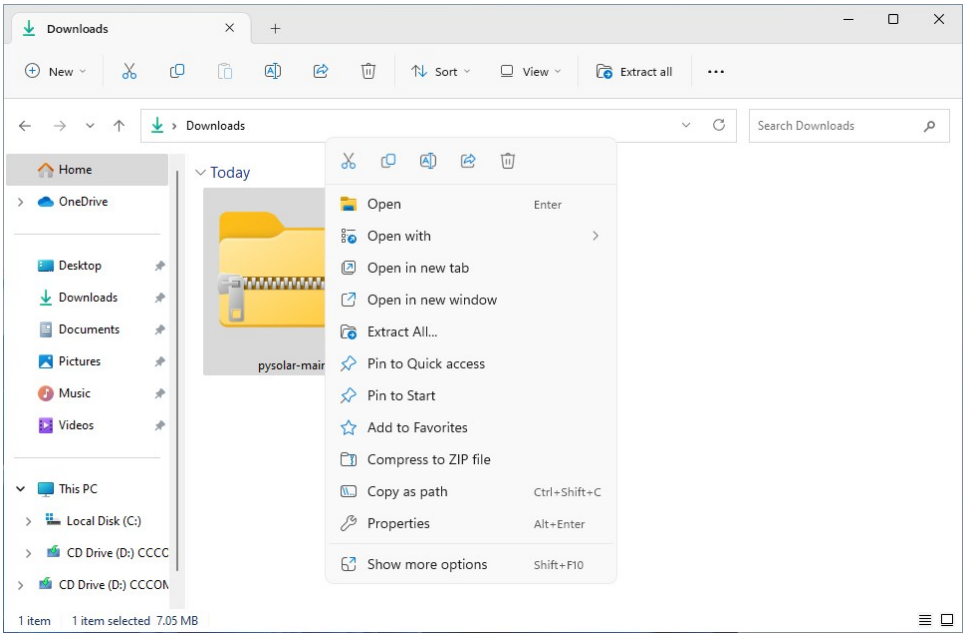
## Solsystem i Python.

Før vi begynner må vi laste ned en Template (ferdig laget kode) som skal hjelpe oss på veien.

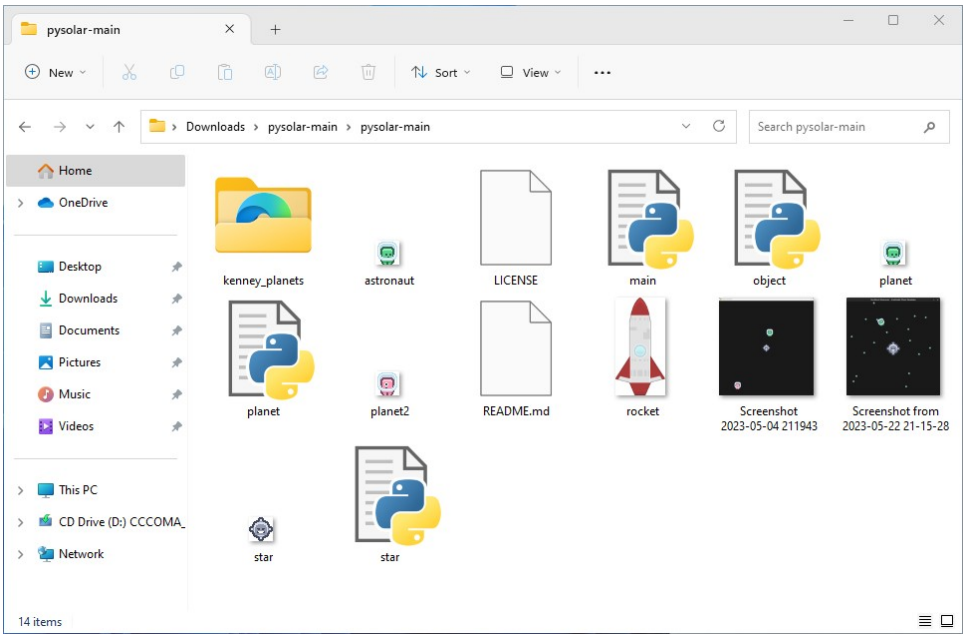
Vi finner denne koden på nettsiden til *GitHub*:

https://github.com/ktndbrg/pysolar/archive/refs/heads/main.zip

Dette vil laste ned en .zip fil, dette er en komprimert mappe som vi må utvide ved å trykke *Extract All*.



