

Einführung in Python und in das Webframework Django




Ursprung und Einsatz

- Ursprung:

- 1980 entwickelt als ABC
- 1990 Neuentwicklung – "Python"
- 2000 Version 2.0 – Unicode und GC
- 2009 Version 3.0 - CleanUp

- Einsatz:

- Google (youtube, spider, AppEngine)
 - Yahoo (maps, groups)
 - Maya, Blender3d, Gimp, OpenOffice
 - Desktop (z.B. BitTorrent)
- 

Warum Python

- Every Thing is a Object (100% OO)
- Paradigmafrei (Modular, OOP usw)
- AOP Support
- Anwendungsunabhängig (Web, Desktop, 3D, Realtime, Mathematik)
- Riesige Modul auswahl
- Interobilität (CORBA, CLR, JAVA, ICE usw)
- CLR, JAVA und PERROT implementationen



Django Web framework

The Web framework for perfectionists with deadlines.

Django makes it easier to build better Web apps more quickly and with less code.

<http://djangoproject.com>



Syntax

- Übersichtliche Programmierung basierend auf Formatierung als Programmieranweisungen
- Weniger Code aufgrund von Einsparungen überflüssiger Zeichen

```
class BlockObjectTemplate(models.Model):
    ... name = models.CharField(max_length=200)
    ... desc = models.TextField()
    ... template = models.TextField()

    ... def __unicode__(self):
    ...     ... return u"%s (BlockObjectTemplate)" % self.name

    ... def render(self, config, pageContent):
    ...     ... fields_config = config["items"]
    ...     ... context = {"BlockObjectTemplate": self}
    ...     ... for k, v in fields_config.items():
    ...     ...     field = self.fields.get(name=k)
    ...     ...     context[k] = field.fetch(v)
    ...     ... tpl = DjTemplate(self.template)
    ...     ... return tpl.render(RequestContext(get_current_request()), context)

    ... def get_field_options(self):
    ...     ... options = []
    ...     ... for field in self.fields.all():
    ...     ...     field_options = field.options()
    ...     ...     options.append(field_options)
    ...     ... return options

class BlockObjectTemplateField(models.Model):
    ... OBJECT = "Object"
    ... QUERYSET = "QuerySet"
```

Blöcke – 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)
```



Block - def

- Funktionsdefinition
- Flexibles Parameterhandling (objC)


```
>>> def doSomething (foo, bar=None):
...     return foo
...
>>> doSomething(0, 1)
0
>>> def doSomethingByName (foo=1, bar=2):
...     return bar
...
>>> doSomethingByName (bar=4)
4
>>> def doSomethingByArgs (*args, **kwargs):
...     """ doSomethingByArgs
...     returns args and kwargs
...     """
...     return args, kwargs
...
>>> doSomethingByArgs(foo=1, bar=2)
((), {'foo': 1, 'bar': 2})
>>> doSomethingByArgs.__doc__
' doSomethingByArgs \n     returns args and kwargs\n'
```

Blöcke - for

- Loop über alle möglichen Listen:
 - list, string, tuple, set usw.
- wird auch zum Aufbau einer Liste benutzt:
 - `>> [x, x+y for x in vec, for y in vec.reverse()]`

Beispiel:

```
for friend in self.friends:  
    call = self.call(friend)  
    if call.simpleAsk(Question.preset, out):  
        loc = call.askWhere(Question.preset, Where)  
        self.goTo(loc)  
        break
```



Block - 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
```



Block - class

- Klassen definition

```
class Lebewesen:  
    BLUTGRUPPEN = ("A", "AB", "0" #usw)  
    def __init__(self, puls=50, blutgruppe="A", dna=None):  
        self.puls = puls  
        if blutgruppe in self.BLUTGRUPPEN:  
            self.blutgruppe = blutgruppe  
        else:  
            raise CanNotLife()  
        self.dna = dna
```

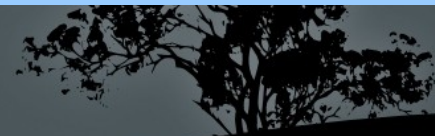
```
    def die(self):  
        self.puls = 0
```

```
class Mensch(Lebewesen):  
    """ Class Mensch  
    currently not implemented  
    """  
    pass
```



Datentypen

Type	Beschreibung	Syntax
str	Unveränderliche liste von Buchstaben	'Python' "Python" """Python - The best script Language"""
unicode	Unveränderliche Liste von Bytes	u'nur ASCII' u"nur ASCII"
list	Veränderbare Sequenz von verschiedenen Typen	[4, 4.0, 'string', True]
tuple	Unveränderliche Sequenz von verschienen 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



Module - Import

- Python arbeitet mit einem Packagesystem
- Dateien und Ordner definieren Namespaces
- `sys.path` definiert die Ordner welche die Module beinhalten

