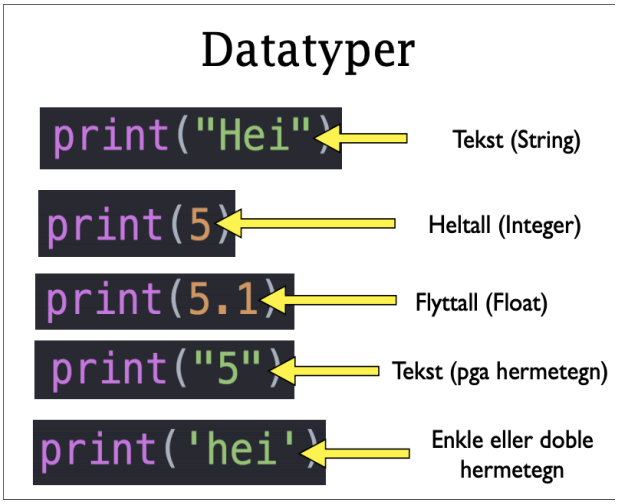
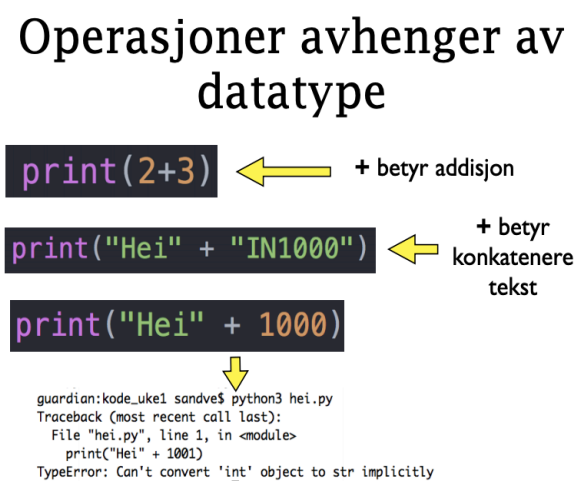
IN1000

Innføring i objektorientert programmering

Innholdsfortegnelse

Datatyper





Funksjoner

Function	Description
bool()	Returns the boolean value of the specified object
float()	Returns a floating point number
input()	Allowing user input
int()	Returns an integer number
len()	Returns the length of an object
list()	Returns a list

open()	Opens a file and returns a file object
print()	Prints to the standard output device
range()	Returns a sequence of numbers, starting from 0 and increments by 1 (by default)
str()	Returns a string object
tuple()	Returns a tuple
type()	Returns the type of an object

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
count()	Returns the number of times a specified value occurs in a string
format()	Formats specified values in a string

index()	Searches the string for a specified value and returns the position of where it was found
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
upper()	Converts a string into upper case

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value

index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list

Method	Description
clear()	Removes all the elements from the dictionary
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair

Method	Description
and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs

False	Boolean value, result of comparison operations
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module
in	To check if a value is present in a list, tuple, etc.
is	To test if two variables are equal
None	Represents a null value
nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator

pass	A null statement, a statement that will do nothing
return	To exit a function and return a value
True	Boolean value, result of comparison operations
while	To create a while loop

Lister

How to create a list?

In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.

It can have any number of items and they may be of different types (integer, float, string etc.).

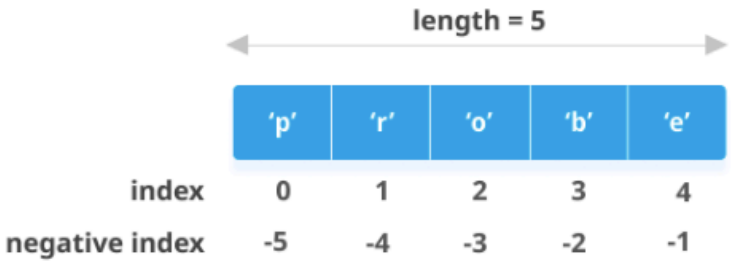
```
# empty list
my_list = []

# list of integers
my_list = [1, 2, 3]

# list with mixed datatypes
my_list = [1, "Hello", 3.4]
```

Also, a list can even have another list as an item. This is called nested list.

```
# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```



How to delete or remove elements from a list?

We can delete one or more items from a list using the keyword `del`. It can even delete the list entirely.

```
my_list = ['p','r','o','b','l','e','m']

# delete one item
del my_list[2]

# Output: ['p', 'r', 'b', 'l', 'e', 'm']
print(my_list)

# delete multiple items
del my_list[1:5]

# Output: ['p', 'm']
print(my_list)

# delete entire list
del my_list

# Error: List not defined
print(my_list)
```

Run Code »

We can use `remove()` method to remove the given item or `pop()` method to remove an item at the given index.

The `pop()` method removes and returns the last item if index is not provided. This helps us implement lists as stacks (first in, last out data structure).

We can also use the `clear()` method to empty a list.

```
my_list = ['p','r','o','b','l','e','m']
my_list.remove('p')

# Output: ['r', 'o', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'o'
print(my_list.pop(1))

# Output: ['r', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'm'
print(my_list.pop())

# Output: ['r', 'b', 'l', 'e']
print(my_list)

my_list.clear()

# Output: []
print(my_list)
```

Run Code »

Finally, we can also delete items in a list by assigning an empty list to a slice of elements.

Ordbok

How to create a dictionary?

Creating a dictionary is as simple as placing items inside curly braces {} separated by comma.

An item has a key and the corresponding value expressed as a pair, key: value.

While values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.

