Utvikle en nordlys-app

Bygg en applikasjon som kan varsle om nordlysaktivitet

– en utforskende læringsreise i verdensrommet

LÆRERVEILEDNING OG ELEVARK

Kort om aktiviteten

Det skal ikke mange linjer kode til for å lage nyttige applikasjoner og programmer.

Vi skal se på hvordan vi kan lage et program (apper) som henter satellittdata og ta beslutning på om det blir nordlys eller ikke. Vi skal se på tre ulike grensesnitt, måter å lage applikasjoner på. Først et rent tekstbasert program, deretter to ulike grafiske grensesnitt: et vindusbasert og et nettbasert.

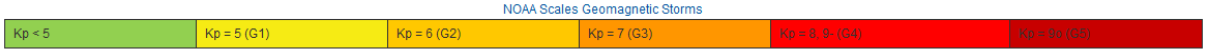
Settet med aktiviteter kan brukes som en samkodingsøvelse, der du som lærer kan kode dette sammen med elevene. Aktivitetene kan også være en veiledning der elevene skriver inn bit for bit selv.

Innhold

Teoridel

K-indeksen som nordlysvarsel

Den planetariske K-Indeksen, også kjent som Kp, er en vitenskapelig måling som brukes for å vurdere nivået av geomagnetisk aktivitet på jorden. Den hjelper forskere og eksperter på romvær med å overvåke hvor stormfullt eller aktivt jordens magnetfelt er til enhver tid. Et forvarsel kan også estimeres ved hjelp av satellittdata.

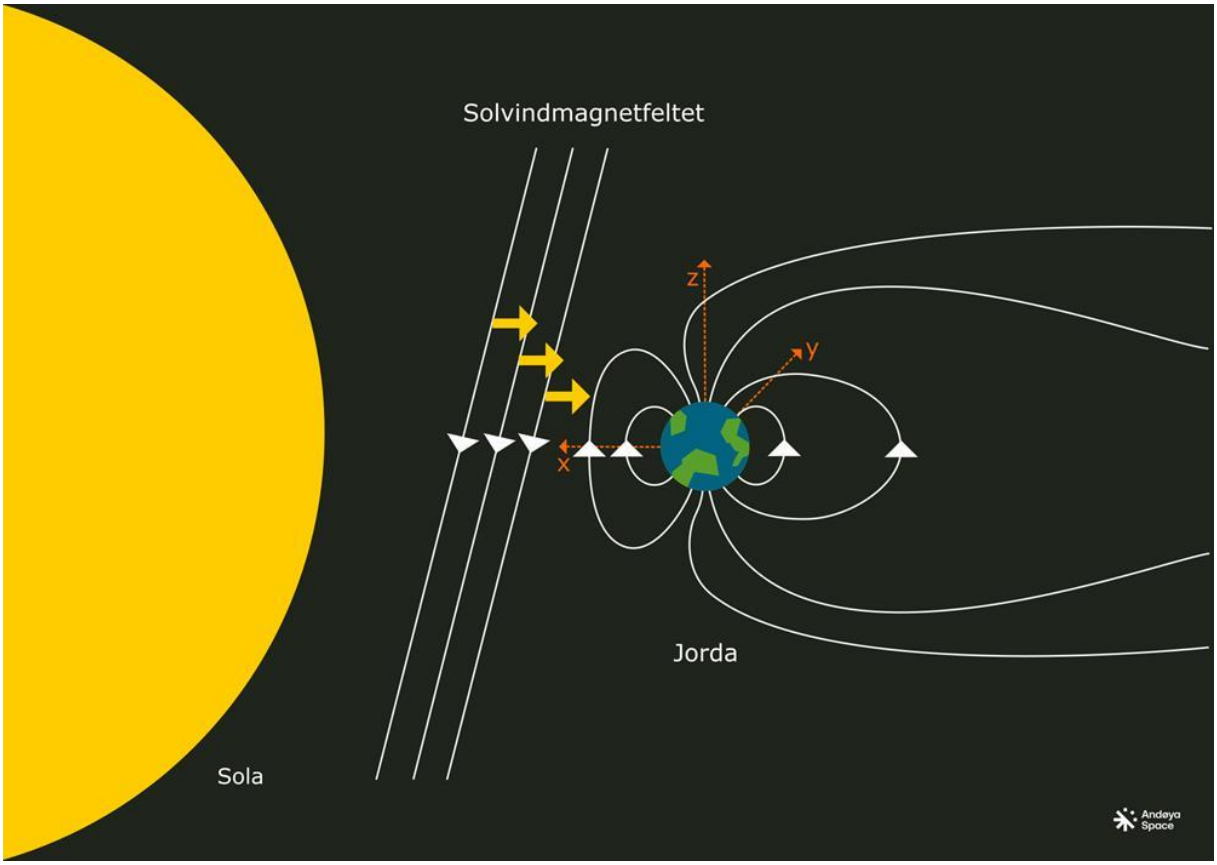


NOAA, et amerikansk direktorat for studier og overvåkning av hav og atmosfære har en side med mer informasjon om den planetariske K-Indeksen. Du kan finne mer informasjon på denne siden: https://www.swpc.noaa.gov/products/planetary-k-index

Satellittdata og solvinden

ACE-satellitten (Advanced Composition Explorer) er en amerikansk satellitt som har som hovedformål å måle solvinden. ACE er plassert i bane rundt L1-punktet (Lagrangepunkt) mellom jorda og sola. Her måler den diverse parametere i solvinden.

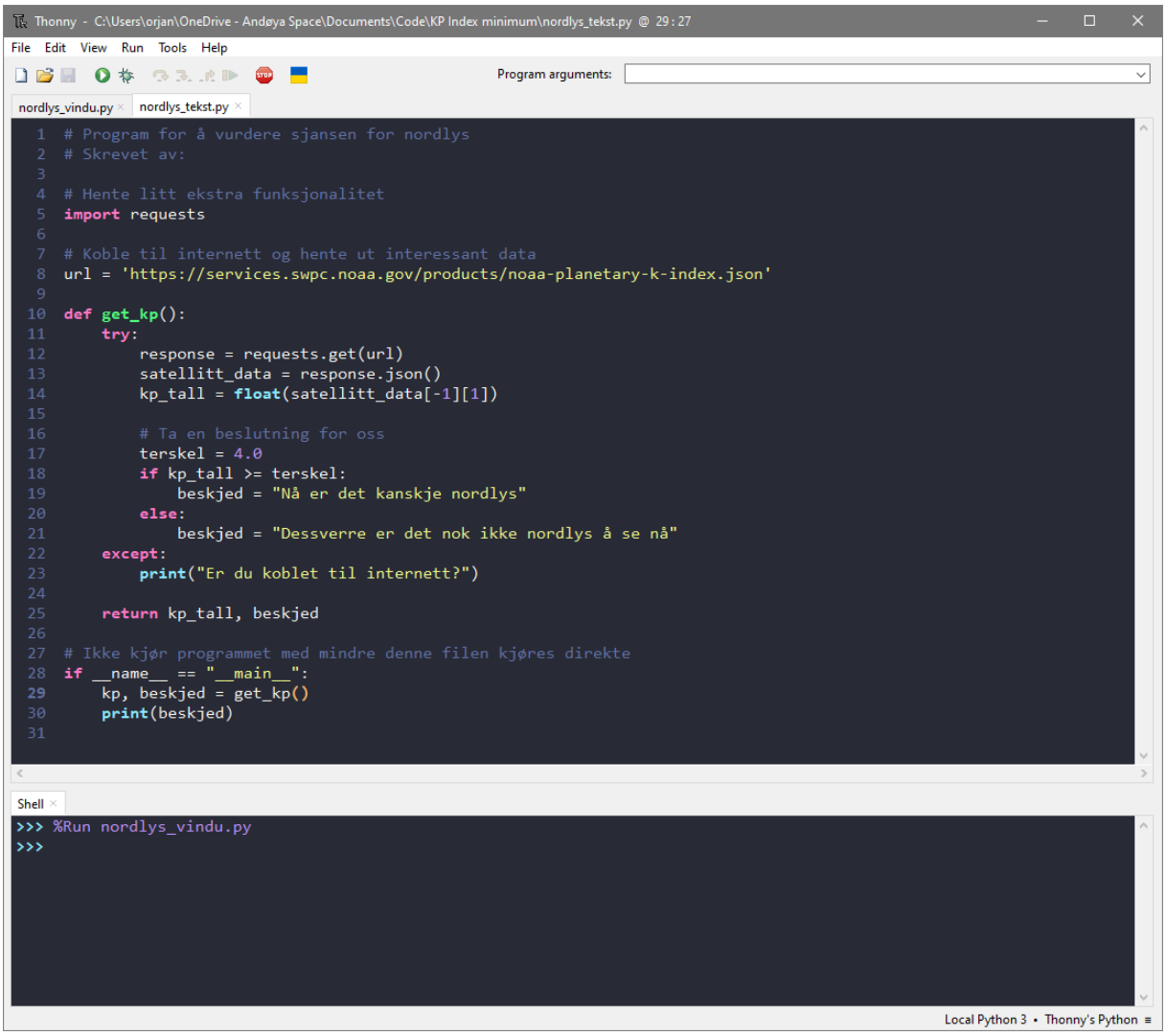
L1 ligger ca. 1.5 mill. km fra jorden, hvor tyngdekreftene fra jorden og solen oppveier hverandre slik at et romfartøy kan holde seg her med minimal bruk av styredrivstoff.



