

Embedded Python

IRISで**Python**を使ってみよう！



この資料の主な目的

この資料では、Embedded Pythonを使用して、IRISからPythonモジュール／スクリプトファイルを使用する方法、Pythonでメソッドを記述する方法を習得し、Pythonの豊富なライブラリをIRIS内で自由に組み合わせて利用できることをご理解いただきます。

具体的には、以下の内容を学習します。

- IRISターミナルからPythonシェルを起動して、Pythonの操作体験
- ObjectScriptからPythonのスクリプトファイル（含クラス）をインポートして使う方法
- Pythonの組み込み関数（builtins）をObjectScriptで操作する方法の確認
- メソッドの記述
 - ObjectScript／Pythonメソッドの記述と実行練習
 - 可変長引数について
 - PythonのTrue／False／NoneをObjectScriptで扱う方法
- Python側でエラーが発生した場合の対応方法

Pythonシェルの起動方法

IRISにログインし、Pythonのシェルを起動する方法は以下の通りです。

- IRISのログイン
 - Windowsはターミナルを起動します。
 - Windows以外は、**iris session インスタンス名** でログインします。
インスタンス名が IRIS の場合は、`iris session iris` でログインできます。

- IRISログイン後のPythonシェル起動方法

```
do ##class(%SYS.Python).Shell()
```

- `do` は戻り値を持たないプロシージャ、メソッドを実行するときに使用する ObjectScript のコマンドです。
- `##class()` は ObjectScript でクラスを操作する際に使用する構文で、カッコ内にクラス名を記入します。以降はメソッドを指定します（指定のクラス名、メソッド名は大小文字を区別します）。

Pythonを使ってみよう：datetime

datetimeモジュールをインポートしていろいろ試してみましょう。

- `datetime.now()`：本日の日付、現在時刻

```
>>> import datetime
>>> datetime.datetime.now()
datetime.datetime(2022, 5, 6, 17, 19, 39, 214916)
```

- `date`オブジェクトのコンストラクタを利用して`date`オブジェクト生成

```
>>> dt=datetime.date(2021,4,12)
>>> dt.month
4
```

- お正月まであと何日？

```
>>> oshogatu=datetime.date(2023,1,1)
>>> today=datetime.date.today()
>>> td=oshogatu-today
>>> td.days
240
```

- 10日前は何日？

```
>>> td_10d=datetime.timedelta(days=10)
>>> today-td_10d
datetime.date(2022, 4, 26)
```

- 本日の経過日数は？

```
>>> today.toordinal()
738281
```

Pythonの経過日数とIRISの経過日数は
起源日が異なります！

Pythonを使ってみよう：datetime

経過日数を確認する

datetimeモジュールを利用して、IRISの内部日付との違いを確認してみましょう。

- **toordinal()**メソッドを利用すると、西暦1年1月1日からの経過日数を得られます。
- IRISの内部日付 (\$horolog) の起源日は1840年12月31日からの経過日数のため、Pythonのdatetimeを利用する場合、現在日付からIRISの起源日 (1840年12月31日) 引いた日数がIRISの現在日付となります。

```
>>> import datetime
>>> today=datetime.datetime.now()
>>> today
datetime.datetime(2022, 5, 6, 17, 19, 39, 214916)
>>> today.toordinal()
738281
>>> iris0day=datetime.date(1840,12,31)
>>> iris0day.toordinal()
672046
>>> today.toordinal()-iris0day.toordinal()
66235
```

```
<IRISターミナル：2022年5月6日の内部日付>
USER>write $ZDATEH(20220506,8)
66235
```

おまけ：IRISの内部日付の変換に便利なメソッド

データタイプ：%Dateが指定されているIRISのプロパティや\$horologで登録しているデータに対してPythonからアクセスする場合、以下メソッドを利用すると便利です。

#表示形式から内部形式

```
dob=iris.system.SQL.TODATE("1999-12-31","YYYY-MM-DD")
```

#内部形式から表示形式

```
irisdob=iris.system.SQL.TOCHAR(58073,"YYYY-MM-DD")
```

※**iris**モジュールはPythonからIRISのクラス、SQL、グローバルを操作する場合に使用するモジュールです。

Pythonのモジュールを使ってスクレイピング 準備

(開発者コミュニティの記事のタイトルを入手してみる)

requestsとBeautifulSoupモジュールをインストールします。

Window以外のOSでは、 **pip3**コマンドを使用してインストールしてください。

Windowsでは、 **irispip**コマンドを使用してください。

- irispipコマンドは、<インストールディレクトリ>%bin 以下にあります。
- Pythonパッケージをインストールするときのターゲットに
<インストールディレクトリ>%mgr%python を指定します。

<Windowsのコマンドプロンプトでの実行例>

```
c:%InterSystems%IRIS%bin>irispip install --target c:%intersystems%iris%mgr%python requests
c:%InterSystems%IRIS%bin>irispip install --target c:%intersystems%iris%mgr%python beautifulsoup4
```


Pythonのモジュールを使ってスクレイピング 実行

(開発者コミュニティの記事のタイトルを入手してみる)

