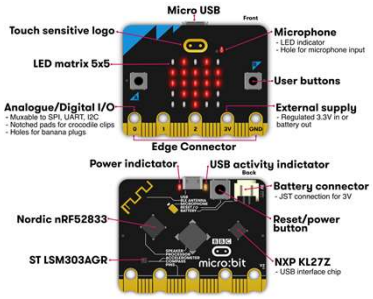
LYD-Kit Spor 1 (LKS1)

Byg med sensorer

Modul a – ”Værkstedskørekort”

Micro:Pyton og indbyggede sensorer



Projekt støttet af:

midt regionmidtjylland

```python
# 3c - lys til lyd.py
# Konvertering af lys intensitet til lyd (pitch) Version 1.1 10-Nov-2020
# Importer de nødvendige biblioteker/moduler.
# Den generelle microbit for at nå display.read_light_level() fundktionen
# og music for at nå music.pitch() funktionen
from microbit import *
import music

# Lys niveau kan have værdier mellem 0 og 255
# pitch kan håndtere værdier mellem 0 og 3906Hz. Vi kan høre fra ca. 50Hz.
# - derfor lys niveau*14+50 som 'fornuftig' frekvens til pitch funktionen
while True:
    x = display.read_light_level()
    music.pitch(x*14+50)
    print(x)
    sleep(100)
```

Til lærer som info/inspiration.

Kan bruges sammen med opslagsværkerne nedenfor som en introduktion til programmering i Python/MicroPython (på Micro:Bit)
Python Tutorial (w3schools.com) og
BBC micro:bit MicroPython documentation — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)

(Klikbare links på slides sidst i stakken – De er desværre ikke klikbare når de er i noterne! – der er referencer til disse med links i slides undervejs.)

Kan findes ved at søge i google på følgende:
”w3schools python tutorial” og
”micropython microbit”

Og det er en god ide at vise de 2 opslagsværker live ifm undervisningen.

Dækker Spor 1 i LYD-Kit materialet.

Slide stak er tænkt som inspiration til lærer – ikke som et færdigt forløb overfor eleverne.

Version 04-Feb-2022:
Rettet/tilføjet sådan at det kan bruges som 3 forløb af kortere varighed.

Modul a:
Det basale og nødvendige for at kunne bruge materialerne i alle 4 spor i LYD-Kit med en basal forståelse for hvad der sker i koden – 'Værkstedskørekort'

Modul b:
Et værkstedsmodul for dem som vil vide mere/gå dybere i Microbitten for dem som vil bygge selv med Micro:Bitten, og grundlag for at kunne kikke ind i den færdige kode til Micro:Bit som der udleveres i Spor 2.

Modul c:
En case, hvor man med udgangspunkt i færdig kode kan lave funktions kritik, teknologivurdering og produktdesign.

Version 23-Nov-2020:
Lagt op efter test med 1x og 2x på SSG. Der blev det kørt som et samlet forløb på 10 lektioner.

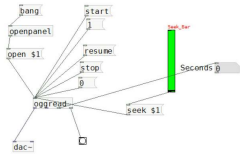
BEMÆRK at LYD-Kit er organiseret sådan at man IKKE behøver at komme dybt i programmeringen, da der er en række færdige programmer og skabeloner, som man kan bygge ovenpå; både uden at ændre i kode og ved at lave simple modifikationer.

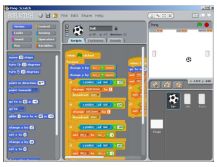
Man skal dog have en basis forståelse for Python og et såkaldt udviklingsmiljø (IDE – Integrated Development Environment) for at kunne benytte de færdige programmer. Det er det vi kalder 'Lukket bog' tilgangen. Det er omvendt også muligt at bruge det som 'åben bog' til at gå dybere i programmeringsdelen, hvis man vil.

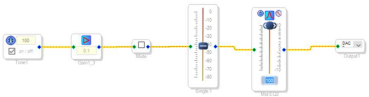
Derved understøttes den didaktiske tilgang – Faded Guidance, og UMC (Use Modify Create) kendt fra informatik og programmerings fagene, samt i tilgangen fra Computational Thinking (CT)

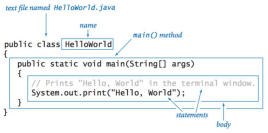
Spor 1 Modul a: Programmeringssprog

Projekt støttet af:









Som introduktion til programmeringssprog og udviklingsmiljøer (IDE)

Der er mange slags programmeringssprog. Visuelle, blokbaserede og tekst baserede. Hør hvad eleverne har stødt på. De fleste har nok stødt på Blokbaseret programmering – evt. i form af SCRATCH, men måske også MakeCode til Micro:Bit.

De tekstbaserede bruges til professionel udvikling af programmer i forskellige sammenhænge. Der er også mange forskellige slags tekstbaserede sprog, der har forskellige styrker.
Der findes på nye og de udvikler sig over tid. Så hvis man vil arbejde med programmering skal man være forberedt på at skulle kunne flere sprog og at de ændrer sig undervejs.

Python:

Python er gennemgående for alle spor i LYD-Kit.
Det anses for at være en fordel for begynderen, at man kan bruge det samme grundlag både på en mikrocontroller som Microbitten (men også andre) såvel som på en computer som PC/MAC samt Raspberry PI.

LYD-Kit bruger Python fordi det er et sprog der er let at komme i gang med, men som har rige muligheder for at skalere til mange forskellige anvendelser, fra styring af Micro

kontrollere til Datascience og Machine learning. Det er et sprog der er populært i tiden og har været i vækst over en længere periode.

Man vil kunne få glæde af, og møde Python i forbindelse med mange videregående uddannelser indenfor STEM fagene.

Klikbar link på billedet i midten, som reference til hvor oversigt kommer fra.

Udviklingsmiljø (IDE):

Der er tilsvarende mange forskellige udviklingsmiljøer (IDE – Integrated Development Environment) man kan bruge til at skrive Python programmer i.

Da LYD-KIT retter sig mod begynder og på at kunne bruges uden den store programmeringsindsats er Thonny valgt. Thonny er et enkelt, let tilgængeligt og gratis IDE som fungerer på både PC/MAC og Raspberry PI. Det betyder ikke at man ikke kan bruge andet. Det er bare det som LYD-kit er testet af med. Thonny er også på udviklingstidspunktet default IDE som kommer sammen med Raspberry OS til Raspberry PI.

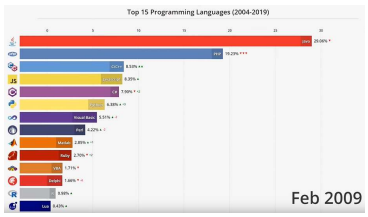
I det følgende er det forudsat at Thonny er installeret på elevernes maskiner og at diverse demoer gennemgås med Thonny. Igen så vil det kunne laves med andre IDE's.

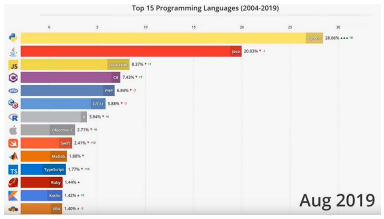
Andre populære IDE's er for eksempel:
- Pycharm
- Visual code

Spor 1 Modul a: Programmeringssprog

Projekt støttet af:

Statistikker og lignende optællinger har altid nogle forudsætninger så man kan finde forskellige bud på ex. hvad der er mest udbredt så vær varsom. Men de 2 der er i denne præsentation indikerer at Python er udbredt og er blevet mere og mere anvendt over tid.





Worldwide, Jan 2022 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	28.74 %	-1.8 %
2		Java	18.01 %	+1.2 %
3		JavaScript	9.07 %	+0.6 %
4	↑	C/C++	7.4 %	+1.1 %
5	↓	C#	7.27 %	+0.7 %
6		PHP	6.06 %	+0.0 %
7		R	4.19 %	+0.3 %
8		Objective-C	2.27 %	-1.4 %
9		Swift	1.91 %	-0.2 %
10		TypeScript	1.74 %	-0.0 %
11		Matlab	1.74 %	+0.0 %

https://www.youtube.com/watch?v=yL704C1PI4o

Endnu et eksempel til at underbygge historien om at Python er i vækst og benyttes af mange.

