Oblig 1 i INF120

Mats Hoem Olsen

19/2-2020

1 Oppgave 1 (mangkantkunst.py)

Oppgaven sier vi skal lage et program som tegner et polygram.

[1]:
```
import turtle #! ALDRI importer en hel modul uten at du vet hva som er inni den,␣
 ↪og jeg vet ikke...

user_input = int(input("Gi meg et tall større enn 0\n"))
if user_input <= 0:
        raise ValueError("input må være større enn 0")
angle = 360/user_input
length = 80 # 1 = 1 anus

for shit in range(user_input):
        turtle.forward(length)
        turtle.right(angle)

turtle.done()

"""
cmd: python.exe mangekantkunst.py
Gi meg et tall større enn 0
1
*turtle starter*
*tegner en linje*
*program termineres*
"""
```

```
Gi meg et tall større enn 0
 4
```

Dette ga oss en 4 siders sirkel (jeg tar nå en ekte sirkel til å ha uendelig mange sider)

2 Oppgave 2 (mangkantkunst2.py)

Oppgaven utvider Oppgave 1 ved å rotere figuren x ganger rundt sentrum.

[1]:
```
import turtle #! samme beskjed som i forrige oppgave...

user_input = int(input("gi et tall større enn 0\n"))
if user_input <= 0:
        raise ValueError("input må være større enn 0")
blad_tall = int(input("hvor mange blader vil du ha? n > 0\n"))
if blad_tall <= 0:
        raise ValueError("input må være større enn 0")

length = 80
penta = 360/user_input #vi sier navnet er informativt... Dette er veldig kort␣
 ↪kode, gled deg til oppgave 3
blad = 360/blad_tall

for b in range(blad_tall):
        for l in range(user_input):
                turtle.forward(length)
                turtle.right(penta)
        turtle.right(blad)
turtle.done()

"""
cmd: python.exe mangekantkunst2.py
gi et tall større enn 0
6
hvor mange blader vil du ha?
3
*turtle kjører*
*turtle tegner 3 6-kanter hvor sidene er rører hverandre.*
cmd: python.exe mangekantkunst2.py
gi et tall større enn 0
0
*viser en feilmelding*
*Program termineres*

"""
```

```
gi et tall større enn 0
 4
hvor mange blader vil du ha? n > 0
 6
```

3 Oppgave 3 (hemmelig_beskjed.py)

For å være ærlig leste jeg ikke oppgaven nøye... Så her får dere 25 tegn med egen compiler som lager runer. Les README.txt før dere kjører programet.

Vi starter med hva som skal skrives. Siden mitt program er automatiser så spiller det ikke noen rolle hva vi skriver siden LEXEN ordner opp for oss. Det er laget en kode som debugger output-en for retting.

[ ]:
```
temp = ""
for le in encodet:
        temp += chr(le)
print(temp)
```

Det lexen gir oss er unicode versjonen av bokstavene, men ikke tallene siden vikinger ikke hadde tall, de hadde noen form for tall skriving men for det meste skrev de ned tallene på ord form. Lexen min ordner opp i dette for oss. "temp += chr(le)" tar disse tallverdiene og konverterer dem til str som er "lesbare", gitt du har den nyeste versjonen av utf-8. Dette blir brukt i selve loopen.

[ ]:
```
turtle.penup() #Må ha for å sette posisjon
for letter in encodet:
        hight = hight - 15 if counter % limit == 0 else hight # Når man ønsker å␣
 ↪gå lavere
        counter = 0 if counter % limit == 0 else counter #Speeeeeeeeeed
        avdrag = 0 if counter % limit == 0 else avdrag # ved å la avdraget være␣
 ↪uforandret får vi skråtekst
        turtle.goto(-250 + 10*counter + 1 - avdrag,screen_hight / 2 + hight -␣
 ↪25) #Start på font
        letterl = chr(letter) #oversetter fra int til en bokstav, fjernes ved␣
 ↪mangel på lexen
        turtle.pendown()
        avdrag += eval("bm.tegn_{}()".format(letterl)) if letterl not in [" ",".
 ↪",",","[U+16EC]"] else eval("bm.tegn_{}()".format({",":"kom",".":"punk"," ":
 ↪"sp","[U+16EC]":"sp"}[letterl])) #tegner og opdaterer avdrag, SPEED CODE!
        turtle.setheading(0)
        counter += 1 #Oppdatering
turtle.hideturtle() #trengs ikke men bør være med for å lese siste font.
turtle.done()
```

Variablene hight, counter, og avdrag holder programmet sammen og organisert.

1. hight; holder styr på hvor høyt opp turtle skal tegen. Denne blir forandret jo lengere inn i beskjeden vi kommer.

2. counter;teller hvilke font vi er på og dermed oppdatere alle andre variabler.

3. avdrag; hver font er ikke like store og vil derfor være mindre eller større. Dette reflekteres i linje 51.

Istedet for å hente avdraget til hver font for så tegne valgte jeg heller at funksjonene returnerer avdraget etter de har tegnet fonten. Dette gjør at koden bli mindre og mer automatisert.

Her er et eksempel på en tegn modul:

[ ]:
```
def tegn_[U+16B1]():
        turtle.left(90)
        turtle.forward(8)
```

```
        turtle.right(90)
        turtle.forward(1)
        turtle.right(45)
        turtle.forward(3)
        turtle.right(90)
        turtle.forward(3)
        turtle.left(90)
        turtle.forward(7)
        turtle.penup()
        return 2 #avdrag
```

Som en kan se er meste parten av funksjonen et kall på turtle til å tegne utifra der den er. Return verdien beskriver hvor mye som er igjen av fonten. Siden mesteparten av tegnene vil være mindre eller lik 8 px brei vil verdiene for manglende plass være positiv mens ekstra bruk av plass vil være negativ.

4 Tillegg

Lex.py analyserer text som den tar inn med hensyn til ordboken som vi mater den. Den deler teksten opp inn i ord forså til bokstaver og tall. Tall blir håndtert av en ennen klasse som vil dekomponere tallet til bokstaver, men den kjører ikke 20- tallsystem som vi er vant med, den kjører 10-tallsystem. Dette vil si at istedet for tjue gir den oss "to ti" som er gramatisk riktig og logisk tiktig siden vi har 2 av 10 istedet for 1 av 20. Det andre systemet som kjører i "lex" er at den vil først se etter bokstaver som ikke støttes av ordboken vi ga den for så bytte de ut med relevante forslag også gitt av ordboken. Etter konverteringen vil den gå igjenom blokker av ordene for å finne "combo"-er, disse er spesielle i den form at selv om de kan erstasses bokstav for bokstav har vi ekvivalente tegn som vil minke tekst lengde. Det siste den vil gjøre er å konvertere bokstaver med relevante bokstaver i ordboken. Det vi får til slutt er en liste med int som representerer hvilke unicode bokstav det er. Mellomrom vil også konverteres i dette tilfelle siden vikinger ikke hadde vanlige mellomrom. Andre spesial tegn vil ikke erstates men istedet gjort om til int som resten i riktig unicode. Aksagner på bokstaver vil fjernes med modulen "unidecode" siden den vil gjøre arbeidet litt mer vanskelig, med mindre vi kan simpelten legge til aksagn på runer i dette tilfellet.