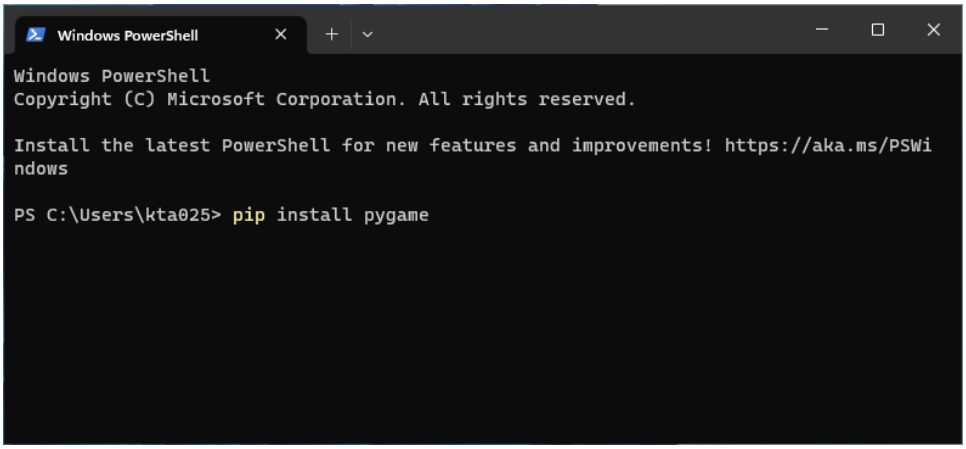
Resultatet skal se slikt ut.




For å kjøre programmet må vi ha installert *Python*, og *PyGame*.

Vi trenger senere en kode-editor, slik som *Visual Studio Code*.

Vi kan installere *PyGame* slik: *pip install pygame*




Åpne kildekoden (filene), som heter *main.py*, *planet.py*, og *star.py* i *Visual Studio Code.*

## Variabler.

Variabler er *lagret* verdier.

Disse er lagret i *minne* på datamaskinen, som ofte kalles for *RAM* (*Random-Access-Memory*).

Variabler kan tenkes som *egenskaper* til objekter, slik som *posisjon*, *hastighet*, og *utseende*.

På linje 23 i filen *planet.py*, står det:

```
21
22          # Radius of the orbit
23          self.radius = 250
24
```

Hva skjer om man endrer denne verdien?

Variabler kan skifte verdier, de kan også plusses med andre variabler.

De kan også inneholde tekst.

```
24
25          fornavn = "Kristoffer"
26          etternavn = "Tandberg"
27          print (fornavn + etternavn)
28
```

## Oppgaver

Kjør programmet etter hver endring.

1. Endre verdien `self.radius` til 25, hva skjer?
2. Endre verdien på `self.hastighet`, hva gjør denne?
3. Klarer du å få astronauten til å gå motsatt vei?
4. Hva skjer om du bruker piltastene i spillet?
5. I metoden render (linje 111 cirka), skift verdiene til DARK_GREY, hva gjør denne variabelen?
6. Endre skalering verdien til Planeten, kan du gjøre han flat?

## Lage nye objekter.

I filen *main.py* så kan vi lage flere objekter som går i sirkelbane.

På linje 30 lager vi et Planet-objekt.

Kopier (Ctrl-C) denne linjen og paste (Ctrl+V) den rett under, slik som bilde viser.

Vi kan endre bilde slik at vi ser forskjellen på objektene ved å endre linjen: `bilde = "rocket.png".`

```
28
29          # Create a Planet object
30          self.planet = Planet (bilde = "planet.png",
31                                vinkel = 0,
32                                radius = 250,
33                                hastighet = 0.25,
34                                skalering = (64, 64))
35
36          self.planet = Planet (bilde = "rocket.png",
37                                vinkel = 0,
38                                radius = 250,
39                                hastighet = 0.25,
40                                skalering = (64, 64))
41
```

Når vi kjører denne ser vi at verdiene her ikke gjør noe, vi må endre filen *planet.py* slik at den tar i bruk *argumentene*, og ikke *hard*-kodet verdier.

Vi endrer linjene 23, 26, og 29 slik som bildet viser.

Nå kan vi lage så mange objekter som mulig.

Dette programmet er laget som ett spill.

Det vil si at den har en *gameloop*, som blir oppdatert 60 ganger hvert sekund, dette kalles for FPS (*Frames-Per-Seconds*).

For at vi skal kunne se det nye objektet vi har laget må vi legge det til på linje 110 inni **update** metoden.

```
104
105     """
106         Update the game logic
107     """
108     def update (self):
109         # Update the planet
110         self.planet.update (self.clock, self.star.posisjon)
111
```

Vi må også gjøre samme inni **render** metoden, slik at den blir tegnet på skjermen.

