Jython クイック・リファレンス

このマニュアルでは、**Jython** スクリプト言語の参照を提供します。このマニュアルは、統合のシナリオで Jython スクリプトを使用する開発者を対象としています。

このマニュアルの構成

このマニュアルには次の章が含まれています。

- **第 1 章「基礎」**では、Jython 構文の基礎を説明します。
- **第 2 章「Oracle Data Integrator での Jython の使用」**では、Oracle Data Integrator での Jython の使用方法を説明します。
- **第 3 章「例」**では、Jython のサンプル・スクリプトを提供します。

包括的な Jython のドキュメントは、http://www.jython.org から入手できます。

基礎

Oracle Data Integrator と Jython

Jython（Python の Java バージョン）は、オブジェクト指向のスクリプト言語です。Jython スクリプトは、Java 仮想マシンを使用するすべてのプラットフォームで実行できます。

Oracle Data Integrator の実行エージェントには、Jython インタプリタが含まれています。このエージェントは、Jython で記述されたスクリプトを実行できます。

Oracle Data Integrator では、Jython を使用してプロシージャまたはナレッジ・モジュールを記述できる他、Jython コードを SQL、PL/SQL、OS コールなどと併用することも可能です。

Jython を利用することで、Oracle Data Integrator のプログラミング能力は劇的に向上しています。文字列、リスト、ディクショナリの複雑な処理、FTP モジュールのコール、ファイルの管理、外部 Java クラスの統合などを実行できます。

注意: Jython コードを KM プロシージャ・コマンドで使用するには、テクノロジを一貫して `Jython` に設定する必要があります。

留意点

Jython プログラムを記述する際の基本的な規則は次のとおりです。

コードの実行

文は、シーケンス内の制御構造（`if`、`for`、`while`、`raise`）またはファンクション・コールまで実行されます。

ブロック

ブロックは、インデント・レベル（空白またはタブ）が同じである行によって定義されます。

文

文は行の最後で終わりますが、行の最後が¥の場合や、文が()、[]、{}または'''で囲まれている場合は、行が変わっても継続されます。複数の命令を;で区切ると同じ行に記述できます。

コメント

コメントはハッシュ文字#で始まり、物理行の最後で終わります。

ドキュメンテーション文字列

関数、モジュールまたはクラスが文字列定数で始まる場合、その文字列はオブジェクトの__doc__属性に格納されます。

例

Hello World を表示する単純なプログラム

```
# Assign a value to a string
s = 'Hello World'
# Display the value
print s
```

Hello World を 4 回表示するプログラム

```
s = 'Hello World %d'
for i in range(4):
   j = i * 2
   print s % j
```

キーワード

次に示す識別子は、Jython で予約済の単語つまりキーワードです。

```
and       del       for       is        raise
assert    elif      from      lambda    return
break     else      global    not       try
class     except    if        or        while
continueexec      import    pass
def       finally   in        print
```

次の点にも注意してください。

- 1 つの文は、単一の行で記述する必要があります。1 つの文を複数の行に分割する場合は、¥（円）記号を使用する必要があります。
- カッコ()、大カッコ[]または中カッコ{}で囲まれている式は、円記号を使用せずに複数の物理行に分割できます。
- セミコロン（;）で区切ると、複数の文を同じ行に記述できます。
- ハッシュ文字（#）で始まるコメントは、文字列リテラルには含まれず、行の終わりまで続きます。

演算子

演算子（優先順位の高い順）

演算子	説明
`lambda args: expr`	匿名関数コンストラクタ
`x or y`	論理 OR
`x and y`	論理 AND
`not x`	論理 NOT

`x<yx<=yx>yx>=yx==y`
`x!=yx<>y`
`x is y  x is not y`
`x in s  x not in s`
	比較演算子（等しい、等しくない、同一オブジェクト、シーケンスに含まれるかどうかなど）

`x|y`	ビット単位 OR
`x^y`	排他 OR
`x&y`	ビット単位 AND
`x<<y  x>>y`	左シフト、右シフト
`x+yx-y`	加算/連結、減算
`x*yx/yx%y`	乗算/繰返し、除算、剰余
`x**y`	指数
`+x、-x、~x`	恒等、単項 NOT、ビット単位補数
`s[i]s[i:j] s.attr` `f(...)`	インデックス、範囲、属性限定、ファンクション・コール
`(...) [...] {...} '...'`	タプル、リスト、ディクショナリ、文字列への変換

データ型

数値

- **10 進整数**: 1234、1234567890546378940L　（または l）
- **8 進整数**: 0177、01777777777777777777L（0 で始まる）
- **16 進整数**: 0xFF、0XFFFFffffFFFFFFFFFFL（0x または 0X で始まる）
- **LONG 整数**（精度の制限なし）: 1234567890123456L（L または l で終わる）
- **FLOAT**（倍精度）: 3.14e-10、.001、10.、1E3
- **複素数**: 1J、2+3J、4+5j（J または j で終わる。ゼロでない実数部を持つ複素数を作成するには、浮動小数点数および+を追加します）

文字列

次に示すシーケンスは、文字列として定義されます。

- '一重引用符で囲まれた文字列'
- "引用符で囲まれた別の文字列"
- '一重引用符で囲まれ 1 つの"（二重引用符）を含む文字列'
- "二重引用符で囲まれ 1 つの'（一重引用符）を含む文字列"
- '''改行および'を含む文字列は 3 つの一重引用符で囲むことが可能'''
- """ 三重引用符も使用可能"""
- r' 解釈されない文字列（¥はそのまま）。Windows のパスに使用すると便利。'
- R" 解釈されない文字列"