Pythonシェルで以下実行します。

```
USER>do ##class(%SYS.Python).Shell()
```

```
Python 3.9.10 (tags/v3.9.10:f2f3f53, Jan 17 2022, 15:14:21) [MSC v.1929 64 bit (AMD64)] on win32  
Type quit() or Ctrl-D to exit this shell.
```

```
>>>  
>>> import requests  
>>> from bs4 import BeautifulSoup  
>>> response=requests.get('https://jp.community.intersystems.com/node/517211')  
>>> soup=BeautifulSoup(response.text,'html.parser')  
>>> title=soup.find('title').get_text()  
>>> title  
'InterSystems グランプリ・プログラミングコンテスト 2022 開催！ | InterSystems Developer Community'  
>>>
```

requests.get()の引数には任意のURL
を指定してみてください

PythonスクリプトファイルをIRISから呼び出す(1)

前頁のスクレイピングのコードをPythonスクリプトファイルに記述し、Pythonシェルからテスト実行します。

```
import requests                                     scraping.py
from bs4 import BeautifulSoup

def gettitle(url):
    response=requests.get(url)
    soup=BeautifulSoup(response.text,'html.parser')
    title=soup.find('title').get_text()
    return title
```

Pythonスクリプトファイルを
配置したパスを追加しています。

Pythonシェルでの実行

```
>>> import sys
>>> sys.path+=['c:¥workspace¥TryIRIS']
>>> import scraping
>>> scraping.gettitle('https://jp.community.intersystems.com/node/517211')
'InterSystems グランプリ・プログラミングコンテスト 2022 開催！ | InterSystems Developer Community'
>>>
```

PythonスクリプトファイルをIRISから呼び出す(2)

前頁の内容をIRISのターミナルから実行する方法を体験します。

(C:¥Workspace¥TryIRIS¥scraping.pyにファイルがあるとして)

USER>set sys=##class(%SYS.Python).Import("sys")

1. Pythonスクリプトファイルの配置されたパスをsysモジュールに追加するため、sysモジュールをインポートします。

USER>do sys.path.append("c:¥Workspace¥TryIRIS")

2. パスを追加します。

USER>set scraping=##class(%SYS.Python).Import("scraping")

3. scraping.pyをインポートします。

USER>write scraping.gettitle("https://jp.community.intersystems.com/node/517211")

InterSystems グランプリ・プログラミングコンテスト 2022 開催！ | InterSystems Developer Community

USER>

4. gettitle()関数を実行します。

※sys.pathへ追加するパスが固定の場合、管理ポータルで設定することもできます。

管理ポータル > システム管理 > 構成 > 追加設定 > メモリ詳細 > **PythonPath**

PythonスクリプトファイルをIRISから呼び出す(3)

(戻り値がインスタンスの場合)

Historyクラスのインスタンスが戻り値で戻ってきた場合もIRIS内のインスタンスとして利用できます。

```
def gettitle2(url):  
    response=requests.get(url)  
    soup=BeautifulSoup(response.text, 'html.parser')  
    title=soup.find('title').get_text()  
    return History(url,title)
```

```
class History:  
    url=None  
    title=None  
  
    def __init__(self,url,title):  
        self.url=url  
        self.title=title
```

```
def print(self):  
    print(f"URL:{self.url} のタイトルは {self.title}です")
```

```
USER>set o=scraping.gettitle2("https://jp.community.intersystems.com/node/517211")  
  
USER>write o.url  
https://jp.community.intersystems.com/node/517211  
USER>write o.title  
InterSystems グランプリ・プログラミングコンテスト 2022 開催！ | InterSystems Developer  
Community  
USER>do o.print()  
URL:https://jp.community.intersystems.com/node/517211 のタイトルは InterSystemsグラ  
ンプリ・プログラミングコンテスト 2022 開催！ | InterSystems Developer Communityです
```

ここまでの流れで確認できたこと

- IRISのターミナルからPythonシェルに切り替える方法を確認できました。
- Pythonの豊富なモジュールを利用する方法を確認できました。
- PythonスクリプトファイルをObjectScriptから操作する方法を確認できました。

ご参考： irispythonコマンドでPythonシェルを起動する方法

- IRISのインストールで用意されるirispythonコマンドを使用してもPythonシェルにログインできます。
- irispthonコマンドでログインしたシェルからは、IRISのUSERネームスペースにアクセスできます。
- irispthonコマンドは、[<インストールディレクトリ>/bin](#) 以下にあります。

例)

```
c:¥>cd intersystems¥iris¥bin
```

```
c:¥InterSystems¥IRIS¥bin>irispython
```

```
Python 3.9.5 (default, Mar 11 2022, 10:30:25) [MSC v.1927 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>>
```

```
>>> import datetime
```

```
>>> dt=datetime.datetime(2022,4,25,12,30,15,1500)
```

```
>>> print(dt)
```

```
2022-04-25 12:30:15.001500
```

ここからは・・・

Pythonの組み込み関数（builtins）をObjectScriptでどのように操作できるか確認します。

- Pythonの関数やメソッドの引数にPythonのリストや辞書などのコレクションを渡す必要がある場合、また戻り値で返される場合（またはその逆も含めて）ObjectScript側でどのような操作が必要になるかを確認します。

