```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# """
# Created on Fri Oct 19 16:59:39 2018
#
# @author: Achim Schaedle
# """
```

1 Eine Einführung in die objektorientierte Programmierung

Eine ausführliche Beschreibung finden Sie hier: https://www.python-kurs.eu/klassen.php
Wir haben bereits vordefinierte Objekte und Methoden von Klassen benutzt.

```
liste1 = list((1,2,3))
```

liste1 ist eine Instanz oder ein Objekt der Klasse list

```
liste1.append()
```

```
---------------------------------------------------------------------------TypeError
call last)<ipython-input-1-0220eec9d549> in
<module>()
----> 1liste1.append()
TypeError: append() takes exactly one argument
(0 given)
```

1

```
a = -23
type(a)
dir(a)
```

```
['__abs__',
 '__add__',
 '__and__',
 '__bool__',
 '__ceil__',
 '__class__',
 '__delattr__',
 '__dir__',
 '__divmod__',
 '__doc__',
 '__eq__',
 '__float__',
 '__floor__',
 '__floordiv__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getnewargs__',
```

```
'__gt__',
'__hash__',
'__index__',
'__init__',
'__init_subclass__',
'__int__',
'__invert__',
'__le__',
'__lshift__',
'__lt__',
'__mod__',
'__mul__',
'__ne__',
'__neg__',
'__new__',
'__or__',
'__pos__',
'__pow__',
'__radd__',
'__rand__',
'__rdivmod__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__rfloordiv__',
'__rlshift__',
'__rmod__',
'__rmul__',
'__ror__',
'__round__',
'__rpow__',
'__rrshift__',
'__rshift__',
'__rsub__',
'__rtruediv__',
'__rxor__',
'__setattr__',
'__sizeof__',
'__str__',
'__sub__',
'__subclasshook__',
'__truediv__',
'__trunc__',
'__xor__',
'bit_length',
'conjugate',
'denominator',
'from_bytes',
'imag',
'numerator',
'real',
'to_bytes']
```

1.1 Eine erste Klasse, die nichts kann

Die minimale Definition einer Klasse in Python hat die Form

```
class ErsteKlasse(object):
    pass # pass gibt an, dass hier noch irgendwas hin soll
```

1.2 Eine einfache Klasse

```
class ZweiteKlasse(object):
#     """ eine zweite Klasse
#     """
    n = 1234
    def f(self,x):
        return x**2

z = ZweiteKlasse()
z.f(2)
help(z)
```

```
Help on ZweiteKlasse in module __main__ object:

class ZweiteKlasse(builtins.object)
 |  Methods defined here:
 |
 |  f(self, x)
 |
 |
----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |
----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  n = 1234
```

1.3 Klasse Studenten

```
class Student(object):

    def __init__(self,Vn,Nn,MatNr):
        self.Vorname = Vn
        self.Nachname = Nn
        self.MatrikelNummer = MatNr
    #'\_\_init\_\_ wird automatisch beim Erstellen eines Objektes aufgerufen

    # Klassenmethode
    def hallo(self):
        print('Hallo mein Name ist {0} {1}'.format(self.Vorname,self.Nachname))

a = Student('Li','Ping',101)
b = Student('Lisa','Paul',102)
```

Klasse Studenten

```
class Student(object):

    noten = []                          #

    def __init__(self,Vn,Nn,MatNr):
        self.Vorname = Vn
        self.Nachname = Nn
        self.MatrikelNummer = MatNr
    #'\_\_init\_\_ wird automatisch beim Erstellen eines Objektes aufgerufen

    # Klassenmethoden
    def notehinzufuegen(self,note):
        self.noten.append(note)

    def hallo(self):
        print('Hallo mein Name ist {0} {1}'.format(self.Vorname,self.Nachname))

a = Student('Li','Ping',101)
b = Student('Lisa','Paul',102)
a.notehinzufuegen(1)
b.notehinzufuegen(4)

a.noten
```

```
[1, 4]
```

1.4 Bessere Lösung

Klasse Student

```
class Student(object):

    def __init__(self,Vn,Nn,MatNr):
        self.Vorname = Vn
        self.Nachname = Nn
        self.MatrikelNummer = MatNr
        self.noten = []

    def notehinzufuegen(self,note):
        self.noten.append(note)

    def hallo(self):
        print('Hallo mein Name ist {0} {1}'.format(self.Vorname,self.Nachname))


a = Student('Li','Ping',101)
b = Student('Lisa','Paul',102)
a.notehinzufuegen(1)
b.notehinzufuegen(4)

a.noten, b.noten
```

```
([1], [4])
```

Rules regarding self (from Lantangen's book, page 419):

1) Any class method must have self as first argument. (The name can be any valid variable name, but the name self is a widely established convention in Python.)
2) self represents an (arbitrary) instance of the class.
3) To access any class attribute inside class methods, we must prefix with self, as in self.name, where name is the name of the attribute.
4) self is dropped as argument in calls to class methods.

```
class Student(object):

    def __init__(self,Vn,Nn,MatNr):
        self.Vorname = Vn
        self.Nachname = Nn
        self.MatrikelNummer = MatNr
        self.noten = []

    def saghallo(self):
        print('Hallo mein Name ist {0} {1}'.format(self.Vorname,self.Nachname))

    def brief(self):
        return 'HALLO {0} Deine email ist {1}'.format(self.Vorname,self.email())

    def email(self):
        return '{0}.{1}@hhu.de'.format(self.Vorname,self.Nachname)

a = Student('Li','Ping',101)
b = Student('Lisa','Paul',102)
a.saghallo()
print(a.brief())
```

```
Hallo mein Name ist Li Ping
HALLO Li Deine email ist Li.Ping@hhu.de
```

