INF100 Innføring i programmering

Introduksjon

Crystal Chang Din
David Grellscheid
20.01.2023

UNIVERSITY OF BERGEN

Husk å slå av mikrofonen!

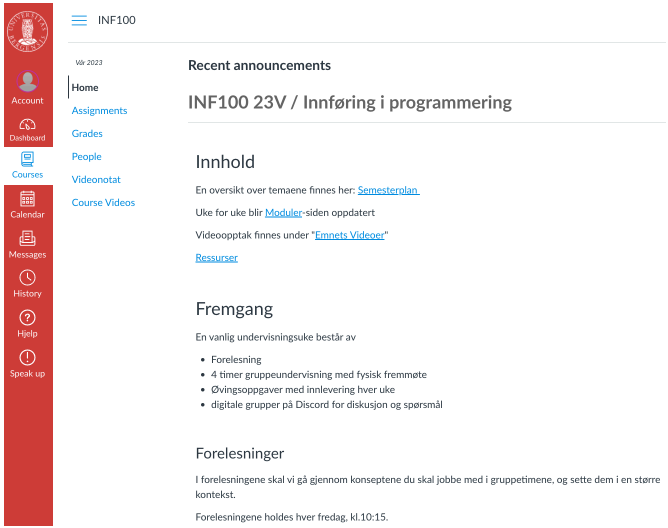
Forelesningene blir tatt opp, videoen legges ut noen timer etter

Timeplan

- Forelesning: fredag 10:15–12:00 på Zoom
- Gruppeundervisning: 4 timer fysisk (2+2)
- UiB Timeplan https://www.uib.no/emne/INF100

Kommunikasjon

- https://mitt.uib.no/courses/39806
- Discord: https://discord.gg/mDSTKFrGrP
  (så bruk #join_group med din gruppenummer)



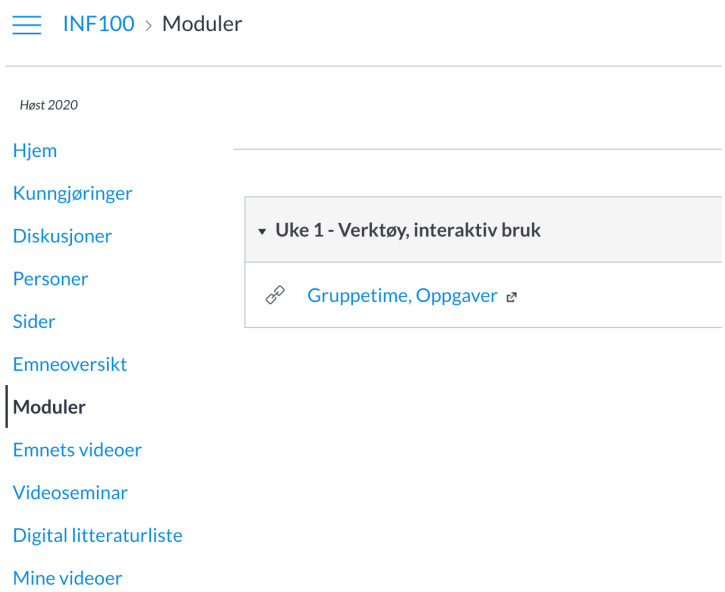
Faglige spørsmål: Discord-kanaler

Crystal.Din@uib.no
David.Grellscheid@uib.no

Høyteksenteret, Datablokken, 406P1/407P1

Organisatoriske spørsmål: studieveileder@ii.uib.no

mitt.uib -> Moduler



Oppgaver

- Øvingsoppgaver (~hver uke)
- Innlevering er obligatorisk, 9 av 12 ukentlige oppgaver pluss en større oppgave til slutten må godkjennes
- Eksamen: 26.05.2022, 09:00-13:00
  egen datamaskin med Inspera, digital eksamen på campus, alle hjelpemidler er tillatt

Automate the boring stuff

https://automatetheboringstuff.com/2e/

omtrent halvparten (kap. 0-6, 8-10)

Nyttige biblioteker

random, datetime, csv
matplotlib, numpy

Hva er informatikk?

