IN1000 - Seminaroppgaver til uke 9

Oppgave 1

Hvilke verdier får variablene til instansene av objektene av type Person her, gitt opprettelsen av person slik:
test_person = Person(13, “Kari”)

1)

```
def __init__(self, alder, navn):
    self._alder = 0
    self._navn = “ “
```

Løsningsforslag:

```
_alder = 0
_alder = ""
```

Kommentar:
I konstruktøren står det at _alder skal være 0, og at _navn skal være "", da har det ikke noe å si hva som sendes inn som parametre.

2)

```
def __init__(self, alder, navn):
    self._alder = navn
    self._navn = alder
```

Løsningsforslag:

```
_alder = "Kari"
_alder = 13
```

Kommentar:
Her er parameterne byttet om, slik at _alder får verdien til parameteren *navn,* og _navn får verdien til parameteren *alder.* Python bryr seg ikke om at _alder holder på en streng, eller at _navn blir et heltall.

3)

```
def __init__(self, alder, navn):
    self._alder = alder
    self._navn = navn
```

Løsningsforslag:

```
_alder = 13
_alder = "Kari"
```

Kommentar:
Dette er det vanlige oppsettet. Her sier vi at _navn skal ha verdien til parameteren *navn*, og at _alder skal ha verdien som gis parameteren *alder*.

4)

```
def __init__(self, a, b):
    self._alder = a
    self._navn = b
```

Løsningsforslag:

```
_alder = 13
```

```
    _alder = "Kari"
```

Kommentar:

Dette er som det vanlige oppsettet, men navnene til parameterne er annerledes. Husk at understreken er en del av navnet til instansvariablene, og at det ikke har noe å si om vi gir parameterne helt andre navn. Men husk også at dette gjør det vanskeligere å vite hva man skal sende inn som parametre. Skal *a* være en streng? Et heltall? En liste?

```
    5)
    def __init__(self, alder, navn):
        self._a = alder
        self._n = navn
```

Løsningsforslag:

```
    _a = 13
    _n = "Kari"
```

Kommentar:

Her er det navna til instansvariablene som er annerledes. Dette vil være greit for de utenfor som bruker klassen, fordi parametrene er lette å forstå, men det vil være vanskelig for andre som skal lese koden, fordi det er vanskelig å se hva _n og _a egentlig er for noe. Både parameternavn og instansvariabelnavn bør være innholdsrike og lette å forstå.

Oppgave 2

Bil

```
class Bil:
    def __init__(self, regnr, type, aarsModell):
        self._regnr = regnr
        self._type = type
        self._aarsModell = aarsModell
```

Løsningsforslag:

```
    bil1 = Bil("AB12345", "Toyota", 1997)
    bil1 = Bil("EL45678", "BMW", 2016)
    bil1 = Bil("DB54355", "Citroën", 2004)
```

Kommentar:

Registreringsnumre og typer inneholder bokstaver, og bør lagres som strenger. I tillegg har biler bare ett registreringsnummer om gangen, og bare én biltype. aarsModell bør være et heltall.

Koffert

```
class Koffert:
    def __init__(self, toalettsakerListe, klesListe):
```

```
        self._toalettsaker = toalettsakerListe
        self._klaer = klesListe
```

Løsningsforslag:

Man kan gjøre det enten ved å opprette listene på forhånd, eller ved å sende lister som argumenter uten å mellomlagre adressene i variabler, eller en kombinasjon av disse:

Gi lister som argumenter uten mellomlagring.

```
bag1 = Koffert(["tannborste", "tannkrem"], ["jakke", "genser", "bukse"])
```

Opprette to lister først:

```
toalettS = ["haarborste", "neglsaks", "tannborste"]
klaer = ["sokker", "tskjorte", "shorts"]

bag2 = Koffert(toalettS, klaer)
```

Bruke én liste, den andre uten mellomlagring.

```
bag3 = Koffert(toalettS, ["badetøy", "skjorte", "jeans"])
```

Kommentar:

Her har instansvariblene allerede *liste* som en del av navna.

Hund

```
class Hund:
    def __init__(self, a, b, c):
        self._alder = a
        self._rase = b
        self._bjefferMye = c
```

Løsningsforslag:

```
labrador = Hund(10, "Labrador", False)
shiba = Hund(4, "Shiba Inu", True)
dachs = Hund(7, "Dachs", True)
```

Kommentar:

Alder er typisk heltall, og rase er typisk streng. bjefferMye kan man for eksempel se på som en et slags ja/nei-spørsmål, og det er typisk True/False, altså en boolsk verdi. Merk at denne klassen ikke har gode navn på parameterne.