Hvad er objekt-orienteret programmering (OOP)?

- Et paradigme (en måde at tænke på) inden for programmering.
- En metode til at organisere og strukture ens programkode, så den er lettere at implementere, videreudvikle og vedligeholde.

Den grundlæggende ide i OOP

- Programkoden opdeles i klasser, der hver har sit velafgrænsede ansvarsområde i programmet.
- En klasse består af attributter (variabler) og metoder (funktioner), der er knyttet til hinanden i programkoden.

De 4 grundprincipper i OOP

- Abstraktion.
- Indkapsling.
- Nedarving.
- Polymorfi.

Eksempel: Punkt i planen (uden brug af OOP)

```
from math import sqrt

# ... En masse irrelevant kode står her ...

# Vi opretter variabler for punktet (3,4)
x = 3 # Ulempe: måske benytter vi allerede x og y
y = 4 # andet steds i koden.

# Vi printer punktet ud på skærm
print("x-koordinatet er", x) # Ulempe: Print-til-skærm
print("y-koordinatet er", y) # her og der og alle vejne.

# Afstand til origo
print("Afstand til origo")  # Ulempe: Ny print-til-skærm.
a = sqrt(x**2 + y**2)       # Ulempe: Om 2 år...
print(a)                    # Hvad foregår der egentlig her?

# ... En masse irrelevant kode står her ...
```

Eksempel: Punkt i planen (vha. OOP)

```
from math import sqrt

class Punkt2D:  # definition af klasse

    def __init__(self, x, y):    # metode (konstruktør)
        self.x = x                # attribut
        self.y = y                # attribut

    def udskriv_koordinater(self):  # metode
        print("x-koordinatet er:", self.x)
        print("y-koordinatet er:", self.y)

    def afstand_til_origo(self):  # metode
        return sqrt(self.x**2 + self.y**2)

# Test
punkt = Punkt2D(3,4)  # objekt oprettes (instantiering)
punkt.udskriv_koordinater()  # metode til objekt kaldes
print(punkt.afstand_til_origo())  # metode til objekt kaldes
```

Kommentarer til punkt-eksemplet

- Kun 3 linjers kode i OOP-version. Men 7 linjer uden brug af OOP!
- Hvad nu hvis vi får brug for flere punkter i koden?
- Hvordan tester/afprøver vi isolerede dele af koden?
- Er det en fordel/ulempe at benytte mange variabler i ens kode?
- Hvordan kan flere programmører arbejde på samme projekt, hvis vi ikke bruger OOP?
- Pas på med spaghetti-kode! Svært at vedligeholde/overskue/fejlfinde.

Eksempel: LinjeSegment

```
from math import sqrt
from punkt2d import Punkt2D # Punkt2D ligger punkt2d.py

class LinjeSegment:

    def __init__(self, punkt_1, punkt_2): # konstruktør
        self.endepunkt_1 = punkt_1
        self.endepunkt_2 = punkt_2
        self.beregn_længde()

    def beregn_længde(self): # metode
        x_afstand = self.endepunkt_1.x - self.endepunkt_2.x
        y_afstand = self.endepunkt_1.y - self.endepunkt_2.y
        self.længde = sqrt((x_afstand)**2 + (y_afstand)**2)

# Test
p1 = Punkt2D(1, 2)
p2 = Punkt2D(4, 6)
ls = LinjeSegment(p1, p2)
```

Øvelse 1

1. Hent kildekoden for Punkt2D og LinjeSegment her:

   `https://drive.google.com/open?id=1aCb7VyCmMDsmCEFt552cn1ibPNvkufqS`

2. Importér kildekoden for Punkt2D og LinjeSegment i din IDE.
3. Afprøv Punkt2D og LinjeSegment.

Øvelse 2

1. Udvid Punkt2D med ny metode, der udskriver afstand til origo på skærm.
2. Udvid Punkt2D med string-attribut, der indeholder valgfrit navn på punktet.

Øvelse 3

1. Udvid LinjeSegment med metode, der udskriver endepunkter på skærm.
2. Udvid LinjeSegment med metode, der udskriver længden af linjesegment på skærm.
3. Udvid LinjeSegment med metode, så man kan ændre endepunkterne.
4. Udvid LinjeSegment med metode, der afgør om et punkt ligger på linjesegmentet.
5. Udvid LinjeSegment med metode, der beregner midtpunkt af linjesegment. Gem midtpunktet som en attribut.
6. Udvid LinjeSegment med metode, der beregner korteste afstand til et tredje punkt.

Øvelse 4

I denne øvelse skal vi se, hvordan OOP hjælper med at organisere vores kode.

1. Opret ny python-fil ved navn `main.py` i samme mappe som `punkt2d.py` og `linjesegment.py`.
2. Tilføj følgende linjer til `main.py`:

```
from punkt2d import Punkt2D
from linjesegment import LinjeSegment

def main():
    pass
```

3. Skriv kode i funktionen `main`, der beder brugeren om et punkt og udskriver afstanden til origo på skærmen.
4. Skriv kode i funktionen `main`, der beder brugeren om et punkt og udskriver midtpunktet af linjestykket.
5. Kunne vi rykke ovenstående kode, der beder brugerne om input, hen et andet sted?

