Object Oriented Programming (OOP)

Fordelen ved at arbejde/udvikle i et **OOP** (objektorienteret programmering) er, at vi kan organisere implementeringen omkring vores data som "objekter" frem for en organisering omkring funktioner eller logik. En **OOP**-implementering gør det dog også mere ligetil at trække data ud sammen med deres tilhørende relationer.

Kodespecifikke-læringsmål:

1. Strukturere jeres data som objekter og klasser.
2. Udnytte OOP til at organisere data effektivt og gøre koden nemmere at vedligeholde og udvide.

```
# Definer object/klasser
class Person: # markerer klassens body.
    def __init__(self, navn, alder, køn):
        self.navn = navn
        self.alder = alder
        self.køn = køn

def __str__(self):
        return f"Navn: {self.navn}, Alder: {self.alder}, Køn: {self.køn}"
```

Hvad sker der i kodeblokken ovenfor?

1. Opretter en `klasse` Person (en ny **brugerdefineret datatype**).

Klasser modellerer en begrebstype (her: en person) — en skabelon for objekter (instanser).

`class` er en Python-syntaks for at oprette en klasse. Når interpreter'en når denne linje, konstrueres et `klassobjekt` og navngives Person i det aktuelle namespace ("hukommelse"; `x = 10`, x peger på 10). Klassen i sig selv er et objekt.

2. Definerer en `initializer` (`__init__`), som kører når vi opretter en `instans` (`p = Person(...)`).

`__init__` kaldes en initializer der etablerer objektets indre tilstand (initialisering) og er en særlig metode (*dunder-metode*) som Python automatisk kalder når vi kører `Person(...)`.

`self`: reference til den instans der oprettes. Det er ikke et keyword, blot en konvention; vi kunne bruge et andet navn, men self er standard. `self` gør det muligt at gemme tilstande på instansen (`self.attribut`).

3. Tildeler `instansattributter` (navn, alder, køn) til objektet via `self`.

Hver tildeling skaber (eller opdaterer) en attribut på instansens.

Abstraktion: data vs. adfærd

`klasse` nu er primært **data-bærer**. I OOP ønsker man typisk også at samle relevant adfærd (*metoder*).

Abstraktion betyder at klassen udstiller et simpelt interface, mens den skjuler detaljer (fx intern validering), se nedenfor.

```
jeppe = Person(navn = "Jeppe", alder = 21, køn = "M", )
```

```
print(jeppe)
```

```
<__main__.Person object at 0x1120c5e50>
```

```
print(jeppe.alder)
```

```
21
```

```
class Person:
    def __init__(self, navn: str, alder: int, køn: str):
        self.navn = navn
        self.alder = alder   # kalder setter
        self.køn = køn

    def __str__(self):
        return f"Navn: {self.navn}, Alder: {self.alder}, Køn: {self.køn}"

    @property
    def alder(self) -> int:
        return self._alder

    @alder.setter
    def alder(self, value):
        try:
            value = int(value)
        except (TypeError, ValueError):
            raise TypeError("Alder skal være et heltal") from None
        if value < 0:
            raise ValueError("Alder kan ikke være negativ")
        self._alder = value
```

```
jeppe = Person(navn = "Jeppe", alder = 31, køn = "M")
print(jeppe)
```

```
Navn: Jeppe, Alder: 31, Køn: M
```

Hvad sker der i kodeblokke ovenfor?

Vi definerer vores tekstrepræsentation af objektet

str definerer den menneske-venlige tekstrepræsentation af objektet. Den bruges af print(obj) og af str(obj). Når du skriver print(p), kaldes p.__str__() automatisk. Hvis **str** ikke findes, falder Python tilbage til **repr** (eller en standardrepræsentation fra object).

property skaber en managed attribute.

@property gør alder til en property — en attribut hvis aflæsning viderestilles til en metode (fget).

@alder.setter definerer, hvad der sker, når man sætter p.alder = ...; koden i setter udfører validering og skriver så til et internt felt (self._alder).

Internt gemmes den faktiske værdi i self._alder (konvention: ledende underscore = "privat").

Indkapsling / abstraktion

Vi kan tilbyde et simpelt, rent API (p.alder) mens vi skjuler implementeringsdetaljer (her: _alder) og sørger for invariants (fx alder >= 0).

Encapsulation: property skjuler intern repræsentation (_alder) og eksponerer kontrolleret adgang (alder).

