

Content-Management-Systeme

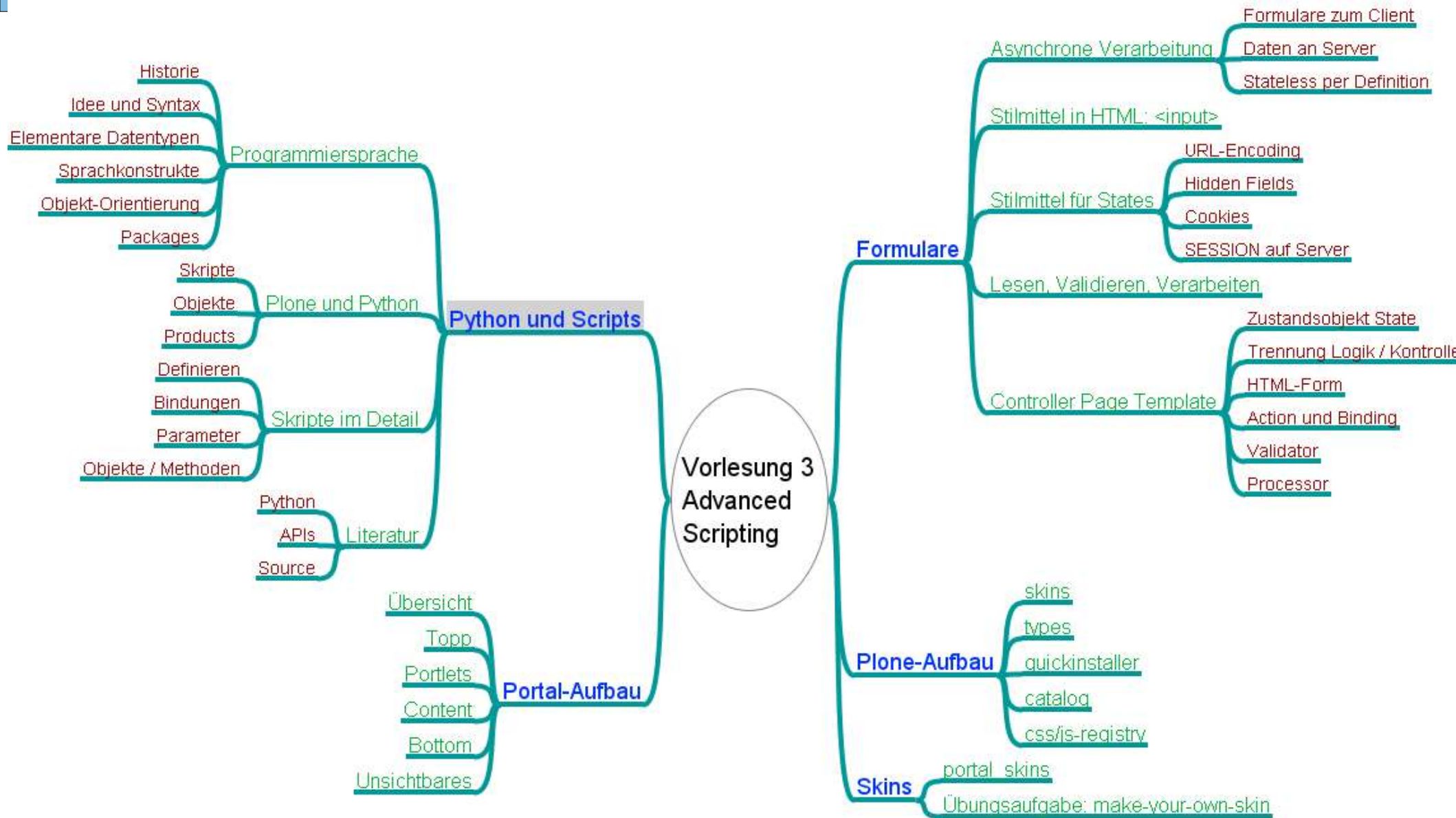
Dipl.-Inform. Roman Jansen-Winkeln

Vorlesung 4: Zope Skripting für Fortgeschrittene

Inhalt und Organisation

- Übungen
 - Plone an HTW in Betrieb nehmen
- Nächste Woche
 - Themen für Projektarbeit
 - Wünsche? Ideen?

