OOP

- Objektorienterad programmering
  - Paradigm
    - cirka 1960-talet (Simula 67)
  - Ett sätt att tänka
  - Systemdesign / arkitektur utifrån *objekt*
  - Återanvändbar kod+data(state) genom *objekt*
- OBS: OOP är *inte* en viss typ av programmeringsspråk
  - det är ett sätt att tänka, lösa / bryta ner problem

OOP: modularisering

- Lättare att skriva och underhålla kod om man:
  - Delar upp det i mindre bitar
  - Bygga en bit i taget
- -> Modularisering
- Dela upp applikationen, och bygga/testa varje del för sig
- Sätta sedan ihop alla delar
- Lättare att skriva tio små program
  - än att skriva en stor (10x) program

OOP: modularisering

Om modularisering ska fungera

- Så måste modulerna vara skilda från varandra,
- De ska gömma information från andra moduler,
- Men de måste också ge ett gränssnitt utåt

-> Objekt

= svart låda med DATA och FUNKTIONALITET – som användare av objektet struntar vi i innanmätet (tänk random.randint)

https://en.wikipedia.org/wiki/Mersenne_Twister

Vad är en klass .. och vad är ett objekt?

- En klass är en mall som man bygger upp kring ett informationsobjekt som man hanterar. Det är alltså något man kan hantera information om tex en kund, order, produkt.

- Den hanterar både informationen och funktionalitet kring informationen.

- När man skall använda en klass skapar man en sk instans av klassen, man kan säga att det är en kopia av mallen som man använder. En instans av klassen kallas även för ett objekt. Därifrån kommer begreppet objektorienterad programmering

OOP: Klass mer

Vad är en klass..?

- En klass speglar ett koncept i programmet
- Är också en användar-definierad typ
  - Där man själv definierar hur man skapar och använder instanser (objekt) av denna typ
- En samlingsobjekt för
  - Variabler
  - Metoder (funktioner)
- Byggsten i stora, komplexa program

OOP: Vad är skillnad på klass och objekt

- Klass = ”ritning/modell”
- Objekt = en sak som skapats utifrån ritningen

- Ex finns ritning på hus. Det är en klass/definition på hur huset ska se ut
- Sen kan MÅNGA verkliga hus (objekt) skapas utifrån denna ritning

OOP: Hur kan det bli olika objekt om samma ritning?

Ritning – det finns 4 väggar. Alla väggar har en egen färg

Olika hus(objekt) kan ju ha olika färg på sina väggar

”Variabler, state”

OOP:Klass

Vad består en klass (ritning) av?

Interface (??? = gränssnittsyta) med

Data (attribut / variabler)

Metoder (funktioner)

Constructor (speciell metod)

Destructor (speciell metod)

Access (private, protected, public)

Python: OOP

class
Metoder
Constructor (_init_)
self
Destructor
Attribut
Static attribut / metoder
Access modifiers

Python class

- I Python definierar man:
  - Funktioner med **def**
  - Klasser med **class**
- Objekt (instanser) av en **class** skapas med hjälp av “()”

```
class Person:
    def __init__(self, name, age):
        self.name = name
         self.age = age



 p = Person("Kalle", 23)

 print(p.name)
```

OOP: Klass - metoder

Publika (”interface”)

Interna (”helpers”)

Varianter på interna:

- private = osynlig för ALLA utom för kod i egna klassen
- protected = osynlig utifrån men tillgänglig för subklasser

Python: metoder

- I klassen kan man definiera ett antal metoder
- Varje metod definieras som en vanlig funktion innanför klassen (dvs: med **def**)
- Varje metod kan anropas i en objekt med hjälp av "." variabelnamn.metod

```
class Person:
    def sleep(self, minutes):
        pass

mark = Person()
mark.sleep( 60 )
```

