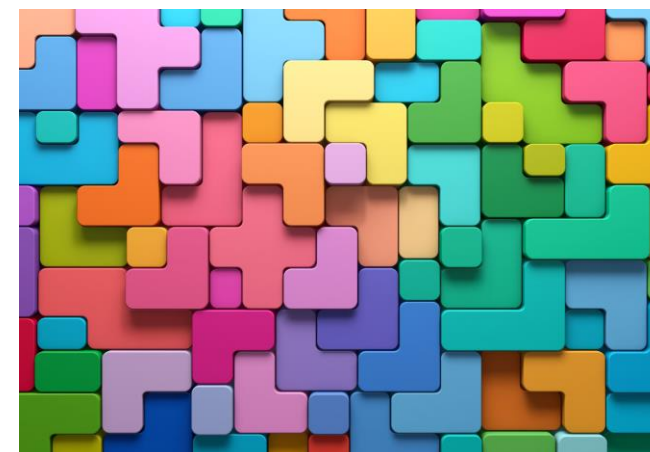


TESTING

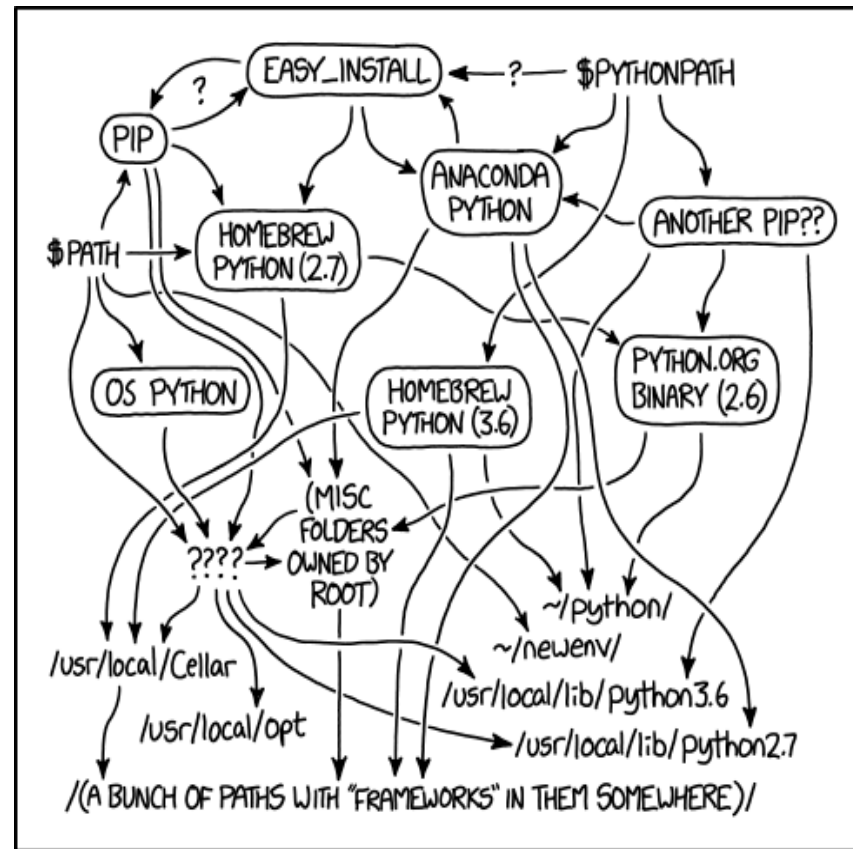
FORELESNING 4

FREDAG 30/8



Vi vet noen har litt terminal-utfordringer

- Spesielt hvis man har en annen python-installasjon fra før i tillegg til Anaconda (macOS har det)
- Det *går an* å bruke Anaconda Prompt, det eneste er at man må navigere til riktig mappe i terminalen (demonstrasjon)
- Det viktigste er at man ikke tester ting med play-knapp i editoren
- Vi svarer på spørsmål på [Mattermost](#), men for raskere eller avansert hjelp, prøv gjerne [IT-hjelp sin chat-tjeneste](#)



