Ditt første script

Vi ser kort på følgende script.

```
print('Hello, World!')
```

Du kan selv lagre teksten under i en fil `hello_world.py`, og kjøre den i spyder ved å trykke på "play"-knappen, eller kommandoen `python hello_world.py` i en terminal.

Under ser du en kjøring fra en `cmder`-terminal.

```
 python hello_world.py
Hello, World!
```

Oppgave 1:

Kjør programmet `hello_world.py` på din egen datamaskin.

filnavn: `hello_world.py`

Kort dissekering av programmet `hello_world.py`

Strengen `Hello, World!` blir gitt som input til funksjonen `print`. Funksjonen `print` skriver så ut teksten `Hello, World!` til standard-output. Standard-output (`stdout`) kan være terminalen, eller ipython-vinduet ditt i Spyder.

Du skal senere lære mer om funksjoner. Akkurat nå kan du tenke på `print` som en måte å skrive ut verdien av variable og annet programmet ditt har beregnet.

Variable

Vi kan lage scriptet mer dynamisk ved å bruke variable.

```
fornavn = 'Kato'
etternavn = 'Pedersen'

print('Hello ' + fornavn + ' ' + etternavn + '!')
```

Vi kjører scriptet:

```
 python hello_kato.py
Hello, Kato Pedersen!
```

Du finner scriptet i filen `hello_kato.py`.

Oppgave 2

Bytt verdien til variablene `fornavn` og `etternavn` til strenger for henholdsvis ditt eget fornavn og etternavn.

filnavn `hallo_navn.py`

Valg av variabelnavn.

Merk at vi har valgt å bruke hele ord som variabelnavn. Det kan være fristende å la `f` være variablen for fornavn og `e` variabelen for etternavn. Vi får da følgende kode:

```
f = 'Kato'
e = 'Pedersen'

print('Hello ' + f + ' ' + e + '!')
```

Problemet med denne koden er at det ikke er like tydelig hva variablene `f` og `e` inneholder, selv om det i dette eksempelet fortsatt er ganske oversiktlig. Legg til deg en god vane og velg variabelnavn som er

- beskrivende og mest mulig fullstendige
- ikke for lange

Dersom variablene brukes i matematiske beregninger bør du sørge for å velge navnet slik at koden blir mest mulig lik matematikken. Da er det lettere å finne feil når du sammenlikner skisser og beregninger for hånd med koden du eller andre har skrevet.

Repetisjon: Bruk av variable til å gjøre matematiske beregninger

Varmestråling med Stefan-Boltzmanns lov

Hvor mye varme stråler det ut fra en ovn som holder 300 grader celcius? Dette tilsvarer 593.15 K (grader Kelvin).

Varmestrålingen Φ fra et såkalt sort legeme med absolutt temperatur T, målt i Kelvin, er gitt ved

$$\Phi = \sigma T^4,$$

hvor σ er Stefan-Boltzmanns konstant, gitt ved

$$\sigma = \frac{2\pi^5 k_B^4}{15c^2h^3} = 5.670400 \cdot 10^{-8}\mathrm{Wm}^{-2}\mathrm{K}^{-4}.$$

Vi kan nå regne ut varmefluksen (varmestråling per kvadratmeter) med `python`.

```
sigma = 5.6704E-8
o = sigma
T = 593.15

phi = o*T**4

print("Energifluks :", phi, " Watt/kvadratmeter")
```

Vi kjører scriptet:

```
 python stephan_boltzmann.py
Energifluks : 7018.944927276633  Watt/kvadratmeter
```

Et sort legeme kan ofte brukes med god tilnærming for objekter som absorberer stråling godt. For støpejern er strålingen ofte 70% - 90% av strålingen beregnet med Stephan - Boltzmanns lov.

Vi kan skrive fluksen for støpejern, Φ_s som f.eks

$$\Phi_s = 0.8\sigma T^4.$$

Generelt kan vi skrive

$$\Phi_s = \epsilon \cdot \sigma T^4,$$

der ϵ er emissiviteten til materialet. Den er ofte en funksjon av temperaturen.

Oppgave

Du fyrer i en støpejernsovn slik at overflatetemperaturen blir 250 grader celsius. Ovnen har et overflateareal på 1.9m^2. Anta at emmisiviteten til støpejernet ved denne temperaturen er på $\epsilon = 0.68$. Bruk Stephan-Boltzmanns lov og beregn strålingseffekten til denne ovnen. Gi svaret i kilowatt (kW).

filnavn: `stefan_boltzmann_ovn.py`.

Beregning av stoffmengde i kjemi

Dersom du slipper ut 15 kg CO_2, så har du sluppet ut 15000 gram. Hvor mange molekyler er dette?

Fra kjemi har du kanskje lært om formelen

$$\text{stoffmengde (mol)} = \frac{\text{masse (g)}}{\text{molar masse (g/mol)}}, \qquad n = \frac{m}{M_m}$$

Vi kjenner massen $m = 15000$g, og for CO_2 er den molare massen $M_m = 44$g/mol. Vi regner ut antall mol og antall molekyler med `python`

```
avogadro_konstant = 6.28E23   # avogadros tall
M_oksygen = 16
M_karbon = 12
M = M_karbon + 2*M_oksygen    # molar masse (g/mol)
m = 15000                     # masse (g)

n = m/M                       # antall mol

antall_molekyler = n*avogadro_konstant

print("antall mol: ", n)
print("antall molekyler: ", antall_molekyler)
```

Vi kjører scriptet:

```
 python molekyltelling.py
antall mol:  340.90909090909093
antall molekyler:  2.140909090909091e+26
```

Beregning av konsentrasjon i kjemi

Løkker, lister, tupler og litt om dictionaries

Gruppering og indeksering av data

Python tilbyr flere måter å samle og strukturere data på. Blant disse er listen den enkleste å bruke. Listen er en svært fleksibel datastruktur, den kan inneholde hva som helst! Python tilbyr en enkel syntax for lister.

