組み込み Python の使用法

Version 2023.1
2024-01-02

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

組み込み Python の使用法
InterSystems IRIS Data Platform　Version 2023.1　2024-01-02

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

組み込み Python の使用法

組み込み Python により、InterSystems IRIS アプリケーションをプログラミングするネイティブ・オプションとして Python を使用できるようになります。組み込み Python を使用するのが初めてである場合は、まず "組み込み Python の概要" を参照し、その後組み込み Python のより詳細な説明として、このドキュメントをお読みください。

このドキュメントは組み込み Python について学習する人すべてに役立ちますが、理解するうえで ObjectScript についてのある程度の知識を持っていることが望ましいでしょう。InterSystems IRIS および ObjectScript に不慣れな Python 開発者の場合、"サーバ側プログラミングの入門ガイド" も参照してください。

1 前提条件

組み込み Python の使用に必要な Python のバージョンは、実行しているプラットフォームによって異なります。ほとんどの場合は、お使いのオペレーティング・システムの Python の既定バージョンです。オペレーティング・システムと対応する Python のサポート対象バージョンの完全なリストは、"その他のサポート対象機能" を参照してください。組み込み Python の使用時に異なるバージョンの Python を使用すると、エラーが発生します。

Microsoft Windows では、InterSystems IRIS インストール・キットによって、組み込み Python 専用の正しいバージョンの Python (現在は 3.9.5) がインストールされます。開発マシンを使用していて、Python を一般用途に使用したい場合は、この同じバージョンを https://www.python.org/downloads/ からダウンロードしてインストールすることをお勧めします。

多くの UNIX ベースのオペレーティング・システムでは、既に Python がインストールされています。インストールする必要がある場合は、パッケージ・マネージャによりご使用のオペレーティング・システムに推奨されるバージョンを使用してください。以下に例を示します。

- macOS : Homebrew (https://formulae.brew.sh/formula/python@3.9) を使用して Python 3.9 をインストール
- Ubuntu : `apt-get install python3`
- Red Hat Enterprise Linux または Oracle Linux : `yum install python3`
- SUSE : `zypper install python3`

"Python をロードできませんでした" というエラーが表示された場合は、システムに Python がインストールされていないか、Python の予期しないバージョンが検出されたかのどちらかです。"その他のサポート対象機能" を調べて、必要なバージョンの Python がインストールされていることを確認し、必要に応じて、上記のいずれの方法で Python をインストールまたは再インストールしてください。

重要　コンピュータに複数のバージョンの Python がインストールされている場合、コマンド行から組み込み Python の実行を試行すると、irispython は最初に検出された python3 実行可能ファイル (PATH 環境変数よって決まる) を実行します。正しいバージョンの実行可能ファイルが最初に検出されるように、パスのフォルダが適切に設定されていることを確認してください。irispython コマンドの使用の詳細は、"コマンド行からの Python シェルの開始" を参照してください。

UNIX ベースのシステムでは、pip3 コマンドを使用して Python パッケージをインストールできます。pip3 をまだインストールしていない場合は、システムのパッケージ・マネージャでパッケージ `python3-pip` をインストールしてください。

組み込み Python の実行時に `IRIS_ACCESSDENIED` エラーが発生しないようにするには、`%Service_Callin` を有効にします。管理ポータルで、**[システム管理]→[セキュリティ]→[サービス]** に移動し、[%Service_CallIn] を選択し、**[サービス有効]** ボックスにチェックを付けます。

2 組み込み Python の実行

このセクションでは、組み込み Python を実行するいくつかの方法を説明します。

- Python シェルから実行
- Python スクリプト・ファイル (.py) 内で実行
- InterSystems IRIS クラスのメソッド内で実行
- SQL 関数およびストアド・プロシージャ内で実行

これらすべての方法で、`iris` Python モジュールをインポートしてそのメソッドを使用することで、InterSystems IRIS API を呼び出すことができます。

2.1 Python シェルから実行

インターシステムズのターミナル・セッションまたはコマンド行から Python シェルを開始できます。

2.1.1 ターミナルからの Python シェルの開始

インターシステムズのターミナル・セッションから Python シェルを開始するには、**%SYS.Python** クラスの Shell() メソッドを呼び出します。これにより、Python インタプリタがインタラクティブ・モードで起動します。ターミナル・セッションからユーザとネームスペースが Python シェルに渡されます。

コマンド `quit()` を入力して、Python シェルを終了します。

以下の例は、ターミナル・セッションの **USER** ネームスペースから Python シェルを起動します。これは、フィボナッチ数列の最初の数個の数字を出力し、その後 InterSystems IRIS の %SYSTEM.OBJ.ShowClasses() メソッドを使用して現在のネームスペース内のクラスのリストを出力します。

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

Python 3.9.5 (default, Jul  6 2021, 13:03:56) [MSC v.1927 64 bit (AMD64)] on win32
Type quit() or Ctrl-D to exit this shell.
>>> a, b = 0, 1
>>> while a < 10:
...     print(a, end=' ')
...     a, b = b, a+b
...
0 1 1 2 3 5 8 >>>
>>> status = iris.cls('%SYSTEM.OBJ').ShowClasses()
User.Company
User.Person
>>> print(status)
1
>>> quit()

USER>
```

メソッド %SYSTEM.OBJ.ShowClasses() は、InterSystems IRIS の **%Status** の値を返します。この場合、1 はエラーが検出されなかったことを意味します。

注釈　**%SYS.Python** クラスの **Shell()** メソッドを使用した Python シェルの実行時に、`iris` モジュールを明示的にインポートする必要はありません。続行して、モジュールを使用してください。

2.1.2 コマンド行からの Python シェルの開始

irispython コマンドを使用して、コマンド行から Python シェルを開始します。これは、ターミナルからのシェルの開始とほぼ同様に機能しますが、InterSystems IRIS のユーザ名、パスワード、およびネームスペースを渡す必要があります。

以下の例は、Windows コマンド行から Python シェルを起動します。

