Kapittel 12

Objektorientert programmering

12.1 Innledning

Alle "ting" i Python er objekter. Begrepet objektorientert programmering (OOP) virker derfor som et veldig generelt begrep som dekker alt en kan gjøre i Python. Men begrepet objektorientert programmering brukes gjerne spesifikt for utvikling og bruk objekter som er *klasser* (Pythonsk: classes).

Hva er hensikten med klasser? Med klasser kan vi modularisere og strukturere programmer, hvilket er spesielt viktig i større programmeringsprosjekter. Klasser er på en måte et alternativ til funksjoner, som er beskrevet i kap. 5, men klasser er også mer enn funksjoner. Det kan være noe mer innfløkt å programmere klasser enn funksjoner, og i relativt enkle programmer er det nok greiere å lage funksjoner enn klasser.

12.2 Klasse, objekter, instanser, typer

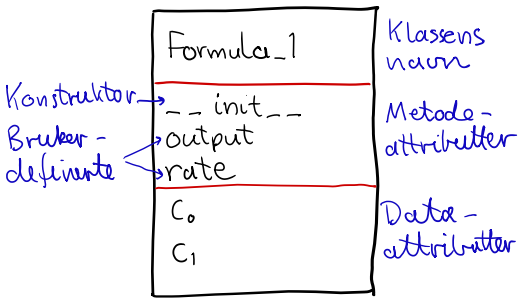
Klasser

En klasse er en programmert "enhet" som har et (klasse)navn. Klasser har to slags *attributter* (engelsk: attributes):

- *Dataattributter*, som er som variabler eller parametre som hører med til klassen.
- *Metodeattributter*, eller bare *metoder* (engelsk: methods), som er funksjoner som hører med til klassen og som utfører databehandling, f.eks. beregninger, på brukerdefinerte data eller inndata, som er data som brukeren oppgir ved bruk av metoden.

Så, dataattributtene *er* noe, mens metodeattributtene *gjør* noe.

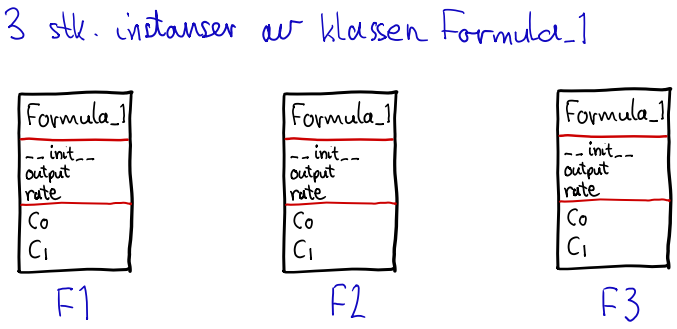
Figur 12.1 illustrerer disse begrepene. Betegnelsene i figuren er fra eksempel 12.1. Begrepet konstruktor beskrives i kap. 12.3.



Figur 12.1: Illustrasjon av klasse med metodeattributter og dataattributter, à la UML-diagram (Unified Modeling Language).

Objekter, instanser og typer

Objekter er "forekomster" av en spesifikk klasse. I stedet for objekt kan vi si *instans* (engelsk: instance). I boken vil vi stort sett bruke begrepet instans når det dreier seg om objekter som er forekomster av klasser vi selv har utviklet. Figur 12.2 illustrerer instanser.



Figur 12.2: Intansene F1, F2 og F3 av klassen Formula_1.

Objekter av en spesifikk klasse har denne klassen som *type*. Anta f.eks. at vi har laget en klasse med navn A. Et objekt av klassen A har da type A. Et ikke-Pythonsk eksempel er at du er et objekt av klassen menneske. Du som objekt har type menneske.

Python har også en rekke innebygde typer, som list, tuple, float, m.m. For eksempel vil følgende kode:

```
x = 1.2
type(x)
```

gi som svar at objektet eller instansen x er av type:

```
float
```

12.3 Hvordan lage en klasse og bruke instanser av klassen

Eksempel 12.1 demonstrerer hvordan vi kan lage en klasse og bruke instanser av klassen.[1]

Eksempel 12.1 *En klasse for en lineær formel*

Klassen vi skal lage, skal implementere følgende lineære formel:

$$y = c_1 x + c_0 \tag{12.1}$$

der c_0 og c_1 er parametre, x er innverdien (eller uavhengig variabel), og y er utverdi (eller avhengig variabel eller utverdien).