Figur 1: Solvindmagnetfeltet, illustrasjon fa Andøya Space Education

Kom i gang med kodeverktøyet Thonny

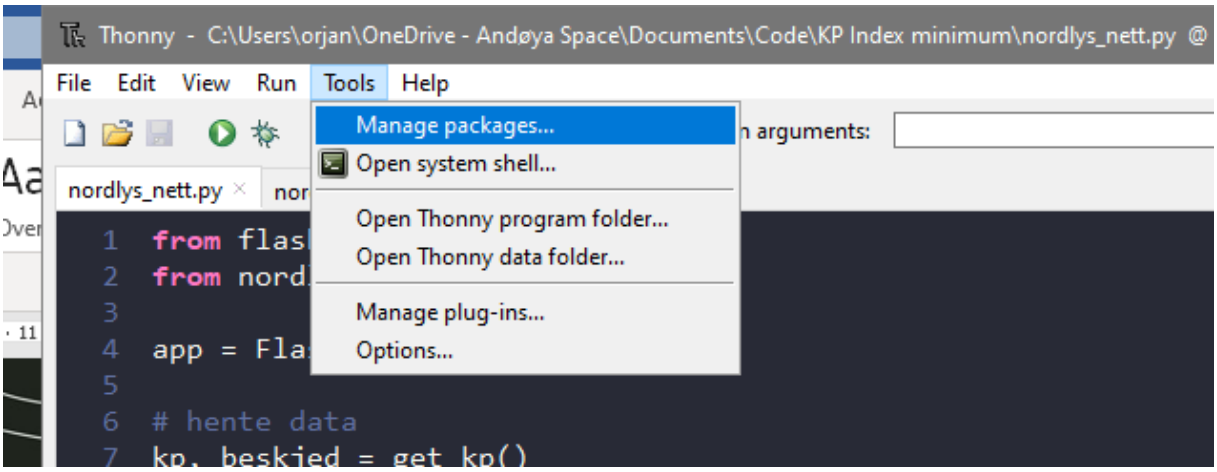
Thonny er et verktøy for å programmere i Python. Det har Python innebygd og kan lastes ned og installeres helt gratis og raskt for Linux, Windows og Mac fra https://thonny.org.



Figur 2: Skjermbilde av kodeverktøyet Thonne, tatt av Andøya Space Education

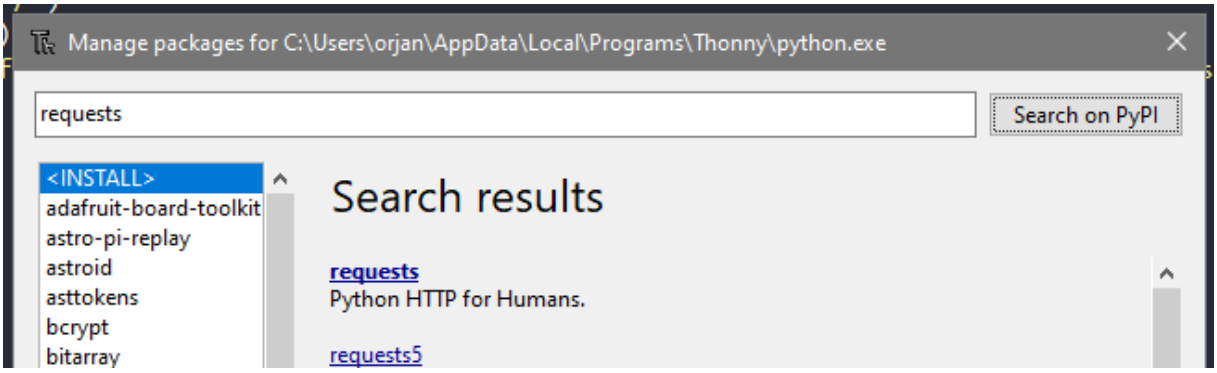
Legge til moduler i Thonny

Python har en del innebygd funksjonalitet som kan tas i bruk direkte uten å måtte importere noe utenfra. I vårt tilfelle har vi behov for å koble til internett og sette opp en webserver. Dette kunne vært gjort uten å hente inn ekstra funksjonalitet, men det finnes noen gode pakker som gjør dette fint og lett forståelig. Slike pakker er laget for at slikt arbeid skal gå lettere og da er det godt å kunne bruke de. Requests og Flask er de to pakkene vi skal importere i denne øvelsen. For å inkludere disse i Thonny, åpner vi Tools → Manage packages:



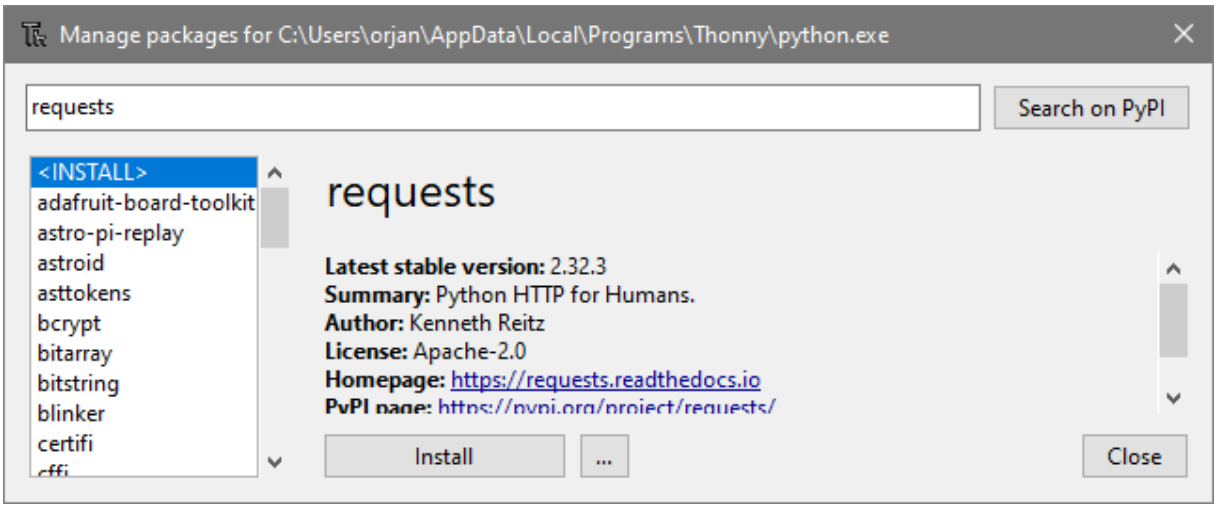
Figur 3: Skjermbilde av Thonny, tatt av Andøya Space Education

Søk opp *requests* (Python HTTP for Humans) først og trykk på navnet i resultatet:



Figur 4: Skjermbilde av Thonny, tatt av Andøya Space Education

Start installasjonen (Install) og gjør en omstart av Thonny etterpå.