```
C:\InterSystems\IRIS\bin>set IRISUSERNAME = <username>

C:\InterSystems\IRIS\bin>set IRISPASSWORD = <password>

C:\InterSystems\IRIS\bin>set IRISNAMESPACE = USER

C:\InterSystems\IRIS\bin>irispython
Python 3.9.5 (default, Jul  6 2021, 13:03:56) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

UNIX ベースのシステムでは、set ではなく、export を使用します。

```
/InterSystems/IRIS/bin$ export IRISUSERNAME=<username>
/InterSystems/IRIS/bin$ export IRISPASSWORD=<password>
/InterSystems/IRIS/bin$ export IRISNAMESPACE=USER
/InterSystems/IRIS/bin$ ./irispython
Python 3.9.5 (default, Jul 22 2021, 23:12:58)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

注釈　`import iris` を実行しようとして、`IRIS_ACCESSDENIED` というメッセージが表示された場合は、`%Service_Callin` を有効にしてください。管理ポータルで、[**システム管理**]→[**セキュリティ**]→[**サービス**] に移動し、[%Service_CallIn] を選択し、[**サービス有効**] ボックスにチェックを付けます。

2.2 Python スクリプト・ファイル (.py) 内で実行

irispython コマンドを発行して、Python スクリプトを実行することもできます。

Windows システムで、以下のコードを持つ C:¥python¥test.py ファイルを見てみましょう。

```
# print the members of the Fibonacci series that are less than 10
print('Fibonacci series:')
a, b = 0, 1
while a < 10:
    print(a, end=' ')
    a, b = b, a + b

# import the iris module and show the classes in this namespace
import iris
print('\nInterSystems IRIS classes in this namespace:')
status = iris.cls('%SYSTEM.OBJ').ShowClasses()
print(status)
```

test.py は、コマンド行から以下のように実行できます。

```
C:\InterSystems\IRIS\bin>set IRISUSERNAME = <username>

C:\InterSystems\IRIS\bin>set IRISPASSWORD = <password>

C:\InterSystems\IRIS\bin>set IRISNAMESPACE = USER

C:\InterSystems\IRIS\bin>irispython \python\test.py
Fibonacci series:
0 1 1 2 3 5 8
InterSystems IRIS classes in this namespace:
User.Company
User.Person
1
```

UNIX ベースのシステムでは、set ではなく、export を使用します。

```
/InterSystems/IRIS/bin$ export IRISUSERNAME=<username>
/InterSystems/IRIS/bin$ export IRISPASSWORD=<password>
/InterSystems/IRIS/bin$ export IRISNAMESPACE=USER
/InterSystems/IRIS/bin$ ./irispython /python/test.py
Fibonacci series:
0 1 1 2 3 5 8
InterSystems IRIS classes in this namespace:
User.Company
User.Person
1
```

注釈　`import iris` を実行しようとして、`IRIS_ACCESSDENIED` というメッセージが表示された場合は、`%Service_Callin` を有効にしてください。管理ポータルで、**[システム管理]→[セキュリティ]→[サービス]** に移動し、[%Service_CallIn] を選択し、**[サービス有効]** ボックスにチェックを付けます。

2.3 InterSystems IRIS クラスのメソッド内で実行

`Language` キーワードを使用して、InterSystems IRIS クラス内で Python メソッドを記述できます。その後、ObjectScript で記述されたメソッドを呼び出す場合と同じように、メソッドを呼び出せます。

例えば、Python で記述されたクラス・メソッドを持つ以下のクラスを取り上げます。

```
Class User.EmbeddedPython
{

/// Description
ClassMethod Test() As %Status [ Language = python ]
{
    # print the members of the Fibonacci series that are less than 10
    print('Fibonacci series:')
    a, b = 0, 1
    while a < 10:
        print(a, end=' ')
        a, b = b, a + b

    # import the iris module and show the classes in this namespace
    import iris
    print('\nInterSystems IRIS classes in this namespace:')
    status = iris.cls('%SYSTEM.OBJ').ShowClasses()
    return status
}

}
```

このメソッドは、ObjectScript から以下のように呼び出せます。

```
USER>set status = ##class(User.EmbeddedPython).Test()
Fibonacci series:
0 1 1 2 3 5 8
InterSystems IRIS classes in this namespace:
User.Company
User.EmbeddedPython
User.Person

USER>write status
1
```

または Python から以下のように呼び出せます。

```
>>> import iris
>>> status = iris.cls('User.EmbeddedPython').Test()
Fibonacci series:
0 1 1 2 3 5 8
InterSystems IRIS classes in this namespace:
User.Company
User.EmbeddedPython
User.Person
>>> print(status)
1
```

2.4 SQL 関数およびストアド・プロシージャ内で実行

組み込み Python を使用して SQL 関数またはストアド・プロシージャを記述することもできます。これには、以下のように CREATE 文で引数 LANGUAGE PYTHON を指定します。

