Micro:bit

Python: Knapper

Skrevet av: Oversatt fra microbit-micropython.readthedocs.io (https://microbit-micropython.readthedocs.io/en/latest/tutorials/buttons.html)

Oversatt av: Øistein Søvik og Susanne Rynning Seip

Kurs: Microbit
Tema: Elektronikk, Tekstbasert
Fag: Programmering, Matematikk
Klassetrinn: 5.-7. klasse, 8.-10. klasse, Videregående skole

Denne oppgaven er en del av oppgavesamlingen *Programmering i micro-python* og bygger videre på Python: Bilder (../python_images/python_images_nb.html).

Vi anbefaler at du laster ned og skriver koden din i mu editor (https://codewith.mu/) når du jobber med disse oppgavene. Instruksjoner for hvordan man laster ned Mu finner du på nettsiden via linken.

Når Mu er installert kan du koble micro:biten din til datamaskinen via en USB-kabel. Skriv koden din i editor-vinduet og trykk på “Flash”-knappen for å laste koden over på micro:biten. Hvis det ikke fungerer, sørg for at micro:biten har dukket opp som en USB-enhet på datamaskinen din.

Introduksjon

Så langt har vi laget kode som får micro:bit'en til å gjøre noe. Dette kalles for *output*. Men, vi trenger og at micro:bit'en kan reagere på ulike handlinger. Slike handlinger kalles gjerne for *input*.

Forskjellen er enkel å huske: output er hva micro:bit'en viser til verden, mens input er hva som blir prosessert av micro:bit'en.

Den mest åpenbare måten å gi input til micro:bit'en på er de to knappene `A` og `B`. Første oppgave er å finne ut hvordan vi få MicroPython til å registrere at knappene trykkes.

Dette er overraskende enkelt:

```
from microbit import *

sleep(10000)
```

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

Det eneste dette skriptet gjør er å sove i ti tusen millisekunder (med andre ord 10 sekunder) og deretter scroller antallet ganger du trykket knappen `A` over skjermen. Det er det!

På tross av at dette er et veldig enkelt script så introduserer det et par interessante nye idéer.

1. `sleep` *funksjonen* får micro:bit'ten til å vente et gitt antall milliseukunder. *Dersom du ønsker å pause programmet ditt, er dette hvordan det gjøres*.
2. Det er et objekt `button_a` som tillater deg å hente ut antall ganger den har blitt aktivert via `get_presses` metoden.

Siden `get_presses` gir en numerisk verdi og `display.scroll` bare viser tekst, trenger vi å konvertere den numeriske verdien til en tekststreng. Vi gjør dette med `str` funksjonen (Navnet er kort for *string* som kanskje gjør det enklere å huske).

Den tredje linjen er litt som en løk. Dersom du tenker på parentesene som lagene på løken vil du legge merke til at `display.scrolll` inneholder `str` som igjen inneholder `button_a.get_presses`. Python begynner å skrelle løken innenfra og ut, i klartekst begynner den altså med de innerste parentesene og går utover. Dette kalles gjerne for *nesting* av funksjoner.

La oss late som du har trykket knappen 10 ganger. Her er hvordan Python tolker hva som skjer på den tredje linjen:

Python leser hele linjen og henter ut verdien av `get_presses`:

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

Nå som Python vet hvor mange knappetrykk som har skjedd, kan den konvertere denne numeriske verdien til en tekststreng.

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

Endelig så vet Python hva som skal scrolles over displayet:

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

Handlingsløkker

Ofte trenger du at programmet ditt venter på at noe skal skje. For å gjøre dette kan du få en liten bit kode til å kjøre igjen og igjen som forteller hvordan programmet skal reagere på ulike input som for eksempel at du trykker på `A`

på ulike input som for eksempel at du trykker på `A`.

Slike kodesnutter kalles for `while`-*løkker*. I Python kan disse lages ved å bruke `while` som sjekker om noe er `True` og dersom det er kjøres en *kodeblokk* som er innmaten i løkken. Dersom det ikke er det, så bryter den ut av løkka og resten av programmet fortsetter.