1 つの文字列に複数の行を使用する場合は、行の最後に¥を使用します。

隣接した 2 つの文字列は連結されます（たとえば、'Oracle Data Integrator and' ' Python'は'Data Integrator and Python'と同じです）。

エスケープ・シーケンス

¥newline : 無視（改行をエスケープ）

¥¥ :円記号（¥）

¥e: エスケープ（ESC）

¥v: 垂直タブ（VT）

¥' : 一重引用符（'）

¥f :改ページ（FF）

¥OOO : 8 進値 OOO の文字

¥" : 二重引用符（"）

¥n : 改行（LF）

¥a : ビープ（BEL）

¥r : 改行（CR）

¥xHH : 16 進文字 HH

¥b : バックスペース（BS）

¥t : 水平タブ（TAB）

¥uHHHH: 16 進 Unicode 文字 HHHH

¥AllCharacter:そのままの状態

文字列の書式化

文字列の書式化は非常に便利です。C 関数 sprintf() :によく似ています。

例:

`"My tailor is %s..." % "rich"`は`"My tailor is rich..."`を戻します。

`"Tea %d %d %s" % (4, 2, "etc.")`は`"Tea 4 2 etc."`を戻します。

`"%(itemNumber)d %(itemColor)s" % {"itemNumber":123, "itemColor":"blue"}`は`"123 blue"`を戻します。

文字列を書式化する%コード:

コード	説明
%s	文字列または任意のオブジェクト
%r	%s と同じだが repr()を使用
%c	文字
%d	10 進整数
%i	整数
%u	符号なし整数
%o	8 進整数
%x、%X	16 進整数
%e、%E	浮動小数指数
%f、%F	浮動小数
%g、%G	%e または%f 浮動小数
%%	'%'リテラル

文字列に最もよく使用されるメソッド

文字列に使用される最も一般的なメソッドを次の表に要約します。たとえば、s が文字列の場合、s.lower()は s を小文字で戻します。シーケンスのすべての演算は認可されます。

コード	説明
s.capitalize()	s のコピーを大文字で戻します。
s.center(width)	width 文字の文字列を中央揃えにした s のコピーを戻します。
s.count(sub[,start[,end]])	s での sub の発生回数を戻します。
s.encode([encoding[,errors]])	s のエンコードされたバージョンを戻します。
s.endswith(suffix[,start[,end]])	suffix で終わる場合は TRUE を戻します。
s.expandtabs([tabsize])	すべてのタブがタブと同サイズの空白に置換された s のコピーを戻します。
s.find(sub[,start[,end]])	sub が見つかった場合に s の最初のインデックスを戻します。
s.index(sub[,start[,end]])	find と同じですが sub が見つからなかった場合はエラーを戻します。
s.isalnum()	s のすべての文字が英数字の場合に TRUE を戻します。
s.isalpha()	s のすべての文字が英文字の場合に TRUE を戻します。
s.isdigit()	s のすべての文字が数字の場合に TRUE を戻します。
s.islower()	s が小文字の場合に TRUE を戻します。
s.isspace()	s に空白しか含まれない場合に TURE を戻します。
s.istitle()	s の各単語が大文字で始まる場合に TRUE を戻します。
s.isupper()	s のすべての文字が大文字の場合に TRUE を戻します。
s.join(seq)	s で区切られた、シーケンス seq の文字列の連結を戻します。
s.ljust(width)	width 文字の最大長を持つ、左揃えの s のコピーを戻します。
s.lower()	s の小文字のコピーを戻します。

`s.lstrip()`	左端のすべての空白を削除した s のコピーを戻します。
`s.replace(old, new[, maxsplit])`	s に含まれる old を new に置換します。
`s.rfind(sub[,start[,end]])`	sub が見つかった場合に s の最後のインデックスを戻します。
`s.rindex(sub[,start[,end]])`	rfind と同じですが見つからなかった場合はエラーを戻します。
`s.rjust(width)`	width 文字の最大長を持つ、右揃えの s のコピーを戻します。
`s.rstrip()`	右端のすべての空白を削除した s のコピーを戻します。
`s.split([sep[,maxsplit]])`	sep をセパレータとして使用して s の単語のリストを戻します。
`s.splitlines([keepends])`	s の行のリストを戻します。
`s.startswith(prefix[,start[,end]])`	prefix で始まる場合に TRUE を戻します。
`s.strip()`	両端のすべての空白を削除した s のコピーを戻します。
`s.swapcase()`	大文字を小文字、小文字を大文字に変換した s のコピーを戻します。
`s.title()`	すべての単語が大文字で始まる s のコピーを戻します。
`s.translate(table[,deletechars])`	table に従って変換します。
`s.upper()`	s の大文字のコピーを戻します。

