Repetition

Klasser

Objekt

- Ett objekt i Python är en del av programmet där ett antal funktioner och variabler finns och dess funktion.

Variabler och funktioner

- Variabler som finns i ett objekt är ett **attribut**.
- Funktioner som finns i ett objekt är en **metod**
- Exempel på detta är ex .keys(), .random(), .math() eller .datetime()

Vad är en klass?

- En klass ses som en ritning eller en mall för ett objekt
- Klassen beskriver hur objekt är uppbyggt och vilka operationer som kan utföras.
- Dessa operationer definieras genom attribut och metoder
- ALLA metoder har self i metoden.
- Self används i metoder för att komma åt sina egna attribut

Uppbyggnad

#min klass

```
class Klassnamn (object):
```

#definierar objektens attribut och värdetilldelning

```
def __ init __(self, attribut1, attribut2)
        self.attribut1 = attribut1
        self.attribut2 = attribut2
```

#namn tilldelas värde

```
namn = Klassnamn(” attributNamn1” , attributNamn2” )
```

```
#min klass, Bok
class Bok(object):

#definierar objektens attribut och värdetilldelning
    def __init__(self, titel, forfattare, utg, ):
        self.titel = titel
        self.forfattare = forfattare
        self.utg = utg

#kursbok tilldelas värde från klassen
kursbok = Bok(”The Shining ", ”Stephen King", 1977)
```

exempel

Import, klass

#importerar klassen

import Klassnamn

 # använder klassen och dess funktion och tillsätter värde

Klassnamn.funktion (värde1, värde2))

Papegoja

- Klass = Parrot
- Obj = Parrot()
- Klassattribute = art
- Instansattribut = name & age
- Instanser av klassen = Blu & Woo

- Steg 1 = Få tillgång till klassattributen
- Steg 2 = Få tillgång till instansattributen

```
class Parrot:

    # class attribute
    species = "bird"

    # instance attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age

# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))

# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

Skapa metoder

- Def sing
- Def dance

```
class Parrot:

    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def sing(self, song):
        return "{} sings {}".format(self.name, song)

    def dance(self):
        return "{} is now dancing".format(self.name)

# instantiate the object
blu = Parrot("Blu", 10)

# call our instance methods
print(blu.sing('"Happy"'))
print(blu.dance())
```

Arv

- Superklass/föräldrarklass

Bird

Subklass/barnklass

Pingvin

Superklass:

```
# parent class
class Bird:

    def __init__(self):
        print("Bird is ready")

    def whoisThis(self):
        print("Bird")

    def swim(self):
        print("Swim faster")
```

```
# child class
class Penguin(Bird):

    def __init__(self):
        # call super() function
        super().__init__()
        print("Penguin is ready")

    def whoisThis(self):
        print("Penguin")

    def run(self):
        print("Run faster")

peggy = Penguin()
peggy.whoisThis()
peggy.swim()
peggy.run()
```

Inkapsling

- OOP gör att vi kan förhindra tillgång till metoder och variabler.
- Att vi kan förhindra förändring av data kallas inkapsling
- I python skriver vi privata attribute _ eller __

```
class Computer:

    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

Polymorphism

- Användandet av ett interface för att använda flera former av datatyper
- Ex: Vi har olika figurer men vi skall kunna använda samma metod för att ge figurerna olika färger

```
class Parrot:

    def fly(self):
        print("Parrot can fly")

    def swim(self):
        print("Parrot can't swim")

class Penguin:

    def fly(self):
        print("Penguin can't fly")

    def swim(self):
        print("Penguin can swim")

# common interface
def flying_test(bird):
    bird.fly()

#instantiate objects
blu = Parrot()
peggy = Penguin()

# passing the object
flying_test(blu)
flying_test(peggy)
```

- OOP gör ert program lätt att förstå och effektivt
- Ni kan återanvända er kod
- Data är säker

Saker att komma ihåg

Länkar

Börja här:

https://realpython.com/python3-object-oriented-programming/

Vill du läsa mera? ☺

https://www8.cs.umu.se/kurser/5DV128/VT12/forel/klasser.pdf