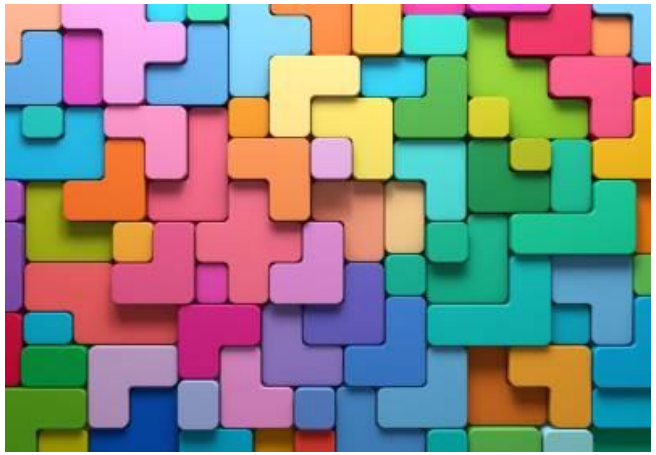
OBJEKTORIENTERT DESIGN

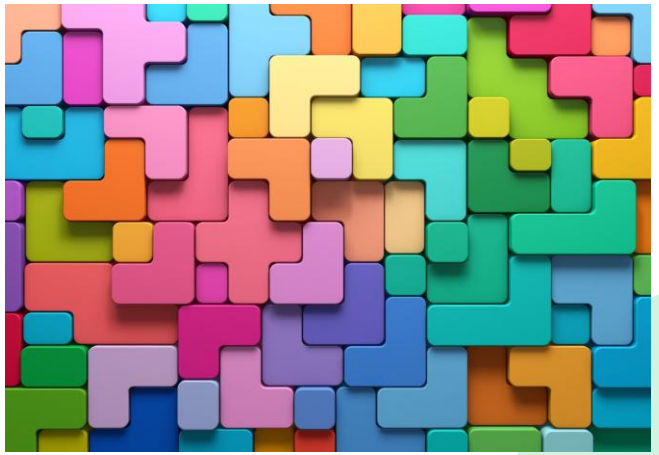
OG

FUNKSJONER SOM OBJEKTER I PYTHON

FORELESNING 10

FREDAG 19/9





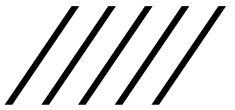
(bilder generert av bing image creator)

- https://www.menti.com

4411 8305

Noen begreper (prosjekt 1)

- Overload = flere metoder med samme navn og ulike parametre
  - **def spill_kamp(k: Kamp) -> None:**
  - **def spill_kamp(hjemmelag: Lag, bortelag: Lag) -> None:**
  - Funker ikke direkte i Python – den siste overskriver den første!
- Override = overskrive metode fra foreldreklassen (ved arv)
  - **KlassiskKortstokk.__init__** kan gjøre noe annet enn **Kortstokk.__init__**
  - **Pendulum._create_result** kan gjøre noe annet enn **ODEModel._create_result**



Tips #1: Docstrings for test-modul

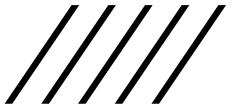
test_exp_decay.py > test_negative_decay_raises_ValueError_constructor

```python
import numpy as np
import pytest

from exp_decay import ExponentialDecay

def test_rhs() -> None:
    """Tester høyre side (RHS) av difflikningen du/dt = -au
    Kjøres med: pytest test_exp_decay.py
    """

    # høyre side inneholder ikke t, så t kan være hva som helst
    t = 0.0

    model = ExponentialDecay(0.4)    # a = 0.4
    u = np.array([3.2])              # u(t) = 3.2

    du_dt = model(t, u)
    assert np.isclose(du_dt, -1.28)  # tester at u'(t) = -1.28

def test_negative_decay_raises_ValueError_constructor() -> None:
    """Tester at a ikke kan være negativ
    Kjøres med: pytest text_exp_decay.py
    """

    a = -1.0
    with pytest.raises(ValueError):
        should_fail = ExponentialDecay(a)

def test_negative_decay_raises_ValueError() -> None:
    """Tester at a ikke kan endres til å være negativ
    Kjøres med: pytest text_exp_decay.py
    """

    a = 0.4
    model = ExponentialDecay(a)
    with pytest.raises(ValueError):
        model.decay = -1.0
```

```
>>> import test_exp_decay
>>> print(test_exp_decay.__doc__)
```

Tips #2: mypy kan sjekke om du har glemt type-hinting

```
(base) PS C:\GitHub\gnk> mypy --disallow-untyped-defs gnk.py
type.py:4: error: Function is missing a type annotation  [no-untyped-def]
type.py:20: error: Function is missing a return type annotation  [no-untyped-def]
```