リスト

リストは、インデックスを使用してアクセスできる、オブジェクトへの変更可能な参照の配列です。

リストは、カンマで区切られカッコで囲まれた一連の値です。

- `[]`は空のリストです。
- `[0, 1, 2, 3, 4, 5]`は、0～5 のインデックスが付いている 6 つの要素のリストです。
- `mylist = ['john', 1, ['albert', 'collin']]`は、インデックス 2（3 番目の要素）もまたリストである 3 つの要素のリストです。

mylist[2]は['albert', 'collin']を戻します。

mylist[2][1]は'collin'を戻します。

リスト関数の一部

メソッド	説明
mylist.append(x)	リストの最後に要素を追加します。
mylist.sort([function])	オプションの[function]比較関数を使用してリストをソートします。
mylist.reverse()	リストを（末尾から先頭へ）逆にします。
mylist.index(x)	インデックス x を検索します。
mylist.insert(i, x)	インデックス i の位置に x を挿入します。
mylist.count(x)	リスト内での x の発生回数を戻します。
mylist.remove(x)	リストで最初に発生した x を削除します。
mylist.pop([i])	リストの最後の要素またはインデックス i にある要素を削除して戻します。

ディクショナリ

ディクショナリは、インデックスではなくキー（文字列値）に基づいて索引付けされたオブジェクトの配列です。

ディクショナリへのアクセスには、カッコで囲んだカンマ区切りのタプル **key:value** を使用します。

- {}は空のディクショナリです。
- {'P1':'Watch', 'P2': 'Birds', 'P3':'Horses'}は、P1、P2 および P3 というキーを持つ 3 つの要素を含むディクショナリです。
- adict = {'FR_US':{'Bonjour':'Hello', 'Au revoir':'Goodbye'}, 'US_FR':{'Hello': 'Bonjour','Goodbye':'Au Revoir'}}は、他のディクショナリを含むディクショナリです。Hello をフランス語に翻訳する場合:
  adict['US_FR']['Hello']

ディクショナリを処理するメソッドの一部

メソッド	説明
adict.has_key(k)	キーk がディクショナリに存在する場合は、TRUE（1）を戻します。

adict.keys()	ディクショナリ・キーのリストを戻します。
adict.values()	ディクショナリ値のリストを戻します。
adict.items()	ディクショナリの要素ごとにタプル（キー、値）のリストを戻します。
adict.clear()	adict のすべての要素を削除します。
adict.copy()	adict のコピーを戻します。
dic1.update(dic2)	ディクショナリ dic1 を、キーの値に基づいて、dic2 の値で更新します。
adict.get(k[,default])	adict[k]と同じですが、k が見つからない場合は default を戻します。
adict.popitem()	要素を取得してディクショナリから削除します。

タプル

タプルは、インデックスを使用して解析される**変更できない**オブジェクト配列です。

タプルは、カンマで区切られカッコで囲まれた一連の値として処理されます。

- ()は空のタプルです。
- (0,1,2,3) は、0～3 のインデックスが付いている 4 要素のタプルです。
- tuple = (0, (1, 2), (4,5,6)) は、他のタプルを含むタプルです。tuple[1][0] は 1 を戻します。

タプルにはシーケンスの演算を使用できます。

シーケンス

シーケンスには、文字列、リスト、タプルまたはディクショナリを使用できます。

次の表に、最も一般的な演算を示します。

すべてのシーケンス

演算	説明
X in S、X not in S	含まれるかどうか
for X in S:	反復
S+S	連結
S*N、N*S	繰返し

S[i]	索引付け
S[i:j]	配列索引付け
len(S)	長さ（サイズ）
iter(S)	反復オブジェクト
min(S)	最小要素
max(S)	最大要素

変更可能なリスト

演算	説明
S[i]=X	割当て/インデックス i の変更
S[i:j]=S2	配列への S2 の割当て
del S[i]	要素 i の削除
del S[i:j]	配列の i から j までの削除

ディクショナリ

演算	説明
D[k]	キーの索引付け
D[k] = X	割当て/キーの変更
del D[k]	キーk での要素の削除
len(D)	D に含まれるキーの数

例

```
>>> s='ABCDEFGH'
>>>s[0]
'A'
>>>s[0:2]
'AB'
>>>s[:2]
'AB'
```

```
>>>s[-3:-1]
'FG'
>>>s[-1:]
'H'
```

ファイル

ファイル・オブジェクトは、ビルトイン関数を使用して処理されます。演算の読取りまたは書込みのためにファイルを開くには、open メソッドを使用します。

ファイルに使用される最も一般的なメソッドを次の表に示します。

演算	説明
f = open(filename [, mode='r'])	適切なモードでファイルを開きます。 モード: 'r': 読取り。 'w': 書込み。ファイルが存在しない場合は作成されます。 'a': 追加。 '+': ('a+'のように前のモードに追加されます) 更新のためにファイルを開きます。 'b': ('rb'のように前のモードに追加されます) バイナリ・モードでファイルを開きます。
f.close()	ファイルを閉じます。
f.fileno()	f ファイルの記述子を戻します。
f.flush()	f の内部バッファを空にします。
f.isatty()	f が TTY の場合に true を戻します。
f.read([size])	ファイルから最大 size バイトを読み取り、文字列を戻します。
f.readline()	f から 1 行を読み取ります。
f.readlines()	ファイルの終端 (EOF) まで読取り、行のリストを戻します。
f.xreadlines()	ファイルの終端まで読み取らずにシーケンスを戻します (readlines()より優先)。
f.seek(offset[, whence=0])	ファイルの現在の位置を whence からの offset バイト数に設定します。 0: ファイルの先頭から