Pythonの組み込み関数 (builtins) を使う

Pythonですぐに使える関数=builtin関数をObjectScriptで使用するには、%SYS.Pythonクラスを使用してbuiltinsをインポートします。

```
set builtins=##class(%SYS.Python).Import("builtins")
```

builtinsインポート時に生成された
インスタンスを使用して、print()や
len()、list()、tuple()など利用できます

```
USER>set builtins=##class(%SYS.Python).Import("builtins")
```

```
USER>do builtins.print("Hello World!")  
Hello World!
```

```
USER>set list=builtins.list()
```

```
USER>do list.append("これは1つ目")
```

```
USER>do list.append("これは2つ目")
```

```
USER>do builtins.print(list)  
['これは1つ目', 'これは2つ目']
```

```
USER>write builtins.len(list)  
2
```


builtins関数 list() – ObjectScriptでの操作

ObjectScriptでbuiltins関数を利用するために、builtinsをインポートします。

```
set builtins=##class(%SYS.Python).Import("builtins")
```

- リストを作成する場合は、list()を利用します。

```
set list=builtins.list()
```

- リストに要素を追加する場合はappend()を利用します。

```
do list.append("あ")  
do list.append("い")
```

- リストの指定番号に要素を追加する場合はinsert(n,要素)を利用します。

```
do list.insert(2,"う")
```

- リストの指定番号の要素を取得する場合は__getitem__(n)を利用します。

```
write list.__getitem__(1)
```

- リストの指定番号の要素を削除する場合は、pop(n)を利用します。

```
do list.pop(1)
```

- リストの全要素を削除するには、clear()を利用します。

```
do list.clear()
```

```
USER>set list=builtins.list()
```

```
USER>do list.append("あ")
```

```
USER>do list.append("い")
```

```
USER>do list.insert(2,"う")
```

```
USER>do builtins.print(list)  
['あ', 'い', 'う']
```

```
USER>write list.__getitem__(1)  
い
```

```
USER>do list.pop(1)
```

```
USER>do builtins.print(list)  
['あ', 'う']
```

```
USER>do list.clear()
```

```
USER>do builtins.print(list)  
[]
```

builtins関数 dict() – ObjectScriptでの操作

ObjectScriptでbuiltins関数を利用するために、builtinsをインポートします。

```
set builtins=##class(%SYS.Python).Import("builtins")
```

- dictionary作成する場合は、dict()を利用します。

```
set dict=builtins.dict()
```

- dictionaryにキーと値を追加する場合はsetdefault(キー,値)を利用します。

```
do dict.setdefault("jp","Japanese")  
do dict.setdefault("en","English")
```

- dictionaryのキーを指定して値を取得する場合はget(キー)を利用します。

```
write dict.get("jp")
```

- dictionaryのキーを指定して値を削除する場合は、pop(キー)を利用します。

```
do dict.pop("jp")
```

- dictionaryの全情報を削除するには、clear()を利用します。

```
do dict.clear()
```

```
USER>set dict=builtins.dict()  
  
USER>do dict.setdefault("jp","Japanese")  
  
USER>do dict.setdefault("en","English")  
  
USER>do dict.setdefault("fr","Franch")  
  
USER>do builtins.print(dict)  
{'jp': 'Japanese', 'en': 'English', 'fr': 'Franch'}  
  
USER>write dict.get("en")  
English  
USER>do dict.pop("en")  
  
USER>do builtins.print(dict)  
{'jp': 'Japanese', 'fr': 'Franch'}  
  
USER>do dict.clear()  
  
USER>do builtins.print(dict)  
{}
```

リストの要素数を調べる

builtin関数のlen()を利用します。

```
USER>set builtins=##class(%SYS.Python).Import("builtins")
```

```
USER>set list=builtins.list()
```

```
USER>do list.append("あいうえお")
```

```
USER>do list.append("かきくけこ")
```

```
USER>write builtins.len(list)
```

```
2
```

```
USER>for i=0:1:(builtins.len(list)-1) { write list."__getitem__"(i),!}
```

```
あいうえお
```

```
かきくけこ
```

```
USER>
```

