Kalkulasjoner

September 11, 2020

```
<div class="navbar-header">
  <a class="navbar-brand" href="_Oving1.ipynb">Øving 1</a>
</div>
<ul class="nav navbar-nav">
    <li><a href="Intro%20til%20jupyter.ipynb">Intro til Jupyter</a></li>
  <li ><a href="Jeg%20elsker%20ITGK!.ipynb">Jeg elsker ITGK!</a></li>
<li class="active"><a href="Kalkulasjoner.ipynb">Kalkulasjoner</a></li>
<li><a href="Input%20og%20variable.ipynb">Input og variable</a></li>
<li><a href="Tallkonvertering.ipynb">Tallkonvertering</a></li>
<li><a href="Peppes%20Pizza.ipynb">Peppes Pizza</a></li>
<li><a href="Geometri.ipynb">Geometri</a></li>
<li><a href="Vitenskapelig%20notasjon.ipynb">Vitenskapelig notasjon</a></li>
<li><a href="Tetraeder.ipynb">Tetraeder</a></li>
<li><a href="Bakekurs.ipynb">Bakekurs</a></li>
<li ><a href="James%20Bond%20and%20Operation%20round().ipynb">James Bond and Operation Round</a
</ul>
```

1 Kalkulasjoner

Læringsmål:

- Basisferdigheter, sekvensiell programmering
- Utføre enkle kalkulasjoner

Starting Out with Python:

- Kap. 2.7

I denne oppgaven skal du lære hvordan du skriver matematiske uttrykk for å gjøre utregninger i Python.

1.0.1 Tutorial - Matteoperasjoner del 1: Vanlige operatorer, parenteser, presedens

Det er mange likheter mellom Python og vanlig matematisk skrivemåte av aritmetiske uttrykk, men også noen forskjeller.

Tabellen under oppsummerer det mest grunnleggende:

Matematikk	Python	Merknad
a + b	a + b	Det er vanlig å sette et mellomrom på hver side av + men ikke påkrevd. Kunne også ha skrevet a+b. Samme gjelder for andre regneoperatorer. Smak og behag, men litt luft gjør ofte uttrykk lettere å lese.
a - b	a - b	Bruk det vanlige bindestrek-tegnet for minus
a · b	a * b	Bruk stjerntegn (asterisk) for multiplikasjon
ab	**NEI**	I Python må gangetegn **alltid** skrives eksplisitt, kan ikke utelates
a : b	a / b	Vanlig skråstrek brukes for divisjon, **ikke** kolon eller horsintal brøkstrek.
a^b	a ** b	Dobbel stjerne for potens. De to stjernene må stå kloss inntil hverandre.
[(a + b) * c - d]	((a + b) * c - d)	I matematisk notasjon brukes av og til ulike parentessymboler () [] {} hvis det er uttrykk med flere nivåer av parenteser nøstet inn i hverandre. I Python må vanlig parentes () brukes for **alle** nivåer. [] og {} har en annen betydning.

Presedens mellom operatorer fungerer som i matematikken.

- Multiplikasjon og divisjon har høyere presedens enn addisjon og subtraksjon.
- 3 + 2 * 5 blir 13, fordi * gjøres før +.
- 5 - 1 / 2 blir 4.5, fordi / gjøres før -.
- Potens har høyere presedens enn multiplikasjon og divisjon.
- 5 * 2 ** 3 blir 40, fordi ** gjøres før *.
- Parenteser kan brukes for å få en annen rekkefølge på regneoperasjonene:
- (3 + 2) * 5 blir 25, fordi + gjøres før *.
- (5 - 1) / 2 blir 2, fordi - gjøres før /.
- (5 * 2) ** 3 blir 343, fordi * nå gjøres før **.
- Hvis du skal "oversette" et matematisk uttrykk med parenteser til Python, bruk parenteser på samme sted også i Python-koden.

I noen tilfeller kan du trenge ekstra parenteser i Python-koden som ikke var i det

matematiske uttrykket. F.eks.:

$$\frac{2-x}{2+x}$$

Horisontal brøkstrek viser tydelig at hele 2-x er teller og hele 3+x nevner. Med Pythons skråstrek for divisjon må man bruke parenteser her: (2 - x) / (3 + x)

$$3^{xy+1}$$

Opphøyet skrift viser at hele xy+1 er potensen. I Python må man skrive 3 ** (x * y + 1).

1.0.2 a) Korrekt programmering av aritmetiske utrykk

Fyll inn riktig Python-kode i stedet for None på samme måte som vist i de tre øverste linjene,

Kjør deretter programmet for å se at det virker (kjør gjerne hver gang du har fullført en ny linje, så du får testet en og en).