```
# empty dictionary
my_dict = {}

# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()
my_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```

How to access elements from a dictionary?

While indexing is used with other container types to access values, dictionary uses keys. Key can be used either inside square brackets or with the `get()` method.

The difference while using `get()` is that it returns `None` instead of `KeyError`, if the key is not found.

script.py IPython Shell

```
my_dict = {'name':'Jack', 'age': 26}

# Output: Jack
print(my_dict['name'])

# Output: 26
print(my_dict.get('age'))

# Trying to access keys which doesn't exist throws error
# my_dict.get('address')
# my_dict['address']
```

Run

Powered by DataCamp

When you run the program, the output will be:

```
Jack
26
```

How to change or add elements in a dictionary?

Dictionary are mutable. We can add new items or change the value of existing items using assignment operator.

If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

script.py IPython Shell

```
my_dict = {'name':'Jack', 'age': 26}

# update value
my_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}
print(my_dict)

# add item
my_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}
print(my_dict)
```

Run

Powered by DataCamp

When you run the program, the output will be:

```
{'name': 'Jack', 'age': 27}
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
```

How to delete or remove elements from a dictionary?

We can remove a particular item in a dictionary by using the method `pop()`. This method removes as item with the provided key and returns the value.

The method, `popitem()` can be used to remove and return an arbitrary item (key, value) form the dictionary. All the items can be removed at once using the `clear()` method.

We can also use the `del` keyword to remove individual items or the entire dictionary itself.

script.py IPython Shell

```
# create a dictionary
squares = {1:1, 2:4, 3:9, 4:16, 5:25}

# remove a particular item
# Output: 16
print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}
print(squares)

# remove an arbitrary item
# Output: (1, 1)
print(squares.popitem())

# Output: {2: 4, 3: 9, 5: 25}
print(squares)

# delete a particular item
del squares[5]

```

Objekter og klasser

An object has two characteristics:

- attributes
- behavior

Let's take an example:

Parrot is an object,

- name, age, color are attributes
- singing, dancing are behavior

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

In Python, the concept of OOP follows some basic principles:

Inheritance	A process of using details from a new class without modifying existing class.
Encapsulation	Hiding the private details of a class from other objects.
Polymorphism	A concept of using common operation in different ways for different data input.

Example 1: Creating Class and Object in Python

script.py IPython Shell

```
class Parrot:

    # class attribute
    species = "bird"

    # instance attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age

# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))

# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

Run

Methods

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object.

Example 2 : Creating Methods in Python

script.py IPython Shell

```
class Parrot:

    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def sing(self, song):
        return "{} sings {}".format(self.name, song)

    def dance(self):
        return "{} is now dancing".format(self.name)

# instantiate the object
blu = Parrot("Blu", 10)

# call our instance methods
print(blu.sing("'Happy'"))
print(blu.dance())
```

Run

Løkker

The "for" loop

For loops iterate over a given sequence. Here is an example:

script.py IPython Shell

```
primes = [2, 3, 5, 7]
for prime in primes:
    print(prime)
```

```
In [1]:
```

Run

Powered by DataCamp

For loops can iterate over a sequence of numbers using the "range" and "xrange" functions. The difference between range and xrange is that the range function returns a new list with numbers of that specified range, whereas xrange returns an iterator, which is more efficient. (Python 3 uses the range function, which acts like xrange). Note that the range function is zero based.

script.py IPython Shell

```
# Prints out the numbers 0,1,2,3,4
for x in range(5):
    print(x)

# Prints out 3,4,5
for x in range(3, 6):
    print(x)

# Prints out 3,5,7
for x in range(3, 8, 2):
    print(x)
```

```
In [1]:
```

Run

"while" loops

While loops repeat as long as a certain boolean condition is met. For example:

script.py

```
# Prints out 0,1,2,3,4

count = 0
while count < 5:
    print(count)
    count += 1  # This is the same as count = count + 1
```

IPython Shell

```
In [1]:
```

Run

Powered by DataCamp

"break" and "continue" statements

break is used to exit a for loop or a while loop, whereas **continue** is used to skip the current block, and return to the "for" or "while" statement. A few examples:

script.py

```
# Prints out 0,1,2,3,4

count = 0
while True:
    print(count)
    count += 1
    if count >= 5:
        break

# Prints out only odd numbers - 1,3,5,7,9
for x in range(10):
    # Check if x is even
    if x % 2 == 0:
        continue
```

IPython Shell

```
In [1]:
```

Mine egne oppgaver

Parametere og returverdier

```
'''Programmet danner en funkjson som adderer to tall og returnerer summen.'''

def adder(tall1, tall2):
    sum = tall1 + tall2
    return sum

'''Funkjsonen kalles to ganger med forskjellige arguemnter, og printer ut resultatet.'''

print("Summen av 13 og 22 er" , str(adder(13, 22)),".")
print("Summen av 69 og 420 er" , str(adder(69, 420)),".")