**update** og **render** kjører med 60fps, som betyr at hvert 0.016666 sekund kjører koden inni disse.

## Oppgaver.

1. Lag 3 nye Planet objekter, disse skal ha forskjellige vinkel, hastighet, og radius.

2. Hva skjer om du skal lage 100 nye Planeter, ville dette ha tatt lang tid å skrive?

3. Se på metoden **run** i filen *main.py*, hva gjør de forskjellige tingene der?

4. I **run** metoden, skift `self.clock.tick (60)` til 10, hvor mye FPS har du nå?

## Lister og Løkker.

Vi kan legge alle planetene våre samlet i én liste.

Dette gjør at vi kan enkelt klare oppgave 2, med å lage 100 nye planeter på en enkel måte.

En liste er en samling av variabler.

Disse kan være tekst, tallverdier, eller lister.

Vi kan faktisk lage lister inni lister som inneholder andre lister.

Vi lager en liste som heter `self.planets`, denne skal inneholde alle planetene vi lager.
Dette kan gjøre ved å bruke en metode som heter *append* (engelsk ord for *Legg til på slutten*).

```
28
29          # Create a Planet object
30          self.planets = []
31          self.planets.append (Planet (bilde = "planet.png",
32                                       vinkel = 0,
33                                       radius = 250,
34                                       hastighet = 0.25,
35                                       skalering = (64, 64)))
36          self.planets.append (Planet (bilde = "planet.png",
37                                       vinkel = 0,
38                                       radius = 250,
39                                       hastighet = 0.25,
40                                       skalering = (64, 64)))
41
```

Men disse vises ikke på skjermen?

Vi trenger en måte å loop igjennom listen.

Dette gjøres med løkker (engelsk *loop*).

Den mest kjente er en for-løkke.

Inni metoden **update** skriver vi:

Denne koden loope igjennom listen, og kjører metoden `planet.update` for hvert objekt inni listen `self.planets`.

```
122
123         # Render the planets
124         for planet in self.planets:
125             planet.render (self.screen)
126
```

## Oppgaver.

1. Hva skjer om du mikser for-løkke inni en liste?
Når vi skal lage planeter, hva gjør denne koden?

```
self.planets = [Planet (image = "astronaut.png", vinkel = math.pi * i / 6, radius = 150 + i * 100 * random.random (), hastighet = random.random(), skalering = (16, 16)) for i in range (250)]
```

2. Hva gjør denne koden:

```
28
29          for i in range (0, 100):
30              print (i)
31
```

## Måne.

Vi skal avslutte med å lage en måne til planeten vår.

Vi ser at metoden update (*main.py*) gir noe som heter self.star.posisjon. Denne verdien er hvor en orbit skal skje.

Vi kan bruke denne til å lage en måne.

Først må vi lage en variabel som heter `self.måne`

```
29          # Create a Planet object
30          self.planet = Planet (bilde = "planet.png",
31                                vinkel = 0,
32                                radius = 250,
33                                hastighet = 0.25,
34                                skalering = (64, 64))
35          self.måne = Planet (bilde = "rocket.png",
36                                vinkel = 0,
37                                radius = 50,
38                                hastighet = 2*math.pi,
39                                skalering = (16, 16))
40
```

Denne kan ha en høyere hastighet, og kanskje være mindre størrelse enn planeten?

Nå må vi legge den til i update, men her gir vi posisjonen til planeten den skal sirkle rundt.

```
107     def update (self):
108         # Update the planet
109         self.planet.update (self.clock, self.star.posisjon)
110         self.måne.update (self.clock, self.planet.posisjon)
111
```

Deretter blir det vanlig å legge den til i render.

```
122
123         # Render the planets
124         self.planet.render (self.screen)
125         self.måne.render (self.screen)
126
```

Resultatet blir en måne som sirkler rundt planeten.

## Oppgaver.

1. Lag en liste av måner

2. Lag 100 måner som sirkler rundt planeten

## Ekstra Oppgaver [Vanskelige].

1. Spillet skal starte med 0 planeter.
Hvis du trykker på tasten "1" så skal det bli laget en (ekstra) planet.
[Her må du gå inn i metoden events, og bruke append].

Skrevet av Kristoffer Tandberg, i bruk til NordNorsk Vitensenteret – SKILLS Kodeklubb.

https://github.com/ktndbrg/pysolar