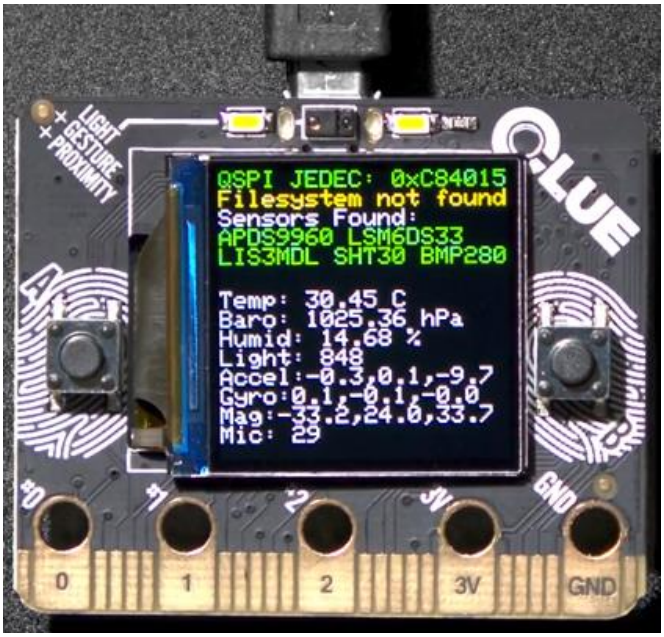
Python med Adafruit CLUE

Af: Michael Hansen, Coding Pirates Furesø, 2020, version 0.01
Dokumentet ligger på: http://www.rclab.dk/joomla/index.php/project/coding-pirates/python-and-microbit



Formålet er at lære simpel programmering i Python hvor man bruger et Adafruit CLUE board som platform. Selve dokumentet er baseret på 'learning by doing', dvs der er ikke meget Python undervisning i selve dokumentet. Jeg har lavet et præsentationssæt som gennemgår de vigtigste ting i Python og som også indeholder lidt generelt viden omkring embedded systemer.

Målgruppen er børn fra som ønsker at lære at programmer i Python.

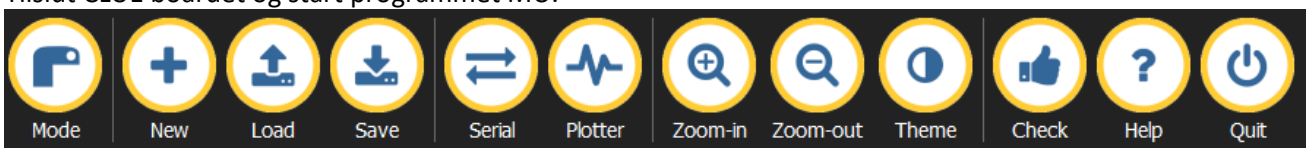
1 Introduktion

1.1 Installering

For at kunne programmere Adafruit CLUE'en (kaldet CLUE i dokumentet) skal man bruge en editor. Det letteste er at installere MU editoren, da den supporterer CLUE boardet direkte.

1.1.1 MU installering

Download MU fra denne hjemmeside: https://codewith.mu/en/download
Installer programmet – det tager nogle minutter.
Tilslut CLUE boardet og start programmet MU.



Start med at trykke på Mode knappen og vælg "Adafruit CircuitPython".
For at kunne bruge CLUE boardet i Python mode, skal man installere Python på det. Man kan se på denne hjemmeside hvordan man gør det https://learn.adafruit.com/adafruit-clue/circuitpython.

1.2 Hello world

Tryk Load (ikon i toppen af editoren) og åben programmet code.py. Slet alt indholdet!
Indtast følgende, brug "tab-completions". Husk Python kode er case-sensitiv (store og små bogstaver er forskellige!)

```
from adafruit_clue import clue

clue_data = clue.simple_text_display(title="Hello world",
title_scale=1, text_scale=2)

clue_data.show()

while True:
    pass
```

Tryk Save (ikon i toppen af editoren):
Gennemgang af kode.

- from — Denne linje skal altid være der i alle programmer man laver
- clue_data = — Opretter et display objekt
- clue_data.show() — Viser teksten på skærmen
- while True — loop for evigt (så teksten ikke forsvinder!)