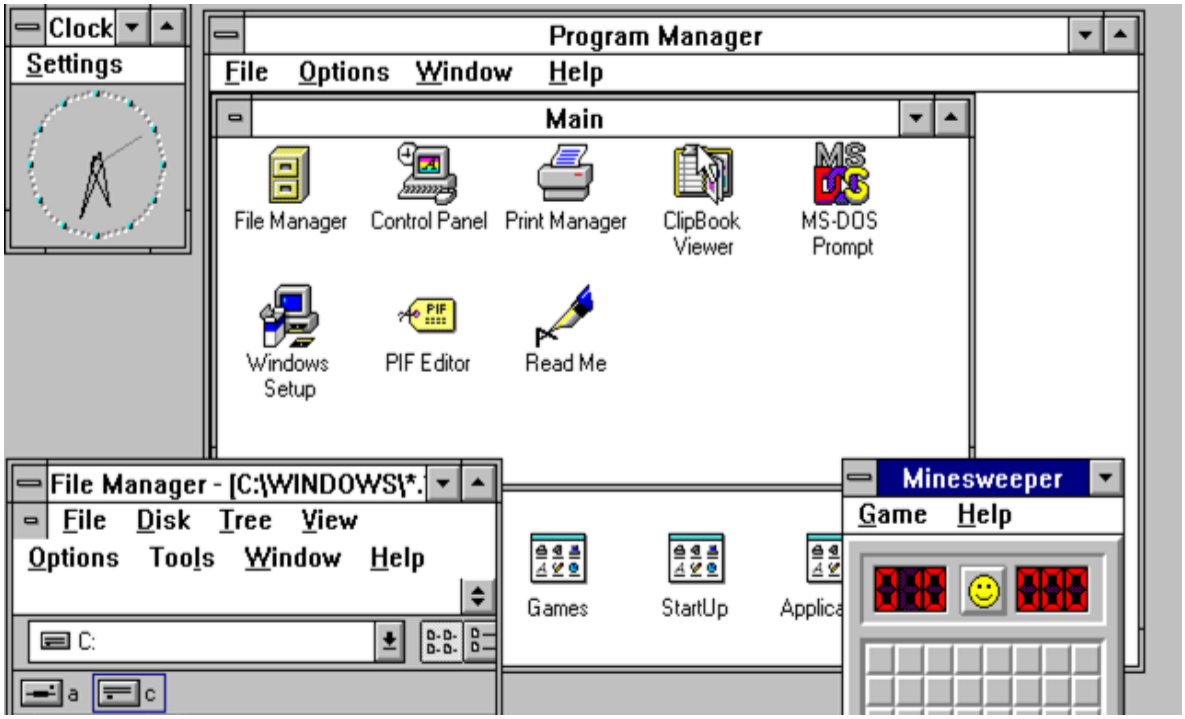


Figur 5: Skjermbilde av Thonny, tatt av Andøya Space Education

Ulike grensesnitt for dataprogrammer

Opp gjennom historien har vi brukt teknologi på ulike måter. De første datamaskinene hadde ikke skjermer slik vi er vant med. Tenk bare på telefonen, de første telefonene hadde en stor synlig høyttaler og en stor synlig mikrofon. Du holdt røret inntil øret og snakket inn i en annen del. Når du skulle ringe noen, måtte du trykke på fysiske knapper eller vri på et hjul for å sette opp samtalen. Knappene og røret var grensesnittet på telefonen den gangen. I dag har vi smarttelefon hvor nesten ingenting er fysiske knapper lengre, bare volum på siden er det.

Datamaskiner fikk etter hvert en skjerm som kunne vise oss informasjon og et tastatur ble vår måte å gi den instruksjoner på. I dag brukes både rene tekstgrensesnitt og vindusbaserte grensesnitt. De har begge sine styrker. For et tegneprogram som Photoshop er det fint å kunne jobbe i et vindusbaserte grensesnitt. Over nett og til rene bestemte oppgaver er derimot tekstbaserte grensesnitt raskere og mer effektivt.



Figur 6: Skjermbilde av Windows 3:11, tatt av Andøya Space Education

Vi skal se på begge disse grensesnittene, flyten av informasjon mellom oss og datamaskinen. Vi skal dessuten se på en annen grafisk metode, nemlig nettbaserte grensesnitt. Det som skjer inne i en nettleser, nettsider eller nettapplikasjoner.

Aktivitet 1: Bygge en terminalapplikasjon

Mellom jorden og solen, i et låst punkt i rommet, er det en satellitt som følger med på solvinder som kommer mot oss. All den informasjonen som denne satellitten samler inn, kan vi få tilgang til. Det er denne informasjonen vi skal benytte oss av her.

NOAA, et amerikansk direktorat for studier og overvåkning av hav og atmosfære mellomlagrer denne dataen i ulike formater. De har valgt å gjøre den nyttige informasjonen på en måte som kalles JSON, en måte å organisere data på som er lett å lese og bruke i programmer.

Satellittdata som vi er ute etter er tilgjengelig på følgende internettadresse:

https://services.swpc.noaa.gov/products/noaa-planetary-k-index.json

som hvem som helst kan se på, helt åpent og fritt tilgjengelig. Vi skal nå bruke Python til å lese denne datakilden. I Thonny, åpne et nytt og blankt dokument og lagre det som `nordlys-tekst.py`

Lag en ny variabel url og gi den adressen som verdi.

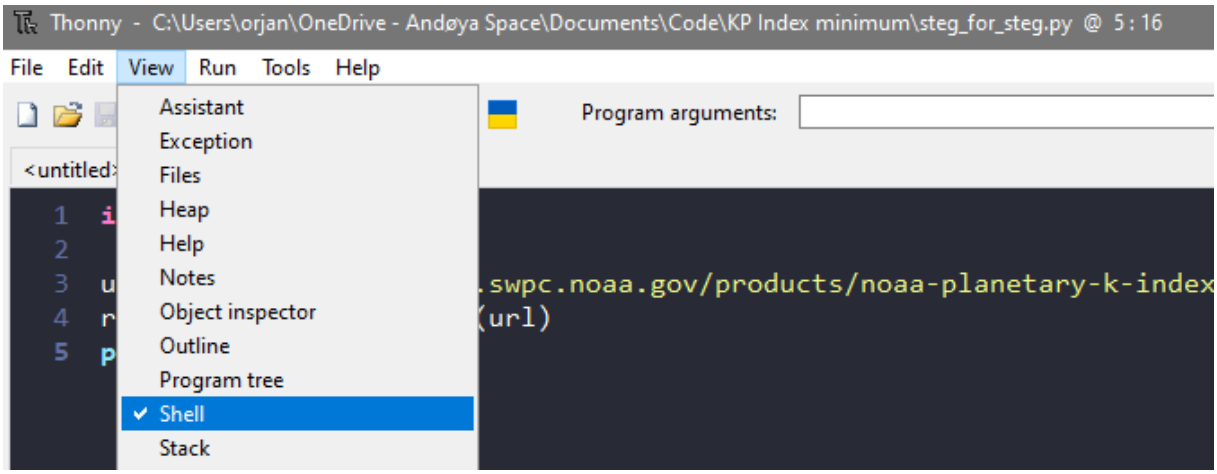
```
url = ‘https://services.swpc.noaa.gov/products/noaa-planetary-k-index.json’
```

Hvis du kjører programmet nå, vil det ikke skje så veldig mye spennende. Vi må be programmet vårt om å koble seg opp til internett for oss, og se om det får det til. For at Python skal kunne gå på internett for oss, må vi hente inn litt ekstra funksjonalitet. Tenk på det som å gå på et bibliotek og finne en bok om et emne som vi har lyst til å lære mer om. Requests er den «boka» vi skal låne nå. (Se hvordan dette kan hentes inn i Thonny i teoridelen.) Legg til en `import requests` øverst i koden din og en linje nederst `response = requests.get(url)` som setter opp tilkoblingen.

For å se om Python får til å koble seg til datakilden, legger vi til en linje `print(response)`.

```
import requests
url = 'https://services.swpc.noaa.gov/products/noaa-planetary-k-index.json'
response = requests.get(url) print(response)
```

Nå kan du kjøre programmet ditt. Pass på at du har aktivert Shell i Thonny:



Figur 7: Skjermbilde av Thonny, tatt av Andøya Space Education

Dersom du får teksten `<Response [200]>` i Shell er alt bra. Du har kanskje vært på internett med en nettleser og fått opp en side der det står 404? Det betyr at den siden du forsøker å åpne ikke er tilgjengelig. En nettleser pleier ikke å fortelle deg at den siden du åpner faktisk er der, det holder at den bare viser deg siden, så skjønner du det. Men nettleseren får egentlig en tilbakemelding fra server likevel, og dersom alt er i orden får den koden 200.

