Forkurs IN1000

dag 2: programmering

Innhold i del 2:

- Litt studieteknikk
- Programmeringsspråk (installere python)
- Editor (installere VS Code), Terminal og Debugger
- Feilsøking (“debugging”)
- Levere en oppgave i Devilry

Litt studieteknikk

- Mange forelesere legger ut forelesningsnotater (slides) før eller etter forelesning – i så fall trenger du ikke notere alt som vises der underveis i forelesningen
- Hvis du programmerer i forelesningen (for eksempel live-koding), kan du skrive forklarende kommentarer direkte i programmet ditt (her i grønt):

```
1    # Venstre side av likhetstegn: variabel
2    # Høyre side av likhetstegn: verdi (eller utregning)
3    beløp = 100
```

Litt studieteknikk

- Programmering handler ikke om å pugge hvordan hver minste ting skrives (syntaks)
- Det viktigste er å forstå hva ting gjør, hva de kan brukes til og ikke brukes til
- På eksamen kan du ha med skrevne notater (inkludert kode-eksempler) og trykte hjelpemidler (inkludert lærebok) slik at du ikke trenger å huske alt utenat
- Men du trenger å forstå hva oppgavene handler om og hva som skjer i maskinen når koden kjøres

Bruk og misbruk av kunstig intelligens (AI)

- Skal du bruke AI, bruk det som et hjelpemiddel til å lære, ikke som en snarvei til å gjøre oppgaver uten å lære noe
- Du får ikke brukt AI på eksamen, da må du stole på det *du* har lært – har du brukt AI som en krykke gjennom semesteret kommer du til å få det vanskelig
- Det er spesielt viktig at du øver deg på selv å finne feil i programmene dine uten hjelp
- OBS: AI er ofte upålitelig og tar feil – men likevel svært overbevisende. Dette er en skummel kombinasjon
- En god måte å bruke AI på er å be den lage spørsmål eller oppgaver til deg som du kan bruke til å teste om du har forstått et tema. For eksempel:

Lag tre korte oppgaver uten løsningsforslag som viser at jeg forstår if og else i Python

Selvfølgelig! Her er tre korte oppgaver for å teste forståelsen din av `if` og `else` i Python:

Oppgave 1:

Skriv et program som ber brukeren om et bruker navn. Hvis navnet er "admin", skriv ut "Velkommen, admin". Hvis navnet ikke er "admin", skriv ut "Hei, [navn]".

Oppgave 2:

Lag et program som leser inn en brukerens karakter (en bokstav fra A til F). Hvis karakteren er A, B eller C, skriv ut "Bestått" og ellers skriv ut "Ikke bestått".

Oppgave 3:

Skriv et program som lar brukeren skrive inn en temperatur i Celsius. Hvis temperaturen er 0 grader eller kaldere, skriv ut "Det er frost". Hvis temperaturen er høyere enn 0 grader, skriv ut "Ingen frost".

Lykke til med oppgavene!

Gi tilbakemelding på hva jeg har forstått i denne løsningen til oppgave 1:

```
navn = input("Skriv navn:")
if navn == "admin":
    print("Velkommen, admin")
if navn != "admin":
    print("Hei", navn)
```

Forbedringsmuligheter:

Selv om koden din fungerer, kan den optimaliseres ved å bruke en `if-else`-konstruksjon. Dette fordi når du bruker to separate `if`-setninger, blir begge sjekket uavhengig av hverandre, noe som i dette tilfellet ikke er nødvendig. `if-else`-konstruksjonen gir en klarere struktur og sparer ressurser ved å unngå unødvendige sjekker.

Optimalisert løsning:

```
navn = input("Skriv navn:")
if navn == "admin":
    print("Velkommen, admin")
else:
    print("Hei", navn)
```

Hva dette viser:

- **Riktig bruk av `else`:** Du viser at du forstår forskjellen mellom `if` og `else` ved å sørge for at en av de to blokkene alltid vil kjøre, men aldri begge.
- **Klarere logikk:** `else` gir en tydelig indikasjon på at det finnes et alternativ tilfelle som skal kjøres når `if`-betingelsen ikke er oppfylt.