ここからは・・・

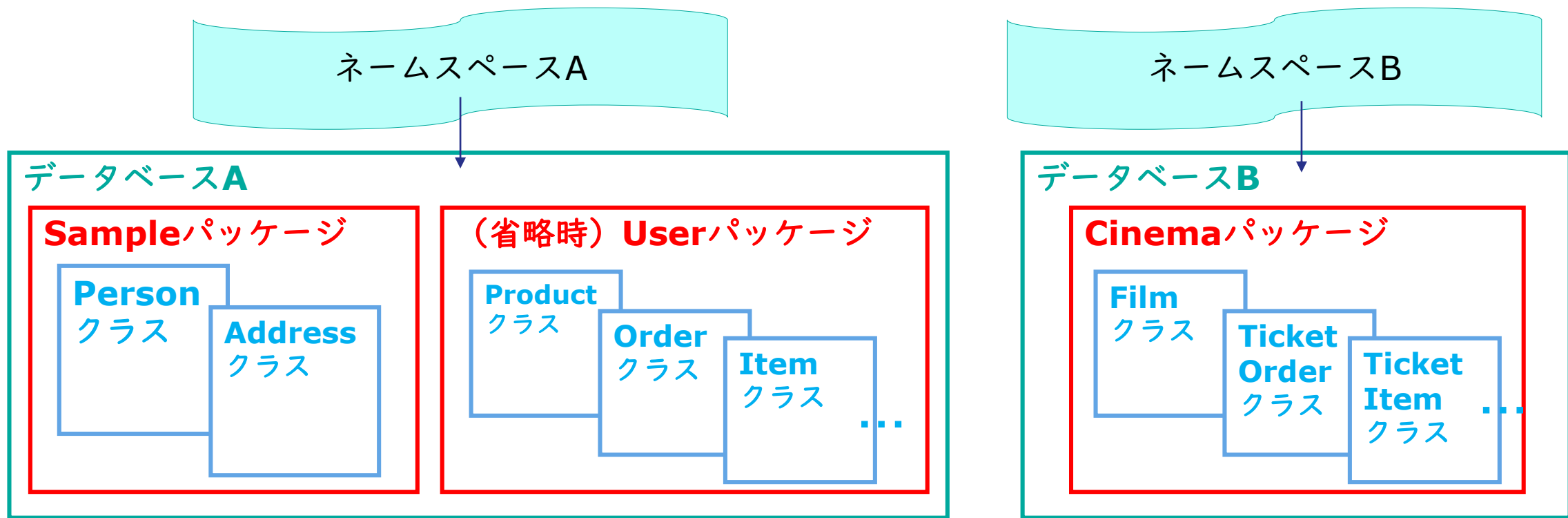
IRISのクラス定義に記述できるメソッド作成方法、実行方法を体験します。

- 前の演習で作成したスクレイピングを行うPythonの関数を、ObjectScriptメソッドから呼び出してみます（IRISにクラス定義を用意して試します）。
- スクレイピングを行うPython関数をIRISのPythonメソッドで記述してみます。
- PythonからObjectScriptメソッド呼び出す方法を確認します。
- ObjectScriptメソッドにPythonから可変長の引数を渡す場合の引数定義方法を確認します。
- PythonのTrue／False／NoneをObjectScriptで扱う方法を確認します。

IRISにクラスを作成する場合のルール

クラスは、パッケージ名.クラス名 で表現します。

パッケージ名は省略できますが、省略した場合は、システムデフォルトで設定されている **User** が使用されます (User.クラス名)。



IRISのクラス定義

(インスタンスを永続化する？しない？を選択できます)

最も簡単なクラスは、メソッドだけを定義するメソッドのコンテナとしたクラスです。

- プロパティを持たないクラスでロジック（メソッド）だけを定義します。
- アプリケーションロジックのユーティリティコードや、ストアドプロシージャ用ロジックを定義するクラスとして使用します。

メソッドは、特定の動作を実行するためのコードです。

- メソッドコード中で使用する引数や変数は、プライベートスコープです。

メソッドの記述には、ObjectScript または **Python** を使用します（Language属性で指定します）。

<以下、本コースでは取り扱いません>

メソッドの他にデータ要素であるプロパティを定義するクラスもあります。

- データベースに格納する機能を持つクラス（永続クラス=%Persistentを継承）
- データベースに格納しないクラス（メモリ上にインスタンスは作成できるけど、データベースには保存できないクラス=%RegisteredObjectを継承）
- 永続クラスのプロパティにオブジェクト参照を使用して埋め込むことが前提のクラス（埋め込みクラス= %SerialObjectを継承）

クラスメソッドだけを定義するクラスの例

クラス定義文は以下の規則で記述します。

Class **パッケージ名**.**クラス名** Extends スーパークラス

- 特に何も継承する必要がないとき（クラスメソッドだけを定義する場合など）は、**Extends**以降は不要です。

Class **FS**.**Utils**

<以下補足>

- 多重継承する場合は、スーパークラスをカンマで区切って指定し、Extends以降を括弧で括ります。
 - 同じ名称の定義が存在する場合は、左に書いたクラスが優先されます。

Class **FS**.**Person** Extends (%Persistent,%Populate)

ObjectScriptメソッドの作成と実行

クラス**FS.Utils**を作成し、ターミナルで操作したPythonスクリプトファイルの呼び出しをクラスメソッドに記述します。

```
ClassMethod getTitle(url As %String) As %String
{
    // スクリプトファイルがあるパスをsys.pathに追加します
    set sys=##class(%SYS.Python).Import("sys")
    do sys.path.append("c:¥Workspace¥TryIRIS")
    // スクリプトファイルをインポートします。
    set scraping=##class(%SYS.Python).Import("scraping")
    set title=scraping.gettitle(url)
    return title
}
```

```
USER>write ##class(FS.Utils).getTitle("https://jp.community.intersystems.com/node/517211")
InterSystems グランプリ・プログラミングコンテスト 2022 開催！ | InterSystems Developer Community
```

Pythonメソッドの作成と実行

language=pythonの指定

クラス**FS.Utils**に新しいクラスメソッドをlanguage=pythonで用意し、スクレイピングの関数を書き写して実行できるか確認します。

- 引数のURLは文字列なので引数のデータタイプは %String を設定します。
- タイトルが文字列で戻るなので戻り値のデータタイプは %String を設定します。

```
ClassMethod scraping(url As %String) As %String [ Language = python ]
{
    import requests
    from bs4 import BeautifulSoup
    response=requests.get(url)
    soup=BeautifulSoup(response.text,'html.parser')
    title=soup.find('title').get_text()
    return title
}
```

```
USER>write ##class(FS.Utils).scraping("https://jp.community.intersystems.com/node/517211")
InterSystems グランプリ・プログラミングコンテスト 2022 開催！ | InterSystems Developer Community
```

