

# 1 Merkblatt Python

## 1.1 Syntax

Programmblöcke werden unter der Anweisung eine Ebene drunter eingerückt geschrieben, statt häufig „{}“ Klammern umschlossen. Tabulatoren oder Leerzeichen sind erlaubt, dürfen aber nicht in der gleichen Datei zusammen vermischt werden.

Programmieranweisungen enden mit einem newline.

## 1.2 Datentypen

Die wichtigsten Datentypen

Type	Beschreibung	Syntax
str	Unveränderliche Liste von Buchstaben	'Python' "Python" """Python - The best script Language"""
unicode	Unveränderliche Liste von unicode chars	u'nur ASCII' u'nur ASCII'
list	Veränderbare Sequenz von verschiedenen Typen	[4, 4.0, 'string', True]
tuple	Unveränderliche Sequenz von verschiedenen Typen	(4, 4.0, "string", True)
set, frozenset	Ungeordnetes Set von verschiedenen Typen (keine Duplikate)	{4.0, 4, "string", True} Frozenset([4.0, 4, "string", True])
dict	Veränderbare key, value-Liste	{'key': 4.0, 'key2': False }
int	Nummer	37
float	Komma Zahl	3.7432
complex	Komma Zahl und imaginäre Zahl	7+4.2j
bool	True False	True, False

## 1.3 Kontrollstrukturen

### 1.3.1 if elif else

Bedingung braucht keine Klammer  
Doppelpunkt beendet die Bedingung

Beispiel:

```
if self.location.flat.currentRoom.isKitchen():  
    if self.isHungry():
```

```

        self.eat()
    elif self.isThirsty():
        self.drink()
    else:
        self.goTo(self.location.flat.LivingRoom)

```

### 1.3.2 while

Normale bekannte Schlaufe  
do...while existiert nicht

**Beispiel:**

```

loc = None
while loc == None:
    friend = self.friends.pop()
    call = self.call(friend)
    if call.simpleAsk(Question.presets.GoingOut):
        loc = call.askWhere(Question.presets.Where)
        self.goTo(loc)
        break

```

### 1.3.3 for

Loop über alle möglichen Listen:  
list, string, tuple, set usw.  
wird auch zum Aufbau einer Liste benutzt:  
>> [x for x in vec]

**Beispiel:**

```

for friend in self.friends:
    call = self.call(friend)
    if call.simpleAsk(Question.presets.GoingOut):
        loc = call.askWhere(Question.presets.Where)
        self.goTo(loc)
        break

```

## 1.4 Module

### 1.4.1 Module benutzen

Python kennt keine Includes

Python arbeitet mit Modulen in einem include path (sys.path list)

Pythonmodule sind Namespaces

Mit Import werden fremde Namespaces, Objekte usw. in den aktuellen Kontext des Programmcodes importiert, und somit verwendbar

**Beispiel Import:**

```

>>> import datetime
>>> datetime.datetime.today()
datetime.datetime(2009, 6, 19, 15, 37, 36, 681781)
>>> from hashlib import sha256
>>> sha256("mySecretPass").hexdigest()
'0f20a8005b5675e80bf8b6d89d24096c4bb0d0285eb884a74603920a9caecb8a'
>>> |

```

### Beispiel sys.path bearbeiten:

```
>> import sys
>> print sys.path
[... ..]
>> sys.path.add(../Users/user/pymodules")
>> sys.path.remove(../Users/user/pymodules")
>> sys.path.insert(0, ../test/pymodules")
```

Mit Hilfe einer .pth Textdatei mit einem Pfad kann der Standardpfad einfach erweitert werden.

## 1.4.2 Module erstellen

Ein Modul kann eine Datei oder ein Ordner sein

Einen Ordner und eine Datei definiert immer einen Namespace

Ein Ordner wird nur als Module erkannt wenn er eine `__init__.py` Datei besitzt.

## 1.5 Funktionen

Funktionsdefinition beginnen mit dem Schlüsselwort „def“

Eine Definition, gibt keine Angaben zum Rückgabewert

Verschiedene Parameter Möglichkeiten.

```
def doSomething (foo, bar=None):
    return foo
doSomething(0, 1)
def doSomethingByName (foo=1, bar=2):
    return bar
doSomethingByName(bar=3)
def doSomethingByArgs (*args, **kwargs):
    """ doSomethingByArgs
    returns args and kwargs
    """
    return args, kwargs
doSomethingByArgs(foo=0,bar=3,xyz=5)
print doSomethingByArgs.__doc__
```

## 1.6 Klassen und Objekte

Python ist eine 100%-ige Objektorientiertesprache, trotzdem zwingt es einem nicht auf objektorientiert zu programmieren.

Eine Klassendefinition beginnt mit dem Schlüsselwort „class“

Die Vererbungen werden in einem anschliessenden tuple - normale Klammern mit Komma getrennt definiert.

Die Definition endet wie bekannt mit einem „:“

## Beispiel:

```
class Lampe(StromKonsument):
    """definiert eine lampe und
    erbt die funktionen und attribute von der Klasse StormKonsument
    nicht definiert in diesem Beispiel."""

    def __init__(self, birne, on=False):
        """Python objekt konstuktor
        die Lampe braucht um erstellt zu werden eine Birne
        zusätzlich kann definiert werden ob sie schon angeschaltet ist oder nicht.
        """
        self.birne = birne
        self.on = on
        super(Lampe, self).__init__(power_need=self.birne.power)

    def switch(self):
        if on == True:
            on = False
        else:
            on = True
```

### Benutzung

```
>> birne = Stromsparbirne(Watt=20, Lichtwert=5500K)
>> lampe = Lampe(birne)
>> lampe.switch()
```

Erstellt eine Birne, erstellt eine Lampe mit der Birne und schaltet die Lampe an.

## ***1.7 Weiterführende Einführung***

Eine gute und umfangreiche Einführung findet ihr unter dem frei erhältlichen Buch unter:

<http://diveintopython.org/>

und für python 3 gibts eine neue Version unter:

<http://diveintopython3.org/>