'''Programmet ber bruekren om tekst og bokstav som blir lagret som variabler.
Deretter dannes en funksjon med disse variabelene som parametere.'''

minTekst = input("Skriv en kort setning :")
minBokstav = input("Skriv en bokstav :")

def tellForekomst(minTekst, minBokstav):
    forekomst = minTekst.count(minBokstav)
    return forekomst

'''Programmet teller hvor mange ganger bokstaven forekommer i setningen ved bruk
av .count og returnerer resultatet.'''

print("Bokstaven forekommer", str(tellForekomst(minTekst, minBokstav)), "gang(er) i setningen.")
```

Regning med løkker (While-løkke, For-løkke)

```
'''Programmet tar inn input fra brukeren helt til verdien 0 blir oppgitt, dette
skjer ved bruk av en while loop. Tallen legges inn i en tom liste.'''

tall = int(input("Skriv inn et tall :"))
liste = []
while tall != 0:
    liste.append(tall)
    tall = int(input("Skriv inn et tall :"))

'''Programmet printer så ut alle elementene i listen ved bruk av en for loop.'''

for x in liste:
    print(x)

'''For loopen går gjennom listen og adderer, oppdaterer listen, adderer for alle
elementene i listen før summen printes.'''

minSum = 0
for x in liste:
    minSum = minSum + x
print("Summen av listen er :", minSum)

'''To nesten identiske for loops går igjennom alle elementene i listen til de ha funnet
den laveste og høyeste tallverdien, og printer denne.'''

minst = liste[0]
for x in liste:
    if x < minst:
        minst = x
print("Det minste tallet i listen er :", minst)

storst = liste[0]
for x in liste:
    if x > storst:
        storst = x
print("Det største tallet i listen er :", storst)
```

Reiseplan (Lister, nøstede lister, For-løkke)

```
'''Først oppretter jeg fire lister, tre tomme, og en siste som inneholder
de tre første listene.'''
steder = []
plagg = []
dato = []
reiseplan = [steder, plagg, dato]

'''Så oppretter jeg en loop der brukeren kan legge inn informasjon i listene 5 ganger.'''

for i in range(5):
    steder.append(input("Skriv inn et reisemål: "))
    plagg.append(input("Skriv inn et klesplagg: "))
    dato.append(str(input("Skriv inn en dato: ")))

'''Her printer programmet ut listen "reiseplan" ved bruk av en for loop.'''
for i in reiseplan:
    print(i)

'''Bukeren får mulighet til å få ut informasjon fra listene. Først hvilken av de
tre listene, deretter hvilket element i listen'''

i1 = int(input("Skriv inn et tall fra og med 0 til og med " + str(len(reiseplan)-1) + " :"))
i2 = int(input("Skriv inn et tall fra og med 0 til og med " + str(len(reiseplan[i1])-1) + " :"))
if i1 <= len(reiseplan)-1 and i2 <= len(reiseplan[i1])-1:
    print(reiseplan[i1][i2])
else:
    print("Ugyldig input!")

'''If sjekken gir brukeren tilbakemelding om det blir oppgitt input som ikke stemmer
overens med listene.'''
```

Ordtelling (Ordbok, .count())

```
'''Først lager programmet en funkjson som teller antall bokstaver i et ord.
Deretter kaller programmet på funksjonen.'''

def funksjon1(tekst):
    return len(tekst)

funksjon1(input("Skriv inn et ord: "))

'''Programmet lager nok en funkjson. I funksjonen dannes jeg en ordliste som inneholder
en setning hvor hvor væt ord blir representert som et element i listen.
Deretter opprettese en tom ordbok, som blir fylt opp av en for loop. Nøkkelverdien
blir antall ganger ordet er å finne i setningen.'''

def funksjon2(setning):
    ordliste = setning.split()
    ordbok = {}
    for index in ordliste:
        ordbok[index] = ordliste.count(index)
    return ordbok

funk2 = funksjon2(input("Skriv inn en setning: "))
print(funk2)

'''Det dannes et program som tar inn en setning fra bruker, og printer antall ord i
setningen. Setningen legges så inn i en ordbok ved bruk av funksjonen fra oppgave 2.
Så brukes en for loop til å printe antall ganger ordene forekommer i setnigen,
etterfulgt av hvor mange bokstaver det er i hvert ord.'''

def program():
    setning = (input("Skriv inn en setning: "))
    liste1 = setning.split()
    print("Det er ", len(liste1), "ord i setningen.")
    ordbok = funksjon2(setning)
    for i in ordbok:
        print("Ordet", i, "forekommer", ordbok[i], "antall ganger i setningen.")
        print("Ordet", i, "har", funksjon1(i), "antall bokstaver.")
program()
```

Temperatur (Filbehandling)

```
'''Programmet lager variabelen minFil der tekstfilen tempuratur.txt åpnes.
Det lages også en tom liste. Programmet itererer så, via en for-løkke, over
elementene i listen, stripper de, og legger de ann i den tomme listen.
Til slutt printes listen.'''

minFil=open("temperatur.txt")
minListe = []

for linje in minFil:
    minListe.append(linje.strip())

print(minListe)

'''Funksjonen funksjon defineres med liste som parameter. Variabelen sum får en verdi.
En for-løkke brukes til å iterere gjennom listen for hver linje, og legger sammen
verdiene som lagres i variabelen sum. Sum deles på lengden av listen, og lagres som
variabeln snitt. Snitt returneres. Deretter kalles funksjonen med minListe som
argument, sammen med en passende tekst.'''

