Repetisjon

Erfaren Python PDF

Introduksjon

I denne oppgaven skal vi repetere litt Python-syntaks, det er en god blanding av alle tingene du har lært i Python til nå.

Kodeblokker

I oppgavene er noen eksempler så korte at vi kan skrive de direkte inn i Python. Her er en kodeblokk som illustrerer et kort eksempel:

```
>>> 1 == 2
False
```

Her betyr `>>>` at Python er klar til å ta imot kode, `1 == 2` er koden, mens `False` er svaret.

I andre eksempler er det ikke noe svar, men en utskrift av tekst isteden:

```
>>> print("Hei!")
Hei!
```

Når vi ikke bruker `>>>` i kodeblokkene, er det fordi koden er flere linjer lang. Da er det bedre å bruke en fil:

```
for i in range(5):
  if i == 3:
    print(i)
  else:
    print(2*i)
```

Og når vi kjører denne filen i IDLE får vi utskriften:

```
>>>
0
2
4
3
8
```

Noen ganger har vi eksempler med input fra brukeren. Da vil teksten brukeren skriver være grønn, mens det programmet skriver ut vil være svart:

```
>>>
Hva heter du? Ada
Hei, Ada!
```

Input og output

input() og print()

Vi kan bruke `print()` når vi skal skrive ut tekst til brukeren. Koden etter `>>>` er kode vi skriver inn i f.eks. IDLE, og som kjøres med en gang.

```
>>> print("Hei, verden")
Hei, verden
```

`input()` brukes når du ønsker å la brukeren gi input til programmet ditt.

```
>>> number = input("Skriv inn et tall: ")
Skriv inn et tall: 15
>>> print("Du skrev inn: " + str(number))
Du skrev inn: 15
```

Skriv et program som spør om brukerens navn, og så skriver ut en hilsen til brukeren. Det kan for eksempel fungere slik:

```
>>>
Hei! Hva er navnet ditt?
Per
Hyggelig å treffe deg, Per!
```

Dette må du gjøre:

- [ ] Spør om brukerens navn.
- [ ] Lagre brukerens navn i en variabel.
- [ ] Skriv ut en hilsen til brukeren som inneholder navnet brukeren skrev inn.

if-elif-else

Vi bruker `if`, `elif` og `else` for å bestemme hva som skjer i et program. Etter `if` og `elif` kommer en test og deretter `:`, mens etter `else` kommer alltid `:` uten noen test. På linjen under `:` skrives kodeblokken som skal kjøres dersom testen er sann (`if` eller `elif`), eller dersom alle testene usanne (`else`).

Husk at du alltid må starte med en `if`-setning, og må ha alle `elif`-setningene før en `else`-blokk. Du *trenger ikke* å bruke verken `elif`-setninger eller `else`-blokk dersom du ikke ønsker det.

For eksempel slik:

```
name = "Ada"
if name == "Per":
    print("Per er et guttenavn")
elif name == "Ada":
    print("Ada er et jentenavn")
elif name == "Kim":
    print("Kim kan være både guttenavn og jentenavn.")
else:
    print("Jeg vet ikke om " + navn + " er en gutt eller ei jente.")
```

Du skal nå lage et program som finner ut hvilken aldersgruppe brukeren er i; barn, ungdom, voksen eller pensjonist. Du kan selv bestemme hvor aldersgrensene skal gå. Det kan for eksempel fungere slik:

```
>>>
Hei! Hva er alderen din?
77
Du er visst en pensjonist.
```

Det du trenger å gjøre er:

- [ ] Spør om brukerens alder.
- [ ] Lagre alderen til en variabel.
- [ ] Test om alderen er barn, ungdom, voksen eller pensjonist.
- [ ] Skriv ut hvilken aldersgruppe brukeren er i.