Informatikk handler om å løse problemer
på en **algoritmisk** måte med **programmering**.

Algoritmisk problemløsning

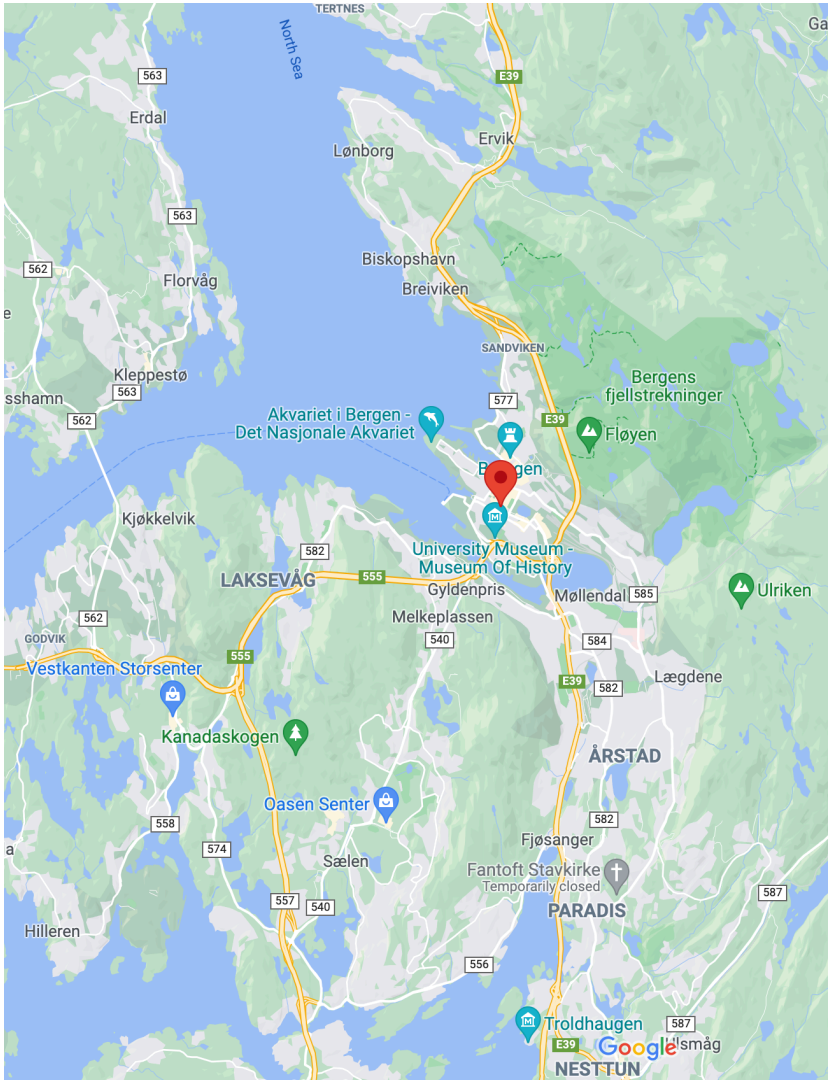
To ting som trengs for å utføre **algoritmisk problemløsning**:

- **en representasjon** som fanger opp alle relevante aspekter av problemet
- **en algoritme** som løser problemet steg for steg ved bruk av representasjonen

Bruken av abstraksjon i algoritmisk problemløsning

En representasjon som utelater detaljer om det som blir representert, er en form for abstraksjon.

Abstraksjon kan finnes overalt. For eksempel **er kart abstraksjoner**.