def funksjon(liste):
    sum = 0
    for linje in liste:
        sum += float(linje)
    snitt = sum / len(liste)
    return snitt
print("Gjennomsnittstempreaturen er", funksjon(minListe), "grader.")
```

Repetisjon (while-løkke)

```
'''Den tomme listen mineOrd dannes.'''

mineOrd = []

'''Deretter defineres funksjonene slaaSammen og skrivUt. slaaSammen bruker parameterene
a og b, slår sammen disse to, og returnerer resultatet. skrivUt bruker en
for-løkke til å itterere gjennom en liste, og printe hver linje for seg selv.'''

def slaaSammen(a, b):
    sammen = a, b
    return sammen

def skrivUt(liste):
    for linje in liste:
        print(linje)

'''Her defineres prosedyren lokke(). Variabelen variabel1 får verdien "i".
En while-løkke brukes på variabel1, slik at den stopper om den får verdien "s".
Variabelen variabel1 får ny verdi gjennpm input. Er input "i" får brukeren skrive inn
to setninger, som legges sammen ved bruk av slaaSammen, og legges inn i listen mineOrd.
Er input "u" printes listen mineOrd. Om input er "s" stopper prosedyren.
Om brukeren skriver noe annet enn i, u eller s, får den beskjed
om at noe har gått galt, og prosedyren kalles igjen.'''

def lokke():
    variabel1 = "i"
    while variabel1 != "s":
        variabel1 = input("Skriv inn en bokstav (i/u/s): ")
        if (variabel1.lower()) == "i":
            a = input("Skriv inn en setning: ")
            b = input("Skriv inn en setning: ")
            mineOrd.append(slaaSammen(a, b))
        elif (variabel1.lower()) == "u":
            skrivUt(mineOrd)
        elif (variabel1.lower()) == "s":
            break
        else:
            nytt_forsok = input("Noe gikk galt, vil du prøve igjen? (ja/nei): ")
            if nytt_forsok == "ja":
                lokke()
            elif nytt_forsok == "nei":
                break
            else:
                break
```

Regnefunksjoner

```
'''Her definerer jeg funkjsonen addisjon, og printer den med argumenter jeg har valgt
selv. I funkjsonen adderes tall1 og tall2 og summen av disse returneres.'''

def addisjon(tall1, tall2):
    sum_add = tall1 + tall2
    return sum_add
print(addisjon(4, 7))

'''Funkjsonene subtraksjon og divisjon er satt sammen på samme måte som addisjon.
Tall1 og tall2 trekkes fra hverandre eller deles, og resultatet lagres i sum_sub/divisjon
variablene. Resultatet returneres.'''

def subtraksjon(tall1, tall2):
    sum_sub = tall1 - tall2
    return sum_sub

def divisjon(tall1, tall2):
    sum_div = tall1 / tall2
    return sum_div

'''Her kalles subtraksjon og divisjon funkjsonene tre ganger hver, og uttrykket
assert brukes til å sjekke om verdien som er forventet stemmer med resultatet i funksjonen.
Om antagelsen stemmer vil det ikke skje noe, men om den er feil vil
"Svaret skal være ..." printes.'''


assert subtraksjon(15, 5) == 10, "Svaret skal være 10"
assert subtraksjon(-15, -5) == -10, "Svaret skal være -10"
assert subtraksjon(-15, 5) == -20, "Svaret skal være -20"

assert divisjon(20, 10) == 2, "Svaret skal være 2"
assert divisjon(-20, -10) == 2, "Svaret skal være 2"
assert divisjon(-20, 10) == -2, "Svaret skal være -2"

'''Her er funksjonen tommerTilcm definert med antallTommer som parameter.
Først brukes en assert-test til å sjekke om parameteren er større enn 0.
Om det stemmer vil variabelen cm få verdien av antallTommer * 2.54, og returnere
antall cm.'''

def tommerTilcm(antallTommer):
    assert antallTommer > 0
    cm = antallTommer * 2.54
    return cm
print(tommerTilcm(10))

'''Først defineres prosedyren skrivBeregninger. Deretter tar den inn to tall fra
bruker, og uttrykket float brukes i tilfelle input er flyttall.
Input lagres under variablene tall1 og tall2. Deretter brukes tall1 og tall2 som
argumenter i funksjonene addisjon, subtraksjon og divisjon, og resultatet printes.
Variabelen antallTommer får så en floatverdi via input, før den brukes i funkjsonen
tommerTilcm, og resultatet printes med passende tekst.
Prosedyren skrivBeregninger kalles på.'''

def skrivBeregninger():
    tall1 = float(input("Skriv inn tall 1: "))
    tall2 = float(input("Skriv inn tall 2: "))
    print("Resultatet av summering er :", addisjon(tall1, tall2))
    print("Resultatet av subtraksjon er :", subtraksjon(tall1, tall2))
    print("Resultatet av divisjon er : ", divisjon(tall1, tall2))
    print("Konvertering av tommer til cm:")
    antallTommer = float(input("Skriv inn et tall: "))
    print("Resultat: ", tommerTilcm(antallTommer))
skrivBeregninger()
```

Salgsstatistikk (Filbehandling)

```
'''Denne funksjonen tar inn en fil som parameter. Inne i funkjsonen opprettese en tom
ordbok, og filen lagres som variabelen tekst. Deretter ittereres det over linjene
i teksten, og den strippes og deles. Verdiene legges så inn i ordboken, og den returneres.'''

