Programmeringsteknik

Föreläsning 7

Mer om klasser och objekt:

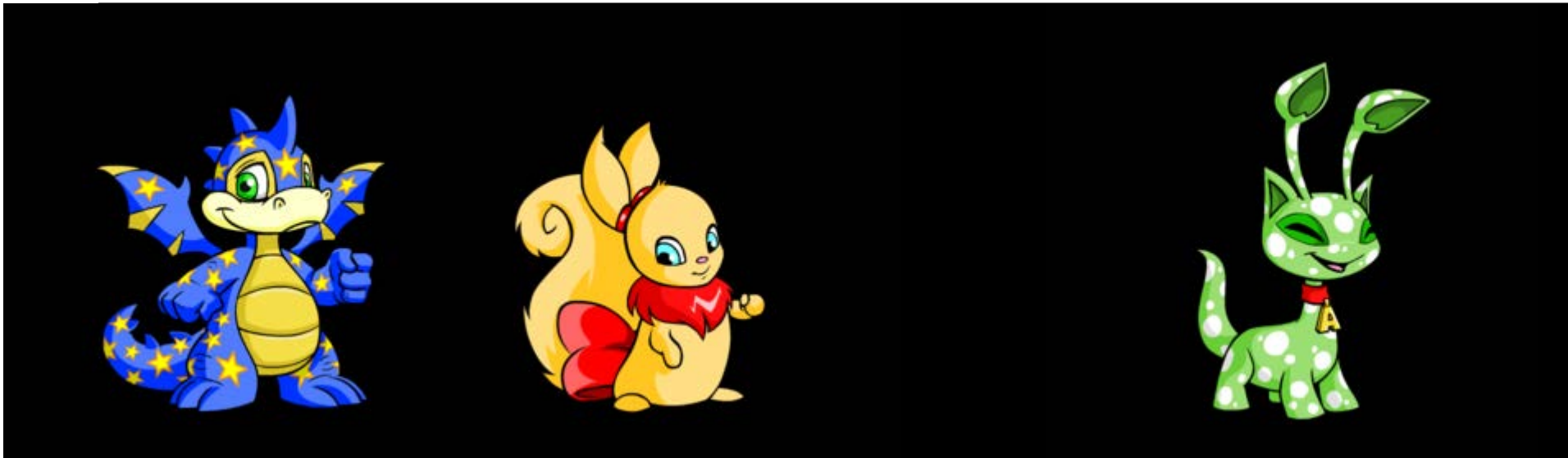
- Klass, instans och self
- Speciella metoder
- Polymorfism
- Publikt och privat
- Lista av objekt

self

- `self` ska stå överallt i klassdefinitionen:
  - först bland parametrarna: `def metod(self,…)`
  - framför varje användning av ett attribut: `self.a`
- …men *aldrig* i huvudprogrammet

Klass, instanser

- Om du definierar en klass i början av programmet…
- …så kan du skapa så många objekt (instanser av klassen) du vill i huvudprogrammet.



Speciella metoder

- `__init__`

  Anropas automatiskt när nya objekt skapas.

  Använd den för initiering av attributen!

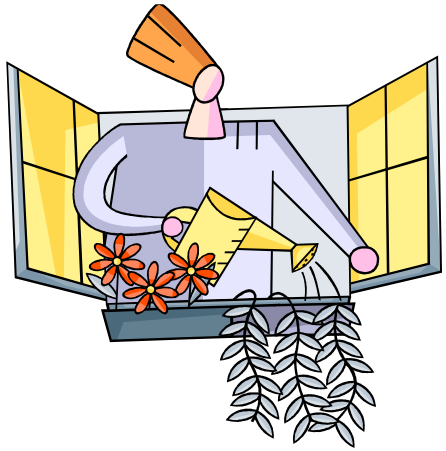
- `__str__`

  Låt den returnera en strängrepresentation av objektet, så vet *print()* hur det ska skrivas ut.

- `__lt__`

  Skriv en sån metod om du vill kunna jämföra två objekt (lt står för "less than", operatorn <)

  jämförelse = comparison



Polymorfism

- Kommer av grekiskans πολλοι *(många)* och μορφη *(form)*
- Med *polymorfism* menas här möjligheten att ha en metod med samma namn i olika klasser och få olika resultat.
- Metoden `__str__` som automatiskt anropas av print är ett exempel.

Rekommenderat byte

Det finns ett billigare alternativ. Den rekommenderade varan innehåller samma verksamma ämnen och effekten är densamma.

Föreslagen vara ?

Bricanyl® Turbuhaler®®
AstraZeneca
inhalationspulver, 0,25 mg/dos, 200 doser, inhalator.

Giltigt t o m:
2015-05-02

Antal: 1

Välj

Varunummer:093195

144,00 kr

Leverans 1-3 vardagar

Läkemedel på ditt recept

Bricanyl® Turbuhaler®®
AstraZeneca
inhalationspulver, 0,25 mg/dos, 200 doser, inhalator.

Giltigt t o m:
2015-05-02

Antal: 1

Välj

Varunummer:093195

144,00 kr

Leverans 1-3 vardagar

```
class Läkemedel:

  def __init__(self, substans,\
     beredning, namn, styrka):
      self.substans = substans
     self.beredning = beredning
     self.namn = namn
     self.styrka = styrka
     self.pris = random.randrange(25, 2201)

  def __str__(self):
     return self.namn + " " + str(self.pris) + " kr"
```

Interaktion mellan objekt

Hur kan man skriva en metod som använder två objekt?

Ha två parametrar: *self* och *other*

METOD-DEFINITION	ANROP
`def metod(self, other)`	`objekt1.metod(objekt2)`

```
def __lt__(self, other):
    if self.attribut < other.attribut:
        return True
    else:
        return False
```

```
def __lt__(self, other):
    if self.pris < other.pris:
        return True
    else:
        return False
```

Inkapsling

- I större program vill man se till att attributen bara kan ändras *inuti* klassdefinitionen.
- I huvudprogrammet anropar man en åtkomstmetod eller ändringsmetod istället!
- Mer att skriva i början men enklare när man vill använda klassen senare.
- Knepigt? Använd då privata attribut!

Publikt och privat

- Om ett attribut eller en metod definieras med ett namn som börjar med två understreck (t ex `__namn`) så är den *privat*.
- Det innebär att den endast kan användas inom klassen (man kommer inte åt den från main).
- Annars är den *publik*, och kan användas i vilken del av programmet som helst.

```
def  namn(self):
    """Åtkomstmetod för namnet"""
    return self.__namn


def  bytNamn(self, nyttNamn):
    """Ändringsmetod för namnet"""
    self.__namn = nyttNamn
```

Kan ett attribut och en metod ha samma namn?

Lista av objekt

- Flera objekt i samma program?

```
djur1 = Husdjur()
djur2 = Husdjur()
```

…

- Enklare att lägga husdjuren i en lista!

Skapa listan

```
lista = []
for i in range(n):
    nytt = Husdjur()
    lista.append(nytt)
```

Anropa metod för varje djur

```
for djur in lista:
    djur.banna()
```