Open Source 協議会 IBM i

## オープンソース協議会 勉強会　- Python 分科会 -

Ver.　2.0 | 2021-03-19

# IBM i で Python やってみた. 1


ティアンドトラスト株式会社 北原 征夫

T&Trust Tenderness and Trust
tat.co.jp

# アジェンダ

- python 概説
- インストール
- データベース・アクセス１
- データベース・アクセス２
- データの操作など
- 今後の python 分科会活動
- おまけ

# Pytho 概説

# Python 概説

- 歴史
  - 1991年 Guido van Rossum (グイド＝ヴァンロッサム:オランダ)　v0.9 公開
  - 1994年 Python 1.0 リリース
  - 2000年 Python 2.0 リリース
  - 2008年 Python 3.0 リリース 最新の安定版 3.9 (2020/10)

Java ： 1995年公開
PHP ： 1995年公開
RPG ： 1959年開発

- ライセンス形態
  - PSFL：Python Software Foundation License
    - GPL互換
    - 改変したプログラムの配布時に改変部分の公開は不要
    - PSFL 以外のライセンスの配布物と一緒に配布可能

# Python 概説

## 特徴

- インデント
  - 波括弧{} の代わりに コロン ： とインデント でブロックを表現
  - 行の終端にセミコロン ； は不要（改行で良い）
- インタープリター、マルチプラットフォーム
  - コンパイル不要
- pypi ：Python Package Index
  - 豊富なモジュール群を利用できる：268,148 project
  - Pip (Python Package Installer) によりインストール
- PEP：Python Enhancement Proposales
  - Python の決め事や標準化などが記されている
    - PEP8：Python コーディング規約

Python

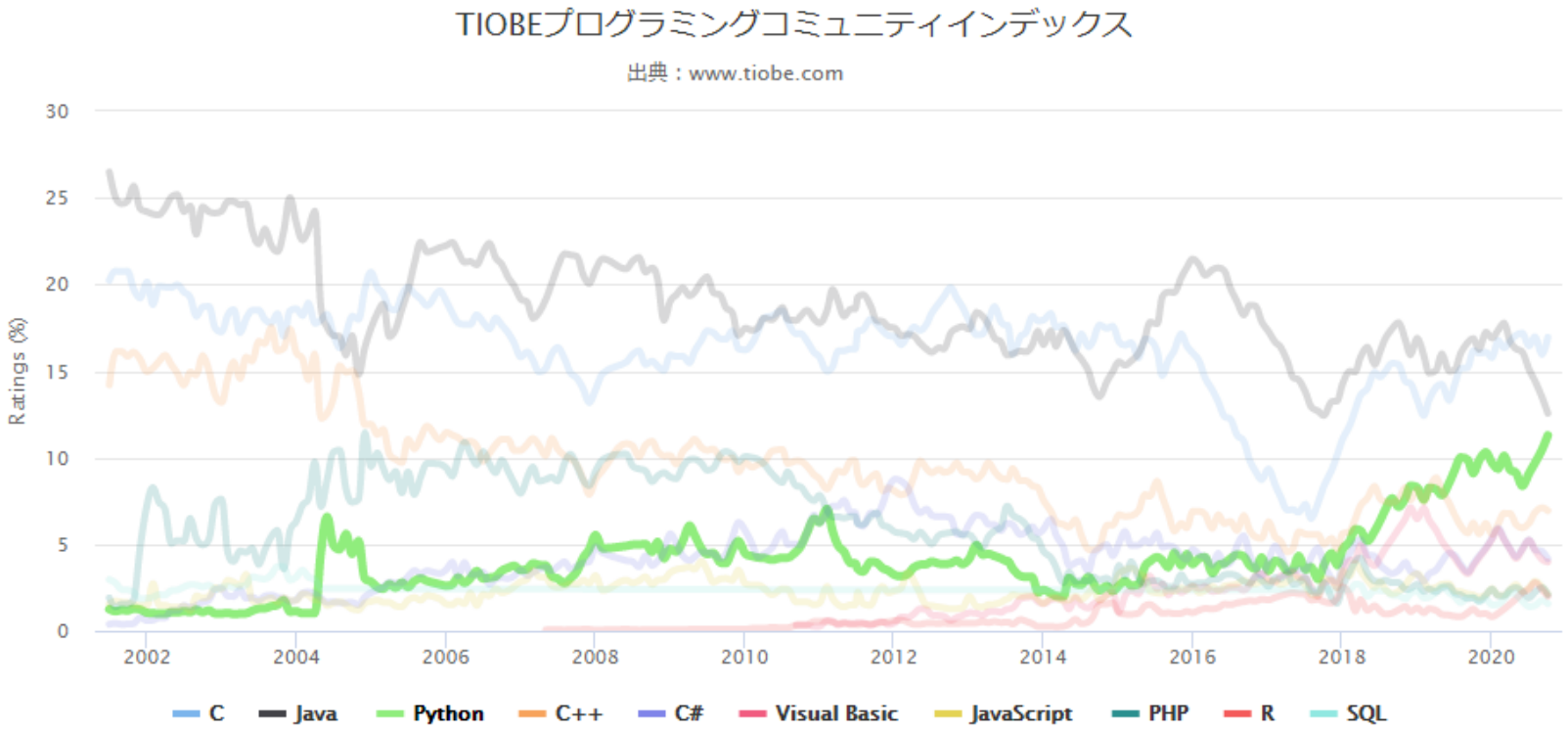
```python
if night:
    print("Hello Night World")
else:
    print("hello Morning Wrold")
```

PHP

```php
if(night){
    print("Hello Night World");
} else {
    print("Hello Morning World");
}
```

# Python 概説

## ■ 2020年10月 人気度：TIOBE




| Oct 2020 | Oct 2019 | Change | Programming Language | Ratings | Change |
|---|---|---|---|---|---|
| 1 | 2 | ▲ | C | 16.95% | +0.77% |
| 2 | 1 | ▼ | Java | 12.56% | -4.32% |
| **3** | **3** | | **Python** | **11.28%** | **+2.19%** |
| 4 | 4 | | C++ | 6.94% | +0.71% |