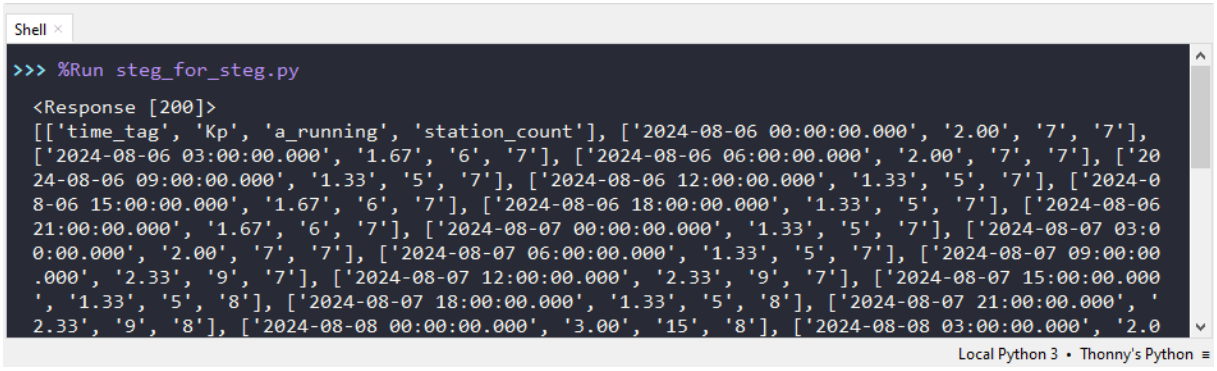
Så når Python nå skriver 200 er det et godt tegn. Får du 404, kan det bety at du enten har skrevet noe feil eller at maskinen din ikke er tilkoblet internett.

Videre ønsker vi å se selve innholdet, ikke bare en bekreftelse på at vi koblet til riktig. Lag en ny variabel som peker direkte på innholdet, som vi vet er i JSON-format:

```
satellitt_data = response.json()
```

Dersom du ser på innholdet nå er det en god del informasjon der i ren tekst. Du kan trykke på informasjonslinjen i Shell for å vise all informasjonen om du ønsker det.

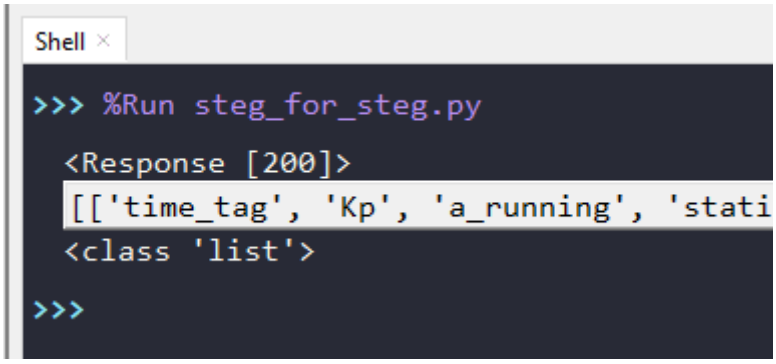
```
print(satellitt_data)
```



Figur 8: Skjermbilde av Shell i Thonny, tatt av Andøya Space Education

Vi ser at informasjonen er satt i et system med tegnet [og] og da betyr det at informasjonen er egentlig en liste i Python. Vi kan bekrefte det med å legge til en linje

```
print(type(satellitt_data))
```



Figur 9: Skjermbilde av Shell i Thonny, tatt av Andøya Space Education

Der ser vi at satellitt_data er en listetype.

Som en hvilken som helst liste, enten i et dataprogram eller på en skriveblokk, er det noe som er først og sist. Det første elementet i en Python-liste kan vises med koden:

```
first_entry = satellitt_data[0]
```

Ser vi på innholdet i variabelen `first_entry` med `print(first_entry)` får vi til svar `['time_tag', 'Kp', 'a_running', 'station_count']`

Der ser vi at det er fire «kolonner» i listen og denne første viser hva innholdet i hver er. Vi er ute etter «Kp», et tall som vil fortelle oss litt om aktiviteten fra solen. (Les mer om det i teoridelen.)

Den siste raden i den lange listen er det som er interessant for oss, for det er det nyeste. Vi skal altså plukke ut Kp-tallet fra bunnen av listen. Det siste elementet i en Python-liste kan hentes ut med `last_entry = satellitt_data[-1]` og da ser vi med `print(last_entry)` følgende verdier: `['2024-08-13 06:00:00.000', '2.33', '9', '8']`

Den første og den siste raden forteller oss at informasjonen ikke bare er en liste, men faktisk en liste med flere små lister inni seg. En liste over lister. Det kan fort bli litt forvirrende å tenke på, men her må vi grave oss litt dypere inn i data. Vi er ute etter Kp-tallet som vi så var nummer to i den indre listen. For å hente ut kun Kp-tallet fra den siste listen i listen legger vi til følgende linje: `print(last_entry[1])`

Legg merke til at element nummer to i en Python-liste er nummer 1. Årsaken til det er at en liste begynner på 0. Da blir 1 nummer to: (0, 1, 2, 3).

Hvis du nå kjører koden din, bør du se et desimaltall dukke opp i Shell. Verdien på tallet vil variere med tiden ettersom solvind og mengde partikler fra solen endrer seg hele tiden.

Den fullstendige koden ser nå slik ut:

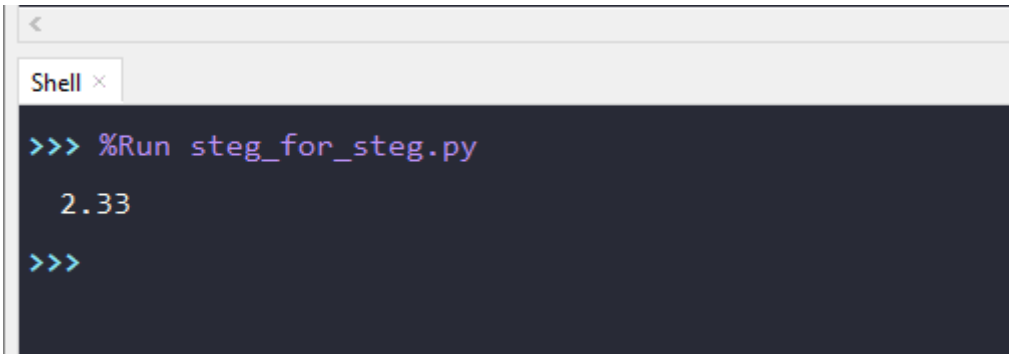
```
import requests
```

```
url = 'https://services.swpc.noaa.gov/products/noaa-planetary-k-index.json'
response = requests.get(url)
print(response)
satellitt_data = response.json()
print(satellitt_data)
print(type(satellitt_data))
first_entry = satellitt_data[0]
print(first_entry)
last_entry = satellitt_data[-1]
print(last_entry)
print(last_entry[1])
```

og er kanskje litt rotete. Vi kan rydde opp litt her og ta vekk noen linjer som vi har lagt inn underveis for å undersøke innholdet i variabler og data underveis. Vi er bare ute etter det Kp-tallet egentlig:

```
import requests
url = 'https://services.swpc.noaa.gov/products/noaa-planetary-k-index.json'
response = requests.get(url) satellitt_data = response.json()
last_entry = satellitt_data[-1]
print(last_entry[1])
```

Hvis du kjører det, skal programmet bare skrive ut til Shell en desimalverdi:



Figur 10: Skjermbilde av Shell i Thonny, tatt av Andøya Space Education

Denne verdien er det viktige, og den vil fortelle oss om det er sjanse for å se nordlys eller ikke. Vi trenger å fortelle Python litt til. Vi må la Python ta en beslutning for oss, basert på verdien av dette tallet. Dersom tallet er høyere enn 4, vil det være mulig å se nordlys, gitt at det er litt mørkt ute, ingen skyer og at du befinner deg litt nord i verden. Nord-Norge er ideelt for slike observasjoner.

