

Skilpaddetekst

Introduksjon

I denne oppgaven skal vi skrive kode, slik at vi kan skrive stor tekst ved



Steg 1: Tekst på flere linjer

Vi har allerede lært at tekststrenger skrives slik:

```
tekst = "Hei, verden!"
```

Men hva hvis vi ønsker tekst på flere linjer? Da kan vi bruke tre `"""`-teg

```
tekst = """  
Dette er en  
tekst  
over  
mange linjer.
```



- ```
from turtle import *
```

Du skal nå se teksten printet ut i IDLE, men kan vi ikke få skilpa

- ```
LINES = TEXT.split('\n')
```

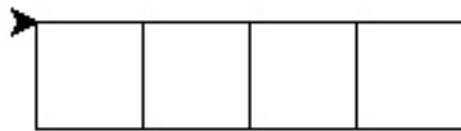
```
print(LINES)
```

Steg 2: Tegn med skilpadd

Vi ser at teksten over består av tegnene `\ | / _`. Det å lage disse hv

Hvis vi tenker oss at vi tegner hvert tegn i en tenkt, kvadratisk boks, r
så bør det være mulig å tegne teksten tegn for tegn. For å holde kont
skilpadden går inn i en ny boks, så må den peke mot høyre (øst), og v
tegne går den opp til hjørnet øverst til høyre, og peker til høyre. Den v

Tenk deg at boksene er i svart, og vi lager rød skrift. Da vil det se slik



✓ Sjekkliste

- ☐ Vi begynner med å legge til størrelsen på tegnene, slik:
(Pass på at denne koden ligger i samme fil som `TEXT`-variabelen)

```
SIZE = 15
```

`SIZE` er nå en variabel som inneholder størrelsen på boksen vår

- ☐ Vi lager en funksjon `underline` for å lage en understrek:

```
def underline():
    penup()

    # Beveg skilpadden ned til bunnen av boksen
    right(90)
    forward(SIZE)
    left(90)

    # tegn understreken
    pendown()
    forward(SIZE)
    penup()

    # beveg skilpadden opp til hjørnet øverst til høyre
    left(90)
    forward(SIZE)
    right(90)
```

- ☐ Kjør koden, og se hva som skjer:

```
underline()
```

- ☐ Hva om vi ønsker å lage 10 understreker?

```
for n in range(10):
    underline()
```


Det skal se slik ut, hvis du du ikke har feil i koden: _____

- ☐ Hva skjer hvis du endrer størrelsen på "boksen"? Prøv å endre p
eksempel 5 og 50)

Steg 3: Enda et tegn

La oss prøve å lage tegnet . Dette er rett og slett bare en rett strek

Sjekkliste

- ☐ Vi lager funksjonen `bar` for å tegne .

```
def bar():  
    penup()  
  
    # flytt til midten av boksen  
    forward(SIZE/2)  
    right(90)  
  
    # tegn en strek nedover  
    pendown()  
    forward(SIZE)  
    penup()  
  
    # flytt skilpadden til hjørnet øverst til høyre  
    left(180)  
    forward(SIZE)  
    right(90)  
    forward(SIZE/2)
```

- ☐ Endre `for`-løkka vi lagde tidligere til å inneholde dette

```
for n in range(10):  
    bar()
```

- ☐ Tegner skilpadden nå strekene på samme linje, slik som på bilde

Steg 4: Skilpaddetegn på f

Det er jo litt kjedelig om alle tegnene bare skal være på en linje, så hva gjør vi oss?

For å kunne lage en ny linje må funksjonen vite hvor mange tegn den skal tegne med et parameter - en variabel som vi kan gi til funksjonen når vi skal tegne.

Sjekkliste

☐ Skriv inn koden under:

```
def newline(lineLength):
    penup()

    right(90)
    forward(SIZE)
    right(90)

    forward(SIZE*lineLength)

    right(180)
```

Denne koden går først ned til linjen under, så går den tilbake be-
 forward med `SIZE*lineLength` som argumenter. `lineLength`
`SIZE` er hvor stort hvert tegn er - dermed må skilpadden flytte

- ☐ For å teste koden vår erstatter vi de tidligere `for`-løkkene med `for` (i denne filen):

```
for i in range(10):  
    underline()  
newline(10)  
for i in range(15):  
    bar()
```

Legg merke til at `newline` blir fortalt hvor mange tegn som ble skrevet på linjen under!

Dette skal se omtrent slik ut: 

Steg 5: Skilpadder på skrå

Nå har vi bare to tegn igjen å lage! Nemlig `/` og `\`. Disse tegnene må vi lage med streker av lengde `SIZE`, vi er nødt til å regne litt.

Hvis du går på ungdomsskolen har du kanskje lært at sammenhenger trekanter er slik $a^2 + b^2 = c^2$, det er dermed mulig å regne ut diagonalen i en trekant.