TERTNES
North Sea
Erdal
Lønborg
Ervik
Florvåg
Biskopshavn
Breiviken
SANDVIKEN
Kleppestø
Bergens fjellstrekninger
Akvariet i Bergen - Det Nasjonale Akvariet
Fløyen
Kjøkkelvik
University Museum - Museum Of History
LAKSEVÅG
Gyldenpris
Møllendal
Ulriken
Melkeplassen
GODVIK
Vestkanten Storsenter
Lægdene
Kanadaskogen
ÅRSTAD
Oasen Senter
Fjøsanger
Sælen
Fantoft Stavkirke
Temporarily closed
PARADIS
Hilleren
Troldhaugen
NESTTUN
Google


1 Byparken - Lagunen
2 Strandkaiterminalen - Birkelundstoppen
3 Støbotn - Sletten
4 Flaktveit - Hesjaholtet
5 Åsane terminal - Eidsvågneset - Loddefjord terminal
6 Lønborglien - Vadmyra
9 Festplassen - Kronstad
10 Mulen - Søndre Skogveien/Wergeland
11 Nordnes - Starefossen
12 Smiberget - Montana
13 Sentrum - Solheimsviken
14 Bønes - Fjøsanger - Bergen busstasjon
15 Sentrum - Øvre Kråkenes
16 Sentrum - Nipedalen
17 Sentrum - Gravdal
18 Formanns vei - Barlveien
19 Løvstakkskiftet - Sentrum
Støbotn
Viddalen
Flatevad
Rollandkrysset
IKEA
Toppe
Åsane terminal
Gullgruven
Ervik
Hesthaugen
Liakroken
Eidsvågneset
Eidsvåg
Eidsvåg E39
Flaktveitsvingane
Flaktveit
Lønborglien
Eidsvågtunnelen
Lønborg
Handelshøyskolen
Ludebryggen
Formanns vei
Sandviksgaten
Amalie Skrams vei
Kirkegaten
Skutevikstorget
Nordnes
Tollboden
Mulen
Dreggen
Ladegårdsgaten
Torget
Strandkaiterminalen
Strandkaien
Byparken
Småstrandgaten
Stadsporten
Olav Kyrres gate
Starefossen
Gravdal
Kringsjåveien
Laksevåg senter
Nygaten
Bergen busstasjon
Svarteviksveien
Sjøkrigs-skoleveien
Nygård kirke
Møhlenpris
Fest-plassen
Vadmyra
Lyngbø RV 555
Damsgårdstunnelen
Arstadveien
Statsarkivet
Vestre Vadmyra
Carl Konows gate
Nygård
Florida
Nygårdslien
Gyldenpris
Møllendalsveien
Melkeplassen
Loddefjord Terminal
Nipedalen
Lotheveien
Michael Krohns gt.
Haukeland sjukehus N
Haukeland sjukehus S
Vestlund
Solheimsviken
Solheimslien
Danmarks plass
Blekenberg
Kronstad
Haukelands-veien
Langhaugen
Barli-veien
Løvås
Løvstakk-skiften
Fabrikk-gaten
Montana
Hesjaholtet
Sikt-haugen
Brann Stadion
Sollien
Lynghaug
Oasen
Søndre Skogveien
Landås-torget
Helgeplasset
Kanalveien
Hagerups vei
Skarp-haugen
Mindeveien snuplass
Wergeland
Mannsverk garasje
Torgny Segerstedts vei
Sletten senter
Bråtet
C. J. Hambros vei
Slettebakken
Bolstad
Fjøsangerveien
Nattlandsveien
Fjøsanger
Birkelundstoppen
Bønestoppen
Fantoft
Sletten
Bønes senter
Øvre Kråkenes
Paradis
Hop

Abstraksjon i databehandling

Programmeringsspråk er en abstraksjon.

Programmeringsdesign innebærer bruk av abstraksjon.

Mann, kål, geit, ulv problem

Kilde: Introduction to Computer Science Using Python: A Computational Problem-Solving Focus

Mann, kål, geit, ulv problem

En mann bor på østsiden av en elv. Han ønsker å bringe en kål, en geit og en ulv til en landsby på vestsiden av elven for å selge. Båten hans er imidlertid bare stor nok til å holde seg selv, og enten kålen, geiten eller ulven.

