)

- **Grunnleggende datatyper**

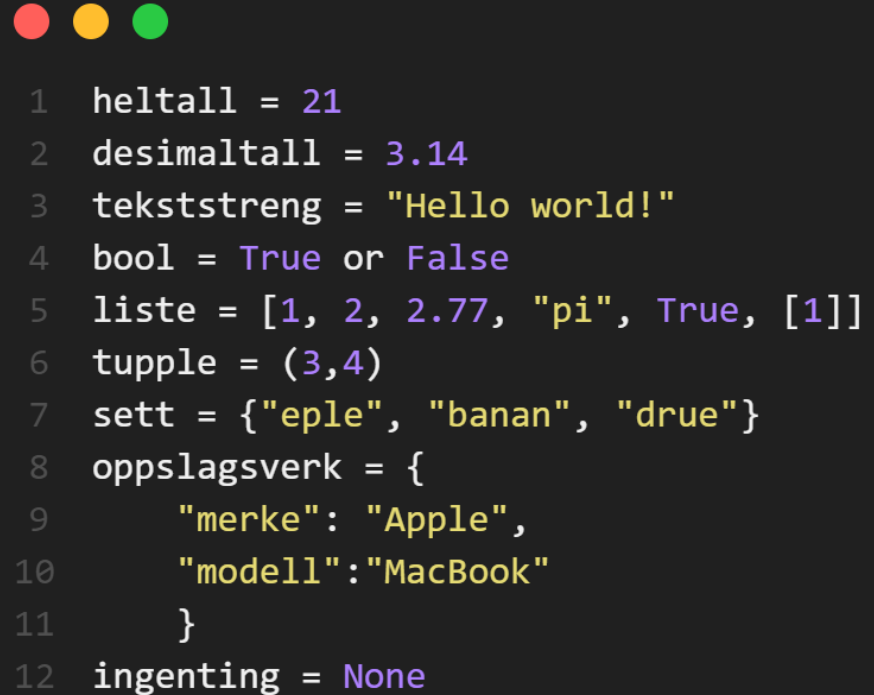
- Heltall (`int`)
- Desimaltall (`float`)
- Tekststrenger (`str`)
- Boolske verdier (`bool`)

- **Samlinger**

- Lister (`list`)
- Tuppler (`tuple`)
- Sett/mengder (`set`)
- Oppslagsverk/ordbok (`dict`)

- **Spesialtyper**

- "Ingenting"/"ingen verdi" (`None`)



```
1 heltall = 21
2 desimaltall = 3.14
3 tekststreng = "Hello world!"
4 bool = True or False
5 liste = [1, 2, 2.77, "pi", True, [1]]
6 tuple = (3,4)
7 sett = {"eple", "banan", "drue"}
8 oppslagsverk = {
9     "merke": "Apple",
10    "modell": "MacBook"
11 }
12 ingenting = None
```

# Dat typer i Python (eksempler)



```
1  # Eksempel
2
3  print("Hva er din alder?")
4
5  alder = input()
6
7  ny_alder = alder + 10
8
9  print("Om 10 år er du "+ny_alder+" år gammel!")
```

# Dat typer i Python (eksempler)

```
⊗ ubuntu@Matebook14:~/05_UIB/INF-100-Krasjkurs$ /bin/python /home/ubuntu/05_UIB/INF-100-Krasjkurs/typefeil.py
Hva er din alder?
27
Traceback (most recent call last):
  File "/home/ubuntu/05_UIB/INF-100-Krasjkurs/typefeil.py", line 5, in <module>
    ny_alder = alder + 10
TypeError: can only concatenate str (not "int") to str
```







# Datatyper i Python (fasit)

```
a = [1, 2.2, "3", "a"]
```

```
b = "foo"
```

```
c = 42
```

```
d = { "a": "b", "b": "c" }
```

```
e = True
```

| Uttrykk                 | int | dict | None | (-error-) | list | str | bool | float |
|-------------------------|-----|------|------|-----------|------|-----|------|-------|
| <code>a in a</code>     |     |      |      |           |      |     | X    |       |
| <code>a[0:1]</code>     |     |      |      |           | X    |     |      |       |
| <code>b + "a[0]"</code> |     |      |      |           |      | X   |      |       |

