Terminalbruk for nybegynnere

Hvis du kjører Windows: Hopp til "Hvordan sette opp wsl", så gå tilbake til start

Hva dekker dette dokumentet

- Kort om hva terminalen er, og hvorfor du bør kunne noe om den
- Intro til grunnleggende terminalkommandoer
- Kort om pakkehåndtering fra terminalen (apt og brew)
- Hvordan sette opp et python-miljø og kjøre python-filer fra terminalen.
- En oppsummering av hyppig brukte terminalkommandoer

- Hva er WSL
- Hvordan sette opp WSL (Windows subsystem for Linux)
  - Når du har satt opp WSL kan du få bruk for resten av dokumentet

Hva er terminal?

- Terminalen er et slags tekstbasert "vindu" inn i det som skjer under panseret i pc'en din
- Det har ikke noe å si hvordan du åpner terminalen – alt er bare forskjellige måter å vise frem samme "underliggende" terminal. Det betyr at du kan gjøre det samme enten du åpner terminalen i
  - PyCharm
  - VScode
  - "Terminal"
  - Noe annet
- Når du skriver/leser i terminalen kan du interagere med pc'en din gjennom kommandoer og ved å kjøre programmer.
- Alt som kan gjøres med en GUI (grafisk brukergrensesnitt) kan også gjøres fra en terminal.
  - Det som skjer under panseret når du trykker "run" i pycharm eller vscode, er at appen effektivt kjører en terminalkommando for deg.

Hvorfor er terminal?

- Fordi vi kan interagere direkte med pc'en gjennom terminalen, uten noen app mellom oss og kommandoene som kjøres, får vi mye bedre kontroll på hva som egentlig skjer under panseret.
  - Dette gjør det mye lettere for oss å sikre at vi faktisk gjør det vi vil, og ikke det utvikleren av en app tror at vi vil.
- Terminalen gir ofte veldig mye gode tilbakemeldinger, som forteller deg hva du skal gjøre når noe går galt.
  - Det er mye lettere å fikse enn "knappen funker ikke"
- Alle "terminaler" på pc'en din er det samme
  - Du slipper å leke "hvor er, hvor er, hvor er knappen" hvis du bruker en annen app eller en annen versjon enn den du har veileder/dokumentasjon til
- Terminalkommandoer er godt dokumentert
  - Veldig ofte når du har et problem, kan du finne et forum hvor noen gir deg en terminalkommando for å fikse det. Det funker stort sett, og selvfølgelig uavhengig av hvilken IDE eller app du bruker (ref. forrige punkt).

Kort om pakkehåndtering

- Når du tidligere har installert apper, er du nok vant til å gå på en nettside, laste ned en sak, og ……
- Både Linux og macOS har et pakkehåndteringssystem som vi kan bruke for å installere ting direkte fra terminalen
  - macOS bruker *brew*
  - Ubuntu bruker *apt*
  - Andre Linux distroer (mint, popOS, etc.) kan bruke andre
- Alle pakkehåndteringsssytemene fungerer rimelig likt
  - *brew*, *apt*, eller liknende er et program – du sier til programmet hva du vil installere.
  - Programmet har en *indeks* over alle eksisterende pakker.
  - Det finner det du vil ha i indeksen og installerer det.
- Fordelen med et pakkehåndteringssystem er at det gir veldig god kontroll over
  - Hva du har installert
  - Hvilke versjoner du har installert
  - Hvor på pc'en de er installert til
  - Hvilken versjon som er "aktiv"