I tillegg kan mannen ikke la geiten være alene med kålen fordi geiten vil spise kålen, og han kan ikke la ulven være alene med geiten fordi ulven vil spise geiten.

Hvordan løser mannen problemet?

Representasjon (1/3)

Hva vil være en passende representasjon for dette problemet?

Skal vi inkludere i representasjonen …

farge til båten?
navnet til mannen?
bredden til elva?

Den eneste informasjonen som er relevant for dette problemet er hvor hvert enkelt element er på hvert steg i problemløsningen. Derfor, ved bruk av **abstraksjon**, definerer vi en representasjon som bare fanger opp denne nødvendige informasjonen.

Representasjon (2/3)

For eksempel kan vi bruke en sekvens for å indikere hvor hvert av objektene befinner seg :

mann	kål	geit	ulv	båt	landsby
[ø s t ,	v e s t ,	ø s t ,	v e s t ,	ø s t ,	v e s t]

Det første elementet i sekvensen er plasseringen av mannen, det andre elementet i sekvensen er plasseringen av kålen, osv.

Representasjon (3/3)

- Landsbyen flytter ikke. Det er alltid på vestsiden av elven. Plasseringen av landsbyen er fast og trenger derfor ikke å være representert.
- Båten er alltid på samme sted som mannen. Så å representere plasseringen til både mannen og båten er overflødig informasjon.
- Den relevante, **minimale representasjonen** er gitt nedenfor (erstatter "øst"/"vest" med enkelttegn).

mann kål geit ulv

[Ø , V , Ø , V]

Starttilstand og måltilstand

Det faktiske problemet er å finne ut hvordan mannen kan ro gjenstander over elven, med visse begrensninger på hvilke gjenstandspar ikke kan være sammen alene.

Beregningsproblemet er å finne en måte å konvertere representasjonen av **starttilstanden** til problemet, når alle objektene er på østsiden av elven,

mann kål geit ulv
[Ø , Ø , Ø , Ø]

til **måltilstanden**, når alle gjenstander på vestsiden av elven,

mann kål geit ulv
[V , V , V , V]

med den begrensningen at visse **ugyldige tilstander** aldri skal brukes.

For eksempel, fra starttilstanden er det tre mulige trekk som kan gjøres.
Men bare én av dem resulterer i en gyldig tilstand.

Vi sjekker om den nye problemtilstanden er måltilstanden.

mann kål geit ulv

[Ø , Ø , Ø , Ø] **STARTTILSTAND**

Mannen ror geita over

[V , Ø , V , Ø]

Er det måltilstanden [V , V , V , V] **?** **Nei**

Derfor fortsetter vi å søke fra den nåværende tilstanden.

Siden mannen bare kan ro på tvers av gjenstander på samme side av elven som ham selv, og den eneste gjenstanden med ham for øyeblikket er geiten, **er det bare to mulige trekk herfra**,

mann kål geit ulv

[V , Ø , V , Ø] **MELLOMTILSTAND**

Mannen ror tilbake alene

[Ø , Ø , V , Ø]

GYLDIG TILSTAND

Geit er alene på vestsiden
Mann, kål og ulv er på østsiden

Mann ror geiten over

[Ø , Ø , Ø , Ø]

GYLDIG TILSTAND (men uproduktivt)

Alle er på østsiden. **Det er tilbake til starttilstanden. Derfor er ingen fremgang gjort!**

Dette vil fortsette til måltilstanden er nådd,

Dermed oversettes beregningsproblemet med å generere måltilstanden fra starttilstanden til en løsning av det faktiske problemet siden hver overgang mellom tilstander har en tilsvarende "virkelig" handling - av mannen som ror over elven med (eller uten) en bestemt objekt.

Algoritme

En **algoritme** er

- et begrenset antall klart beskrevne, utvetydige, gjennomførbare steg
- som kan følges systematisk
- for å produsere et ønsket resultat for gitt input
- i en begrenset tidsperiode (det vil si at den til slutt avsluttes).

