Velkommen til gruppetime i IN1000 ☺

13. oktober 2021

Jessie Yue Guan

Planen for i dag

- Kahoot! :D
- __eq__
- __str__
- Oppgaver

__eq__

__eq__

- Hvordan kan vi vite at to tall er like?
  - 42 == 42
  - 42 != 43
- Hvordan kan vi vite at to strenger er like?
  - "doggo" == "doggo"
  - "doggo" != "pupper"

__eq__

- Hvordan vi vite at to objekter er like?
  - Hvis de har samme verdi for instansvariabelen navn?
  - Men hva om navn ikke er en passende instansvariabel?

- For eksempel, hvordan kan vi vite at to katte-objekter er like?
  - Er to katte-objekter like hvis de har samme navn?
  - Er to katte-objekter like hvis de har samme alder?
  - Er to katte-objekter like hvis de har samme farger?
  - Er to katte-objekter like hvis de har samme kjønn?

__eq__

- Svaret er at vi bestemmer, det er opp til programmereren å lage gode ID-er
- Vi kan definere hva som skal til for at to objekter regner som like vha. __eq__
- Det er en spesiell metode som kjøres automatisk når vi bruker == mellom to objekter
- NB!!! Den må alltid ta imot objektet vi skal sammenligne med og returnere True/False!!!

__eq__

- Hvis vi skal holde styr på et lite antall katter kan vi simpelheten bare bruke navn:

- **class Katt:**
  - **def __init__(self, n, a, f, k):**
    - **self.navn = n**
    - **self.alder = a**
    - **self.farger = f**
    - **self.kjonn = k**

  - **def __eq__(self, annen):**
    - **return self.navn == annen.navn**

__eq__

- Hvis vi skal holde styr på et passe antall katter kan vi bruke både navn, alder, farger, og kjønn:

- **class Katt:**
  - **def __init__(self, n, a, f, k):**
    - **self.navn = n**
    - **self.alder = a**
    - **self.farger = f**
    - **self.kjonn = k**

  - **def __eq__(self, annen):**
    - **return (self.navn == annen.navn) and (self.alder == annen.alder) and (self.farger == annen.farger) and (self.kjonn == annen.kjonn)**

__eq__

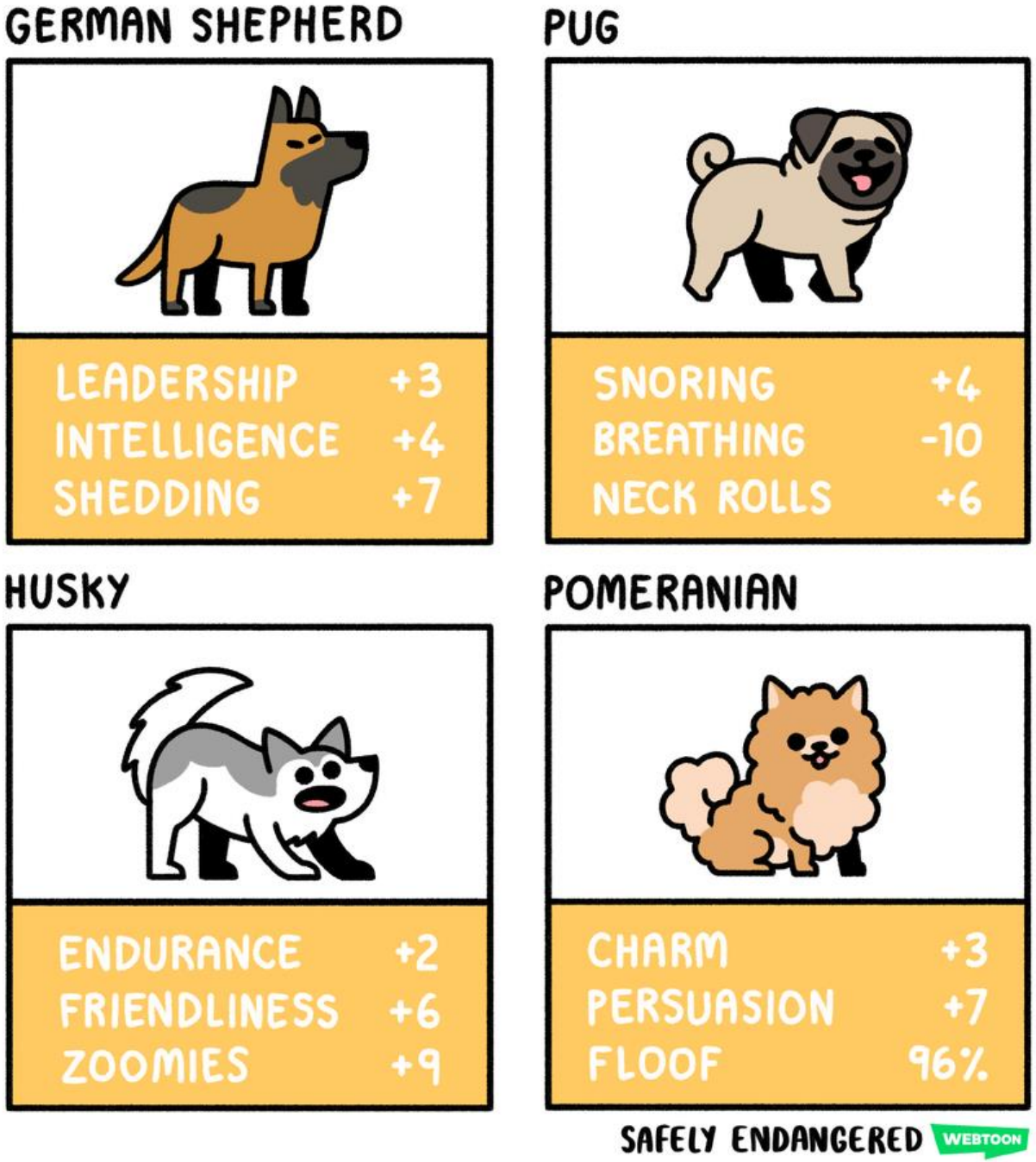
- Hvis vi skal holde styr på et stort antall katter kan vi bruke et ID-nummer:

```
class Katt:
    teller = 0

    def __init__(self):
        self.id = Katt.teller
        Katt.teller += 1

    def __eq__(self, annen):
        return self.id == annen.id
```