Utklippet under fra en session i `ipython` viser hvordan en liste kan lages, og hvordan du kan hente ut informasjon fra listen. Merk at elementene i listen nummereres fra 0.

```
In [1]: # lag en liste

In [2]: bokstaver = ['a', 'b', 'c', 'd', 'e', 'f']

In [3]: bokstaver[0]       # hent ut første element, med indeks 0
Out[3]: 'a'

In [4]: bokstaver[5]       # hent ut sjette element, med indeks 5
Out[4]: 'f'

In [5]: bokstaver[-1]      # hent ut siste element i listen
Out[5]: 'f'

In [6]: bokstaver[-2]      # hent ut nest-siste element i listen
Out[6]: 'e'

In [7]: len(bokstaver)     # hvor mange elementer er det i listen (tell!)?
Out[7]: 6
```

Oppgave

Lag en liste navn som inneholder alle navnene dine (fornavn, etternavn og mellomnavn). Print ut listen med kommandoen `print(navn)`.

filnavn: `list_navn.py`

Slicing

Vi kan lage en "underliste" av listen `bokstaver`

```
In [8]: # lag en liste som inneholder en del av den opprinnelige listen

In [9]: bokstaver[1:5]
Out[9]: ['b', 'c', 'd', 'e']
```

Ved kommandoen `bokstaver[1:5]` får vi altså 1. element, 2. element, 3.element og 4.element av listen `bokstaver`.

Den siste kommandoen `bokstaver[1:5]` gir oss en liste med elementene i indeksene 1, 2, 3, og 4 fra `bokstaver`. Kommandoen er altså inklusiv fra start, og ekslusiv til slutt.

Lister kan inneholde ulike typer data

Lister kan inneholder objekter av ulike typer:

```
In [1]: mixedlist = ['a', 5, 5.2, 'b', 'c']

In [2]: mixedlist[3]
Out[2]: 'b'

In [3]: mixedlist[4]
Out[3]: 'c'
```

Lister kan inneholde lister

En liste som inneholder lister med `[fornavn, etternavn, alder]`.

```
elever = [['Adrian', 'Hansen', 16], ['Kim', 'Pedersen', 17],\
['Stine','Johansen', 16], ['Mona', 'Jensen', 17]]
```

Merk at vi bruker en back-slash for å fortelle python at samme linjen kode fortsetter på neste linje.

Metoder for lister

Siden en liste også er et objekt, kommer den med et sett innebygde metoder. En svært nyttig metode er `append`

```
In [1]: somelist = [0.1, 0.2, 0.3, 0.4]

In [2]: somelist.append(0.5)

In [3]: somelist
Out[3]: [0.1, 0.2, 0.3, 0.4, 0.5]
```

Metoden tilføyer altså et element på slutten av listen.

En annen nyttig metode er `pop`.

```
In [4]: somelist.pop()
Out[4]: 0.5

In [5]: somelist
Out[5]: [0.1, 0.2, 0.3, 0.4]
```

Metoden `pop` returnerer ("gir ut") den siste verdien i lista, men fjerner også denne verdien fra listen. Dette er svært nyttig dersom man bruker listen til å modellere for eksempel en kø, eller må sette ulike operasjoner datamaskinen skal gjøre i en kø.

Du får opp metodene for en liste ved å lage en liste, f.eks `somelist` som over i python, og skrive `dir(somelist)`. Du trenger ikke tenke på metodene med dobbel understrek rundt på dette tidspunktet, kun de siste metodene `append, clear, copy, count, extend, index, insert, pop, remove, reverse` og `sort`. I tillegg får du lengden av listen (antall elementer) når du skriver `len(somelist)`.

Løkker: while-løkken

Frem til nå har python i beste fall vært en avansert kalkulator, som i tillegg kan telle som en helt og beregne lengden på ord og korte tekster. Problemet er at vi må gjøre svært mange instruksjoner selv, og gjenta nesten identiske instruksjoner. Hvis programmering er kjedelig, og man gjentar seg selv, finnes det garantert en bedre måte å gjøre det på!

For å virkelig få fart på sakene, og slippe å gjenta oss selv, bruker vi løkker. Python kommer med to typer løkker: `for` - løkken og `while`-løkken.

Vi begynner med sistnevnte.

```
while (condition):
    #do something interesting
    ...
```

For-løkker

Lister og strenger

```
navn = "Jens Ove Karlsen Kristiansen"

navn_liste = navn.split()
```

Argumenter til `.split`-metoden.

`.strip`-metoden.

`.join`-metoden.

Litteratur og tutorials

Deep copy og shallow copy

Se på følgende kode:

```
x = 5      # variabelen x får verdien 5
y = x      # variabelen y får samme verdi som x
y = 10
print(x)
```

Skriver programmet ut verdien til `x` - altså 5, eller skriver den ut 10? Uansett utfall - dette behøver en grundig forklaring! Tutorial - deep copy og shallow copy av lister