AI（機械学習，統計，分析）
での利用が起因？
- 第三次 AIブーム（2006年?～）
- 非プログラマーによる利用

・習得の容易性
・豊富なライブラリー
・活用範囲の広さ

2018年ごろから急激な増加
→ Java に変わる言語として注目か

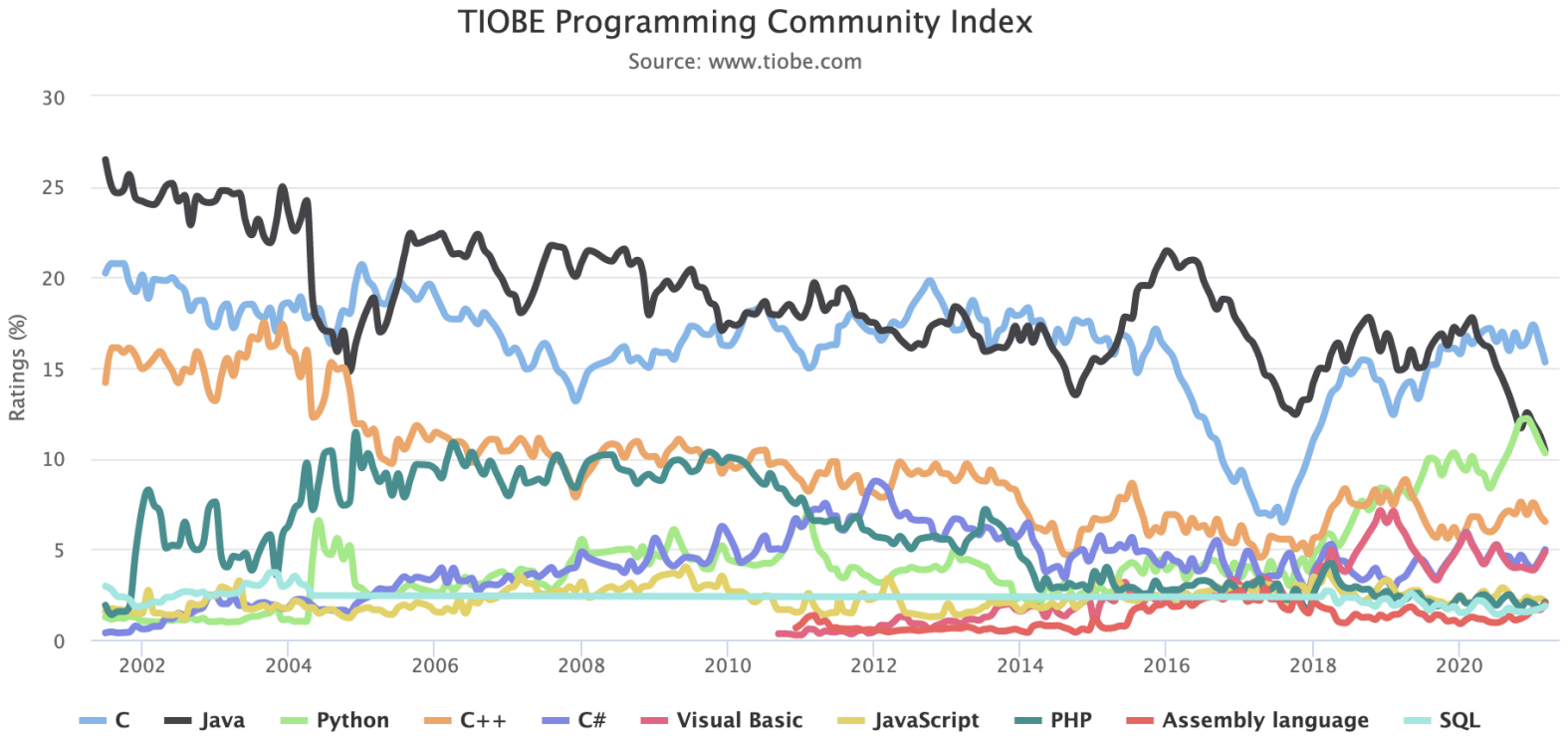
※1) TIOBE のソース
条件を満たした 25の検索エンジンのヒット数
(詳細は以下のリンクを参照)

https://www.tiobe.com/tiobe-index/

https://www.tiobe.com/tiobe-index/programming-languages-definition/

# Python 概説

## 2021年3月 人気度：TIOBE




| Mar 2021 | Mar 2020 | Change | Programming Language | Ratings | Change |
|---|---|---|---|---|---|
| 1 | 2 | ^ | C | 15.33% | -1.00% |
| 2 | 1 | v | Java | 10.45% | -7.33% |
| 3 | 3 | | Python | 10.31% | +0.20% |
| 4 | 4 | | C++ | 6.52% | -0.27% |

AI（機械学習，統計，分析）
での利用が起因？
- 第三次 AIブーム（2006年?～）
- 非プログラマーによる利用

・習得の容易性
・豊富なライブラリー
・活用範囲の広さ

2018年ごろから急激な増加
→ Java に変わる言語として注目か

※1) TIOBE のソース
条件を満たした 25の検索エンジンのヒット数
(詳細は以下のリンクを参照)

https://www.tiobe.com/tiobe-index/

https://www.tiobe.com/tiobe-index/programming-languages-definition/

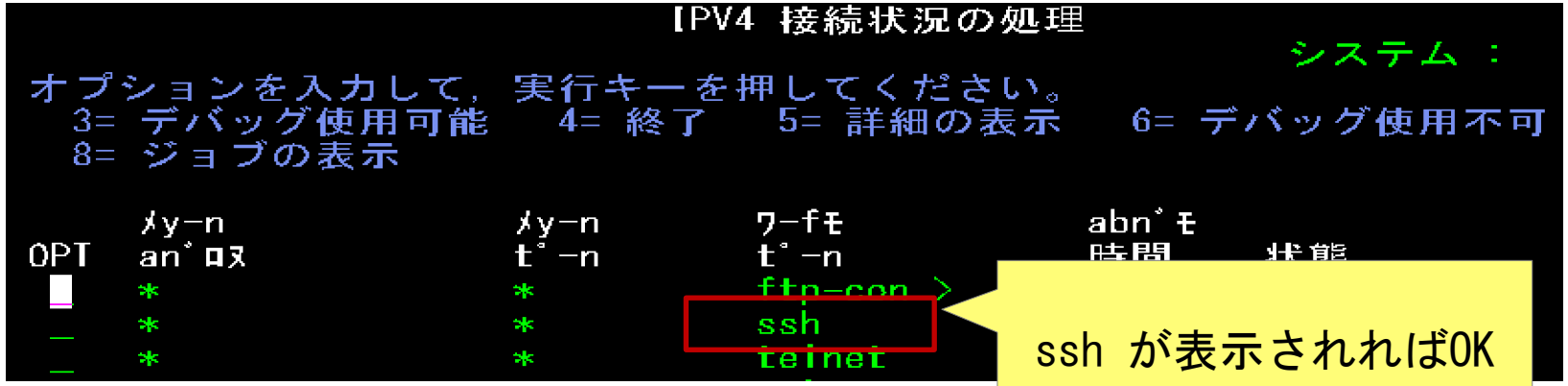
# Python のインストール

# インストール

- 流れ
  1. sshd の開始
  2. yum を導入
  3. Python の導入

# インストール：1. sshd の開始

- 5250 を利用
  - 5250エミュレーターで以下の CL コマンドを実行
    - STRTCPSVR SERVER(*SSHD)
  - 開始は以下 の CL コマンドで確認
    - NETSTAT OPTION(*CNN)