Inhalt und Organisation



Python

Universelle Programmiersprache
vom Skript bis zum OO-Allrounder

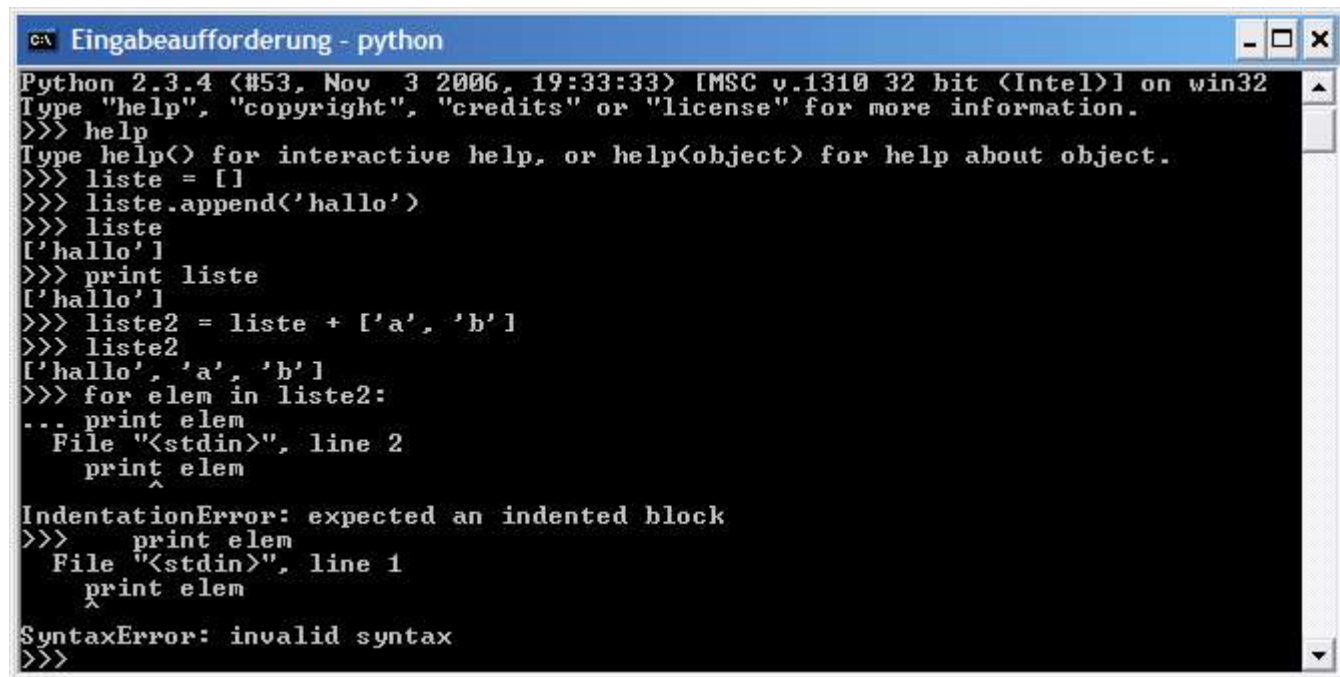
Python: Historie

- Python ist eine Programmiersprache; Unterstützt
 - objektorientierte Programmierung
 - aspektorientierte Programmierung
 - funktionale Programmierung.
- Entwickelt Anfang der 1990er Jahre
 - Guido van Rossum
 - Centrum voor Wiskunde en Informatica in Amsterdam
 - Benannt nach *Monty Python's Flying Circus*
- Python → Zwischencode → Interpreter
Bekannte Implementierungen:
 - CPython
 - Jython
 - PyPy

Python: Historie

- Verfügbarkeit: Windows, Unix, Linus, Symbian, Embedded, ...
- Kommandozeile-Interpreter
 - Einfacher Interaktiver Zugang
 - Inkrementeller Byte-CODE-Compiler und Python-VM-Interpreter

- Diverse GUIs:
 - Windows-IDE
 - Eclipse
 - PDB



```
C:\> Eingabeaufforderung - python
Python 2.3.4 (#53, Nov  3 2006, 19:33:33) [MSC v.1310 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> help
Type help() for interactive help, or help(object) for help about object.
>>> liste = []
>>> liste.append('hallo')
>>> liste
['hallo']
>>> print liste
['hallo']
>>> liste2 = liste + ['a', 'b']
>>> liste2
['hallo', 'a', 'b']
>>> for elem in liste2:
...     print elem
      File "<stdin>", line 2
        print elem
        ^
IndentationError: expected an indented block
>>>     print elem
      File "<stdin>", line 1
        print elem
        ^
SyntaxError: invalid syntax
>>>
```

Python: Datenstrukturen

- Primitive Datenstrukturen

- Integer, Floats, Character, Strings (variable Länge), Boolean
- Logisch äquivalent: [] == {} == 0 == "" == False

- Listen

- [], [1, 2, 3], ['abc', 'def'], ['xyz', 1, [2,'a'], 5]
- Einfacher Zugriff: L = [1,2,3,4] ; L[0] == 1 ; L[-1] == 4 ; L[2] = 3
- L[1:2] == [2,3] ; L[2:] == [3,4] ; L[:] == *Kopie von L*
- Tupel: L = (1,2,3,4) = *Read-Only-Liste*
Set, Frozenset: S = set([1,2,3,4]) analog

- Dictionary alias Assoziative Liste

- D = {'a' : 1 , 'hallo' : [1,2] , 'id' : 'd4711'}
- D['a'] == 1 ; D.keys() == ['a', 'hallo', 'id'] ; D.values() == *analog*
- D.items() = [('a',1), ('hallo', [1,2]), ('id', 'd4711')]

Python: Kontrollstrukturen

- Zuweisung
 - Einfach: `a = 5 ; oFarbe = "blau" ; flag = True`
 - Verkettet: `a = b = c = 5`
 - Parallel: `a , b , c = 1 , 3, 17` entspricht `(a,b,c) = (1,3,17)`
 - Typisierung: implizit bei erster Zuweisung
- Blöcke:
 - Einrücken bildet neuen Block!!
- Bedingungen: `if (Cond) ... elif (Cond) ... else ...`
- Schleifen: `for (Liste) ... else ... ; while (Cond) ... ; break;continue`
- Exceptions: `try ... except ... finally ...`