```
CREATE FUNCTION tzconvert(dt TIMESTAMP, tzfrom VARCHAR, tzto VARCHAR)
    RETURNS TIMESTAMP
    LANGUAGE PYTHON
{
    from datetime import datetime
    from dateutil import parser, tz
    d = parser.parse(dt)
    if (tzfrom is not None):
        tzf = tz.gettz(tzfrom)
        d = d.replace(tzinfo = tzf)
    return d.astimezone(tz.gettz(tzto)).strftime("%Y-%m-%d %H:%M:%S")
}
```

コードは Python の datetime および dateutil モジュールから関数を使用します。

注釈　プラットフォームによっては、datetime および dateutil モジュールが既定でインストールされていないことがあります。この例を実行して ModuleNotFoundError が発生した場合は、"Python ライブラリの使用" の説明に従って欠落しているモジュールをインストールします。

以下の SELECT 文は、SQL 関数を呼び出し、現在の日付/時刻を東部標準時から協定世界時 (UTC) に変換します。

```
SELECT tzconvert(now(), 'US/Eastern', 'UTC')
```

この関数は以下のような結果を返します。

```
2021-10-19 15:10:05
```

3 ObjectScript からの組み込み Python コードの呼び出し

このセクションでは、ObjectScript から組み込み Python コードを呼び出すいくつかの方法を説明します。

- Python ライブラリの使用
- Python で記述された InterSystems IRIS クラス内でのメソッドの呼び出し
- Python で記述された SQL 関数またはストアド・プロシージャの実行
- 任意の Python コマンドの実行

ObjectScript コードを呼び出すのとほとんど同じ方法で Python コードを呼び出せる場合もありますが、%SYS.Python クラスを使用して 2 つの言語間のギャップを埋める必要がある場合もあります。詳細は、"ObjectScript と組み込み Python 間のギャップを埋める" を参照してください。

3.1 Python ライブラリの使用

組み込み Python では、何千もの便利なライブラリに簡単にアクセスできます。一般に "パッケージ" と呼ばれますが、これらは使用する前に Python Package Index (PyPI) から <installdir>/mgr/python ディレクトリにインストールする必要があります。

例えば、ReportLab Toolkit は、PDF およびグラフィックを生成するためのオープン・ソース・ライブラリです。以下のコマンドでは、パッケージ・インストーラ irispip を使用して Windows システムに ReportLab をインストールします。

```
C:\InterSystems\IRIS\bin>irispip install --target C:\InterSystems\IRIS\mgr\python reportlab
```

UNIX ベースのシステムでは、以下を使用します。

```
$ pip3 install --target /InterSystems/IRIS/mgr/python reportlab
```

パッケージをインストールしたら、%SYS.Python クラスの Import() メソッドにより、それを ObjectScript コードで使用できます。

ファイルの場所を指定すると、以下の ObjectScript メソッド CreateSamplePDF() は、サンプルの PDF ファイルを作成し、その場所に保存します。

```
Class Demo.PDF
{

ClassMethod CreateSamplePDF(fileloc As %String) As %Status
{
    set canvaslib = ##class(%SYS.Python).Import("reportlab.pdfgen.canvas")
    set canvas = canvaslib.Canvas(fileloc)
    do canvas.drawImage("C:\Sample\isc.png", 150, 600)
    do canvas.drawImage("C:\Sample\python.png", 150, 200)
    do canvas.setFont("Helvetica-Bold", 24)
    do canvas.drawString(25, 450, "InterSystems IRIS & Python. Perfect Together.")
    do canvas.save()
}

}
```

このメソッドの最初の行は、ReportLab の pdfgen サブパッケージから canvas.py ファイルをインポートします。コードの 2 番目の行は Canvas オブジェクトをインスタンス化し、その後続行して、InterSystems IRIS オブジェクトのメソッドを呼び出すのとほぼ同じ方法でそのメソッドを呼び出します。

その後、以下に示すように通常の方法でメソッドを呼び出せます。

```
do ##class(Demo.PDF).CreateSamplePDF("C:\Sample\hello.pdf")
```

以下の PDF が生成され、指定された場所に保存されます。

3.2 Python で記述された InterSystems IRIS クラス内でのメソッドの呼び出し

組み込み Python を使用して InterSystems IRIS クラスでメソッドを記述し、その後 ObjectScript で記述されたメソッドを呼び出すのと同じ方法で、ObjectScript からそのメソッドを呼び出すことができます。

次の例は `usaddress-scourgify` ライブラリを使用しています。このライブラリは Windows のコマンド行から以下のようにインストールできます。

```
C:\InterSystems\IRIS\bin>irispip install --target C:\InterSystems\IRIS\mgr\python usaddress-scourgify
```

UNIX ベースのシステムでは、以下を使用します。

```
$ pip3 install --target /InterSystems/IRIS/mgr/python usaddress-scourgify
```

以下のデモ・クラスには、米国の住所の各部分のプロパティと、Python で記述されたメソッドが含まれます。これは、`usaddress-scourgify` を使用して住所をアメリカ合衆国郵便公社の標準に従って正規化します。

```
Class Demo.Address Extends %Library.Persistent
{

Property AddressLine1 As %String;

Property AddressLine2 As %String;

Property City As %String;

Property State As %String;

Property PostalCode As %String;

Method Normalize(addr As %String) [ Language = python ]
{
```

```
    from scourgify import normalize_address_record
    normalized = normalize_address_record(addr)

    self.AddressLine1 = normalized['address_line_1']
    self.AddressLine2 = normalized['address_line_2']
    self.City = normalized['city']
    self.State = normalized['state']
    self.PostalCode = normalized['postal_code']
}

}
```

住所の文字列を入力として指定すると、クラスの Normalize() インスタンス・メソッドは、住所を正規化し、各部分を Demo.Address オブジェクトのさまざまなプロパティに格納します。

このメソッドは以下のように呼び出せます。

```
USER>set a = ##class(Demo.Address).%New()