def innlesning(filnavn):
    min_ordbok = {}
    tekst = open(filnavn)
    for linje in tekst:
        biter = linje.strip().split()
        min_ordbok[biter[0]] = int(biter[1])
    return min_ordbok

'''Denne prosedyren finner den høyeste verdien i ordboken. Først opprettes en variabel
med verdien 0. Dereter bruks en for-løkke til å gå gjennoom alle verdiene i
ordboken og oppdatere variabelen hver gang en verdi er høyere enn den aktuelle verdien.
Når den høyeste verdien er funnet printer prosedyren den høyeste verdien og hvilken
nøkkel den ligger under.'''

def maanedensSalgsperson(ordbok):
    max_verdi = 0
    for i in ordbok:
        if ordbok[i] > max_verdi:
            max_verdi = ordbok[i]
            max_key = i
    print("Månedens selger er", max_key, "med", max_verdi, "salg.")

'''Denne funkjsonen tar inn en ordbok som parameter, og legger sammen alle verdiene
som finnes i ordboken.'''

def totaltAntallSalg(ordbok):
    sum = 0
    for i in ordbok:
        sum = sum + ordbok[i]
    return sum

'''Denne funksjonen regner ut snittet av ordboken. Den tar inn en ordbok som parameter,
og bruker funkjsonen totaltAntallSalg til å renge ut summen av verdiene. deretter
deles summen på lengden av ordboken, og gjennomsnittet returneres. '''

def gjennomsnittSalg(ordbok):
    sum = totaltAntallSalg(ordbok)
    snitt = sum / len(ordbok)
    return snitt

'''I hovedprogrammet brukes funksjonene og prosedyrene fra tidligere i oppgaven
til å gi output (månedens selger, antall selgere, antall salg og gjennomsnittet).'''


def hovedprogram():
    min_ordbok = innlesning("salgstall.txt")
    maanedensSalgsperson(min_ordbok)
    print(" ")
    print("Antall selgere denne maaneden:", len(min_ordbok))
    print("Totalt antall salg:", totaltAntallSalg(min_ordbok))
    print("Gjennomsnittling antall salg per person:", gjennomsnittSalg(min_ordbok))

'''Hovedprogrammet kalles.'''
hovedprogram()
```

Motorsykkel (Klasser)

```
#Klassen Motorsykkel opprettes, med en kostruktør som har instansvariablenene
#merke, regNr og kmStand.
class Motorsykkel():
    def __init__(self, merke, regNr, kmStand):
        self._merke = merke
        self._regNr = regNr
        self._kmStand = kmStand

#Metoden kjor tar den aktuelle motorsykklen og en verdi for km som parameter.
#Så oppdaterer den kmStand ved å legge til den gitte km-verdien.

    def kjor(self, km):
        self._kmStand += km

#Denne metoden returnerer variabelen kmStand for den aktuelle motorsykkelen.

    def hentKilometerstand(self):
        return self._kmStand

#Denne metoden printer ut variablene for den aktuelle motorsykkelen med relevant tekst.

    def skrivUt(self):
        print("Merke:" + self._merke)
        print("Registreringsnummer: " + self._regNr)
        print("Kilometerstand:", self._kmStand)



'''Programmet henter klassen Motorsykkel fra filen med samme navn.'''

from motorsykkel import Motorsykkel

'''I hovedprogrammet opprettes tre objekter av klassen Motorsykkel.
Deretter printes de tre objektene ved metoden skrivUt fra klassen.
Den tredje motorsykkelen får oppdatert kmStand med 10 kilometer.
Deretter printes den nye kilometerstanden på sykkel3 med metode fra Klassen
Motorsykkel.'''

def hovedprogram():
    sykkel1 = Motorsykkel("Honda", "KN69420", 83142)
    sykkel2 = Motorsykkel("Yamaha", "WE34256", 120475)
    sykkel3 = Motorsykkel("Kawasaki", "DP18811", 55430)
    sykkel1.skrivUt()
    sykkel2.skrivUt()
    sykkel3.skrivUt()
    sykkel3.kjor(10)
    print("Ny kilometerstand:", sykkel3.hentKilometerstand())

'''hovedprogrammet kalles'''
hovedprogram()
```

Hund (Klasser)

```
#Klassen Hund opprettes. Konstruktøren tar imot verdier for alder og vekt, og har
#en bestemt verdi for metthet.

class Hund():
    def __init__(self, alder, vekt):
        self.alder = alder
        self.vekt = vekt
        self._metthet = 10

#Metoden printer alder på den akutelle hunden.

    def hentAlder(self):
        print(self.alder)

#Metoden printer vekt på den aktuelle hunden.

    def hentVekt(self):
        print(self.vekt)

#Metoden trekker fra 1 på variabelen metthet (som fra før har verdien 10).
#Om metthet kommer under 5 vil instansvariabelen vekt også miste 1 verdi.

    def spring(self):
        self._metthet -= 1
        if self._metthet < 5:
            self.vekt -= 1

#Denne metoden tar imot et heltall og legger det til metthet. Om metthet overstiger
#7 vil variablen for vekt få + 1.

    def spis(self, tall):
        self._metthet += tall
        if self._metthet > 7:
            self.vekt += 1
```