- Fakultätsfunktion in C:

- ```
int fakultaet(int x)
{
 if (x > 1)
 return x * fakultaet(x-1);
 else
 return 1;
}
```

- Fakultätsfunktion in Python:

- ```
def fakultaet(x):
    if x > 1:
        return x * fakultaet(x - 1)
    else:
        return 1
```

- Robuste Eingabeschleife

- while True:
 try:
 num = raw_input("Eine Zahl eingeben: ")
 num = int(num)
 break
 except ValueError:
 print "Eine _Zahl_, bitte!"
 except KeyboardInterrupt:
 break

- while True:
 try:
 num = raw_input("Eine Zahl eingeben: ")
 except KeyboardInterrupt:
 break
 try:
 num = int(num)
 break
 except ValueError:
 print "Eine _Zahl_, bitte!"

Python: Funktionen

- Funktionen / Methoden (Funktion in Klasse)
 - Def *fname* (p1, p2 = 5, **args) :
 anweisungen
 return "hallo"
 - Aufruf: a = *fname*(1, 2) ; a = *fname*(1, p2 = 3) ;
 a = *fname*(1, p2 = 3 , p3 = 5, p4 = '4711')
- Einfache Funktionale Programmierung
 - Funktionen als Parameter, z.B. in 'map', 'reduce', ...
 - Implizites 'map' im Listenkonstrukt
zahlen = [1, 2, 3, 4]
potenzen = [2 ** n for n in zahlen]

Python: Funktionen (Fortsetzung)

- Fortgeschrittene Funktionale Programmierung
 - Funktionen wie Werte behandeln ; Variable Fkts.-aufrufe zulassen
 - Statische Bindung, Lambda-Closures, Continuation Passing
- Beispiel
 - Definition:

```
def make_pot_closure(basis) :  
    def pot_closure(zahlen):  
        return [ basis ** n for n in zahlen ]  
    return pot_closure
```
 - Anwendung 1. Teil: Lambda-Closure erzeugen
`pot4 = make_pot_closure(4) ≡ Funktion, die 4er-Potenzen erzeugt`
 - Anwendung 2. Teil: Lambda-Closure anwenden
`potenzen = pot4([1,2,3,4])`

Python: Klassen

- Python ist OO bis in den Kern: `','.join([1,2,3,4])`
- Hello-World-Klasse:
 - `class HelloWorld (HelloPlanet) :`
 `greeting = "Hello World"`
 `def hello(self) :`
 `"""gibt greeting aus"""`
 `return self.greeting`
 - Anwendung:
 `hello1 = HelloWorld()`
 `print hello1.hello()`
 - Namenskonventionen:
 1. Methoden-Argument = 'self'
 Namen mit '_'-beginnend = *private*
 Superklasse per Name aufrufen, z.B. `HelloPlanet.hello()`
 - Interface über Klassenvariable: `implements = HelloPlanet.implements`
 - Multiple Vererbung: `class HelloWorld (HelloPlanet, VisualClass)`

Python: Klassenbibliotheken, Module

- Importieren

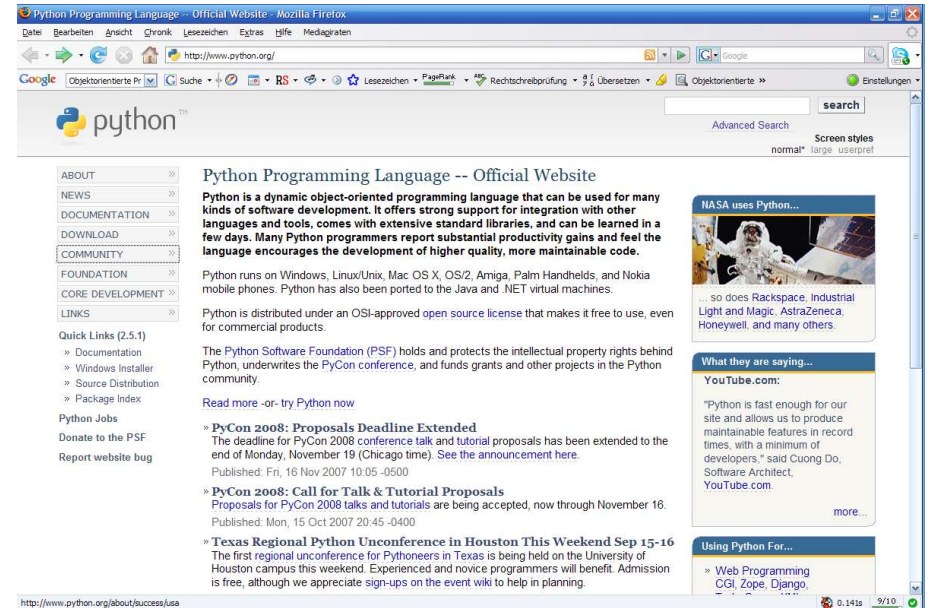
- import Zope
from Products.Archetypes.Interface import getAllAccessors
from Zope import *
- Products.Archetypes.Interface ≡ Pfad Products/Archetypes/Interface.py
- __init__.py – Datei macht Verzeichnis zur Bibliothek

