

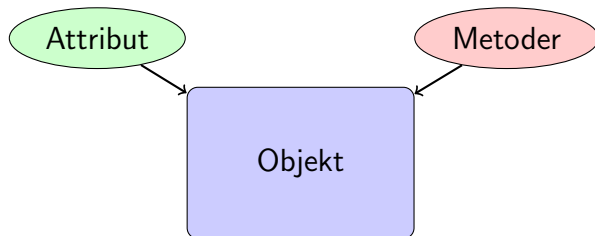
Introduktion till Objektorienterad Programmering i Python

Holger Rosencrantz

17 november 2025

Vad är Objektorienterad Programmering (OOP)?

- Ett sätt att strukturera kod genom att gruppera data och funktioner
- Tänk på verkliga objekt: bil, telefon, person
- Varje objekt har:
 - **Egenskaper** (attribut) - vad objektet *har*
 - **Beteenden** (metoder) - vad objektet *kan göra*



Grundläggande begrepp

Klass

En mall eller ritning för att skapa objekt

```
1 class Bil:  
2     # Klassens kod h r
```

Objekt

En specifik instans av en klass

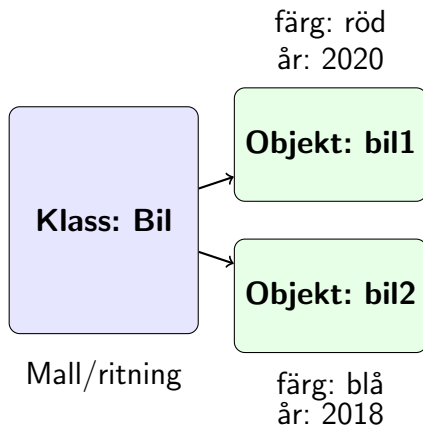
```
1 min_bil = Bil() # Skapar ett objekt
```

Attribut

Variabler som tillhör ett objekt

```
1 min_bil.farg = "vit"  
2 min_bil.ar = 2020
```

Klass vs Objekt



Metoder

Metod

Funktioner som tillhör ett objekt

```
1 class Bil:
2     def tuta(self):
3         print("Tuuut!")
4
5 min_bil = Bil()
6 min_bil.tuta() # Skriver ut "Tuuut!"
```

Konstruktör (__init__)

En speciell metod som körs när ett nytt objekt skapas

```
1 class Bil:
2     def __init__(self, farg):
3         self.farg = farg
```

Vårt exempel: Adventurer-klassen

```
1 class Adventurer:
2     def __init__(self, name):
3         self.name = name
4         self.inventory = []
```



Adventurer

name

inventory

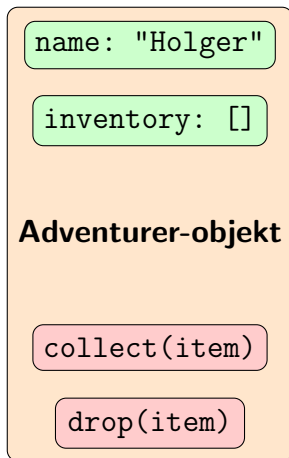
- `__init__`: Konstruktorn som sätter namn och inventory
- `self.name`: Attribut för äventyrarens namn
- `self.inventory`: Attribut för att lagra föremål

Metoderna i Adventurer-klassen

```
1 def collect(self, item):
2     if len(self.inventory) < 3:
3         self.inventory.append(item)
4         print(f"Lade till {item} i ryggsacken.")
5     else:
6         print("Fel: Du far inte ha fler an tre foremal
! ")
```

```
1 def drop(self, item):
2     if item in self.inventory:
3         self.inventory.remove(item)
4         print(f"Tog bort {item} fran ryggsacken.")
5     else:
6         print(f"Fel: {item} finns inte i ryggsacken!")
```

Adventurer-objektets struktur

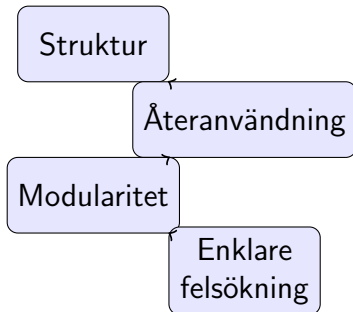


Skapa och använda objekt

```
1 # Skapa ett Adventurer-objekt
2 my_adventurer = Adventurer("Holger")
3 print(f"Aventyraren heter {my_adventurer.name}.")
4
5 # Använd metoderna
6 my_adventurer.collect("bredsward")
7 my_adventurer.collect("trolldryck")
8 my_adventurer.collect("rep")
9 my_adventurer.collect("ringbrynja") # Felmeddelande
10
11 print(my_adventurer.inventory) # Visa inventory
```

Varför använda OOP?

- **Struktur:** Kod blir lättare att läsa och underhålla
- **Återanvändning:** Klasser kan återanvändas i olika projekt
- **Modularitet:** Olika delar av programmet är oberoende
- **Enklare felsökning:** Problem är lokala till specifika klasser



OOP:s fyra pelare

Döljer implementation

Inkapsling

Ärver egenskaper

Arv

Samma gränssnitt

Polymorfism

Förenklar komplexitet

Abstraktion

- **Inkapsling:** Döljer implementation och exponerar bara nödvändigt
- **Arv:** Klasser kan ärva från andra klasser
- **Polymorfism:** Olika objekt kan användas på samma sätt
- **Abstraktion:** Döljer komplexitet och visar bara relevant information

Sammanfattning

- **Klass:** Mall för att skapa objekt
- **Objekt:** Instans av en klass
- **Attribut:** Data som tillhör ett objekt
- **Metod:** Funktioner som tillhör ett objekt
- **Konstruktör:** `__init__` som körs vid skapande

Nästa steg:

- Prova att modifiera Adventurer-klassen
- Lägg till fler metoder eller attribut
- Skapa flera äventyrare med olika egenskaper

- Python Official Documentation - Classes
- W3Schools - Python Classes
- Real Python - Object-Oriented Programming

Fortsätt experimentera med Adventurer-klassen!