```
#Klassen Sang opprettes.
#I konstruktøren får instansvariablene _tittel og _artist verdi.
class Sang:
    def __init__(self, tittel, artist):
        self._tittel = tittel
        self._Artist = artist

#Denne metoden printer titel og artist på objektet med tilhørende tekst.
    def spill(self):
        print("Spiller:", self._tittel, "-", self._Artist)

#Denne metoden tar inn parameteren navn, og splitter den.
#Videre itterer metoden over de splittede delene, og bruker en if test til å
#sjekke om deler av navn finnes i self._Artist. Til slutt returneres false eller true.
    def sjekkArtist(self, navn):
        x = navn.split()
        for x in x:
            if x.lower() in self._Artist.lower():
                return True
        return False

#Denne metoden tar inn parameteren tittel, og sjekker om tittel er det samme som
#self._tittel. Om dette stemmer brueks en for løkke til å sjekke om deler av
#tittel er å finne i self._tittel, og returnerer true eller false.
    def sjekkTittel(self, tittel):
        x = tittel.split()
        if tittel.lower() == self._tittel.lower():
            return True
            for x in x:
                if x.lower() in self._tittel.lower():
                    return True
        return False

#Denne metoden sjekker først om tittel er det samme som self._tittel på samme måte som tidligere.
#Om dette stemmer sjekker den artist. Om begge to stemmer returneres true, ellers false.

    def sjekkArtistogTittel(self, artist, tittel):
        x = tittel.split()
        for x in x:
            if x.lower() in self._tittel.lower():
                y = artist.split()
                for y in y:
                    if y.lower() in self._Artist.lower():
                        return True
        return False
```

```python
#Klassen Sang importeres fra sang.
from sang import Sang

#_sanger og _navn dannes i konstruktøren til klassen Spilleliste.
class Spilleliste:
    def __init__(self, listenavn):
        self._sanger = []
        self._navn = listenavn

#Denne metoden leser inn en en fil med bruk av parameteren filnavn.
#Videre ittereres det over linjene i filen, og de legges til i listen
# _sanger en etter en.
    def lesFraFil(self, filnavn):
        data = open(filnavn)
        for linje in data:
            biter = linje.strip().split(";")
            self._sanger.append(Sang(biter[0], biter[1]))

#Denne metoden legger til parameteren nySang i listen _sanger.
    def leggTilSang(self, nySang):
        self._sanger.append(nySang)

#Denne metoden fjerner en sang fra listen _sanger.
    def fjernSang(self, sang):
        self._sanger.remove(sang)

#Denne metoden kaller på metoden spill() fra klassen Sang, og bruker den til å spille av
#paramteteren sang.
    def spillSang(self, sang):
        sang.spill()

#Denne metoden gjør det samme som den forrige, men her ittereres det over alle linjene
#i listen_sanger før .spill() kalles på. Alle sangene i listen printes etter tur.
    def spillAlle(self):
        for linje in self._sanger:
            linje.spill()

#Denne metoden itterer over _sanger. Den bruker metoden .sjekktittel() fra Sang.
#Hvis x finnes iblant sangtittlene returneres den.
    def finnSang(self, tittel):
        for x in self._sanger:
            if x.sjekkTittel(tittel):
                return x
        return None

#Denne metoden oppretter først en tom liste. Videre itterer den over _sanger.
#Metoden .sjekkArtist() fra Sang brukes til å sjekke om artistnavnet finnes i
#_sanger. Om dette stemmer legges sangen under navnet til i den tomme listen.
#Når alle sangene er sjekket returneres listen med sanger.
    def hentArtistUtvalg(self, artistnavn):
        sanger2 = []
        for x in self._sanger:
            if x.sjekkArtist(artistnavn):
                sanger2.append(x)
        return sanger2
```

Game of life (Klasser, nøstedelister,

```
#Klassen Celle opprettes, med en konstruktør der alle objektene får status doed fra start.

class Celle:
    def __init__(self):
        self._status = "doed" #0 = død, 1 = levende

# Metoden endrer status på celleobjektet til "doed".
    def settDoed(self):
        self._status = "doed"
        return self._status

# Metoden endrer status til "levende".
    def settLevende(self):
        self._status = "levende"
        return self._status

# Om cellen er levende il denne metoden returnere True
    def erLevende(self):
        if self._status == "levende":
            return True
        else:
            return False

# Denne metoden bruker .erLevende() til å sjekke status på en celle, og returnerer "O"
# om den er levende.
    def hentStatusTegn(self):
        if self.erLevende():
            return "O"
        else:
            return "."
```

```python
#Importerer randint fra random modulen, og klassen Celle fra celle.

from random import randint
from celle import Celle

# Klassen Spillebrett opprettes. Konstruktøren innehar instnasvariabler for rader,
# kolonner, rutenett og en generasjonsnummer.
class Spillebrett:
    def __init__(self, rader, kolonner):
        self._rader = rader
        self._kolonner = kolonner
        self._rutenett = self._generer()
        self._generasjonsnummer = 0