1.5 Ein mathematisches Beispiel

Eine Klasse für Polynome

```
class Polynom(object):
#     """ Polynomklasse Beschreibung .....
#     """
    def __init__(self,dp):
#         """ Initialisiere Polynom mit einem Dictionary dp
#         Die Keys stehen fuer die Potenzen
#         die zugehoerigen Values fuer die Koeffizienten
#         a_2 X^2+ a_0  wird durch {0:a_0,2:a_2}
#         """
        self.dp = dp
        self.degree = max(dp.keys())

    def __repr__(self):
        polystr = ''
        for k in self.dp:
```

```
            polystr = polystr + '{0:+g}*X^{1}'.format(self.dp[k],k)
        return 'Polynom: ' + polystr


p1d = {0:1,10:2.5}
p2d = {0:2/3,5:34}
p = Polynom(p1d)
q = Polynom(p2d)
print(p)
help(Polynom)
```

```
Polynom: +1*X^0+2.5*X^10
Help on class Polynom in module __main__:

class Polynom(builtins.object)
 |  Methods defined here:
 |
 |  __init__(self, dp)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |
-----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

Klasse Polynom

```
class Polynom(object):
#     """ Polynomklasse Beschreibung .....
#     """
    def __init__(self,dp):
        if isinstance(dp,(int,float,complex)):
            dp = {0:dp}

        self.dp = dp
        self.degree = max(dp.keys())

    def __repr__(self):
        polystr = ''
        for k in sorted(self.dp):
            polystr = polystr + '{0:+g}*X^{1}'.format(self.dp[k],k)
        return 'Polynom: ' + polystr

    def __add__(self,other):
#         """ Addition zweier Polynomen
#         """
        spow = set(self.dp.keys())
        opow = set(other.dp.keys())
        pows = spow.union(opow)
        erg = dict()
```

```
        for k in pows:
            if k in spow:
                if k in opow:
                    erg[k] = self.dp[k] + other.dp[k]
                else:
                    erg[k] = self.dp[k]
            else:
                erg[k] =  other.dp[k]

        return Polynom(erg)


    def __sub__(self,other):
#         """ Substraktion zweier Polynomen
#         """
        spow = set(self.dp.keys())
        opow = set(other.dp.keys())
        pows = spow.union(opow)
        erg = dict()
        for k in pows:
            if k in spow:
                if k in opow:
                    erg[k] = self.dp[k] - other.dp[k]
                else:
                    erg[k] = self.dp[k]
            else:
                erg[k] = - other.dp[k]
        return Polynom(erg)


    def __call__(self,x):
#         """ Auswertung des Polynoms an der Stelle x
#         """
        return sum([self.dp[k]*x**k for k in self.dp])

p1d = {0:1,10:2.5}
p2d = {0:5/2,5:34}
p = Polynom(p1d)
q = Polynom(p2d)
c = Polynom(1)
p+q, p-q, p, p(2)
```

```
(Polynom: +3.5*X^0+34*X^5+2.5*X^10,
 Polynom: -1.5*X^0-34*X^5+2.5*X^10,
 Polynom: +1*X^0+2.5*X^10,
 2561.0)
```

Klasse Polynom

```
class Polynom(object):
#     """ Polynomklasse Beschreibung .....
#     """
    def __init__(self,dp):
        if isinstance(dp,(int,float,complex)):
            dp = {0:dp}
        self.dp = dp
        self.degree = max(dp.keys())
```

```
    def __repr__(self):
        polystr = ''
        for k in sorted(self.dp):
            polystr = polystr + '{0:+g}*X^{1}'.format(self.dp[k],k)
        return 'Polynom: ' + polystr

    def __add__(self,other):
        spow = set(self.dp.keys())
        opow = set(other.dp.keys())
        pows = spow.union(opow)
        pps = dict()
        for k in pows:
            if k in spow:
                if k in opow:
                    pps[k] = self.dp[k] + other.dp[k]
                else:
                    pps[k] = self.dp[k]
            else:
                pps[k] =  other.dp[k]
        return Polynom(pps)

    def __sub__(self,other):
#         """ Substraktion zweier Polynomen
#         """
        spow = set(self.dp.keys())
        opow = set(other.dp.keys())
        pows = spow.union(opow)
        erg = dict()
        for k in pows:
            if k in spow:
                if k in opow:
                    erg[k] = self.dp[k] - other.dp[k]
                else:
                    erg[k] = self.dp[k]
            else:
                erg[k] = -other.dp[k]
        return Polynom(erg)

    def __call__(self,x):
#         """ Auswertung des Polynoms
#         """
        return sum([self.dp[k]*x**k for k in self.dp])

    def __mul__(self,skalar):
#         """ Multiplikation eines Polynoms mit einem Skalar
#         """
        erg = dict()
        if isinstance(skalar,(int,float,complex)):
            for k in self.dp:
                erg[k] = self.dp[k] * skalar
            return Polynom(erg)


    __rmul__ = __mul__


p1d = {0:1,10:2.5}
p2d = {0:2/3,5:34}
p = Polynom(p1d)
```

```
q = Polynom(p2d)

2*p, p*2
```

```
(Polynom: +2*X^0+5*X^10, Polynom: +2*X^0+5*X^10)
```