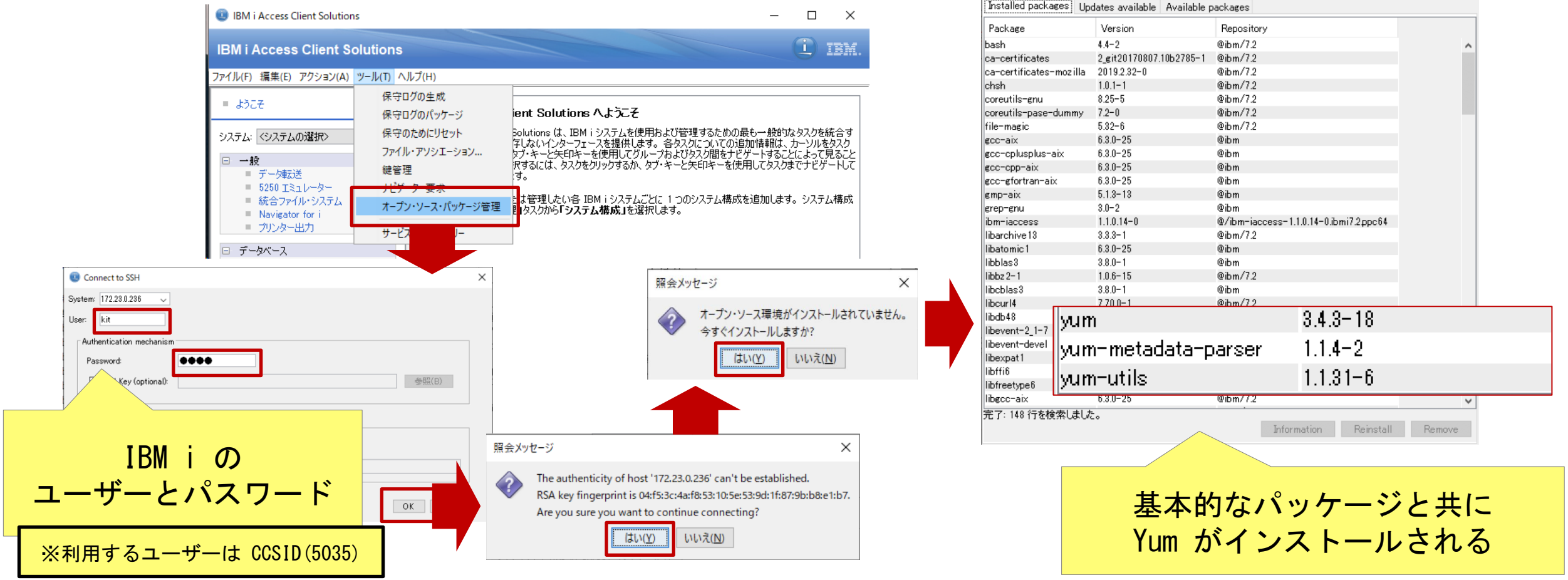

  - 停止は以下 の CL コマンドを実行
    - ENDTCPSVR SERVER(*SSHD)

# 参考： STRTCPSVR で開始できない場合

- QCCSID が 65535 の場合、開始できない？
  - 以下の手順で開始が可能
    1. CHGJOB CCSID(5035)
    2. SBMJOB CMD(CALL PGM(QP2SHELL) PARM('/usr/sbin/sshd')) JOBQ(QUSRNOMAX)
    3. CHGJOB CCSID(65535)
  - 開始の確認や停止方法は STRTCPSVR の時と同じ