	1: 現在の場所から 2: ファイルの終端から
`f.tell()`	ファイルの現在の位置を戻します。
`f.write(str)`	文字列をファイルに書き込みます。
`f.writelines(list)`	文字列のリストをファイルに書き込みます。

構文

識別子

識別子の名前は次のように付けられます。

```
(letter | "_")(letter | number | "_")*
```

注意: 識別子、キーワードおよび属性は、大文字と小文字が区別されます。

特別な書式:

`_ident` , `__ident__` and `__ident` は特に重要です。Jython のドキュメントを参照してください。

割当て

次に示す割当て書式は、すべて有効です。

```
x = v
x1 = x2 = v
x1, x2 = v1, v2
x1, x2, ..., xn = v1, v2, ..., vn
(x1, x2, ..., xn) = (v1, v2, ..., vn)
[x1, x2, ..., xn] = [v1, v2, ..., vn]
```

次に示す特別な割当て書式も有効です。

`x += y` は `x = x + y` と同じ

`x *= y` は `x = x * y` と同じ

`x /= y` は `x = x / y` と同じ

`x -= y` は `x = x - y` と同じ

`x %= y` は `x = x % y` と同じ

`x &= y` は `x = x & y` と同じ

x ^= y は x = x ^ y と同じ

x **= y は x = x ** y と同じ

x |= y は x = x | y と同じ

式

```
expression
function([value1, arg_name=value2, ...])
object.method([value1, arg_name=value2, ...])
```

ファンクション・コールのたびに、パラメータ・リストに示されているすべてのパラメータに、固定引数、キーワード引数またはデフォルト値から値が割り当てられます。

命令

break 文

break

内側にある最も近い while ループまたは for ループを終了します。ループにオプションの else 句が含まれる場合は、これをスキップします。

class 文

```
class class_name [(super_class1 [,super_class2]*)]:
 instructions
```

新しい class_name クラスを作成します。このクラスを後でインスタンス化すると、オブジェクトを作成できます。

例

```
class c:
  def __init__(self, name, pos):
    self.name = name
    self.pos = pos
  def showcol(self):
    print "Name : %s; Position :%d" % (self.name, self.pos)

col2 = c("CUSTNAME", "2")
col2.showcol()
```

戻り値:

```
Name : CUSTNAME, Position :2
```

continue 文

```
continue
```

内側にある最も近い while ループまたは for ループの次のサイクルに移行します。

def 文

```
def func_name ([arg, arg=value, ...*arg, **arg]):
  instructions
```

func_name 関数を定義します。

パラメータは値によって渡され、次のように定義されます。

パラメータ	説明
arg	値によって渡されるパラメータ
arg=value	デフォルト値を伴うパラメータ（コール時に arg が渡されなかった場合）
*arg	自由パラメータ。arg は全パラメータのタプルの値を取得します。
**arg	自由パラメータ。arg は各パラメータのディクショナリ値を取得します。

例

デフォルト・パラメータを使用した関数:

```
def my_function(x, y=3):
        print x+y

my_function(3,5)
```

8 が表示されます。

```
my_function(3)
```

6 が表示されます。

自由パラメータを使用した関数:

```
def my_print(*x):
     for s in x:
             print s,
```

```
my_print('a','b','c','d')
```
a b c d が表示されます。

del 文

```
del x
del x[i]
del x[i:j]
del x.attribute
```

名前、参照、スライシング、属性を削除します。

exec 文

```
exec x [in globals [,locals]]
```

指定された名前空間で x を実行します。デフォルトでは、現在の名前空間で x が実行されます。x に使用できるのは、文字列、ファイル・オブジェクトまたは関数オブジェクトです。

for 文

```
for x in sequence:
  instructions
[else:
  instructions]
```

シーケンスの要素に対して処理を繰り返す場合に使用します。*sequence* の各項目は、割当ての標準規則に従って *x* に割り当てられます。その後、コードが実行されます。項目がなくなると（つまり、シーケンスが空になったらすぐに）、else 句内のコードがある場合はそのコードが実行され、ループが終了します。

例

0～3 のループ:
```
for i in range(4):
  print i
```

2～5 のループ:
```
for i in range(2, 6):
  print i
```

2～10 の偶数のループ:
```
for i in range(2, 11, 2):
```

```
  print i
```

リストの全要素のループ:

```
l = [ 'a', 'b', 'c', 'd']
for x in l:
  print x
```

from 文

```
from module import name1 [as othername1] [, name2]*
from module import *
```

モジュールの名前を現在の名前空間にインポートします。

例

c:¥のディレクトリを表示します。

```
from os import listdir as directorylist
dir_list = directorylist('c:/')
print dir_list
```

global 文

```
global name1 [, name2]
```

global 文は、現在のコード・ブロック全体で保持される宣言です。つまり、リストされた識別子はグローバル識別子として解釈されます。global を使用せずにグローバル変数に割当てを行うことは不可能ですが、自由変数を使用すると、グローバルと宣言せずにグローバル識別子を参照できます。*name1* は、グローバル変数 name1 への参照です。

if 文

```
if condition:
  instructions
[elif condition:
  instructions]*
[else:
  instructions]
```

If、else if ...、else の条件付き実行に使用します。

例

```
x = 2
y = 4
if x == y :
  print "Waooo"
elif x*2 == y:
  print "Ok"
else:
  print "???"
```

import 文

```
import module1 [as name1] [, module2]*
```

アクセスできるようにするために、モジュールまたはパッケージをインポートします。モジュールには、その module_name.name 修飾子を介してアクセス可能な名前が含まれます。

