Intro til OOP - Classes:

Overordnet er det vigtigt at huske på, at når vi koder økonomiske modeller, så arbejder vi med object-oriented programming (herefter: OOP). Hvorfor nu det?

OOP gør, at vi har et framework/skitse til at lave økonomiske modeller. Det er det, vi kalder for en "class". I vores tilfælde er "BabyMAKROModelClass" vores class, som sætter et framework for at lave en model. Så grundet "BabyMAKROModelClass" har vi en skitse til at lave nogle modeller.

```
class BabyMAKROModelClass(EconModelClass):

    # This is the BabyMAKROModelClass
    # It builds on the EconModelClass -> read the documentation

    # in .settings() you must specify some variable lists
    # in .setup() you choose parameters
    # in .allocate() all variables are automatically allocated

    def settings(self):
        """ fundamental settings """

        # a. namespaces
        self.namespaces = ['par','ss','ini','sol']

        # b. blocks
        self.blocks = [
```

Men hvad er vores model så?...et object!

Pba. vores class "BabyMAKROModelClass" kan vi lave et object (i vores tilfælde er object en model!). Et object kaldes også en "instance" fra et framework. Dermed kan et eksempel på et object være BaselineMAKRO, fordi vi laver en model baseret på frameworket "BabyMAKROModelClass".

Nedenfor ses eksempel, hvor vi definerer "model" som baseline instance fra frameworket "BabyMARKOModelClass".

```
model = BabyMAKROModelClass(name='baseline') # create an instance of the model
```

Altså, når vi laver en økonomisk model (fx BaselineMAKRO), så anvender vi vores class: "BabyMAKROModelClass" til at genere den. Det er netop smart, fordi vi derved let kan

lave et ny object. En nyt object kunne være modellen ExtensionMAKRO, som vil være baseret på samme class (altså samme framework).

Hvad f* er formålet med `self`?**
Nu hvor vi ved hvad class og object er, kan vi bedre forstå `self´.
`self` er en parameter, som repræsenterer en specifik instance af en class. I vores tilfælde betyder det, at `self` repræsenterer baselineMAKRO, som jo er generet vha. vores class "BabyMAKROModelClass".
Det er nærliggende at spørge, hvorfor vi så ikke skriver "BaselineMAKRO" i stedet for `self`, hvis `self` jo repræsenterer denne specifikke instance.
Men her er det vigtigt at huske, at en instance jo lige så godt kunne være "ExtensionMAKRO", som vi jo også gerne vil lave.
Vi ønsker, at bygge vores økonomiske framework så generelt som muligt, dvs. når vi bygger "BabyMAKROModelClass" skal det ikke være med henblik på at lave objectet "BaselineMAKRO", men alle mulige objecter fx også ExtensionMAKRO.
Dermed bliver `self` en placeholder for en given instance. Derfor anbefaler jeg at tænke på self som "BaselineMAKRO", men KUN når vi koder BaselineMAKRO.

Det er derfor også vigtigt at have `self` argument i alle argumenter på methods (dvs. funktioner) eller attributes i vores class "BabyMAKROModelClass".
Hvis vi betragter følgende method i scriptet `BabyMAKROModel.py`, så er det første argument - hvilket ALTID er tilfældet - `self`. Selvom vi ikke har lavet et object (fx BaselineMAKRO), så indgår det i vores class "BabyMAKROModelClass".

```
def find_ss(self,m_s,do_print=False):
    """ find steady state """

    steady_state.find_ss(self.par,self.ss,m_s,do_print=do_print)
```

Når vi kalder "model.find_ss()" i notebook - husk "model" her referer til vores baseline instance - så ved vores method "find_ss()" at den skal anvende parametre og steady state værdier tilhørende baselineMAKRO.

```
model.find_ss(0.50,do_print=True)
```