# インストール：2. yum の導入

## ■ ACS オープンソース管理 から導入




IBM i の
ユーザーとパスワード

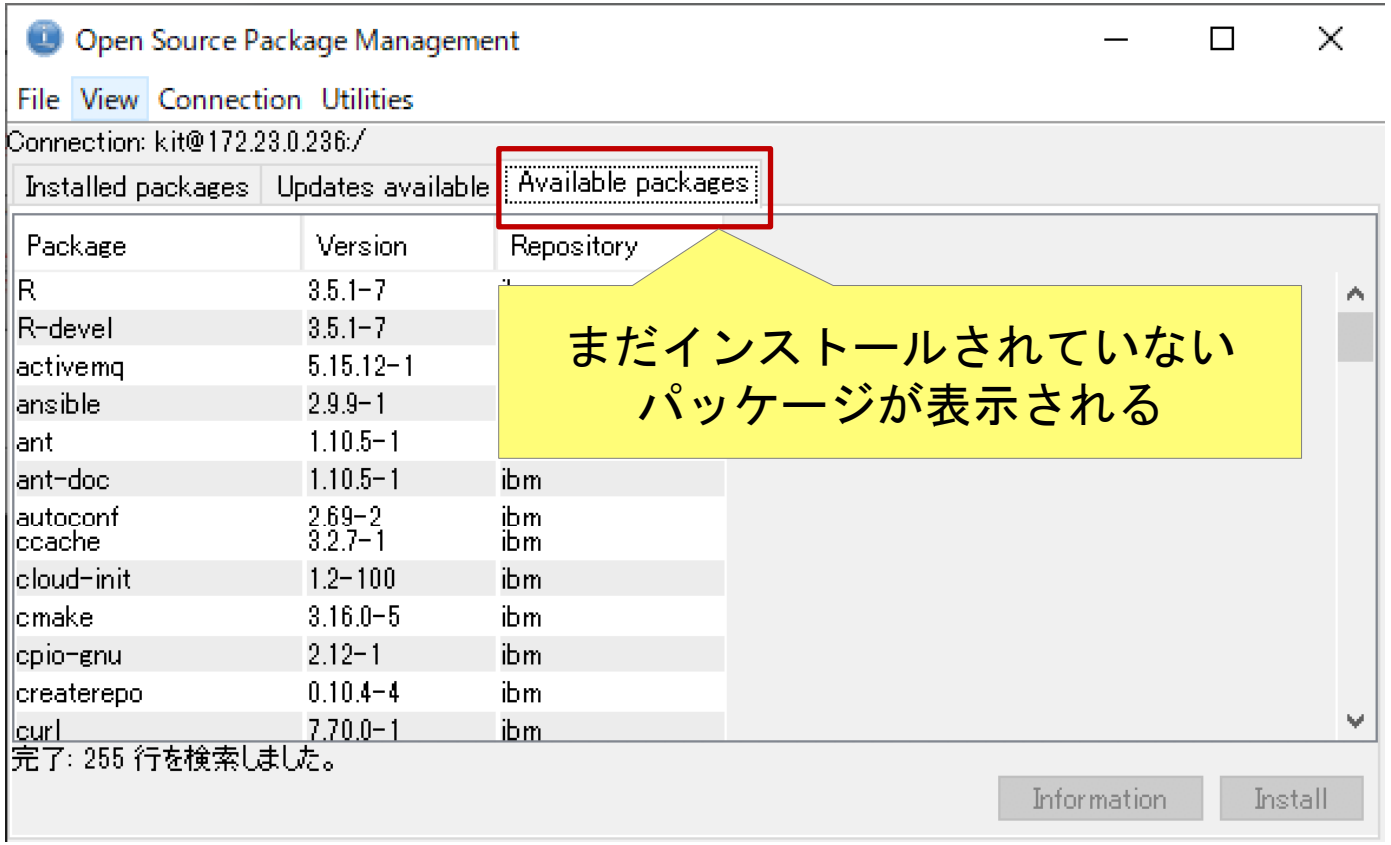
※利用するユーザーは CCSID(5035)

基本的なパッケージと共に
Yum がインストールされる

※ 詳細は yum 分科会の発表資料をご参照ください。 https://i5php.jp/wp-content/uploads/2019/09/yum_guide.pdf

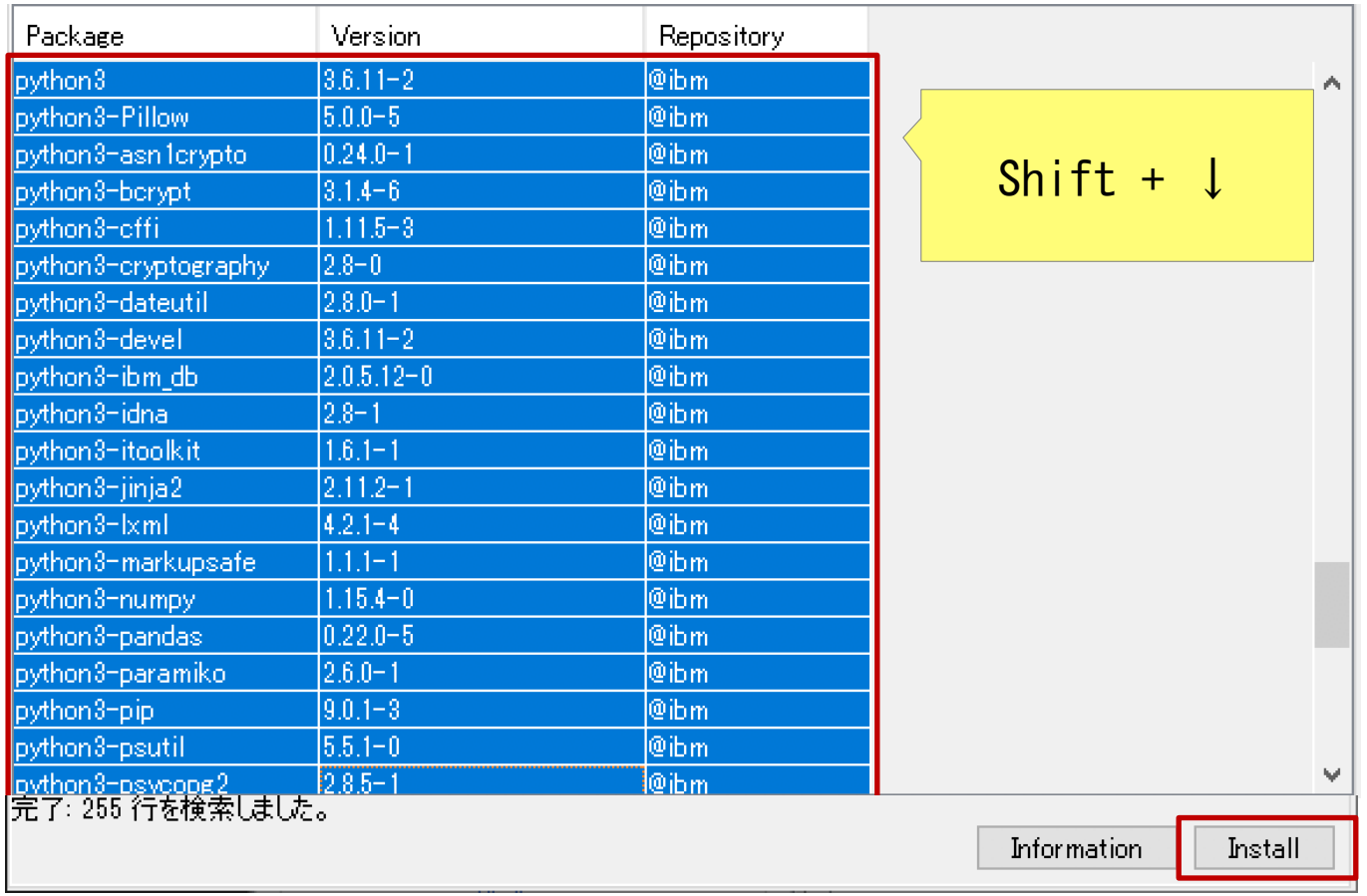
# インストール：3. Python の導入 1

- ACS オープンソース管理 から導入
  1. Open Source Package Management の「Available Packages」タブを開く