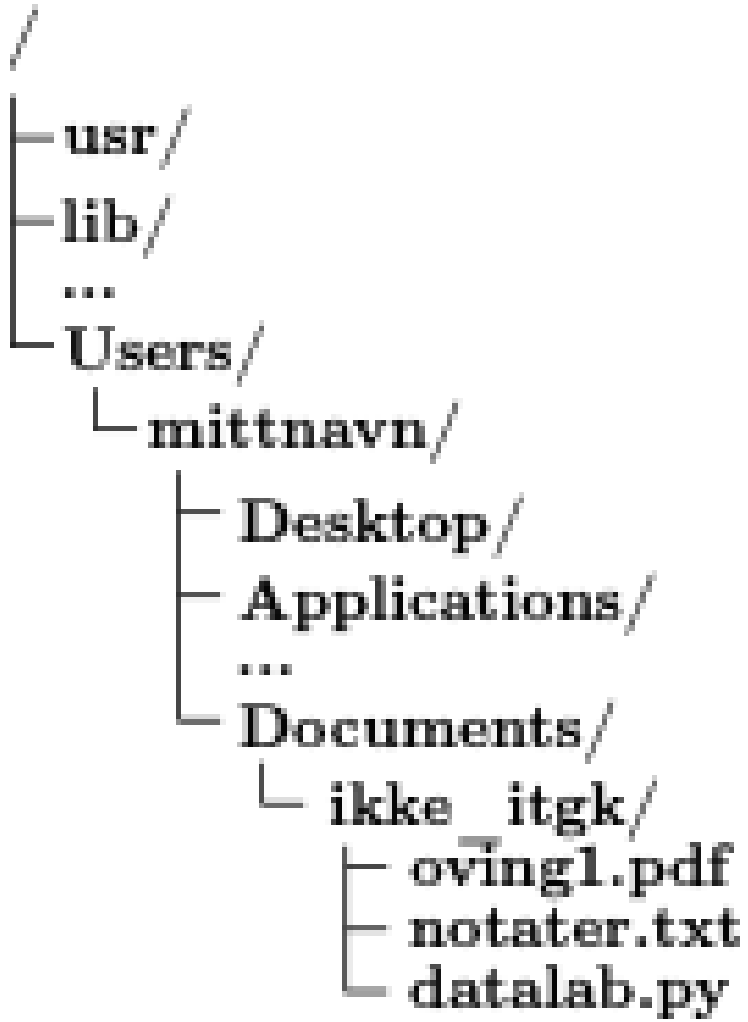
Intro til terminalbruk

Nå skal vi kjøre de første terminalkommandoene våre. For å følge de neste stegene bør du ha en terminal åpen, det har ingenting å si hvordan du åpner den.

Hvis du bruker Windows må du sørger for å sette opp **og aktivere** WSL før du kan følge Linux-stegene.

Filsystemer

- Når vi skal holde på i terminalen gjelder det å ha en oversikt over *filsystemet* til pc'en vi jobber på, enda mer enn når vi jobber i en app.
  - Derfor er det *ekstra* viktig at man vet hvor på pc'en ting er når man jobber fra terminalen.
- Filsystemet på pc'en din er organisert noe som i bildet til høyre.
  - "/" er navnet på "rotmappen" i systemet.
  - Alt som ender med en "/" er en mappe