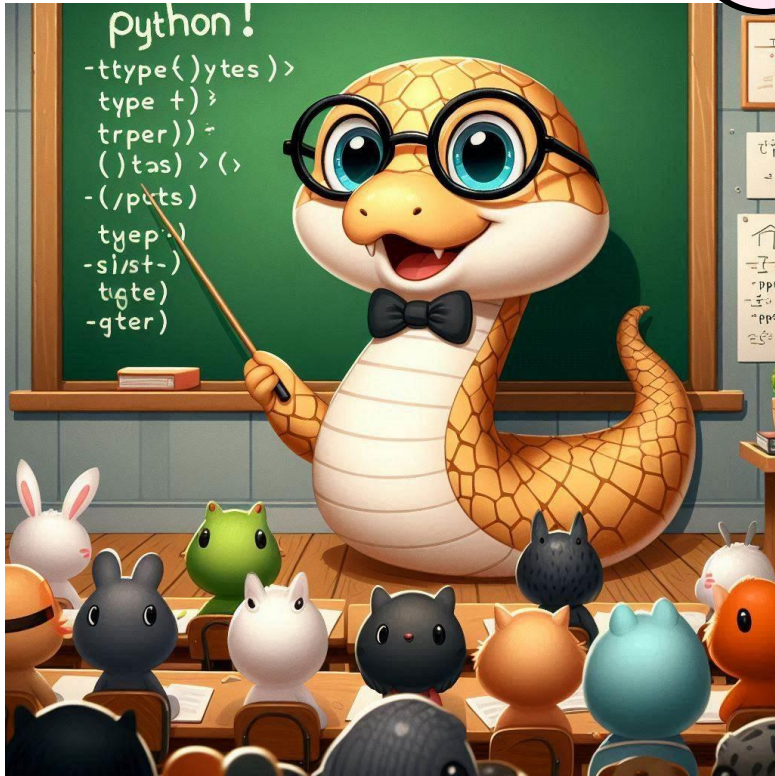
bilde: xkcd.com



○ Tilbakemeldinger

- Vi vil gjerne høre fra dere hvis dere har tanker, tilbakemeldinger eller forslag
- Du kan kontakte emneansvarlig direkte på Mattermost eller be gruppelærere melde videre
- Om det skulle oppstå alvorlige ting, kan du også gå til [studieadministrasjonen](#) eller kontakte et av fagutvalgene ([FUI](#), [Matematisk Fagutvalg](#))

