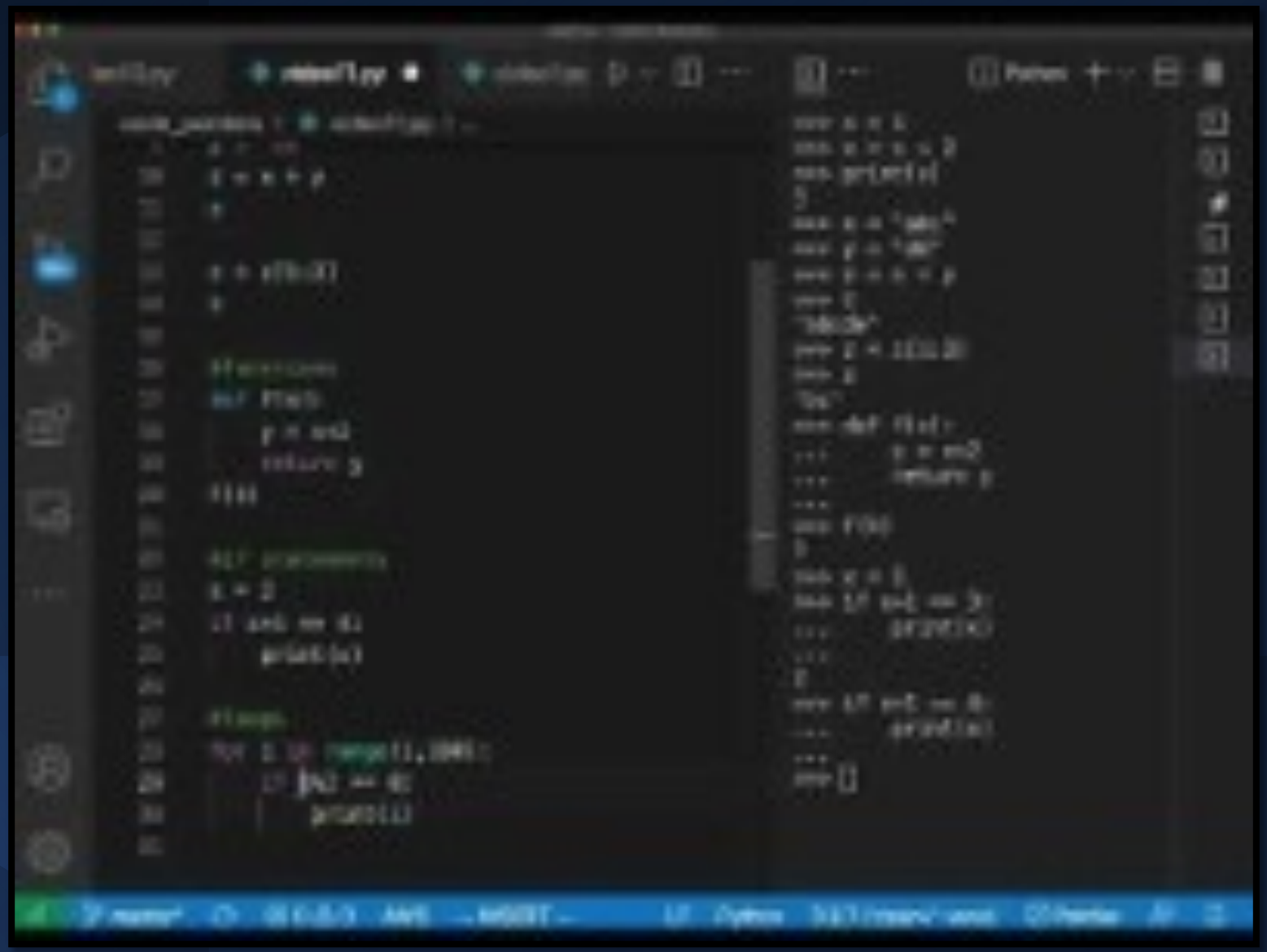
INF100 KRÆSJKURS H23

m/Gnist

Av Aleksander L. Fedøy



Plan for kurset

- **Python-pensum:**
  - Typer/datatyper/utrykk (med print)
  - Betingelser, boolske utrykk 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`)
  - Tupler (`tuple`)
  - Sett/mengder (`set`)
  - Oppslagsverk/ordbok (`dict`)
- **Spesialtyper**
  - "Ingenting"/"ingen verdi" (`None`)