Alt i alt har du en god forståelse av bruken av `if`. Ved å bruke `else` i stedet for en ny `if` kan koden bli mer effektiv og lett forståelig. Flott jobbet!

Det er mye forståelse i en god metafor

Oppsummer if og else i Python slik en sjørøverkaptein ville gjort det

Arrr, lend me yer ears, mateys, for I'm about t' tell ye the tale of the mighty `if` and `else` in the land o' Python!

If - The Lookout:

"Arrr, when yer out at sea, ye need a lookout! `if` be yer trusty lookout, always keepin' an eye for th' condition ye give 'im! If he spies the condition be true, he hollers loud:

```
if the_sail == 'ripped':
    print("Hoist the mainsail!")
```

Else - The First Mate: "But wait, mateys, if the lookout sees naught, ye got yer first mate `else` ready t' take the helm! He steps in if the lookout sees no trouble."

```
if the_sail == 'ripped':
    print("Hoist the mainsail!")
else:
    print("Keep sailing steady!")
```

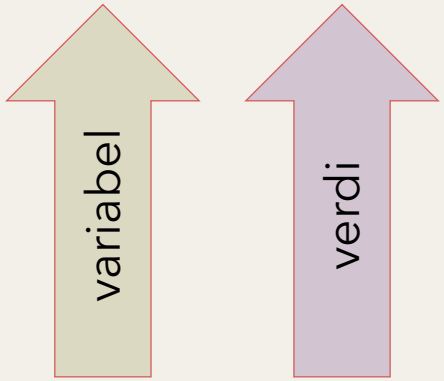
The Tale Complete: "So here it be, me hearties, when ye write this Python code, ye tell the lookout (`if`) to be sharp, and if he sees nothin', ye first mate (`else`) takes charge! Together they keep th' ship sailin' smooth, no matter what th' seas throw at ye!"

```
treasure = int(input("How many gold doubloons did ye find? "))
if treasure > 100:
    print("We be rich! Celebrate, ye scallywags!")
else:
    print("Keep huntin', we need more gold!")
```

Arrr, now be gone with ye and write yer scripts true as any seadog worth his salt!

Hvordan lese programkode – tre nivåer

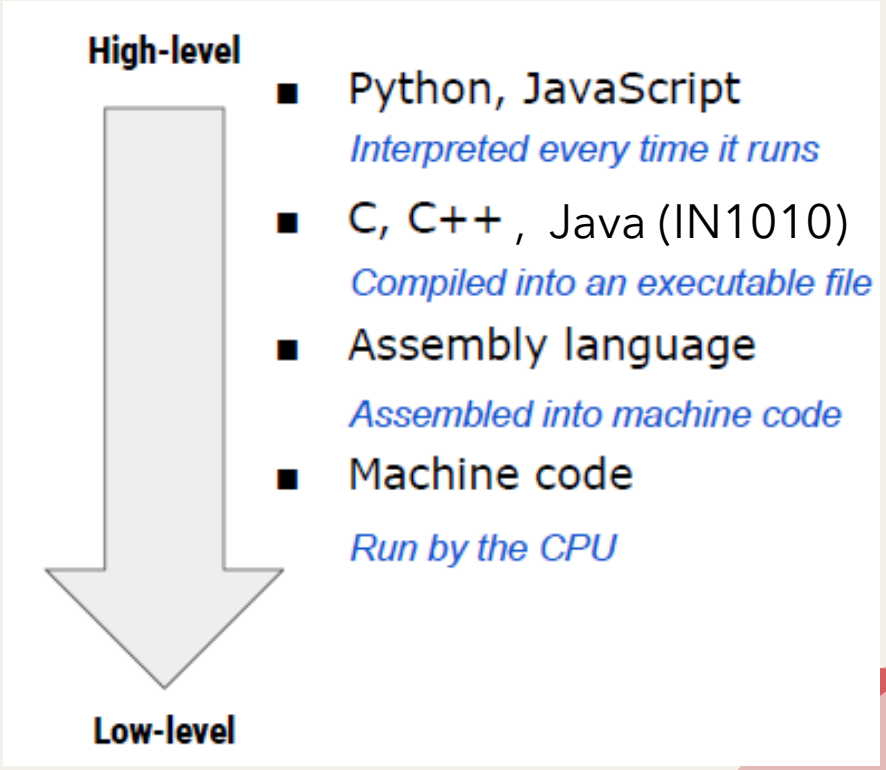
```
1    beløp = 100
2    renter = (5*beløp/100)
3    beløp = beløp + renter
```