USER>do a.Normalize("One Memorial Drive, 8th Floor, Cambridge, Massachusetts 02142")

USER>zwrite a
a=3@Demo.Address  <OREF>
+----------------- general information ---------------
|      oref value: 3
|      class name: Demo.Address
| reference count: 2
+----------------- attribute values ------------------
|       %Concurrency = 1  <Set>
|       AddressLine1 = "ONE MEMORIAL DR"
|       AddressLine2 = "FL 8TH"
|              City = "CAMBRIDGE"
|         PostalCode = "02142"
|              State = "MA"
+-----------------------------------------------------
```

3.3 Python で記述された SQL 関数またはストアド・プロシージャの実行

組み込み Python を使用して SQL 関数またはストアド・プロシージャを作成する際、InterSystems IRIS は、他のメソッドを呼び出すのと同じように ObjectScript から呼び出せるメソッドを持つクラスを投影します。

例えば、このドキュメントで前述した例に示す SQL 関数は、tzconvert() メソッドを持つクラス User.functzconvert を生成します。これを ObjectScript から呼び出すには、以下のようにします。

```
USER>zwrite ##class(User.functzconvert).tzconvert($zdatetime($h,3),"US/Eastern","UTC")
"2021-10-20 15:09:26"
```

ここでは、$zdatetime($h,3) を使用して現在の日付と時刻が $HOROLOG 形式から ODBC 日付形式に変換されます。

3.4 任意の Python コマンドの実行

組み込み Python コードを開発またはテストしている際、任意の Python コマンドを ObjectScript から実行すると都合がよい場合があります。これは、%SYS.Python クラスの Run() メソッドによって行えます。

例えば、このドキュメントの前半で使用した usaddress_scourgify パッケージからの normalize_address_record() 関数をテストしたいのに、その動作を思い出せない場合があるかもしれません。%SYS.Python.Run() メソッドを使用して、以下のようにターミナルから関数のヘルプを出力できます。

```
USER>set rslt = ##class(%SYS.Python).Run("from scourgify import normalize_address_record")

USER>set rslt = ##class(%SYS.Python).Run("help(normalize_address_record)")
Help on function normalize_address_record in module scourgify.normalize:
normalize_address_record(address, addr_map=None, addtl_funcs=None, strict=True)
    Normalize an address according to USPS pub. 28 standards.

    Takes an address string, or a dict-like with standard address fields
    (address_line_1, address_line_2, city, state, postal_code), removes
    unacceptable special characters, extra spaces, predictable abnormal
    character sub-strings and phrases, abbreviates directional indicators
    and street types.  If applicable, line 2 address elements (ie: Apt, Unit)
    are separated from line 1 inputs.
.
.
.
```

%SYS.Python.Run() メソッドは、成功した場合に 0 を返し、失敗した場合に -1 を返します。

4 ObjectScript と組み込み Python 間のギャップを埋める

ObjectScript と Python の言語には違いがあるため、その言語間のギャップを埋めるのに知っておく必要のある情報があります。

ObjectScript 側では、**%SYS.Python** クラスにより、ObjectScript から Python を使用できるようになります。詳細は、InterSystems IRIS のクラス・リファレンスを参照してください。

Python 側では、**iris** モジュールにより、Python から ObjectScript を使用できるようになります。Python から、モジュールと関数のリストを表示するには「**help(iris)**」と入力します。

このセクションでは、以下の項目について説明します。

- Python の組み込み関数の使用
- 識別子名
- キーワードまたは名前付き引数
- 引数の参照渡し
- True、False、None の値を渡す方法
- ディクショナリ
- リスト
- グローバル
- 組み込み Python からの ObjectScript コマンドの実行
- 例外処理
- Bytes 型と String 型
- シグナル処理

4.1 Python の組み込み関数の使用

builtins パッケージは Python インタプリタの起動時に自動的にロードされます。このパッケージには、ベース・オブジェクト・クラスや、すべての組み込みのデータ型クラス、例外クラス、関数、および定数など、この言語に組み込まれたすべての識別子が含まれます。

これらの識別子すべてにアクセスできるようにするには、以下のようにこのパッケージを ObjectScript にインポートします。

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

Python の print() 関数は、実際には builtins モジュールのメソッドであるため、この関数を以下のように ObjectScript から使用できるようになります。

```
USER>do builtins.print("hello world!")
hello world!
```

次に zwrite コマンドを使用して builtins オブジェクトを調べることができます。これは Python オブジェクトであるため、builtins パッケージの str() メソッドを使用して、そのオブジェクトの文字列表現を取得します。以下に例を示します。

```
USER>zwrite builtins
builtins=5@%SYS.Python  ; <module 'builtins' (built-in)>  ; <OREF>
```

同じトークンによって、builtins.list() メソッドを使用して Python リストを作成できます。以下の例は、空のリストを作成します。

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

USER>zwrite list
list=5@%SYS.Python  ; []  ; <OREF>
```

builtins.type() メソッドを使用して、変数 list がどの Python の型であるかを調べることができます。

```
USER>zwrite builtins.type(list)
3@%SYS.Python  ; <class 'list'>  ; <OREF>
```

興味深いことに、list() メソッドは、実際にはリストを表す Python のクラス・オブジェクトのインスタンスを返します。以下のようにリスト・オブジェクトで dir() メソッドを使用することで、list クラスが持つメソッドを確認できます。

```
USER>zwrite builtins.dir(list)
3@%SYS.Python  ; ['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__',
'__delitem__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__',
'__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__',

'__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__','__repr__', '__reversed__',
'__rmul__', '__setattr__', '__setitem__','__sizeof__', '__str__', '__subclasshook__', 'append','clear',

'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']  ; <OREF>
```