Program 12.1 gjør tre ting:

- Lager klassen Formula_1. (Det er vanlig å bruke stor forbokstav i klassenavn.)
- Lager en instansen F1 av klassen Formula_1.
- Tar i bruk instansen F1 på flere måter, nemlig kall av instansens metoder (eller metodeattributter) og avlesing av en av instansens dataattributter, som forklart i kommentarene til programmet.

http://techteach.no/python/files/prog_with_class_Formula_1.py

Listing 12.1: prog_with_class_Formula_1

```
# %% Import of e.g. numpy placed here if needed

# import numpy as np

# %% Defintion of class Formula_1:

class Formula_1(object):

    def __init__(self, c0, c1):
        self.c0 = c0
        self.c1 = c1

    def output(self, x):
        c0 = self.c0
        c1 = self.c1
        y = c1*x + c0
        return y

    def rate(self):
        r = self.c1
        return r
```

[1] Eksempelet er inspirert av et av eksemplene om objektorientert programmering i (Langtangen 2016).

```
# %% Application of class Formula_1:

# Defining variables used in calling instance of class:
k0 = 0
k1 = 1
x = 5

# Creating instance here named F1:
F1 = Formula_1(k0, k1)

# Calling method named output of F1:
print('Output value y1:', F1.output(x))

# Calling method named rate of F1:
print('Calculating rate coeff of formula:', F1.rate())

# Showing data attribute c0 of F1:
print('Data attribute c0: ', F1.c0)

# Showing type of F1:
print('Instance type of F1: ', type(F1))
```

Resultatet av å kjøre program 12.1 er:

```
Output value: y1 = 5
Calculating rate coeff of formula: 1
Data attribute c0 = 0
Instance type of F1 = <class '__main__.Formula_1'>
```

Kommentarer til program 12.1:

1. Klassen Formula_1 er definert med nøkkelordet class øverst i programmet. "object" skal stå i parentesen (i tidligere Python-versjoner kunne en droppe "object").

2. Kodedelen som starter med det innrykkede uttykket def __init__(self, c0, c1), er klassens constructor, som vi på norsk godt kan kalle konstruktor[2]. De lange understrekene fås med to stk. understreker på tastaturet. Klasser skal ha en konstruktor. Konstruktorens hensikt er å definere variabler som brukes internt i koden som definerer klassen. Nøkkelordet def indikerer at __init__ er en funksjon (vi bruker jo def for å lage funksjoner, jf. kap. 5), men i forbindelse med klasser sier vi i stedet metoder (engelsk: methods).

3. Innargumentene c0 og c1 i definisjonslinjen def __init__(self, c0, c1) er klassens dataattributter. Merk at kodelinjene self.c0 = c0 og self.c1 = c1 er innrykket i forhold til def-nøkkelordet.

4. Nøkkelordet self representerer litt enkelt sagt den aktuelle instansen av klassen når vi lager en instans. Vi bruker imidlertid ikke self når vi faktisk lager en instans; vi bruker self bare i koden der vi definerer klassen.

[2]Noen sier konstruktør.

5. Metoden som vi har gitt navn output, beregner og returnerer formelens utverdi. Merk at self skal være med i metodens argumentliste. I denne argumentlisten inngår også innverdien x. Men parametrene c0 og c1 inngår ikke i argumentlisten. Vi trenger ikke angi dem der; de er tilgjengelig som en slags globale variabler for alle metodene som inngår i definisjonen av klassen.

6. I output-metoden har vi brukt koden:
   c0 = self.c0
   c1 = self.c1
   y = c1*x + c0
   Alternativt kunne vi her ha brukt koden:
   y = self.c1*x + self.c0
   Det er en smakssak hvilken av disse to alternativene du vil velge.

7. Metoden kalt rate returnerer raten i formelen y = c1*x + c0, dvs. stigningstallet (verdien av c1) for linjen som formelen framstiller.

8. Denne klassen inneholder to stk. metoder, nemlig output og rate, i tillegg til konstruktoren. Generelt kan du legge inn så mange metoder du ønsker i en klasse.

9. Nedenfor koden som definerer klassen Formula_1, følger kode som viser hvordan klassen kan tas i bruk. Først defineres variablene k0, k1 og x1 med verdier som angitt.