Algoritmer og datamaskiner: en perfekt match

Algoritmer er sentrale i informatikk. De gir trinnvise beregningsmetoder som hvem som helst kan utføre.

Datamaskiner kan utføre et stort antall instruksjoner veldig raskt og pålitelig uten feil. Derfor er algoritmer og datamaskiner en perfekt match!

Men **beregningen som en datamaskin kan utføre er bare så god som den underliggende algoritmen.**

Programmeringsspråk

Hvordan beskriver vi en algoritme slik at maskinen forstår?

Vi trenger programmeringsspråk!

Språk

Java:

```java
class HelloWorld
{
    public static void main(String args[])
    {
        System.out.println("Hello, World");
    }
}
```

Piet:

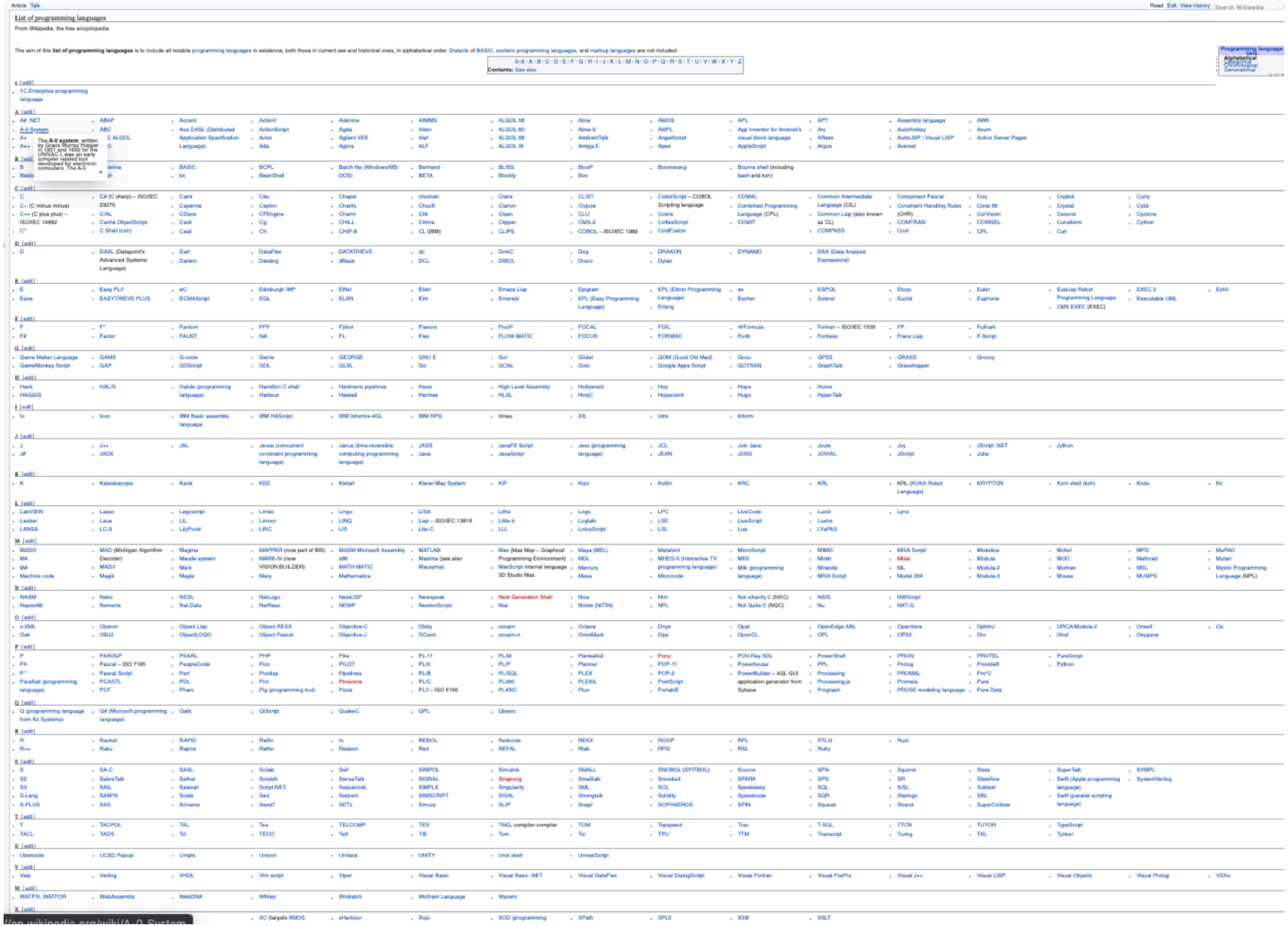
brainf*:**

```
++++++++++[>+++++++>++++++++++>+++<<<-]>++. >+. +++++++
 . .+++. >++. <<+++++++++++++++. >. +++. ------. --------. >+.
```