# Skrive ut i Python (teori)

- **Konkatenering**

- ``print("Hei, " + navn + "!")``

- **Elementutlisting**

- ``print("Hei,", navn, "!")``

- **Formatert strenger (f-streng)**

- ``print(f"Hei, {navn}!")``

- Flere finnes, men brukes lite



```
1 pris_per_enhet = 5.98765
2 antall_enheter = 8
3
4 print(f"Totalpris: {pris_per_enhet * antall_enheter:.2f} kr")
5 # Totalpris: 47.90 kr
6
7 # Vanlig feil: Glemmer å avslutte uttrykket før formattering
8 print(f"Totalpris: {pris_per_enhet * antall_enheter.2f} kr")
9 # Dette vil gi en syntax error
```

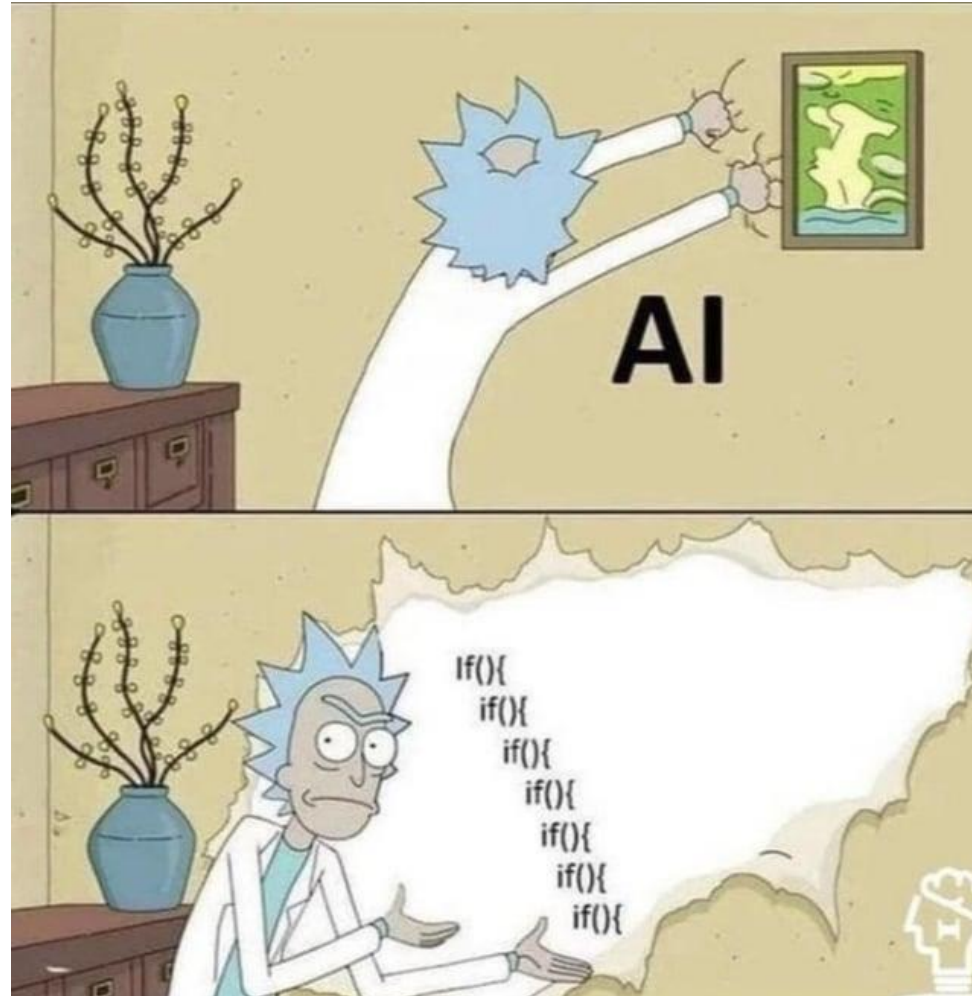
# Skrive ut i Python (oppgaver)