Opgave

1) Ret teksten 'Hello world' til noget andet

1.3 Håndteringen af filer på CLUE

Ens filer bliver generelt gemt direkte på CLUE boardet og ikke på ens computer så hvis CLUE boardet bliver slettet eller får det dårligt, så mister man alle sine programmer. Det er derfor vigtigt at man også gemmer sine (store) programmer lokalt på sin PC.
Det gør man ved at trykke på New, rette filnavnet så det ligger på ens PC (spørg en voksen). Kopier derefter koden over i den nye fil og tryk Save. Nu har man en kopi af sit program. Det er vigtigt at gøre dette en gang imellem, specielt hvis andre skal låne kortet eller når man skal hjem.

1.4 Basic debug

Ved tryk på 'Check' knap, checker editoren om koden ser rigtig ud. Hvis noget er galt bliver det tydeligt markeret.
Kommentarer bruges til lettere at forstå programmet, men kan også bruges til at fjerne en linje midlertidig.

```
# Dette er en kommentar
```

Når programmet bliver Savet til boardet og dermed kører, så skriver det fejl ud hvis noget går galt. Det kan ses på CLUE displayet, men det er lidt småt og kan let scrolle ud. Hvis man trykker på Serial knappen, så kommer de vigtigste beskeder direkte ind i bunden af ens skærm.

2 Display tekst

Displayet har 240x240 pixels, med 0,0 i øverste venstre hjørne (hvilket er helt normalt).

2.1 Tekst, størrelse og placering

Indtast følgende kode:

```
from adafruit_clue import clue
import board
import displayio
import terminalio
from adafruit_display_text import label

display = board.DISPLAY

# Lav tekst label
tekst = "Hello world"
tekstLabel = label.Label(terminalio.FONT, text=tekst, color=0xFF00FF,
scale=2)
# Set placering
tekstLabel.x = 10
tekstLabel.y = 80

# Vis tekst label
display.show(tekstLabel)

while True:
    pass
```

Opgave

1. Ret teksten til at skrive noget andet
2. Ændre tekststørrelsen
3. Ret farven
4. Flyt teksten til et andet sted på skærmen

Hint

Man kan lave flere linjer ved at skrive

```
view.append(tekstLabel)     # Første linje
view.append(tekstLabel2)    # Endnu en linje
display.show(view)          # Vis alle linjer
```

Der er to måder at slette linjerne igen:

```
view.remove(tekstLabel)     # Fjern specifik linje
view.pop()                  # Fjern sidste linje element
                            # Kan bruges til at slette alle linjer med
```

2.2 Knapper

Man aflæser knapperne med følgende linje:

```
if clue.button_a:           # Tast a
    # tast trykket
```

Opgave

1. Brug tasterne A og B til at flytte teksten op og ned med

Hint

Man får brug for at lave en variabel til at holde den aktuelle position med. Det gøres på følgende måde:

```
posy = 10          # Erklær variablen
...
posy = posy + 10   # Læg 10 til variablen
```

Gem filen som **tekst_knap.py**.

2.3 Gestures

Gestures er håndbevægelser. Boardet har en gesture sensor i toppen lige over displayet, som kan genkende følgende bevægelser: op, ned, venstre og højre.
Man læser en gesture med følgende linje:

```
value = clue.gesture
```

hvor value er 0-4 alt afhængig af bevægelsen.

Opgave

1. Lav et program som skriver gesturen ud på skærmen
2. Lav det så værdien står på skærmen i 1 sekund
3. Flyt teksten ved hjælp af gestures. Hint: brug to variable.
4. Bonus: Skriv en tekst (op, ned, …) ud i stedet for tallet. Hint: brug en liste!

Hint

Man kan skrive en værdi ud på følgende måde:

```
tekst = ”Tal: %i” %value    # Giver fx ”Tal: 4”
```

Man kan lave en pause med følgende linjer:

```
import time
...
time.sleep(1)     # Pause i sekunder (kan være kommatal)
```

Gem filen som **gesture.py**.

2.4 Reaktion test

Lav et lille spil, hvor man kan måle hvor hurtigt man kan trykke på tasterne. Til at måle tiden med kan man bruge følgende funktion:

```
t = time.monotonic()        # Tid i us (micro sekunder)
```

Vis en tekst, vent på at man trykker på en tast og mål hvor lang tid der gik. Vis resultatet på skærmen.