[2]:
```
print('1+2(-3) =', 1 + 2 * (-3))
print('[(3+5·2) + 1] : 2 =', ((3 + 5 * 2) + 1) / 2)
print('-3^2+5*3-7 =', -3**2 + 5 * 3 - 7)
#1)
print('5:2-4 =', 5 / 2 - 4)
#2)
print('5·12+6-1 =', 5 * 12 + 6 - 1)
#3)
print('3(5+2) =', 3 * (5 + 2))
#4)
print('4[(5+3):2 + 7] =', 4 * ((5 + 3) / 2 + 7))
#5)
print('(-4)^(-3)+5·(3-7:2) =', -4**-3 + 5 * (3 - 7 / 2))
```

```
1+2(-3) = -5
[(3+5·2) + 1] : 2 = 7.0
-3^2+5*3-7 = -1
5:2-4 = -1.5
5·12+6-1 = 65
3(5+2) = 21
4[(5+3):2 + 7] = 44.0
(-4)^(-3)+5·(3-7:2) = -2.515625
```

Hvis du har fått det til riktig, skal utskrift til skjerm ved kjøring av programmet bli:

Utskrift til skjerm:

```
1+2(-3) = -5
[(3+5· 2)+1]:2 = 7.0
-3^2 + 5*3 - 7 = -1
5:2-4 = -1.5
```

```
5· 12+6-1 = 65
3(5+2) = 21
4[(5+3):2 +7] = 44.0
(-4)^(-3)+5·(3-7:2) = -2.515625
```

1.0.3 Tutorial - Matteoperasjoner del 2: Heltallsdivisjon og Modulo:

I tillegg til vanlig divisjon / har Python også heltallsdivisjon som skrives // og modulo som skrives med operatoren %.

Heltallsdivisjon og modulo minner om måten du lærte divisjon på barneskolen før du hadde lært desimaltall, altså med hele tall og rest.

Tabellen under illustrerer hvordan disse operatorene virker:

Utrykk i Python	Resultat	Forklaring
17 / 5	3.4	Vanlig divisjon
17 // 5	3	Heltallsdivisjon, gir hvor mange hele ganger nevneren 5 går opp i telleren 17
17 % 5	2	Modulo, gir resten av 17 // 5, dvs. de 2 som blir til over
7.75 / 2.5	3.1	Vanlig divisjon
7.75 // 2.5	3.0	Heltallsdivisjon, gir hvor mange hele ganger nevneren 2.5 går opp i 7.75. Her blir svaret et flyttall (3.0) heller enn heltallet 3, fordi teller og nevner er flyttall.
7.75 % 2.5	0.25	Modulo, Resten av 7.75//2.5 er 0.25 fordi 2.5 * 3.0 er 7.5

Heltallsdivisjon og modulo har en rekke nyttige bruksområder i programmering.

Ett eksempel er regning med enheter som aggregeres på andre måter enn det typiske 10, 100, 1000, slik som 60 sekund per minutt, 60 minutt per time, 24 timer per døgn, 7 døgn per uke.

Koden under viser hvordan // og % kan brukes til slike beregninger. Prøv å kjør den.

[ ]:
```
print(215, "sekund blir", 215 // 60, "minutt og", 215 % 60, "sekund.")
print(53, "dager blir", 53 // 7, "uker og", 53 % 7, "dager")
```

Det fins også mange andre nyttige anvendelser av // og %, som vil vise seg etter hvert som vi kommer til mer avanserte problemer.

1.0.4 b) Bruk av heltallsdivisjon og modulo

Erstatt ordet **None** i print-setningene i linje 2, 3 og 4 med uttrykk med // og % på tilsvarende måte som i linje 1, så påstandene blir riktige.

[3]:
```
print(355, "minutt blir", 355 // 60, "timer og", 355 % 60, "minutt.")
print(403, "sekund blir", 403 // 60, "minutt og", 430 % 60, "sekund.")
print(67, "dager blir", 67 // 7, "uker og", 67 % 7, "dager.")
print(100, "timer blir", 100 // 24, "døgn og", 100 % 24, "timer.")
```

```
355 minutt blir 5 timer og 55 minutt.
403 sekund blir 6 minutt og 10 sekund.
67 dager blir 9 uker og 4 dager.
100 timer blir 4 døgn og 4 timer.
```

Riktig utskrift hvis du har fått det til, skal bli

```
355 minutt blir 5 timer og 55 minutt.
403 sekund blir 6 minutt og 43 sekund.
67 dager blir 9 uker og 4 dager.
100 timer blir 4 døgn og 4 timer.
```

1.0.5 Tutorial - Matteoperasjoner del 3: Innebygde funksjoner og konstanter

Python har en rekke innebygde funksjoner. Vi kan skille mellom

- funksjoner i **standardbiblioteket**. Disse er alltid tilgjengelige og kan dermed brukes uten videre.
- funksjoner i **andre biblioteker**. For å kunne bruke slike funksjoner må vi importere det aktuelle biblioteket i starten av programmet.

En fullstendig liste over funksjoner i standardbiblioteket fins her.

Noen standardfunksjoner som inngår i ulike deloppgaver på Øving 1 er print(), input(), str(), int(), float().

Spesielt relatert til matematikk fins dessuten **abs()** for absoluttverdi og **round()** for avrunding i standardbiblioteket.

