

Python: knappar

Skrevet av: Omsett frå *microbit-micropython.readthedocs.io* (<https://microbit-micropython.readthedocs.io/en/latest/tutorials/buttons.html>)

Oversatt av: Stein Olav Romslo

Kurs: Microbit

Tema: Elektronikk, Tekstbasert

Fag: Programmering

Klassetrinn: 5.-7. klasse, 8.-10. klasse, Videregående skole

Introduksjon

Til no har me laga kode som får micro:bit-en til å gjere noko. Det kallar me *output*. Men me treng òg at micro:bit-en kan reagere på ulike handlingar. Det kallar me gjerne for *input*.

Skilnaden er enkel å hugse: input er det me sender inn til micro:bit-en og som den prosesserer, medan output er det micro:bit-en viser ut til verda.

Den mest åpenbare måten å gi input til micro:bit-en på er dei to knappane A og B. Fyrste oppgåve er å finne ut korleis me kan få MicroPython til å registrere at knappane blir trykka på.

Det er overraskande enkelt:

```
from microbit import *  
  
sleep(10000)  
display.scroll(str(button_a.get_presses()))
```

Det einaste dette skriptet gjer er å sove i ti tusen millisekund (med andre ord 10 sekund), og så scrolle så mange gonger du trykka knappen A over skjermen. Det er det!

Trass i at det er eit veldig enkelt skript, så introduserer det eit par interessante idear.

1. *sleep* -funksjonen får micro:bit-en til å vente eit gitt antal millisekund. Viss du vil pause programmet ditt, så gjer du det slik.
2. Det er eit objekt `button_a` som let deg hente ut kor mange gonger den har blitt aktivert via `get_presses` -metoden.

Sidan `get_presses` gir ein numerisk verdi, og `display.scroll` berre viser tekst, så må me konvertere den numeriske verdien til ein tekststreng. Me gjer det med `str`-funksjonen (namnet er kort for *string* som kanskje gjer det enklare å hugse).

Den tredje linja er litt som ein lauk - den har fleire lag. Viss du tenker på parentesane som laga på lauken vil du leggje merke til at `display.scroll` inneheldt `str` som igjen inneheldt `button_a.get_presses`. Python startar å skrelle lauken innanfrå og ut. I klartekst startar den altså med dei inste parantesane og går utover. Det kallar me gjerne for *nesting* av funksjonar.

La oss late som du har trykka knappen 10 gonger. Her er korleis Python tolkar det som skjer på den tredje linja:

Python les heile linja og hentar ut verdien frå `get_presses`:

```
display.scroll(str(button_a.get_presses()))
```

No som Python veit kor mange knappetrykk som har skjedd, så kan den konvertere den numeriske verdien til ein tekststreng.

```
display.scroll(str(10))
```

Endeleg veit Python kva som skal rulle over skjermen:

```
display.scroll("10")
```

Handlingsløkker

Ofte treng du at programmet ditt skal vente på at noko skal skje. For å gjere det kan du få ein liten bit kode til å køyre om att og om att, og reagere på ulike input - til dømes at du trykkar på A.

Slike kodesnuttar kallar me *while-løkker*. I Python kan du lage dei ved å bruke `while` for å sjekke om noko er sant, og viss det er det blir ei *kodeblokk* (innmaten i løkka) køyrt. Viss det ikkje er sant, så bryt den ut av løkka, og fortset med reisten av programmet.

Python gjer det enkelt å definere kodeblokker. Si at me har ei hugseliste skrive på papir. Den kan sjå om lag slik ut:

```
Handle  
Reingjere takrenna  
Klyppe plena
```

Viss me vil gjere hugselista meir detaljert, så kan den bli utvida slik:

```
Handling:
    Egg
    Bacon
    Tomater
Reingjere takrenna:
    Låne stige frå naboen
    Finne hanskar og plastpose
    Returnere stige til naboen
Klyppe plena:
    Tømme grasklypparen når den er full
    Sjekke drivstoffnivå til grasklypparen
```

Det er åpenbart at hovudoppgåvene er brotne ned til mindre oppgåver med *innrykk* (indented) under hovudoppgåvene. Slik at `Egg` , `Bacon` og `Tomater` åpenbart er kopla til `Handling` . Ved å bruke innrykk blir det enklare å få oversikt, og me kan enklare sjå samanhengen mellom ulike oppgåver.

Dette kallast *nesting*. Me kan bruke nesting til å definere ei kodeblokk som følgjer:

```
from microbit import *

while running_time() < 10000:
    display.show(Image.ASLEEP)

display.show(Image.SURPRISED)
```

Funksjonen `running_time` returnerer antal millisekund sidan micro:bit-en vart starta.

Linja `while running_time() < 10000:` sjekkar om tida er mindre enn ti tusen millisekund (ti sekund). Viss den er det, så visast `Image.ASLEEP` . Merk korleis dette er rykka inn under `while`-løkka *akkurat som i hugselista vår*.

Det er klart at viss køyretida er større enn eller lik 10000 millisekund, så vil skjermen vise `Image.SURPRISED` . Kvifor? Fordi vilkåret til `while`-løkka er usant (`running_time` er ikkje lengre `< 10000`). I det tilfellet er løkka ferdig, og programmet vil fortsetje under `while`-løkka si kodeblokk.

Handtering av løkker

Viss me vil at MicroPython skal reagere på eit knappetrykk bør me plassere den i ei løkke som går evig og sjekke om knappen `is_pressed` .

Å lage ei løkke som går for alltid er enkelt:

```
while True
    # gjer noko
```

Hugs at `while` sjekkar om `noko` er sant (`True`) for å finne ut om den skal køyre kodeblokka eller ikkje. Det er åpenbart at `True` alltid er sant, så dermed har du laga ei evig løkke!

La oss lage eit veldig enkelt elektronisk kjæledyr. Den er alltid trist viss du ikkje trykkar A. Viss du trykkar B så døyr dyret. (Eg innser at dette ikkje er eit veldig hyggeleg spel, så kanskje du kan klare å forbetre det).

```
from microbit import *

while True:
    if button_a.is_pressed():
        display.show(Image.HAPPY)
    elif button_b.is_pressed():
        display.show(Image.SKULL)
        sleep(1000)
        break
    else:
        display.show(Image.SAD)

display.clear()
```

Ser du korleis me sjekkar kva knappar som blir trykt? Me brukte `if`, `elif` (kort for "else if") og `else`. Desse kallast for *vilkår* og virkar som følgjer:

```
if noko is True:
    # gjer ein ting
elif ein annan ting is True:
    # gjer ein annan ting
else:
    # gjer endå ein ny ting.
```

Dette er veldig likt engelsk!

Metoden `is_pressed` produserer berre to resultat: `True` eller `False`. Når du heldt inne knappen returnerer den `True`, elles returnerer den `False`. Koden over seier det følgjande på norsk: "For evig og alltid, viss A blir trykt vis eit smilande ansikt, viss B blir trykt vis ein hovudskalle i 1 sekund og bryt deretter løkka, elles vis eit trist ansikt". Me bryt ut av løkka (stoppar programmet frå å køyre for alltid) ved å bruke `break`.

Lag spelet mindre tragisk

Kan du tenke på måtar å gjere spelet mindre tragisk på? Korleis kan du sjekke om *begge* knappane er trykka ned samstundes?

Hint: Python har `and`, `or` og `not` - logiske operatorar for å sjekke fleire logiske påstandar samstundes (ting som anten er `True` eller `False`).

Lisens: The MIT License (MIT)

(<https://github.com/bbcmicrobit/micropython/blob/master/LICENSE>)