Her skal du bare få svaret og slippe å regne det ut selv. Diagonalen i en trekant betyr "opphøyd i" slik at $3**2$ blir 9 . Når du opphøyer noe med et tall. Dermed vil $9**0.5$ bli 3.0 . Dersom du lurer på hvordan dette fungerer, spør mattelæreren din.

Sjekkliste

- ☐ Koden for en 'slash' - `/` blir slik:

```
def slash():
```

```
penup()  
right(90)  
forward(SIZE)  
left(135)  
  
pendown()  
forward((2*SIZE**2)**0.5)  
penup()  
  
right(45)
```



Koden for en 'backslash' - \ blir slik:

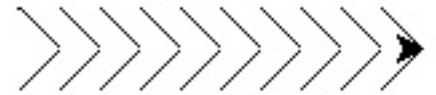
```
def backslash(): # \  
    penup()  
    right(45)  
  
    pendown()  
    forward((2*SIZE**2)**0.5)  
    penup()  
  
    left(135)  
    forward(SIZE)  
    right(90)
```



La oss endre på **for**-løkkene våre, og teste at koden blir korrekt

```
length = 10  
for i in range(length):  
    backslash()  
newline(length)  
for i in range(length):  
    slash()
```


Denne gangen skal mønsteret bli slik:



Nå er vi nesten ferdige! Bare litt igjen nå...

Steg 6: Skilpaddetekst

Vi trenger en funksjon for å skrive blanke tegn, og vi trenger å overse det enkleste.

Sjekkliste

- ☐ For å skrive blanke tegn, så må vi, enkelt og greit, bare bevege

```
def blank():  
    forward(SIZE)
```

- ☐ For å oversette fra teksttegn til funksjoner kommer vi til å bruke akkurat slik den fungerer. Vi "slår opp" noe i ordboka, og får noe funksjon tilbake igjen.

Først lager vi ordboka:

```
MOVES = {  
    " " : blank,  
    "_" : underline,  
    "/" : slash,  
    "|" : bar,  
    "\\" : backslash,  
    "(" : bar,
```

```
")" : bar,  
"" : blank,  
"," : blank  
}
```

Nå kan vi slå opp på tegnet `-` og få funksjonen `underline` tilbake

```
function = MOVES["_"]
```

Når vi så kaller `function`, vil den gjøre det samme som `underline`

```
function = MOVES["_"]  
function()
```

Dersom vi ønsker å sjekke om et tegn er i ordboka, så kan vi sjekke

```
if "_" in MOVES:  
    function = MOVES["_"]
```



Nå kan vi lage en ny funksjon, `create_text` som lager teksten

For å passe på at vi får plass til all teksten vår, ønsker vi å begynne ved hjelp av `setx` og `sety` som lar oss flytte skilpadden til den

```
def create_text():  
    penup()  
    setx(-window_width()/2)  
    sety(window_height()/2)  
  
    for line in LINES:  
        for char in line:  
            if char in MOVES:  
                move = MOVES[char]  
            else:  
                move = blank  
            move()  
        newline(len(line))
```

Som du kanskje ser, så har vi en `for`-løkke inni en annen `for`-løkke som går gjennom alle linjene i `LINES`, mens den innerste går igjennom alle tegnene om vi har en funksjon for tegnet, og hvis vi ikke har det så hoppes den stedenfor.

- ☐ For å kjøre funksjonen vår, lager vi en `main`-funksjon som sørger for at alt blir kjørt.

```
def main():  
    shape("turtle")  
  
    speed(11)  
    width(5)  
    create_text()  
  
main()
```

- ☐ Kjør koden og se resultatet ditt!

Kjøre koden uendelig mange ganger

Dersom du ønsker å kjøre koden uendelig mange ganger, kan du erstatte `for`-løkken med en `while`-løkke.

```
def main():  
    shape("turtle")  
  
    while True:  
        speed(11)  
        width(5)  
        create_text()  
        sleep(5)  
        reset()
```

For at dette skal fungere må vi importere `sleep`-funksjonen. Dette kan vi gjøre ved å legge til følgende linje i begynnelsen av koden:

```
from turtle import *  
from time import sleep
```

Skilpadden vil nå lage teksten, vente i fem sekunder (`sleep(5)`), og starter på nytt.

Utfordringer

Dersom du går tilbake til ordboka vi deklarte i MOVES-konstanter og `)` som `|`. Vi "jukset" også ved å bare tegne et blankt tegn istedenfor `|`.

Prøv å lage disse på egenhånd! `,` og `'` er lettest, for disse kan tegnes med `circle` som du har lært om sirkler!

Lag kode for hvert tegn i en egen funksjon, og husk og oppdatere funksjonen så du kan endre den.

```
" , ": blank
```

til

```
" , ": comma ,
```

der `comma` er navnet på funksjonen din.

Lisens: CC BY-SA 4.0 **Forfatter:** Ole Kristian Pedersen, Kodeklubb