3 Display og grafik

Displayet kan også lave grafik. Det er beskrevet på følgende side:
https://circuitpython.readthedocs.io/projects/display-shapes/en/latest/api.html

3.1 Linjer

Tast følgende kode ind:

```
import board
import displayio
from adafruit_display_shapes.rect import Rect
from adafruit_display_shapes.circle import Circle
from adafruit_display_shapes.line import Line

display = board.DISPLAY

group = displayio.Group(max_size=25)
display.show(group)

line = Line(100, 100, 120, 200, color=0xffff00)
group.append(line)

while True:
    pass
```

Parametrene er:

Line(x0, y0, x1, y1, color)

Opgave

1. Tegn en kop ved brug af linjer

3.2 Firkant

Man tegner en firkant med følgende linjer:

```
rect = Rect(0, 0, 80, 40, fill=0x00FF00)
group.append(rect)
```

Parametrene er:

Rect(x0, y0, width, height, fill)

Opgave

1. Fyld vand i koppen i ca. 10 steps.

Hint

Man laver en loop på følgende måde:

```
for i in range(10):         # Laver en loop for i=0-9.
    ...
```

Gem filen som **fyld_kop.py.**

3.3 Accelerometer

Man læser acceleration måleren på følgende måde:

```
(dx, dy, dz) = clue.acceleration
```

Man fjerne et objekt på følgende måde:

```
group.remove(circle)
```

Opgave

1. Tegn en cirkel (eller firkant) og få den til at flytte sig rundt på skærmen ved at tippe skærmen.

Gem filen som **acc_prik.py**.

3.4 Kompas pil

Man aflæser kompasset på følgende måde:
TBD

Opgave

1. Lav en kompaspil som peger mod nord.

4 Led og NeoPixel

CLUE har flere forskellige LEDs. Øverst sidder der to kraftige hvide LEDs. De tændes på følgende måde (de er ret kraftige, så kik ikke direkte ind i dem):

```
clue.white_leds = True
```

Bag på sidder der en rød LED. Denne tændes på følgende måde:

```
clue.red_led = True
```

Bag på sidder også en neopixel LED, som også bruges under download af koden. Den styres på følgende måde:

```
clue.pixel.fill((255, 0, 255))        # OBS dobbelt parentes
```

Opgave

1. Lav et lille program som blinker med de forskellige LEDs

Gem filen som **led_test.py**.

5 Porte

For at kunne kommunikere med omverdenen bruger man porte også kaldet GPIO. De er i den konnektor som er i bunden af printet. De kan bruges til at aflæse eksterne sensorer og styre eksterne ting som et relæ.

```
led = digitalio.DigitalInOut(board.D0)          # Ben 0
led.direction = digitalio.Direction.OUTPUT      # Port er output
led.value = True                                # Tænd LED
```

Opgave

1. Tænd en ekstern LED og få den til at blinke
2. Aflæs en ekstern kontakt og tænd LED'en. Obs: kontakt skal være INPUT med pull-down, se hint.

Hint

```
kontakt.pull = digitalio.Pull.DOWN
```

Gem filen som **knap_led.py**.

5.1 Afstandssensor

Ultra sonic sensoren virker ved at sende en ping ud og måle hvor lang tid det taget den at komme tilbage igen. Kunsten er at sende en ping, starte stoppeuret, vente på at pulsen kommer tilbage og så aflæse uret. Det går ret stærkt, da lyden bevæger sig med ca. 340m/s! Man kan bruge følgende funktion til at måle tiden med:

```
t = time.monotonic_ns()     # tid i ns (nano sekunder)
```

Sensoren har to ben. TRIG som bruges til at sender en puls med, og ECHO som måler når pulsen kommer tilbage. Man skal vente på at ECHO går høj og så måle hvor langt tid den er høj. Det svarer til hvor langt tid det har taget pulsen at komme fra senderen til modtageren.

Opgave (svær)

1. Sæt de to porte op (GPIO) så de vender den 'rigtige' vej, dvs. output og input.
2. Få ultra sonic sensoren til at sende en ping og mål hvor lang tid ECHO er høj.
3. Omregn tiden til afstanden i cm – check at det passer!
4. Lav en funktion som returner afstanden i cm

Hint

Man laver en funktion på følgende måde:

```
def SonarAfstand():
   <kode>
   return afstand
```

6 Sensorer

TBD

7 Appendiks – ekstra opgaver

Her er nogle ekstra opgaver som er lidt sværere, da der ikke er så meget hjælp til dem.