- Standardbibliothek: das Extra-Plus für Python

- Betriebssystem (os, os.path), DateTime, Regular Expressions
- XML, HTML, SAX- und DOM-Parser
- Math: array, decimal, math, random, md5,
- Internet: http, ftp, imap, pop, smtp, telnet, xmlrpc, twister, ldap, webdav,
- Datenbanken: anydbm, dbhash, sqlalchemy, ZODB, ...
- GUI (Tkinter), Bilder (PIL), Threads,

Python: Weblinks

- <http://python.org>
 - Reicht eigentlich. Alles da!
 - www.python.org/~guido
Findet Guido van Rossum auch ;))
- <http://wiki.python.de/>
 - Es geht auch deutsch
 - Nettes Sprungbrett
 - Section für Umsteiger sowie Tipps und Tricks
- [http://de.wikipedia.org/wiki/Python_\(Programmiersprache\)](http://de.wikipedia.org/wiki/Python_(Programmiersprache))
 - Gute Literaturliste
 - Als Sprungbrett auch ok.



Python: Erfolgsgeschichten

- Google

- *"Python has been an important part of Google since the beginning, and remains so as the system grows and evolves. Today dozens of Google engineers use Python, and we're looking for more people with skills in this language."*

said Peter Norvig, director of search quality at Google, Inc.

- NASA

- *"NASA is using Python to implement a CAD/CAE/PDM repository and model management, integration, and transformation system which will be the core infrastructure for its next generation collaborative engineering environment. We chose Python because it provides maximum productivity, code that's clear and easy to maintain, strong and extensive (and growing!) libraries, and excellent capabilities for integration with other applications on any platform. All of these characteristics are essential for building efficient, flexible, scalable, and well-integrated systems, which is exactly what we need. Python has met or exceeded every requirement we've had"*

said Steve Waterbury, Software Group Leader, NASA STEP Testbed.

- Thawte

- *„Python makes us extremely productive, and makes maintaining a large and rapidly evolving codebase relatively simple,"* said Mark Shuttleworth.

- MIT

- Informatik-Ausbildung beginnt mit Python!

Python: Plone revisited

- Plone ist eine erweiterbare Python-Anwendung
 - Basis: Programmiersprache Python
 - Erweitert um Python-Standardbibliothek
 - Zope als Web-Applikations-Server =
Klassen für Dokumenttypen, User, Sessions, ...
Methoden-Interfaces = Zope-API
 - CMF = Content-Management-Factory
Generisches CMS-Gerippe in Zope
Weitere Klassen für Content, Catalogs, Skins, ...
Spezialisieren Zope-Basisklassen
 - PLONE = Spezielle CMF-Ausprägung
Weitere Klassen für Content, Catalogs, Skins, ...
Spezialisiert und nutzt CMF-, Zope- und Standardbibliothek-Klassen