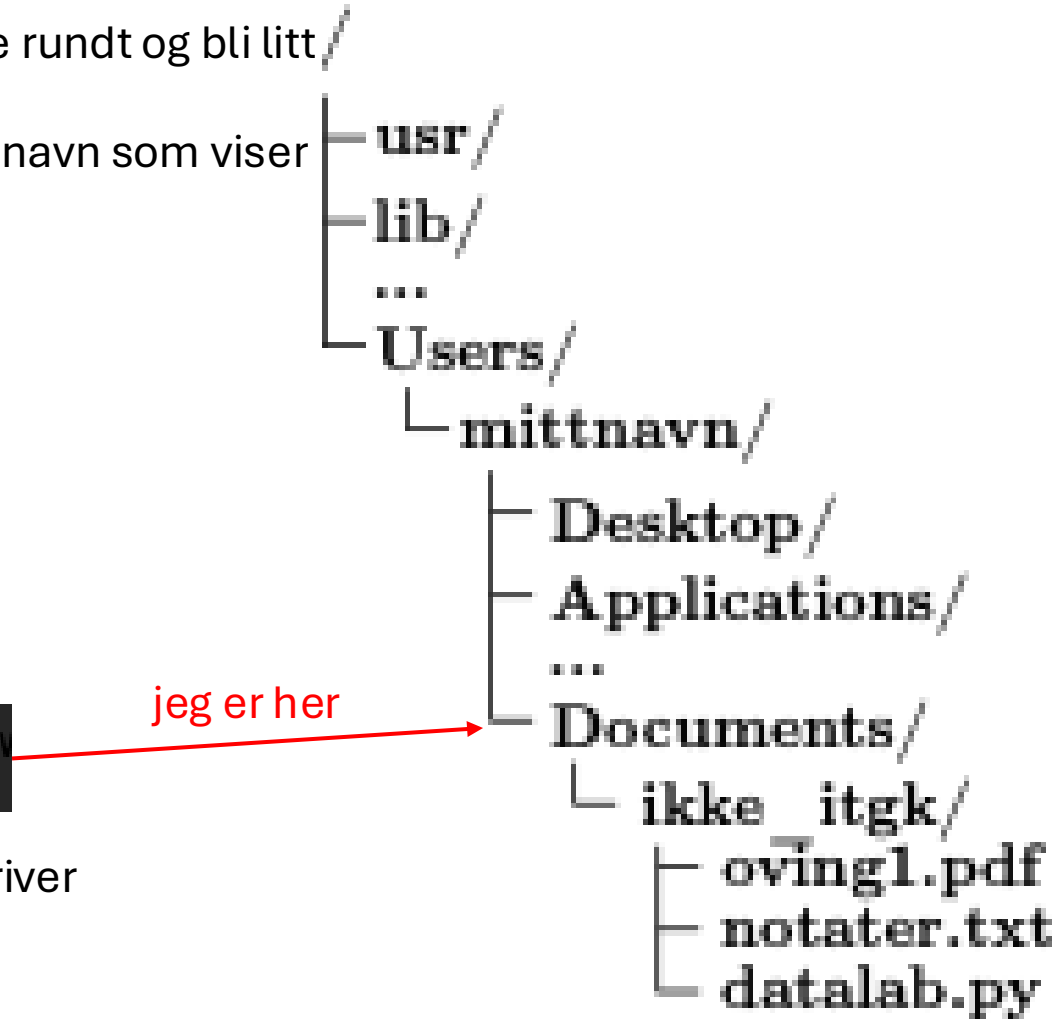
Min første terminalkommando

- Nå som vi har åpnet en terminal skal vi begynne med å vandre rundt og bli litt kjent.
- I terminalprompten ser du til venstre en "path" eller et mappenavn som viser hvor på pc'en "du er"

```
MacBook-Pro-4:Documents vegardjervell$
```

- I dette bildet er jeg inni en mappe som heter "Documents"
- Hvis det står "~" betyr det at du er i "hjem"-mappen din
  - "~" er bare en forkortelse for "hjem"
- Hvis det står "/noe/noe_annet/noe_mer"
  - Er det en "absolutt path" fra "roten" av filsystemet ditt til der du er
- Skriv "echo $PWD" + "enter"
  - Nå skrives hele stien (path'en) til mappen du er i ut i terminalen

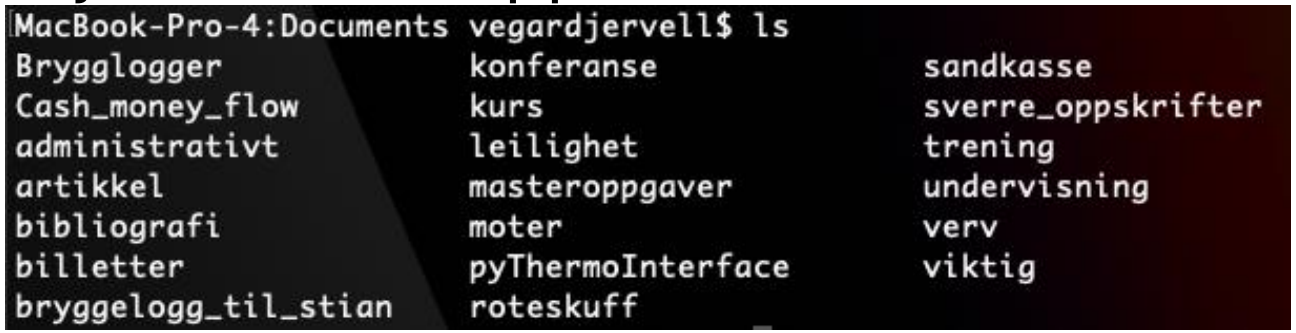
```
MacBook-Pro-4:Documents vegardjervell$ echo $PWD
/Users/vegardjervell/Documents
```



