# Plone を使う人のためのPython入門

Page TemplateとScript(Python)で使う
Python

かとうまさや a.k.a. jack

# Agenda

- TALとPython Script で使う Python
  - Pythonについてのちょっとしたおさらい
  - 便利なDateTime(datetime.datetimeより便利)
  - portal_catalogをいじる
  - コンテンツをスクリプトで扱う
  - ユーザ情報を取得する
    - Proxy Roleについて
  - TALでロジック書くな(でも書ける)
    - [python:]禁止の件
  - 外部メソッド(External Method)もやっぱり必要
  - デバッグ時の手法
    - dir() を External Methodにおいておく

# Python とは

- 覚えやすく、使いやすい
  - 小さな言語仕様
  - わかりやすい構文
  - 強力なリストプロセッシング
- 自由すぎる
  - 動的型付け
  - メンバアクセス
    - C++でいえば全部public で virtual
  - プラットフォーム
  - 超豊富なライブラリ
- 一部、不自由
  - インデントは文法
  - 日本語処理
    - 今日では概ね問題なし。エンコードとstr でマルチバイトを扱わない限りに置いては
- (Script)Python
  - Zope や Page Template で用いられる制限付き(Restricted)Python
  - 使えるモジュールは極めて制限されている
  - 追加されているモジュールもある

# Python 超入門

- 変数型(組み込みオブジェクト)
  - 数(immutable)
    - int, long
    - float
    - 真偽値
  - シーケンス
    - tuple(immutable)
    - list(mutable)
    - 文字列(immutable)
  - マップ
    - dict(mutable)
      - いわゆる連想配列
  - set(Zope内では使えない)
    - set(mutable)
    - flozenset(immutable)
- 関数
  - def
- クラス
  - class

- 変数について
  - 型宣言は不要
  - 基本的に参照
    - 全部ポインタ
  - 正確には「識別子」なので、関数も変数も関係ない
- (im)mutableって？
  - 「変更(不)可能」ということ
  - 数字がimmutable?
    - 1は常に1
    - x = 1; x = x + 1 なら x は1への参照から2への参照へ変化する
  - dict のキーはimmutable
- インデントは？
  - 4。スペースに統一したほうがよい
    - ZMIでは面倒なので2でやることもある(個人的見解)
- list or tuple?
  - 変更不要ならtuple。関数によっては変換する必要があることも

# Python 超入門

- **制御構造**
  - if c1: p1()
    elif c2: p2()
    elif c3: p3()
    else: p4()
  - for x in range(10):
    - if x == 3: continue
      if x > 10: break
  - else: # for **の** else
    - break**が使われなく**for**が終了したときのみ実行**
  - while condition:
    else:
    - continue, break, else **については** for **と同様**
    - **あまり使わない気がする**
- try:
  - **エラーがでるかもしれない処理**
- except TypeError:
  - TypeError**の処理**
- except:
  - **ともかくエラー処理**
- else:
  - **エラーがなかったときの処理**
- try:
  - **エラーがでるかもしれない処理**
- finally:
  - **クリーンナップなど、ともかく実行しなければならない処理**
- def func(x, y, z)
  if not x : return None
  return x + y + z

# Pythonコードの例示

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

def factorial(n):
    """n の階乗を返す。
    intで充分ならintで、そうでなければlongの値を返す

    >>> [factorial(n) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> [factorial(long(n)) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> factorial(30)
    265252859812191058636308480000000L
    >>> factorial(30L)
    265252859812191058636308480000000L
    >>> factorial(-1)
    Traceback (most recent call last):
        ...
    ValueError: n must be >= 0
    float値でもいいが、値はintでないとだめ :
    >>> factorial(30.1)
    Traceback (most recent call last):
        ...
    ValueError: n must be exact integer
    >>> factorial(30.0)
    265252859812191058636308480000000L

