# IN1000 - Seminaroppgaver til uke 8 Løsningsforslag

## Oppgave 1

Skriv en klasse sirkel. En sirkel har en radius, lag en konstruktør som setter radius.

Lag tre metoder til: en som returnerer diameteren til sirkelen, en som returnerer omkretsen til sirkelen og en som returnerer arealet til sirkelen.

*Hint: omkrets av en sirkel er diameter * pi, arealet av en sirkel er radius$^2$ * pi. Eksponenter skrives som <base>**<eksponent>, feks. $2^8$ skrives 2 **8.*

Lag deretter 2 sirkler med ulik radius.
Skriv ut den ene sirkelens omkrets og areal, og den andre sirkelens diameter.

## Løsningsforslag

### Om importering av math

Hvis vi vil bruke en mer nøyaktig versjon av π, istedenfor å skrive fra hukommelsen, kan vi bruke pakka som heter *math*. Math inneholder mye nyttig, men i denne omgangen skal vi bare bruke den til å få tak i en god verdi for π. Det er to måter vi kan bruke den på i denne sammenhengen. Vi kan skrive:

```
from math import pi
```

Da kan vi bruke *pi* som om vi hadde gitt denne variabelen verdien selv. Da importerer vi bare pi, ikke resten av math-pakka.
Hvis vi skriver:

```
import math
```

så importerer vi hele pakka, og da må vi vise at det er *pi* fra *math* vi vil bruke, ved å skrive math.pi.

### Klassen sirkel:

```
class Sirkel:
    def __init__(self, radius):
        self._radius = radius          #instansvariabelen _radius få verdien til radius

    def diameter(self):
        return self._radius * 2

    def omkrets(self):
        return self.diameter() * pi            #her kan vi bruke metoden vi allerede definerte
        #return self._radius * 2 * pi          #også en mulig løsning

    def areal(self):
        return self._radius ** 2 * pi

#Oppretter to objekter
sirkel1 = Sirkel(10)
sirkel2 = Sirkel(5)

#Skriver omkrets og areal til den første.
print(sirkel1.omkrets())
```

```
print(sirkel1.areal())

#Skriver diameteren til den andre sirkelen
print(sirkel2.diameter())
```

**Merknad:**
Vi kunne også lagd en metode som skriver ut all informasjonen for oss, som gir oss "all" informasjon om en sirkel. Jeg anbefaler dere å prøve å skrive en sånn metode selv.

## Oppgave 2

Denne oppgaven hoppet vi over på grunn av endringer i pensum.

## Oppgave 3

Skriv en klasse *Leilighet*. Konstruktøren skal definere instansvariabler som representerer nummer (leilighetsnummer) og etasje. Skriv også metoder for å hente disse verdiene.

Løsningsforslag

```
class Leilighet:
    def __init__(self, leilighetsnummer, etasje):
        self._leilighetsnummer = leilighetsnummer
        self._etasje = etasje

    def hentLeilighetsnummer(self):
        return self._leilighetsnummer

    def hentEtasje(self):
        return self._etasje
```

## Oppgave 4

1. Skriv en klasse *Bygaard*. Konstruktøren til klassen skal definere en tom liste for leiligheter.
2. Skriv en metode som legger til en leilighet i listen med leiligheter.
3. Skriv en metode som tar i mot et leilighetsnummer og forsøker å finne og returnere leiligheten med dette nummeret. Hvis det ikke eksisterer en slik leilighet så skal metoden returnere *None*.

```
class Bygaard:
    def __init__(self):
        self._leiligheter = []

    def lettTilLeilighet(self, leilighet):
        self._leiligheter.append(leilighet)

    def finnLeilighet(self, leilighetsnummer):
        for leilighet in self._leiligheter:
            if leilighet.hentLeilighetsnummer() == leilighetsnummer:
                return leilighet
        return None
```

**Oppgave 5**
Gitt klassen under, lag et hovedprogram som skal bestå av deloppgavene under:

```
class Student:
        def __init__(self, n):
                self._navn = n

        def hentNavn(self):
                return self._navn
```

Løsningsforslag under hver oppgave:

**5.1**
Lag en liste for studenter. Opprett 5 studenter, en av studentene skal ha ditt navn, og legg alle studentene i listen.

```
studentliste = []
studentliste.append(Student("Petter"))
studentliste.append(Student("Nora"))
studentliste.append(Student("Silje"))
studentliste.append(Student("Gunnar"))
studentliste.append(Student("Lars"))
```

**5.2**
Sjekk om du finnes i listen (en student med ditt navn).

```
for student in studentliste:
  if student.hentNavn() == "Petter":
    print("Fant deg")
```

**5.3**
Fjern deg selv fra listen.

```
for student in studentliste:
  if student.hentNavn() == "Petter":
    studentliste.remove(student)
```

**5.4**
Sjekk hvor mange studenter som er i listen.
#Uten innebygde metoder:

```
teller = 0

for student in studentliste:
  teller += 1

print(teller)
```

#Med innebygde metoder:

```
print(len(studentliste))
```

**5.5**

Endre oppgave 5.2 til å ligge i en egen funksjon, som ser slik ut:

```
def sjekk(liste, navn):
```

Denne skal returnere Studenten dersom det finnes en student med det oppgitte navnet, ellers skal den returnere None.

```
def sjekk(liste, navn):
    for student in liste:
        if student.hentNavn() == navn:
            return student
    return None
```

Pass på at vi ikke kan sammelikne direkte med *student*. *student* refererer til et objekt, og hvis vi prøver å sammenligne det objektet med en streng, vil vi få False. Vi må derfor sammenligne navnet med navnet til studenten, altså returverdien til metoden *hentNavn*.

**5.6**

Endre oppgave 5.3 til å ligge i en funksjon som ser slik ut:
def fjern(liste, navn):
Denne skal returnere True dersom den fikk til å fjerne en student med det oppgitte navnet, og False dersom det ikke gikk (ikke fant noen med det navnet i listen).

```
def fjern(liste, navn):
    for student in liste:
        if student.hentNavn() == navn:
            liste.remove(student)
            return True
    return False
```

Her er det viktig å ikke returnere True før vi har fjernet studenten, siden ingen kodelinjer etter at vi kommer til *return* blir utført.