Vi legger inn en IF-sjekk for å ta denne beslutningen:

```
if last_entry[1] >= 4:
    print('Nå er det kanskje nordlys')
else:
    print('Dessverre er det nok ikke nordlys å se nå')
```

Hvis du kjører programmet nå, vil du oppdage at det ikke fungerer som forventet. Det er en liten feil i programmet nå, ser du feilen? Thonny forteller oss at jeg forsøker å sammenligne to forskjellige verdier: `TypeError: '>=' not supported between instances of 'str' and 'int'`.

Vi forsøker å sammenligne tekst (str) med et heltall (int) og for Python blir det helt feil. Det blir som å sammenligne en katt og en hund, det er helt forskjellige typer dyr.

Vi må sørge for at de er begge samme type. I dette tilfellet vil jeg velge å gjøre `last_entry[1]` om til et desimaltall. Når vi leste inn data, klarte ikke Python å skjønne at det som så ut som et desimaltall var det, men trodde det var tekst.

```
if float(last_entry[1]) >= 4:
    print('Nå er det kanskje nordlys')
else:
    print('Dessverre er det nok ikke nordlys å se nå')
```

Slik, nå har vi et fullstendig dataprogram som vil koble seg til internett, hente ned ferske satellittdata og ta en beslutning for oss. Alt vi trenger å gjøre er å kjøre programmet, så gjør den jobben nå.

Vi kan videreutvikle koden vår slik at den blir litt mer funksjonell og lettere å lese. Vi kan

- Legge inn kommentarer som forklarer hva som skjer i koden
- Forkorte og eliminere bruk av variabler

Forsikre oss om at programmet fungerer som det skal hver gang Her er en oppdatert og fullstendig kode med noen forbedringer:

```
# Program for å vurdere sjansen for nordlys
# Skrevet av:

# Hente litt ekstra funksjonalitet
import requests

# Koble til internett og hente ut interessant data
url = 'https://services.swpc.noaa.gov/products/noaa-planetary-k-index.json'
try:
    response = requests.get(url)
    satellitt_data = response.json()
    kp_tall = float(satellitt_data[-1][1])

    # Ta en beslutning for oss
    terskel = 4.0
    if kp_tall >= terskel:
        print('Nå er det kanskje nordlys')
    else:
        print('Dessverre er det nok ikke nordlys å se nå')
except:
    print('Er du koblet til internett? ')
```

Aktivitet 2: Bygge en vinduapplikasjon

Programmet vi satte sammen i forrige aktivitet hadde det vi kaller et rent tekstbasert grensesnitt. Altså at det vi ser av programmet når vi kjører det foregår bare i et shell, en *terminal* som det også heter. Det var ingen vinduer, knapper eller andre mer «grafiske» elementer.

Nå skal bygge om programmet slik at det får et grafisk vindusbasert grensesnitt.

Ettersom vi allerede har satt sammen et program som gjør grovarbeidet med å hente ut Kp-tallet og gir oss en besluttende tekst om det blir nordlys eller ikke. Med en liten endring på det forrige programmet, kan vi gjenbruke det i vår nye applikasjon. Vi må gjøre om programmet slik at det er en funksjon som gir ut et tall og en tekst.

I den nye Python-filen, kall den `nordlys_vindu.py`, vil vi hente inn/importere funksjonen fra nordlys_tekst.py og legge tall- og tekstverdiene inn i to variabler samtidig:

```
from nordlys_tekst import get_kp
kp, beskjed = get_kp()
```

Før vi går videre åpner vi nordlys_tekst.py og endrer vi slik at det kjører som før, bare at arbeidet skjer i en funksjon med navn get_kp()

```
# Program for å vurdere sjansen for nordlys
# Skrevet av:

# Hente litt ekstra funksjonalitet
import requests

# Koble til internett og hente ut interessant data
url = 'https://services.swpc.noaa.gov/products/noaa-planetary-k-index.json'
def get_kp():
    try:
        response = requests.get(url)
        satellitt_data = response.json()
        kp_tall = float(satellitt_data[-1][1])

        # Ta en beslutning for oss
        terskel = 4.0
        if kp_tall >= terskel:
            beskjed = 'Nå er det kanskje nordlys'
        else:
            beskjed = 'Dessverre er det nok ikke nordlys å se nå'
    except:
        print('Er du koblet til internett? ')

    return kp_tall, beskjed
```

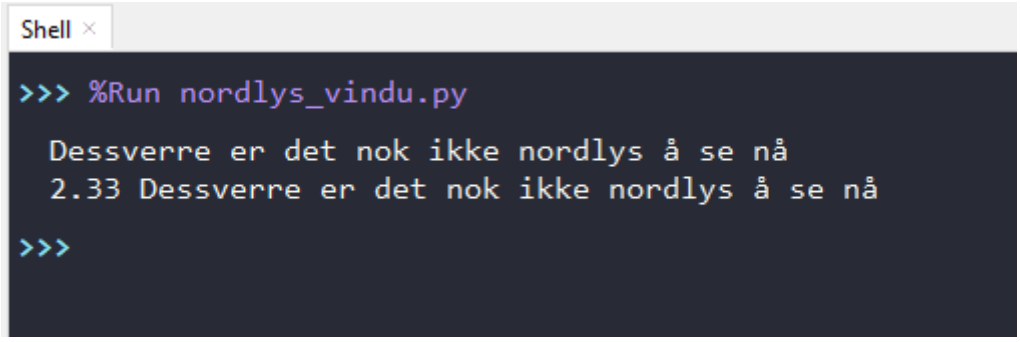
```
kp, beskjed = get_kp()
print(beskjed)
```

Den tidligere koden er nå pakket inn (innrykket) i en funksjon vi har gitt navnet `get_kp` og når den brukes, returnerer den et tall og en beskjed. På slutten bruker vi funksjonen og resultatet er som før, at den gir ut et nordlysvarsel for oss.

I det nye programmet vårt, `nordlys_vindu.py` kan vi nå forsøke å kjøre programmet, slik vi gjør det i `nordlys_tekst.py`:

```
from nordlys_tekst import get_kp
kp, beskjed = get_kp()
print(kp, beskjed)
```

I Shell ser jeg at det fungerer som forventet, men at det kommer et nordlysvarsel før det jeg har i det nye programmet:

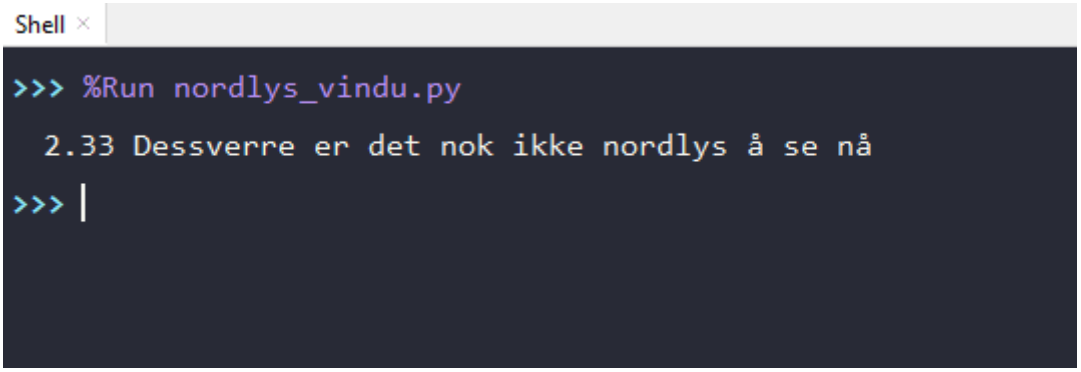


Figur 11: Skjermbilde av Shell i Thonny, tatt av Andøya Space Education

Årsaken er at nordlys_tekst.py har kode på slutten som også kjøres selv om det bare importerer funksjonen get_kp(). Python har en løsning på dette, og det er å pakke koden som bruker get_kp() inn i en litt spesiell funksjon som kalles main, som betyr at denne koden bare blir kjørt dersom nordlys_tekst.py kjøres direkte, og ikke når jeg bruker en funksjon fra filen.

```
if __name__ == '__main__':
kp, beskjed = get_kp()
print(beskjed)
```

Nå kan hver av filene kjøres som forventet.



Figur 12: Skjermbilde av Shell i Thonny, tatt av Andøya Space Education

Nå er første del gjort, og vi kan begynne på det vindusbaserte grensesnittet. For å få det til, skal vi låne litt vindufunksjonalitet fra noe som heter tkinter.