同様に、help() メソッドを使用して、リスト・オブジェクトについてのヘルプを取得できます。

```
USER>do builtins.help(list)
Help on list object:
class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
```

```
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
.
.
.
```

注釈　ObjectScript に builtins モジュールをインポートする代わりに、%SYS.Python クラスの Builtins() メソッドを呼び出すことができます。

4.2 識別子名

識別子の命名規則は、ObjectScript と Python で異なります。例えば、アンダースコア (_) は Python のメソッド名で許可されており、実のところいわゆる “ダンダー (dunder)” メソッドおよび属性 (“dunder” は “double underscore” の省略形) 用に広く使用されています (__getitem__ や __class__ など)。ObjectScript からそのような識別子を使用するには、以下のようにそれらを二重引用符で囲みます。

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

USER>zwrite mylist."__class__"
2@%SYS.Python  ; <class list>  ; <OREF>
```

逆に、InterSystems IRIS メソッドは多くの場合パーセント記号 (%) で始まります。例えば、%New() や %Save() のようになります。Python からそのような識別子を使用するには、パーセント記号をアンダースコアで置き換えます。永続クラス User.Person がある場合、Python コードの以下の行は、新しい Person オブジェクトを作成します。

```
>>> import iris
>>> p = iris.cls('User.Person')._New()
```

4.3 キーワードまたは名前付き引数

Python では、メソッドを定義する際にキーワード引数 (または ″名前付き引数″ とも呼ばれる) を使用するのが一般的な方法です。これにより、必要でない場合に引数を削除したり、場所ではなく名前に応じて引数を指定したりすることが容易になります。例として、以下の簡単な Python メソッドを見てみましょう。

```
def mymethod(foo=1, bar=2, baz="three"):
    print(f"foo={foo}, bar={bar}, baz={baz}")
```

InterSystems IRIS にはキーワード引数の概念がないため、ダイナミック・オブジェクトを作成して、キーワード/値のペアを保持する必要があります。例えば、以下のようになります。

```
set args={ "bar": 123, "foo": "foo"}
```

メソッド mymethod() が mymodule.py というモジュール内にある場合、それを ObjectScript にインポートして、以下のように呼び出すことができます。

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

USER>set args={ "bar": 123, "foo": "foo"}

USER>do obj.mymethod(args...)
foo=foo, bar=123, baz=three
```

baz はメソッドに渡されていないため、既定で "three" の値が割り当てられます。

4.4 引数の参照渡し

ObjectScript で記述されたメソッドの引数は、値または参照によって渡すことができます。以下のメソッドでは、シグニチャの 2 つ目と 3 つ目の引数の前にある ByRef キーワードは、それらを参照渡ししようとしていることを示しています。

```
ClassMethod SandwichSwitch(bread As %String, ByRef filling1 As %String, ByRef filling2 As %String)
{
    set bread = "whole wheat"
    set filling1 = "almond butter"
    set filling2 = "cherry preserves"
}
```

ObjectScript からメソッドを呼び出す際は、以下のように引数の前にピリオドを配置して、それを参照渡しします。

```
USER>set arg1 = "white bread"

USER>set arg2 = "peanut butter"

USER>set arg3 = "grape jelly"

USER>do ##class(User.EmbeddedPython).SandwichSwitch(arg1, .arg2, .arg3)

USER>write arg1
white bread
USER>write arg2
almond butter
USER>write arg3
cherry preserves
```

出力から、変数 arg1 の値が、SandwichSwitch() を呼び出した後も同じであるのに対し、変数 arg2 および arg3 の値は変化したことがわかります。

Python は参照による呼び出しをネイティブでサポートしないため、iris.ref() メソッドを使用して、参照渡しする各引数についてメソッドに渡す参照を作成する必要があります。

```
>>> import iris
>>> arg1 = "white bread"
>>> arg2 = iris.ref("peanut butter")
>>> arg3 = iris.ref("grape jelly")
>>> iris.cls('User.EmbeddedPython').SandwichSwitch(arg1, arg2, arg3)
>>> arg1
'white bread'
>>> arg2.value
'almond butter'
>>> arg3.value
'cherry preserves'
```

value プロパティを使用して arg2 と arg3 の値にアクセスし、それらがメソッドへの呼び出しの後に変化したことを確認できます。

4.5 True、False、None の値を渡す方法

%SYS.Python クラスには True()、False()、および None() のメソッドがあり、これらはそれぞれ Python の True、False、および None の識別子を表します。

以下に例を示します。

```
USER>zwrite ##class(%SYS.Python).True()
2@%SYS.Python  ; True  ; <OREF>
```

これらのメソッドは、Python メソッドに True、False、および None を渡す必要がある場合に役立ちます。以下の例は、"キーワードまたは名前付き引数" で示したメソッドを使用しています。

```
USER>do obj.mymethod(##class(%SYS.Python).True(), ##class(%SYS.Python).False(),
##class(%SYS.Python).None())
foo=True, bar=False, baz=None
```

キーワード引数を期待している Python メソッドに名前のない引数を渡すと、Python はそれらを渡された順序で処理します。

Python メソッドによって ObjectScript に返される値を調べる際に True()、False()、および None() のメソッドを使用する必要がないことに注意してください。

Python モジュール `mymodule` に、isgreaterthan() メソッドもあるとします。これは、以下のように定義されます。

```
def isgreaterthan(a, b):
    return a > b
```

Python で実行すると、このメソッドは引数 `a` が `b` よりも大きい場合に `True` を返し、そうでない場合に `False` を返すことがわかります。

```
>>> mymodule.isgreaterthan(5, 4)
True
```