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

Datatyper i Python (eksempler)

```python
# Eksempel

print("Hva er din alder?")

alder = input()

ny_alder = alder + 10

print("Om 10 år er du "+ny_alder+" år gammel!")
```

Datatyper i Python (eksempler)

```
ubuntu@Matebook14:~/05_UIB/INF-100-Krasjkurs$ /bin/python /home/ubuntu/05_UIB/INF-100-Krasjk
urs/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 (eksempler)

```
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
`f"{c}"`	o	o	o	o	o	o	o	o
`e and not e`	o	o	o	o	o	o	o	o
`a[0]`	o	o	o	o	o	o	o	o
`d["c"]`	o	o	o	o	o	o	o	o

Datatyper i Python (oppgaver)

```
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
`a in a`	o	o	o	o	o	o	o	o
`a[0:1]`	o	o	o	o	o	o	o	o
`b + "a[0]"`	o	o	o	o	o	o	o	o

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
`a in a`							X	
`a[0:1]`					X			
`b + "a[0]"`						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

```python
pris_per_enhet = 5.98765
antall_enheter = 8

print(f"Totalpris: {pris_per_enhet * antall_enheter:.2f} kr")
# Totalpris: 47.90 kr

# Vanlig feil: Glemmer å avslutte uttrykket før formattering
print(f"Totalpris: {pris_per_enhet * antall_enheter.2f} kr")
# Dette vil gi en syntax error
```

Skrive ut i Python (oppgaver)

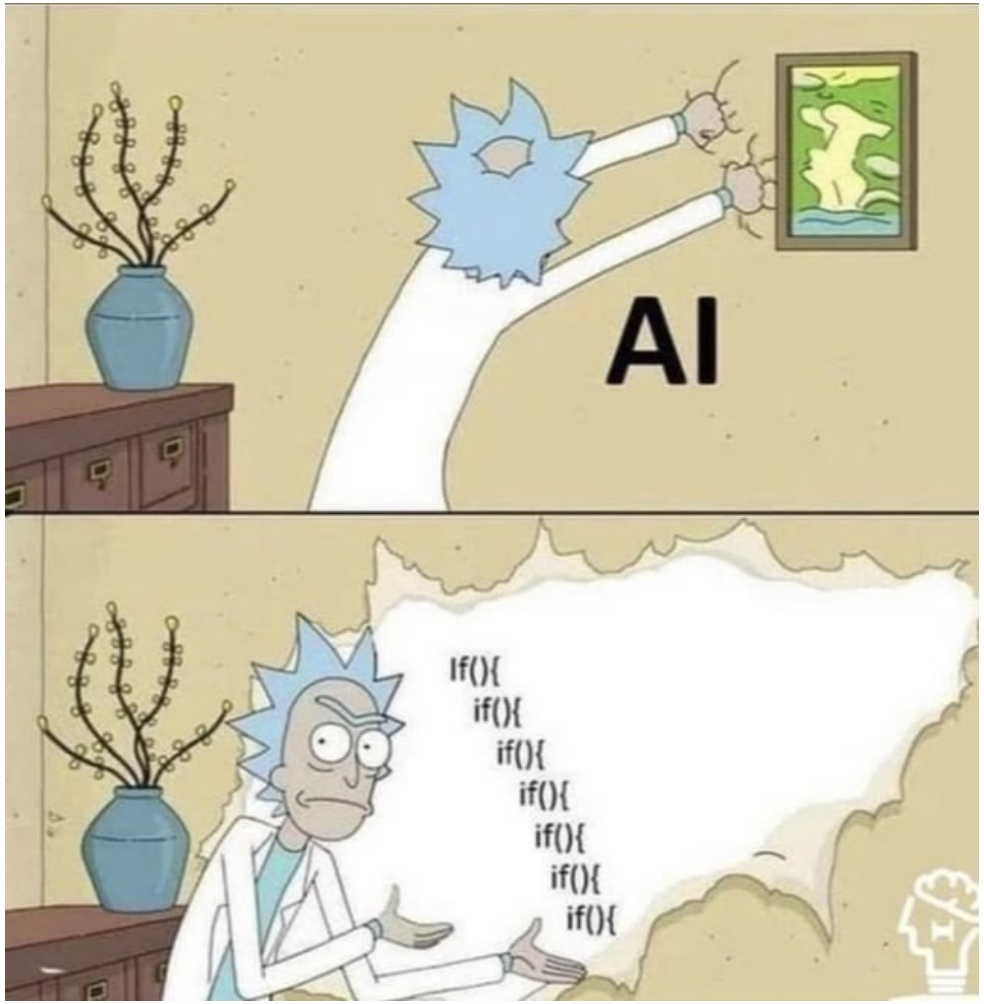
- **Oppgave:**
  - **Skriv ferdig programmet slik at det skriver ut ønsket «output»**