Klikbare links på figurerne – hvis man vil uddybe – vise en udvikling over tid.
Det kan også gives som lektier eller baggrundsmateriale, som eleverne selv kan kikke i.

Spor 1 Modul a: Basis strukturer

Projekt støttet af:

```
while <betingelse>:
        <gør dit>
        <gør dat>
```

```
if <betingelse 1>:
        <gør dit når betingelse 1 er opfyldt>
        <gør dat når betingelse 1 er opfyldt>
elif <betingelse 2>:
        <gør noget andet dit når betingelse 2 er opfyldt>
        < gør noget tredje når betingelse 2 er opfyldt>
 else:
        <gør noget femte når hverken betingelse 1 eller 2 er opfyldt>
```

The while Loop

With the `while` loop we can execute a set of statements as long as a condition is true.

Python While Loops (w3schools.com)

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

Python Conditions (w3schools.com)

HUSK ":" og indrykning er afgørende for at fortælle fortolkeren hvilken kode stump der hører til i løkker og udføres på baggrund af en betingelse! Det er en del af sprogets 'syntax'
Python Syntax (w3schools.com)

Andre programmeringssprog benytter andre former for at afgrænse kode stumper. Ex. Do .. End, Begin .. End, og { …. }.

Kan bruges til en overordnet snak om syntaks inden der laves kodning, og som introduktion til hvor man kan finde mere information.

Slå gerne op på nettet på referencer og vis det som del af gennemgang.
("w3schools Python Tutorial" – Kan bruges som søgeord til google, så kommer man direkte dertil).

Der er klikbare links i slide direkte til relevante steder i reference opslagsværk.
De findes også samlet sidst i slide stak, hvis man ønsker at dele dem på anden vis.

Efter at have vist slide(s) med struktur bør man tage en runde hvor man viser programmeringen i Thonny, og kan så bruge de efterfølgende slides (med programeksempler) som opsamling/hand-outs til en opsamlende snak/repetition.

Vis at eksempler kan eksekveres både på egen PC/MAC såvel som på Microbitten. Først gennemgå koden – byg den op og vis den bliver eksekveret på PC

Tilslut Micro:bit på USB og installer Python på den – hvis det ikke allerede er gjort se: "notat om at komme i gang med Thonny.pdf" under installationsvejledninger under folderen generel i Lyd-Kit.
HUSK – Der skal være USB kabler med det rigtige stik til elevernes PC'er – Det kabel der

følger med Micro:Bit har et USB-A, men visse MAC's har kun usb-c – så der skal man have en adapter eller et andet kabel.

Gør en del ud af at få forklaret at der er en væsentlig forskel – på 2 vidt forskellige maskiner! (processoren i hhv. PC/MAC og på Microbitten). ←

Men altså at resultatet er det samme, for de eksempler der gennemgås! De 2 makiner kan finde ud af at lave det samme grundlæggende!

Spor 1 Modul a: Basis strukturer

Projekt støttet af:

midt regionmidtjylland

<listenavn> = (<liste element 1>,<liste element 2>,…)

List

Lists are used to store multiple items in a single variable.

Python Lists (w3schools.com)

for <elementer i en sekvens>:
    <gør dit>
    <gør dat>

Python For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

Python For Loops (w3schools.com)

def <funktions navn>(<evt. parametre>):
    <gør dit>
    <gør dat >
    <og mere til, som man kan udnytte flere steder/gange>

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

Python Functions (w3schools.com)

KAN udelades, da det kun er nødvendigt med grund forståelsen for while-løkken og if-betingelsen til eksemplerne i modul a.

Men er en god 'overbygning' til den foregående slide, hvis man vil lidt mere.

Igen bør man viser programmeringen i Thonny, og kan så bruge de slides med program som opsamling/handouts efter at man har vist eksempel.

Spor 1 Modul a: Et simpelt program – betingelse og løkke

Projekt støttet af: midt regionmidtjylland

```
1  # En variabel med navnet 'a' som initielt får tildelt værdi nul (tallet 0)
2  a = 0
3  # En while løkke der gentages sålænge betingelsen (a<=5) "a er mindre eller lig med 5" er opfyldt
4  while a<=5:
5      # udfører en handling/en funktion der udskriver værdien af variablen a i shell
6      print(a)
7      # en betinget sætning, der tester om a er større elelr mindre end 3 og udskriver det i shell
8      if a<3:
9          print("a mindre end 3")
10     elif a>3:
11         print("a større end 3")
12     else:
13         print("a er lig med 3")
14     # der lægges 1 til værdien af variablen a
15     a = a+1
16 print("while løkken er nu slut - Bemærk at koden både kan køre på PC/MAC og på Micro:Bitten")
```

HUSK – tekstbaseret programmering kræver at man skriver det rigtige. Ellers forstår fortolkeren det ikke og man får en syntaks fejl. Der er også forskel på store og små bogstaver.

Men prøv bare der er ikke noget der går i stykker og man får en fejlmelding i shell som man kan bruge til at få løst fejlen.

Ambitionen er IKKE at eleverne forstår alt her – men at de kort introduceres til begreberne og syntaks for de væsentligste strukturer.
Og at de får mulighed for selv at komme videre vha. opslagsværker og tutorials, hvis de ønsker.

De skal bare her få muligheden for at kunne læse færdige programmer, og få en basal fornemmelse for hvad simple programmer gør.

Som lærer kan man vurdere hvor meget denne del skal fylde og også hvor dybt man går. Der er en risiko for at man får givet for meget information og at man taber nogen.

Det kan være bedre at komme hurtigt til eksemplerne med Micro:Bitten og få nogle oplevelser med noget der sker/virker.
Har forsøgt med at kun benytte niveau fra slide 4/6 og så gå derfra til slide 10 eksemplet for at komme hurtigt til et hands-on resultat med microbitten.

Hands-on kommer efterfølgende med fokus på Micro:Bitten.

HUSK igen at modul a primært er tænkt til at eleverne efterfølgende vil kunne bruge færdig kode til forløb i spor 2.

Kode eksempel findes under LKS1>Ma>LKS1Ma-Slide6.py.
De kan udleveres til eleverne efter at man har lavet det i Thonny og vist funktionen. De kan så selv læse kommentarerne og prøve at køre det.

Spor 1 Modul a: Et simpelt program – liste, funktion og for løkke

Projekt støttet af:
midt regionmidtjylland

```
# Liste med navne som tekst strenge
# Index    0      1       2     3      4     - Der er 5 navne i listen - også kaldet længden af listen
navne =['Jens','Peter','Ole','Anne','Sofie']

# Definition af en funktion der bliver døbt til at hedde "hmd".
# Funktion tager et navn som argument og udskriver 4 liniers velkomst i shell
def hmd(n):
    print ('Hej med dig')
    print (n)
    print("Godt at se dig")    # bemærk at både ' og " kan bruges omkring en tekst.
    print()

# En for løkke der for alle navne i listen navne kalder funktionen hmd
for n in (navne):
    hmd(n)

# Det samme i modsat rækkefølge vha indexering i listen - man udpeger selv index i listen med navne
print("Længden af listen er: "+str(len(navne)))   # Kan sammensætte tekster med + str() funktionen laver tal om til tekst.
for i in range(len(navne)):
    hmd(navne[len(navne)-i-1])                    # Der er 5 navne, men index går fra 0 til 4.
```

KAN udelades, da den ikke er nødvendig for at komme i gang med MicroPython og Micro:Bitten.

I resten af forløb a med Micro:Bit er der for eksempel ikke lagt op til noget hvor man selv skal lave funktioner.

Det kan evt. bruges som en option til dem der vil vide mere. Passer sammen med spor 2 eksempler, hvor funktioner og lister er brugt i cases.

Kode eksempel findes under LKS1>Ma>LKS1Ma-Slide7.py.
De kan udleveres til eleverne efter at man har lavet det i Thonny og vist funktionen. De kan så selv læse kommentarerne og prøve at køre det – evt. som hjemmeopgave.