しかし、ObjectScript から呼び出した場合、返される値は `1` であり、Python 識別子の `True` ではありません。

```
USER>zwrite obj.isgreaterthan(5, 4)
1
```

4.6 ディクショナリ

Python では、ディクショナリは一般的にキー/値のペアの形式でデータを格納するために使用されます。例えば以下のようになります。

```
>>> mycar = {
...     "make": "Toyota",
...     "model": "RAV4",
...     "color": "blue"
... }
>>> print(mycar)
{'make': 'Toyota', 'model': 'RAV4', 'color': 'blue'}
>>> print(mycar["color"])
blue
```

ObjectScript 側では、Python の `builtins` モジュールの dict() メソッドを使用して、Python ディクショナリを操作できます。

```
USER>set mycar = ##class(%SYS.Python).Builtins().dict()

USER>do mycar.setdefault("make", "Toyota")

USER>do mycar.setdefault("model", "RAV4")

USER>do mycar.setdefault("color", "blue")

USER>zwrite mycar
mycar=2@%SYS.Python  ; {'make': 'Toyota', 'model': 'RAV4', 'color': 'blue'}  ; <OREF>

USER>write mycar."__getitem__"("color")
blue
```

上記の例では、ディクショナリ・メソッド setdefault() を使用してキーの値を設定し、_getitem_() によってキーの値を取得しています。

4.7 リスト

Python では、リストは値のコレクションを格納しますが、キーを使用しません。リスト内の項目には、インデックスによってアクセスします。

```
>>> fruits = ["apple", "banana", "cherry"]
>>> print(fruits)
['apple', 'banana', 'cherry']
>>> print(fruits[0])
apple
```

ObjectScript では、Python の `builtins` モジュールの list() メソッドを使用して、Python リストを操作できます。

```
USER>set l = ##class(%SYS.Python).Builtins().list()

USER>do l.append("apple")

USER>do l.append("banana")

USER>do l.append("cherry")

USER>zwrite l
l=13@%SYS.Python  ; ['apple', 'banana', 'cherry']  ; <OREF>

USER>write l."__getitem__"(0)
apple
```

上記の例では、リスト・メソッド append() を使用してリストに項目を追加し、_getitem_() によって指定したインデックスの値を取得しています (Python リストはゼロベースです)。

4.8 グローバル

ほとんどの場合、InterSystems IRIS に保存されているデータには、SQL を使用するか、永続クラスとそのプロパティおよびメソッドを使用してアクセスできます。しかし、グローバルと呼ばれる、基盤となるネイティブの永続データ構造に直接アクセスしたいこともあります。従来のデータにアクセスする場合、または SQL テーブルや永続クラスには適していないスキーマレスのデータを保存している場合は特にそうです。

単純化しすぎではありますが、グローバルは、キーと値のペアのディクショナリと考えることができます。(正確な説明は、"グローバルの概要" を参照してください。)

Python で記述された 2 つのクラス・メソッドを持つ、以下のクラスを考えてみます。

```
Class User.Globals
{
ClassMethod SetSquares(x) [ Language = python ]
{
    import iris
    square = iris.gref("^square")
    for key in range(1, x):
        value = key * key
        square.set([key], value)
}
ClassMethod PrintSquares() [ Language = python ]
{
    import iris
    square = iris.gref("^square")
    key = ""
    while True:
        key = square.order([key])
        if key == None:
            break
        print("The square of " + str(key) + " is " + str(square.get([key])))
}
}
```

メソッド SetSquares() はキーの範囲をループして、グローバル `^square` の各ノードで各キーの二乗を格納します。メソッド PrintSquares() はグローバルを走査し、キーに格納されている各キーと値を出力します。

Python シェルを起動して、クラスをインスタンス化し、コードを実行してどのように動作するかを確認してみましょう。

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

Python 3.9.5 (default, May 31 2022, 12:35:47) [MSC v.1927 64 bit (AMD64)] on win32
Type quit() or Ctrl-D to exit this shell.
>>> g = iris.cls('User.Globals')
>>> g.SetSquares(6)
>>> g.PrintSquares()
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
```

今度は、組み込みの `iris` モジュールのいくつかのメソッドでグローバルにアクセスする方法を見てみましょう。

メソッド SetSquares() では、文 `square = iris.gref("^square")` はグローバル `^square` への参照 (gref とも呼ばれます) を返します。

```
>>> square = iris.gref("^square")
```

文 `square.set([key], value)` は、キー `key` を持つ `^square` のノードを値 `value` に設定します。例えば、`^square` のノード 12 を値 144 に設定できます。

```
>>> square.set([12], 144)
```

グローバルのノードを、次の短い構文で設定することもできます。

```
>>> square[13] = 169
```

メソッド PrintSquares() では、文 `key = square.order([key])` は ObjectScript の $ORDER 関数のように、キーを入力として取り、グローバルの次のキーを返します。グローバルを走査するには、通常、キーがもう残っていないことを示す None が返されるまで order() を使用し続けます。キーが連続している必要はないため、キーの間にギャップがある場合でも、order() は次のキーを返します。

```
>>> key = 5
>>> key = square.order([key])
>>> print(key)
12
```

次に、`square.get([key])` はキーを入力として取り、グローバルのそのキーの値を返します。

```
>>> print(square.get([key]))
144
```

再び、以下の短い構文を使用できます。

```
>>> print(square[13])
169
```

グローバルのノードがキーを持つ必要はありません。以下の文は、`^square` のルート・ノードに文字列を格納します。