- Oppgave:
  - Skriv ferdig programmet slik at det skriver ut ønsket «output»



```
1  # Avrund pi til to desimaler
2  pi = 3.14159265
3
4  print(...)
5
6  # Ønsket output: "pi = 3.14"
```

# Betingelser, boolske uttrykk og presedens (teori)



# Betingelser, boolske uttrykk og presedens (teori)

- Vi kan bruke ``if``, ``elif`` og ``else`` for å kontrollere programflyt.
- Boolske operatører:
  - ``and``, ``or`` og ``not``.
- Sammenligningsoperatorer:
  - ``==``, ``!=``, ``<``, ``>``, ``<=``, ``>=`` og ``is``.
- Presedens:
  - ``not`` har høyest presedens.
  - ``and`` evalueres før ``or``.
  - Paranteser ``(`` og `)`` brukes for å gruppere uttrykk og overstyre presedens.

```
1  temperatur = 25
2
3  if temperatur > 25:
4      print("Det er varmt")
5  elif temperatur >= 10:
6      print("Husk jakke")
7      # printes
8  else:
9      print("Huff, så kaldt!")
10
11 vær = "regn"
12
13 if temperatur > 25 and vær == "sol":
14     print("Husk solkrem")
15 elif temperatur >= 10 or vær != "regn":
16     print("Norsk sommer :-)")
17     # printes
18 elif temperatur is None or not vær:
19     print("Ingen data")
```


# Lister i Python (teori)

- Indeks:

- ``list[index]``
  - Henter ut verdien på plasseringen

- Listemetoder

- ``append(element)``
  - Legger til et element på slutten av listen
- ``extend(list2)``
  - Utvider listen med flere elementer
- ``insert(index, element)``
  - Setter inn et element på spesifisert indeks
- ``remove(element)``
  - Fjerner første forekomst av et element
- ``pop(index)``
  - Fjerner og returnerer element på indeks
- ``sort()``
  - Sorterer listen på stedet
- ``reverse()``
  - Reverserer listen



```
1  liste = [1, 2]
2  print(liste[1]) # 2
3  liste.append(2.77)
4  print(liste) # [1, 2, 2.77]
5  liste2 = ["pi", True]
6  liste.extend(liste2)
7  print(liste) # [1, 2, 2.77, "pi", True]
8  liste.insert(2, 25)
9  print(liste) # [1, 2, 25, 2.77, 'pi', True]
10 liste.remove("pi")
11 print(liste) # [1, 2, 25, 2.77, True]
12 liste.sort()
13 print(liste) # [1, True, 2, 2.77, 25]
14 liste.reverse()
15 print(liste) # [25, 2.77, 2, True, 1]
```



# Lister i Python (oppgaver)

- Oppgave:
  - Skrives det ut True eller False her?

```
1 frukter = ['eple', 'banan']
2 flere_frukter = ['appelsin', 'druer']
3 alle_frukter = ['eple', 'banan', 'appelsin', 'druer']
4
5 frukter.append(flere_frukter)
6 print(frukter == alle_frukter)
```

# Løkker i Python (teori del 1)

- **Iterative for-løkker:**

- ``for element in liste:``
  - Iterasjon direkte over elementer i en liste.

- **Indekserte for-løkker:**

- ``for i in range(len(liste)):``
  - Bruk indeks for å manipulere elementer.

- **While-løkker:**

- ``i = 0``
- ``while i < len(liste)``
  - Basert på en betingelse, kan bruke indeks.