10. Koden F1 = Formula_1(k0, k1) lager en instans (eller objekt) med vårt selvvalgte navn F1 av klassen Formula_1. F1 er en instans av formelen (12.1) med parameterverdier c0 = k0 = 0 og c1 = k1 = 1. I en annen sammenheng ønsker vi kanskje å lage en annen instans med helt andre verdier av c0 og c1.

11. Koden F1.output(x1) kaller (eller utfører) metoden output i instansen F1 med innargument x1, som er gitt verdi 5 tidligere i programmet. Verdien som dette kallet returnerer, vises i konsollen med print-funksjonen. Merk skrivemåten: Instansen står foran punktum, og metodeattributtet står etter punktum.

12. Koden F1.rate() kaller (eller utfører) metoden rate i instansen F1 uten innargument. Dette kallet returnerer en verdi (som blir vist i konsollen med print-funksjonen).

13. Koden F1.c1 kaller ikke noen metode i F1, men returnerer verdien av dataattributtet c1 (som blir vist i konsollen med print-funksjonen). Merk skrivemåten: Instansen foran og dataattributtet etter punktum.

14. Koden type(F1) returnerer denne typen av F1 (og viser typen i konsollen med print-funksjonen).

15. Det er faktisk ikke nødvendig å lage en instans med eget navn (her F1). Eksempelvis vil følgende kode også fungere:
    F1 = Formula_1(k0, k1).output(x1).
    Instansen her er Formula_1(k0, k1). Den er en ikke-navngitt instans.

[Slutt på eksempel 12.1]

12.4 Hvordan legge definisjonen av en klasse i en modul

I eksempel 12.1 la vi inn både koden for definisjon av klassen og koden for bruk av instanser av klassen i ett og samme program. Alternativt kan vi legge koden for definisjon av klassen inn i en modul (dvs. et eget py-skript) og så importere modulen med den aktuelle klassen inn til hovedprogrammet (der klassen skal brukes). Dette er samme opplegg som da vi definerte funksjoner i moduler og importerte modulen inkl. funksjonene til hovedprogrammet, jf. kap. 5.3.

Eksempel 12.2 demonstrerer bruk av klasser som er definert i moduler.

Eksempel 12.2 *Klasse i modul*

Vi tar utgangspunkt i eksempel 12.1. Men vi skal nå legge koden for definisjonen av klassen Formula_1 inn i en modulen module_with_def_Formula_1.py, mens vi legger koden for å lage og bruke instansen F1 av klassen i en annen py-fil, som blir hovedprogrammet.

Modulen 12.2 inneholder klassedefinisjonen.

http://techteach.no/python/files/module_def_Formula_1.py

Listing 12.2: module_def_Formula_1

```
# %% Import of e.g. numpy placed here if needed

# import numpy as np

# %% Defintion of class Formula_1:

class Formula_1(object):

    def __init__(self, c0, c1):
        self.c0 = c0
        self.c1 = c1

    def output(self, x):
        c0 = self.c0
        c1 = self.c1
        y = c1*x + c0
        return y

    def rate(self):
        r = self.c1
        return r
```

Program 12.3 er hovedprogrammet.

http://techteach.no/python/files/prog_using_imported_class_Formula_1.py

Listing 12.3: prog_using_imported_class_Formula_1

```
# %% Import of packages (e.g. numpy) and your own classes:

from module_def_Formula_1 import Formula_1

# %% Application of class Formula_1:

# Defining variables used in calling instance of class:
k0 = 0
k1 = 1
x = 5

# Creating instance here named F1:
F1 = Formula_1(k0, k1)

# Calling method named output of F1:
print('Output value y1:', F1.output(x))

# Calling method named rate of F1:
print('Calculating rate coeff of formula:', F1.rate())

# Showing data attribute c0 of F1:
print('Data attribute c0: ', F1.c0)

# Showing type of F1:
print('Instance type of F1: ', type(F1))
```

Resultatet av å kjøre program 12.3 er som i eksempel 12.1. Resultatet gjengis allikevel her for oversiktens skyld:

```
Reloaded modules: class_def_temp_conv
Output value y1: 5
Calculating rate coeff of formula: 1
Data attribute c0: 0
Instance type of F1: <class 'module_def_Formula_1.Formula_1'>
```

Kommentarer til program 12.2:

1. Koden er stort sett som i program 12.1 i eksempel 12.2.
2. Hvis noe av koden i definisjonen av klassen krever f.eks. numpy, må du importere numpy med koden import numpy as np lagt inn f.eks. øverst i filen, som indikert i program 12.2.

