Mats Vik Pettersen

Teori oblig 4

Exception

En *exception* er en hendelse som oppstår under kjøring av et program, og som bryter den normale flyten av programutførelsen. Dette skjer når programmet støter på en uventet situasjon, som for eksempel en feil i koden, en uventet brukerinput, eller en ekstern feil, som et nettverksproblem. Eksempler på slike situasjoner kan være deling med null, tilgang til en ugyldig indeks i en liste, eller forsøk på å åpne en fil som ikke eksisterer.

For å håndtere exceptions og unngå at programmet krasjer, kan vi bruke *try-except*-blokker (i Python) eller lignende strukturer i andre programmeringsspråk. Denne strukturen fungerer slik:

- **try**-blokken inneholder koden som kan kaste en exception.
- **except**-blokken definerer hvordan programmet skal håndtere feilen hvis en bestemt type exception oppstår.

```
try:
    resultat = 100 / 0
except ZeroDivisionError:
    print("Kan ikke dele med null.")
```

Når vi håndterer exceptions, kan vi kontrollere feilsituasjoner på en mer elegant måte, som gjør programmet mer robust. Slik håndtering er spesielt relevant i tilfeller der vi vil sørge for at programmet fortsetter å kjøre, selv om en feil oppstår, som for eksempel ved:

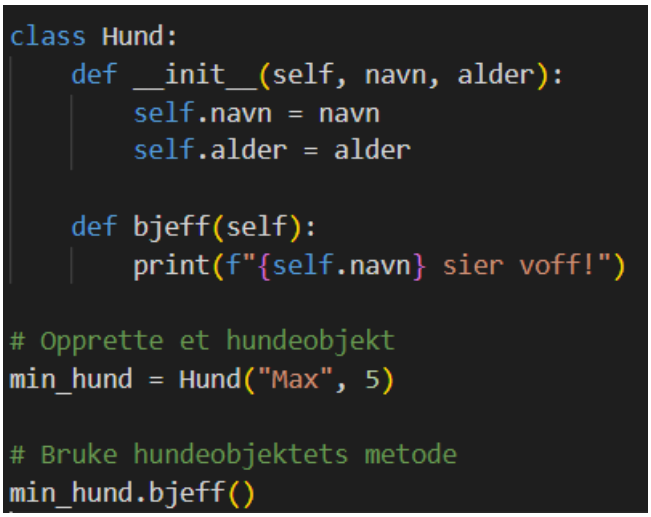
- **Filoperasjoner:** Hvis en fil ikke finnes, kan vi håndtere det uten at programmet krasjer.
- **Nettverkskall:** Hvis nettverkskall feiler, kan vi varsle brukeren uten å stoppe hele applikasjonen.
- **Brukerinput:** Hvis brukeren gir feil type input, kan vi be om ny input fremfor at programmet stopper opp.

Mats Vik Pettersen

Klasse

En *klasse* er en mal eller "blåkopi" for å lage objekter, og definerer hvilke egenskaper (variabler) og metoder (funksjoner) disse objektene vil ha. Klassen fungerer som en typebeskrivelse, som beskriver hvordan et objekt av denne typen skal være strukturert.

En klasse er nyttig når vi ønsker å modellere komplekse strukturer i programmering, som en bil, en person, eller et produkt. Vi kan bruke en klasse til å organisere relaterte data og funksjoner som en helhet.



```python
class Hund:
    def __init__(self, navn, alder):
        self.navn = navn
        self.alder = alder

    def bjeff(self):
        print(f"{self.navn} sier voff!")

# Opprette et hundeobjekt
min_hund = Hund("Max", 5)

# Bruke hundeobjektets metode
min_hund.bjeff()
```

Eksempel: Klasse for Hund

I dette eksempelet lager vi en klasse Hund som representerer en hund med egenskaper navn og alder. Klassen har også en metode bjeff som simulerer at hunden bjeffer. Denne klassen definerer hvordan alle hundeobjekter skal struktureres.

Forklaring

I dette eksempelet har vi laget en Hund-klasse med to egenskaper:

- navn: Hundens navn (f.eks. "Max")
- alder: Hundens alder (f.eks. 5)

Vi har også definert en metode, bjeff(), som simulerer at hunden bjeffer ved å skrive ut en melding.

Når vi oppretter objektet min_hund av klassen Hund med navn som "Max" og alder som 5, kan vi deretter kalle bjeff()-metoden. Dette vil skrive ut "Max sier voff!".

Mats Vik Pettersen

Objekt

Hva er et objekt?

Et *objekt* er en konkret instans av en klasse. Klassen fungerer som en mal eller en oppskrift, mens objektet er den faktiske "ting" som opprettes fra denne malen. Hvert objekt har sine egne verdier for egenskapene som klassen definerer, og kan utføre de handlingene (metodene) som er spesifisert i klassen.

Relasjon mellom klasse og objekt

I objektorientert programmering definerer vi klasser for å beskrive generelle konsepter, som Hund. Klassen Hund kan inneholde egenskaper som navn og alder, samt en metode som bjeff() for å få hunden til å bjeffe. Når vi oppretter et objekt basert på klassen, lager vi en spesifikk hund med sitt eget navn og alder.

For eksempel, hvis vi har klassen Hund, kan vi opprette et objekt som representerer en bestemt hund, for eksempel "Max", som er 5 år gammel. Objektet min_hund er da en unik instans av klassen Hund og kan bruke metodene som er definert i klassen.

```python
class Hund:
    def __init__(self, navn, alder):
        self.navn = navn
        self.alder = alder

    def bjeff(self):
        print(f"{self.navn} sier voff!")

# Opprette et hundeobjekt
min_hund = Hund("Max", 5)

# Bruke hundeobjektets metode
min_hund.bjeff()
```

I dette eksempelet:

- Hund er klassen som definerer strukturen og funksjonaliteten for alle hunder.
- min_hund er et objekt, en spesifikk instans av klassen Hund, som representerer hunden Max, som er 5 år gammel.

min_hund har egne verdier for navn og alder, men den har også tilgang til metoden bjeff() som gjør at den kan "bjeffe". Når vi kaller min_hund.bjeff(), vil objektet bruke sin egen navn-verdi og skrive ut "Max sier voff!".