8 Appendiks – info

Dette afsnit er en blanding af ekstra information, løsninger og ideer til fremtidige programmer.

8.1 Løsninger

Her er løsningsforslag til de fleste af opgaverne.

tekst_knap.py

```
from adafruit_clue import clue
import board
import displayio
import terminalio
from adafruit_display_text import label
import time

display = board.DISPLAY

posy = 100
while True:
    if clue.button_a and posy < 230:
        posy += 10
    if clue.button_b and posy > 10:
        posy -= 10

    # Lav tekst label
    tekst = "Hello world"
    tekstLabel = label.Label(terminalio.FONT, text=tekst,
color=0xFF00FF, scale=2)
    # Set placering
    tekstLabel.x = 40
    tekstLabel.y = posy

    # Vis tekst label
    display.show(tekstLabel)
    time.sleep(0.2)
```

gesture.py

```
from adafruit_clue import clue
import board
import displayio
import terminalio
from adafruit_display_text import label
import time

display = board.DISPLAY

gestureText = ("", "Op", "Ned", "Venstre", "Hojre")

posx = 40
posy = 100

while True:
    value = clue.gesture
    if value == 1 and posy > 10:
        posy -= 10
    if value == 2 and posy < 220:
        posy += 10
    if value == 3 and posx > 10:
        posx -= 10
    if value == 4 and posx < 220:
        posx += 10
    # Lav tekst label
    tekst = "%i: %s" %(value, gestureText[value])
    tekstLabel = label.Label(terminalio.FONT, text=tekst,
color=0xFF00FF, scale=2)
    # Set placering
    tekstLabel.x = posx
    tekstLabel.y = posy

    # Vis tekst label
    display.show(tekstLabel)
    if value > 0:
        time.sleep(1)
```

fyld_kop.py

```
import board
import displayio
from adafruit_display_shapes.rect import Rect
from adafruit_display_shapes.circle import Circle
from adafruit_display_shapes.line import Line
import time

display = board.DISPLAY

group = displayio.Group(max_size=25)
display.show(group)

color = 0xffffff
line = Line(50, 100, 70, 200, color=color)
group.append(line)
line = Line(70, 200, 150, 200, color=color)
group.append(line)
line = Line(150, 200, 170, 100, color=color)
group.append(line)

for i in range(9):
    y = 10*i
    x = 2*i
    rect = Rect(70-x, 189-y, 80+2*x, 10, fill=0x0000ff)
    group.append(rect)
    time.sleep(0.3)

while True:
    pass
```

acc_prik.py

```
from adafruit_clue import clue
import board
import displayio
from adafruit_display_shapes.rect import Rect
from adafruit_display_shapes.circle import Circle
from adafruit_display_shapes.line import Line
import time

display = board.DISPLAY

group = displayio.Group(max_size=25)
display.show(group)

color = 0xffffff
posx = 100
posy = 100
circle = Circle(posx, posy, 20, fill=color)
group.append(circle)

while True:
    (dx, dy, dz) = clue.acceleration
    posx += dx
    posy += dy
    circle2 = Circle(int(posx), int(posy), 20, fill=color)
    group.append(circle2)
    group.remove(circle)
    circle = circle2
    time.sleep(0.01)
```

led_test.py

```
from adafruit_clue import clue
import time
import random

for i in range(10):
    clue.white_leds = True
    time.sleep(0.05)
    clue.white_leds = False
    time.sleep(0.05)

clue.red_led = True
time.sleep(1)
clue.red_led = False

for i in range(10):
    clue.pixel.fill((random.randrange(10, 100, 1),
random.randrange(10, 100, 1), random.randrange(10, 100, 1)))
    time.sleep(0.5)

while True:
    pass
```