```python
# Avrund pi til to desimaler
pi = 3.14159265

print(...)

# Ønsket output: "pi = 3.14"
```

Betingelser, boolske utrykk og presedens (teori)



Betingelser, boolske utrykk 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 `()` brukes for å gruppere uttrykk og overstyre presedens.

```python
temperatur = 25

if temperatur > 25:
    print("Det er varmt")
elif temperatur >= 10:
    print("Husk jakke")
    # printes
else:
    print("Huff, så kaldt!")

vær = "regn"

if temperatur > 25 and vær == "sol":
    print("Husk solkrem")
elif temperatur >= 10 or vær != "regn":
    print("Norsk sommer :-)")
    # printes
elif temperatur is None or not vær:
    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

```python
liste = [1, 2]
print(liste[1]) # 2
liste.append(2.77)
print(liste) # [1, 2, 2.77]
liste2 = ["pi", True]
liste.extend(liste2)
print(liste) # [1, 2, 2.77, "pi", True]
liste.insert(2, 25)
print(liste) # [1, 2, 25, 2.77, 'pi', True]
liste.remove("pi")
print(liste) # [1, 2, 25, 2.77, True]
liste.sort()
print(liste) # [1, True, 2, 2.77, 25]
liste.reverse()
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.

```python
frukter = ["eple", "banan", "kiwi"]
frukt_farge = {
    "eple": "rødt",
    "banan": "gul",
    "kiwi": "grønn"
    }

for frukt in frukter:
    print(f"Frukt: {frukt}")
for i in range(len(frukter)):
    frukter[i] = frukter[i].upper()
    print(f"Frukt: {frukter[i]}")

i = 0
while i < len(frukter):
    print(f"Frukt: {frukter[i]}")
    i += 1

for frukt, farge in frukt_farge.items():
    print(f"Frukten {frukt} er {farge}.")
```

Løkker i Python (typisk feil)

```python
frukter = ["eple", "banan", "kiwi"]

for i in range(frukt):
    frukter[i] = frukter[i].upper()
    print(f"Frukt: {frukter[i]}")
    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.

```python
for nummer in range(10):
    if nummer == 5:
        break
    print(nummer)
# Printer tall fra 0 til 4 og stopper ved 5

for nummer in range(10):
    if nummer % 2 == 0:  # Hopper over partall
        continue
    print(nummer)  # Printer kun oddetall

for nummer in range(10):
    if nummer % 2 == 0:
        pass  # Gjør ingenting for partall
    else:
        print(f"Oddetall: {nummer}")