Løkker

for-løkker

`for`-løkker brukes når vi ønsker å gjøre ting flere ganger.

```
# print Hello three times
for i in range(3):
    print("Hello")
```

Da får vi ut:

```
>>>
Hello
Hello
Hello
```

Vi kan også bruke `for`-løkker når vi ønsker å gå igjennom ei liste:

```
# print all elements in the list food_list
food_list = ["eggs", "ham", "spiced ham", "jam"]
for food in food_list:
    print(food)
```

Dette programmet vil skrive ut:

```
>>>
eggs
ham
spiced ham
jam
```

Du skal nå lage ei liste med navn, og skrive ut alle navnene i lista. Resultatet kan se omtrent slik ut:

```
>>>
Per
Ada
Kim
```

Dette du må gjøre:

- [ ] Lag ei liste med navn.
- [ ] Bruk ei løkke for å gå igjennom lista med navn.
- [ ] Skriv ut hvert navn.

range()

`range()` lager en rekke med tall. Rekken kan brukes til å gjøre noe mange ganger med hjelp av en `for`- eller `while`-løkke. `range()` tar inn tre argumenter `start`, `stop`, `step`:

- `start` forteller hva vi skal telle fra.
- `stop` forteller hva vi skal telle til, merk at vi **ikke** teller med slutt-tallet.
- `step` forteller hvor store steg vi skal telle med. Vi kan for eksempel telle med steg på `2` eller steg på `100`.

Ettersom rekken lages etterhvert som man teller over den, må man konvertere rekken til en liste dersom vi ønsker å se tallene i rekken. Rekken konverteres til en liste med `list()`. Her er noen eksempler:

```
>>> list(range(1, 10, 1))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(200, 500))
[200, 201, 202, ..., 497, 498, 499]
>>> list(range(0, 50, 5))
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

`range()` kan brukes på mange måter, vi kan for eksempel gå igjennom den og summere alle tallene fra 1 til 100:

```
sum = 0
for number in range(1, 101):
    sum += number
print(sum)
```

while-løkker

`while`-løkker har mange ulike bruksområder. De kan for eksempel brukes når du vil kjøre kode inntil noe inntreffer:

```
word = ""
while word != "exit":
    print(word)
    word = input("Please write a word: ")
```

Den samme løkken kan også skrive slik:

```
while True:
    word = input("Please write a word: ")
    if word == "exit":
        break
    print(word)
```

Skriv et program som summerer alle tallene fra `1` til `100` ved hjelp av ei `while`-løkke. Pass på at du får `5050` som svar.

Dette må du gjøre:

- [ ] Lag en variabel som inneholder summen.
- [ ] Lag en tellevariabel som inneholder tallet du er kommet til.
- [ ] Så lenge tellevariabelen ikke er større enn 100:
  - [ ] Oppdater summen.
  - [ ] Inkrementer tellevariabelen din.

Funksjoner

Funksjoner lar oss gjenbruke kode, og er svært nyttig når vi skal programmere mer enn noen få linjer. En funksjon er på formen:

```
def greet(name):
    print("Hei, " + name + "!")

greet("Per")
```

Her har vi en funksjon med navn `greet`, som skriver ut en hilsen. `name` er et **parameter**, det vil si at `name` er en

variabel som funksjonen `greet` tar imot. Når vi **kaller** funksjonen `greet`, med `greet("Per")` er `"Per"` et **argument** til funksjonen. Et argument er den variabelen vi gir til funksjonen når vi kaller den.

Vi kan også lage funksjoner som returnerer en verdi. Det vil se slik ut:

```
def multiply(x, y):
    product = x*y
    return product
```

Nå skal vi lage en funksjon som adderer to tall. Test at funksjonen din fungerer som dette:

```
>>> sum = add(3, 4)
Fikk inn 3 og 4
>>> print(sum)
7
```

Dette må du gjøre:

- [ ] Definer en funksjon som tar inn to tall som parametre.
- [ ] Skriv ut tallene du fikk inn.
- [ ] Regn ut summen.
- [ ] Returner summen.

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