    値が大きすぎるときはOverflowを返す:
    >>> factorial(1e100)
    Traceback (most recent call last):
        ...
    OverflowError: n too large
    """
    import math
    if not n >= 0:
        raise ValueError("n must be >= 0")
    if math.floor(n) != n:
        raise ValueError("n must be exact integer")
    if n+1 == n:  # catch a value like 1e300
        raise OverflowError("n too large")
    result = 1
    factor = 2
    while factor <= n:
        result *= factor
        factor += 1
    return result

def _test():
    import doctest
    doctest.testmod()

if __name__ == "__main__":
    _test()
```

# よく使う関数

- zip()
  - 直交リストを返します
  - zip(range(9),range(9),range(9))[0] は [0,0,0]です
  - いわゆる二次元配列 x の直交リストを求めたいときには zip(*x)とします
- dict()
  - 辞書型を作ります
    - dict({'one': 2, 'two': 3})
    - dict(one=2, two=3)
    - 他にもいろいろ呼びかたがあります
- dir(obj)
  - objの属性を取得します。
  - オブジェクトの使い方を探れます
  - Zope上では使えません(cheatアリ)
- filter(), map()
  - 最近ではリスト内包表現が主流？
- len()
  - 文字列やリストの長さを返します
- int(), float()
  - 文字から数字への変換
  - try: 〜 except :が必須です
- max(), min()
  - そのままです(笑)
- str(), repr(), unicode()
  - strは8bit文字列、unicodeはunicode文字列です。reprはオブジェクトのあるべき姿を示します
- range(), xrange()
  - range(start, stop, step)で数字のリストを作ります。引数が一つのときはstopのみ、二つのときはstart, stopとなります
- tuple(), list()
  - 呼出し先の関数に合わせるために変換するときくらいです
- lambda
  - 主な出番はsort()かもしれません。

# よく使うメソッド

- x in seq
  - リスト、タプル、文字列、マップ型のキー内に xがあればTrueです
- string.find(x),.index(x)
  - string内に xがあればインデックスを返します
  - 無いときはfind()は -1を返し index()はValueErrorです。
- string.join() .split() # 後述
- string%(list)
  - printfです
  - "%d:%s"%(10,'abc')
    - "10:abc"
- list.append(x), pop()
  - append(x) と x = pop() でFIFOとなります
- list.reverse()
  - listを逆順にし、値は返しません
  - listの元の値は保存されません
    - reversed()を使う
- list.sort(cmp, key, reverse)
  - cmpを指定するとsort順を関数またはlambdaで指定できます。
  - keyも同様
  - reverse は True のときcmpの逆順となります。
  - list.reverse()同様、元の値は保存されません。
    - やはり、sorted()がある
- dict.items(), .keys()
  - for key, value in dict.items()の形でよく使います
  - .keys()はキーのみのリストです
- dict.update({key:value})
  - TALでときどき使うかも・・・
- listに関する注意
  - a = range(9)
    b = a
    b[2]=99
    とすると a[2]も99です
    b = a[:] とすると別のリストとしてコピーされます

# portal_catalogをいじる

- items = portal_catalog.searchResult(
  REQUEST=None, # request
  sort_on, # sort index
  sort_limit, # **最大値**
  sort_order, # 'reverse'
  path, # PATH**の一部**
  Portal_Type, #**探す**type
  Type, #
  meta_type, #
  ,...)
- for x in items:
  - obj = x.getObject()
    **とすると、カタログから実オブジェクトが取れる**

# スクリプトでコンテンツをいじる

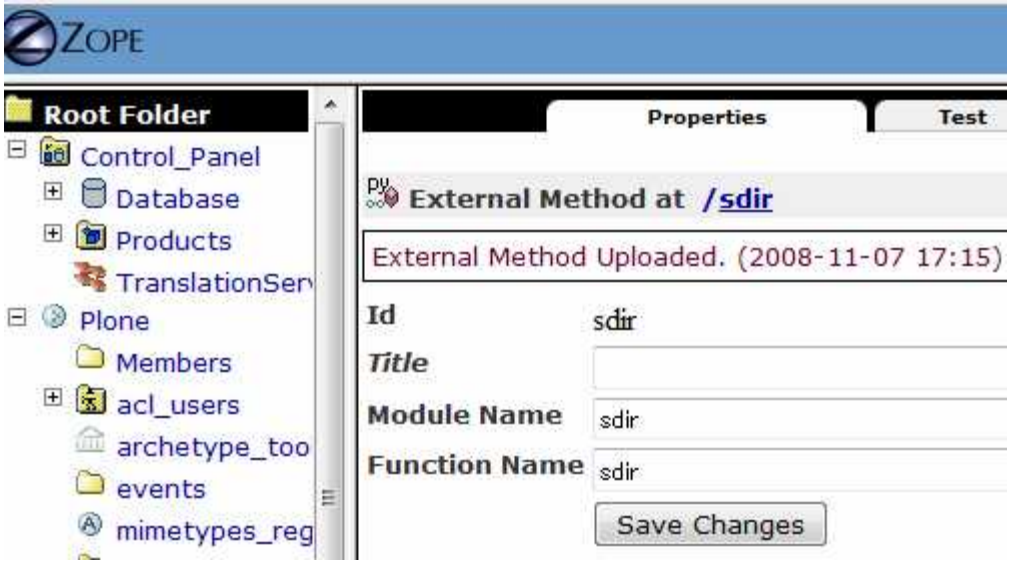
- ポータル内のイベントを探して、そのDescription を "This is Event Description" とする
  - for x in catalog.searchResults(REQUEST=request, Type="Event"):
    obj = x.getObject()
    # print obj
    # for y in context.sdir(obj):
    # prop = str(y)
    # if 'Description' in prop: print prop

    obj.setDescription("This is event description")
    obj.indexObject()
- では、このsetなんちゃらはどうやって探すのか？

- →気合とgrep
  - ではなんなんで・・・

# 開発、デバッグに便利なExternalMethod

- ExternalMethod とはZopeの機能で、フル機能のPythonのスクリプトを
  - $INSTANCE_HOME/Extensions/ 以下に置いて
  - ZMI から追加することにより使えるようになる機能




- dir() ってとても便利だけど、Script(Python)では使えない。
- じゃぁ・・・ ExternalMethod にしよう
  - % more sdir.py

