1 Part II

1.1 1

En klasse er en slags blåprint for å lage objekter, dette kan ses på som en fabrikk som lager biler, der objekter er bilene og fabrikkene er klassene.

```
1  class fabrikk:
2          def __init__(self, wheelsize, color, motor):
3              wheel.self = wheelsize
4              color.self = color
5              motor.self = motor
6          def functions_that_does_stuff(self):
7              ...
8
9  Bil1 = (50, 'green', 'RollsRoyce100X')
10
```

I denne kodesnippeten over er et eksempel på fabrikk/bil metaforet.

1.2 2

Arv(Inheritance) er at man "kopierer" en annen klasse, og derfra endre på akuratt de funksjonene/atributtene man ønsker. Dette kan brukes hvis man for eksempel skal ha en ny klasse som er ganske lik en annen, men man må endre litt på hvordan en/flere funksjon(er) fungerer. Syntaksen for dette er vist i figur(1).

```
1      class Child(Parent):
2          ....
3
```

Figur 1: Child arver fra Parent

1.3 3

Is-a relasjon er det komme fra en klasse f.eks hvis man lager et objekt fra en klasse som heter *hund* så vil det objektet ha en *is a dog* relasjon. *has-a* relasjon er hva noe har, for eksempel ved bruk av hund-eksempelet så kan hunden f.eks ha en funksjon som heter *walk* så vil klassen *hund* ha *has-a* relasjon med walk funkjonen, det samme vil gjelde objektene som blir laget fra denne klassen.

1.4 4

Encapsulation er det å gjemme implementasjon bak grensesnittet i programmet ditt, i python kan man ikke blokkere noen ute av deler av koden(utenom å kompilere til .exe eller annet.)

så derfor har det blitt laget en konvensjon der man bruker _. før en funksjon/klasse/variabel som betyr at dette er *privat* og dermed ikke bør endres. Dette kan brukes hvis hvis disse har systemkritiske funksjoner som kan ødelegge programmet hvis disse endres på.

1.5 5

Polymorfisme er det at man har noe som endrer seg basert på hva det skal gjøre. Hvis vi for eksempel har en klasse som er en gjenstand, og denne gjenstanden skal ha muligheten til å kun bevege seg diagonalt. Da vil man har en metode som heter *move*. Videre vil man at en gjenstand også skal kunne bevege seg, men denne ganger kun i en retning. Herfra kan man arve fra den første klassen og endre på move funksjonen slik at objekter fra dette kun beveger seg i en retning. Her har man da polymorfisme fordi nå kan man f.eks kalle move på objektene uten å tenke over hva de er for noe også vil de bevege seg med forskjellige regler, et eksempel på polymorfisme er *duck typing*, også kjent som "*if it walks like a duck or swims like a duck, it's a duck*. I precoden har det blitt brukt vector2 fra pygame, her er det mest sannsynelig polymorfisme. I koden har vi polimorfisme mange steder. Ett eksempel er der vi har update-funksjon eller draw funksjon. Her bruker jeg kun en funksjon og det samme skjer (blir tegnet ut på skjermen) uansett hva slags objekt det er (kule/romskip/vegg osv.)