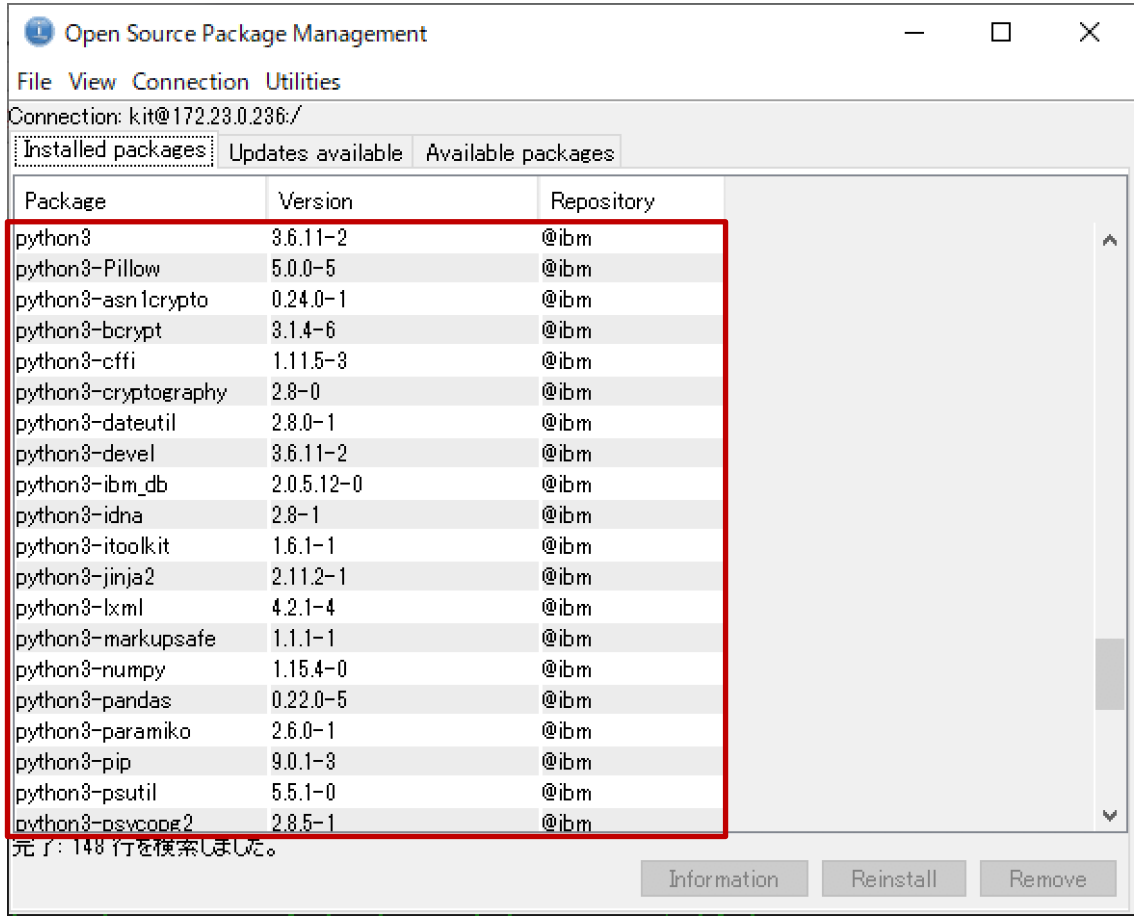
# インストール：3. Python の導入 2

- ACS オープンソース管理 から導入
  2. Python3 で始まるパッケージを全て選択し「Install」



# インストール：3. Python の導入 2

- ACS オープンソース管理 から導入
  3. Install package タブに表示されればOK




yum コマンド の場合
$ yum install python3*

# 参考：パッケージのインストールについて

- IBM i でのパッケージインストールは２種類
  - パッケージの**実装方法によりインストール方法に違い**がある
    - 100% python で実装されているパッケージ
    - C言語など、python 以外の言語で実装されているパッケージ
  - 100% **python で実装**されているパッケージのインストール
    - 基本的に **pip** でインストールが可能
    - https://pypi.org/
  - C言語など、**python　以外の言語で実装**されているパッケージ
    - 基本的に **yum**　でインストールが可能（リポジトリで提供されている場合）

| | | |
|---|---|---|
| python3-Pillow | 5.0.0-5 | @ibm |
| python3-asn1crypto | 0.24.0-1 | @ibm |
| python3-bcrypt | 3.1.4-6 | @ibm |
| python3-cffi | 1.11.5-3 | @ibm |
| python3-cryptography | 2.8-0 | @ibm |
| python3-dateutil | 2.8.0-1 | @ibm |
| python3-devel | 3.6.12-1 | @ibm |
| python3-ibm_db | 2.0.5.12-0 | @ibm |
| python3-idna | 2.8-1 | @ibm |
| python3-itoolkit | 1.6.1-1 | @ibm |

| | | |
|---|---|---|
| python3-jinja2 | 2.11.2-1 | @ibm |
| python3-lxml | 4.2.1-4 | @ibm |
| python3-markupsafe | 1.1.1-1 | @ibm |
| python3-numpy | 1.15.4-0 | @ibm |
| python3-pandas | 0.22.0-5 | @ibm |
| python3-paramiko | 2.6.0-1 | @ibm |
| python3-pip | 9.0.1-3 | @ibm |
| python3-psutil | 5.5.1-0 | @ibm |
| python3-psycopg2 | 2.8.5-1 | @ibm |
| python3-pycparser | 2.19-2 | @ibm |

| | | |
|---|---|---|
| python3-pynacl | 1.2.1-4 | @ibm |
| python3-pyodbc | 4.0.27-0 | @ibm |
| python3-pytz | 2018.5-3 | @ibm |
| python3-pyyaml | 5.3.1-1 | @ibm |
| python3-pyzmq | 17.1.2-0 | @ibm |
| python3-rpm | 4.13.1-7 | @ibm |
| python3-scikit-learn | 0.19.1-7 | @ibm |
| python3-scipy | 1.1.0-1 | @ibm |
| python3-setuptools | 36.0.1-3 | @ibm |
| python3-six | 1.10.0-1 | @ibm |
| python3-tkinter | 3.6.12-1 | @ibm |
| python3-wheel | 0.29.0-3 | @ibm |