Python:

```python
print("Hello, World!")
```

https://en.wikipedia.org/wiki/Esoteric_programming_language



https://en.wikipedia.org/wiki/List_of_programming_languages

Syntaks og semantikk

Slikt som “**naturlige språk**” som norsk og engelsk, er syntaks **og** semantikk **viktige begreper som finnes i alle programmeringsspråk**.

Syntaks

- **Syntaksen** til et språk er et **sett med tegn og de akseptable sekvensene (arrangementene)** av disse tegnene.

- Norsk inkluderer for eksempel bokstavene i alfabetet, tegnsetting og riktig stavede ord og riktige setninger. **Følgende er en syntaktisk korrekt setning på norsk**,
  "Hei, hvordan går det med deg?"

- **Det følgende er imidlertid ikke syntaktisk korrekt,**
  "Hei, hwortan går det med deg?"

- Bokstavsekvensen "hwortan" er ikke et ord på norsk.

- `print("hello")` ✓ `print("hello)` ✗

Semantikk

- **Semantikken** til et språk er **betydningen** knyttet til hver syntaktisk korrekt sekvens av tegn.

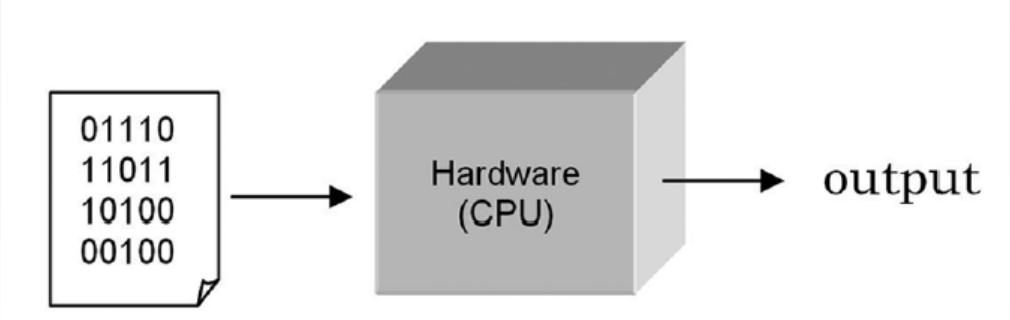
- Tenk på følgende setning:
  "De hyggelige trærne leser fryktløst."

- Denne setningen er syntaktisk korrekt, men har ingen betydning.
  Dermed er det *semantisk feil*.

- `(num1 + num2 + num3) / 3.0` ✓
- `(num1 + num2 + num3) / 2.0` ✗

Programmkjøring

En **central processing unit** (CPU) er **designet for å utføre et spesifikt sett med instruksjoner representert i binær form** (dvs. 1-ere og 0-ere) kalt **maskinkode**. Bare programmer i maskinkode kan kjøres av en CPU.



Compiler og interpreter

Å skrive programmer på dette "lave nivået" er kjedelig og utsatt for feil. Derfor er de fleste programmer skrevet i et "høyt nivå" programmeringsspråk som Python. Siden instruksjonene til slike programmer ikke er i maskinkode som en CPU kan utføre, må et oversetterprogram brukes.