Kommentarer til program 12.3:

1. Koden
   from module_def_Formula_1 import Formula_1
   importerer klassen Formula_1.

2. Selv om det ikke er vist i dette programmet, kan du importere Formula_ 1 med et kallenavn (engelsk: alias), f.eks. slik:
   from module_def_Formula_1 import Formula_1 as alias_F1
   der alias_F1 er kallenavnet på den importerte klassen Formula_1. Du kan så bruke alias_F1 i stedet for Formula_1 i programmet.

3. Resten av koden, som lager instansen F1 av klassen Formula_1, er identisk med koden i program 12.1.

[Slutt på eksempel 12.2]

12.5 Hvordan lage nye klasser med arv fra eksisterende klasser

Vi kan lage en ny klasse, la oss her kalle den K2, basert på en *arv* (engelsk: heritage) eller kopi av en opprinnelig klasse, K1, og med utvidelser av attributtene til K1. Vi kan si at K1 er foreldreklassen (engelsk: parent class) eller superklassen, mens K2 er barneklassen (child class) eller subklasse. Slike arvede klasser utvides gjerne med *ny funksjonalitet*. Med andre ord, avkommet representerer en videreutvikling av opphavet!

Eksempel 12.3 demonstrerer hvordan vi kan lage en subklasse fra en superklasse og bruke en instans av subklassen.

Eksempel 12.3 *Subklasse basert på arv fra superklasse*

Vi tar utgangspunkt i program 12.1 i eksempel 12.1. Superklassen skal være klassen Formula_1 i program 12.1. Vi skal nå lage en subklasse med navn Formula_2 som en arv fra Formula_1-klassen. Formula_2 skal implementere følgende formel:

$$y_2 = c_2x^2 + y_1 = c_2x^2 + c_1x + c_0 \tag{12.2}$$

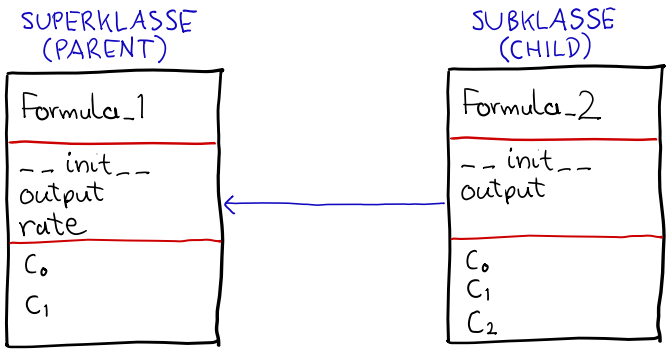
der y_1 beregnes med klassen Formula_1. Subklasse Formula_2 er altså en utvidelse av superklasse Formula_1.[3] Figur 12.3 illustrerer de to klassene.

Program 12.4 gjør følgende:

- Implementerer superklassen Formula_1 og subklasse Formula_2.
- Lager og bruker instansen F2 av Formula_2.

http://techteach.no/python/files/prog_using_inherited_class_formula.py

[3]Det kan nok høres litt rart ut at en subklasse har utvidet funksjonalitet sammenliknet med en superklasse, men slik er terminologien.



Figur 12.3: Superklasse (parent) Formula_1 og subklasse (child) Formula_2.

Listing 12.4: prog_using_inherited_class

```
# %% Import of e.g. numpy placed here if needed

# import numpy as np

# %% Defintion of class Formula_1:

class Formula_1(object):

    def __init__(self, c0, c1):
        self.c0 = c0
        self.c1 = c1

    def output(self, x):
        c0 = self.c0
        c1 = self.c1
        y = c1*x + c0
        return y

    def rate(self):
        r = self.c1
        return r


class Formula_2(Formula_1):

    def __init__(self, c0, c1, c2):
        Formula_1.__init__(self, c0, c1)
        self.c2 = c2

    def output(self, x):
        y1 = Formula_1.output(self, x)
        y2 = self.c2*x**2 + y1
        return y2

# %% Application of class Formula_1:

# Defining variables used in calling instance F2:
```

```
k0 = 0
k1 = 1
k2 = 2
x = 5

# Creating instance here named F2:
F2 = Formula_2(k0, k1, k2)

# Calling method named output of F2:
print('Output value of F2:', F2.output(x))

# Showing data attribute c2 of F2:
print('Parameter c2 of F2:', F2.c2)

# Showing type of F2:
print('Instance type of F2:', type(F2))
```