1. **Problem-nivået**
   - Hva slags problem er det dette programmet forsøker å løse?
   - Navn på variabler (“beløp”, “renter”) er hjelpsomme her
2. **Syntaks-nivået: ‘gramatikken’ i Python**
   - Tallene til venstre er linjenummer, ikke en del av programmet
   - 3 *tilordninger* på formen: *variabel* = *verdi*
3. **Maskin-nivået: hva skjer når vi kjører koden?**
   - En linje kjøres av gangen
   - Når vi skriver *renter* senere, vil maskinen ha lagret i minnet at dette betyr verdien 5
   - *beløp* får først én verdi (100) og litt senere en ny (105) – da er det den siste verdien som er lagret i minnet, den første dermed er overskrevet (“glemt”).

Det finnes forskjellige språk å lage programmer i

- Tolket kode / skript (høynivå): Et program som oversettes til maskinkode hver gang det skal kjøres. Treigest, men også enklest for mennesker.
- Kompilert kode (middels): Et program som oversettes til maskinkode på forhånd av et spesielt program (kompilator). Treigere, men enklere.
- Assembly-kode (lavnivå): Ekstremt detaljert (men raskt), det laveste nivået som er noenlunde forståelig for mennesker (IN1020)
- Maskinkode (helt på bunn): Kjøres direkte på den fysiske datamaskinen (hardwaren) : 00101001…



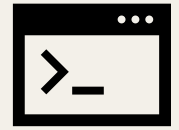
Litt om Python

- Python er et høynivå-språk: Ikke så raskt, men lettere å lese og lære seg
- Et av de mest brukte språkene – det finnes mye hjelp og mange programbiblioteker som kan brukes
- Men **python** er også et program som tolker og kjører programmer skrevet i Python

Tre grunnleggende verktøy:

- **Editor:** Programmet som vi bruker til å skrive programmer
- **Terminal:** Del av OSet (som vi så på i del 1) hvor vi kan kjøre Python-programmer

  (programmene våre kan også åpne vinduer, men det er mer avansert)
- **Debugger:** Verktøy som lar oss se hvordan programmet fungerer i maskinen, og som hjelper oss med å finne feil (“bugs”)

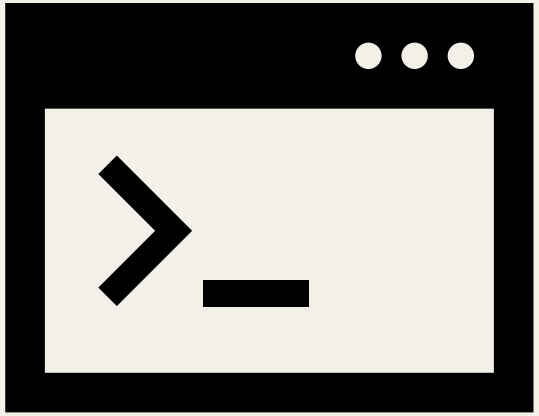


Editor: Visual Studio Code (VS Code)

- Program som vi bruker til å skrive programmer
- VS Code er den eneste editoren vi anbefaler og gir hjelp med
- Vi gir dere et oppsett som gjør editoren mer oversiktlig og tilpasset læring
- (hvis du foretrekker en annen editor kan du bruke den på eget ansvar)

Hvorfor kjøre programmer fra terminalen?

- Er det slik at alle som skal kjøre programmene våre må ha en editor?
- Nei!
- Det er viktig at andre kan kjøre koden din på sin maskin uten å ha de samme programmene som deg (det hjeper lite å si "men det funka på min maskin...")
- Det eneste du kan forvente er at **python** må være installert!