# 参考：パッケージのインストールについて

## ■ 2021年3月時点： yum で提供されているパッケージ

| | | |
|---|---|---|
| python3-Pillow | 5.0.0-5 | @ibm |
| python3-asn1crypto | 0.24.0-1 | @ibm |
| python3-bcrypt | 3.1.4-6 | @ibm |
| python3-cffi | 1.11.5-3 | @ibm |
| python3-cryptography | 2.8-0 | @ibm |
| python3-dateutil | 2.8.0-1 | @ibm |
| python3-devel | 3.6.12-1 | @ibm |
| python3-ibm_db | 2.0.5.12-0 | @ibm |
| python3-idna | 2.8-1 | @ibm |
| python3-itoolkit | 1.6.1-1 | @ibm |
| python3-jinja2 | 2.11.2-1 | @ibm |
| python3-lxml | 4.2.1-4 | @ibm |
| python3-markupsafe | 1.1.1-1 | @ibm |
| python3-numpy | 1.15.4-0 | @ibm |
| python3-pandas | 0.22.0-5 | @ibm |
| python3-paramiko | 2.6.0-1 | @ibm |
| python3-pip | 9.0.1-3 | @ibm |
| python3-psutil | 5.5.1-0 | @ibm |
| python3-psycopg2 | 2.8.5-1 | @ibm |
| python3-pycparser | 2.19-2 | @ibm |
| python3-pynacl | 1.2.1-4 | @ibm |
| python3-pyodbc | 4.0.27-0 | @ibm |
| python3-pytz | 2018.5-3 | @ibm |
| python3-pyyaml | 5.3.1-1 | @ibm |
| python3-pyzmq | 17.1.2-0 | @ibm |
| python3-rpm | 4.13.1-7 | @ibm |
| python3-scikit-learn | 0.19.1-7 | @ibm |
| python3-scipy | 1.1.0-1 | @ibm |
| python3-setuptools | 36.0.1-3 | @ibm |
| python3-six | 1.10.0-1 | @ibm |
| python3-tkinter | 3.6.12-1 | @ibm |
| python3-wheel | 0.29.0-3 | @ibm |

pillow：画像処理

bcrypt：ハッシュ関数

cryptography：暗号化

devel：python 開発に必要なヘッダーやライブラリー

idna：コーデック

jinja2：テンプレートエンジン

lxml：XML や HTML のエスケープ

pandas：データ解析の支援機能

pip：パッケージインストーラー

psycopg2：postgresql の利用

pynacl：暗号化,復号化,署名

pytz：タイムゾーン処理

pyzmq：ZeroMQ の利用

scikit-learn：機械学習

setuptools：python パッケージの作成

tkinter：GUI パッケージ

asn1crypto：ASN.1 構造の解析

cffi：C 言語のコードを呼び出す

dateutil：日付の処理の拡張

ibm_db：Db2 for i の利用

itoolkit：IBM i の資源を利用

lxml：XML や HTML を扱う

numpy：数値計算の拡張

paramiko：python で SSH 接続する

psutil：プロセスとシステムの監視

pycparser：C言語パーサー

pyodbc：ODBC の利用

pyyaml：YAML の取り扱い

rpm：RPM パッケージ管理

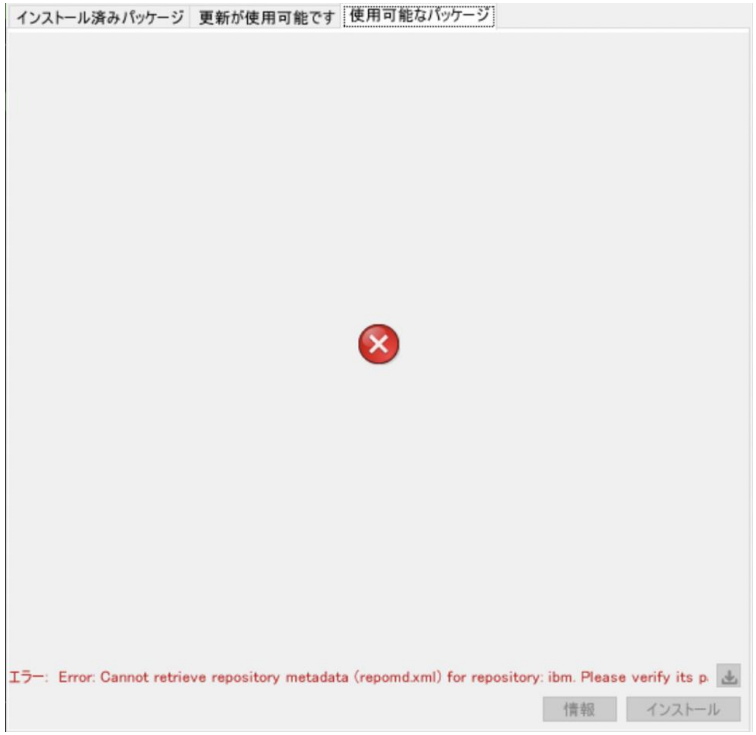
scipy：科学計算

six：python2 と python3 の互換性

wheel：python ライブラリーの標準パッケージ形式を作成

# 参考： 更新が可能です、使用可能なパッケージでエラー

- IBM i からリポジトリに直接アクセスできない場合
  - 以下が表示された場合
    - Error: Cannot retrieve repository metadata(repomd.xml) for repository: ibm...






■ オープンソース管理の画面から

1. メニューのユーティリティー→
   オフライン使用のためのリポジトリを複製...
2. 右画面の「リポジトリの複製」をクリック

※注意
ACS 経由でリポジトリのデータが流れます。
ネットワークの利用にご注意ください。

internet → ACS (PC) → IBM i

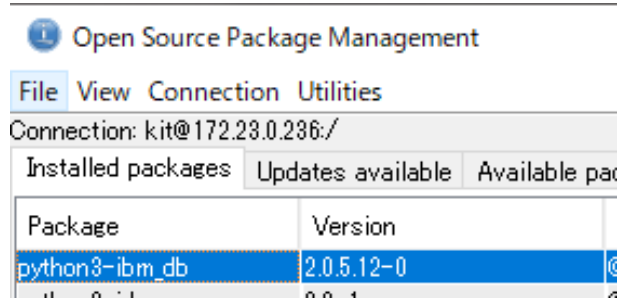
# データベース・アクセス１

# データベース・アクセス１

- Db2 for i へのアクセス方法は 2種類
  - **ibm_db**  今回はこちら
  - **pyODBC**

# データベース・アクセス１：ibm_db

- インストール
  - ACS の場合
    - ACS で python3-ibm_db を選択し Install
  - yum コマンドの場合
    - yum install python3-ibm_db