# Denne metoden lager rutenettet som skal brukes til Game of Life.
# Den itererer over det oppgitte antallet rader, og med en nøstet liste får hver rad et gitt antall kolonner.
# Randint brukes når et nytt celleobjekt dannes, og gir cellen 1/3 sjanse for å være levende.
# Den nyopprettede cellen legges inn på riktig plass i rad/kolonne, og loopen hopper videre til neste
# "tomme" rute i rutenettet.
    def _generer(self):
        rutenett = []
        for j in range(self._rader):
            rutenett.append([])
            for i in range(self._kolonner):
                #rutenett[j].append(Celle())
                giStatus = randint(0, 2)
                nycelle = Celle()
                if giStatus == 0:
                    #objekt = Celle().settLevende()
                    #rutenett[j].append(Celle().settLevende())
                    nycelle.settLevende()
                rutenett[j].append(nycelle)
        return rutenett

# Denne metoden skriver ut rutenettet på en ordnet måte, og bruker metoden hentStatusTegn()
# til å gi alle objektene et grafisk tegn i rutenettet.
    def tegnBrett(self):
        for linje in self._rutenett:
            streng = ""
            for objekt in linje:
                streng += objekt.hentStatusTegn()
            print(streng)

# Denne metoden finner naboer hos en bestemt celle og lagrer de i en liste.
    def finnNabo(self, rad, kol):
        naboer = []
        for j in range(-1, 2): #Finner naboer på raden før, samme rad eller raden etter.
            for i in range(-1, 2): #Finner naboer på kolonnen før, samme eller etter.
                naboRad = rad + j
                naboKol = kol + i
                gyldig = True #Variabelen gyldig får verdien True.
                if naboRad == rad and naboKol == kol: #Denne linjen utelukker det objektet vi sjekker naboene til.
                    gyldig = False
                if naboRad >= self._rader or naboRad < 0: #Sjekker at det er en rad som finnes.
                    gyldig = False
                if naboKol >= self._kolonner or naboKol < 0: #Sjekker at det er en kolonne som finnes.
                    gyldig = False
                if gyldig:
                    naboer.append(self._rutenett[naboRad][naboKol]) #Hvis kriteriene stemmer legges objektet til i nabolisten med bruk av indeksene for rad og kolonne.
        return naboer #Listen returneres
```

```
#Denne metoden oppdaterer spillebrettet etter gitte regler.
#Først itereres det over alle objektene i rutenettet. Det lages en liste over naboer for alle objektene.
#Deretter legges de levende naboene til i en egen liste. Reglene til Game of Life brukes til å bestemmme
#om naboen skal endres eller forbli slik den er, og de legges enten i skalDo eller skalLeve listen.
#Til slutt itereres det over objektene i skalDo og status endres til doed med bruk av metoden settDoed():
#Tilsvarende skjer med skalLeve. For hver oppdatering får også instansvariabelen generasjonsnummer + 1.

    def oppdatering(self):
        skalDo = []
        skalLeve = []
        for rad in range(len(self._rutenett)):
            for kol in range(len(self._rutenett[rad])):
                naboliste = self.finnNabo(rad, kol)
                levendeNaboer = []
                for objekt in naboliste:
                    if objekt.erLevende():
                        levendeNaboer.append(objekt)
                antallLevende = int(len(levendeNaboer))
                if self._rutenett[rad][kol].erLevende() and antallLevende in range(2, 4):
                    skalLeve.append(self._rutenett[rad][kol])
                elif self._rutenett[rad][kol] and antallLevende == 3:
                    skalLeve.append(self._rutenett[rad][kol])
                else:
                    skalDo.append(self._rutenett[rad][kol])
        for x in skalDo:
            x.settDoed()
        for x in skalLeve:
            x.settLevende()
        self._generasjonsnummer += 1

#Denne metoden går igjennom alle objektene i rutenettet, og bruker en teller til
#å returnere antall levende celleobjekter.
    def finnAntallLevende(self):
        teller = 0
        for rad in range(len(self._rutenett)):
            for kol in range(len(self._rutenett[rad])):
                if self._rutenett[rad][kol].erLevende():
                    teller +=1
        return teller
```

Hytte (Klasser, filbehandling, ordbøker, lister)

```
class Hytte():
    def __init__(self, hytteNavn, antallSenger, prisOvernatting):
        self._hytteNavn = hytteNavn
        self._antallSenger = antallSenger
        self._prisOvernatting = prisOvernatting #Per seng en natt, alle senger koster det samme.

    def hentHytteNavn(self):
        return self._hytteNavn

    def totPris(self, antallGjester):
        return antallGjester * self._prisOvernatting

    def sjekkPlass(self, antallGjester):
        return antallGjester <= self._antallSenger

    def __str__(self):
        s = self._hytteNavn + "  Antall senger: " + str(self._antallSenger) + "  Pris per overnatting: "
        s += str(self._prisOvernatting)
        return s

    def __eq__(self, annen):
        return self._hytteNavn == annen.hentHytteNavn()

class Tur():
    def __init__(self, hytteListe, turBeskrivelse):
        self._hytter = hytteListe
        self._turBeskrivelse = turBeskrivelse

    def skrivTur(self):
        print(self._turBeskrivelse)
        for hytte in self._hytter:
            print(hytte)

    def sjekkPrisPlass(self, antallGjester, maksPris):
        for hytte in self._hytter:
            if hytte.sjekkPlass(antallGjester):
                if hytte.totPris(antallGjester) < maksPris:
                    return True
            else:
                return False