Debugger: Python Tutor

- Hjelper oss å forstå hva som skjer i maskinen når vi kjører programmet
- En og en linje kjøres av gangen
- Vi kan se variabler og verdiene deres for hvert steg
- Hjelpsomt når vi skal oppdage og rette opp feil
- (VS Code har også en innebygd debugger som er litt for avansert for oss foreløpig)

Feil ("bugs") er en helt naturlig, uunngåelig del av programmering (selv etter mange år!)

- Det er umulig å skrive feilfri programmer på første forsøk – syklusen er:
- Skriv litt kode (ikke for mye)
- Test at det fungerer
- Ofte er det en bug: Finn → forstå → fiks
  Bruk debugger!
- Test igjen etterpå – det er ofte flere bugs
- Når alle er funnet, kan vi gå videre og skrive mer kode
- Å oppdage feil er en **god ting** ☺
  (Det er mye verre om vi ikke oppdager dem)
- Det gjelder også "bugs i forståelse": Test også forståelsen deres ofte!

Etter hvert: Levere oppgaver i Devilry

- Dere vil få hjelp med dette på gruppetimene når dere kommer så langt at dere skal levere noe
- Da finner dere det på devilry.ifi.uio.no
- Der lurt å legge alt som skal leveres i en mappe og levere alle filene fra den mappen
- Husk at koden som leveres skal kunne kjøre på en annen maskin – lever **alle** filene som trengs for å kjøre den (men ikke mer enn det)!
- Lurt å ha en mappe per innlevering, med alle filene i

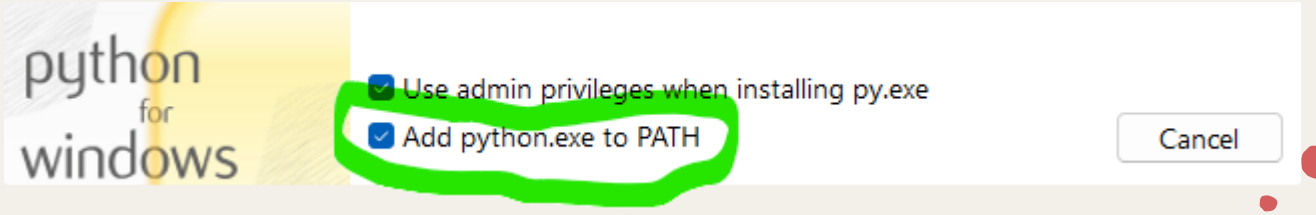
Oppgaver

spør oraklene (hjelperne på gruppetimen) om hjelp hvis du står fast ☺

Oppgave 1: Last ned og installer Python

- Last ned siste versjon av Python her:
  https://www.python.org/downloads/
  Stor gul knapp: 

- Windows: HUSK å krysse av for "Add python.exe to PATH" under installasjonen



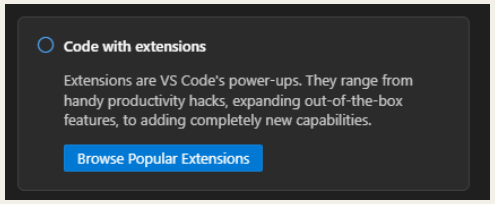
- Hvis du glemte det likevel, se her for en oppskrift på å fikse det

Oppgave 2: Installer VS Code (hvis du ønsker å bruke den anbefalte editoren)

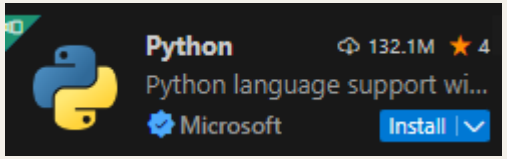
- https://code.visualstudio.com/Download

Oppgave 3: Legg inn Python-utvidelsen

- Start VS Code og velg "Browse Popular Extensions" i bildet som dukker opp:



- Den eneste utvidelsen du trenger foreløpig er denne (trykk Install):



- Du kan også installere denne utvidelsen manuelt herfra: https://marketplace.visualstudio.com/items?itemName=ms-python.python