```


I'm a programmer
Which programming Languages You Use
Chatgpt

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.

```python
grid = [
    [2, 3, 5],
    [1, 4, 7],
]

rows = len(grid) # 2
cols = len(grid[0]) # 3

grid2 = []

for row in range(rows):
    row2 = []
    for col in range(cols):
        celle = grid[row][col]
        row2.append(celle*2)
    grid2.append(row2)

print(f"{grid2 = }")
# 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

```python
liste = list(range(0, 20)) # 0 t.o.m 19
streng = "Hello world!"

første_halvdel = liste[0:10]
print(første_halvdel) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
bare_partall = liste[:10:2]
print(bare_partall) # [0, 2, 4, 6, 8]
fem_til_15 = liste[5:15]
print(fem_til_15) # [5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

hello = streng[:5]
print(hello) # "Hello"
world = streng[6:]
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

```python
print("Tekst meD sMå Og STOre BoksTaver".lower())
# "tekst med små og store bokstaver"

print(" \n    Litt vel mye mellomrom her      \n".strip())
# "Litt vel mye mellomrom her"

print("Litt vel mye mellomrom fortsatt".replace(" ",""))
# "Littvelmyemellomromfortsatt"

print("".join(["liste", "av", "strings", "til", "en", "string"]))
# "listeavstringstilenstring"

print("tilbake til en liste av strings".split())
# ['tilbake', 'til', 'en', 'liste', 'av', 'strings']

```

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)

```python
streng = "    abXYcd    \n"
ønsket_streng = "abxycd"

streng.lower().strip()

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`?

```python
def addere(x, y):
    return x + y

print(addere(2,5))
# 7
print(addere("femti","to"))
# "femtito"

def dividere(x, y):
    return x / y

print(addere(10, dividere(10, 2)))
# 15.0
```

Funksjoner i Python (eksempler/typiske feil)

```python
# Eks 1
def dividere(x, y):
    x / y

print(dividere(20,2))
```

```python
# Eks 2
def dividere(x, y):
    print(x / y)

print(dividere(20,2))
```

```python
# Eks 3
def dividere(x, y):
    return x / y

print(dividere(20,2))
```

```python
# Eks 4
def alle_oddetall(numre):
    for tall in numre:
        if tall % 2:
            return tall

numre = list(range(10))
print(alle_oddetall(numre))
```

Funksjoner i Python (eksempler/typiske feil)

```python
# Eks 1
def dividere(x, y):
    x / y

print(dividere(20,2)) # None
```

```python
# Eks 2
def dividere(x, y):
    print(x / y)

print(dividere(20,2))
# 10.0
# None
```

```python
# Eks 3
def dividere(x, y):
    return x / y

print(dividere(20,2)) # 10.0
```

```python
# Eks 4
def alle_oddetall(numre):
    for tall in numre:
        if tall % 2:
            return tall

numre = list(range(10))
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.

```python
try:
    tall = int(input("Skriv inn et tall: "))
    resultat = 10 / tall
except:
    print("Det har skjedd en feil")

try:
    tall = int(input("Skriv inn et tall: "))
    resultat = 10 / tall
except ZeroDivisionError:
    print(f"Kan ikke dele med null.")
except ValueError:
    print(f"Vennligst skriv inn et gyldig tall.")
except Exception as e:
    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

```python
relativ_filsti = './prosjekt/data.txt'
absolutt_filsti = '/home/brukernavn/dokumenter/prosjekt/data.txt'