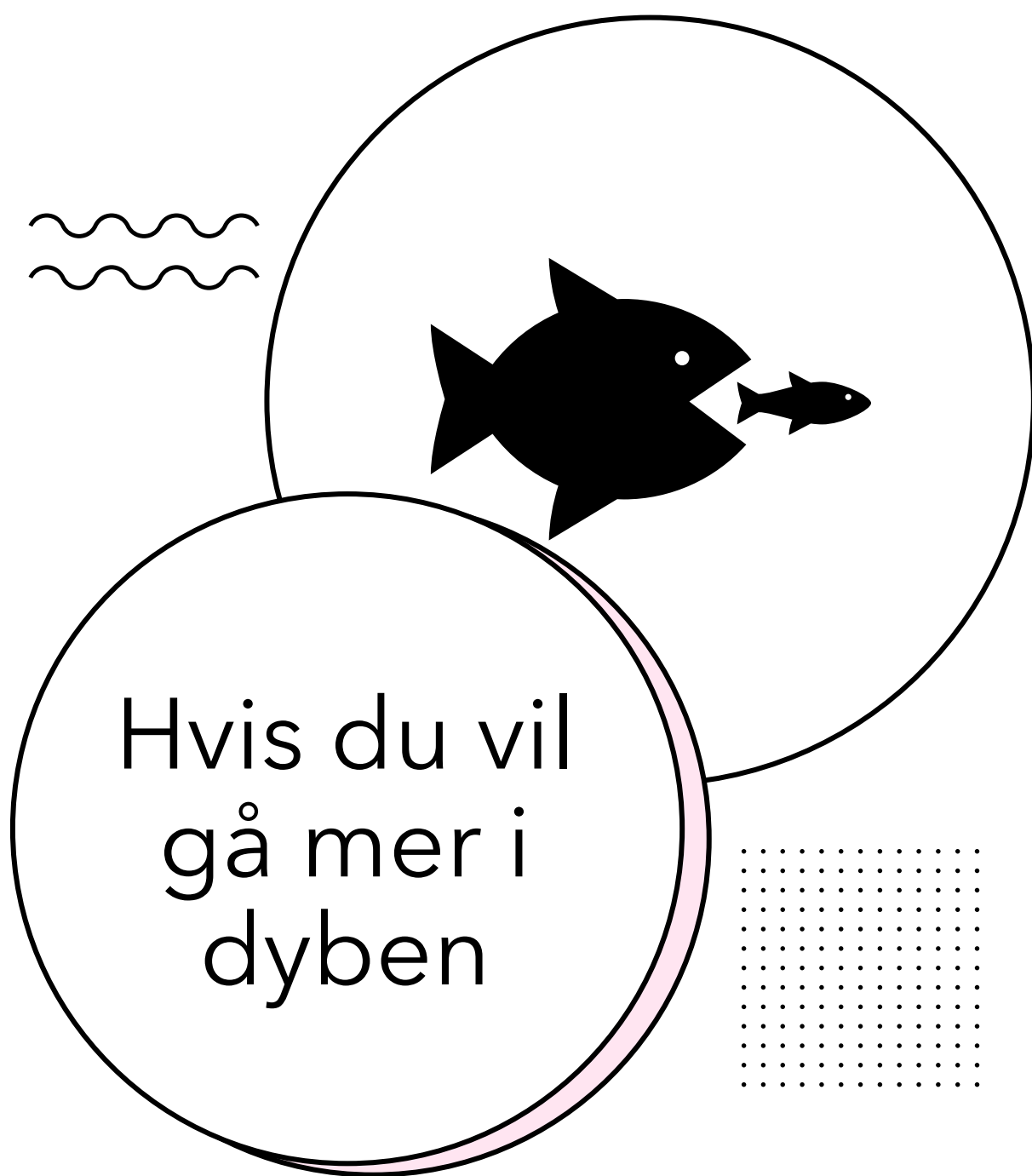




LIVE CODING : TYPE - HINTING

(BILDE: BING IMAGE CREATOR)





- [Litt mer om typer i Python og andre språk](#) (ikke pensum)
- Kan gjøre introduksjonen til C++ lettere senere

○ Læremål: Versjonskontroll og testing

- Bruke GitHub til å holde styr på forskjellige versjoner av programmet (og gå tilbake til noe som fungerte de gangene ting ikke virker eller det bare blir rot)
- Samarbeide uten å ødelegge for hverandre (hver utvikler jobber med sin egen testversjon av moderprogrammet)
- Enhetstesting: automatisk test av at en liten del av et program virker som den skal
- Regresjonstesting: hver gang vi gjør en endring i et program, kjøres tester på nytt for å sikre at ikke en feil har oppstått



○ Motivasjon: Test-drevet utvikling

- Vi ønsker å skrive *pålitelige* programmer som...
- ...er grundig testet før de tas i bruk
- ...heller kræsjer og gir feilmelding enn å gi feil resultat uten feilmelding
- ...kræsjer på en måte som lar oss forstå hva som gikk galt
- ...ikke ødelegger datafiler hvis de kræsjer (kræsjer kontrollert)



○ Noen nyttige verktøy

- assert
- exceptions
- pytest
- test-drevet utvikling (en måte å tenke og jobbe på)
- exit-koder



LIVEKODING: TESTING MED PYTHON



(BILDE: BING IMAGE CREATOR)

DET FINNES MANGE TYPER EXCEPTIONS



Dette er et eksempel på ARV, som vi forklarer grundigere om noen uker!

```
+-- Exception
+-- StandardError
|   +-- ArithmeticError
|   |   +-- FloatingPointError
|   |   +-- OverflowError
|   |   +-- ZeroDivisionError
|   +-- AssertionError
|   +-- AttributeError
|   +-- EnvironmentError
|   |   +-- IOError
|   +-- EOFError
|   +-- ImportError
|   +-- LookupError
|   |   +-- IndexError
|   |   +-- KeyError
|   +-- NameError
|   +-- RuntimeError
|   |   +-- NotImplementedError
|   +-- SyntaxError
|   |   +-- IndentationError
|   |       +-- TabError
|   +-- SystemError
|   +-- TypeError
|   +-- ValueError
```