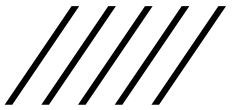


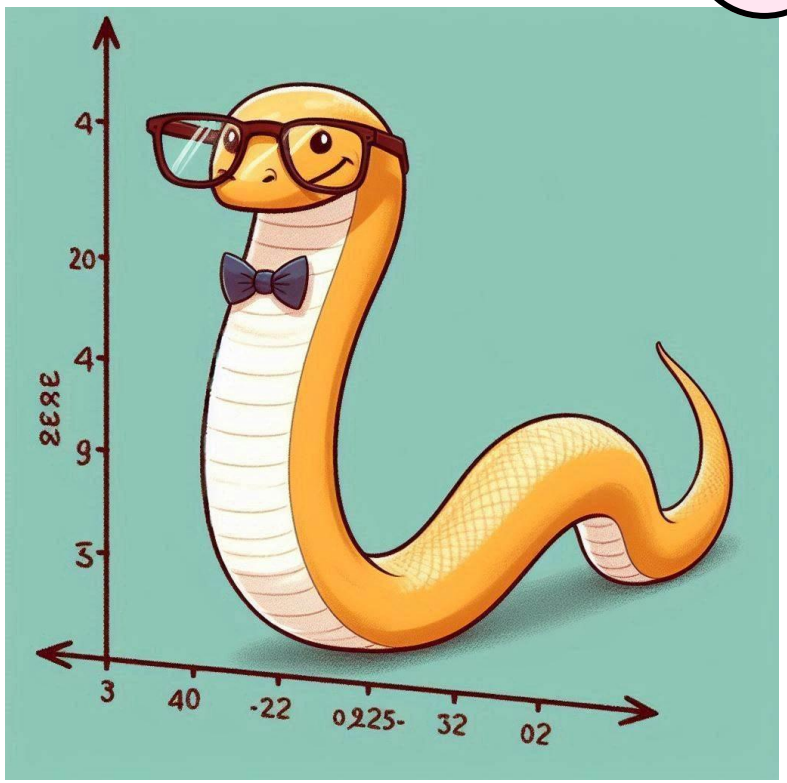
- Tips #3: mypy må ikke kjøre uten feil i prosjekt 1 (takket være numpy & scipy)

```python
@abc.abstractmethod
def __call__(self, t: float, u: np.ndarray[float]) -> np.ndarray[float]:
```

Læremål: Avansert bruk av Python

- Gjøre programmene lettere å lese og finne fram i
- *Objektorientert programmering* er en måte å tenke på når vi skriver programmer som åpner nye muligheter
- Lar oss lage spesialtilfeller fra mer generelle tilfeller (arv)
- (+ mer de kommende ukene)

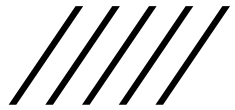




LIVEKODING: DATAKLASSER OG MER OM FUNKSJONER

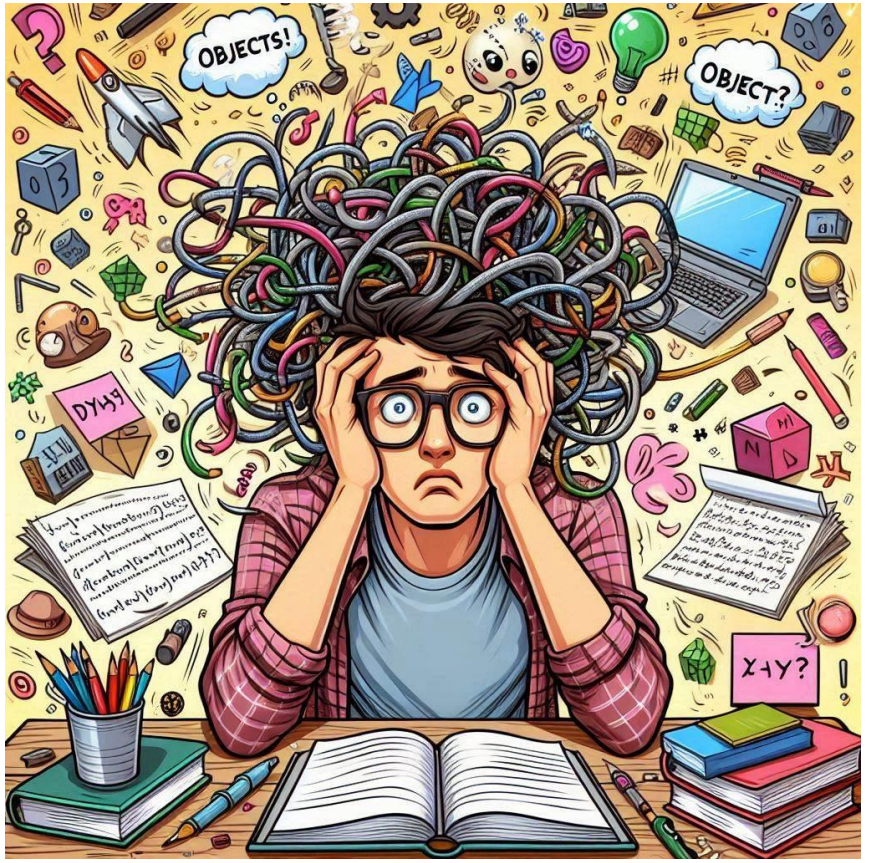
Dataobjekter – èn måte å jobbe på:

- Prøv først med å arve fra NamedTuple
  - Hvis det ikke funker (f.eks. hvis noe må endres)…
- Prøv en dataclass
  - Hvis det heller ikke funker
    (f.eks. hvis __init__ må gjøre mer enn å bare sette atributter)…
- Bruk en vanlig klasse
  - (fordel: ingen import nødvendig)
- Dette er ikke en regel! Bare et forslag til de tilfellene hvor man ikke vet helt hva man trenger!