- **Løkker over oppslagsverk:**

- ``for key, value in dict.items():``
  - Itererer over nøkkel-verdi par.

```
1 frukter = ["eple", "banan", "kiwi"]
2 frukt_farge = {
3     "eple": "rødt",
4     "banan": "gul",
5     "kiwi": "grønn"
6 }
7
8 for frukt in frukter:
9     print(f"Frukt: {frukt}")
10 for i in range(len(frukter)):
11     frukter[i] = frukter[i].upper()
12     print(f"Frukt: {frukter[i]}")
13
14 i = 0
15 while i < len(frukter):
16     print(f"Frukt: {frukter[i]}")
17     i += 1
18
19 for frukt, farge in frukt_farge.items():
20     print(f"Frukten {frukt} er {farge}.")
```

# Løkker i Python (typisk feil)



```
1 frukter = ["eple", "banan", "kiwi"]
2
3 for i in range(frukt):
4     frukter[i] = frukter[i].upper()
5     print(f"Frukt: {frukter[i]}")
6     i = i + 1
```

# Løkker i Python (teori del 2)

- Nyttige funksjoner og metoder
  - `range(start, stop)`
    - For indekserte løkker.
  - `enumerate(iterable)`
    - For indeks og verdi samtidig.
  - `len(iterable)`
    - For å vite antall elementer.
- «Løkkeflyt»
  - `break`
    - For avslutte en løkke.
  - `return`
    - Brukes for å avslutte løkker innenfor funksjoner.
  - `continue`
    - Hopp til neste iterasjon.
  - `pass` og `...` Ingen operasjon, placeholder.

```
1  for nummer in range(10):
2      if nummer == 5:
3          break
4      print(nummer)
5  # Printer tall fra 0 til 4 og stopper ved 5
6
7  for nummer in range(10):
8      if nummer % 2 == 0: # Hopper over partall
9          continue
10     print(nummer) # Printer kun oddetall
11
12 for nummer in range(10):
13     if nummer % 2 == 0:
14         pass # Gjør ingenting for partall
15     else:
16         print(f"Oddetall: {nummer}")
17
```



# Løkker i Python (teori del 3)

- **Nøstede Løkker**

- Løkker inne i andre løkker.
- "Indre løkke" kjører for hver iterasjon av "ytre løkke".

- **Bruksområder**

- Arbeide med flerdimensjonale datastrukturer.
- Utføre komplekse beregninger.
- Kombinere elementer fra ulike samlinger.

- **Eksempel på Bruk**

- Matematiske tabeller (f.eks., multiplikasjonstabell).
- Behandle matriser og rutenett.
- Sammenligne elementer mellom to lister.

- **Viktig å Huske**

- **BRUK FOR-LØKKER OM DU KAN!**
- Ytelse: nøstede løkker er «trege»
- Kan være forvirrende.
  - Vurder å bryte ned i funksjoner for klarhet.

```
1  grid = [  
2      [2, 3, 5],  
3      [1, 4, 7],  
4  ]  
5  
6  rows = len(grid) # 2  
7  cols = len(grid[0]) # 3  
8  
9  grid2 = []  
10  
11 for row in range(rows):  
12     row2 = []  
13     for col in range(cols):  
14         celle = grid[row][col]  
15         row2.append(celle*2)  
16     grid2.append(row2)  
17  
18 print(f"{grid2 = }")  
19 # grid2 = [[4, 6, 10], [2, 8, 14]]
```

# Slicing i Python (teori)


- `liste[start:slutt:steg]`
  - Henter en del av listen fra start til slutt, med gitt stegstørrelse
- `streng[start:slutt]`
  - Henter en del av strengen fra start til slutt

```
1 liste = list(range(0, 20)) # 0 t.o.m 19
2 streng = "Hello world!"
3
4 første_halvdel = liste[0:10]
5 print(første_halvdel) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
6 bare_partall = liste[:10:2]
7 print(bare_partall) # [0, 2, 4, 6, 8]
8 fem_til_15 = liste[5:15]
9 print(fem_til_15) # [5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
10
11 hello = streng[:5]
12 print(hello) # "Hello"
13 world = streng[6:]
14 print(world) # "world!"
```



# Slicing i Python (oppgaver)

- Oppgave:
  - Skriv ferdig programmet slik at ved å bruke «streng» og «ekstra» får «ønsket\_streng»



```
1  streng = "abcd"
2  ekstra = "XY"
3  ønsket_streng = "abXYcd"
4
5  ... == ønsket_streng
```