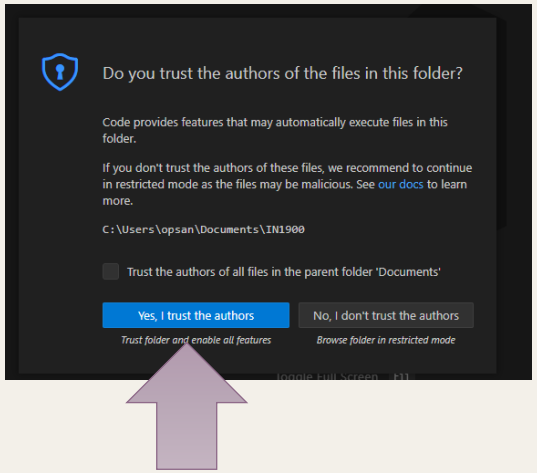
Oppgave 4: Oppsett av VS Code

- Enklere, mindre forvirrende
- Bedre tilpasset læring av Python
- Last ned denne filen: https://tinyurl.com/in1000-vscode
- Windows: File → Preferences → Settings → 
  Mac: Code → Settings → Settings → 
- Lim inn fra nedlastet fil og lagre innstillingene
- Lukk VS Code, start på nytt og sjekk at det nå ser annerledes ut

Oppgave 5: Lag et Python-program

- Åpne "Forkurs"-mappa fra i går med File → Open Folder
- Åpne en ny fil med File → New File
- Skriv inn følgende på de fire første linjene:

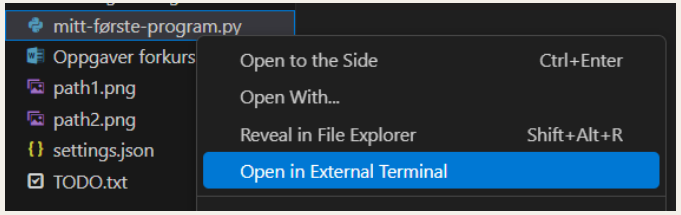
```
1   beløp = 100
2   renter = 5*beløp/100
3   beløp = beløp + renter
4   print(beløp)
```



- Lagre filen i "Forkurs"-mappa med navnet **mitt-første-program.py**

Oppgave 6: Kjør programmet ditt

- Åpne terminalen ved å høyreklikke på python-filen i utforskervinduet (View → Explorer i menyen) og velg:



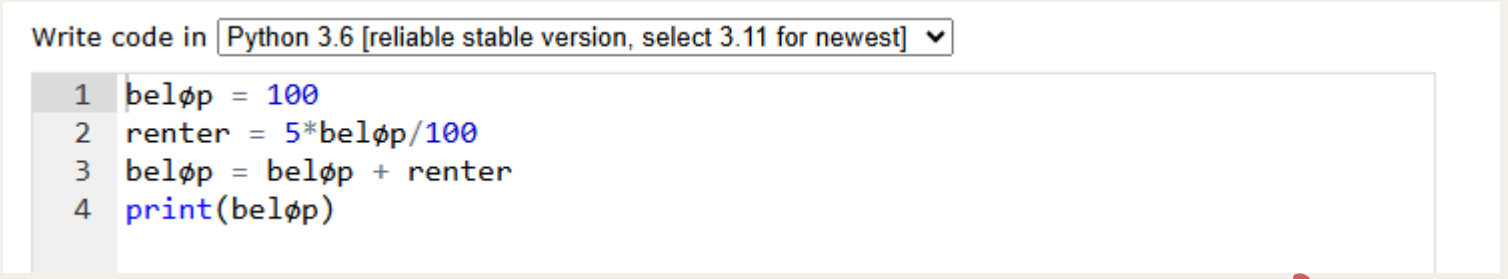
Kjør alltid programmene dine fra terminalen!

- I terminalvinduet som dukker opp, skriv:
  python mitt-første-program.py (Windows)
  python3 mitt-første-program.py (Mac og Linux)
- Hvis du skrev alt riktig, skal programmet skrive ut dette til skjermen:

```
Nytt beløp: 105.0
```

Oppgave 7: Python Tutor

- Gå til pythontutor.com
- Trykk på "Python"
- I vinduet et stykke ned, lim inn programmet ditt fra VS Code:



- Trykk Visualize Execution
- Trykk Next > for hver linje og se hva som skjer i maskinen når hver linje kjøres (se hvordan verdiene til variablene **beløp** og **renter** endrer seg for hver linje)