Spor 1 Modul a: Introduktion til biblioteker/moduler

Projekt støttet af:

```python
# import af bibliotek/modul med random funktionaliteter.
import random
# Liste med navne som tekst strenge
# Index    0      1      2     3      4     - Der er 5 navne i listen - også kaldet længden af listen
navne =['Jens','Peter','Ole','Anne','Sofie']

# Definition af en funktion der bliver døbt til at hedde "hmd".
# Funktion tager et navn som argument og udskriver 4 liniers velkomst i shell
def hmd(n):
    print ('Hej med dig')
    print (n)
    print("Godt at se dig")    # bemærk at både ' og " kan bruges omkring en tekst.
    print()

# Et tilfældigt udvalgt navn fra listen med navne vha importeret random modul/bibliotek
hmd(random.choice(navne))      # virker både i Python og MicroPython
```

Python Modules (w3schools.com)

Python Random Module (w3schools.com)

Random Number Generation — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)

```python
# import af bibliotek/modul med random funktionaliteter.
import random

# Definition af funktion der sammenligner parameteren i med konstanten 3,
# og udskriver om i er mindre end, større end eller lig med 3
def lms(i):
    if i<3:
        print(str(i)+" er mindre end 3")  # Kan sammensætte tekster med + str() funktionen laver tal om til tekst.
    elif i>3:
        print(str(i)+" er større end 3")
    else:
        print(str(i)+" er lig med 3")

# metoden/funktionen randrange() fra modulet/biblioteket random finder et tal mellem 0 og 7
# returnerer det og gemmer det i variablen i.
# Efterfølgenden kaldes funktionen lms med i som argument.
i=random.randrange(7)                          # virker både i Python og MicroPython
lms(i)
```

Kan udelades og man kan springe direkte til microbitten og biblioteket Micro:bit for ikke at skabe 'overload' eftersom fokus er på at give eleverne en fornemmelse for sensorerne i Microbitten, og for at kunne loade og bruge færdige kode eksempler til Micro:bitten i Spor 2.

Funktionalitet af basis Python udvides vha. biblioteker (også kaldet moduler) som kan importeres og benyttes i programmerne.
Mange biblioteker virker til flere maskiner og operativsystemer (OC/Windows, MAC/MACos, Raspberry PI/Rapsberry OS/Linux)

Og nogle få er helt generelle og kan også benyttes i MicroPython – Ex. random –MEN ikke nødvendigvis med fuld funktionalitet!

De 2 eksempler i slide kan køre både på PC/MAC og på Micro:bit.

MEN bemærk at der er flere funktioner i random som ikke er med i MicroPython.
Vis det evt. ved at bruge dir(random) funktion i shell både med Python og med MicroPython som valgt fortolker.

Kan også vises ved at åbne dokumentationen for Python (W3schools linken) hhv. MicroPython for Micro:bit. (micro:bit MicroPython linken)

Der er biblioteker der er koblet direkte til den maskine som man kører programmerne på. Det er tilfældet for de specifikke sensorer som findes på Micro:Bit.

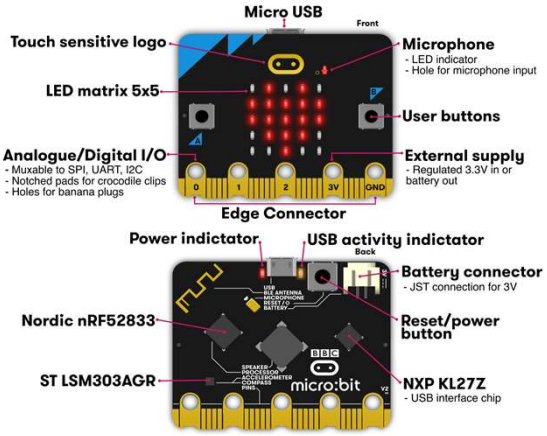
De tilgås gennem et bibliotek der hedder Micro:bit, som bruges i det følgende, hvor programmerne bliver eksekveret/udført på Micro:Bitten.

Kode eksemplerne findes under LKS1>Ma>LKS1Ma-Slide8-ex1.py og ..-ex2.py

De kan udleveres til eleverne efter at man har lavet det i Thonny og vist funktionen. De kan så selv læse kommentarerne og prøve at køre det – evt. som hjemmeopgave.
De kan så også få en simpel opgave som at udskifte navnene i liste – lave den længere etc.

Spor 1 Modul a: Micro:Bit v2

Projekt støttet af:
midt regionmidtjylland



En kort introduktion til Micro:Bitten i version 2, og dens indbyggede sensorer.

- Accelerometer
- Magnetometer (compass)
- Lyssensor
- Mikrofon
- Påmonterede knapper (A,B), samt logo m touch.

- Påmonteret højtaler til at lave lyde
- 5*5 LED's som display.

Version 1 havde ikke monteret mikrofon og højtaler, og logo kunne ikke bruges som knap.

HW reference:
[Hardware (microbit.org)]

Spor 1 Modul a: Brug Micro:bit bibliotek

Projekt støttet af: midt regionmidtjylland

```
1  from microbit import *
2
3  while True:
4      if button_a.is_pressed():
5          display.show(Image.HAPPY)
6      elif button_b.is_pressed():
7          break
8      else:
9          display.show(Image.SAD)
10
11 display.clear()
```

Microbit biblioteket indeholder bl.a. funktioner (også kaldet metoder) der eksempelvis kan fortælle om knap a og/eller knap b er blevet trykket ned/aktiveret, og til at udskrive ting på displayet.

Bemærk at "button_a.is_pressed()" er en funktion der returnerer en værdi som er sand eller falsk (True eller False (None)) som så kan bruges som betingelse i en if … sætning.

Der er en "break" kommando i python generelt, som kan bruges til at komme ud af en igangværende løkke – Bemærk at betingelsen i while løkken er en konstant True, som jo aldrig vil blive ændret – så hvis ikke break blev brugt ville den aldrig stoppe og komme til den kode som ligger efter while løkken

Prøv med andre billeder/evt. en tekst.
Prøv med display.scroll("En tekst") og se hvad der sker.
Se/Find flere eksempler i MicroBit/MicroPython dokumentationen.

Buttons — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)

Vis dokumentationen for Micropython på Microbit. Vis overordnet hvordan den er struktureret.

- En tutorial del
- En API del med flere detaljer i beskrivelsen
- Et søge felt.

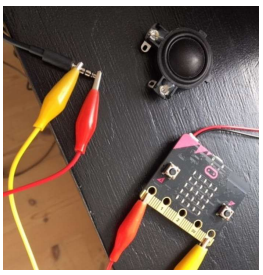
Gå så til siden med buttons – scroll ned til eksemplet som er i slide.
Kopier det ind i Thonny fra dokumentationen ind i editor
Skift fortolker til MicroPython på Micro:Bit (se notat om at komme i gang med Thonny)
kør koden.
Lad alle elever gøre det samme!

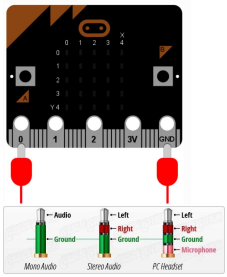
Lad dem lave et par ændringer i koden som beskrevet i Slide.

Spor 1 Modul a: Brug speech og music bibliotek

Projekt støttet af: midt regionmidtjylland

```
1 import speech
2
3 speech.say("Hi - Is there anyone out there")
```






```
1 import music
2
3 music.pitch(440,1000)
```

Med version 1 af Microbitten (MB) skal man tilslutte hovedtelefon eller lille højtaler som vist ovenfor. Med version 2 er det ikke nødvendigt da der er monteret højtaler på MB'en.

Prøv speech og music funktionerne til venstre. Find dem i dokumentationen
BBC micro:bit MicroPython documentation — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)

Eksperimenter med andre tekster i speech og andre kald til music.
Find inspiration i dokumentationen. Der er for eksempel forud definerede melodier der kan spilles med music.play(…).