# Strenger i Python (teori)


## • Strengmetoder

- ``upper()`` / ``lower()``
  - Gjør om strengen til store/små bokstaver
- ``strip()``
  - Fjerner whitespace fra begynnelsen og slutten av strengen
- ``split(separator)``
  - Deler opp strengen i en liste basert på separator
- ``join(liste)``
  - Slår sammen elementene i en liste til en streng med gitt separator

```
1 print("Tekst meD sMå Og STOrE BokSTaver".lower())
2 # "tekst med små og store bokstaver"
3
4 print(" \n    Litt vel mye mellomrom her    \n".strip())
5 # "Litt vel mye mellomrom her"
6
7 print("Litt vel mye mellomrom fortsatt".replace(" ", ""))
8 # "Littvelmyemellomromfortsatt"
9
10 print("".join(["liste", "av", "strings", "til", "en", "string"]))
11 # "listeavstringstilenstring"
12
13 print("tilbake til en liste av strings".split())
14 # ['tilbake', 'til', 'en', 'liste', 'av', 'strings']
15
```

# Strenger i Python (oppgaver)

- Oppgave:
  - Skriv ferdig programmet slik at ved å bruke «streng» og strengmetoder får «ønsket\_streng»



```
1  streng = "  abXYcd  \n"  
2  ønsket_streng = "abxycd"  
3  
4  ... == ønsket_streng  
5
```

# Strenger i Python (typisk feil)



```
1 streng = "    abXYcd    \n"  
2 ønsket_streng = "abxycd"  
3  
4 streng.lower().strip()  
5  
6 print(streng == ønsket_streng)
```

# Funksjoner i Python (teori)

- Bryter opp strukturen av programmet
- Samle kode som brukes flere ganger
  - Gjenbruk
- Økt lesbarhet
- Viktig med gode funksjonsnavn
  - Og kommentarer
- Funksjoner defineres med ``def``
- ``return`` eller ``print``?

```
1  def addere(x, y):
2      return x + y
3
4  print(addere(2,5))
5  # 7
6  print(addere("femti","to"))
7  # "femtito"
8
9  def dividere(x, y):
10     return x / y
11
12  print(addere(10, dividere(10, 2)))
13  # 15.0
```

# Funksjoner i Python (eksempler/typiske feil)



```
1 # Eks 1
2 def dividere(x, y):
3     x / y
4
5 print(dividere(20,2))
```



```
1 # Eks 2
2 def dividere(x, y):
3     print(x / y)
4
5 print(dividere(20,2))
```



```
1 # Eks 3
2 def dividere(x, y):
3     return x / y
4
5 print(dividere(20,2))
```



```
1 # Eks 4
2 def alle_oddetall(numre):
3     for tall in numre:
4         if tall % 2:
5             return tall
6
7 numre = list(range(10))
8 print(alle_oddetall(numre))
```

# Funksjoner i Python (eksempler/typiske feil)



```
1 # Eks 1
2 def dividere(x, y):
3     x / y
4
5 print(dividere(20,2)) # None
```



```
1 # Eks 2
2 def dividere(x, y):
3     print(x / y)
4
5 print(dividere(20,2))
6 # 10.0
7 # None
```



```
1 # Eks 3
2 def dividere(x, y):
3     return x / y
4
5 print(dividere(20,2)) # 10.0
```



```
1 # Eks 4
2 def alle_oddetall(numre):
3     for tall in numre:
4         if tall % 2:
5             return tall
6
7 numre = list(range(10))
8 print(alle_oddetall(numre)) # 1
```

# Feilhåndtering i Python (teori)

- Feil håndteres av `try/except` - blokker!
- Brukes når vi forventer visse feil
  - Vi kan hindre programmet i å stoppe
  - Gir oss kontroll
- **Grunnleggende håndtering:**
  - Bruk `try`-blokken for å omslutte kode som kan forårsake feil.
  - `except`-blokken fanger og håndterer feilene.
    - Flere `except`-blokker kan fange spesifikke feiltyper.

```
1  try:
2      tall = int(input("Skriv inn et tall: "))
3      resultat = 10 / tall
4  except:
5      print("Det har skjedd en feil")
6
7  try:
8      tall = int(input("Skriv inn et tall: "))
9      resultat = 10 / tall
10 except ZeroDivisionError:
11     print(f"Kan ikke dele med null.")
12 except ValueError:
13     print(f"Vennligst skriv inn et gyldig tall.")
14 except Exception as e:
15     print(f"En uventet feil oppstått: {e}")
```