Nesten *alt* i Python er objekter!

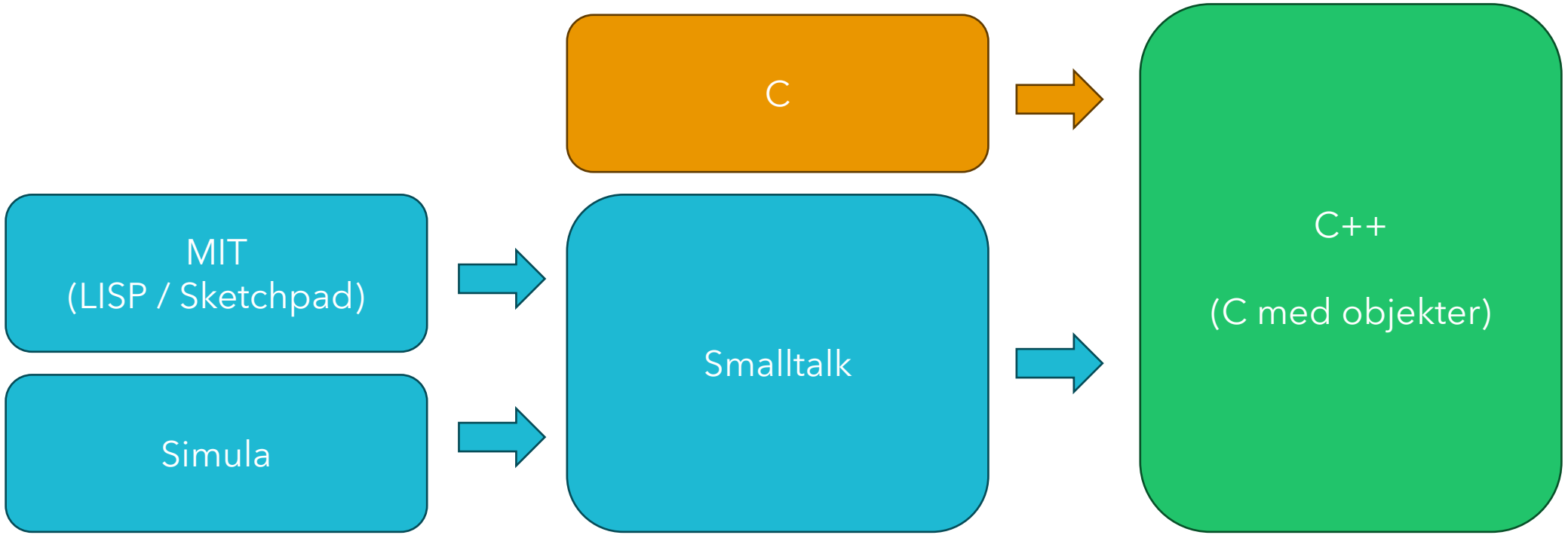
- Det tomme objektet **None** er et objekt (av klassen **NoneType**)
- Funksjoner er objekter (av klassen **function** eller **builtin_function_or_method**)
- Objekters metoder er objekter (av klassen **method** eller **builtin_function_or_method**)
- Selv *klasser* er objekter (av klassen **type**)



- https://www.menti.com

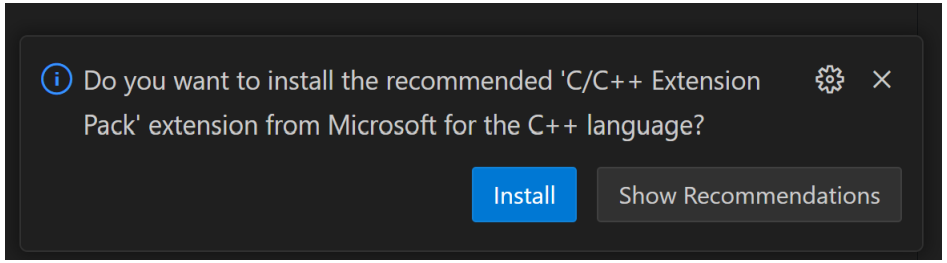
1750 2476

Litt objektorientert historikk



Til neste gang: Installere kompilator

- Først lag en .cpp-fil i VA Code:



- Mac / Linux: følg oppskriften her
- Windows: følg oppskriften her