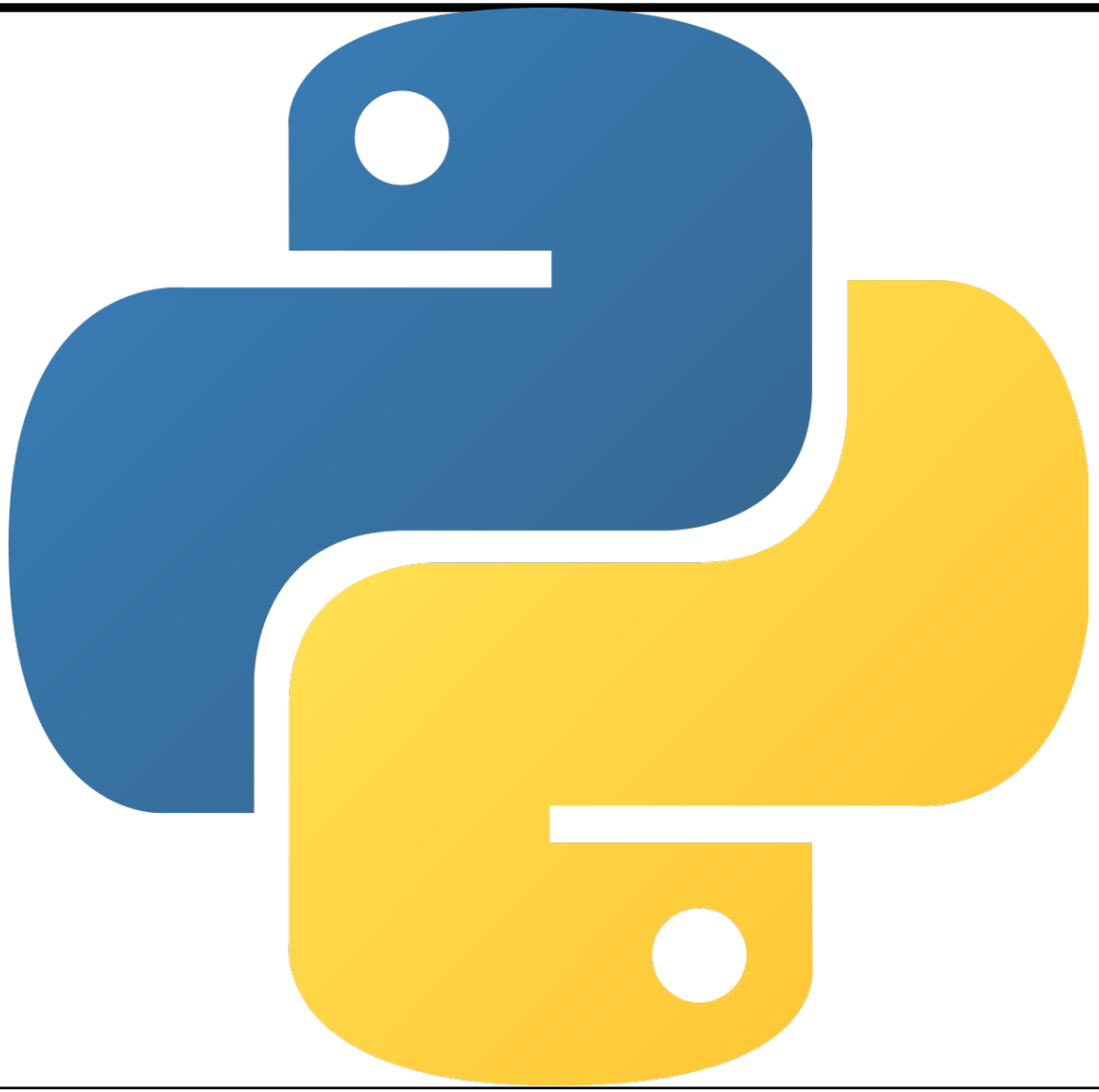
DAGENS FRÅGA

- Har du någonsin testat något du visste att du skulle vara riktigt dålig på, vad var det?