# Lese og skrive til fil i Python (teori)


- Bruk `open(filnavn, modus)` for å åpne en fil for lesing eller skriving
- Bruk `with` - god praksis!
  - Lukker filen automatisk
- Behandle potensielle feil med try/except blokker.
- Typiske feil:
  - Kjøre python-filen i feil mappe
  - Overskrive samme fil
  - Feil i filsti

```
1 relativ_filsti = './prosjekt/data.txt'
2 absolutt_filsti = '/home/brukernavn/dokumenter/prosjekt/data.txt'
3
4 # Skrive til en fil
5 with open(relativ_filsti, 'w') as fil:
6     fil.write('Tekst i fil')
7
8 # Lese fra en fil
9 with open(relativ_filsti, 'r') as fil:
10     innhold = fil.read()
11     print(innhold)
12 # Skriver ut 'Tekst i fil'
```



# Kodesporing på eksamen


- Oppgave:
  - Hva skriver dette programmet ut?  
Hvis programmet krasjer, skriv kun 'Error'.



```
1  x = 5
2  y = 2
3  x += y
4  y = x - y
5  x *= 2
6  print(x // y)
7
```

# Kodesporing på eksamen

- **Forstå oppgavens kontekst**
  - Les hele oppgaven nøye før du begynner.
  - Identifiser hva som blir spurt om.
- **Bryt ned koden i små deler**
  - Ta for deg én linje om gangen.
  - Forstå hvert enkelt kodeutsagn og dets effekt.
- **Spor variablene**
  - Skriv ned verdien av hver variabel ved siden av koden.
- **God tid?**
  - Gå tilbake og dobbeltsjekk!



```
1  x = 5
2  y = 2
3  x += y
4  y = x - y
5  x *= 2
6  print(x // y)
7
```

# Forklaringer på eksamen

- **Kodesporing:**
  - Samme som sist, men sett ord på prosessen
  - Øv
- **Fagbegreper/termologi:**
  - Bruk kursnotatene
  - Pugg
- Bruk klart og konsist språk.

## (b) Destruktive og ikke-destruktive funksjoner i snake (10 poeng)

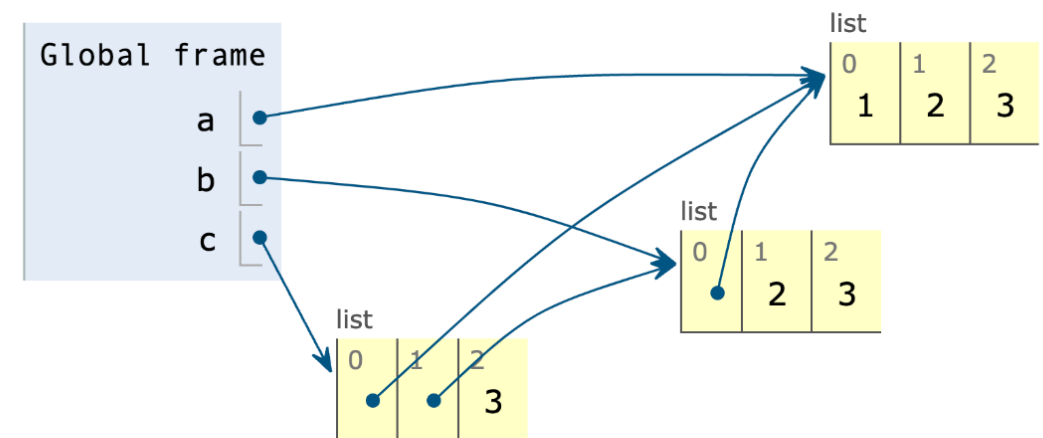
Forklar forskjellen på en destruktiv og en ikke-destruktiv funksjon. Vis til eksempler på begge deler i det vedlagte løsningsforslaget til lab6.