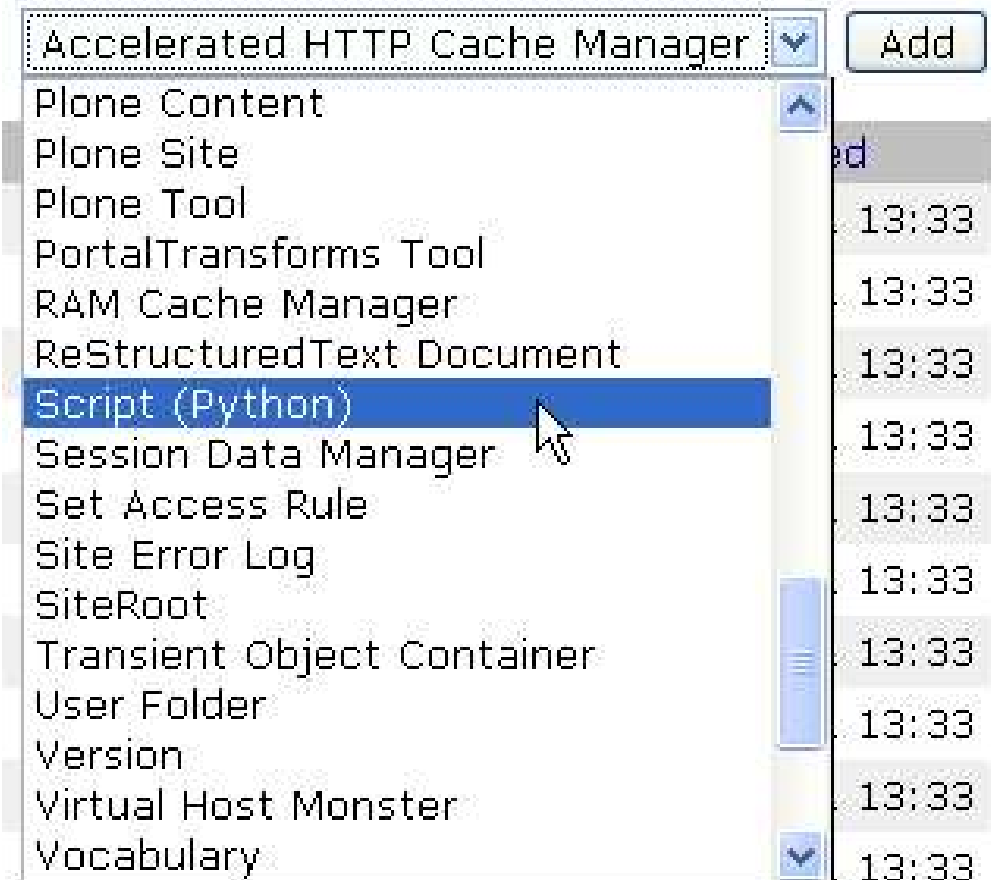
Python: Plone revisited

- Python in Plone verwenden
 - TAL-Ausdrücke: „python: here.hello()“
Ausdrücke, Funktionen und Methoden auswerten
[siehe letzte Vorlesung]
 - Dokumenttyp: Script (Python)
Python-Skript analog Page-Template aufrufen
(Verwendet 'restricted' Python weil TTW)
[HEUTE]
 - Externe Methoden-Objekte
komplexerer Dokumenttyp als 'Script (Python)',
Python ohne Einschränkungen
[Selbststudium]
 - Python-Produkte:
Python-Klassenbibliothek, basierend auf Zope-, CMF- oder Plone-
Objekten;
[nächste Vorlesung]

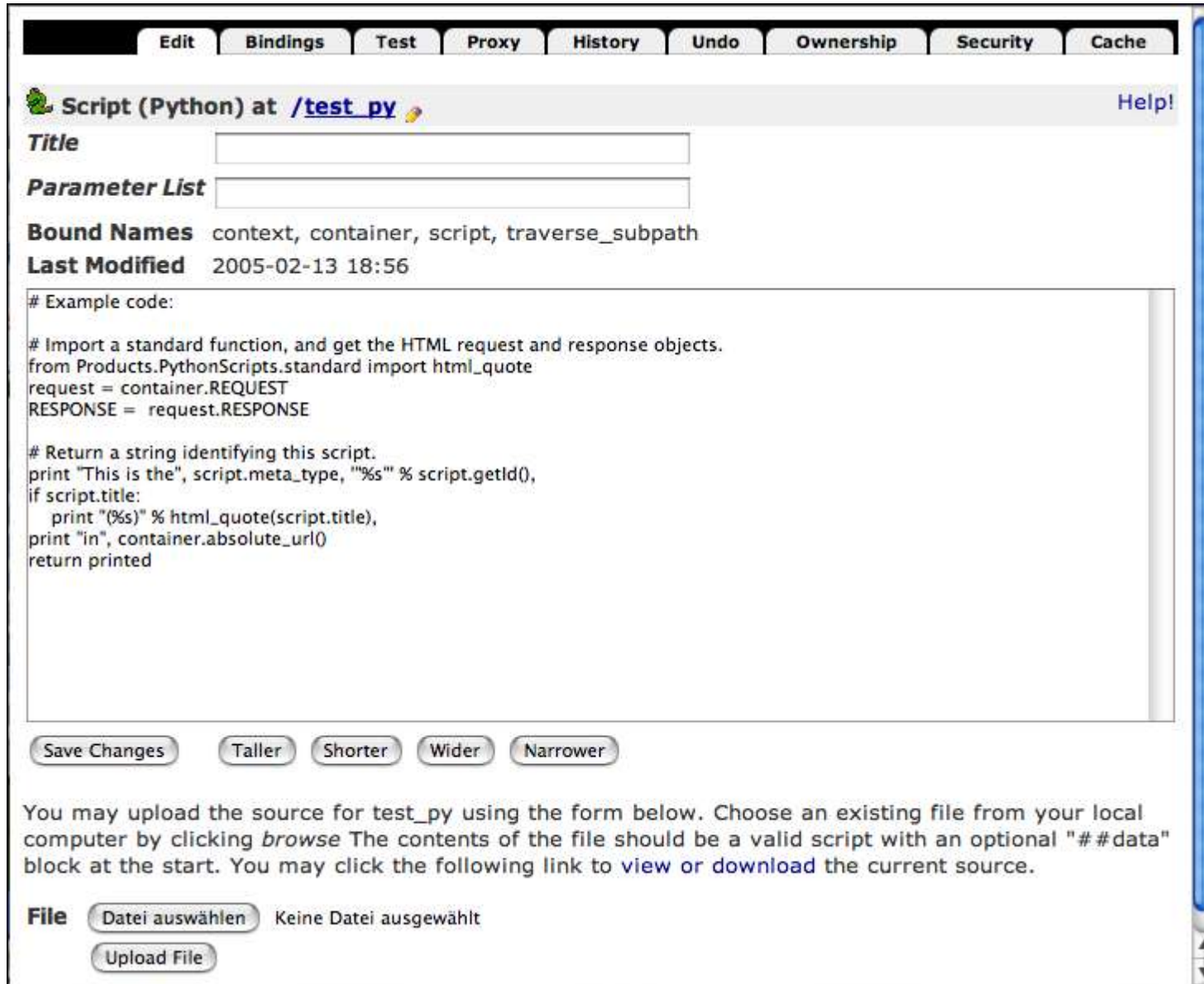
Python - Scripts

Python (Script): Skript anlegen

- Skript im ZMI anlegen
 - Über „Script (Python)“ und „Add“ anlegen



Python (Script): Skript bearbeiten



The screenshot shows a web-based editor for a Python script. At the top, there is a navigation bar with tabs: Edit, Bindings, Test, Proxy, History, Undo, Ownership, Security, and Cache. Below this, the title bar reads "Script (Python) at /test_py" with a "Help!" link on the right. The main editing area contains several fields: "Title" (empty), "Parameter List" (empty), "Bound Names" (context, container, script, traverse_subpath), and "Last Modified" (2005-02-13 18:56). A large text area contains the following Python code:

```
# Example code:

# Import a standard function, and get the HTML request and response objects.
from Products.PythonScripts.standard import html_quote
request = container.REQUEST
RESPONSE = request.RESPONSE

# Return a string identifying this script.
print "This is the", script.meta_type, "%s" % script.getId(),
if script.title:
    print "(%s)" % html_quote(script.title),
print "in", container.absolute_url()
return printed
```