Resultatet av å kjøre program 12.4 med følgende argumenter til dataattributtene: k0 = 0, k1 = 2, k2 = 2 og innargument (uavhengig variabel) x = 5:

```
Output value of F2: 55
Parameter c2 of F2: 2
Instance type of F2: <class '__main__.Formula_2'>
```

Kommentarer til program 12.4:

1. Kodedelen som starter med class Formula_1(object) definerer superklassen Formula_1.

2. Koden class Formula_2(Formula_1) lager subklassen Formula_2 som en arv av Formula_1.

3. Koden Formula_1.__init__(self, c0, c1) sørger for at konstruktoren til Formula_2 inngår i konstruktoren til Formula_2.

4. Kodedelen
   y1 = Formula_1.output(self, x)
   y2 = self.c2*x**2 + y1
   i output-metoden i Formula_2 sørger for at beregningen av y2 bygger på (eller utvider) beregningen av y1 som utføres av output-metoden i Formula_1-klassen.

5. Koden F2 = Formula_2(k0, k1, k2) lager en instans med navn F2 av klassen Formula_2.

6. De tre print-kallene demonstrerer ulike måter å bruke instansen F2 på (vi antar at kodedelene er greie å forstå).

[Slutt på eksempel 12.3]

12.6 Oppgaver til kapittel 12

Oppgave 12.1 *Bruk av instans av klasse*

Denne oppgaven er basert på program 12.1, som er beskrevet i Eksempel 12.1.

Kjør programmet med følgende parametre (som brukes som dataattributter) til klassens instans (i samme rekkefølge som i programmet): 10 og 20, og med innargument (uavhengig variabel) 30. Hva blir resultatene?

Oppgave 12.2 *Klasse for omregning fra celsius til fahrenheit og kelvin*

Lag et program som definerer en klasse med navn f.eks. Temp_conv for omregning av oppgitt celsius-verdi til tilsvarende verdier i fahrenheit og kelvin. Det er ikke nødvendig å gi denne klassen dataattributter, kun metodeattributter. Du kan derfor droppe konstruktoren i koden som definerer denne klassen. Til info: En instans av en klasse uten konstruktor kalles med tom parentes, f.eks. instans1().

Lag og bruk en instans av klassen med innargument 100 °C. Prøv da både med navngitt instans og med ikke-navngitt instans (jf. den aller siste kommentaren til programmet i eksempel 12.1). Hva blir fahrenheit- og kelvin-verdiene?

Oppgave 12.3 *Klassedfinisjonen i en modul*

Se program 12.5, som er en løsning på oppgave 12.2. Flytt nå klassedefinisjonen til en modul (et py-skript), og lag et hovedprogram som importerer klassen. Kjør hovedprogrammet. Blir fahrenheit- og kelvin-verdiene som med program 12.5? (Ja, forhåpentligvis.)

Oppgave 12.4 *Arv av klasse*

Denne oppgaven er basert på program 12.4, som er beskrevet i eksempel 12.3, der formel (12.2) er implementert i en klasse kalt Formula_2.

Modifiser program 12.4 ved at du lager en ny klasse, som du kan kalle Formula_3, som skal implementere følgende formel:

$$y_3 = c_3x^3 + c_2x^2 + y_1 = c_2x^2 + c_1x + c_0 \tag{12.3}$$

som er en utvidelse av formel (12.2) med det additive leddet c_3x^3. Klasse Formula_3 skal være basert på arv fra klasse Formula_2.

Kjør ditt ferdige program med følgende innargumenter (til klassens dataattributter): k0 = 0, k1 = 2, k2 = 2, k3 = 3 og følgende innargument (uavhengig variabel) x = 5. Hva blir resultatet (y3)?

12.7 Løsninger til kapittel 12

Løsning til oppgave 12.1

Program 12.1 kjøres med følgende variabelverdier: k0 = 10, k1 = 20 og x1 = 30.