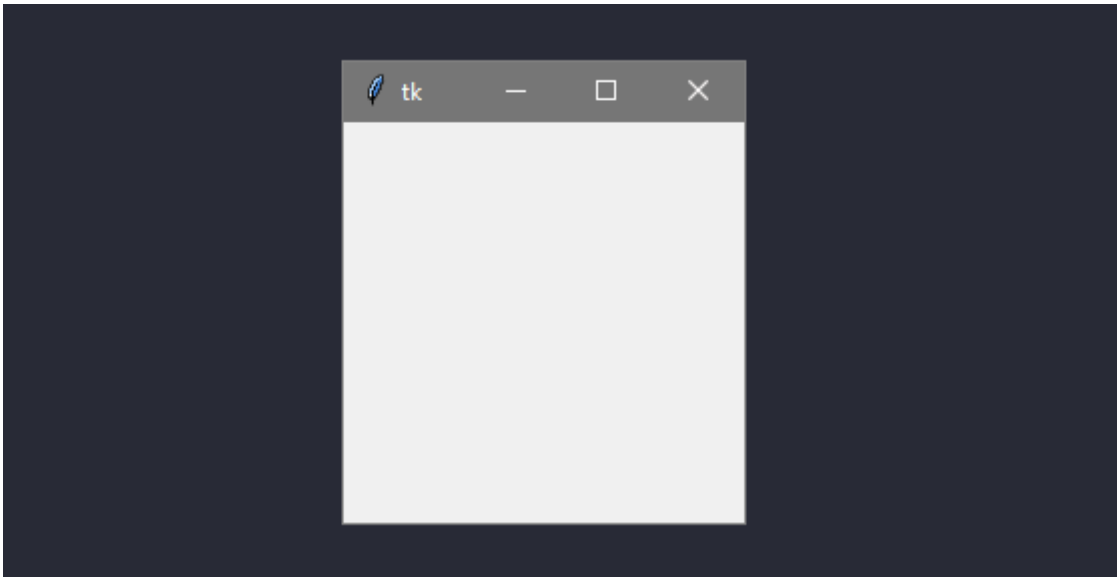
Vi legger inn `from tkinter import *` i toppen og et par linjer som setter opp et vindu for oss:

```
from tkinter import *
from nordlys_tekst import get_kp

kp, beskjed = get_kp()
print(kp, beskjed)

nordlys_vindu = Tk()
nordlys_vindu.mainloop()
```

Kjøres programmet nå, dukker det opp et bittelite vindu på skjermen, riktignok uten noe innhold, men vi har fått et vindu:



Figur 13: Skjermbilde av et kjørende grafisk vindu, tatt av Andøya Space Education

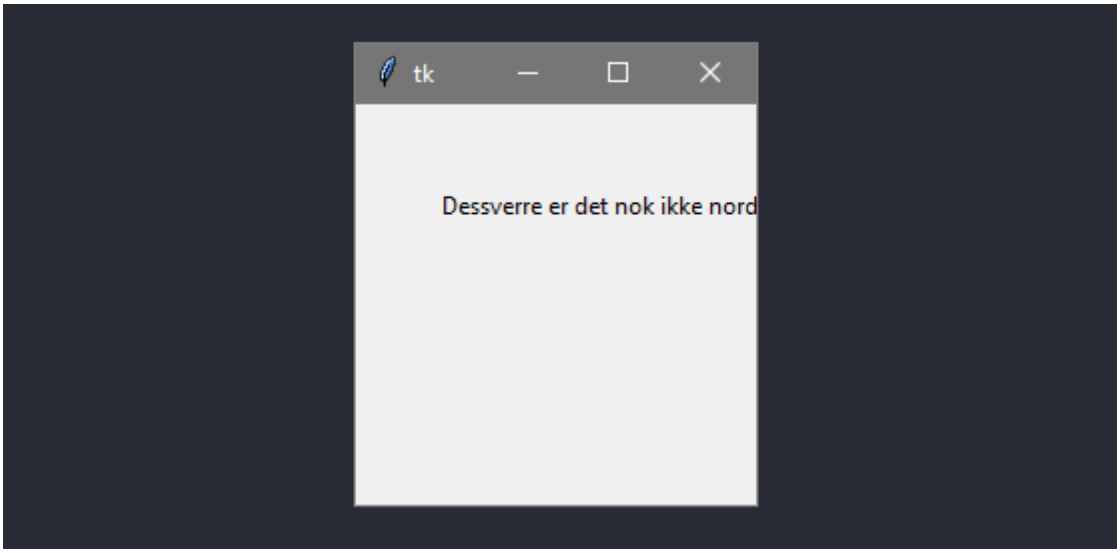
For å gjøre vinduet nyttig skal vi nå plassere nordlysvarselet inni vinduet. Vi oppretter en etikett som i tkinter-verden kalles en Label. Vi kan bruke følgende kode for å knytte variabelen *beskjed* med vinduet:

```
msg_label = Label(nordlys_vindu, text=beskjed)
```

Når de to er tilknyttet, kan vi plassere beskjeden der vi ønsker det. Vinduet er som et koordinatsystem med x og y. Oppe til venstre er x = 0 og y = 0. Vi forsøker med 40 i begge retninger:

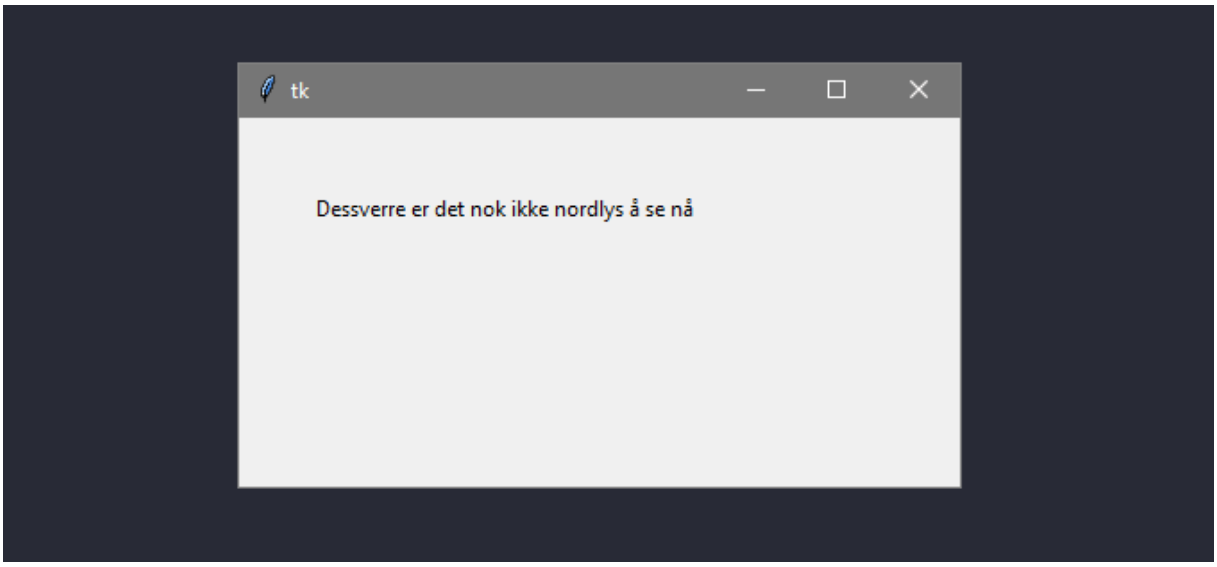
```
msg_label.place(x = 40, y = 40)
```

Den siste linjen må være nordlys_vindu.mainloop() som er den som setter i gang vinduet med det vi har satt sammen. Prøv selv.