例

c:¥内のディレクトリを表示します。

```
import os
dir_list = os.listdir('c:/')
print dir_list
```

JDBC クラスを使用して SQL 問合せを実行し、結果を表示します。

```
import java.sql as jsql
import java.lang as lang
driver, url, user, passwd = (
     "oracle.jdbc.driver.OracleDriver",
     "jdbc:oracle:thin:@pluton:1521:pluton",
     "user",
     "pass")
lang.Class.forName(driver)
c = jsql.DriverManager.getConnection(url,user,passwd)
s = c.createStatement()
sql_stmt = "select * from user_tables"
print "executing " , sql_stmt
rs = s.executeQuery(sql_stmt)
while (rs.next()):
        print rs.getString("TABLE_NAME"), rs.getString("OWNER")
c.close()
```

pass 文

```
pass
```

これはヌル操作です。実行しても何も起こりません。構文的に文が要求されるがコードを実行する必要がない場合のプレースホルダとして役立ちます。

print 文

```
print [value [, value]* [,]]
print >> file_object [, value [, value]* [,]]
```

順番にそれぞれの式を評価し、生成されたオブジェクトを標準出力（stdout）または file_object ファイルに書き込みます。print 文がカンマで終わる場合を除いて、最後に¥n 文字が記述されます。

raise 文

```
raise [exception [, value]]:
```

オプションの value 値を使用して exception 例外を生成します。

exception オブジェクトがクラスの場合、そのオブジェクトは例外の種類になります。 value は、例外値の決定に使用されます。これがクラスのインスタンスの場合は、そのインスタンスが例外値になります。value がタプルの場合は、クラス・コンストラクタの引数リストとして使用されます。None の場合は、空の引数リストが使用され、その他のすべてのオブジェクトはコンストラクタの単一の引数として処理されます。そのため、コンストラクタをコールして作成されたインスタンスは、例外値として使用されます。

例外がない場合は、現在のスコープで最後にアクティブだった例外が raise によって再生成されます。現在のスコープでアクティブだった例外が 1 つもない場合は、エラーであることを示す例外が生成されます。

return 命令

```
return [expression]
```

expression（もしくは None）を戻り値として、現在のファンクション・コールを終了します。

try 文

```
try:
 suite1
[except [exception [, value]]:
 suite2]*
[else :
```

```
 suite3]

try:
 suite1
finally:
 suite2
```

複数の文に対する例外ハンドラまたはクリーンアップ・コード（もしくはその両方）を指定します。try 文の書式は 2 通りあります。

1 つ目の書式: suite1 で例外が発生しなかった場合、例外ハンドラは実行されません。suite1 で例外が発生した場合、exception の検索が開始されます。例外が検出されると、suite2 が実行されます。検出されなかった場合は、suite3 が実行されます。例外が値を持つ場合は、トリガーされたインスタンスに割り当てられます。

2 つ目の書式: suite1 で例外が発生しなかった場合、例外ハンドラは実行されません。suite1 で例外が発生した場合、すべてのケースで suite2 が実行され、例外が生成されます。

例

すべてのケースでファイルを開いて閉じます。

```
f = open('c:/my_file.txt', 'w')
try:
  f.write('Hello world')
  ...
finally:
  f.close()
```

既存のファイルを開いて例外をトラップします。

```
try:
  f = open('inexisting_file.txt', 'r')
  f.close()
except IOError, v:
  print 'IO Error detected: ', v
else:
  print 'Other Error'
```

while 文

```
while condition:
  instructions
[else:
```

```
  instructions]
```

式が true であるかぎり繰り返される式に使用します。最初のスイートで break 文が実行されると、*condition* が false になる前に、else 句のスイートを実行せずにループが終了します。

例

0～8 の i を表示します。

```
i = 0
while i < 9:
  print i
  i+=1
```

モジュール

内部モジュールは import 文を使用してインポートする必要があります。モジュールは多数あります。

モジュールの完全なリストおよび参照ドキュメントは、http://www.jython.org から入手できます。

次の表に、最もよく使用されるモジュールの概要を示します。

モジュール	説明	メソッドの一部
None	標準関数	apply、callable、chr、cmp、coerce、compile、complex、delattr、eval、execfile、filter、getattr、globals、hasattr、hash、hex、id、input、intern、isinstance、issubclass、list locals、ord、pow、range、reduce、reload、repr、round、setattr、slice、str、tuple、type、vars、xrange
sys	インタプリタに対して相対的な関数が含まれます。環境コンポーネント（コマンドライン、標準 I/O）へのアクセスを可能にします。	argv、builtin_module_names、check_interval、exc_info、exit、exitfunc、getrecursionlimit、getrefcount、maxint、modules、path、platform、setcheckinterval、setdefaultencoding、setprofile、setrecursionlimit、settrace、stderr、stdin、stdout、version
os	プラットフォームとは独立して、オペレーティング・システムと連動します。	_exit、altsep、chdir、chmod、curdir、environ、error、execv、fork、getcwd、getpid、kill、linesep、listdir、makedirs、name、pardir、path、pathsep、pipe、removedirs、renames、sep、system、times、wait、waitpid
math	数学関数	acos、asin、atan、atan2、ceil、cos、cosh、e、exp、fabs、floor、fmod、frexp、ldexp、log、