Øvelse 5

1. Skriv Python-kode for klasser, der implementerer:
   1. Et punkt i 3D.
   2. En linje i 2D.
   3. En cirkel i 2D og kugle i 3D.
   4. En vektor i 2D og 3D samt en plan (i 3D).
   5. En trekant i 2D.

   Husk at tilføje passende attributter til dine klasser.
   - Et punkt i 3D har f.eks. 3 koordinater (og måske et navn).
   - En cirkel har f.eks. en radius og et centrum (og måske et navn).

   Husk at tilføje passende metoder til dine klasser.
   - En metode til at beregne korteste afstand fra linje til andet punkt.
   - En metode til at afgøre om et punkt ligger på cirkelperiferien.
   - En metode til at afgøre om to vektorer er ortogonale eller parallelle.
   - Metoder til at udskrive oplysninger vedrørende objektet.
2. Skriv Python-kode, der afprøver dine klasser.

Øvelse 6

1. Skriv Python-kode for en klasse, der implementerer et observationssæt bestående af tal.
2. Tilføj passende metoder og attributter for følgende deskriptorer:
   - Gennemsnit.
   - Typetal.
   - Mindsteværdi.
   - Størsteværdi.
   - Fraktil-beregner.
   - Kvartilafstand.
   - Varians.
   - Standardafvigelse.

Duck typing i Python

Hvis det ligner en and og går som en and, mon ikke det er en and?

```
class And:
    def flyv(self):
        print("And flyver")

class Hval:
    def svøm(self):
        print("Hval svømmer")

def afsted(objekt):
    objekt.flyv()

# Test af klasser
a = And()
h = Hval()

afsted(a)  # udskriver 'And flyver'
afsted(h)  # fejl - hvorfor?
```

Konklusion: I Python må programmøren selv holde styr på objekternes typer.

Grundlæggende begreber i OOP

- Klasse. En model[a] af noget programkode der indgår i det samlede program.
- Metode. En funktion knyttet til en klasse, der implementerer en funktionalitet ved klassen.
- Attribut. En variabel knyttet til en klasse, der beskriver en egenskab ved klassen.
- Objekt. Et instans af klassen. Dvs. en container/kopi af klassen med konkrete værdier for attributterne.

---

[a] Dvs. en beskrivelse/skabelon/blueprint af entiteter (latin: 'det som er') i vores kode. Med entitet menes en stump afgrænset kode, som producerer/behandler/arrangerer eksempelvis data i det samlede program.

Abstraktion og indkapsling

I klassen `Punkt2D` så vi eksempel på abstraktion og indkapsling:

- I `Punkt2D` findes en abstrakt og generel beskrivelse af et punkt i planen.
- I `Punkt2D` indkapsles al funktionalitet og data, der vedrører et punkt i planen.

Eksempel på nedarving: Hund og kat er eksempler på dyr

```
class Dyr:
    def __init__(self, race, alder, vægt):
        self.race = race
        self.alder = alder
        self.vægt = vægt

class Hund(Dyr):
    def __init__(self, race, alder, vægt):
        super(Hund, self).__init__(race, alder, vægt)

    def lav_larm():
        print("Vov vov!")

class Kat(Dyr):
    def __init__(self, race, alder, vægt):
        super(Kat, self).__init__(race, alder, vægt)

    def lav_larm():
        print("Miaaaww!")
```

Eksempel på nedarving: Hund og kat er dyr (fortsat)

```
# Test
h = Hund(race="Golden Retriever", alder=5, vægt=20)
k = Kat(race="Pixie-bob", alder=3, vægt=8)

h.lav_larm()                    # Vov vov!
k.lav_larm()                    # Miaaaww!
```

Findes der mon attributter, som kun knytter sig til en hund og ikke en kat (og omvendt)?

Kommentarer til hund-og-kat-eksemplet

- Fælles attributter og metoder for hunde og katte samles i klassen `Dyr`,
- I hver underklasse placeres kun attributter og metoder, der er relevante for denne underklasse.

Konklusion:

- Nedarving sikrer, at vi ikke gentager os selv. Vi undgår at vedligeholde ens kode i underklasserne.
- Nedarving sikrer, at vi kan udvide eksisterende data og funktionalitet med ny, samtidig med at koden forbliver afgrænset.
- Nedarving opnås ved en af disse metoder:
  - Specialisering (oprettelse nye underklasser som hund og kat)
  - Generalisering (ekstrahere fælles attributter og metoder fra underklasserne og placere disse i superklassen).

Øvelse 7

1. Giv konkrete eksempler fra den virkelige verden, hvor nedarving forekommer.
2. Implementér et af eksemplerne i Python.
3. Afprøv og test din kode.

Øvelse 8

I et supermarked sælges mange forskellige varer. F.eks. sælges der cola-dåser og cola-flasker. Begge er eksempler på drikkevarer. I supermarked sælges også frugt og grønt f.eks. bananer og æbler.

1. Overvej hvilke attributter og metoder, der knytter sig til den enkelte varer.
2. Er der attributter og metoder, som alle varer har?
3. Skriv Python-klasser, der repræsenterer nogle af varerne.
4. Afprøv og test din kode.

Mere information om OOP

- Video: *Object-oriented Programming in 7 minutes | Mosh*
  `https://www.youtube.com/watch?v=pTB0EiLXUC8`
- Tutorial: *Beginner's guide - Object Oriented Programming*
  `https://dev.to/charanrajgolla/beginners-guide---object-oriented-programming`