# Skrive til en fil
with open(relativ_filsti, 'w') as fil:
    fil.write('Tekst i fil')

# Lese fra en fil
with open(relativ_filsti, 'r') as fil:
    innhold = fil.read()
    print(innhold)
# Skriver ut 'Tekst i fil'
```

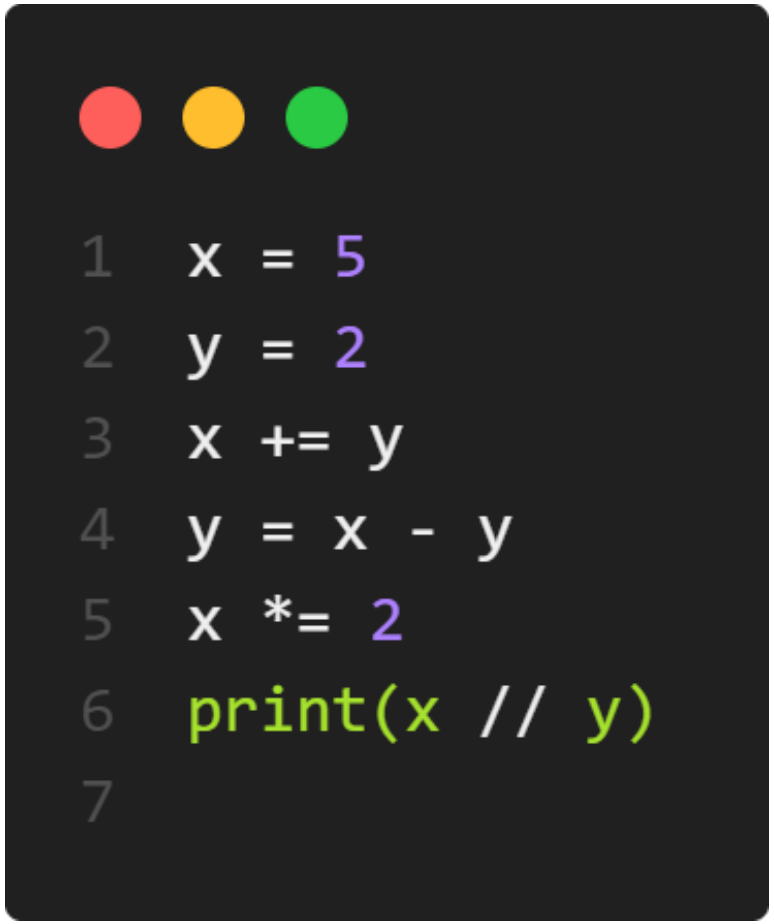
Kodesporing på eksamen

- **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!



Forklaringer på eksamen

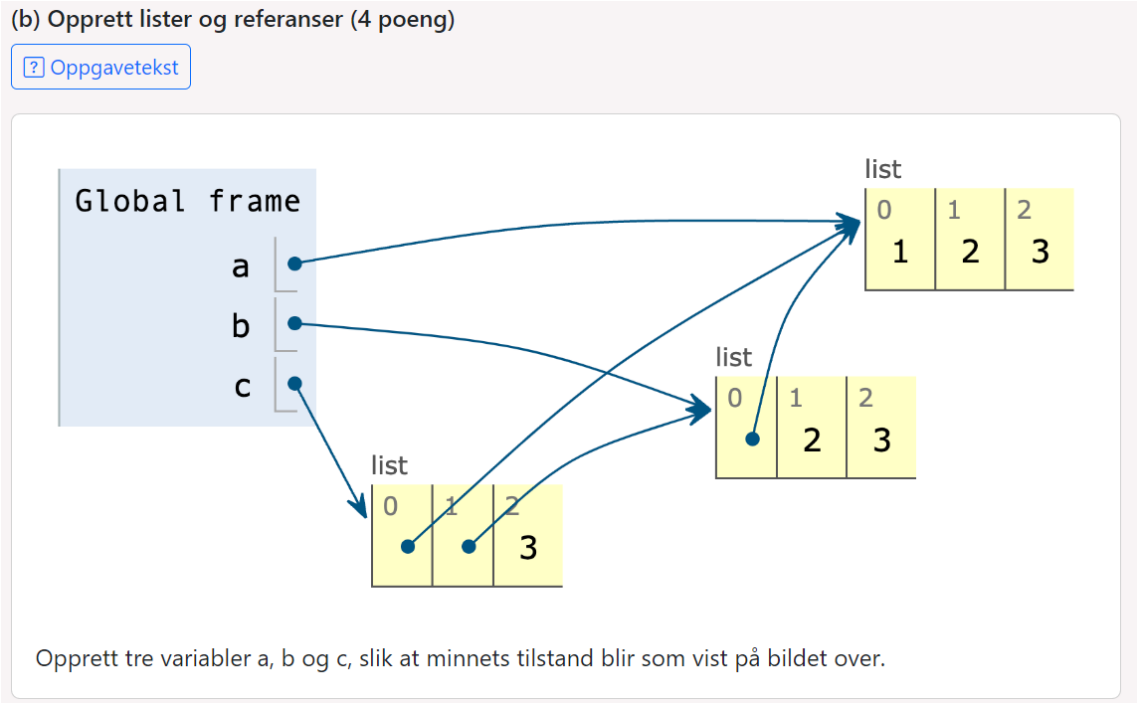
- **Kodesporing:**