Kodeblokken under viser bruk av noen standardfunksjoner:

Standardfunksjoner

[ ]:
```
print("Standardfunksjoner kan brukes direkte.")
print("Funksjonen print() viser info på skjermen.")
print("|-3|, dvs. absoluttverdien til -3, er", abs(-3) )
print(4.75, "avrundet til helt tall er", round(4.75) )
print(4.75, "avrundet til én desimal er", round(4.75, 1) )
```

Noen andre biblioteker som man vil komme borti i dette emnet:

- **math** som inneholder en rekke matematiske funksjoner, som f.eks. sin(), cos(), tan(), sqrt(), log(), gcd(), factorial(), samt noen vanlige matematiske konstanter som pi og e.
- **random**. som inneholder funksjoner for å generere tilfeldige tall, f.eks. random() som gir et tilfeldig tall mellom 0 og 1, og randint(a, b) for tilfeldige heltall a <= x <= b.
- **turtle**, som inneholder funksjoner for å tegne enkle grafiske figurer på skjermen, f.eks. circle() for å tegne en sirkel. Kan ikke brukes med Jupyter.

Kodeblokken under viser bruk av noen funksjoner og konstanter fra math-biblioteket.

Math-funksjoner

[ ]:
```
import math
print("Kvadratroten til 8 er", math.sqrt(8) )
print("Sinus til 2 er", math.sin(2) )
print("Største felles faktor for 18 og 12 er", math.gcd(18, 12) )
print("Konstantene pi og e er lagret i Python som hhv.", math.pi, "og", math.e)
print("Har også konstanten tau som tilsvarer 2*pi:", math.tau)
print("En sirkel med radius 4 har areal", math.pi * r**2, "og omkrets", math.
 ↪tau * r)
```

Til forskjell fra standardbiblioteket må math-biblioteket importeres, jfr. setningen import math

Hver funksjon fra math-biblioteket er også prefikset med ordet math. i kodeeksemplet over.

Hvis du ønsker å slippe det gjentatte prefikset math foran funksjonsnavn, kan alternativ import-setning brukes:

Alternativ import

[ ]:
```
from math import sqrt, sin, gcd, pi, e, tau
print("Kvadratroten til 8 er", sqrt(8) )
print("Sinus til 2 er", sin(2) )
print("Største felles faktor for 18 og 12 er", gcd(18, 12) )
print("Konstantene pi og e er lagret i Python som hhv.", pi, "og", e)
print("Har også konstanten tau som tilsvarer 2*pi:", tau)
print("En sirkel med radius 4 har areal", pi * 4**2, "og omkrets", tau * 4)
```

Et enda latere alternativ er å skrive **from math import *** i første linje over, * betyr da "alt mulig". Da slipper du å liste opp sqrt, sin, … Ulempen vil være at du i så fall importerer alt som fins i math-biblioteket, ikke bare de få funksjonene og konstantene du faktisk bruker. I profesjonelle programmer hvor hurtighet og ryddighet vil være viktig, er det derfor ikke anbefalt å bruke **from ... import *** , men det kan være greit i små eksempler.

1.0.6 c) Bruk av innebygde funksjoner og konstanter

Se på koden i blokken under. Legg til en passende import-setning øverst i programmet i stedet for # ??

Bytt ut alle forekomster av **None** med bruk av innebygde funksjoner og konstanter fra standard-biblioteket og math-biblioteket, slik at svarene blir riktige.

Dvs., der det nå står None skal det fylles inn en eller annen funksjon eller matematisk uttrykk tilsvarende det som allerede er gjort i print-setning nr 3.

Hvis du er usikker på hvilke funksjoner og konstanter det er lurt å bruke, ta først en kikk på Tutorial like foran deloppgaven.

[4]:
```
import math as m
# importerer fra math-biblioteket
```

```
print("|-8|, dvs. absoluttverdien til -8, er", abs(-8))
print(2.544, "avrundet til helt tall er", round(2.544))
print("Funksjonen int() derimot bare kutter vekk desimalene:", int(2.544) )
print(2.544, "avrundet til to desimaler er", round(2.544, 2))
print("Kvadratroten til", 10, "er", m.sqrt(10))
print("En sirkel med radius 7 har omkrets", m.pi * 7 * 2)
print("En sirkel med radius 7 har areal", m.pi * 7**2)
```

```
|-8|, dvs. absoluttverdien til -8, er 8
2.544 avrundet til helt tall er 3
Funksjonen int() derimot bare kutter vekk desimalene: 2
2.544 avrundet til to desimaler er 2.54
Kvadratroten til 10 er 3.1622776601683795
En sirkel med radius 7 har omkrets 43.982297150257104
En sirkel med radius 7 har areal 153.93804002589985
```

Utskrift til skjerm

```
|-8|, dvs. absoluttverdien til -8, er 8
2.544 avrundet til helt tall er 3
Funksjonen int() derimot bare kutter vekk desimalene: 2
2.544 avrundet til to desimaler er 2.54
Kvadratroten til 10 er 3.1622776601683795
En sirkel med radius 7 har omkrets 43.982297150257104
En sirkel med radius 7 har areal 153.93804002589985
```