    def hentAntHytter(self):
        return len(self._hytter)
class Turplanlegger():
    def __init__(self, hytteFil, turFil):
        self._hytter = self._hytterFraFil(hytteFil)
        self._turer = self._turerFraFil(turFil)

    def _hytterFraFil(self, filnavn):
        hFil = open(filnavn, "r")
        hytter = {}
        for linje in hFil:
            hData = linje.strip().split()
            hytte = Hytte(hData[0], int(hData[1]), float(hData[2]))
            hytter[hData[0]] = hytte
        return hytter
        hFil.close()

    def _turerFraFil(self, filNavn):
        tFil = open(filNavn, "r")
        turer = []
        linje = tFil.readline().strip()
        while linje != "":
            linje2 = tFil.readline()
            hyttenavn = linje2.split()
            hytteListe = []
            for ettNavn in hyttenavn:
                hytteListe.append(self._hytter(ettNavn))
            turer.append(Tur(hytteListe, linje))
            linje = tFil.readline().strip()
        tFil.close()
        return turer

    def finnTurer(self, antallGjester, maksPris, maksDager):
        for tur in self._turer:
            if (tur.hentAntHytter <= maksDager) and (tur.sjekkPrisPlass(antallGjester, maksPris)):
                tur.skrivTur()
```

Bruktmarked (klasser)

```
class Bud():
    def __init__(self, budGiver, budStr):
        self._budgiver = budGiver
        self._budStr = budStr
        if budStr < 0:
            self._budStr = 1

    def hentBudGiver(self):
        return self._budgiver

    def hentBudStr(self):
        return self._budStr

class Annonse():
    def __init__(self, annTekst):
        self._anntekst = annTekst
        self._annonseBud = []

    def hentTekst(self):
        return self._anntekst

    def giBud(self, hvem, belop):
        bud = Bud(hvem, belop)
        self._annonseBud.append(bud)

    def antBud(self):
        return len(self._annBud)

    def hoyesteBud(self):
        hoyest = None
        hoyestVerdi = 0
        for bud in self._annonseBud:
            if bud.hentBudStr() > hoyestVerdi:
                hoyest = bud
                hoyestVerdi = bud.hentBudStr()
        return hoyest

    def kraftBud(self, hvem, belop, maks):
        hoyest = self.hoyesteBud().hentBudStr()
        if belop <= hoyest:
            if (hoyest + 1) < maks:
                budBelop = hoyest + 1
                kraftBud = Bud(hvem, budBelop)
                self._annonseBud.append(kraftBud)
        else:
            kraftBud = Bud(hvem, belop)
            self._annonseBud.append(kraftBud)

    def kraftBudFasit(self, hvem, belop, maks):
        budBelop = belop
        hoyest = self.hoyesteBud().hentBudStr()
        maksVerdi = maks
        if belop < hoyest:
            budBelop = hoyest + 1
        if budBelop > maksVerdi:
            budBelop = maksVerdi
        self.giBud(hvem, budBelop)
```

```
class Kategori():
    def __init__(self, katNavn):
        self._kategoriNavn = katNavn
        self._kategoriAnnonser = []

    def nyAnnonse(self, annTekst):
        nyAnnonse = Annonse(annTekst)
        self._kategoriAnnonser.append(nyAnnonse)
        return nyAnnonse

    def hentAnnonser(self):
        return self._kategoriAnnonser

class Bruktmarked():
    def __init__(self):
        self._kategorier = {}

    def nyKategori(self, katNavn):
        if self.finnKategori(katNavn) == None:
            nyKategori = Kategori(katNavn)
            self._kategorier[katNavn] = nyKategori
            return nyKategori
        else:
            return None


    def finnKategori(self, katNavn):
        for kategori in self._kategorier:
            if kategori == katNavn:
                return self._kategorier[kategori]
        return None
```

Kortspill (Klasser, lister)

```
import random

class Kort():
    def __init__(self, verdi, type, farge):
        self._verdi = verdi
        self._type = type
        self._farge = farge

    def hentVerdi(self):
        return self._verdi

    def hentFarge(self):
        return self._farge

    def hentType(self):
        return self._type

    def __str__(self):
        if self._verdi == 11:
            return self._type + " knekt"
        elif self._verdi == 12:
            return self._type + " dame"
        elif self._verdi == 13:
            return self._type + " konge"
        elif self._verdi == 1:
            return self._type + " ess"
        else:
            return self._type + " " + str(self._verdi)

class Stokk():
    def __init__(self):
        self.stokk = []
        self.lagStokk()
        self.riktige = 0
        self.brukteKort = []

    def lagStokk(self):
        typer = ["Spar", "Hjerter", "Klover", "Ruter"]
        for type in typer:
            for verdi in range(1, 14):
                if type == "Spar" or type == "Klover":
                    farge = "s"
                else:
                    farge = "r"
                self.stokk.append(Kort(verdi, type, farge))

    def visStokk(self):
        for x in self.stokk:
            x.visKort()

    def fjern(self, kort):
        self.stokk.remove(kort)
        self.brukteKort.append(kort)

    def reStack(self):
        for kort in self.brukteKort:
            self.stokk.append(kort)
        for kort in self.brukteKort:
            self.brukteKort.remove(kort)

    def hentKort(self): #henter random kort
        kort = random.choice(self.stokk)
        self.fjern(kort)
        return kort
```