- 2種類のモジュールが含まれる
  - **ibm_db** ：ベース・モジュール
    - SQL 照会の発行、ストアード・プロシージャーの呼び出し等
  - **ibm_db_dbi** ： Python データアクセス仕様に準拠
    - PEP249：Python Database API Specification v2.0
    - ibm_db のラッパー

# データベース・アクセス１：ibm_db

- 2種類のモジュールが含まれる
  - **ibm_db** ：ベース・モジュール
    - https://github.com/ibmdb/python-ibmdb/wiki/APIs#ibm_dbclose
    - SQL 照会の発行、ストアード・プロシージャーの呼び出し等
  - **ibm_db_dbi** ： Python データアクセス仕様に準拠
    - PEP249：Python Database API Specification v2.0
      - https://www.python.org/dev/peps/pep-0249/#introduction
    - ibm_db のラッパー

# データベース・アクセス１：ibm_db_dbi モジュールの利用

## コーディングと shell からの実行

exdb01_app_cli.py

```python
import ibm_db_dbi as db
import io,sys
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')

conn = db.connect("DATABASE=*LOCAL;","KIT","<password>")
cursor = conn.cursor()
cursor.execute("SELECT * FROM QEOL.TOKMSP")
for r in cursor.fetchall():
    print(r[0], end=' ')
    print(r[1], end=' ')
    print(r[2])
```

$ python exdb01_app_cli.py

# データベース・アクセス１：ibm_db_dbi モジュールの利用

## ■ コードの解説

ibm_db_dbi
モジュールの読み込み

shell で実行する際、
標準出力を utf-8 に設定しないと
UnicodeEncodeError になる

```
1  import ibm_db_dbi as db
2  import io,sys
3  sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
4
5  conn = db.connect("DATABASE=*LOCAL;","KIT","<password>")
6  cursor = conn.cursor()
7  cursor.execute("SELECT * FROM QEOL.TOKMSP")
8  for r in cursor.fetchall():
9      print(r[0], end=' ')
10     print(r[1], end=' ')
11     print(r[2])
```

IBM I との接続。リレーショナルデータベース名を指定

カーソルの作成
SQLの実行

フェッチし結果セット取得
※fetchall() の実行で全件が list で取り出される
ループ毎にレコードを取得
レコード1件毎に出力

# データベース・アクセス１：ibm_db

- 2種類のモジュールが含まれる
  - **ibm_db** ：ベース・モジュール
    - https://github.com/ibmdb/python-ibmdb/wiki/APIs#ibm_dbclose
    - SQL 照会の発行、ストアード・プロシージャーの呼び出し等
  - **ibm_db_dbi** ： Python データアクセス仕様に準拠
    - PEP249：Python Database API Specification v2.0
      - https://www.python.org/dev/peps/pep-0249/#introduction
    - ibm_db のラッパー

# データベース・アクセス１：ibm_db モジュールの利用

## コーディングと shell からの実行

exdb02_app_cli.py

```
1  import ibm_db as db
2  import io,sys
3  sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
4
5  conn = db.connect("*LOCAL", "KIT", "<password>")
6  stmt = db.exec_immediate(conn, "SELECT * FROM QEOL.TOKMSP")
7  rs = db.fetch_both(stmt)
8  while(rs):
9      print(rs[0], end=' ')
10     print(rs[1], end=' ')
11     print(rs[2])
12     rs = db.fetch_both(stmt)
13
```

$ python exdb02_app_cli.py

# データベース・アクセス１：ibm_db モジュールの利用

## ■ コードの解説

```
1   import ibm_db as db
2   import io,sys
3   sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
4
5   conn = db.connect("*LOCAL", "KIT", "<password>")
6   stmt = db.exec_immediate(conn, "SELECT * FROM QEOL.TOKMSP")
7   rs = db.fetch_both(stmt)
8   while(rs):
9       print(rs[0], end=' ')
10      print(rs[1], end=' ')
11      print(rs[2])
12      rs = db.fetch_both(stmt)
13
```

ibm_db
モジュールの読み込み

IBM I との接続。リレーショナルデータベース名を指定

SQLの実行
ステートメントハンドルの取得
フェッチし、先頭レコードを取得

レコード1件毎に list で出力
フェッチし、次のレコードを取得

# データベース・アクセス１：ibm_db と ibm_db_dbi の比較

- レコードの**読み込み速度**を比較
  - 10万件
  - 100フィールド（10A　X 100）
- 利用したソース

結果は環境に依存します

**ibm_db_dbi**

```
1  import ibm_db_dbi as db
2  import time
3  import io,sys
4  sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
5  start = time.time()
6
7  conn = db.connect("DATABASE=*LOCAL;", "KIT", "<password>")
8  cursor = conn.cursor()
9  cursor.execute("SELECT * FROM KIT.TESTPF")
10 rs = cursor.fetchone()
11 mid_time = time.time() - start
12 print ("mid_time:{0}".format(mid_time) + "[sec]")
13 while(rs):
14     rs = cursor.fetchone()
15
16 elapsed_time = time.time() - start
17 print ("elapsed_time:{0}".format(elapsed_time) + "[sec]")
```

**ibm_db**

```
1  import ibm_db as db
2  import time
3  import io,sys
4  sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
5  start = time.time()
6
7  conn = db.connect("*LOCAL", "KIT", "<password>")
8  stmt = db.exec_immediate(conn, "SELECT * FROM KIT.TESTPF")
9  rs = db.fetch_both(stmt)
10 mid_time = time.time() - start
11 print ("mid_time:{0}".format(mid_time) + "[sec]")
12 while(rs):
13     rs = db.fetch_both(stmt)
14
15 elapsed_time = time.time() - start
16 print ("elapsed_time:{0}".format(elapsed_time) + "[sec]")
```

※　ロジックを合わせるため、fetchone() を利用

# データベース･アクセス１：ibm_db と ibm_db_dbi の比較

- レコードの読み込み速度を比較：結果

結果は環境に依存します

| | ibm_db_dbi | ibm_db |
|---|---|---|
| 1回目 | 297.63 sec | 157.92 sec |
| 2回目 | 309.14 sec | 158.18 sec |
| 3回目 | 307.51 sec | 157.95 sec |
| 3回の平均 | 304.76 sec | 158.01 sec |

- ibm_db_dbi の存在意義は？
  - PEP249 への準拠
    - 接続先のDBを意識しない
    - 移植性の高いコード（DB接続、SQLの実行、結果の取得）を実現
  - 他のDB製品ではこのタイプのみ