Spørsmål?

- Ikke vær redd for å spørre, det finnes ingen dumme spørsmål! ☺

__str__


GERMAN SHEPHERD
LEADERSHIP +3
INTELLIGENCE +4
SHEDDING +7
PUG
SNORING +4
BREATHING -10
NECK ROLLS +6
HUSKY
ENDURANCE +2
FRIENDLINESS +6
ZOOMIES +9
POMERANIAN
CHARM +3
PERSUASION +7
FLOOF 96%
SAFELY ENDANGERED
WEBTOON

__str__

- Hvordan kan vi vite hvordan vi skal printe et tall?
  - tall = 42
  - print(tall) #42

- Hvordan kan vi vite hvordan vi skal printe en streng?
  - tekst = "shibe"
  - print(tekst) #shibe

__str__

- Hvordan vi vite hvordan vi skal printe et objekt?
  - Skal vi bare printe verdien til for instansvariabelen navn?
  - Men hva om navn ikke er en passende instansvariabel?

- For eksempel, hvilke instansvariabler er naturlig å printe for katte-objekter?
  - Hva med navnet til katte-objektet?
  - Hva med alderen til katte-objektet?
  - Hva med kjønnet til katte-objektet?
  - Hva med rasen til katte-objektet?

__str__

- Svaret er at vi bestemmer, det er opp til programmereren å lage gode print-setninger
- Vi kan definere hvilke instansvariabler som skal vises når vi printer et objekt vha. __str__
- Det er en spesiell metode som kjøres automatisk når vi bruker print() på et objekt
- NB!!! Den må alltid ta imot ingen parametere (bortsett fra self) og returnere en streng!!!

__str__

- Hvis vi skal holde styr på et lite antall katter kan vi simpelheten bare printe navn:

- **class Katt:**
  - **def __init__(self, n, a, f, k):**
    - **self.navn = n**
    - **self.alder = a**
    - **self.farger = f**
    - **self.kjonn = k**

  - **def __str__(self):**
    - **return self.navn**

__str__

- Hvis vi skal holde styr på et passe antall katter kan vi printe både navn, alder, farger, og kjønn:

- **class Katt:**
  - **def __init__(self, n, a, f, k):**
    - **self.navn = n**
    - **self.alder = a**
    - **self.farger = f**
    - **self.kjonn = k**

  - **def __str__(self):**
    - **return "\n\nNavn: " + self.navn + "\nAlder: " + str(self.alder) + "\nFarger: " + str(self.farger) + "\nKjønn: " + str(self.kjonn)**

__str__

- Hvis vi skal holde styr på et stort antall katter kan vi printe navn og et ID-nummer:

```
class Katt:
    teller = 0

    def __init__(self, n):
        self.navn = n
        self.id = Katt.teller
        Katt.teller += 1

    def __str__(self):
        return str(self.id) + ": " + self.navn
```

Spørsmål?

- Ikke vær redd for å spørre, det finnes ingen dumme spørsmål! ☺

Objekter inni objekter

- La oss si at vi har fått et oppdrag i å lage et program som fungerer som et katteregister
- Du har fått informasjon om navn, alder, farger og kjønn til alle kattene i en tekstfil
- Din oppgave er å sjekke om det ved et uhell har blitt ført duplikater i den lista
- Du skal også kunne vise fram informasjon om alle kattene på en oversiktlig måte
- Hvilken datastruktur passer best til dette problemet? Hvilke klasser og filer burde vi ha?

Oppgave 1

- Dere skal få filen katt.py som inneholder klassen Katt
- Den har allerede en konstruktør som initialiserer følgende instansvariabler
  - Navn
  - Alder
  - Farger
  - Kjønn
- Lett oppgave: Dere skal lage de manglende metodene str og eq

Oppgave 2

- Dere skal få filen katteregister.py som inneholder klassen Katteregister
- Den har allerede en konstruktør som initialiserer følgende instansvariabler
  - Katter
- Vanskelig oppgave: Dere skal lage de manglende metodene les_fra_fil, finn_duplikater og vis_alle_katter

Oppgave 3

- Dere skal få filen hovedprogram.py som skal lese inn fra fil, finne duplikater, og vise informasjon
- Den har allerede en prosedyre som er tom til å begynne med
  - Hovedprogram
- Lett oppgave: Dere skal fylle inn prosedyren, dvs. lage objekter og kjøre metoder