Below the code area are five buttons: "Save Changes", "Taller", "Shorter", "Wider", and "Narrower". A paragraph of text explains that users can upload source code for "test_py" using a form below, choosing an existing file from their local computer by clicking "browse". It also mentions that the contents should be a valid script with an optional "##data" block at the start, and provides links to "view" or "download" the current source. At the bottom, there is a "File" section with a "Datei auswählen" button (which shows "Keine Datei ausgewählt") and an "Upload File" button.

Python (Script): Details

- Python Script at: Pfad plus ID
- Title: Beschreibender Titel, dient nur der Anzeige im ZMI
- Parameter List
 - Liste mit Parametern, die das Script erwartet
 - Syntax analog Funktionen/Methoden
 - Stellungparameter, Schlüsselwortparameter mit Default
 - Bindung durch Zope-Object-Request-Broker
 - Redundant im Dictionary REQUEST zu finden
- Bound Names
 - Durch Zope-Laufzeitumgebung vorbesetzte Variable
 - Kennen wir schon von Zope Page Templates
 - context, container, script, namespace, traverse_subpath

Python (Script): Aufruf

- Über Reiter „Test“ im Editor
 - Zope erzeugt Formular für Parameter-Abfrage
- Through-the-Web: `http://../test_py`
 - Parameter in URL-Notation `http://../test_py?p1=hallo&p2=4711`
 - Interaktiver Python-Interpreter TTW
- Durch TAL-Expression
 - `<em tal:content="here/test_py">Test-Ausgabe`
 - Geht nur bei parameterlosen Skripten
- Durch TAL-Python-Expression
 - `<em tal:content="python: here.test_py('hallo',p2=4711)>Test`

Python (Script): Größeres Beispiel

- Vorbemerkung:
 - Scripts verschaffen den Zugriff zu den **APIs** von
 - Python-Standardbibliothek: <http://docs.python.org>
 - Zope-API: http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/AppendixB.stx
 - Plone-API http://www.ifpeople.net/fairsource/courses/material/apiPlone_en
und <http://plone.org/documentation/manual>
- Details zu den APIs werden in der Vorlesung nicht besprochen
- APIs sind notwendig, um Vorlesung/Übung/Praktikum zu verstehen.
- Selbststudium und aktive Suche ist notwendig!

Python (Script): Größeres Beispiel

- Aufgabenstellung:
 - Inhaltsobjekte, die bis zu 5 Tage alt sind, finden.
 - Script findet Objekte
 - Page Template zeigt Objekte an
 - (Trennung von Layout und Logik)
 - Schnittstelle: Liste mit Dictionary-Records

Python (Script): Größeres Beispiel

- Script-Code recentlyChanged

```
##title=recentlyChanged
##parameters=objects
from DateTime import DateTime

now = DateTime()
difference = 5 # in Tagen
result = []

for object in objects:
    diff = now - object.bobobase_modification_time()
    if diff < difference:
        dct = {"object":object,"diff":int(diff)}
        result.append(dct)
return result
```

Python (Script): Größeres Beispiel

- Ergebnis-Code

```
[  
  {  
    'diff': 1,  
    'object': <PloneFolder instance at 02C0C110>  
  },  
  {  
    'diff': 4,  
    'object': <PloneFolder instance at 02FE3321>  
  },  
  ...  
]
```

Python (Script): Größeres Beispiel

- Aufruf-Code `recentlyChangedView`

```
<ul>
```

```
  <li tal:repeat="updated python: context.recentlyChanged(context.contentValues())">
```

```
    <a href="#"
```

```
      tal:attributes="href updated/object/absolute_url"
```

```
      tal:content="updated/object/title_or_id">Der Titel des Artikels</a>
```

```
    Vor <em tal:content="updated/diff" /> Tagen
```

```
  </li>
```

```
</ul>
```

- Ergebnis:

- HTML-Punkt-Aufzählung für alle neueren Objekte
- Titel/ID des Objekts und Änderungszeit
- HREF verlinkt auf das Objekt selbst

- Bitte in Übung nachvollziehen !!

Server-basierte HTML-Formulare

Controller Page Templates und Freunde

Formulare: Roundtrip asynchron

- Interaktive Eingabe in Web-Applikationen

- 1) Server erzeugt/benutzt HTML-Seite mit Formularfeldern

- 2) Client zeigt Formular an

- 1) Benutzer füllt Formular autonom aus,
oder lässt es sein, ruft andere Seite auf, öffnet neues Fenster , ...

- 1) Benutzer sendet ausgefülltes Formular an Server

- 2) Aktive Serverkomponente verarbeitet Formular

- 1) Nächster Interaktionsschritt ... Server sendet HTML-Seite ...