```
>>> square[None] = 'Table of squares'
```

これらの Python コマンドが実際にグローバルに値を格納したことを示すために、Python シェルを終了し、ObjectScript で zwrite コマンドを使用して ^square の内容を出力します。

```
>>> quit()

USER>zwrite ^square
^square="Table of squares"
^square(1)=1
^square(2)=4
^square(3)=9
^square(4)=16
^square(5)=25
^square(12)=144
^square(13)=169
```

グローバル参照で使用できるメソッドの完全なリストを取得するには、Python シェルから help(iris) を入力してください。

4.9 組み込み Python からの ObjectScript コマンドの実行

システムが提供する"特殊変数"にアクセスする場合、ObjectScript で記述されたルーチンを呼び出す場合、呼び出すことができるメソッドがないその他のタスクを実行する場合など、組み込み Python から ObjectScript コマンドを実行したい場合があります。このような場合、Python から iris.execute() メソッドを使用できます。

次の例では、InterSystems IRIS のバージョン文字列を含む特殊変数 $zversion を記述します。

```
>>> iris.execute("write $zversion,!")
IRIS for Windows (x86-64) 2022.3 (Build 602U) Mon Jan 23 2023 14:05:04 EST
```

iris.execute() から値を返して、Python 変数に割り当てたい場合があります。この例では、InterSystems IRIS の内部ストレージ形式で現在のプロセスのローカルな日付と時刻を含む特殊変数 $horolog の値を、変数 t に割り当てます。

```
>>> t = iris.execute("return $horolog")
>>> t
'66499,55283'
```

これは、クラス %SYSTEM.SYS の Horolog() メソッドを使用する t = iris.cls('%SYSTEM.SYS').Horolog() と等価です。

クラスやメソッドの代わりにルーチンを使用する古い ObjectScript コードが使用されていて、組み込み Python からルーチンを呼び出したい場合があります。2 つの数値の合計を返す関数 Sum() を持つルーチン ^Math がある場合、次のように、2 つの数値を足して、返り値を Python 変数 sum に割り当てることができます。

```
>>> sum = iris.execute("return $$Sum^Math(4,3)")
>>> sum
>>> 7
```

組み込み Python からグローバルにアクセスする方法として推奨されるのは、iris.gref() メソッドを使用する方法ですが、iris.execute() を使用してグローバルを設定したり、グローバルから値を取得することもできます。以下の例では、グローバル ^motd を値 "hello world" に設定して、グローバルから値を取得します。

```
>>> iris.execute("set ^motd = \"hello world\"")
>>> iris.execute("return ^motd")
'hello world'
```

4.10 例外処理

InterSystems IRIS 例外ハンドラは、Python の例外を処理し、それらをシームレスに ObjectScript に渡します。以前の Python ライブラリの例 を基にして、存在しないファイルを使用して canvas.drawImage() の呼び出しを試行し、ObjectScript で例外を検出すると、以下のようになります。

```
USER>try { do canvas.drawImage("C:\Sample\bad.png", 150, 600) } catch { write "Error: ", $zerror, ! }
Error: <THROW> *%Exception.PythonException <THROW> 230 ^^0^DO canvas.drawImage("W:\Sample\isc.png",
150, 600)
<class 'OSError'>: Cannot open resource "W:\Sample\isc.png" -
```

ここで、`<class 'OSError'>: Cannot open resource "W:\Sample\isc.png"` は Python から戻された例外です。

4.11 Bytes 型と String 型

Python では、“bytes” データ型 (単に 8 ビット・バイトのシーケンス) のオブジェクトと、string (文字列を表す UTF-8 バイトのシーケンス) の間に明確な区別があります。Python では、bytes オブジェクトは決して変換されませんが、string はホスト・オペレーティング・システムによって使用される文字セットに応じて変換される場合があります (Latin-1 など)。

InterSystems IRIS では、bytes と string を区別しません。InterSystems IRIS では Unicode 文字列 (UCS-2/UTF-16) をサポートしていますが、256 より小さい値を含む文字列は、string と bytes のどちらにもなり得ます。このため、Python との間で string および bytes をやり取りする際は、以下のルールが適用されます。

- InterSystems IRIS の string は文字列と見なされ、ObjectScript から Python に渡される際に UTF-8 に変換されます。
- Python の string は、ObjectScript に戻されるときに、UTF-8 から InterSystems IRIS 文字列に変換され、その結果ワイド文字になります。
- Python の bytes オブジェクトは、8 ビット文字列として ObjectScript に返されます。bytes オブジェクトの長さが最大文字列長を超えると、Python の bytes オブジェクトが返されます。
- ObjectScript から Python に bytes オブジェクトを渡すには、##class(%SYS.Python).Bytes() メソッドを使用します。このメソッドは、基礎となる InterSystems IRIS 文字列を UTF-8 に変換しません。

以下の例は、InterSystems IRIS 文字列を bytes 型の Python オブジェクトに変換します。

```
USER>set b = ##class(%SYS.Python).Bytes("Hello Bytes!")