Findes i afsnittet "Music" i "Tutorial" sektionen i dokumentationen for MicroPython på MicroBit.

Præsenter eleverne for modulerne/bibliotekerne speech og music
Vis dem søgefunktionen i dokumentationen så de finder referencen.

(klikbare links i slide – og direkte i slide sidst i stakken – MEN bedre at finde dem i dokumentationen på nettet – Bedre at eleverne selv får fornemmelsen for at navigere i dokumentationen og finde det frem som de har behov for.)

Giv dem en opgave med at få sat lyd ind sammen med displayvisningen fra eksemplet med knapper.

Spor 1 Modul a: Udbyg knap eksempel med lyd.

Projekt støttet af:

```
1  from microbit import *
2  import music
3
4  while True:
5      if button_a.is_pressed():
6          display.show(Image.HAPPY)
7          music.pitch(440)
8      elif button_b.is_pressed():
9          break
10     else:
11         display.show(Image.SAD)
12         music.stop()
13
14 display.clear()
15 music.stop()
```

Tilføjet at der spilles en tone med frekvens 440 (Kammertonen) når knap a aktiveres – og at tonen stoppes når knap slippes, samt når knap b trykkes ned.

Når der ikke angives flere parametre/argumenter til funktionen ”music.pitch(440)” end de 440 så bliver den ved med at spille tonen til den aktivt bliver stoppet med funktionen ”music.stop()”

Man kan også kalde music.pitch med flere parametre. Ex. ”music.pitch(440,50)” – Det vil resultere i at tonen kun spiller i 50 millisekunder – og så vil ”music.stop()” ikke være så vigtig i koden ved siden af – og pitch bliver genstartet så længe knap a er aktiveret – Men det vil lyde anderledes – Prøv det..

PS – det vil fortsat være fornuftigt at bruge ”music.stop()” da det sikrer at PIN’s på MB er i en veldefineret tilstand når koden afsluttes.

Dette er ‘svaret’ (et svar) på opgaven med at kombinere det de fandt under buttons med noget lyd.

Skal/kan vises efter opgaven, men naturligvis ikke før.

BEMÆRK – Det er også vigtigt at få udført funktionen music.stop(), for at få ryddet op i PIN tilstand. Hvis ikke kan man risikere at få en fejl hvis/når man bruge pins til noget andet.
Det kan også løses ved at strømmen kommer af Micro:Bitten.

Med eksemplerne i modul a bør man ikke komme i problemer.

Kode eksempel fra slide kan findes i LKS1Ma-Slide12-udbyg-knap-eksempel.py.

Kan evt. udlevere, så elever kan finde tilbage og gense/køre.
Til dem som måtte køre fast og have problemer med syntaks.

Spor 1 Modul a: Find grænsen for hvad pitch kan.

Projekt støttet af:
midt regionmidtjylland

```
from microbit import *
import music

for x in range(<start>,<slut>,<step>):
    music.pitch(x)
    sleep(100)
    print(x)
```

```
from microbit import *
import music

for x in range(20,20000,100):
    music.pitch(x)
    sleep(100)
    print(x)
```

Det hørbare område for lyd er 20-20000 Hz

Med music.pitch() funktionen kan I generere toner med frekvenser i området, men når man når under en vis værdi og over en vis værdi kan man ikke længere høre en forskel..

Lav eksperimenter med kode for at finde de værdi'er

Så hvad er den mindste hhv. den største værdi man kan angive som frekvens i pitch funktionen så den er tydeligt anderledes end den/de foregående.

Brug evt. en for løkke til at eksperimentere med..
Og måske over flere gange for at indsnævre..
Kom med et bud ..

HUSK – Eksperimenter kan man lære af. Brug det til at få en bedre fornemmelse/forståelse af hvad MB'en kan/ikke kan. Eksperimenter er et godt supplement til at læse om tingen. Men begge dele er ikke så tossede. I dokumentationen eller ved at søge på nettet kan I finde inspiration til hvad man kan og så kan jeres egne eksperimenter gøre jer endnu klogere på den specifikke ting.

Kan udelades/Option. Mest som illustration af "for-løkke", så hvis man ikke har fokus på åben bog tilgang (at lære lidt mere om programmering) er det at gå for langt.

Tag en snak om frekvenser og det hørbare område – spørg eleverne om det inden slide.

Højtaler og frekvensgenerator i Micro:Bitten kan ikke lave lyd i hele området, men hvad kan vi lave og høre med den?

Det er opgaven at finde ud af. Med Version 1 af microbitten er der en 'hård' grænse for hvad den kan – Den går simpelt hen ned når man når en bestemt pitch.

Med version 2 er det en anden sag, da den ikke går ned – men man kan så lede efter en værdi, hvor man ikke længere synes at man kan høre forskel mere. Det er så et lidt blødere svar man kan finde.

Men i begge tilfælde kan man bruge eksemplet til at lade eleverne først prøve sig frem, og komme med et svar, og så efterfølgende give dem en måde til at finde det lidt mere struktureret.

Formål overordnet set er at give dem mod på at eksperimentere, for at lære – her for at lære MB'en at kende – efterfølgende for at lære de indbyggede sensorer at kende.

Program findes: LKS1Ma-Slide13-PitchRange.py

Spor 1 Modul a: Lys til lyd.

Projekt støttet af:
midt regionmidtjylland

```
1 from microbit import *
2
3 while True:
4     x = display.read_light_level()
5     print(x)
6     sleep(100)
```

Først eksperimenteres lidt med sensoren, der læser lys niveau.

Lav et simpelt program der læser lys niveau og putter det over i én variabel, der udskrives. Her bruger vi navnet 'x' til variablen. Det vil være en god ide at bruge et mere sigende navn i længere programmer. Det vil hjælpe med læsevenligheden mht. at forstå koden.

I dokumentationen står at man kan få værdier mellem 0 og 255. Stemmer det overens med det I oplever. Det I får udskrevet i shell med print() funktionen? Prøv med lommelygten.

Det hørbare område er 20-20000 Hz. Ved at eksperimentere med music.pitch() funktionen så vil man opleve at man kan skelne forskel på lyd på MB fra 50 til 4000 Hz.

Så hvad skal der til for at få lys sensor input til at passe med en fornuftig pitch værdi?

Lad eleverne finde "read light level" med søgefunktionen i dokumentationen.
Lad dem køre program fra slide
Lad dem forholde sig til sensorværdierne som læses ud mens de hhv. skygger for display og lyser på det med lommelygte (evt. fra deres mobil telefoner)

-De kan selv taste det ind, men så skal man forvente at der er nogle der får problemer mht. syntaks, så der skal være tid til at komme rundt.

Man kan også udlevere koden færdig så eleverne bare skal loade det og opleve sensor værdien der ændres.

Og så snakke om opgaven med konvertering til lyd og lave den sammen.

Brug lejlighed til at tage en snak om forskellen på at importere biblioteker/moduler vha.

import microbit
Versus
from microbit import *

Der er nemlig lidt inkonsistens i dokumentationen. I tutorial benyttes den sidste, hvor den første bruges i API delen.

Når man bruger den sidste så behøver man ikke lave en fuld reference til modul og metode/funktion. Men det skal man hvis man bruger den første. Den første anses for at være den mest korrekte, da der så ikke kan være tvivl, hvis man har flere moduler med metoder/funktioner med samme navn. Den kræver til gengæld ikke så lange navne – og hvis man opererer med få biblioteker, som her, så kan det være en fordel.

Altså:

microbit.display.read_light_level()
Versus
display.read_light_level()

Program findes: LKS1Ma-Slide14-LysSensor.py

Spor 1 Modul a: Lys til lyd.