URLをパースしてグローバルに登録する

Pythonの`urllib.parse`をインポートして、引数のURLをパースし、パスを取得します。

任意のグローバル変数を用意し、サブスクリプトにパスを設定し、タイトルをデータに設定してみます。

- グローバルの操作のために、**irisモジュール**をインポートする必要があります。
- グローバルを操作するために、グローバル参照を **`iris.gref("グローバル変数名")`**で取得します。

```
ClassMethod scraping(url As %String) As %String [ Language = python ]
{
    import requests
    from bs4 import BeautifulSoup
    import urllib.parse as parser
    import iris
    response=requests.get(url)
    soup=BeautifulSoup(response.text,'html.parser')
    title=soup.find('title').get_text()
    path=parser.urlparse(url).path
    glo=iris.gref("^DEVCOM")
    glo[path]=title
    return title
}
```

必要なモジュールをインポートします。

実行後、以下のグローバル変数が設定されていることを確認できます。
(実行方法は前頁と同じです)

グローバル
^DEVCOMに設定

```
USER>zwrite ^DEVCOM
^DEVCOM("/node/517211")="InterSystems グランプリ・プログラミングコンテスト 2022開催！ | InterSystems Developer Community"
```

グローバルの設定を行うObjectScriptメソッドの作成 + Pythonからメソッドを呼び出す

パスとタイトルを引数に指定できるように
ObjectScriptのクラスメソッドを用意します
(グローバル変数名は任意名で作成してくだ
さい)。

```
ClassMethod setGlo(path As %String, title As %String)
{
    set ^DEVCOM(path)=title
}
```

```
ClassMethod scraping(url As %String) As %String [ Language = python ]
{
    import requests
    from bs4 import BeautifulSoup
    import urllib.parse as parser
    import iris
    response=requests.get(url)
    soup=BeautifulSoup(response.text,'html.parser')
    title=soup.find('title').get_text()
    path=parser.urlparse(url).path
    #glo=iris.gref("^DEVCOM")
    #glo[path]=title
    iris.cls("FS.Utils").setGlo(path,title)
    return title
}
```

PythonからIRISのクラスメソッドを呼び出す時は、
`iris.cls("パッケージ名.クラス名").メソッド名()`
を利用します。

ObjectScriptメソッドに可変長引数を渡す場合の書き方

ObjectScriptメソッドに可変長引数を渡す場合は、メソッドに定義する引数も可変長引数用の記述（**引数名...**）で定義します。

- Pythonの***arg**で渡す可変長引数にも対応できます。

```
ClassMethod Kahen(input... As %String)
{
    zwrite input
}
```

可変長引数は、配列変数を利用してアクセスできます。

```
>>> import iris
>>> list=["これは","可変個の引数です","3つあります"]
>>> iris.cls("FS.Utills").Kahen(*list)
input=3
input(1)="これは"
input(2)="可変個の引数です"
input(3)="3つあります"
>>>
>>> iris.cls("FS.Utills").Kahen("a","b","c","d")
input=4
input(1)="a"
input(2)="b"
input(3)="c"
input(4)="d"
>>>
```

PythonのTrue／False／None ObjectScriptで操作する方法

ObjectScriptでは True／False／None が存在しないため、%SYS.Pythonクラスのメソッドを利用して対応させます。

- True : %SYS.PythonのTrue()
- False : %SYS.PythonのFalse()
- None : %SYS.PythonのNone()

```
USER>do ##class(FS.Utills).TFTest(99)  
Flase返ってきた
```

```
USER>do ##class(FS.Utills).TFTest(1)  
True返ってきた
```

```
ClassMethod TrueFalse(in As %Integer) As %SYS.Python  
{  
    if in=1 {  
        return ##class(%SYS.Python).True()  
    }  
    else {  
        return ##class(%SYS.Python).False()  
    }  
}
```

```
ClassMethod TFTest(input As %Integer) [ Language = python ]  
{  
    import iris  
    ret=iris.cls("FS.Utills").TrueFalse(input)  
    if ret==True:  
        print("True返ってきた")  
    else:  
        print("Flase返ってきた")  
}
```

ここからは・・・

IRISからPythonスクリプトファイルやPythonのメソッド呼出時、エラーが発生した場合の対応方法をご紹介します。

Pythonでエラーが発生した場合

Pythonでエラーが発生した場合、ObjectScriptのシステムエラーと同じ扱いになります。

```
ClassMethod errtest1(a As %Integer, b As %Integer) As %Integer
[ Language = python ]
{
    result=a/b
    return result
}
```

USER>write \$ZE

実行結果

USER>write ##class(FS.Utills).errtest1(1,0)

WRITE ##CLASS(FS.Utills).errtest1(1,0)

<THROW> *%Exception.PythonException 230 ^0^WRITE ##CLASS(FS.Utills).errtest1(1,0) <class 'ZeroDivisionError': division by zero -