```
def sdir(x=None):
    return dir(x)
if __name__ == "__main__":
    print sdir("abc")
```

# どうせ大したマニュアルないんだし

Script (Python) at /Plone/portal_skins/custom/demoPortalCatalog

*Title*

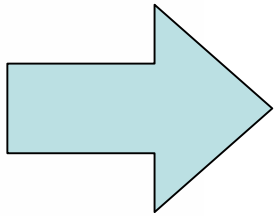
*Parameter List*

**Bound Names** context, container, script, traverse_subpath

**Last Modified** 2008-11-07 17:09

```
# 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

for x in catalog.searchResults(REQUEST=request, Type="Page"):
  obj = x.getObject()
  print obj
  for y in context.sdir(obj):
    prop = str(y)
    if prop[:3] == 'get': print prop
```



```
<ATDocument at front-page>
getActionInfo
getActionInfo_roles_
getAttribute
getAttributeNode
getAttributeNode_roles_
getAttribute_roles_
getAttributes
getAttributes_roles_
getAvailableLayouts
getAvailableLayouts_roles_
getBRefs
getBRelationships
getBackReferenceImpl
getBackReferences
getCatalogs
getCharset
getCharset_roles_
getChildNodes
getChildNodes_roles_
getContentType
getContentType_roles_
getDefault
getDefaultLayout
getDefaultLayout_roles_
getDefaultPage
getDefaultPage_roles_
getDefault_roles_
getDocumentComparisons
getDocumentComparisons_roles
```

**検索エンジン等で調べてもわからない場合、**
**こんな感じでアタリを付けて、CMFPlone、ATCT以下**
**の既存Product での使われかたが一番参考になるかも**

# Page Template と TAL

- Page Template は xhtml を生成するためのツール
  - 基本的には xhtml + 動的生成のためのタグが付く