Formulare: Roundtrip asynchron

- Umsetzung in Plone

- 1) Server erzeugt HTML-Seite aus Page Template

- 2) Client zeigt Formular an

- 1) Benutzer füllt Formular autonom aus,
oder lässt es sein, ruft andere Seite auf, öffnet neues Fenster , ...

- 1) Im `<form action="...">` wird URL eines Scripts angegeben

- 2) ORB bindet Formularfeld-Werte an Script-Parameter;
Script in Plone auswerten

- 1) Nächster Interaktionsschritt ... Server sendet HTML-Seite ...

Formulare: Roundtrip Beispiel 1

- **Ausschnitt Formular**

```
<form method="post" action="recentlyChangedDays">  
  <legend>Änderungen der letzten wie viel Tage?</legend>  
  <input type="text" name="tage" value="3"/>  
  <input type="submit" name="ok" value="Anzeigen"/>  
</form>
```

- Fragt Benutzer nach den Tagen, in denen sich das Objekt geändert haben darf. Feld-Name= „tage“
- Zeigt Submit-Knopf als „Anzeigen“. Feld-Nam= „ok“
- Sendet Eingabe per „post“ an „./recentlyChangedDays“

Formulare: Roundtrip Beispiel 1

- **Script-Code Neu**

```
##title=recentlyChangedDays
##parameters=tage
result = context.recentlyChanged(context.contentValues(), tage)
return result
```

- **Script-Code angepasst**

```
##title=recentlyChanged
##parameters=objects, difference = 5
from DateTime import DateTime

now = DateTime()
## difference = 5 # in Tagen
result = []
```

Formulare: Roundtrip Beispiel 1 Verbesserung

- Viel Verbesserungs-Potenzial
 - `result = ...` ist zuwenig;
das liefert nur die Liste, nicht die Aufbereitung
 - Eingabe ist nicht robust;
Laufzeitfehler, falls Eingabe kein Integer
 - Typfehler;
ZORB bindet '3' (String) an „tage“, nicht 3 (Integer)
- Typfehler:
 - Erweiterte Interpretation von „name“-Attribut bei `<input>` als „*name:type*“.
 - `<input type="text" name="tage:int" value="3"/>`
 - Erlaubte Typen: boolean, int, log, float, text, list, tuple, tokens,lines, date, required
 - Achtung: falsche Eingabe führt zu Server-Fehler!

Formulare: Roundtrip Beispiel 1 Verbesserung

- Eingabe robust machen
 - Client-Seitig mit Javascript testen;
Bei Typ-Überprüfung hinnehmbar; Komplexere Logik --> Server
 - Typ-Erweiterung aus „name“-Attribut --> Scheußliche Fehlermeldung
 - Validierungs-Logik in Script einbauen;
Sauberste Lösung, Bestandteil von Controller Page Templates
- Resultat zu wenig
 - Korrekte Arbeitsweise wäre:
 - 'Difference' für 'recentlyChanged' dauerhaft speichern
 - 'recentlyChangedView' von Client erneut aufrufen lassen
 - SESSION-Objekt bietet Dictionary mit session-persistenten Speicher
 - HTTP-Redirect lässt Client Seite anfordern

Formulare: Roundtrip Beispiel 2 mit SESSION

- Script-Code Neu

```
##title=recentlyChangedDays
##parameters=tage
session = context.REQUEST.SESSION
session['tage'] = tage
return context.REQUEST.RESPONSE.redirect('recentlyChangedView')
```

- Script-Code angepasst

```
##title=recentlyChanged
##parameters=objects
from DateTime import DateTime
session = context.REQUEST.SESSION

now = DateTime()
difference = session.get('tage',5) # entspricht session['tage'] mit Default 5
result = []
```

Formulare: Exkurs HTTP/REDIRECT

- *Tafelbild zur Wirkung von*
 - *context.REQUEST.RESPONSE.redirect*
 - *Ablauf REDIRECT*
- *Vorschlag Übungsaufgabe*
 - *Erweitern um handgemachte Validierung*
 - *Redirects mit Fehlermeldungen*
z.B. in SESSION speichern, dass Fehler aufgetreten ist,
redirect auf Formular-Template,
im Formular-Template SESSION abfragen, ob Fehlerflag gesetzt ist,
mit tal:condition eine Fehlermeldung zusätzlich einblenden
- *Ergebnis:*
 - *Umständliche Programmierung*
 - *Systematischerer Ansatz*

Formulare mit Controller

- Der Ablauf
 - Anzeige eines Controller Page Templates
 - Benutzer füllt aus und schickt ab
 - Validierungs-Skript(e) ausführen
 - Aktions-Skript(e) ausführen in Abhängigkeit von
 - Submit-Button und
 - Validierungsergebnis
- Controller bedeutet:
 - Zusätzliche globale Variable **state**
 - Zusätzliche Kontrollstrukturen
 - Zusätzliche state.Methoden und state/Expression

- Form-Code angepasst

```
<form method="post"
      tal:define="errors options/state/getErrors"
      tal:attributes="action template/id;">
  ...
  <input type="hidden" name="form.submitted" value="1" />
</form>
```