Projekt støttet af:

```
# Konvertering af lys intensitet fra MB's indbyggede lyssensor til til lyd (pitch/frekvens)
# afspillet på MB's indbyggede højtaler (V2 af MB) eller på ekstern højtaler/hovedtelefon
# koblet til PIN 0 og GND.
#
# Benytter funktionen/metoden read_light_level() fra display i modulet/biblioteket microbit
# til at aflæse lys niveau, og pitch funktionen/metoden til at spille en frekvens/lyd fra biblioteket
# music, hvorfor de 2 (microbit og music) skal importeres.
#
# Værdierne fra lysmåler er fra 0 til 255. For at få det til at blive en 'fornuftig' (hørbar) frekvens
# ganges lys intensitet med 15 (giver værdier mellem 0 og 15*255 = 3825) og adderer 50 for at komme til
# en frekvens der ligger mellem 50 og 3825+50 = 3875. Altså i det hørbare område og i et område som den indbyggede
# højtaler på MB'en kan reproducere.
#
# For at kunne følge lysintensiteten udskrives den til shell med den indbyggede print() funktion
# Og for at der skal være lidt tid til at høre lyden og se intensiteten udskrevet ventes der 100 msek mellem
# hver omgang i while løkken.
#
from microbit import *
import music

while True:
    x = display.read_light_level()
    music.pitch(x*15+50)
    print(x)
    sleep(100)
```

Et svar. Ganger med 15 og adderer 50 til lys niveau for at konvertere til en pitch. HUSK – Der er ikke kun eet rigtigt svar mht. kode!
Det er en god ide at tilføje kommentarer til sin kode. Det kan hjælpe med til at man forstår den bedre når man på et senere tidspunkt kommer tilbage til den.
Også værdifuldt hvis andre skal se og forstå koden. Naturligvis alt med måde ;-) MEN tænk over læseren/modtageren!

Et svar til gennemgang på klassen efter at eleverne selv har prøvet.

Program findes: LKS1Ma-Slide15-LysTilLyd.py

BEMÆRK at program aldrig afsluttes – andet end når det stoppes med stop eller strømmen tages.
Det betyder:

1) At man kan putte det ud lokalt på MB og få den til at have funktionen fra der kommer strøm på indtil strømmen fjernes. Vent med at vise det indtil man har været rundt om flere eksempler. Bedst at have programmer gemt lokalt på PC/MAC indtil man er lidt mere tryg med kodedelen.
2) At man som tidligere nævnt får efterladt PIN0 i en mode så den ikke kan bruges til andet, og det er så heller ikke noget problem i modul a, men der burde være en music.stop() inden man bruger pin 0 til andet. Det kan man gøre i shell, eller ved at tage strømmen af MB (tage den ud af USB), hvis altså ikke programmet kører lokalt på MB'en (mere om det i Slide 21). Det er IKKE noget man behøver at snakke om her, men god viden at have hvis der pludselig opstår 'sjove' fejl….

Spor 1 Modul a: Magnet felt til lyd.

Projekt støttet af: midt regionmidtjylland

```
1  from microbit import *
2
3  while True:
4      x = compass.get_field_strength()
5      print(x)
6      sleep(100)
```

Først eksperimenteres lidt med sensoren, der læser magnet feltet.

Lav et simpelt program der læser styrken af magnet feltet og putter det over i én variabel, der udskrives. Her bruger igen navnet 'x' til variablen. Her kan I se at det måske ville være godt med et andet navn – for det er jo IKKE længere det samme som i opgaven med lysintensiteten. Så var det et program med begge ville andre navne være værdifuldt.

I dokumentationen står der IKKE noget om hvilke værdier der kan komme tilbage fra funktionskaldet. Kun at det er i nano tesla. Så der er kun en vej til at finde ud af det. Prøv/Eksperimenter med en magnet omkring Micro:Bitten!

Hvad skal der til for at få magnet felt værdi omsat til en pitch der kan håndteres af funktionen music.pitch()?

HUSK – Eksperimenter kan man lære af. Brug det til at få en bedre fornemmelse/forståelse af hvad MB'en kan/ikke kan. Eksperimenter er et godt supplement til at læse om tingen. Men begge dele er ikke så tossede. I dokumentationen eller ved at søge på nettet kan I finde inspiration til hvad man kan og så kan jeres egne eksperimenter gøre jer endnu klogere på den specifikke ting.

Kan udelades, hvis man kun vil give grundlaget for at kunne komme i gang med Spor 2.

Man kan gå fra at have lavet koden med lys til lyd til at vise hvordan man får koden lagt på MB'en (slide 21) , så den kan køre for sig selv, og derfra gå over til at fortælle om øvrige sensor funktioner og derfra til Spor 2, hvor man så fortæller om de forskellige typer af kode som er lavet og som kan loades på MB'en.

Tilsvarende som med lys, men nu med magnetfelt.
Nyt programmeringsteknisk – operators and casting.

Udlever en magnet til eleverne.
Lad eleverne finde get_field_strength() med søgefunktionen i dokumentationen. (Der er også en klikbar link på kodestump i slide – og til sidst i slide stak)
Lad dem køre program fra slide

Få dem til at forholde sig til de værdier de kan få fra sensoren, så de kan få en fornemmelse for hvad det er for en størrelse.

For dem der hurtigt er færdige kan man lade dem udskifte kaldet compass.get_field_strength() med funktionerne:
compass.get_x(), compass.get_y() og compass.get_z()

Og få en fornemmelse for hvad forskellen er.

Og så som med lys få dem til at lave en konvertering fra magnet felts værdier til lyd.

Pas på med magnet på den indbyggede højtaler – Det kan få den slået ud af kurs!!

Så magnet rundt på og omkring MB's forside. Slås den ud af kurs kommer den sandsynligvis tilbage efter at strømmen har været af.

Program findes: LKS1Ma-Slide16-Magnetometer.py

Spor 1 Modul a: Magnet felt til lyd.

Projekt støttet af:

```
# Konvertering af magnetfelts værdier fra MB's indbyggede magnetometer til til lyd (pitch/frekvens)
# afspillet på MB's indbyggede højtaler (V2 af MB) eller på ekstern højtaler/hovedtelefon
# koblet til PIN 0 og GND.
#
# Benytter funktionen/metoden get_field_strength() fra compass i modulet/biblioteket microbit
# til at aflæse magnet feltets styrke, og pitch funktionen/metoden til at spille en frekvens/lyd fra biblioteket
# music, hvorfor de 2 (microbit og music) importeres.
#
# Værdierne fra magnetometer er altid positive og kan gå op til ihvertilfælde 2.5 mill.
# For at få det til at blive en 'fornuftig' (hørbar) frekvens divideres med 1000, og adderer 50.
# MEN skal desuden få det til at være et heltal' og ved normal division (/) får vi decimaler.
# - derfor bruges funktionen int() til at konvertere til heltal!
#
from microbit import *
import music

# Da felt styrken ser ud til at kunne nå op til ca. 2.5 mill og altid er positiv

while True:
    x = compass.get_field_strength()
    print(x)
    music.pitch(int(x/1000)+50)      # Kan også bruge floor division // - Se python aritmetiske operatorer
    sleep(100)
```

Et svar. Deler med 1000 og adderer 50 til magnet felt værdi for at konvertere til en pitch MEN skal også lave det til heltal (ikke decimaltal), da funktionen music.pitch() kunj forstår positive heltal! **Igen - HUSK – Der er ikke kun eet rigtigt svar mht. kode!**

Python Operators (w3schools.com) **(afsnit om aritmetiske operatorer) eller** Python Casting (w3schools.com) **(int() funktionen til konvertering til heltal)**

Kan udelades, hvis man kun vil give grundlaget for at kunne komme i gang med Spor 2.

Program findes: LKS1Ma-Slide17-MagnetTilLyd.py

Spor 1 Modul a: Accelerometer til lyd.

Projekt støttet af:
midt regionmidtjylland

```
from microbit import *

while True:
    acc_x,acc_y,acc_z = accelerometer.get_values()
#    acc_all = accelerometer.get_values()          # alternativ - tuple/sæt af værdier
    print(acc_x)
#    print(acc_all[0])                             # alternativ - udpeg i tuple
    sleep(100)
```

Først eksperimenteres lidt med sensoren, der læser g–værdier fra accelerometeret .

Hvad skal der til for at få magnet felt værdi omsat til en pitch der kan håndteres af funktionen music.pitch()?