```
<table>
 <tr tal:repeat="item here/cart">
  <td tal:content="repeat/item/number">1</td>
  <td tal:content="item/description">Widget</td>
  <td tal:content="item/price">$1.50</td>
 </tr>
</table>
```

  - TAL文は以下のようになっている
    - tal:attributes エレメントのアトリビュートを変更
    - tal:define 変数の定義
    - tal:condition 条件が成立したときのみエレメントを生成
    - tal:content エレメントの内容を指定
    - tal:omit-tag エレメントそのものは削除するが、内容は残す
    - tal:on-error エラー処理
    - tal:repeat 変数を定義してエレメントを繰り返し(for文のようなもの)
    - tal:replace エレメントを削除して、内容を指定。
    - 但し、content と replaceは同一エレメント内に共存できない

- TALの優先順位
  - define
  - condition
  - repeat
  - content or replace
  - attributes
  - omit-tag
- <tal:name> エレメント
  - 名前空間やループを構成するには、以前は <span tal:omit-tag="" tal:define...> などと使われていたが、今日では <tal:namedef define="... ">と記述
  - <tal:name>...</tal:name>と閉じタグを合わせる必要がある
  - tal:define でなく 単に define とする
- path式
  - 名前空間の識別子名を指定する式
  - <p tal:content="request/value"/>
- string式
  - 文字列を指定する式。その際に${var} としてpath式を指定できる
  - <p tal:content="string:name is ${name}"/>
- python式
  - restricted pythonの右辺値を指定する式
  - <p tal:content="python: '%6.2f'%floatval/>
  - 諸悪の根源w

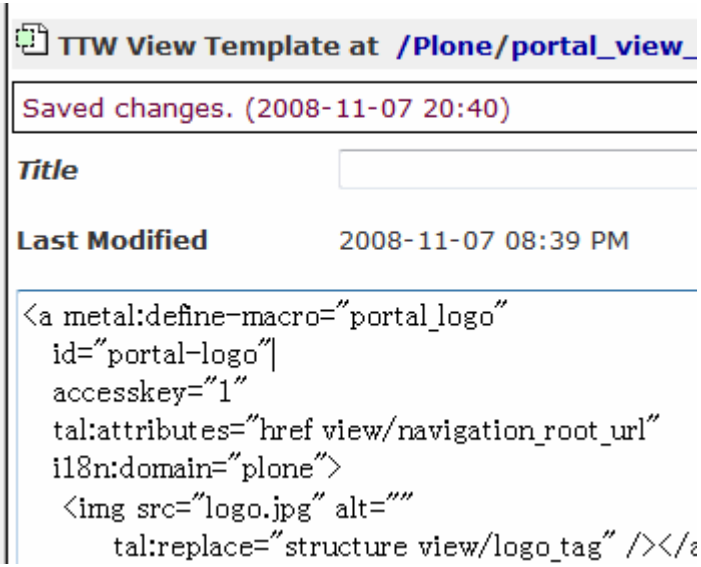
# ユーザ情報の取得とProxy Role

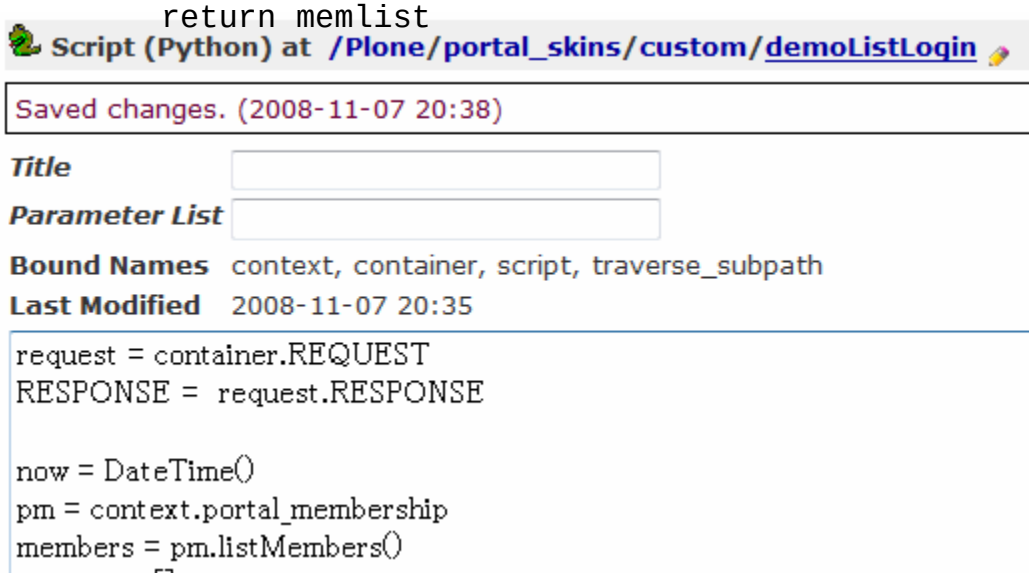
- **ポータルロゴのカスタマイズ**