USER>write \$ZE

<THROW> *%Exception.PythonException 230 ^0^WRITE ##CLASS(FS.Utills).errtest1(1,0) <class 'ZeroDivisionError': division by zero -

ObjectScriptのシステムエラーと同様に
\$ZEにエラー情報が格納されます。（\$ZE
は、\$ZERRORの省略形です）

test1.py 2 x

test1.py > table

```
27 def err1(a,b):
28     result=a/b
29     return result
```



USER>set sys=##class(%SYS.Python).Import("sys")

実行結果

USER>do sys.path.append("c:¥Workspace¥TryIRIS")

USER>set test1=##class(%SYS.Python).Import("test1")

USER>write test1.err1(1,0)

WRITE test1.err1(1,0)

^

<THROW> *%Exception.PythonException 230 ^0^WRITE test1.err1(1,0) <class 'ZeroDivisionError': division by zero -

USER>write \$ZE

<THROW> *%Exception.PythonException 230 ^0^WRITE test1.err1(1,0) <class 'ZeroDivisionError': division by zero -

\$ZEにエラー入る = システムエラー → 自動でCATCHに移動します

Try-Catch利用時、ObjectScript で<UNDEFIND>などシステムエラーが発生した場合、例外オブジェクトを自動生成しCATCHブロックへ移動します。

- 例外オブジェクトは %Exception.SystemException クラスを利用

Pythonでエラーを起こした場合も、ObjectScriptのシステムエラー同様に\$ZEにエラー文字列を設定しているので、ObjectScriptのシステムエラー発生と同様の流れで操作できます。

- 例外オブジェクトは %Exception.PythonException クラスを利用

ご参考： %Statusのエラーはどうなるでしょう

エラーステータスを `iris.check_status()` でチェックすると `RuntimeError` の例外が発生します。

```
ClassMethod statustest() [ Language = python ]
```

```
{
import iris
try:
    a=iris.cls("FS.Person")._New()
    a.DOB="ThisIsError"
    st=a._Save()
    iris.check_status(st)
except RuntimeError as ex:
    [ print("pythonのエクセプション！") ]
    print(str(repr(ex)))
    raise
}
```

\$ZEにセットされると同時に、%objlasterror変数にもエラーステータスの内容がセットされます。

[illegible]

ご参考：SQL実行時のエラーはどうなるでしょう

irisbuiltins.SQLErrorの例外が発生します

```
def sqlerr():  
    import iris  
    import irisbuiltins  
    try:  
        sql="select * from Training.Person"  
        rset=iris.sql.exec(sql)  
        for key,val in enumerate(rset):  
            print(val)  
  
    except irisbuiltins.SQLError as ex:  
        print(str(repr(ex)))  
        print(ex.sqlcode)  
        print(ex.message)  
        print(ex.statement)  
        raise
```

\$ZEがセットされます→

```
USER>write $ZE
```

```
USER>do ##class(%SYS.Python).Shell()
```

```
Python 3.9.10 (tags/v3.9.10:f2f3f53, Jan 17 2022,  
15:14:21) [MSC v.1929 64 bit (AMD64)] on win32  
Type quit() or Ctrl-D to exit this shell.
```

```
>>> import sys
```

```
>>> sys.path+=["c:¥Workspace¥TryIRIS"]
```

```
>>> import test1
```

```
>>> test1.sqlerr()
```

```
SQLError(" テーブル 'TRAINING.PERSON' が見つかりません")
```

```
-30
```

```
テーブル 'TRAINING.PERSON' が見つかりません
```

```
select * from Training.Person
```

```
Traceback (most recent call last):
```

```
File "<input>", line 1, in <module>
```

```
File "c:¥Workspace¥TryIRIS¥test1.py", line 36, in sqlerr
```

```
    rset=iris.sql.exec(sql)
```

```
irisbuiltins.SQLError: テーブル 'TRAINING.PERSON' が見つかりません
```

```
>>> quit()
```

```
USER>write $ZE
```

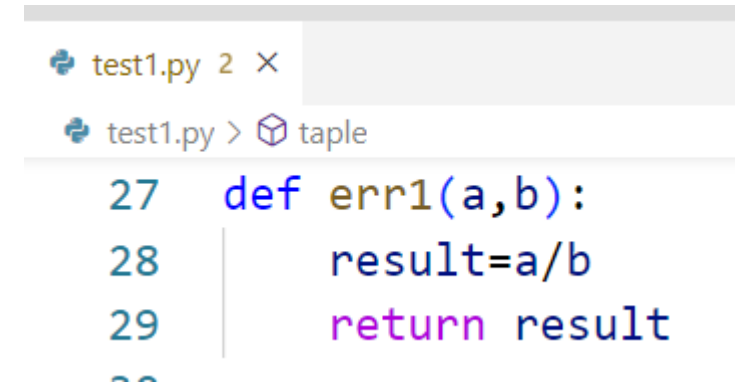
```
<SYNTAX>errdone+2^%qaqqt
```

```
USER>
```

Python側でエラー処理を書かない場合

何もしない（try: except: を書かない）状態で以下のようなコードを実行した場合

```
ClassMethod errtest1(a As %Integer, b As %Integer) As %Integer
[ Language = python ]
{
    result=a/b
    return result
}
```