Læs evt. om tuples:
Python Tuples (w3schools.com)

Kan udelades, hvis man kun vil give grundlaget for at kunne komme i gang med Spor 2.

En gang mere – Denne gang med accelerometer.

Find i dokumentation – søg på ”accelerometer”.
Eksperimenter med sensor, så man får en fornemmelse. Vip med den.
Konverter til lyd.

Bemærk at dokumentationen siger at man får værdier i området +/- 2000, men i test så kan man få værdier der er større end 2000 – Det sker ved et ‘kraftigt’ stød.
Men ved et normalt vip får man værdier der ligger i området +/- 1100. Den viden skal man så bruge i konverteringen efterfølgende.

Ny ting programteknisk – Tuples og/eller flere variable retur i eet kald, samt abs() funktionen i math til at lave en negativ værdi om til en positiv.

Program findes: LKS1Ma-Slide18-Accelerometer.py

Spor 1 Modul a: Accelerometer til lyd.

Projekt støttet af:

```
# Konvertering af g-kraftspåvirknings værdier fra MB's indbyggede accelerometer til til lyd (pitch/frekvens)
# afspillet på MB's indbyggede højtaler (V2 af MB) eller på ekstern højtaler/hovedtelefon
# koblet til PIN 0 og GND.
#
# Benytter funktionen/metoden get_values() fra accelerometer i modulet/biblioteket microbit
# til at aflæse kraftpåvirkningen, og pitch funktionen/metoden til at spille en frekvens/lyd fra biblioteket
# music, hvorfor de 2 (microbit og music) importeres.
#
# Værdierne fra accelerometer i x-aksen er ifølge specifikationen +/- 2000, men kan måle værdier over 2000 ved
# kraftig påvirkning. Ved 'normal' vip højre/venstre får man ca. værdier i området +/- 1100.
# For at få det til at blive en 'fornuftig' (hørbar) frekvens skal det først og fremmest være positivt. Det kan
# man få det til at blive med abs() funktionen, men så kan man ikke skelne højre versus ventre vip. Derfor
# adderes istedet 1500 og så for at være sikre på at det kommer indenfor en fornuftig ranege testes på værdi inden
# kaldet til music.pitch() - MEN det kan laves på andre måder!! De 3907 kommer fra at version 1 af MB går ned hvis
# man bruger en frekvens der er 3907 eller større. Det er ikke tilfældet for version 2!!
#
from microbit import *
import music

while True:
    acc_x,acc_y,acc_z = accelerometer.get_values()
    print(acc_x)
#    music.pitch(abs(acc_x))    # den hurtige løsning   - abs() laver neg til pos

    freq = acc_x+1500          # adderer tilstrækkelig stor værdi
    if freq>50 and freq<3907:  # tester at det er i området for at være sikker
        music.pitch(freq)
    sleep(50)                  # Venter kun 50msek (hvor der i andre eksempler har stået 100
                               # Det er for at give en lidt mere 'flydende' oplevelse - Prøv forskellen!
```

Læs evt.: Python Math (w3schools.com) **om abs() funktionen.**

Kan udelades, hvis man kun vil give grundlaget for at kunne komme i gang med Spor 2.

Ny ting programteknisk – Tuples og/eller flere variable retur i eet kald, samt abs() funktionen i math til at lave en negativ værdi om til en positiv.

MEN – med abs() får vi ikke muligheden for at lave en lav frekvens ved vip til venstre og høj ved vip til højre – så derfor den anden løsning.

Kan bruge eksemplet til at snakke om normalisering af data, men her er der bare en enkelt løsning.

Program findes: LKS1Ma-Slide19-AccTilLyd.py

Spor 1 Modul a: Mikrofon måling.

Projekt støttet af:

```
1  # Udlæsning af lyd niveau fra indbygget mikrofon.
2  from microbit import *
3
4  while True:
5      s = microphone.sound_level()
6      if s>0: print(s)
7      sleep(10)
```

Microphone V2 — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)

Eksperimenter med sensoren, der udlæser lyd tryk.

Hvad skal der til for at give udslag af en vis størrelse?

Hvad kan man bruge en måling af lyd tryk til?

Kan udelades, hvis man kun vil give grundlaget for at kunne komme i gang med Spor 2.

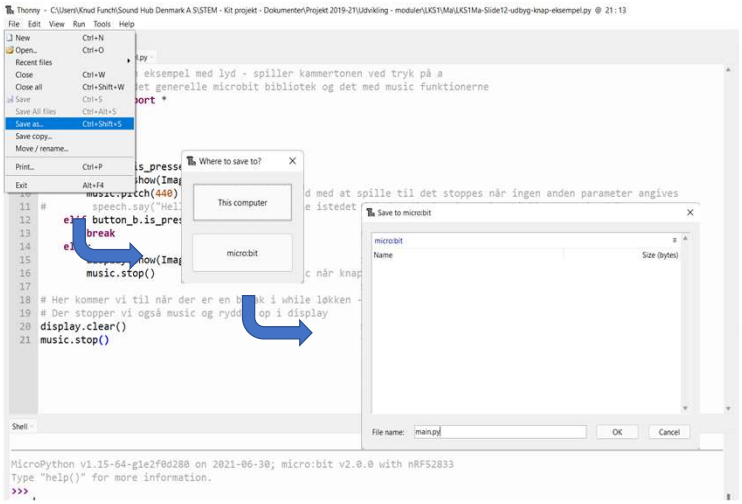
For at vise den sidste indbyggede sensor - mikrofon

Find i dokumentation – søg på ”sound_level”.
Eksperimenter med sensor, så man får en fornemmelse.
Her giver det ikke så meget mening at konvertere til lyd, så det springes over – men man kan jo tage en snak om hvad en støj måling kan bruges til.

Program findes: LKS1Ma-Slide20-Microphone.py

Spor 1 Modul a: Gemme program lokalt på MB

Projekt støttet af:



Man kan gemme et program lokalt på Micro:Bitten og ved at tilslutte en batteripakke så kan det program køre lokalt uden at MB er tilsluttet PC/MAC.

Det gør man ved at stoppe evt. program som er i gang! Med stop! Sikre at Shell står klar som vist til venstre.

Så gå til file>save as og så vælge Micro:Bitten.
Man skal så gemme program med navnet ”main.py” (alle bogstaver **SKAL** være små).

Det program startes så snart MB får strøm.

Når I så efterfølgende skal gennem noget/eller have anden kode til at køre på den MB skal I først stoppe programmet med STOP og sikre jer at I har prompt i shell som vist. Ellers kan I få div fejl meldinger/eller ikke kunne gemme etc.

Udlever batteriholder til elever som kan kobles til MB.

For at gemme et program lokalt på Micro:Bitten (så det kan køre når der kommer strøm på uden at være koblet til PC/MAC) skal man:

1) Sikre sig at der IKKE er noget program som kører på MB – Brug stop – Skal se ”>>>” og ”Micropyton v1… ” i Shell!
2) Gå ind via menu’en Filer og vælge save as, og så vælge microbit – Hvis ikke der står microbit er der noget galt – se 1)
3) Gemme under navnet ”main.py” SKAL være med små bogstaver.

Det er den grundlæggende ting man skal vide for at kunne loade færdig kode ned på MB’s ifm Spor 2.

Og så at kunne skifte tilbage til at køre fortolkeren på PC/MAC/PI fremfor på MB.

BEMÆRK – Ved at gemme et af de programmer der laver sensor input om til lyd på MB’en. Ex LysTilLyd så kommer der lyd ligeså snart der er strøm på MB’en
Også når man kobler den tilbage på PC/MAC via USB kablet! Det kan godt give nogle stressede siuationer! ;-) Lyden bliver også ved selvom man får stoppet program med stop og har promt i shell!

- Man skal så skynde sig at køre et program fra Thonny og stoppe det
- Man kan bruge music.stop() kald direkte fra fra shell, men skal så huske at lave en import music først (også bare i shell)
- Man kan reloade micropython fortolker fra run menu, men det er et lidt voldsomt skridt.