```
<a metal:define-macro="portal_logo"
id="portal-logo"   accesskey="1"
tal:attributes="href
view/navigation_root_url"
i18n:domain="plone">
<img src="logo.jpg" alt=""
tal:replace="structure view/logo_tag"
/></a>
<p tal:repeat="x
python:here.demoListLogin()"
tal:content="python:'%s:%s'%(x[0],x[1]
.strftime('%Y/%m/%d %H:%M'))"/>
```

- **ユーザのログイン情報を取得**

```
request = container.REQUEST
RESPONSE =  request.RESPONSE
now = DateTime()
pm = context.portal_membership
members = pm.listMembers()
pr = context.portal_registration
memlist = [[unicode(x),
x.getProperty('login_time'), int((now
- x.getProperty('login_time'))*86400)]
for x in members if (now -
x.getProperty('login_time')) < 1]
memlist.sort(key=lambda x:x[1],
reverse=1)
return memlist
```



# しかしユーザによって結果が違う


# Portal_membershipなどにはManager必須

- そこで、Proxy_Role
  - 基本的にはセキュリティレベルを下げるものなので、注意してください。

# Fizz Buzz 問題

- 1〜100の整数が与えられた場合
  - 3で割り切れる場合は数字の代わりに 'Fizz' を
  - 5で割り切れる場合は数字の代わりに 'Buzz' を
  - 両方で割り切れる場合は数字の代わりに 'FizzBuzz' を
  - そうでない場合は数字を print しろ

```
for x in range(1, 101):
    output=''
    if x%3 == 0: output = 'Fizz'
    if x%5 == 0: output += 'Buzz'
    if not output: output = x
    print output
```

# これをTALで書くと

- <p tal:define="fb python:['Fizz', 'Buzz'];
  fblist python:fb + [''.join(fb),];
  nlist python:range(1, 101);
  isfb python: lambda x:
  (x%3==0) * 1 + (x%5==0) * 2;
  fizzbuzz python:[isfb(n) and
  fblist[isfb(n)-1] or n for n in nlist];"
  tal:repeat="row fizzbuzz"
  tal:content="row"/>

- **まぁ、書けないことはないです**
  - **但し、数ヶ月経ったあとにわかるかというと・・・**
  - tal で python: **タグを使ってゴリゴリ書くのは基本的にはやめたほうがいいです**

Management System is ©
lemarks of the Plone Found

1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
Fizz
22
23
Fizz
Buzz

# 便利なDateTime

- 本家？Pythonにはないけれど、Zope内で使えるDateTime型は便利です。
  - now = DateTime()
    - 引数無しだとnow を返します
  - DateTime().strftime('%Y/%m/%d')
    - フォーマット文字列も使えます
  - DateTime().rfc822()
    - RFC 822 形式を返します
  - wpday = DateTime("2008/11/07")
    - datetime.datetimeと違い、文字列コンストラクタが使えます
  - wpday.rfc822()
    - wpdayはDateTime型なので、メソッドは自由に使えます
  - print "now - wpday:", now – wpday
    - timedelta を使うことなく、加算、減算ができます。なお、日付同士の計算結果は「日数」単位のfloat値です

# コネタ集

- 文字列の結合
  - ':'.join(list) とすると、list 内の文字列を : で結合
    - ':'.join('a','b','c')
      - 'a:b:c'
    - ':'.join(range(3))
      - TypeError発生。文字列でないとだめ
      - ':'.join((unicode(x) for x in range(3))) とすればOK
    - 'SELECT * FROM table WHERE %s'%' AND '.join(['hogeid=%s'%x for x in hogeidlist])
      - 'SELECT * FROM table WHERE hogeid=xx AND hogeid=yy AND ....'
        - きっとデバッグのときに苦労する
- 文字列の分割
  - 文字列.split(separator)を使う
    - 'a:b:c'.split(':')
      - ['a', 'b', 'c']
      - リストで帰ってくる
    - カンマで区切られた文字列を、文字列、float, int のリストに変換する