PYTHON PROGRAMMERING



Föreläsning 9

DAGENS AGENDA

- Klasser
- Objektorienterad programmering OOP

FÖRRA FÖRELÄSNING

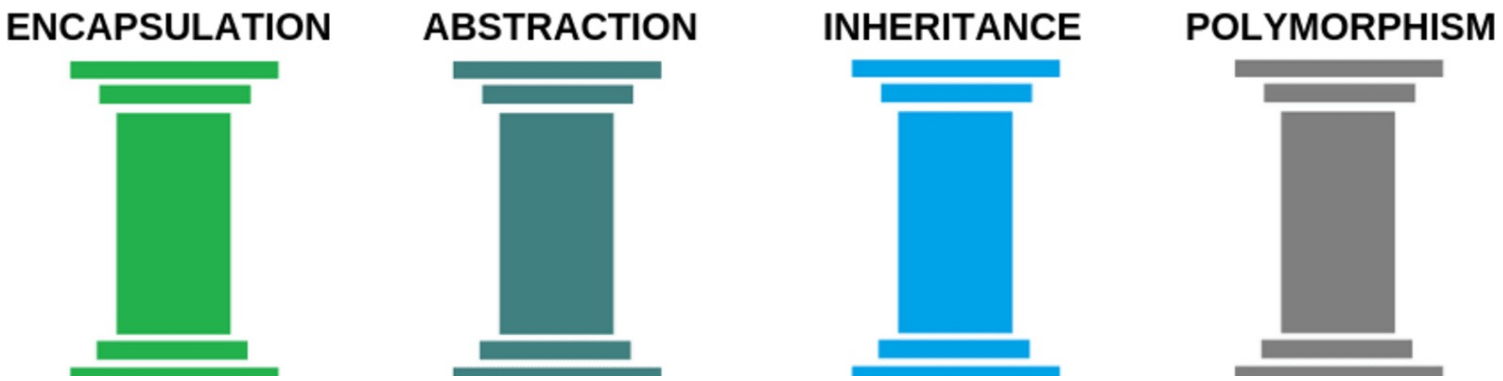
BAKGRUNG

- Hittills har vi skrivit python procuderellt, alltså definierat variabler och skrivit egna funktioner
- När man skriver större program blir det snabbt ineffektivt
- Vi vill ha kod som...
  - går att återanvända
  - är organiserad och lätt att förstå
  - är lätt att modifiera och felsöka

- Lösningen: **Objektorienterad programmering**
- https://www.youtube.com/watch?v=pTB0EiLXUC8&ab_channel=ProgrammingwithMosh 7 min video

GRUNDPELARNA I OOP

- **Abstraktion:** att simplifiera användning genom att gömma implementering → interface
- **Enkapsulering:** att samla funktionalitet och data på ett ställe → classes
- **Arv (Inheritance):** att kunna återanvända gemensam funktionalitet → parent classes
- **Polymorfism:** att kunna ändra beteendet av gemensam funktionalitet → overriding



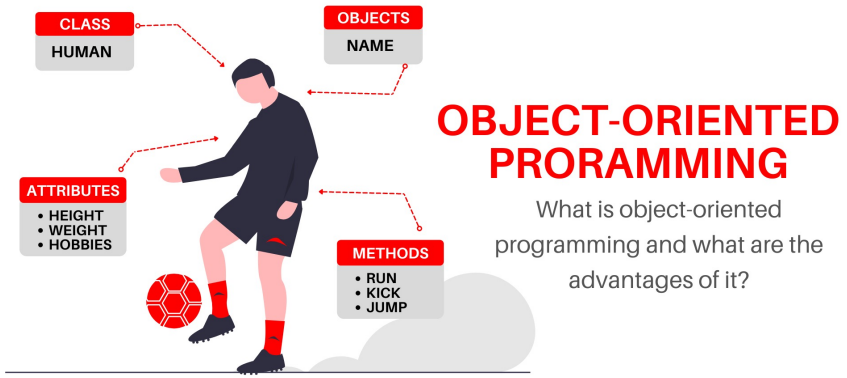
KLASSER

- I vissa fall är det användbart att kunna gruppera data och funktioner som hör ihop. Detta görs med hjälp av **klasser/classes**
- En klass är en sorts mall eller definition av ett objekt som definierar dess egenskaper
  - I python är alla entiteter objekt, även enkla datatyper som int och float