Figur 14: Skjermbilde av et kjørende grafisk vindu, tatt av Andøya Space Education

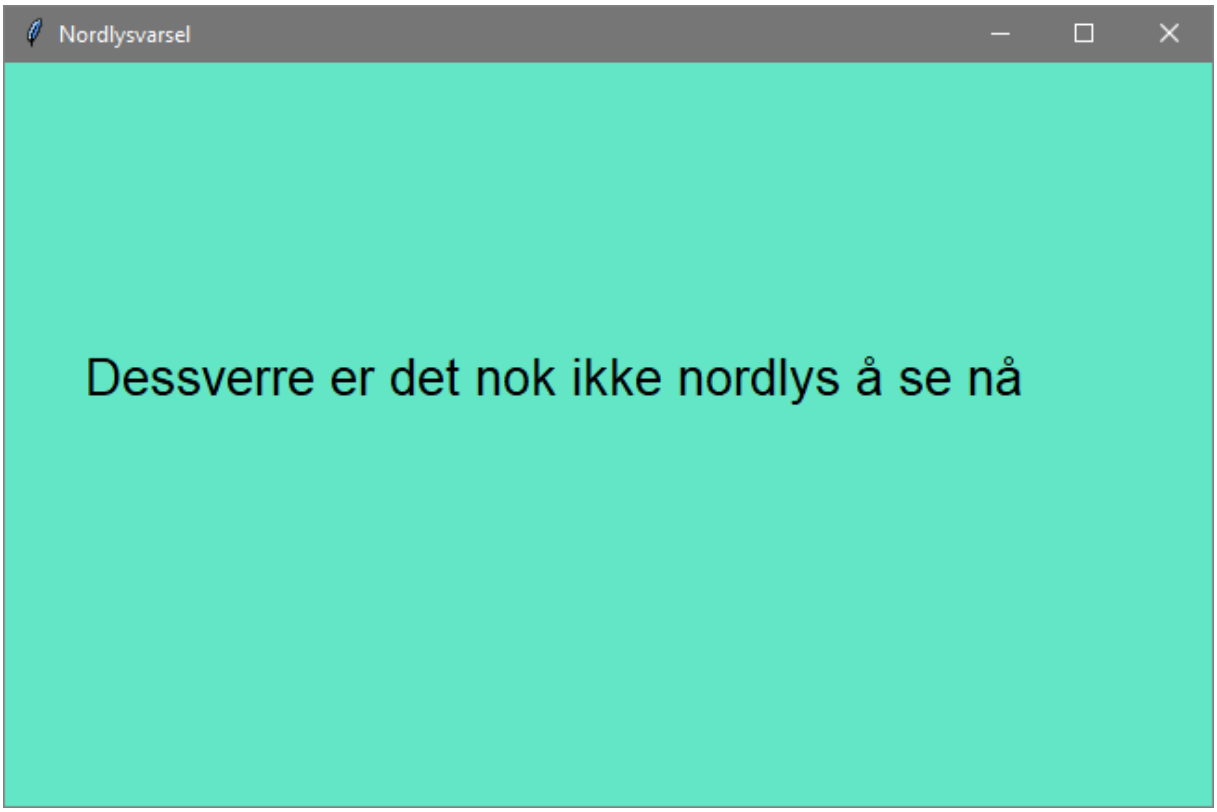
Vinduet er litt lite, siden vi ikke har sagt noe om størrelse, farge, tittel og slikt ennå. Det er kanskje å regne som litt kosmetisk. Trekker du i et hjørne og gjør vinduet større kan vi se at applikasjonen vår fungerer som den skal:



Figur 15: Skjermbilde av et kjørende grafisk vindu, tatt av Andøya Space Education

Gratulerer, du har nå bygd en applikasjon som gir nordlysvarsel fra satellittdata. Det er ganske kult, ikke sant?

Vi kan gjøre noen små grep, legge til noen linjer som forteller at vinduet skal se litt annerledes ut, og her kan du selv gjøre de endringene du ønsker:



Figur 16: Skjermbilde av et kjørende grafisk vindu, tatt av Andøya Space Education

Legg merke til hvor stor forskjell noen farger, skrift, størrelse og tittel på vinduet kan gjøre.

```
from tkinter import *
from nordlys_tekst import get_kp

# Henter inn funksjonen vår og setter opp et vindu
kp, beskjed = get_kp()
nordlys_vindu = Tk()

# Definerer noen farger, skrift, størrelse og tittel på vinduet
farge = '#63E6C5'
skrift = 'Helvetica'
nordlys_vindu.title('Nordlysvarsel')
nordlys_vindu.geometry('650x400')
nordlys_vindu.configure(background=farge)

# Legg på informasjonen
msg_label = Label(nordlys_vindu, bg=farge, text=beskjed, font=(skrift, 20))
msg_label.place(x = 40, y = 150)

# Kjør programmet
```

```
nordlys_vindu.mainloop()
```

Aktivitet 3: Bygge en nettapplikasjon

Til nå har vi sett på tekstbasert- og en grafisk vinduapplikasjon. I tekst bygde vi opp den grunnleggende funksjonen som vi gjenbruker i vindusapplikasjonen. Det samme skal vi gjøre nå når vi bygger opp en web- eller nettapplikasjon.

Denne gangen tar vi i bruk et annet bibliotek, *Flask*. Det inneholder det vi trenger av funksjonalitet for å sette opp en webserver, det som skal drive applikasjonen eller appen vår.

```
from flask import Flask
from nordlys_tekst import get_kp
```

På samme måte som for vinduappen vår, må vi fortelle Python at det skal konstruere en app: `app = Flask(__name__)`

Vi gjør det og bruker funksjonen vi har skrevet selv og legger verdiene i to variabler: `kp, beskjed = get_kp()`

Nå gjenstår det bare å lage en server som viser dette, og slik som med vinduet, må vi styre innholdet i variablene inn til nettsiden som vi setter opp.

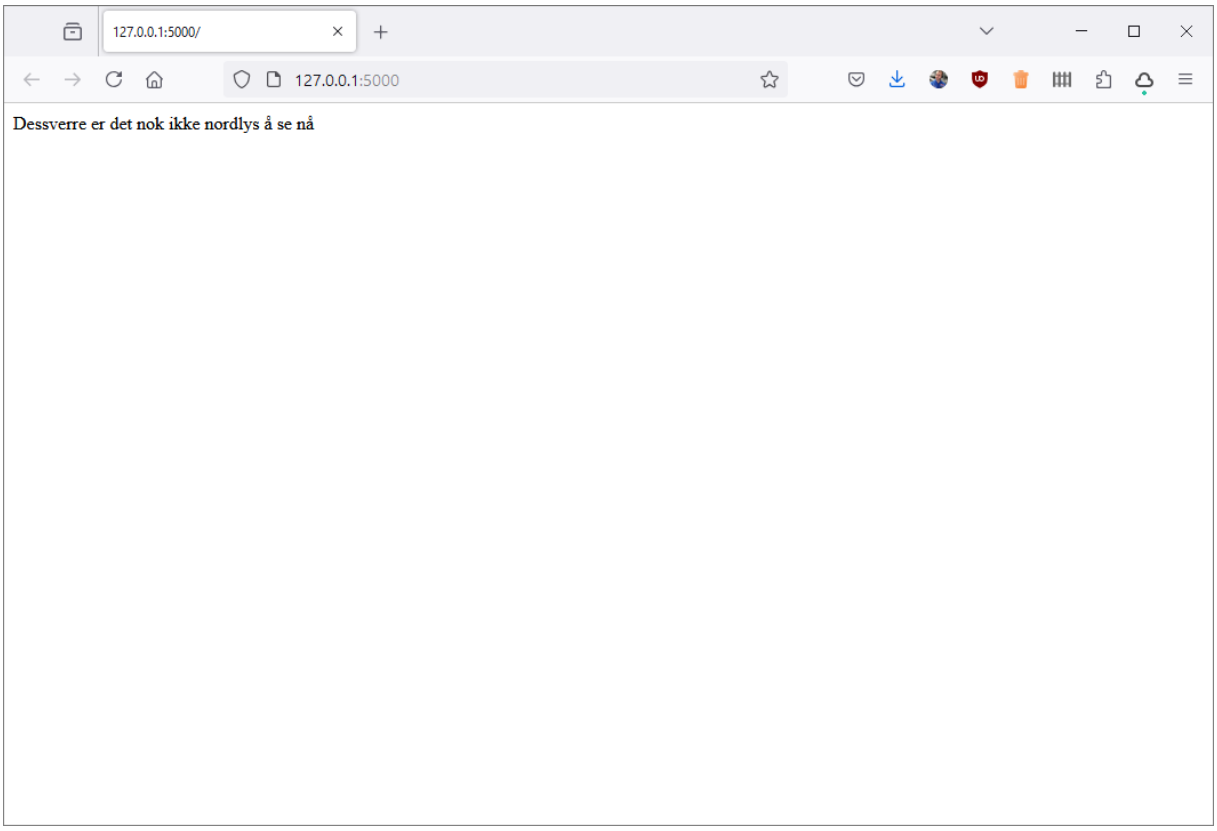
```
@app.route('/')
def varsel():
    return f'<h2 style="font-size:6vw">{beskjed}</h2>'

app.run()
```

Det som skjer med disse fire linjene er at det lages en funksjon `varsel()` som returnerer HTML-kode og pakkes inn i `route(«/»)`. Dette vil sette opp en side hvor HTML-koden vises. Til slutt settes applikasjonen i gang og vi kan bruke en nettleser for å se på nordlysvarselet. Kjør koden.