○ Fordeler med assert

- assert går fort å legge inn i koden
- assert kan skrus av i "optimized mode" (-O)



○ Fordeler med exceptions

- exceptions gir mer detaljert info om hva som gikk galt
- exceptions gir oss mer kontroll over hvordan vi håndterer feilen som har oppstått



○ ...så hva bør jeg bruke når?

- Bruk **assert** for å teste for feil som normalt *aldri* bør skje (så dette kan rettes opp i før brukere kjører programmet)
- Bruk **exceptions** for å teste for feil som kan *forventes* å skje når brukere kjører programmet, så brukere opplever at feilen håndteres fornuftig av programmet
- Du kan tenke på **assert** som et utviklingsverktøy og **exceptions** som en naturlig del av et brukervennlig program



● Enhetstesting (unit tests)

- En "unit" er en liten del av et større program
- En enhetstest tester om denne delen, isolert sett, gjør det den skal
- Test-drevet utvikling handler om å skrive enhetstester for hver del samtidig som vi lager selve delen
- En stor fordel er at disse testene kan kjøres automatisk
- Hvis vi senere endrer noe som gjør at en del som fungerte begynner å gjøre feil, så vil dette automatisk fanges opp!



- Brukere gjør ofte uventede ting...



LIVEKODING: PYTEST



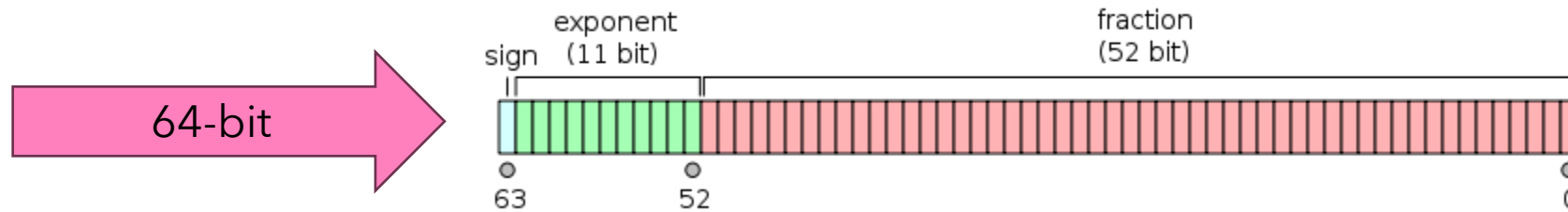
(BILDE: BING IMAGE CREATOR)

○ Avrundingsfeil (float)

- Datamaskinen bruker totalssystemet: 0, 1, 10, 11, 100, 101, ...
- Så da burde et tall som 2.2 gå veldig fint?
- $2.2 = 2\frac{1}{5} =$
 $1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} + 0 \cdot 2^{-5} + \dots$
- 2.2 i totalssystemet blir 10.0011001100110011001100...
- Evig repeterende siffer krever uendelig minne i maskinen
- Det blir en avrunding til slutt, som kan gi *avrundingsfeil*



En **float** i maskinen



<https://ianqvist.blogspot.com/2009/10/floating-and-fixed-point-arithmetic.html>

- S: 1 bit (*binary digit*) til fortegn (+/-)
- E: 11 bits til eksponent
- F: 52 bits til grunntallet
(det er her avrundingsfeil oppstår)
- $x = S \cdot F^E$



○ Etter forelesningen

- Hvis du ikke har gjort det enda – logg inn på github.uio.no (for å få GitHub-bruker) og send melding på [Mattermost](#) om at du er klar for å få repository (repo) til Prosjekt 0
- De første [ukesoppgavene](#) ligger også ute, se gjerne på dem parallelt med prosjektet