		log10、modf、pi、pow、sin、sinh、sqrt、tan、tanh
time	日付/時刻関数	altzone、daylight、gmtime、localtime、sleep、strftime、time
re	正規表現関数	compile、I、L、M、S、U、X、match、search、split、sub、subn、findall、escape
socket	TCP/IP ソケットのサポート	参照ドキュメントを参照してください。
cgi	CGI スクリプトのサポート	参照ドキュメントを参照してください。
urllib、urllib2	Web ページ検索。http、ftp、gopher および file の URL をサポートします。	参照ドキュメントを参照してください。
ftplib	FTP プロトコルのサポート	参照ドキュメントを参照してください。
httplib、nntplib	http および nntp のサポート	参照ドキュメントを参照してください。
poplib、imaplib、smtplib	POP、IMAP および SMTP のプロトコルのサポート	参照ドキュメントを参照してください。
telnetlib、gopherlib	telnet、gopher のサポート	参照ドキュメントを参照してください。

Oracle Data Integrator での Jython の使用

Jython インタプリタの使用方法

Jython プログラムは、標準の Jython インタプリタを使用して Data Integrator 外部でのテスト用に解釈できます。

Jython インタプリタの起動

1. OS プロンプト（コンソール）を起動します。
2. /bin ディレクトリに移動します。
3. jython とキー入力します。
4. インタプリタが起動します。

Jython インタプリタの終了

1. [Ctrl]を押しながら[Z]（^Z）を押し、その後[Enter]を押します。
2. インタプリタが終了します。

Jython スクリプトの実行

1. /bin ディレクトリに移動します。
2. jython <script_path.py>と入力します。
3. スクリプトが実行されます。

プロシージャでの Jython の使用方法

すべての Jython プログラムは、**プロシージャ**または**ナレッジ・モジュール**からコールできます。

Jython をコールするプロシージャの作成

1. **「デザイナ」**で独自の**プロジェクト**に含まれている**フォルダ**を選択し、新しい**プロシージャ**を挿入します。
2. プロシージャの名前を**「名前」**に入力します。
3. **「詳細」**タブにコマンドラインを追加します。
4. コマンド・ウィンドウの**「名前」**に、このコマンドの名前を入力します。
5. **「ターゲットに対するコマンド」**タブのテクノロジのリストから Jython を選択します。
6. コマンド・テキストとして**「コマンド」**に実行する Jython プログラムを入力するか、式エディタを使用します。
7. **「OK」**をクリックして変更を適用します。
8. **「適用」**をクリックして、プロシージャ・ウィンドウでの変更を適用します。
9. **「実行」**タブで**「実行」**ボタンをクリックして、実行ログにある実行結果に従います。

この手順で作成したプロシージャは、他のプロシージャと同様に**「パッケージ」**に追加できます。

セッションでの Jython 変数の永続性

使用される Jython のすべての変数は、実行セッション内で永続します。

たとえば、**プロシージャ** TRT1 に 3 つの**コマンドライン**があり、次のように定義されているとします。

ライン 1 : `Set value for x`

```
x = 'My Taylor is Rich'
```

ライン 2 : `Set value for y`

```
y = 'I wish I could say :'
```

ライン 3 : `Write output file`

```
f = open('test.txt', 'w')
f.write('Result : %s %s' % (y, x))
f.close()
```

`TRT1` の実行後に生成される `test.txt` ファイルの内容は、`Result : I wish I could say : My Taylor is Rich` になります。

Jython 変数 `x` および `y` は、同じプロシージャ内で複数のコマンドラインに渡ってそれぞれの値を保持しています。

同様に、パッケージ内で `TRT1` の後に実行される `TRT2` プロセスでも、同じ実行セッション内で `x` および `y` を使用できます。

標準のディストリビューションへの特定のモジュールの追加

デフォルトのモジュールに新しいモジュールを追加することで、Jython の基本的な関数を拡張できます。

独自の Jython モジュールを記述して（http://www.jython.org から入手できるドキュメントを参照してください）、このモジュールを Oracle Data Integrator のインストール・ディレクトリの `/lib/scripting/Lib` サブディレクトリに置くことができます。

Oracle Data Integrator の追加のモジュール

Oracle Data Integrator で Jython をより簡単に使用できるように、次のモジュールが追加されています。

snpsftp モジュール

このモジュールを使用すると、Jython で簡単に FTP（ファイル転送プロトコル）を使用できます。

このモジュールは、クラス SnpsFTP を実装します。

SnpsFTP クラス

コンストラクタ/メソッド	説明
`SnpsFTP([host [,user [,passwd[, acct[, port]]]]])`	コンストラクタ: ftp オブジェクトを作成し、`user`、`passwd` および `acct` を認証のために使用して、ポート番号 `port` の `host` FTP サーバーに接続します。
`connect(host [,port])`	ポート番号 `port` の FTP ホスト・サーバーに接続します。