Resultat:

Output value: y1 = 610 Calculating rate coeff of formula: 20 Data attribute c0 = 10 Instance type of F1 = <class '__main__.Formula_1'>

Løsning til oppgave 12.2

Program 12.5 er en løsning.

http://techteach.no/python/files/losning_temp_conv.py

Listing 12.5: losning_temp_conv

```
# %% Defintion of class Temp_conv:

class Temp_conv(object):

    def c2f(self, c):
        f = c*(9/5) + 32
        return f

    def c2k(self, c):
        k = c + 273.15
        return k


# %% Application of class Temp_Conversion:

# Defining variables used in calling instance of class:
c = 100

print('Results with named instance:')

# Creating named instance:
my_temp_conv = Temp_conv()

print('Fahrenheit value =', my_temp_conv.c2f(c))
print('Kelvin value =', my_temp_conv.c2k(c))

print('--------------------------------')
print('Results with unnamed instance:')
```

```
print('Fahrenheit value =', Temp_conv().c2f(c))
print('Kelvin value =', Temp_conv().c2k(c))
```

Resultat av kjøring av program 12.5:

```
Results with named instance:
Fahrenheit value = 212.0
Kelvin value = 373.15
------------------------------
Results with unnamed instance:
Fahrenheit value = 212.0
Kelvin value = 373.15
```

Løsning til oppgave 12.3

Modulen (skriptet) 12.6 inneholder klassedefinisjonen.

http://techteach.no/python/files/module_def_class_temp_conv.py

Listing 12.6: module_def_class_temp_conv

```
# %% Defintion of class Temp_conv:

class Temp_conv(object):

    def c2f(self, c):
        f = c*(9/5) + 32
        return f

    def c2k(self, c):
        k = c + 273.15
        return k
```

Program 12.7 er hovedprogrammet.

http://techteach.no/python/files/main_prog_temp_conv.py

Listing 12.7: main_prog_temp_conv

```
# %% Import of class Temp_conv:

from module_def_class_temp_conv import Temp_conv


# %% Application of class Temp_Conversion:

# Defining variables used in calling instance of class:
c = 100
```

```
print('Results with named instance:')

# Creating named instance:
my_temp_conv = Temp_conv()

print('Fahrenheit value =', my_temp_conv.c2f(c))
print('Kelvin value =', my_temp_conv.c2k(c))

print('--------------------------------')
print('Results with unnamed instance:')

print('Fahrenheit value =', Temp_conv().c2f(c))
print('Kelvin value =', Temp_conv().c2k(c))
```

Resultat av kjøring av program 12.7:

```
Results with named instance:
Fahrenheit value = 212.0
Kelvin value = 373.15
--------------------------------
Results with unnamed instance:
Fahrenheit value = 212.0
Kelvin value = 373.15
```

som er samme resultat som med program 12.5 i løsningen til oppgave 12.2.

Løsning til oppgave 12.4

Program 12.8 er en løsning.

http://techteach.no/python/files/losn_inherited_class.py

Listing 12.8: losn_inherited_class

```
# %% Import of e.g. numpy placed here if needed

# import numpy as np

# %% Defintion of class Formula_1:

class Formula_1(object):

    def __init__(self, c0, c1):
        self.c0 = c0
        self.c1 = c1

    def output(self, x):
        c0 = self.c0
        c1 = self.c1
```

```
        y = c1*x + c0
        return y

    def rate(self):
        r = self.c1
        return r


class Formula_2(Formula_1):

    def __init__(self, c0, c1, c2):
        Formula_1.__init__(self, c0, c1)
        self.c2 = c2

    def output(self, x):
        y1 = Formula_1.output(self, x)
        y2 = self.c2*x**2 + y1
        return y2


class Formula_3(Formula_2):

    def __init__(self, c0, c1, c2, c3):
        Formula_2.__init__(self, c0, c1, c2)
        self.c3 = c3

    def output(self, x):
        y2 = Formula_2.output(self, x)
        y3 = self.c3*x**3 + y2
        return y3

# %% Application of class Formula_1:

# Defining variables used in calling instance F3:
k0 = 0
k1 = 1
k2 = 2
k3 = 3
x = 5

# Creating instance here named F3:
F3 = Formula_3(k0, k1, k2, k3)

# Calling method named output of F3:
print('y3 = output value of F3:', F3.output(x))
```

Resultat av kjøring av program 12.8:

```
y3 = output value of F3: 430
```