Begynner 14:20

IN1000 UKE 9

Magiske metoder og
komplekse strukturer



AGENDA

Repetisjon

Referanser

Magiske metoder

En større oppgave

LITT REPETISJON

Variabler med konstant startverdi, trenger ikke å være parameter

```
def __init__(self, tall, tekst):
    self._tall = tall
    self._tekst = tekst
    self._verdi = 0
```

REFERANSER

Hva er en peker til et objekt, og hva er en kopi?

```
def dobbel_sjekk(a,b):
    print(a == b)
    print(a is b)
    print()
```

```
#hva skjer her?
a = 1
b = 1


dobbel_sjekk(a,b)
```

REFERANSER

Hva skjer her?

```
def dobbel_sjekk(a,b):
    print(a == b)
    print(a is b)
    print()
```

```
c = "Hei"
d = "Hei"


dobbel_sjekk(c,d)
```

REFERANSER

Hva skjer her?

```
def dobbel_sjekk(a,b):
    print(a == b)
    print(a is b)
    print()
```

```
e = Person("Vida",1,1,1)
f = Person("Ganda",1,1,1)


dobbel_sjekk(e,f)
```

REFERANSER

Hva skjer her?

```
def dobbel_sjekk(a,b):
    print(a == b)
    print(a is b)
    print()
```

```
g = Person("Vida",1,1,1)
h = Person("Vida",1,1,1)


dobbel_sjekk(g,h)
```

METODER – NÅR BRUKES RETURN?

- Instansmetoder kan returnere noe, men må ikke.
- return og print() har ikke noe med hverandre å gjøre.
- return brukes gjerne når vi vil hente en verdi.

```
def liste_har_negativt_tall(liste:list):
    for tall in liste:
        if tall < 0:
            return True
        else:
            return False
```

MAGISKE METODER

- **Magiske metoder** er spesielle metoder i Python som vi kan overskrive for å bruke bestemte operatorer.
- Markeres med to understreker __ før og etter navnet på metoden.

Eksempelvis vi allerede har sett: __init__

- Vi skal aldri kalle på metoden direkte med navnet, f.eks. ikke: objekt.__str__()

__str__

- Implementering av denne gjør at vi enkelt kan representere objektet vårt som en streng. Det er programmerer som definerer hvordan denne utskriften skal se ut.
- Kalles når vi brukes str(objekt) eller print(objekt)

__eq__

- Gjør det mulig å sammenligne objekter med == basert på det programmerer definerer til å være “like objekter”.
- Kalles automatisk når vi bruker == mellom to objekter av en klasse med denne metoden definert.

Dersom __eq__ ikke er implementert så bruker den **is** som standard.

Sammenligning: is vs. ==

```python
class Person:
    def __init__(self, navn):
        self._navn = navn

    def __eq__(self, annen):
        return self._navn == annen._navn

person1 = Person("Per")
person2 = person1

print(person1 is person2)   # True
print(person1 == person2)   # True

person1 = Person("Per")
person2 = Person("Per")

print(person1 is person2)   # False
print(person1 == person2)   # True
```

X is Y: Er det samme objekt (lik adresse)?

- X == Y: Er det like objekt (likt innhold)?

Sjekke om noe er None

- Generelt: Bruk is i stedet for ==
  - Grunnen til dette er at None er ett objekt – vi kan ikke ha to instanser av None

is gir derfor alltid riktig svar, mens == er avhengig av hvordan en klasse har implementert __eq__

OPPGAVER?









Musikkbibliotek

Klasser for sanger, albumer som holder på sanger, og spillelister som holder på sanger.

Implementer __str__ for å lage en fin utskrift

Implementer __eq__ for å kunne sjekke om en sang blir lagt til flere ganger i en spilleliste.

Lag et hovedprogram der brukeren kan legge til sanger og albumer (kanskje fra fil), få en oversikt over sanger, og legge dem til i en spilleliste.