The screenshot shows a code editor with a tab labeled 'test1.py 2 x'. Below the tab, the text 'test1.py > taple' is visible. The code being edited is a Python function definition:

```
27 def err1(a,b):
28     result=a/b
29     return result
~~
```

- Pythonのエラーが返ります。
- IRISから呼び出した場合、Pythonのエラーが\$ZEにセットされます。

アプリケーション実行中なら停止します。

Python側の選択肢：エラーの対応方法

1. 戻り値を使ってエラーだったことを報告する

- 呼び出し元が何であれ、エラーでもエラーでなくても戻り値が返るので、呼び出し側はそれを考えてコーディングするだけ
- 戻り値のタイプは自由に決定できる
 - 単純値 (IntegerやStringなど)
 - オブジェクトなどもあり
 - IRISのタイプ%Status (※) を返すもあり

2. try: except: を使う + raise でそのままIRISに例外を戻す

- Python側全エラー (SQLも%Statusも一般も) をPythonExceptionとして受け取れる
- PythonExceptionで受け取ったIRIS側は、%Statusや単純な文字への変換ができる

※Interoperabilityのコンポーネントを開発する際、%Statusを戻り値に設定するメソッドが登場します。

Python側の選択肢：エラーの対応方法

1. 戻り値を使ってエラーだったことを報告する例 (*.py)

```
def err2(a,b):  
    try:  
        if b==1:  
            modori="1で割っても答えは同じです"  
            return modori  
        print(f"割り算の答えは={a/b}")  
        modori="OK"  
        return modori  
  
    except ZeroDivisionError as ex:  
        modori=str(repr(ex))  
        print(modori)  
        return modori
```

test1.py

正常な時はOK、エラーが起きたときは文字列を返す

```
USER>set sys=##class(%SYS.Python).Import("sys")  
  
USER>do sys.path.append("C:¥Workspace¥TryIRIS")  
  
USER>set errtest=##class(%SYS.Python).Import("test1")  
  
USER>set ret=errtest.err2(2,2)  
割り算の答えは=1.0  
  
USER>write ret  
OK  
USER>set ret=errtest.err2(2,1)  
  
USER>write ret  
1で割っても答えは同じです  
USER>set ret=errtest.err2(2,0)  
ZeroDivisionError('division by zero')  
  
USER>write ret  
ZeroDivisionError('division by zero')  
USER>
```


Python側の選択肢：エラーの対応方法

1. 戻り値を使ってエラーだったことを報告する例 (language=python)

```
ClassMethod errtest2(a As %Integer, b As %Integer) As %Integer [ Language = python ]
{
    try:
        if b==1:
            modori="1で割っても答えは同じです"
            return modori
        print(f"割り算の答えは={a/b}")
        modori="OK"
        return modori
    except ZeroDivisionError as ex:
        modori=str(repr(ex))
        print(modori)
        return modori
}
```

前頁と同じ内容をlanguage=pythonで書いた場合の例

```
USER>set modori=##class(FS.Utls).errtest2(2,2)
割り算の答えは=1.0
```

```
USER>write modori
OK
```

```
USER>set modori=##class(FS.Utls).errtest2(2,1)
```

```
USER>write modori
1で割っても答えは同じです
```

```
USER>set modori=##class(FS.Utls).errtest2(2,0)
ZeroDivisionError('division by zero')
```

```
USER>write modori
ZeroDivisionError('division by zero')
```

Python側の選択肢：エラーの対応方法

1. 戻り値を使ってエラーだったことを報告する例（%Statusを戻す場合）

language=pythonのコードで%Statusを戻さないといけないとき

- InteroperabilityのコンポーネントのメソッドをPythonで記述する際必要になります。

```
ClassMethod errtest3(a As %Integer, b As %Integer) As %Status [ Language = python ]
{
    import iris
    try:
        print(a/b)
        ret=1
    except Exception as ex:
        moji="エラーが発生しました！"+str(repr(ex))
        ret=iris.system.Status.Error(5001,moji)
    return ret
}
```

%Statusエラーを作成するために、**irisモジュール**の**system.Status**クラスの**Error()**メソッドを使用します。

```
USER>set status=##class(FS.Utills).errtest3(1,0)
```

```
USER>write $system.Status.GetErrorText(status)
```

```
エラー #5001: エラーが発生しました！
```

```
ZeroDivisionError('division by zero')
```

```
USER>
```

```
USER>set status=##class(FS.Utills).errtest3(1,1)
```

```
1.0
```

```
USER>write status
```

```
1
```

Python側の選択肢：エラーの対応方法

2.try: except: を使う + raise でそのままIRISに例外を戻す

```
def err3(a,b):  
    try:  
        ret=a/b  
        print(ret)  
    except:  
        raise
```