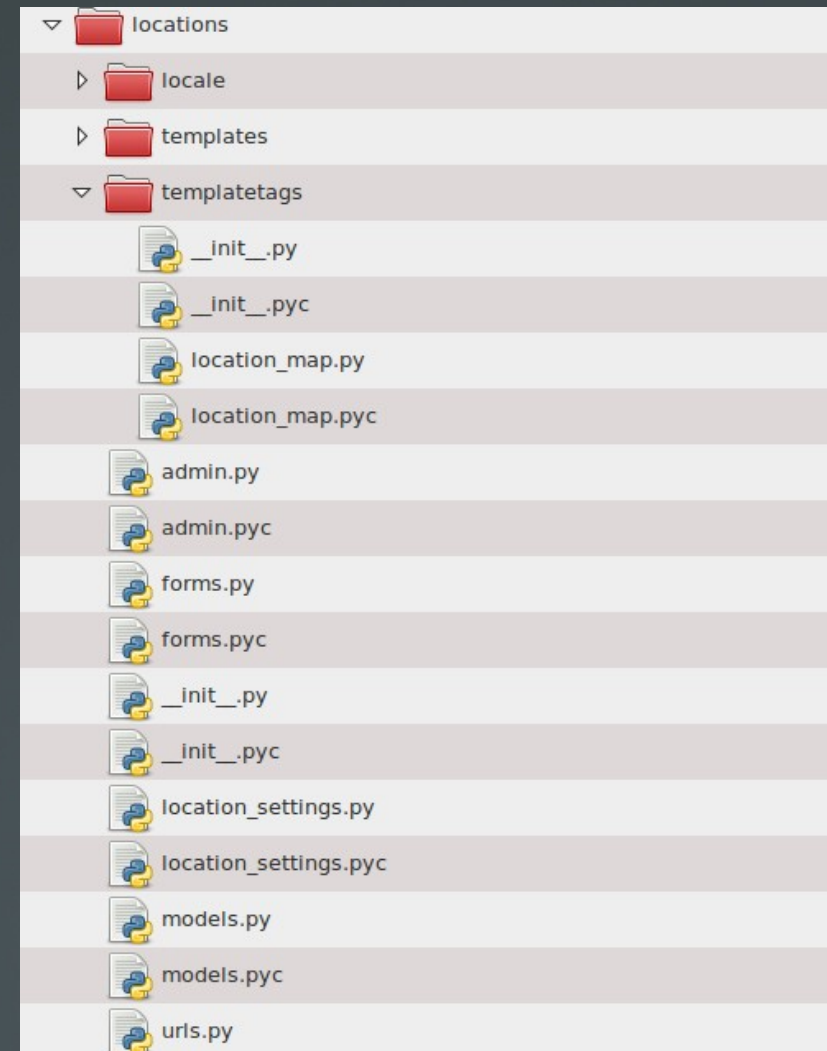
```
>>> import datetime
>>> datetime.datetime.today()
datetime.datetime(2009, 6, 19, 15, 37, 36, 681781)
>>> from hashlib import sha256
>>> sha256("mySecretPass").hexdigest()
'0f20a8005b5675e80bf8b6d89d24096c4bb0d0285eb884a74603920a9caecb8a'
>>> |
```

Module - Struktur

- Ordner braucht `__init__.py`
- Jede Datei ein Namespace
- Überordner von `location` muss im `sys.path` sein
- Import

Beispiel:

```
from locations.models  
import Location
```



Mathematik

- Python wird häufig auch für realtime und wissenschaftliche berechnungs Programmen verwendet
- Viele auch komplexe mathematische Berechnungen standartmässig unterstützt

```
>>> 3 * 3
9
>>> 10 / 2
5
>>> 10 ** 3
1000
>>> m1 = set([x for x in range(0, 10) ])
>>> m2 = set([x for x in range(5, 15) ])
>>> m1 > m2
False
>>> m1 | m2
set([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
>>> m1 & m2
set([8, 9, 5, 6, 7])
>>>
```

Decorators - AOP

```
>>> import logging
>>> def methodLogging(meth):
...     log = logging.getLogger()
...     def func(*args, **kwargs):
...         #log.debug("function %s called with args: %s and kwargs: %s" % (meth, args, kwargs))
...         print "function %s called with args: %s and kwargs: %s" % (meth, args, kwargs)
...         ret = meth(*args, **kwargs)
...         print "function %s finished with return value: %s" % (meth, ret)
...         #log.debug("function %s finished with return value: %s" % (meth, ret))
...     return func
...
>>>
>>> @methodLogging
... def hallo(name="Felix Marthaler", anrede="Herr"):
...     print ("Hallo %s %s" % (anrede, name))
...
>>> hallo(anrede="Frau")
function <function hallo at 0x233c2a8> called with args: () and kwargs: {'anrede': 'Frau'}
Hallo Frau Felix Marthaler
function <function hallo at 0x233c2a8> finished with return value: None
>>>
```

Externe Libraries

- Image Manipulation – PIL, complex
-> gimp
 - WEB – Zope, Django, TurboGears,
Werkzeug
 - Multimedia (Video, Audio) –
Gstreamer
 - GUI – TK, GTK, QT usw
 - Game – PyGame
 - Game and 3D – Blender3D, pysoy
 - SQL – SQLAlchemy
- 