```
a, b = 'a,b,1,3.0,3.14,2,99,c', []
for x in a.split(','):
    try:
        y,z =  float(x), int(x)
        if y == z: b.append(z)
        else: b.append(y)
    except:
        b.append(x)
```

      - ['a', 'b', 1, 3, 3.1400000000000001, 2, 99, 'c'] が得られる

# コネタ集

- <p tal:content="structure request" tal:condition="python:1"/>
  - TALのデバッグ時によく使います。
  - condition は "python:1" にしておくと、1,0の切替がラクなのでこうしてます
  - デバッグが終ったら<p>タグごと消しましょう

form

cookies

__ac 'eQw3L7DbkKccXC+SFj5Fm84eYvkgdXNlcjI='

lazy items

SESSION <bound method SessionDataManager.getSessionData of <SessionDataManager at /session_data_manager>>

other

| | |
|---|---|
| LANGUAGE | 'en' |
| URL | 'http://192.168.192.182:8080/Plone/front-page/document_view' |
| _pts_is_rtl | False |
| SERVER_URL | 'http://192.168.192.182:8080' |
| LANGUAGE_TOOL | <Products.PloneLanguageTool.LanguageTool.LanguageBinding instance at 0xb2294f6c> |
| AUTHENTICATION_PATH | 'Plone' |
| traverse_subpath | [] |
| _ec_cache | {-1308999572: <Products.PageTemplates.Expressions.ZopeContext object at 0xb22b322c>} |
| __ac | 'eQw3L7DbkKccXC+SFj5Fm84eYvkgdXNlcjI=' |
| PUBLISHED | <FSPageTemplate at /Plone/document_view used for /Plone/front-page> |
| ACTUAL_URL | 'http://192.168.192.182:8080/Plone' |
| URL0 | http://192.168.192.182:8080/Plone/front-page/document_view |
| URL1 | http://192.168.192.182:8080/Plone/front-page |
| URL2 | http://192.168.192.182:8080/Plone |
| URL3 | http://192.168.192.182:8080 |
| BASE0 | http://192.168.192.182:8080 |
| BASE1 | http://192.168.192.182:8080 |
| BASE2 | http://192.168.192.182:8080/Plone |
| BASE3 | http://192.168.192.182:8080/Plone/front-page |
| BASE4 | http://192.168.192.182:8080/Plone/front-page/document_view |

environ

| | |
|---|---|
| HTTP_ACCEPT | 'image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-ms-application, application/vnd.ms-xpsdocument, application/xaml+xml, application/x-ms-xbap, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*' |
| CONNECTION_TYPE | 'Keep-Alive' |
| HTTP_USER_AGENT | 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET CLR 2.0.50727; .NET CLR 3.0.04506)' |
| HTTP_REFERER | 'http://192.168.192.182:8080/Plone' |
| SERVER_NAME | 'cfy7jck' |

# コネタ集

- メールアドレスをログインIDにしたい
  - 以下のようなScript(Python)を作成
    pr = context.portal_registration
    pr.manage_editIDPattern('^[A-Za-z0-9][A-Za-z0-9_@.-]*$')
  - Manager権限を持つIDで一回実行すればOK
- ほとんどのものは tal:attributesでいじれる
  - onclick などを動的生成して、条件によって呼ぶ javascriptを変更することもできる
  - onmouseover で javascript で半透明のウィンドウを表示させるより、title 属性を使ったほうがいい。
- PageTemplate内でのモジュールの使いかた
  - <tal:main define="dt modules/DateTime;
    now python:dt.DateTime();">
    ...
    </tal:main>
  - でも読みづらくなりますよ(笑)
- 三項演算子
  - Python的には condition and true_exp or false_expが使われます
    - でも、true_exp が FALSE値になるとき期待通りの動作をしません
  - Script(Python), TAL的には test(condition, true_exp, false_exp)が使えます