- Alle terminalkommandoer har overordnet samme form. Vi skriver
  - program + argumenter
- I dette eksempelet er
  - "echo" et program som tar imot et input og skriver det til terminalen
  - "PWD" er navnet på variabelen som vet hvor du er.
  - "$" bruker vi for å hente verdien til variabelen.

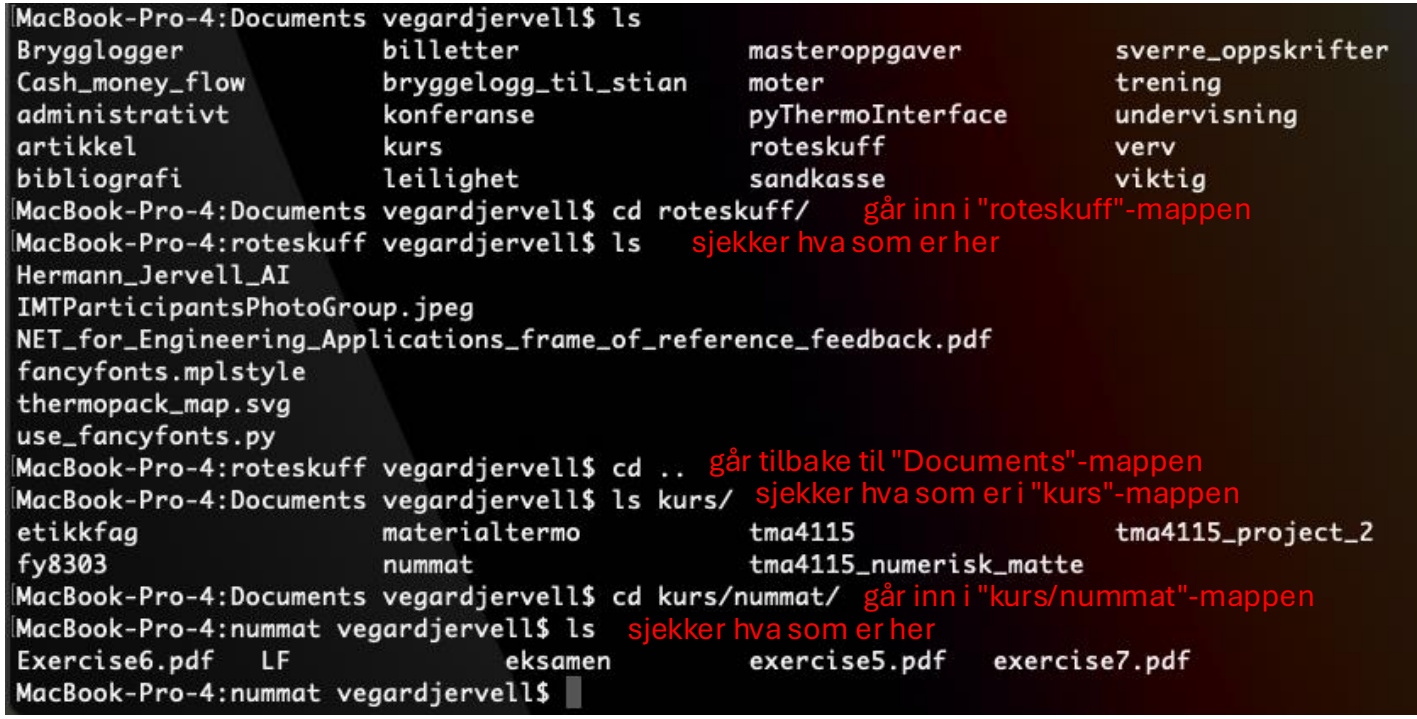
Navigering i terminalen

- Når vi skal flytte oss rundt i terminalen bruker vi primært to kommandoer (programmer)
  - cd sti/til/mappe
    - cd (change directory) flytter deg til en ny mappe
  - ls sti/til/mappe
    - ls (list) ramser opp det som er i en mappe
    - Du kan kjøre "ls" uten noe mappenavn – da ramser den opp det som er i mappen du er i akkurat nå
- Kjør "ls" fra mappen du er i nå for å se hva som er der