Og så kan man jo tage en snak om at man evt. skal lave programmet så det kan stoppe på 'ordentlig vis' ved et knap tryk etc.. ..

Hvis man vil undgå den slags så kan man bruge det første program fra slide 12 som det man gemmer!

Den korteste vej, hvis man ønsker at køre lukket bog og videre i spor 2 vil være:
- **Introduktion til (IDE) Thonny – se notat om Thonny**
- **Introduktion til programmering md fortælling og demo -Slide 4 og 6**
- **Introduktion til MB – Slide 9,10 og 12**
- **Gemme program – Slide 21 med kode fra Slide 12.**

Vil man klæde eleverne på til at kunne gå åben bog går man dybere, samt tilbyder muligheden for Modul B

.

Projekt støttet af:
midt
regionmidtjylland

Supplerende Materiale

Skift mellem fortolkere. På PC/MAC skift til på Micro:Bit

Installation af MicroPython fortolker på Micro:Bit

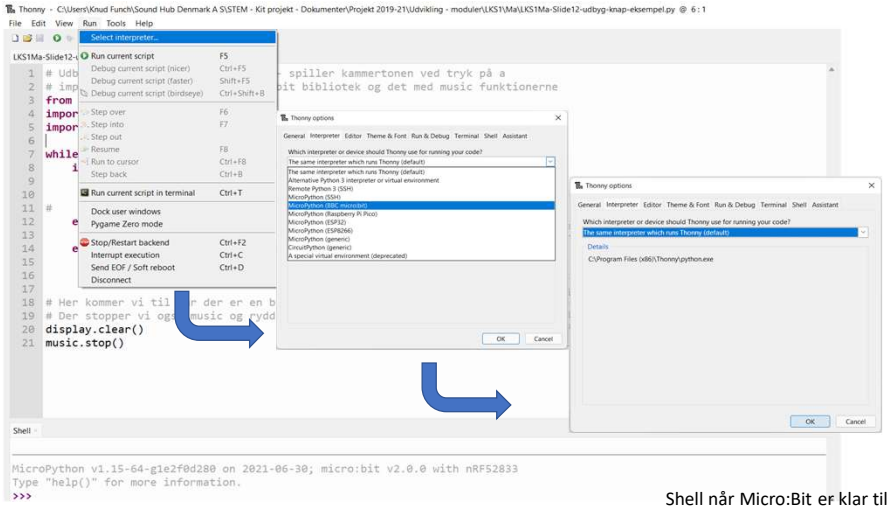
Sammenligning MakeCode Blokprogrammering og MicroPython

Sammenligning MicroPython og Arduino C

Samling af links til materiale, som der henvises til undervejs.

Spor 1 Modul a: Skift mellem fortolkere

Projekt støttet af:



Sekvens når man skifter mellem Fortolkere (maskiner) at udføre Python/Micropython programmer på.

Shell når Micro:Bit er klar til at eksekvere et MicroPython program

Shell når PC/MAC er klar til at eksekvere et Python program

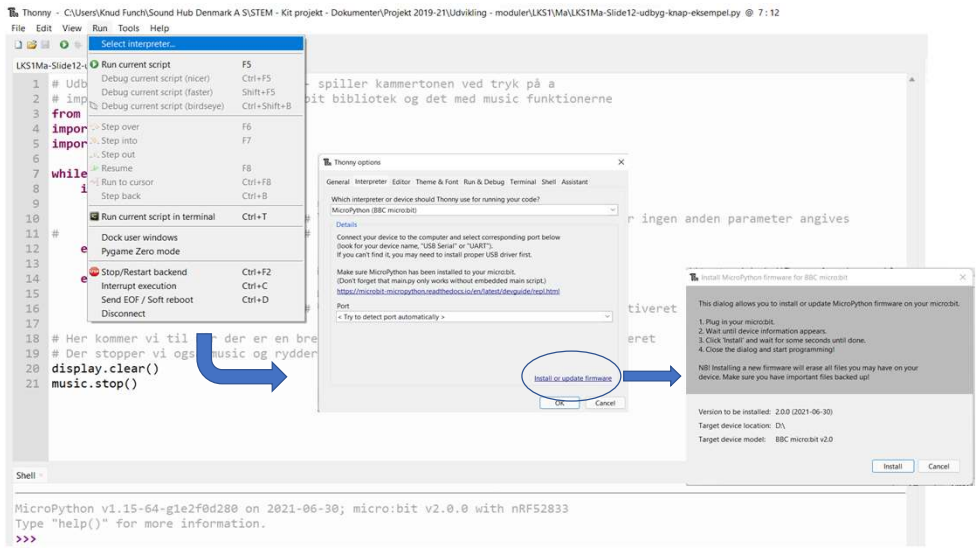
Bedst at demonstrere med Thonny og Micro:bit tilkoblet, men som oversigt til at forklare er der her en slide.

Det er en væsentlig forudsætning for at bruge LYD-Kit (alle spor) at man kan skifte mellem at køre/loade programmer på Micro:Bitten og på PC/MAC/PI – At skifte mellem MicroPython fortolkeren på Micro:Bitten og Python fortolkeren på PC/MAC/PI. Derfor er det vigtigt også i en 'lukket bog' tilgang at få gjort denne forskel tydelig.

Den væsentligste feedback med Thonny som IDE er gennem promt. og fortolker udmelding i Shell.
Når der er >>> er fortolkeren klar – og der er ikke et program som er aktivt
Når der står: "Python …. " er det fortolkeren på PC/MAC/PI der er klar.
Når der står: "MicroPython …." er der fortolkeren på Micro:bitten som er klar.

Spor 1 Modul a: Installation af MicroPython på Micro:Bit

Projekt støttet af:



For at gøre Micro:bitten klar til at køre MicroPython programmer skal der installeres en ny firmware – (Micropyhon fortolker) i stedet for den kode som ligger i Micro:Bitten fra fabrikken, som kan håndtere blokprogrammer fra microsoft Makekode. Det bliver gjort med sekvensen vist i Slide, men igen så er det bedre at demonstrere det med Thonny og Microbit koblet til USB stikket.

Husk USB kabel – og at der er adapter der passer på PC/MAC siden.

BEMÆRK:

- En MicroBit ab fabrik kommer med en lille startsekvens som fortæller om dens egenskaber når der kommer strøm på. Det kan man godt udnytte til at fortælle om den inden man skifter firmware.
- Makecode editor viser at man kan lave python programmer – Det er ikke rigtig MicroPython, men kun pseudo kode så på ingen måde sammenlignelig!!

Spor 1 Modul a: Makecode python vs MicroPython

Projekt støttet af:

The Python in the MakeCode editor is different to the MicroPython in the micro:bit Python Editor. Although both editors appear like Python 3 they actually are completely different in the way their programs are interpreted and run.

This explains why you may find programs that run fine in the MicroPython editor cause bugs in the MakeCode editor, or vice versa. For example, our simple "Hello, World" program in MakeCode Python would appear as:

```
basic.show_string("Hello!")
basic.show_icon(IconNames.HEART)
```

Where as in MicroPython you would have to import modules first, and then call different classes as follows:

```
from microbit import *

while True:
    display.scroll('Hello, World!')
    display.show(Image.HEART)
    sleep(2000)
```

MakeCode Python

MakeCode Python runs the micro:bit DAL (Data Abstraction Layer) just as the other languages in MakeCode do. It is in fact just static TypeScript surfaced with Python syntax. This is the same structure as the blocks in the MakeCode editor, which is why it is possible to switch between the different languages. However, this does mean that there are certain limitations to MakeCode Python, as it shares the same limitations as the other languages in MakeCode. One example of this would be the limitation to access of only 3 PWM pins on the micro:bit. You can access the MakeCode Python API in the documentation https://makecode.microbit.org/reference. This shows how the different functions map on to each other across languages.