Python gjør det enkelt å definere kodeblokker. Si jeg har en huskeliste skrevet på et papir. Den ser omtrent ut som det her:

```
Handle
Rengjøre takrennen
Klippe plenen
```

Dersom jeg ønsker å gjøre huskelisten min mer detaljert, ville jeg kanskje ha utvidet den slik:

```
Handling:
   Egg
   Baccon
   Tomater
Rengjøre takrennen:
   Låne stige fra naboen
   Finne hansker og plastikkpose
   Returnere stige til naboen
Klippe plenen:
   Tømme gressklipper når den er full
   Sjekke drivstoffnivå til gressklipper
```

Det er åpenbart at hovedoppgavene er brutt ned til mindre oppgaver med *innrykk* (indented) under hovedoppgavene. Slik at `Egg`, `Baccon` og `Tomater` åpenbart er relatert til `Handling`. Ved å bruke innrykk blir det enklere å få oversikt, og vi kan enklere se sammenhengen mellom ulike oppgaver.

Dette kalles for *nesting*. vi kan bruke nesting til å definere en kodeblokk som følger:

```
from microbit import *

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

display.show(Image.SURPRISED)
```

Funksjonen `running_time` returnerer antall millisekunder siden micro:bit'en ble startet.

Linjen `while running_time() < 10000:` sjekker dersom tiden er mindre enn 10000 millisekunder (i.e. 10 sekunder). Dersom den er, det så vises `Image.ASLEEP`. Merk

hvordan dette er innhogd under `while` løkka *akkurat som i huskelisten vår*.

Åpenbart, dersom kjøretiden er lik eller større enn 10000 millisekunder så vil displayet vise `Image.SURPRISED`. Hvorfor? Fordi betingelsen til `while`-løkka vil være usann (`running_time` er ikke lengre `< 10000`). I det tilfellet er løkka ferdig, og programmet vil fortsette under `while`-løkka sin kodeblokk.

Håndtering av løkker

Dersom vi ønsker MicroPython til å reagere på et knappetrykk bør vi plassere den i en løkke som går evig og sjekke om knappen `is_pressed`.

Å lage en løkke som går evig er enkelt:

```
while True
    # gjør noe
```

(Husk, `while` sjekker om noe er `True` for å finne ut om den skal kjøre kodeblokken eller ikke. Siden `True` åpenbart er `True` for alltid, vil du få en evig løkke!)

La oss lage et veldig enkelt elektronisk kjæledyr. Den er alltid trist om du ikke trykker `A`. Dersom du trykker `B` så dør dyret. (Jeg innser dette ikke er et veldig hyggelig spill, så kanskje du kan klare å forbedre 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()
```

Kan du se hvordan vi sjekker hvilke knapper som trykkes? Vi brukte `if`, `elif` (kort for "else if") og `else`. Disse kalles for *betingelser* og virker som følger:

```
if noe is True:
    # gjør en ting
elif en annen ting is True:
    # gjør en annen ting
else:
```

```
else:
    # gjør enda en ny ting.
```

Dette er overraskende likt engelsk!

Metoden `is_pressed` produserer bare to resultat: `True` eller `False`. Mens du holder inne knappen returnerer den `True`, ellers returnerer den `False`. Koden ovenfor sier følgende på norsk, "For alltid og alltid, dersom A trykkes vis et smilende ansikt, dersom B trykkes vis en hodeskalle i 1 sekund og bryt deretter løkka, ellers vis en trist ansikt". Vi bryter ut av løkka (stopp programmet fra å kjøre for alltid og alltid) ved å bruke `break`.

Lag spillet mindre tragisk

Kan du tenke på måter å gjøre spillet mindre tragisk på? Hvordan kan du sjekke om *begge* knappene er trykket ned samtidig?

Hint: Python har `and`, `or` og `not` logiske operatorer for å sjekke flere logiske påstander samtidig (ting som enten er `True` eller `False`).

Neste oppgave i samlingen er Python: Tilfeldig (../python_random/python_random_nb.html). Klikk videre for å fortsette gjennom samlingen.