Navigering i terminalen (II)

- Bruk "cd" og "ls" for å finne veien til en mappe hvor du har lyst til å opprette et python-prosjekt
  - Tips: Alle mapper inneholder en "spesiell" mappe som heter ".."
  - Dette er "mappen" som peker på mappen over der du er nå.
  - **Tips**: "tab" for å autocomplete
- Til høyre ser du hvordan min terminal ser ut når jeg gjør dette



Dokumentasjonssider (manualer)

- De fleste terminalkommandoer har en brukermanual som følger med.
- For å lese en manual bruker vi "man" kommandoen
  - For eksempel "man cd"
- Kjør "man <kommando>" for å se manualsiden til en valgfri kommando
- Trykk "q" for å lukke siden.

Installere noen pakker

- **Nå skal vi installere noen pakker på systemnivå.**
- **Det har ingenting å si hvor i filsystemet du befinner deg når du kjører disse kommandoene**

Linux

- sudo apt update
  - Oppdatterer indeksen
- sudo apt upgrade
  - Oppdatterer evt. pakker som trenger å oppdatteres
- sudo apt install python3-pip
  - Installerer python og pip
- sudo apt install python3-venv
  - Installerer "python venv", som vi trenger senere

macOS

- xcode-select --install
  - Installerer "Developer command line tools" (som du trenger)
- /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
  - Installerer homebrew (som av de fleste er regnet for å være bedre enn innebygde macPorts)
- brew install python
  - Installerer python, pip, venv, etc.

En til nyttig kommando

- kjør "which python"
  - Programmet "which" tar imot et annet program som argument, og forteller deg hvor det ligger.
- Hvis "which python" ikke returnerer noe, betyr det at det ikke finnes noe som heter "python" på pc'en din.
  - Kjør "which python3"
- kjør "python --version" (eller "python3 --version")
  - Praktisk talt alle programmer kan ta imot "--version" argumentet

```
MacBook-Pro-4:nummat vegardjervell$ which python
MacBook-Pro-4:nummat vegardjervell$ which python3
/opt/homebrew/bin/python3
MacBook-Pro-4:nummat vegardjervell$ python3 --version
Python 3.12.4
MacBook-Pro-4:nummat vegardjervell$
```

returnerer ingenting – "python" finnes ikke

prøver "python3"

Her er python! (Se på filsystemkartet et par sider siden hvis du ikke skjønner)

Opprette et python-prosjekt

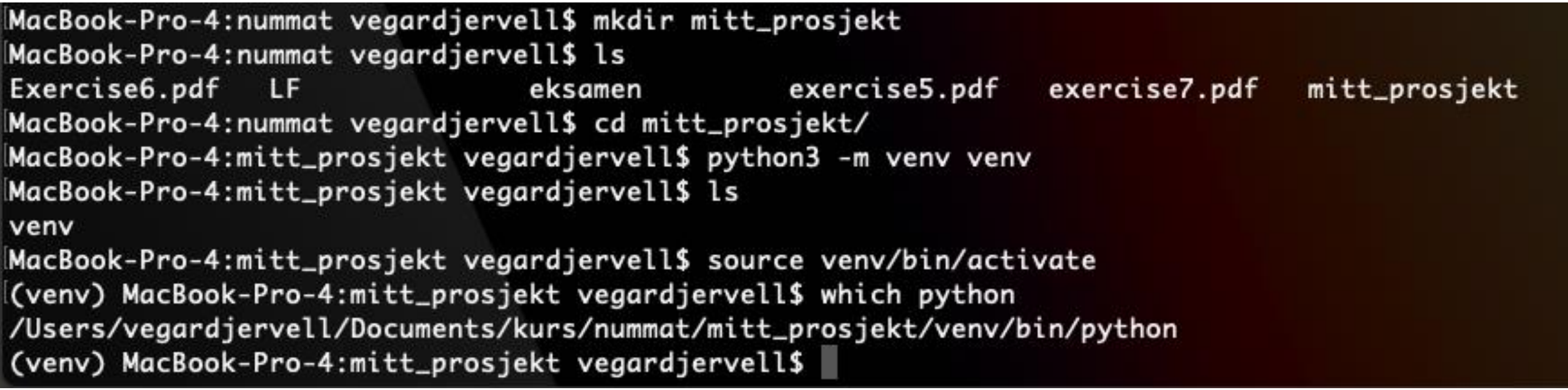
Nå som vi har installert alt vi trenger kan vi opprette et python-prosjekt