- *Inom data science är det inte ovanligt att OOP läggs åt sidan. Men att skriva objektorienterat kan ibland vara extremt effektivt i längden när kod ska återanvändas*
- *I princip alla pythonbibliotek ni interagerar med använder klasser på olika sätt*

KLASSER - TERMINOLOGI

- **Attributes**: den data som sparas i ett objekt
- **Methods**: de funktioner som associeras med ett objekt
- **self**: en referens till ett objekt i dess egna metoders definitioner
- **Initialiser**: den metod som kallas när ett objekt skapas
- **Instance**: en specifik realisering av en klass




```
class Thing():

    def __init__(self, param1, param2):
        """ This is the initializer of the class """
        self.attribute1 = param1
        self.attribute2 = param2
```

PRIVATE/PUBLIC METHODS

```python
class Thing():

    def __init__(self, param1, param2):
        """ This is the constructor of the class """
        self.attribute1 = param1
        self.attribute2 = param2

    def public_func(self):

        return

    def _private_func(self):

        return
```

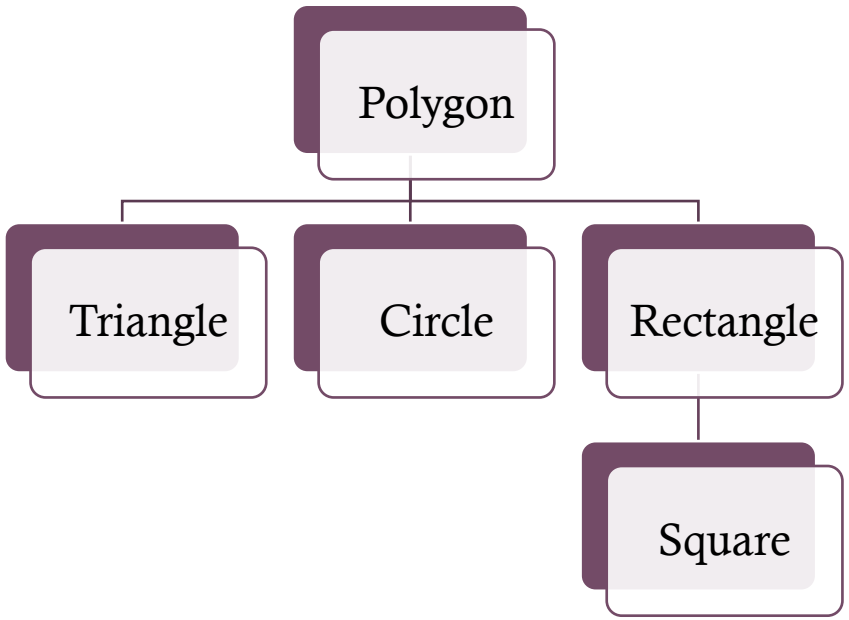
- I andra programmeringsspråk är det vanligt med publika och privata metoder. Detta är inte implementerat i python
  - Public method: ska användas utanför en objektinstans
  - Private method: ska bara användas inom objektet och är oftast hjälpmetoder
- I python är konventionen att markera privata metoder med ett underscore för att berätta för en användare att de inte är lämpliga att använda utanför klassen

```python
class Number():

    def __init__(self, value):
        self.value = value


    @staticmethod
    """ This is a method that does not require an instance of the class to be executed"""
    def static_sum(value1, value2):
        return value1 + value2
```

THE STATIC METHOD

- Ibland stöter man på decoratorn @staticmethod
- En statisk metod kräver inte att en instans av klassen finns och tar därför inte *self* som parameter i metodens definition
- Kan t.ex. användas som en alternativ initialiser eller för en funktion som verkligen knyter an till klassen

ARV OCH SUPERKLASS



- I OOP är arv ett sätt att skapa en hierarki av objekt och klasser
- En klass kan *ärva* från en *super class / parent class* vilket innebär att den har tillgång till samma metoder och attribut
- En subklass kan skugga/override en metod den ärver från sin parent class
- För att använda arv på ett strukturerat sätt så låter man mer specifika klasser ärva från mer generella