login([user [,passwd [,acct]]])	FTP サーバーに対して認証を実行します。
setmode(mode)	モードを ASCII または BINARY に設定します。使用できる値は'ASCII'または'BINARY'です。転送のデフォルト値は ASCII です。
setpassive(0 | 1)	パッシブ（1）またはアクティブ（0）モードで FTP 接続を設定します。
get(src[, dest [, mode]])	フルパスで表されたファイル src（FTP サーバー上）をファイルまたはディレクトリ dest にダウンロードします。mode は'ASCII'または'BINARY'に強制されます。
mget(srcdir, pattern [, destdir [, mode]])	mode モードを使用して、フィルタ pattern と一致するディレクトリ srcdir の一連のファイルを、ディレクトリ destdir にダウンロードします。
put(src [, dest [, mode='' [, blocksize=8192]]])	mode モードを使用して、ローカル・ファイル src をサーバー・ファイル dest に置きます。blocksize バイトのブロック転送サイズを使用します。
mput(srcdir, pattern [, destdir [, mode [, blocksize=8192]]])	mode モードを使用して、フィルタ pattern と一致するディレクトリ srcdir の複数のローカル・ファイルを、サーバー・ディレクトリ destdir に置きます。blocksize オクテットの転送ブロック・サイズを使用します。
quit()	QUIT コマンドを送信してから FTP サーバーとの接続を閉じます。
close()	FTP サーバーとの接続を閉じます。

例

サーバーftp.myserver.com の/home/odi から*.txt ファイルを取得し、ローカル・ディレクトリ c:¥temp に置きます。

```
import snpsftp
ftp = snpsftp.SnpsFTP('ftp.myserver.com', 'mylogin', 'mypasswd')
ftp.setmode('ASCII')
ftp.mget('/home/odi', '*.txt', 'c:/temp')
ftp.close()
```

C:¥odi¥lib の*.zip ファイルを、リモート・ディレクトリ/home/odi/lib のサーバーftp.myserver.com に置きます。

```
import snpsftp
```

```
ftp = snpsftp.SnpsFTP('ftp.myserver.com', 'mylogin', 'mypasswd')
ftp.setmode('BINARY')
ftp.mput('C:/odi/lib', '*.zip', '/home/odi/lib')
ftp.close()
```

例

ファイルの読取りおよび書込み

SRC_AGE_GROUP.txt ファイルには、列を;で区切ったレコードが含まれています。次の例では、タブをセパレータに使用して、SRC_AGE_GROUP.txt ファイルが新しいファイル SRC_AGE_GROUP_NEW.txt に変換されます。

この例では、split()文字列メソッドを使用して;で区切られたフィールドのリストを指定し、join() を使用してタブ（'¥t'）で区切られた新しい文字列を再構築します。

```
fsrc = open('c:/odi/demo/file/SRC_AGE_GROUP.txt', 'r')
ftrg = open('c:/odi/demo/file/SRC_AGE_GROUP_NEW.txt', 'w')
try:
  for lsrc in fsrc.readlines():
    # get the list of values separated by ;
    valueList = lsrc.split(';')
    # transform this list of values to a string separated by a tab
('¥t')
    ltrg = '¥t'.join(valueList)
    # write the new string to the target file
    ftrg.write(ltrg)
finally:
  fsrc.close()
  ftrg.close()
```

上の例のメソッド readlines()は、ファイル全体をメモリーにロードします。そのため、小さいファイルのみに使用する必要があります。ファイルが大きい場合は、次の例のように readline()メソッドを使用します。readline()は 1 行ずつ読取りを実行します。

```
fsrc = open('c:/odi/demo/file/SRC_AGE_GROUP.txt', 'r')
ftrg = open('c:/odi/demo/file/SRC_AGE_GROUP_NEW.txt', 'w')
try:
  lsrc=fsrc.readline()
  while (lsrc):
    valueList = lsrc.split(';')
    ltrg = '¥t'.join(valueList)
```

```
    ftrg.write(ltrg)
    lsrc=fsrc.readline()
finally:
  fsrc.close()
  ftrg.close()
```

ディレクトリの内容のリスト

次に示す例では、ディレクトリ c:/odi の内容がリストされ、このリストが c:/temp/listdir.txt に書き込まれます。メソッド os.path.isdir()により、リストの各要素がファイルかディレクトリかがチェックされます。

```
import os
ftrg = open('c:/temp/listdir.txt', 'w')
try:
    mydir = 'c:/odi'
    mylist = os.listdir(mydir)
    mylist.sort()
    for dirOrFile in mylist:
        if os.path.isdir(mydir + os.sep + dirOrFile):
            print >> ftrg, 'DIRECTORY: %s' % dirOrFile
        else:
            print >> ftrg, 'FILE: %s' % dirOrFile
finally:
    ftrg.close()
```

オペレーティング・システムの環境変数の使用方法

オペレーティング・システムの環境変数を取得すると役立ちます。次の例は、このリストの取得方法を示しています。

```
import os
ftrg = open('c:/temp/listenv.txt', 'w')
try:
  envDict = os.environ
  osCurrentDirectory = os.getcwd()
  print >> ftrg, 'Current Directory: %s'% osCurrentDirectory
  print >> ftrg, '=============================='
  print >> ftrg, 'List of environment variables:'
  print >> ftrg, '=============================='
  for aKey in envDict.keys():
    print >> ftrg, '%s¥t= %s' % (aKey, envDict[aKey])
  print >> ftrg, '=============================='
```

```
  print >> ftrg, 'Oracle Data Integrator specific environment
variables:'
  print >> ftrg, '=============================='
  for aKey in envDict.keys():
    if aKey.startswith('SNP_'):
      print >> ftrg, '%s¥t= %s' % (aKey, envDict[aKey])
finally:
  ftrg.close()
```