Merk: Du kommer ikke til å trenge å installere alt mulig igjen, med mindre du vil oppdattere noe eller får deg en ny pc.

Opprette et python-prosjekt

- Navigér til en mappe hvor du har lyst til å opprette et python-prosjekt (jeg valgte "/Users/vegardjervell/Documents/kurs/nummat")
- Kjør "mkdir mitt_prosjekt"
  - "mkdir" er programmet som oppretter mapper for oss
- Bruk "ls" og "cd" for å sjekke at mappen er opprettet, og gå inn i den.
- Kjør "python3 -m venv venv"
  - Dette oppretter et virtuelt miljø ("python3 –m venv") og putter det i mappen "venv"
  - "ls" etterpå for å sjekke at "venv"-mappen er opprettet
- Kjør "source venv/bin/activate"
  - "source" er et program som leser en fil, og endrer terminalinstillingene våre.
  - filen "activate" (som ligger i "venv/bin") har python generert for oss i forrige steg
  - Hvis du starter en ny terminal kommer du til å få tilbake standardinstillingene
- Kjør "which python"
  - Nå kommer du til å se at "python" peker til en installasjon som ligger i "venv/bin"
  - Det er fordi "activate"-filen har endret terminalinnstillingene til å bruke den python-versjonen som har blitt lagt i "venv/bin" da vi opprettet det virtuelle miljøet.

Min terminal ser sånn her ut etter forrige side



- Legg merke til at det står "(venv)" til venstre i terminalen når jeg har aktivert det virtuelle miljøet.
- Merk også: Hvis du starter en ny terminal kommer den til å åpne seg med standardinstillingene igjen. Hvis du skal kjøre python-filer eller installere python-pakker fra terminalen må du aktivere det virtuelle miljøet på nytt
  - Steget hvor vi kjørte "source sti/til/ven/bin/activate"

Nå kan vi begynne å skrive kode!

- Tips: For å se mappen "mitt_prosjekt" i Finder (macOS)
  - Kjør "open ." ("open" er programmet som åpner ting for oss, "." er det spesielle navnet på mappen vi er i akkurat nå, sånn som ".." er mappen over.)
  - På Linux må du kanskje kjøre "apt install open" eller noe sånn først
  - På Windows tror jeg ikke du kan bruke "open", selv med WSL
- Åpne mappen "mitt_prosjekt" i din favoritt-IDE eller tekstbehandlingsprogram.
- Opprett filen "min_fil.py"
  - Dette kan gjøres fra terminalen med "touch min_fil.py", men hvordan du gjør det er irrelevant
- Skriv et lite "hei-hei" program som printer noe, og lagre filen.
  - Du kan skrive i pycharm, vscode, Notepad, TextEdit eller hva som helst annet. Det har ingenting å si.
  - Hvis du er skikkelig interessert:
    - Kjør "vim min_fil.py" (Åpner filen i editoren "vim", søk det opp)
    - Trykk "i" og skriv kode. (redigerer filen)
    - Trykk "esc" + ":wq" + "enter" (lagrer og lukker filen) (søk på "how to use vim" for å lære mer)

Nå skal vi kjøre koden vår!