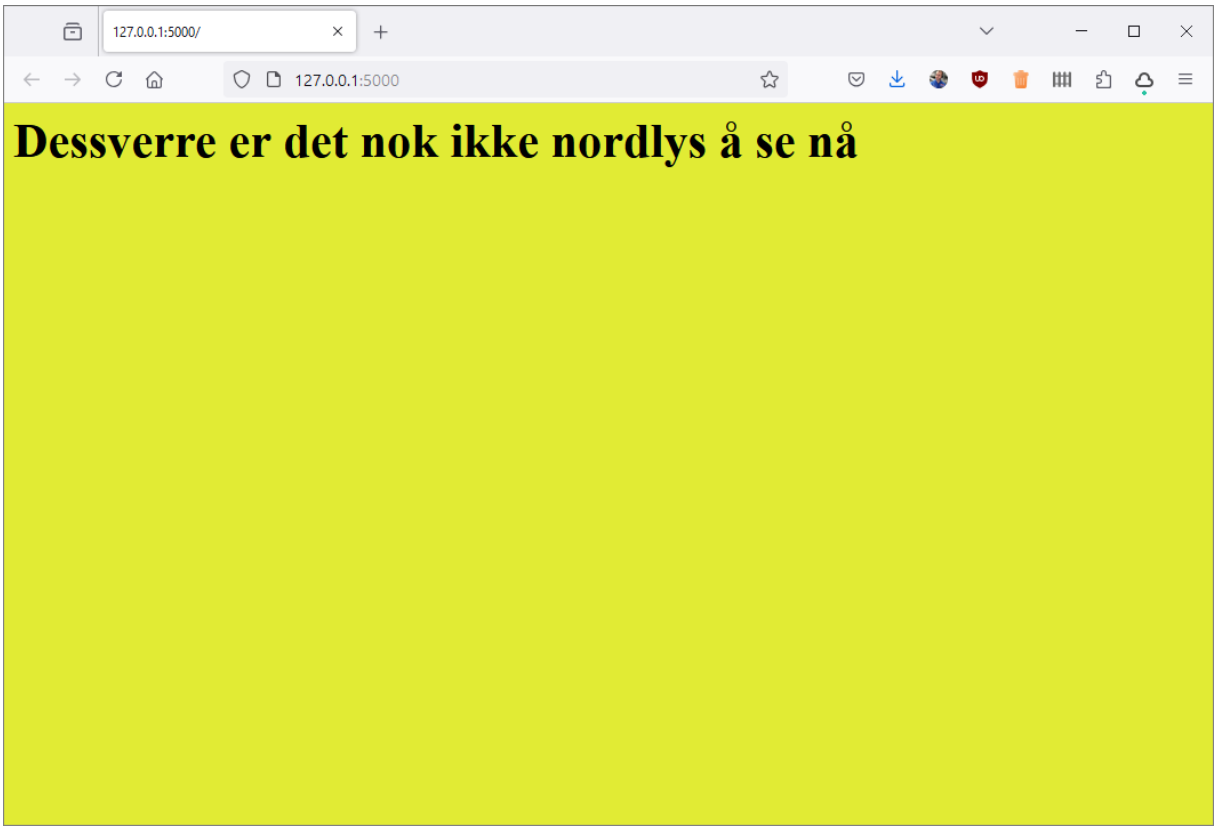
Får du feilmeldingen: `ModuleNotFoundError: No module named 'flask'` betyr det at du må legge den til i Thonny først, og starte Thonny om igjen.

Python har startet webserveren og du kan åpen en nettleser og skrive inn adressen http://127.0.0.1:5000 for å se applikasjonen din:



Figur 17: Skjermbilde av en nettleser, tatt av Andøya Space Education

Vår egen nettapplikasjon eller webapp! Vi kan endre litt på skriftstørrelsen og fargen slik at det ser litt bedre ut. Endre retursetningen fra `return beskjed` til `return f'<body style="background-color: #63E6C5;"><h2 style="font-size:4vw">{beskjed}</h2></body>'`.



Figur 18: Skjermbilde av en nettleser, tatt av Andøya Space Education

Samlet kode:

```
from flask import Flask
from nordlys_tekst import get_kp

app = Flask(__name__)

# hente data
kp, beskjed = get_kp()

@app.route('/')
def varsel():
    return f'<body style="background-color: #63E6C5;"><h2 style="font-
size:4vw">{beskjed}</h2></body>'

app.run()
```

Lærerveiledning

Fremgangsmåte

Aktivitetene i denne øvelsen kan gjennomføres slik du selv ønsker det. Du kan enten gi de den direkte og la de jobbe på egen hånd eller du kan veilede elevene gjennom dem.

Samkoding

Det er en fin øvelse å bygge opp koden sammen, hvor du og elevene sitter alle på hver sine laptoper og starter med blanke ark (i Thonny). Kommentarene underveis i aktivitetene kan være utgangspunkt for en stegvis utbygging av koden. Ved å bruke en slik metodikk kan dere sammen også oppleve og løse eventuelle utfordringer underveis. For eksempel vil kanskje koden ikke kjøre, da er det en nyttig øvelse å finne ut av det sammen. Er det noen elever som ser hva som er feil for eksempel. Det gjør også at bruken av Thonny som kodeverktøy blir mer konkret.

Du kan også begynne sammen for så la elevene kjøre sine egne veier med kode og samles etterpå igjen og snakke i plenum om hva slags løsninger de valgte og hvorfor. Så kan dere sammen implementere en samlet kode basert på innspill.

Jobbe på egen hånd

Du kan ta ut aktivitetene fra denne øvelsen og dele ut til elevene. Det er god trening å skrive av kode og kjøre den selv. Det ligger en del læring i det også. Det vil dukke opp skrivefeil ettersom Python og andre programmeringsspråk er svært pirkete på plassering av tegn, stor og små bokstaver og helt korrekt skriving.

På 80- og 90-tallet ble det distribuert mye kode gjennom magasiner. Det satt tusenvis av ungdom rundt omkring i verden som skrev av kode fra papir, side etter side og som i dag kanskje er de som står bak mye av systemene vi bruker og tar for gitt.

Andre ideer til space-applikasjoner:

YR har et nordlysvarsel som i bunn er basert på samme datakilde som det vi gjør i denne øvelsen, men de har også med skydekke som vil være aktuelt for å faktisk kunne observere nordlys fra bakken.

Figur 19: Skjermbilde av YR sitt nordlysvarsel, tatt av Andøya Space Education

På api.met.no er det mulig å hente ut prosentvis skydekke (cloud_area_fraction), adresse med eksempellokasjon (Andenes):

https://api.met.no/weatherapi/locationforecast/2.0/compact?lat=69.31&lon=16.11

Les mer på https://api.met.no/weatherapi/locationforecast/2.0/documentation

Her er flere datakilder som kan brukes til å lage spennende applikasjoner:

- Tre-dagers nordlysvarsel
  - https://services.swpc.noaa.gov/products/noaa-planetary-k-index-forecast.json
- International Space Station Current Location
  - http://open-notify.org/Open-Notify-API/ISS-Location-Now/
  - Datakilde: http://api.open-notify.org/iss-now.json
- How Many People Are In Space Right Now
  - http://open-notify.org/Open-Notify-API/People-In-Space/
  - Datakilde: http://api.open-notify.org/**astros**.json

Læringsmål

- Bli bedre kjent med Python som programmeringsspråk
- Bli vant med Thonny som kodeverktøy
- Lære om og ta i bruk ulike grensesnitt i dataprogrammer

- Oppleve ulike metoder for å lære programmering

Kilder

Det finnes ingen kilder i gjeldende dokument.

Fant ingen figurlisteoppføringer.

Lisensiering: