Object Oriented Programming in Python

What is OOP?

- OOP is one of the major paradigms in programming.
- Helps us write clean and efficient code (if used well).
- Useful for the same reason abstraction is useful:
    - A *game* consists of a list of players, actions, payoffs, timing protocol.
    - A *general equilibrium theory* consists of a commodity space, preferences, technologies and an equilibrium definition.
- Data and functions are **bundled together**.

Example

```
x = [1,5,4]
x.sort()
x
#[1,4,5]
```

- A *list* is a **class**.
- x is an instance of the class, i.e. an **object**.
- `sort` is a **method** of the list class, which is acting on the object x.

- A class can store:
    - data
    - methods (functions acting on it)

Both data and methods are called **attributes** and can be accessed by a `.`.

Example:

A consumer has a wealth, he spends and he earns.

```python
# consumer.py
class Consumer:
        def __init__(self,w):
        "Initialize consumer with w CZK"
        self.wealth = wealth

    def earn(self,y):
        "The consumer earns y CZK"
        self.wealth += income

    def spend(self,x):
        " The consumer spends x, if feasible"
        new_wealth = self.wealth-x
        if new_wealth <0:
                print("No money")
                else:
                self.wealth = new_wealth
```

Running the example

```
c1 = Consumer(10)
c1.spend(5)
c1.wealth
c1.earn(15)
c1.wealth
c1.spend(100)
```

- Your turn: create another customer with initial fortune 20, that spends 5 and earns 10.

What happened here?

- `__init__` method which is a **constructor** of the class.
- Any instance data should be prepended with `self`.
- Any method referenced within the class should have `self` as argument and should be called as `self.method_name`.

Prisoner Dilemma