USERNAME 環境変数の値を取得する場合は、単純に次のように記述します。

```
import os
currentUser = os.environ['USERNAME']
```

JDBC の使用方法

JDBC（Java DataBase Connectivity）を使用して Jython からデータベースに接続すると便利です。CLASSPATH 内のすべての Java クラスは、Jython で直接使用できます。次の例は、JDBC API を使用してデータベースに接続し、SQL 問合せを実行して結果をファイルに書き込む方法を示しています。

Java の参照ドキュメントは、http://java.sun.com から入手できます。

```
import java.sql as sql
import java.lang as lang
def main():
  driver, url, user, passwd = (
    'oracle.jdbc.driver.OracleDriver',
    'jdbc:oracle:thin:@myserver:1521:mysid',
    'myuser',
    'mypasswd')
  ##### Register Driver
  lang.Class.forName(driver)

  ##### Create a Connection Object
  myCon = sql.DriverManager.getConnection(url, user, passwd)
  f = open('c:/temp/jdbc_res.txt', 'w')
  try:
    ##### Create a Statement
    myStmt = myCon.createStatement()
    ##### Run a Select Query and get a Result Set
    myRs = myStmt.executeQuery("select TABLE_NAME, OWNER from ALL_TABLES
where TABLE_NAME like 'SNP%'")

    ##### Loop over the Result Set and print the result in a file
```

```
    while (myRs.next()):
      print >> f , "%s¥t%s" %(myRs.getString("TABLE_NAME"),
myRs.getString("OWNER") )
  finally:
    myCon.close()
    f.close()

### Entry Point of the program
if __name__ == '__main__':
  main()
```

より柔軟に処理を行うために、Oracle Data Integrator のプロシージャで Jython と odiRef API を組み合せることができます。データベースに接続するためにプログラムでパラメータをハードコードするかわりに、getInfo メソッドを使用できます。

```
import java.sql as sql
import java.lang as lang
def main():
  driver, url, user, passwd = (
    '<%=odiRef.getInfo("DEST_JAVA_DRIVER")%>',
    '<%=odiRef.getInfo("DEST_JAVA_URL")%>',
    '<%=odiRef.getInfo("DEST_USER_NAME")%>',
    '<%=odiRef.getInfo("DEST_PASS")%>')
  ##### Register Driver
  lang.Class.forName(driver)
[...]
```

FTP の使用方法

環境によっては、FTP（ファイル転送プロトコル）を使用して異機種間システムでファイルを転送すると便利です。Oracle Data Integrator には、FTP をより詳細に統合するための追加の Jython モジュールが同梱されています。

次の例は、このモジュールの使用方法を示しています。

サーバーftp.myserver.com の/home/odi からローカル・ディレクトリ c:¥temp へ、*.txt ファイルをプルします。

```
import snpsftp
ftp = snpsftp.SnpsFTP('ftp.myserver.com', 'mylogin', 'mypasswd')
try:
  ftp.setmode('ASCII')
  ftp.mget('/home/odi', '*.txt', 'c:/temp')
finally:
  ftp.close()
```

C:¥odi¥lib からリモート・ディレクトリ/home/odi/lib の ftp.myserver.com へ、*.zip ファイルをプッシュします。

```
import snpsftp
ftp = snpsftp.SnpsFTP('ftp.myserver.com', 'mylogin', 'mypasswd')
try:
  ftp.setmode('BINARY')
  ftp.mput('C:/odi/lib', '*.zip', '/home/odi/lib')
finally:
  ftp.close()
```

IP ソケットの使用方法

IP ソケットは、ネットワーク上の 2 つのプロセス間の IP 通信を開始するために使用されます。Jython を使用すると、IP サーバー（IP パケットの待機）または IP クライアント（IP パケットの送信）の作成が大幅に単純化されます。

次の例は、ごく基本的な IP サーバーの実装を示しています。ここでは、クライアント・ソフトウェアからのデータを待機し、受け取った各パケットをファイル c:/temp/socketserver.log に書き込みます。サーバーがパケット STOPSERVER を受け取ると、サーバーは停止します。

サーバー

```
import socket
import time
HOST = ''
PORT = 9191 # Arbitrary port (not recommended)
LOG_FILE = 'c:/temp/sockserver.log'
mySock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
mySock.bind((HOST, PORT))
logfile = open(LOG_FILE, 'w')
try:
  print >> logfile, '*** Server started : %s' % time.strftime('%Y-%m-
%d %H:%M:%S')
  while 1:
    data, addr = mySock.recvfrom(1024)
    print >> logfile, '%s (%s): %s' % (time.strftime('%Y-%m-
%d %H:%M:%S'), addr, data)
    if data == 'STOPSERVER':
      print >> logfile, '*** Server shutdown at %s by %s' %
(time.strftime('%Y-%m-%d %H:%M:%S'), addr)
      break
```

```
finally:
  logfile.close()
```

クライアント

次の例は、前述のサーバーをテストするために使用できます。この例では、サーバーの停止を指示する前に、2 つのパケットが送信されます。

```
import socket
import sys
PORT = 9191 # Same port as the server
HOST = 'SERVER_IP_ADDRESS'
mySock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
mySock.sendto('Hello World !', (HOST, PORT))
mySock.sendto('Do U hear me?', (HOST, PORT))
mySock.sendto('STOPSERVER', (HOST, PORT))
```