Formulare mit Controller

- Eigenes Validierungs-Skript 'test_validator“

```
##title=Ein Skript zum Überprüfen einer Zahleneingabe
##parameters=
num = context.REQUEST.get('tage', None)           # Alternative zu 'paramters=tage
try:
    int(tage)
except ValueError:
    state.setError("tage", "Eingabe fehlt", new_status="failure")
except TypeError:
    state.setError("tage", "Keine Zahl eingegeben.", new_status="failure")
if state.getErrors():
    state.set(portal_status_message="Bitte Fehler korrigieren.")
return state
```

Formulare mit Controller: Validator registrieren

The screenshot shows a web application interface with a top navigation bar containing buttons for 'Edit', 'Test', 'Validation', 'Actions', 'Properties', 'History', 'Undo', 'Ownership', 'Security', and 'Cache'. Below the navigation bar, the page title is 'Controller Page Template at /portal_skins/custom/test_cpt'. The main heading is 'Form and Script Validators'. A descriptive paragraph states: 'This tool lets you set or override the default sequence of validators that are executed when a Controller Page Template is submitted or before a Controller Python Script is executed. [Learn more.](#)'

The interface is divided into two main sections:

- Edit Form / Script Validator Overrides:** This section contains a table with three columns: 'Template Context', 'Button', and 'Validators'. The first row shows 'test_cpt' in the 'Template Context' column, 'Any' in the 'Button' column, and 'test_validator' in the 'Validators' column. Below the table are 'Save' and 'Delete' buttons.
- Add a New Form / Script Validator Override:** This section contains form fields for 'Template' (set to 'test_cpt'), 'Context type' (set to 'Any'), 'Button', and 'Validators'. An 'Add' button is located below these fields.

Formulare mit Controller: Aktion registrieren

The screenshot shows a web application interface with a navigation bar at the top containing buttons for Edit, Test, Validation, Actions, Properties, History, Undo, Ownership, Security, and Cache. Below the navigation bar, the current page is identified as 'Controller Page Template at /portal_skins/custom/test_cpt'. The main heading is 'Form and Script Actions', followed by a descriptive paragraph: 'This tool lets you set or override the default actions that are executed when a Controller Page Template is submitted or a Controller Python Script returns. Learn more.' Below this is a section titled 'Edit Form/Script Action Overrides' which contains a table with columns: Template/Script Status, Context, Button, Action, and Argument. The table has one row for 'test_cpt' with values: success, Any, (empty), redirect_to, and (empty). There are 'Save' and 'Delete' buttons below the table. At the bottom, there is a section 'Add a New Form Action Override' with a form for 'Template/Script test_cpt' and a 'Status' field.

Controller Page Template at /portal_skins/custom/test_cpt

Form and Script Actions

This tool lets you set or override the default actions that are executed when a Controller Page Template is submitted or a Controller Python Script returns. [Learn more.](#)

Edit Form/Script Action Overrides

Template/Script Status	Context	Button	Action	Argument
<input type="checkbox"/> test_cpt	success	Any		redirect_to

Add a New Form Action Override

Template/Script test_cpt

Status

Formulare mit Controller: E-Mail an Webmaster schicken

- Das Formular (Auszüge)

```
<html
  xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-US" lang="en-US"
  metal:use-macro="here/main_template/macros/master">
  <body>
    <div metal:fill-slot="main"
      tal:define="errors options/state/getErrors;">
    ...
    <form method="post"
      tal:attributes="action template/id;">
    ...
    <div class="field"
      tal:attributes="class python:test(error_email_address, 'field error', 'field')"
      tal:define="error_email_address errors/email_address|nothing;">
    <label>Your email address</label>
    <span class="fieldRequired" title="Required">(Required)</span>
    <div class="formHelp">Enter your email address.</div>
    ...
```

Formulare mit Controller: E-Mail an Webmaster schicken

- Das Formular (Auszüge)

...

```
<div class="formHelp">Enter your email address.</div>
```

```
<div tal:condition="error_email_address">  
    <tal:block content="error_email_address">Error</tal:block>  
</div>
```

```
<input type="text" name="email_address"  
    tal:attributes="tabindex tabindex/next;  
        value request/email_address|nothing" />
```

```
</div>
```

... *Eingabeblock für alle Variable wiederholen* ...

```
<div class="formControls">
```

```
<input class="context" type="submit" tabindex=""  
    name="form.button.Submit" value="Submit" />
```

```
</div>
```

```
<input type="hidden" name="form.submitted" value="1" />
```

Formulare mit Controller: E-Mail an Webmaster schicken

- Validierer (Auszüge)

...

```
email = context.REQUEST.get('email_address', None)
```

```
if not email:
```

```
    state.setError('email_address', 'Email is required', new_status='failure')
```

```
if state.getErrors():
```

```
    state.set(portal_status_message='Please correct the errors.')
```

```
return state
```

Formulare mit Controller: E-Mail an Webmaster schicken

- Action (Auszüge)

...

```
mhost = context.MailHost
emailAddress = context.REQUEST.get('email_address')
administratorEmailAddress = context.email_from_address
comments = context.REQUEST.get('comments')

# the message format, %s will be filled in from data
message = "From: %s \n To: %s \n Subject: Website Feedback \n %s \n URL: %s "
```

format the message

```
message = message % (emailAddress, administratorEmailAddress, \
                    comments, context.absolute_url())
mhost.send(message)

screenMsg = "Comments sent, thank you."
state.setKwargs( {'portal_status_message':screenMsg} )
return state
```

Eigene Dokumenttypes programmieren: Archetypes