ObjectScriptの中ではCATCHブロック内で
すべて受け取れる + 例外オブジェクトから
必要なタイプに変換可

```
ClassMethod errtest4() As %Status  
{  
    #dim ex As %Exception.AbstractException  
    try {  
        set sys=##class(%SYS.Python).Import("sys")  
        do sys.path.append("C:¥WorkSpace¥TryIRIS")  
        set errtest=##class(%SYS.Python).Import("test1")  
        do errtest.err3(1,0)  
    }  
    catch ex {  
        write "エラーが発生しました:",ex.DisplayString(),!  
        //例外から%Statusに変換  
        set st=ex.AsStatus()  
        //例外からSQLCODEとメッセージを取得  
        set SQLCODE=ex.AsSQLCODE()  
        set SQLMessage=ex.AsSQLMessage()  
    }  
}
```

```
USER>do ##class(FS.Utills).errtest4()  
エラーが発生しました: 230 zerrtest4+5^FS.Utills.1^8^  
<class 'ZeroDivisionError': division by zero -
```

```
do errtest.err3(1,0)
```

この資料で確認できたこと

- IRISのターミナルやirispythonコマンドを使用してPythonシェルを起動できることを確認できました。
- ObjectScriptからPythonスクリプトファイルの実行方法を確認できました。
- Pythonの組み込み関数をObjectScriptで使用方法を確認できました。
- Python／ObjectScriptでメソッドを記述し、実行する方法を確認できました。
- irisモジュールを利用することでPythonからIRISのクラス、SQL、グローバルを操作できることを確認できました。
- Pythonでエラーが発生した場合、ObjectScriptのシステムエラーと同様の方法で、エラー処理が行えることを確認できました。

ここからは・・・

参考情報をお伝えします。

- グローバル変数の操作
- グローバル変数の\$order()をPythonで行うには？
- Pythonの○○はObjectScriptの△に似てる？

グローバル変数の操作

グローバルを操作するために、**iris.gref()**を利用して、操作するグローバル変数の参照を取得します：
glo=iris.gref("^Relation")

グローバル変数直下に値を設定する場合は、**glo[None]="テスト"** とします。

グローバル変数の添え字を使用する場合は、**glo["添え字1","添え字2"]="データ"** とします。

この時点のグローバル変数 ☞

```
USER>zwrite ^Relation
^Relation="テスト"
^Relation("添え字1","添え字2")="データ"
```

添え字（添え字1）を指定してグローバル変数を削除 ☞ **glo.kill(["添え字1"])**

グローバル変数をトップノードから削除 ☞ **glo.kill([None])**

グローバル変数の\$order()をPythonで行うには？

<https://jp.community.intersystems.com/node/511476>

人物相関図をグローバルにセットしたデータを利用して、\$Order()関数の操作をPythonから試します。

```
>>> glo["Eren"]="主人公エレン"  
>>> glo["Eren","Armin"]=""  
>>> glo["Eren","Mikasa"]=""  
>>> glo["Eren","Zeke"]=""  
>>> glo["Armin"]="エレンの幼馴染（アルミン）"  
>>> glo["Mikasa"]="エレンの幼馴染（ミカサ）"  
>>> glo["Zeke"]="エレンの異母兄弟"
```

ZWRITEの結果

```
USER>zwrite ^Relation  
^Relation("Armin")="エレンの幼馴染（アルミン）"  
^Relation("Eren")="主人公エレン"  
^Relation("Eren","Armin")=""  
^Relation("Eren","Mikasa")=""  
^Relation("Eren","Zeke")=""  
^Relation("Mikasa")="エレンの幼馴染（ミカサ）"  
^Relation("Zeke")="エレンの異母兄弟"
```

order()メソッドを使用して第2サブスクリプトの情報を取得しています。

```
>>> sub=""  
>>> while True:  
...  
sub=glo.order(["Eren",sub])  
... if (sub==None):  
...     break  
...     print(sub)  
...  
Armin  
Mikasa  
Zeke
```

Pythonの〇〇はObjectScriptの△に似てる？

Pythonの文字列はインデックス番号を添え字に指定すると部分抽出できる = \$Extract() に似てる

Pythonは設定不可、\$Extract()は設定可

```
>>> word="python"
>>> word[0]
'p'
>>> word[1]
'y'
>>> word[2]
't'
>>> word[3]
'h'
>>> word[4]
'o'
>>> word[5]
'n'
>>> word[0:2]
'py'
>>> word[-1:]
'n'
>>> word[-2:]
'on'
>>>
```


Pythonの〇〇はObjectScriptの△に似てる？

リストのポジション指定の設定は、\$Extract()の設定に少し似てる

```
>>> l1=[1,2,3,4]
>>> l1
[1, 2, 3, 4]
>>> l1[2]="値を変更します"
>>> l1
[1, 2, '値を変更します', 4]
>>>
>>> l1[1:2]=["範囲を指定した設定も","できます"]
>>> l1
[1, '範囲を指定した設定も', 'できます', '値を変更します', 4]
>>>
```

```
USER>set moji="1234"
```

```
USER>write moji
```

```
1234
```

```
USER>set $Extract(moji,3)="値を変更します"
```

```
USER>write
```

```
moji="12値を変更します4"
```

```
USER>
```

その他情報について

コミュニティにも利用例が掲載されています。

- [1対多のリレーションシップを使った例（レシートの中身をIRISに登録する例）](#)
- [Excel のデータを IRIS グローバルに格納する方法](#)

最新情報については、以下タグにアクセスしてください。

- [#Embedded Python](#)

より良い方法を発見した場合、エラーが出た場合、使い方が不明な場合は、ぜひコミュニティへ投稿してください！