Micropython

MicroPython is different to this, as this has its own runtime on the micro:bit. MicroPython is in fact a tiny python interpreter that runs on the board (at a low level) and is a popular programming language because it is optimised to work on microcontrollers — just like the micro:bit. It is almost a full re-implementation of python 3 but is designed to be able to run in a low memory and low power environment. The python editor provided by micro:bit is the perfect environment for building programs to run on your micro:bit, as you can easily access the REPL (Read, Evaluate, Print, Loop) and also connect directly to your micro:bit to flash programs via serial connection. Furthermore, there is an easy to use file system where you can drag in modules you have created. You can access the MicroPython API documentation here https://microbit-micropython.readthedocs.io/en/latest/.

MakeCode Python and MicroPython : Help & Support (microbit.org)

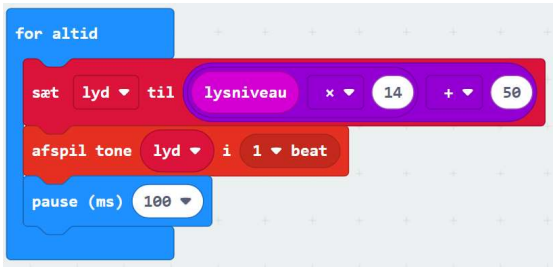
Dette er med MicroPython fortolker fra
BBC micro:bit MicroPython documentation — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)

Og brugt via Thonny IDE'en som pt (2020/2021) er den som understøttes af Raspberry PI Foundation.

Bemærk forskel til den Python som er med i Makecode fra MicroSoft!!

Spor 1 Modul a: Makecode (Blok) versus MicroPython (Tekst)

Projekt støttet af: midt regionmidtjylland




```
from microbit import *
import music

while True:
    lyd = display.read_light_level()*14+50
    music.pitch(lyd)
    sleep(100)
```

MicroPython udgaven her er med MicroPython fortolker fra [BBC micro:bit MicroPython documentation — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)]

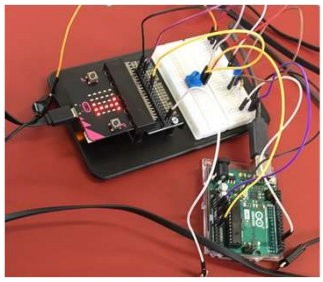
Og brugt via Thonny IDE'en som pt (2021/2021) er den som understøttes af Raspberry PI Foundation. [Thonny, Python IDE for beginners]
Bemærk forskel til den Python som er med i Makecode fra MicroSoft!

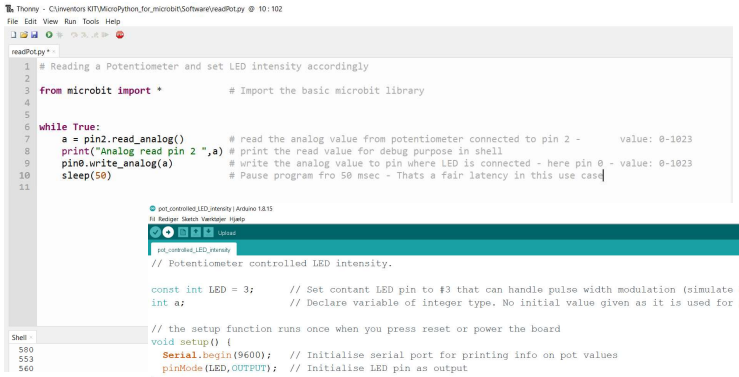
Lyd ændres mere glidende i python udgaven, da den kan sættes til at afspille i baggrund indtil ny.
Pausen i blok kan udelades og så bliver det lidt mere glidende i dette eksempel, men ikke som Python udgaven.

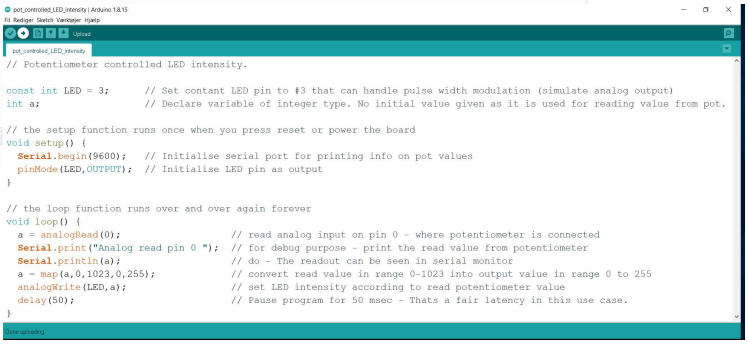
Så kode er næsten ens men ikke helt samme resultat. (blokprogrammeringen har ikke samme grad af muligheder). Men med Blokprogrammering undgår man syntaksfejl, så det er lettere at gå til. Omvendt er Python et sprog, der er forholdsvis let at gå til og som skalerer til 'professionelt' brug i mange domæner.

Spor 1 Modul a: Arduino versus MicroPython

Projekt støttet af: midt regionmidtjylland







'Hello world eksemplet fra Arduino" Kontroller intensiteten af en lysdiode fra et potentiometer.

Med MicroPython på Micro:Bit og med Arduino "C" på en Arduino Uno.

Er man til åben bog tilgangen med hensyn til Mikrocontrollere vil man sikkert allerede kende Arduino universet.
Arduino er både et firma der laver HW og et IDE til mikrokontrollere med support for mange slags HW, og med et hav af færdige programmer (sketches) man kan tage udgangspunkt i.

Grunden til at Python er valgt til LYD-Kit er:

1) Python er lettere at komme i gang med
2) Valget af Micro:Python hænger også sammen med valget af Micro:Bit. Den er valgt fordi:
    1) Den bliver brugt i grundskolen og er muligvis derfor et godt første trin ind til noget mere avanceret
    2) Kan bruge den til at bringe sensor input til både PC/MAC og PI i større opstillinger (systemmæssigt) som bruges i Spor 2 og 4.
    (Det skal dog bemærkes at der også findes Arduino biblioteker til Micro:Bitten – så igen er det som sådan ikke et enten eller, ligesom der findes MicroPython til en række andre HW's)
3) Python skalerer 'bedre' – Er ikke kun til mikrocontrollere, men kan også bruges på PC/MAC/PI som benyttes i Spor 2,3 og 4, Derved er der et fælles teknisk grundlag for alle spor. Desuden benyttes Python til en hel række andre anvendelser end 'styring af mikrokontrollere'; som datascience og machine-learning. Derfor er der en

stor sandsynlighed for at eleverne vil kunne få gælde af Python indenfor mange STEM relateret fag i deres videre uddannelse.

4) Det virker til at der kommer mere og mere 'Python' (MicroPython) på mikrokontrollere eftersom HW (CPU'en) bliver kraftigere og kraftigere på diverse kommercielt tilgængelige HW til DIY folket.

MEN når eleverne når til at ville mere avancerede ting med Micro Kontrollere så er det ikke et valg mellem det ene eller det andet. Så er det både-og. Valget vil blive betinget af hvad man ønsker at lave. Og det er valg af både HW og SW.

Spor 1 Modul a: Reference links

Projekt støttet af:

Thonny, Python IDE for beginners

Python Tutorial (w3schools.com)
BBC micro:bit MicroPython documentation — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)

Stack Overflow Developer Survey 2021
Top 15 Programming Languages by Popularity (2004-2019) – YouTube
PYPL PopularitY of Programming Language index

Python While Loops (w3schools.com)
Python Conditions (w3schools.com)
Python Syntax (w3schools.com)

Python Lists (w3schools.com)
Python For Loops (w3schools.com)
Python Functions (w3schools.com)

Python Modules (w3schools.com)

Spor 1 Modul a: Reference links

Projekt støttet af:

Python Random Module (w3schools.com)

Random Number Generation — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)

Hardware (microbit.org)

Buttons — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)

Music — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)
Speech — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)

Display — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)

Compass — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)
Python Operators (w3schools.com)
Python Casting (w3schools.com)

Accelerometer — BBC micro:bit MicroPython 1.0.1 documentation (microbit-micropython.readthedocs.io)
Python Tuples (w3schools.com)
Python Math (w3schools.com)

MakeCode Python and MicroPython : Help & Support (microbit.org)