Abstraction: brugerfladen (p.alder) er simpel; brugeren behøver ikke kende valideringsregler.

Klassen kan indgå i større OOP-design

Vi kan fx lave `class Elev(Person):` og tilføje felter/metoder, der er relevante for under-klassen "Elever".

```
class Person:
    def __init__(self, navn: str, alder: int, køn: str):
        self.navn = navn
        self.alder = alder   # kalder setter
        self.køn = køn

    def __str__(self):
        return f"Navn: {self.navn}, Alder: {self.alder}, Køn: {self.køn}"

    @property
    def alder(self) -> int:
        return self._alder

    @alder.setter
    def alder(self, value):
        try:
            value = int(value)
        except (TypeError, ValueError):
            raise TypeError("Alder skal være et heltal") from None
        if value < 0:
            raise ValueError("Alder kan ikke være negativ")
        self._alder = value

class Elev(Person):  # Subklasse af Person
    def __init__(self, navn, alder, køn, skole, klassetrin):
        # Kald superklassens __init__ til at sætte fælles felter
        super().__init__(navn, alder, køn)
        # Tilføj felter, som kun gælder for Elev
        self.skole = skole
        self.klassetrin = klassetrin

    @classmethod
```



```
    def fra_person(cls, person, skole, klassetrin):
        return cls(person.navn, person.alder, person.køn, skole, klassetrin)

    def __str__(self):
        # Udvid str-repræsentationen med Elev-specifik info
        return (f"Navn: {self.navn}, Alder: {self.alder}, Køn: {self.køn}, "
                f"Skole: {self.skole}, Klassetrin: {self.klassetrin}")
```

```
p = Person("Jeppe", 31, "M")
e = Elev.fra_person(p, skole="Nørrevang", klassetrin=9)
print(e)
```

```
Navn: Jeppe, Alder: 31, Køn: M, Skole: Nørrevang, Klassetrin: 9
```

```
import csv
import os

# --- Klasser ---
class Person:
    def __init__(self, navn, alder, køn):
        self.navn = navn
        self.alder = alder
        self.køn = køn

    def __str__(self):
        return f"Navn: {self.navn}, Alder: {self.alder}, Køn: {self.køn}"

    @property
    def alder(self) -> int:
        return self._alder

    @alder.setter
    def alder(self, value):
        try:
            value = int(value)
        except (TypeError, ValueError):
            raise TypeError("Alder skal være et heltal") from None
        if value < 0:
            raise ValueError("Alder kan ikke være negativ")
        self._alder = value

class Elev(Person):
    def __init__(self, navn, alder, køn, skole, klassetrin):
        super().__init__(navn, alder, køn)
        self.skole = skole
        self.klassetrin = klassetrin

    def __str__(self):
        return f"{super().__str__()}, Skole: {self.skole}, Klassetrin: {self.klassetri


# --- Filnavn ---
FILENAME = "personliste.csv"
```

```python

# --- Gem listen til CSV ---
def gem_personer_csv(personer):
    # Find mappen hvor .py filen ligger
    script_dir = os.path.dirname(os.path.abspath(__file__))
    # Kombiner med filnavnet
    filepath = os.path.join(script_dir, FILENAME)

    felt_navn = ["navn", "alder", "køn", "skole", "klassetrin"]
    with open(filepath, "w", newline="", encoding="utf-8") as f:
        writer = csv.DictWriter(f, fieldnames=felt_navn)
        writer.writeheader()
        for p in personer:
            row = {
                "navn": p.navn,
                "alder": p.alder,
                "køn": p.køn,
                "skole": getattr(p, "skole", ""),
                "klassetrin": getattr(p, "klassetrin", "")
            }
            writer.writerow(row)
    print(f"Listen er gemt i '{filepath}' (CSV-fil).")


# --- Indlæs liste fra CSV ---
def indlaes_personer_csv():
    # Find mappen hvor .py filen ligger
    script_dir = os.path.dirname(os.path.abspath(__file__))
    # Kombiner med filnavnet
    filepath = os.path.join(script_dir, FILENAME)

    personer = []
    if os.path.exists(filepath):
        with open(filepath, "r", newline="", encoding="utf-8") as f:
            reader = csv.DictReader(f)
            for row in reader:
                navn = row["navn"]
                alder = int(row["alder"])
                køn = row["køn"]
                skole = row.get("skole", "")
                klassetrin = row.get("klassetrin", "")
                if skole or klassetrin:
                    personer.append(Elev(navn, alder, køn, skole, klassetrin))
                else:
                    personer.append(Person(navn, alder, køn))
        print(f"{len(personer)} personer/elev indlæst fra '{filepath}'")
    else:
        print("Ingen tidligere fil fundet, starter med tom liste.")
    return personer


# --- Terminalprogram ---
def main():
```



```python
    personer = indlaes_personer_csv()  # indlæs eksisterende CSV

    while True:
        print("\n--- Person/Elev Registrering ---")
        print("1. Tilføj person")
        print("2. Vis alle personer")
        print("3. Tilføj person til skole")
        print("4. Gem liste som CSV")
        print("5. Afslut")
        valg = input("Vælg en mulighed: ")

        if valg == "1":
            navn = input("Indtast navn: ")
            alder = input("Indtast alder: ")
            køn = input("Indtast køn: ")
            try:
                alder = int(alder)
                p = Person(navn, alder, køn)
                personer.append(p)
                print("Person tilføjet!")
            except ValueError:
                print("⚠ Alder skal være et heltal.")

        elif valg == "2":
            if not personer:
                print("Ingen personer registreret endnu.")
            else:
                print("\n--- Registrerede personer/elev ---")
                for i, person in enumerate(personer, start=1):
                    print(f"{i}. {person}")

        elif valg == "3":
            ikke_elever = [p for p in personer if not isinstance(p, Elev)]
            if not ikke_elever:
                print("Ingen personer at opgradere.")
                continue

            print("\nVælg en person at opgradere til elev:")
            for i, person in enumerate(ikke_elever, start=1):
                print(f"{i}. {person}")

            try:
                valg_index = int(input("Nummer: ")) - 1
                person_valgt = ikke_elever[valg_index]
            except (ValueError, IndexError):
                print("Ugyldigt valg.")
                continue

            skole = input("Indtast skole: ")
            klassetrin = input("Indtast klassetrin: ")
            elev = Elev(person_valgt.navn, person_valgt.alder, person_valgt.køn, skole
            personer[personer.index(person_valgt)] = elev
            print(f"{elev.navn} er nu elev på {skole}, klassetrin {klassetrin}!")
```

```
        elif valg == "4":
            gem_personer_csv(personer)

        elif valg == "5":
            print("Program afsluttes.")
            gem_personer_csv(personer)
            break

        else:
            print("Ugyldigt valg, prøv igen.")


if __name__ == "__main__":
    main()
```

```
--- Personregistrering ---
1. Tilføj person
2. Vis alle personer
3. Afslut
Ingen personer registreret endnu.

--- Personregistrering ---
1. Tilføj person
2. Vis alle personer
3. Afslut
Person tilføjet!

--- Personregistrering ---
1. Tilføj person
2. Vis alle personer
3. Afslut

--- Registrerede personer ---
1. Navn: Jeppe, Alder: 31, Køn: M

--- Personregistrering ---
1. Tilføj person
2. Vis alle personer
3. Afslut

--- Registrerede personer ---
1. Navn: Jeppe, Alder: 31, Køn: M

--- Personregistrering ---
1. Tilføj person
2. Vis alle personer
3. Afslut
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[1], line 58
```



```
     54             print("Ugyldigt valg.")

     57 if __name__ == "__main__":
---> 58     main()


Cell In[1], line 31, in main()
     29 print("2. Vis alle personer")
     30 print("3. Afslut")
---> 31 valg = input("Vælg en mulighed: ")
     33 if valg == "1":
     34     navn = input("Indtast navn: ")


File /usr/local/anaconda3/lib/python3.13/site-packages/ipykernel/kernelbase.py:1282, in
   1280     msg = "raw_input was called, but this frontend does not support input reques
   1281     raise StdinNotImplementedError(msg)
-> 1282 return self._input_request(
   1283     str(prompt),
   1284     self._parent_ident["shell"],
   1285     self.get_parent("shell"),
   1286     password=False,
   1287 )


File /usr/local/anaconda3/lib/python3.13/site-packages/ipykernel/kernelbase.py:1325, in
   1322 except KeyboardInterrupt:
   1323     # re-raise KeyboardInterrupt, to truncate traceback
   1324     msg = "Interrupted by user"
-> 1325     raise KeyboardInterrupt(msg) from None
   1326 except Exception:
```

```
1327     self.log.warning("Invalid Message:", exc_info=True)
```

```
KeyboardInterrupt: Interrupted by user
```