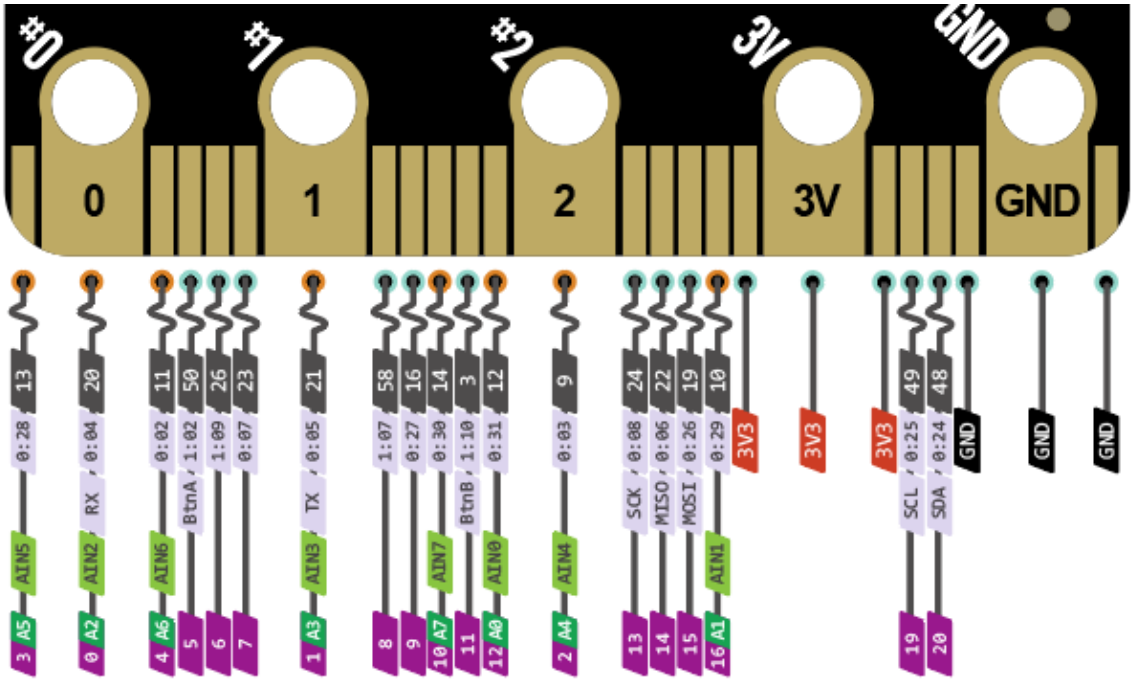
knap_led.py

```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.D0)
led.direction = digitalio.Direction.OUTPUT

key = digitalio.DigitalInOut(board.D1)
key.direction = digitalio.Direction.INPUT
key.pull = digitalio.Pull.DOWN

while True:
    led.value = key.value
    time.sleep(0.1)
```

8.2 Debug tips

Her er lidt tip til at debugge sin kode med.
Hvis man bruger MU editor så er det en god ide at bruge Check knappen til at se om koden ser lovlig ud. Det er dog ikke nok til at sikre at koden er korrekt, men et godt sted at starte.
Hvis CLUE'en skriver en fejlbesked ud på skærmen, så kan man med fordel trykke på Serial knappen for så vil fejlbeskeden komme ud på MU editor vinduet incl. linje nummer og forklaring. Man kan også printe variable ud for bedre at forstå hvad programmet gør.

8.3 CLUE pin-out



8.4 Ideer til videre forløb

8.4.1 Projekter

Andre ideer til projekter.

- Mål magnet felt
- Mål fugtighed på pin0 (brug fingrene)

- LED UV meter, graf!
- Eksterne porte
    - Servo
- Plotter

9 Python reference

Her er en kort oversigt over nogle af de mest brugte Python konstruktioner.

9.1 Generelle

9.1.1 Kommentarer

Kommentarer bruges til at skrive tekster, så det er lettere at forstå koden. De kan også bruges til at slå noget kode fra pga. test.

```
# Dette er en kommentar

""" Start af kommentar

slut på kommentar"""
```

9.1.2 Import

Import bruges til at importere kode fra andre moduler. Al CLUE kode skal starte med at importere CLUE modulet, men vi bruger også andre moduler. `from xxx import yyy`, betyder at man kun importere dele af modulet.

```
from adafruit_clue import clue
from utime import ticks_us, sleep_us
```

9.1.3 Print

Print bruges til at skrive ud på serielporten med. For CLUE bruges dette primært til at debugge sin kode med.

```
print(x)          # Print værdien af x
```

9.1.4 Pass

`pass` gør ingenting, og bruges derfor sjældent, men kan være rar at have alligevel.

```
for i in range(2):
    pass                    # Gør ingenting
```

9.2 Datatyper

Datatyper bruges til at definer ens variable med. Python har flere forskellige datatyper, som bruges til forskellige formål. Det er typisk afgørende at vælge den rigtige datatype for at få ens kode til at blive så simpel som mulig. Det vil være god praksis at begynder variable navne med små bogstaver!

