Objektorientierte Programmierung

OOP

Vererbung

- zweites Ziel der OOP: Wiederverwendbarkeit
- dazu Ableitung neuer Klassen von bestehenden
- immer: Erweiterung

OOP

Vererbung

syntaktisch: Angabe der Basis in Klammern

```
1 class basisklasse:
2     pass
3
4 class tochterklasse (basisklasse):
5     pass
```

OOP

Vererbung

Tochterklasse ererbt Methoden und Attribute von Basis

```python
class Tier:
    def __init__(self, name):
        self.name = name
    def sprich(self):
        pass

class Hund(Tier):
    def sprich(self):
        return "Wau!"

hu = Hund("Bello")
print( hu.name + " sagt " + hu.sprich())
```

```
Bello sagt Wau!
```

OOP

Vererbung

Tochterklasse ererbt Methoden und Attribute von Basis

- Konstruktor ererbt
- sprich() überschrieben

```python
class Tier:
    def __init__(self, name):
        self.name = name
    def sprich(self):
        pass

class Hund(Tier):
    def sprich(self):
        return "Wau!"

```

OOP

Vererbung

Konstruktor überschrieben

- andere Attribute
- unschönes Verhalten

```
1  class Tier:
2      def __init__(self, name):
3          self.name = name
4
5  class Hund(Tier):
6      def __init__(self, hundemarke):
7          self.marke = hundemarke
8
9  hu = Hund("Bello")
10 print( hu.name )

AttributeError: 'Hund' object has no
attribute 'name'
```

OOP

Vererbung

neuer Konstruktor kann alten aufrufen

- super() → Basisklasse

```python
class Tier:
    def __init__(self, name):
        self.name = name

class Hund(Tier):
    def __init__(self, name, hundemarke):
        super().__init__(name)
        self.marke = hundemarke

hu = Hund("Bello", 3452)
print(hu.name + " Marke " + str(hu.marke))
```

```
Bello Marke 3452
```

OOP

Vererbung

super() nicht nur in Konstruktor

```python
class Tier:
    def __init__(self, name):
        self.name = name
    def sprich(self):
        return "Sagt: "

class Hund(Tier):
    def sprich(self):
        return super().sprich() + "Wau!"

hu = Hund("Bello")
print( hu.sprich() )
```

```
Sagt: Wau!
```

OOP

Laden Sie die Datei Tier.py aus dem Verzeichnis

/Dateien/src herunter.

Legen Sie eine Klasse *Katze* an, die von *Tier* abgeleitet ist, so dass

- Katze die Attribute name und farbe hat und initialisiert
- Katze.spricht() "Miau!" ausgibt
- Katze.frisst(etwas) True liefert, wenn etwas gleich "Fleisch" oder "Fisch" ist
- Katze eine Methode zum Auslesen von farbe hat

OOP

Legen Sie eine Klasse *Rassehund* an, die von *Hund* abgeleitet ist, so dass

- Rassehund die Attribute name, marke und rasse hat und initialisiert
- Rassehund.spricht() "Wau!" ausgibt
- Rassehund.frisst(etwas) True liefert, wenn etwas gleich "Filet" ist
- Rassehund eine Methode zum Auslesen von rasse hat

OOP

Funktionen für OOP

getattr(obj,name,[default])

- gibt Attribut name
- der Instanz obj zurück
- default falls nicht vorhanden

```
1 class Tier:
2     def __init__(self, name):
3         self.name= name
4
5 t = Tier("Bello")
6
7 print(getattr(t, "name"))
8 print(getattr(t, "Name", "nö"))


Bello
nö
```

OOP

Funktionen für OOP

setattr(obj,name,val)

- setzt Attribut name
- der Instanz obj auf val

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

t = Tier("Bello")

print(getattr(t, "name"))
setattr(t, "name", "Waldi")
print(getattr(t, "name"))
```

```
Bello
Waldi
```

OOP

Funktionen für OOP

delattr(obj,name)

- löscht Attribut name
- der Instanz obj

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

t = Tier("Bello")

print(getattr(t, "name"))
delattr(t, "name")
print(getattr(t, "name"))

AttributeError: 'Tier' object has
no attribute 'name'
```

OOP

Funktionen für OOP

hasattr(obj,name)

- True wenn Attribut *name* vorhanden in *obj*
- sonst False

OOP

Funktionen für OOP

isinstance(obj,classinfo)

- True wenn obj der Klasse classinfo
- classinfo auch Tupel

OOP

Funktionen für OOP

issubclass(class_,classinfo)

- True wenn class_ von Klasse classinfo abgeleitet
- classinfo auch Tupel

If the class's constructor is declared as below, which one of the assigments is valid?

```
class Class:
    def __init__(self):
        pass
```

```
a) Object = Class(object)

b) object = Class()

c) object = Class(self)

d) object = Class
```

What is the expected output?

```python
class A:

    def __init__(self, v=2):
        self.v = v

    def set(self, v=1):
        self.v += v
        return self.v

a = A()
b = a
b.set()
print(a.v)
```

a) 3

b) 0

c) 1

d) 2

What is the expected output?

```python
class Ceil:
    Token = 1

    def get_token(self):
        return 1

class Floor(Ceil):
    def get_token(self):
        return 2

    def set_token(self):
        pass

holder = Floor()
print(hasattr(holder,"Token"),
      hasattr(Ceil, "set_token"))
```

a) False True

b) True False

c) False False

d) True True

What is the expected output?

```
1  class Aircraft:
2      def start(self):
3          return "default"
4      def take_off(self):
5          self.start()
6
7  class FixedWing(Aircraft):
8      pass
9
10 class RotorCraft(Aircraft):
11     def start(self):
12         return "spin"
13
14 fleet = [FixedWing(), RotorCraft()]
15 for airship in fleet:
16     print(airship.start(),end=" ")
```

a) spin default

b) spin spin

c) default default

d) default spin