# Kodeskriving uten editor

- Les oppgaven nøye!
- Planlegg
- Bryt programmet ned i mindre deler
  - Steg for steg
- Holde hode kaldt
- Begynn! – Ikke overtenk
- Kommenter underveis?
- Fokuser på algoritmens flyt fremfor syntaks
- Husk på notatene!
- Kjør koden mentalt etterpå

(b) Opprett lister og referanser (4 poeng)

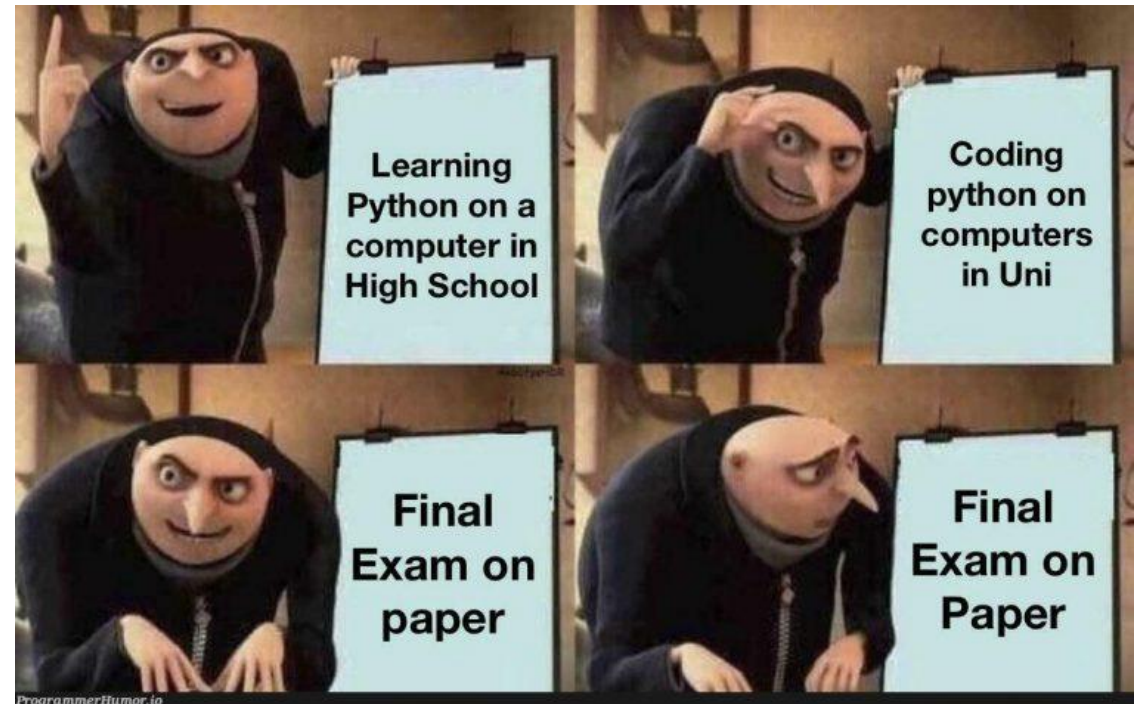
[? Oppgavetekst](#)



Opprett tre variabler a, b og c, slik at minnets tilstand blir som vist på bildet over.

# Frem mot eksamen

- Øv på å programmere!
  - Selv om eksamen ikke lar deg kjøre kode, forsøker vi likevel å teste din evne til nettopp å programmere.
- Har du ikke gjort alle lab-oppgavene fra tidligere? Prøv dem!
  - Har du ikke gjort de ekstra listeoppgavene? Prøv dem også!
- Se på tidligere eksamener.
- Lag en liste over hva du har gjort på hver lab til nå!
- Youtube Tutorials
- Spør mye teite spørsmål på Discord



# Takk!

Masse lykke til på eksamen!