9.2.1 Simple typer

`<navn> = <værdi>`, opret en simpel variabel.

```
x = 1             # Sær x lig med 1
a = 'hej'         # Lav en streng
x += 1            # Læg 1 til x
```

9.2.2 Tupler

Tupler er en ordnet mængde af elementer. Antallet af elementer kan ikke ændres og det samme gælder for værdierne. De defineres med bøjet parenteser `()`. Indeks starter fra 0.
`<navn> = (<elementer>)`

```
t = (2, 4)          # Lav tuple med 2 elementer
print(t[0])         # Print første element
```

9.2.3 Lister

Dette er en list af elementer, hvor antallet af elementer kan ændres over tid dvs. man kan indsætte og fjerne elementer. Værdierne kan også ændres. De defineres med skarpe parenteser `[]`. Indeks starter fra 0.

`<navn> = [<elementer>]`

Operatorer (der findes mange flere):
`<list>.append(<element>)`, tilfæj et element til enden af listen
`<list>.remove(<element>)`, fjern et element fra listen
`len(<list>)`, find antallet af elementer

```
l = []              # Lav en tom liste
l = [1, 2, 6]       # Lav en liste med 3 elementer
l = range(3)        # Lav liste med 3 elementer 0, 1, 2
l.append(23)        # Tilføj 23 til enden af listen
l.remove(6)         # Fjern 6 fra listen
print(len(l))       # Print antallet af elementer
print(l[0])         # Print første element
l[1:2]              # Udtag liste med elemet 1
```

9.3 Kontrol

Disse konstruktioner bruges til at styre program flowet med.

9.3.1 If/else

Hvis 'betingelse' er sand gør A ellers B. `else` grenen er frivillig.

if <betingelse>:
  <indrykket kode>
else:
  <indrykket kode>

```
if x > 3:
    print(”hej”)
else:
    print(”dav”)
```

9.3.2 Try/except

Dette er en speciel konstruktion som bruges til at fange hvis noget går galt, dvs man kan undgå at ens kode 'dør'.

try:
  <kode som kan dø>
except:
  <hvis koden døede, så gør det her>

```
try:
    i2c.read(0x1C, 1)[0]
except:
    print(”fejl”)
```

9.4 Loops

Loops bruges til at gentage den samme kode flere gange.

9.4.1 While

Så længe ’betingelse’ er sand, så forsæt løkken.
while <betingelse>:
  <indrykket kode>

```
while True:                 # Loop for evigt
    print(”hej”)
```

9.4.2 For

Dette er en loop for et bestemt antal iterationer.
for <variable> in <list>:
  <indrykket kode>

```
for i in range(5):          # Loop fra 0 til 4
    print(”hej”)
```

9.4.3 Break, continue

Hvis man gerne vil forlade en loop ’før’ tid, så kan man bruge følgende:
`break` exit loop nu

```
for i in range(5):          # Loop fra 0 til 4
    if i > 3:
        break               # Forlad loop’en før tid
    print(”hej”)
```

`continue` fortsæt med ’næste’ loop, dvs spring resten af loop’en over.

```
for i in range(5):          # Loop fra 0 til 4
    print(”dav”)
    if i > 3:
        continue            # Undlad print(“hej”) hvis i > 3
    print(”hej”)
```

9.5 Funktioner

Funktioner bruges til at opdele sin kode i genbrugelige blokke, som så kan kaldes fra flere forskellige steder. Det vil være god praksis at begynde funktionsnavne med Store bogstaver. Funktioner kan returnere alt fra ingenting til en liste af argumenter.

def <navn>(parameter, …, parameter=<værdi>):
  <indrykket kode>
  return <værdi>

```
def PlusTal(a, b, c=0):     # Definer funktion, c er valgfri
    return a+b+c

x = PlusTal(1, 2, 3)        # Angiv alle parametre
y = PlusTal(3,6)            # Angiv kun 2 værdier, c bliver så 0

def PlusFlereTal(a, b, c):
        Return a+b, a+c     # Her returneres to resultater

x, y = PlusFlereTal(1, 4, 10)
```