Python: self

- Vad betyder self?
- Self är Pythons sätt att “hacka” in OOP
- Varje medlem som behöver ha tillgång till objektets data gör det via *self*
- **Som en pekare till objektet (instansen)**
- Python skickar med *self* automatiskt som en parameter när du anropar metoden med “*objekt.function()*”

```
class Person:
    def getName(self):
        return self.name


p = Person()
print( p.getName() )
```

p skickas med som *self*

Python: constructor

- Klassens constructor i Python är en vanlig metod som
  - heter __init__
  - tar valfri antal parametrar
  - måste ha *self* som första parameter
- To ensure valid state
  - Går inte att skapa ett objekt utan att vissa saker initialiseras

```
class Person:
    def __init( self,name,age)
        self.name = name
        self.age = age

        p = Person(“Stefan”, 48)
        print(p.name)
```

Python: destructor

- Klassens destructor i Python är en vanlig metod som
  - heter_ del_
  - tar endast *self* som parameter

  - En metod som anropas innan ett objekt tas bort

```
class Person:
    def_ del( self ):
        print("Removing person...")

p = Person()
del p
```

Varför tar man bort objekt i kod?

- spara minne
- objekt SKA försvinna i modellen (ex en item tas bort ur shoppingcart)

Med destructorn får vi ETT ställe att kontrollera att all ”städning” sker

Python: attribut

Variabler i en klass

Kan skapas när som helst (Python är interpreterande)

```
class Person:
    def __init__(self, name):
        #self.name = name
        pass

    def getName(self):
        self.name = "Stefan"
        return self.name;
```

Men best practice: i **constructorn**

Alla variabler i ett objekt ska ligga i *self*

self är område i minnet där allting i ett objekt lagras

hej.py > ...

```
1  class Person:
2      def __init__(self,name):
3          self.name = name
4
5      def getName(self):
6          return self.name
7
8  a = Person("312312")
9  print(a)
```

Python: static attribut / metoder

Python stödjer static attribut och metoder

Static attribut
- Definieras i klassen men inte i *self*

Static metoder
- Definieras i klassen utan *self* som parameter
- Good practice: också med @staticmethod

Python: static attribut exempel

```
class WithDrawalTransaction:
    maxAmountToWithdraw = 500

    def __init__(self,amount):
        if amount > maxAmountToWithdraw:
            self._validTransaction = True
        else:
            self._validTransaction = False
        self._amount = amount
```

Vad är det bra för??

Som ”global variabel”
men bara för klassen

Python: static metoder

```
class WithDrawalTransaction:
    maxAmountToWithdraw = 500

    @staticmethod
    def ShouldBeSentToCloud():
        if AreWeOnline(): #fake func
            return True
        return False

    def __init__(self,amount):
        if amount > maxAmountToWithdraw:
            self._validTransaction = True
        else:
            self._validTransaction = False
        self._amount = amount

transaction = WithDrawalTransaction(600)
#if transaction.ShouldBeSentToCloud() Funkar
inte
if WithDrawalTransaction.ShouldBeSentToCloud():
    ...
```

Vad är det bra för??

Funktioner som hör till denna Klass men INTE är kopplat till En speciell instans

I exemplet – transaktioner av denna typ SKA skickas om vi är Online

Python: access modifiers

Python stödjer private och protected

Default: alla attribut och metoder är public

Protected: börjar med “_”
    Tillgång inom klassen och underklasser

Private: börjar med “_ ”
    Tillgång endast inom klassen

Python: access modifiers - protected

```
class Person:
    def __init__( self, name ):
        self._name = name
    def getName( self ):
        return self._name

stefan = Person("Stefan")
print(stefan._name)
print(stefan.getName())
##men men det funkar ju???
```

Python: access modifiers

Varför fungerar stefan._name ?

Konceptet protected finns egentligen inte i Python

Det är mer som en överenskommelse:

- Om en metod eller funktion börjar med “_”
- Använd inte den utanför klassen!

Inte som C++ som blockerar tillgång, mer riktlinje

Python: access modifiers - private (dubbla __ innan namnet)

```python
class Person:
    def __init__( self, name ):
        self.__name = name
    def getName( self ):
        return self.__name

stefan = Person("Stefan")
print(stefan.getName())
print(stefan.__name)
```

Vad är det bra för??

All kod och logik för en klass ska vara
I klassen! INGEN ska utifrån kunna sätta
”state” så objektet blir i felaktigt tillstånd

Jag tar ansvar för min klass! Vem som
helst kan använda den – och ni kommer
Inte kunna ”sabba” den genom felaktig
användning