USER>zwrite b
b=8@%SYS.Python  ; b'Hello Bytes!'  ; <OREF>

USER>zwrite builtins.type(b)
4@%SYS.Python  ; <class 'bytes'>  ; <OREF>
```

InterSystems IRIS の最大文字列長である 3.8MB より大きい Python bytes オブジェクトを構築するには、bytearray オブジェクトを使用して、extend() メソッドにより bytes のより小さなチャンクを追加できます。最後に、bytearray オブジェクトを builtins の bytes() メソッドに渡して、bytes 表現を取得します。

```
USER>set ba = builtins.bytearray()

USER>do ba.extend(##class(%SYS.Python).Bytes("chunk 1"))

USER>do ba.extend(##class(%SYS.Python).Bytes("chunk 2"))

USER>zwrite builtins.bytes(ba)
"chunk 1chunk 2"
```

4.12 標準出力と標準エラーのマッピング

組み込み Python を使用する際、標準出力は InterSystems IRIS コンソールにマッピングされます。つまり、print() 文の出力はすべてターミナルに送信されます。標準エラーは、ディレクトリ <install-dir>/mgr にある InterSystems IRIS messages.log ファイルにマッピングされます。

例として、この Python メソッドについて考えてみましょう。

```
def divide(a, b):
    try:
        print(a/b)
    except ZeroDivisionError:
        print("Cannot divide by zero")
    except TypeError:
        import sys
        print("Bad argument type", file=sys.stderr)
    except:
        print("Something else went wrong")
```

ターミナルでこのメソッドをテストする場合、以下のような画面が表示されます。

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

USER>do obj.divide(5, 0)
Cannot divide by zero

USER>do obj.divide(5, "hello")
```

0 による除算を試行すると、エラー・メッセージがターミナルに転送されますが、文字列による除算を試行すると、メッセージは messages.log に送信されます。

```
11/19/21-15:49:33:248 (28804) 0 [Python] Bad argument type
```

ファイルが乱雑になることを防ぐため、重要なメッセージのみが messages.log に送信されるようにする必要があります。

5 相互運用プロダクション内での組み込み Python の使用法

InterSystems IRIS 内の相互運用プロダクション向けにカスタムのビジネス・ホスト・クラスやアダプタ・クラスを記述している場合、コールバック・メソッドはすべて ObjectScript で記述する必要があります。コールバック・メソッドは既定では何もしない継承メソッドですが、ユーザにより実装されるよう設計されています。ただし、コールバック・メソッドの ObjectScript コードは、Python ライブラリを活用したり、Python で実装されたその他のメソッドを呼び出すことができます。

以下の例は、受信メッセージから文字列値を取得し、Amazon Web Services (AWS) boto3 Python ライブラリを使用して、その文字列を Amazon Simple Notification Service (SNS) を介してテキスト・メッセージとして電話に送信するビジネス・オペレーションを示しています。この AWS ライブラリはここでの説明の範囲外ですが、この例では、OnInit() および OnMessage() コールバック・メソッドは ObjectScript で記述されているのに対し、PyInit() および SendSMS() のメソッドは Python で記述されていることがわかります。

```
/// Send SMS via AWS SNS
Class dc.opcua.SMS Extends Ens.BusinessOperation
{

Parameter INVOCATION = "Queue";

/// AWS boto3 client
Property client As %SYS.Python;

/// json.dumps reference
Property tojson As %SYS.Python;

/// Phone number to send SMS to
Property phone As %String [ Required ];
```

```
Parameter SETTINGS = "phone:SMS";

Method OnMessage(request As Ens.StringContainer, Output response As Ens.StringContainer) As %Status
{
   #dim sc As %Status = $$$OK
   try {
      set response = ##class(Ens.StringContainer).%New(..SendSMS(request.StringValue))
      set code = +{}.%FromJSON(response.StringValue).ResponseMetadata.HTTPStatusCode
      set:(code'=200) sc = $$$ERROR($$$GeneralError, $$$FormatText("Error sending SMS,
         code: %1 (expected 200), text: %2", code, response.StringValue))
   } catch ex {
      set sc  = ex.AsStatus()
   }

   return sc
}

Method SendSMS(msg As %String) [ Language = python ]
{
   response = self.client.publish(PhoneNumber=self.phone, Message=msg)
   return self.tojson(response)
}

Method OnInit() As %Status
{
   #dim sc As %Status = $$$OK
   try {
      do ..PyInit()
   } catch ex {
      set sc = ex.AsStatus()
   }
   quit sc
}

/// Connect to AWS
Method PyInit() [ Language = python ]
{
   import boto3
   from json import dumps
   self.client = boto3.client("sns")
   self.tojson = dumps
}

}
```

注釈　上記の OnMessage() メソッドのコードには、このドキュメントを印刷する際により適した書式設定とするために余分の改行が含まれています。

このルールの 1 つの例外は、アダプタからの入力を使用していない場合に、Python でコールバック・メソッドを実装できることです。

以下のビジネス・サービスの例は poller として知られています。この場合、ビジネス・サービスは間を置いて実行するよう設定でき、処理のためにビジネス・プロセスに送信される要求 (この場合はランダムな文字列値を含む) を生成します。この例では、OnProcessInput() コールバック・メソッドを Python に実装できます。これはこのメソッドのシグニチャで pInput 引数を利用していないためです。

```
Class Debug.Service.Poller Extends Ens.BusinessService
{

Property Target As Ens.DataType.ConfigName;

Parameter SETTINGS = "Target:Basic";

Parameter ADAPTER = "Ens.InboundAdapter";

Method OnProcessInput(pInput As %RegisteredObject, Output pOutput As %RegisteredObject,
    ByRef pHint As %String) As %Status [ Language = python ]
{
    import iris
    import random
    fruits = ["apple", "banana", "cherry"]
    fruit = random.choice(fruits)
    request = iris.cls('Ens.StringRequest')._New()
    request.StringValue = fruit + ' ' + iris.cls('Debug.Service.Poller').GetSomeText()
    return self.SendRequestAsync(self.Target,request)
}

ClassMethod GetSomeText() As %String
{
```

```
    Quit "is something to eat"
}
}
```

相互運用プロダクションのプログラミングに関する詳細は、"ビジネス・サービス、ビジネス・プロセス、およびビジネス・オペレーションのプログラミング" を参照してください。