- Gå tilbake til terminalen din, og bruk "ls" for å verifisere at "min_fil.py" er lagret.
- Kjør "python min_fil.py"
  - Outputet fra filen din kommer nå til å skrives til terminalen
  - Når du trykker på "run"-knappen i pycharm, vscode, eller liknende er det eneste som skjer at programmet kjører "python <filnavn>" i terminalen for deg.
  - Når du har problemer med at "run"-knappen ikke fungerer er det stort sett fordi programmet du bruker ikke bruker riktig python-installasjon (ikke har aktivert det riktige virtuelle miljøet) eller liknende.

La oss installere noen pakker

- Gå tilbake til terminalvinduet ditt
- Kjør "which python" for å verifisere at du har riktig python-installasjon aktivert (den som ligger i det virtuelle miljøet)
- Kjør "pip install <pakke>" for å installere noen pakker du vil bruke.
- Skriv litt kode i "min_fil.py" som bruker pakken.
- Kjør filen (enten ved å sette opp IDE'en din til å bruke riktig python-installasjon, eller ved å kjøre "python min_fil.py".
- **Tips**: Ved å trykke "pil opp" når du er i terminalen kan du spole opp gjennom tidligere kommandoer så du slipper å skrive så mye.
- **Tips**: Bruk "tab" for å få autocomplete i terminalen.

Gratulerer!

Nå har du lært deg å navigere i terminalen din, opprette et python-prosjekt, sørge for at et virtuelt miljø er aktivert, og å ha kontroll på hvilken python-installasjon som brukes for å kjøre python-filene dine. Det er ikke rent lite!

På neste side følger et lite oppslagsark over terminal-kommandoene vi har brukt

Terminalkommandoer

- cd <mappe>
  - ".." er navnet på mappen "ett nivå opp"
- ls
  - ls <mappe>
- which <program>
- man <program>

- python –m venv venv
- source venv/bin/activate
- mkdir <mappenavn>
- touch <filnavn>
- python <filnavn>
- brew/apt install <pakke>
- sudo <kommando>
- "pil opp"
- "tab"

- Flytt (change directory) til <mappe>

- Rams opp innholdet i mappen du er i
  - Rams opp innholdet i <mappe>
- Si hvor <program> ligger på pc'en
- Vis manualen til <program>
  - "q" for å lukke manualen
- Opprett et virtuelt miljø i mappen "venv"
- Aktivere et virtuelt miljø
- Opprett mappen <mappenavn>
- Opprett filen <filnavn>
- Kjør <filnavn> med python
- Installer <pakke> med brew/apt
- Kjør <kommando> som superbruker
- Viser forrige kommando (kan gjentas)
- autocomplete

WSL – Windows subsystem for Linux

Hvis du bruker Windows, og ikke føler deg helt klar for å avinstallere det og bytte til Linux på heltid kommer du nok til å få bruk for WSL på et tidspunkt

Det som følger er en kort intro til hvordan sette opp WSL på windows, samt en liten beskrivelse av hva det er for noe

Hva er WSL?

- WSL er en slags emulator av noe slag, som lar deg bruke en minimal linux-installasjon i terminalen på Windows.
- Hvis du vil ha en mer fullstendig Linux installasjon (som du også kan bruke uten terminalen) kan du søke på "Linux virtualbox windows" eller noe sånn.
- WSL lar deg bruke Linux-programmer, og gir deg tilgang til en Linux-terminal på Windows-maskinen din.

Sette opp WSL

- Åpne et terminalvindu
- Kjør "wsl --install"
- Kjør "wsl"
- Hvis overnevnte ikke fungerte
  - Åpne "systeminstillinger" => "aktiver windows-funksjoner"
    - Skroll nederst i listen over ting
    - Huk av "Windows subsystem for Linux"
    - Restart pc'en din
- Hvis dette ikke fungerte
  - https://duckduckgo.com/?q=activate+wsl+windows
- Når det fungerer skal terminalen din få noen fine farger.
- Husk at du må kjøre "wsl" kommandoen på nytt når du åpner en ny terminal
- Nå kan du følge instruksene for Linux i resten av dokumentet.