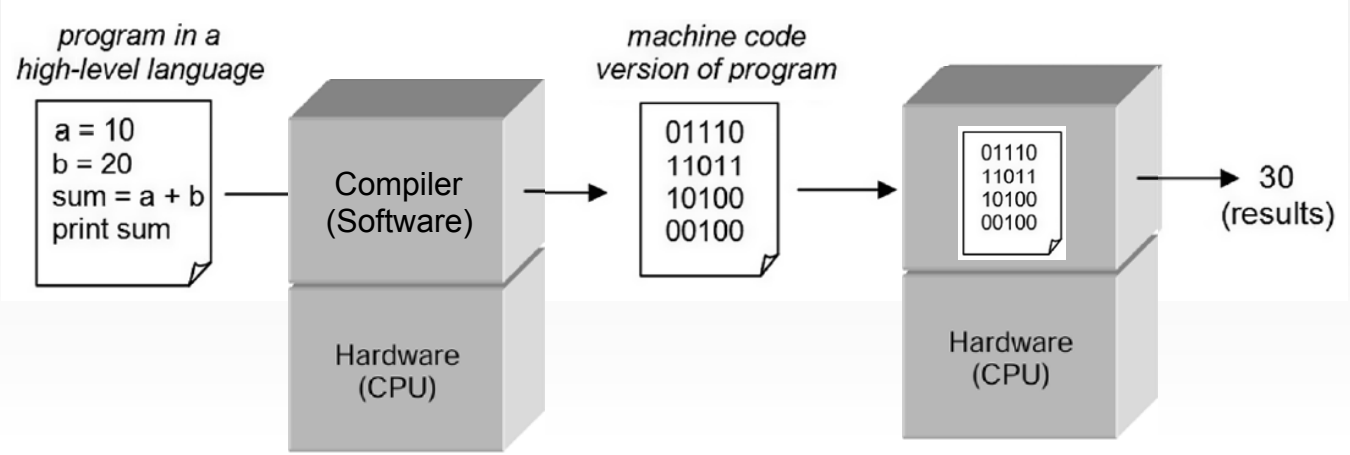
Det er to typer oversetterprogrammer :

- **Compiler** programvare som oversetter programmer til maskinkode som skal kjøres av CPU

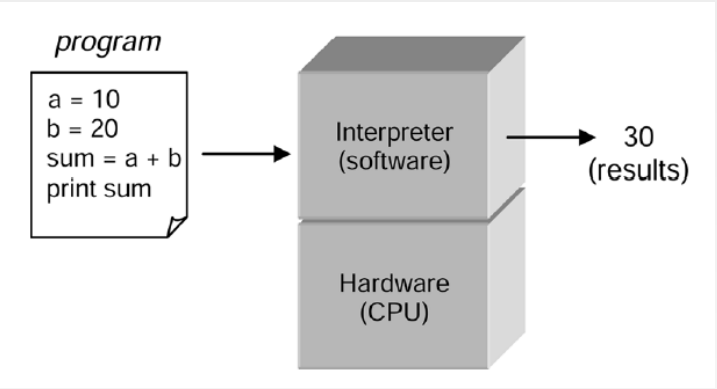
- **Interpreter** programvare som kjører programmer i stedet for CPU

Compiler

Compiler er en programvare som **oversetter programmer** til **maskinkode** som skal kjøres av CPU.



Interpreter



Interpreter er en programvare som **kjører programmer** i stedet for CPU.

En interpreter kan umiddelbart utføre instruksjoner etter hvert som de legges inn. Dette kalles interaktiv modus.

Python, som vi skal se, utføres av en interpreter.

Er programmering vanskelig?

Hvor ligger problemet?

- Verktøy: teksteditor, filhåndtering, kjøring
- Språk: regler, strukturer
- Design:

  løse problemet med en algoritme - gode/dårlige løsninger

  omsetning av algoritmen i programmeringsspråket
- Biblioteker man kan bruke

uib.no