

# 706.088 INFORMATIK 1

# ÜBERBLICK

- › Klassen
  - » Attribute
  - » Methoden
- › Vererbung

# WIEDERHOLUNG

# SLICING

```
s = "This is a test string"  
s[8:14]
```

```
"a test"
```

```
s[0:14:3]
```

```
"Tss s"
```

```
s[:-5:]
```

```
"This is a test s"
```

```
s[-5:-12:-2]
```

```
"t st"
```

```
s[-13:12]
```

```
"a te"
```

# LIST COMPREHENSIONS

```
my_list = [1,2,3,4,5,6,7,8]  
[x**2 for x in my_list if x%2 == 0]
```

```
[4, 16, 36, 64]
```

```
my_list1 = ["A", "B", "C"]
my_list2 = ["D", "E", "F"]
my_list3 = ["G", "H", "I"]
[(a,b,c) for a in my_list1 for b in my_list2 for c in my_list3]
```

```
[('A', 'D', 'G'), ('A', 'D', 'H'), ('A', 'D', 'I'), ('A', 'E', 'G'),
 ('A', 'E', 'H'), ('A', 'E', 'I'), ('A', 'F', 'G'), ('A', 'F', 'H'),
 ('A', 'F', 'I'), ('B', 'D', 'G'), ('B', 'D', 'H'), ('B', 'D', 'I'),
 ('B', 'E', 'G'), ('B', 'E', 'H'), ('B', 'E', 'I'), ('B', 'F', 'G'),
 ('B', 'F', 'H'), ('B', 'F', 'I'), ('C', 'D', 'G'), ('C', 'D', 'H'),
 ('C', 'D', 'I'), ('C', 'E', 'G'), ('C', 'E', 'H'), ('C', 'E', 'I'),
 ('C', 'F', 'G'), ('C', 'F', 'H'), ('C', 'F', 'I')]
```

# LAMBDA FUNKTIONEN

› müssen nicht benannt werden

```
lambda x: x**2  
  
sq = lambda x: x**2  
sq(3) # 9  
  
(lambda x: x**2)(3) # 9
```

```
my_list = [1,2,3,4,5,6,7,8]
list(map(lambda x: x**2, my_list))

[1, 4, 9, 16, 25, 36, 49, 64]
```



# ANONYME FUNKTIONEN

```
sorted(deck, key=lambda x: x[1])
```

# OBJEKTORIENTIERUNG

# OBJEKTORIENTIERUNG IN PYTHON

- › Hilft bei der Abstrahierung von Problemstellungen
- › **Datenstrukturen** und dazu passende **Funktionen** werden zu **einem Objekt** zusammengefasst

# KLASSEN

```
class MyClass:
    globalClassCount = 0
    def __init__(self, initialData):
        self.data = initialData
        MyClass.globalClassCount += 1

    def getData(self):
        return self.data
    def getClassCount(self):
        return MyClass.globalClassCount
```

# DEMO

# VERERBUNG

- › Verbessert die Wiederverwendbarkeit und Erweiterbarkeit von Code
- › Basisklasse vererbt Eigenschaften und Fähigkeiten
- › Tochterklasse wird um Funktionalität und Eigenschaften erweitert

# VERERBUNG

```
class A:
    def __init__(self):
        self.X = 2
        self.Y = 3
        print("Konstruktor in A")
    def getX(self):
        return self.X
class B(A):
    def getArea(self):
        return self.X * self.Y

b = B()
print(b.getArea())
print(b.getX())
```

```
Konstruktor in A
6
2
```

# FRAGEN?