  - Samme som sist, men sett ord på prosessen
  - Øv
- **Fagbegreper/termologi:**
  - Bruk kursnotatene
  - Pugg
- Bruk klart og konsist språk.

(b) Destruktive og ikke-destrukive 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.

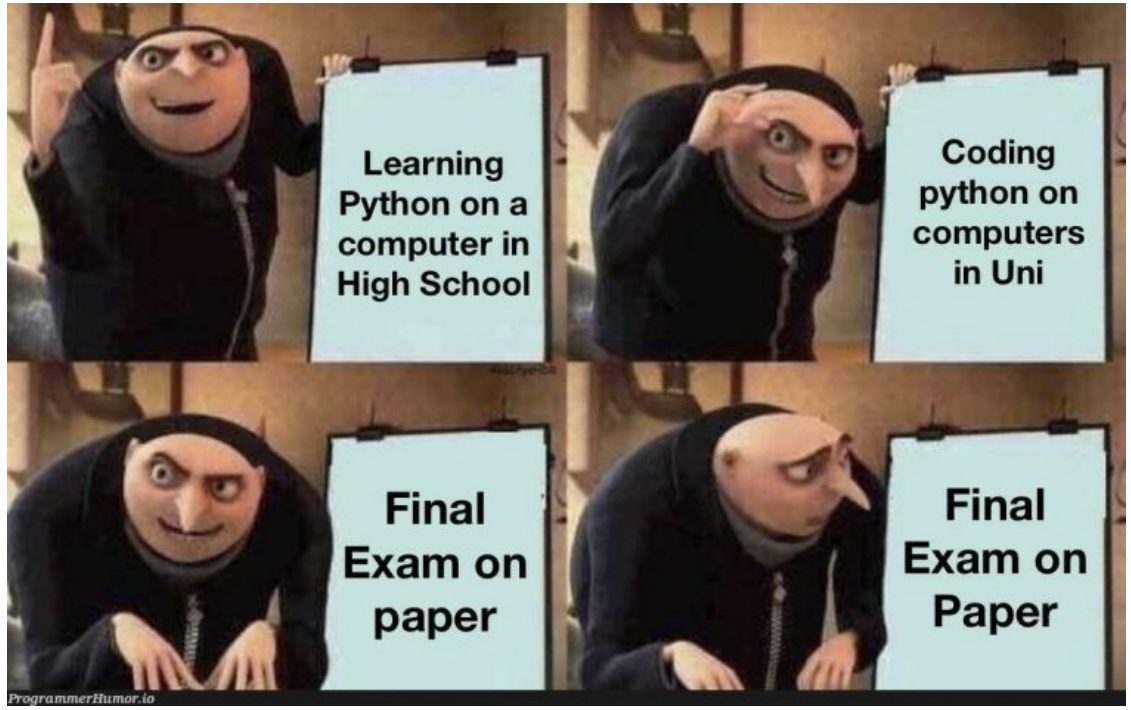
Kodeskriving uten editor

- 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



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!