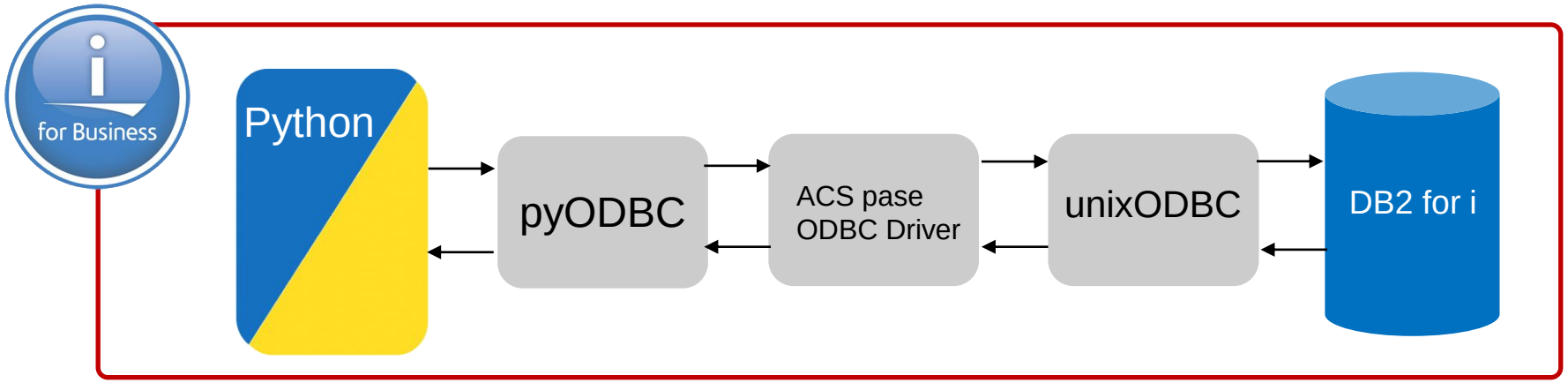
→ **ibm_db は IBM i の強みとして理解**

# データベース・アクセス２

# データベース・アクセス２

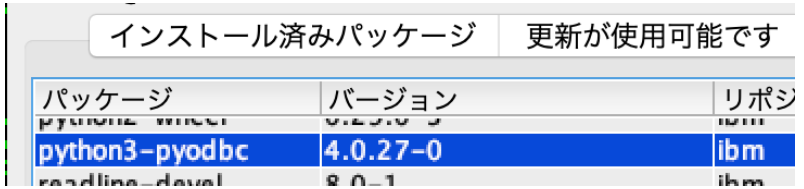
- Db2 for i へのアクセス方法は 2種類
  - **ibm_db**
  - **pyODBC** ← 今回はこちら

- pyODBC
  - Python, PHP, Node.js など IBM i のみで ODBCを利用したアクセスが可能となる
  - 構成イメージ



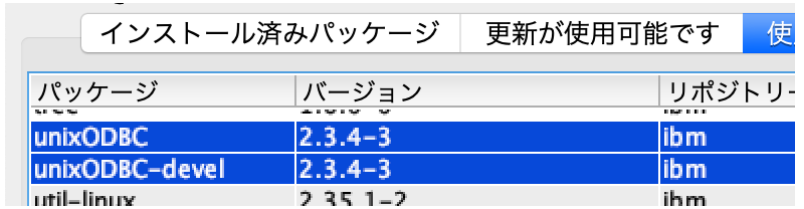
# データベース･アクセス２：pyodbc

## pyODBC のインストール

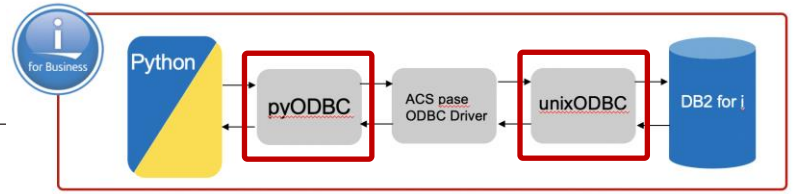



- ACS の場合
  - ACS で python3-pyodbc を選択し Install
- yum コマンドの場合
  - yum install python3-pyodbc

## unixODBC のインストール




- ACS の場合
  - ACS で python3-unixODBC, unixODBC-devel を選択し Install
- yum コマンドの場合
  - yum install python3-unixODBC unixODBC-devel



# データベース・アクセス２：pyodbc

- ACS ODBC ドライバー のインストール
  1. IBM i Access – Client Solutions ページへアクセス
     - https://www.ibm.com/support/pages/ibm-i-access-client-solutions
  2. Download for IBM i Access Client Solutions をクリック

IBM i Access - Client Solutions

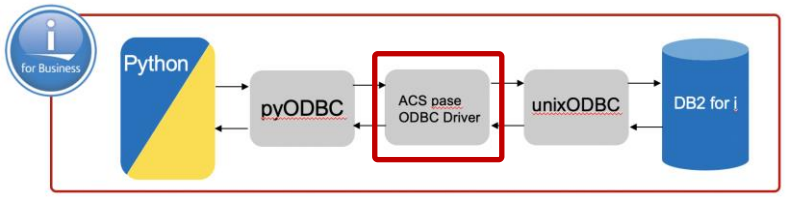
IBM i Access

Overview　Client Solutions　Web/Mobile　Windows

IBM i Access Client Solutions provides a Java based, platform-independent interface that runs on most operating systems that support Java, including Linux, Mac, and Windows™. IBM i Access Client Solutions consolidates the most commonly used tasks for managing your IBM i into one simplified location. The latest version of IBM i Access Client Solutions is available to customers with an IBM i software maintenance contract.

→ Downloads for IBM i Access Client Solutions

→ QuickStartGuide

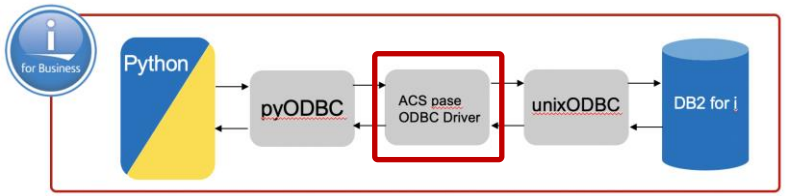


# データベース・アクセス２：pyodbc

- ACS ODBC ドライバー のインストール（続き）

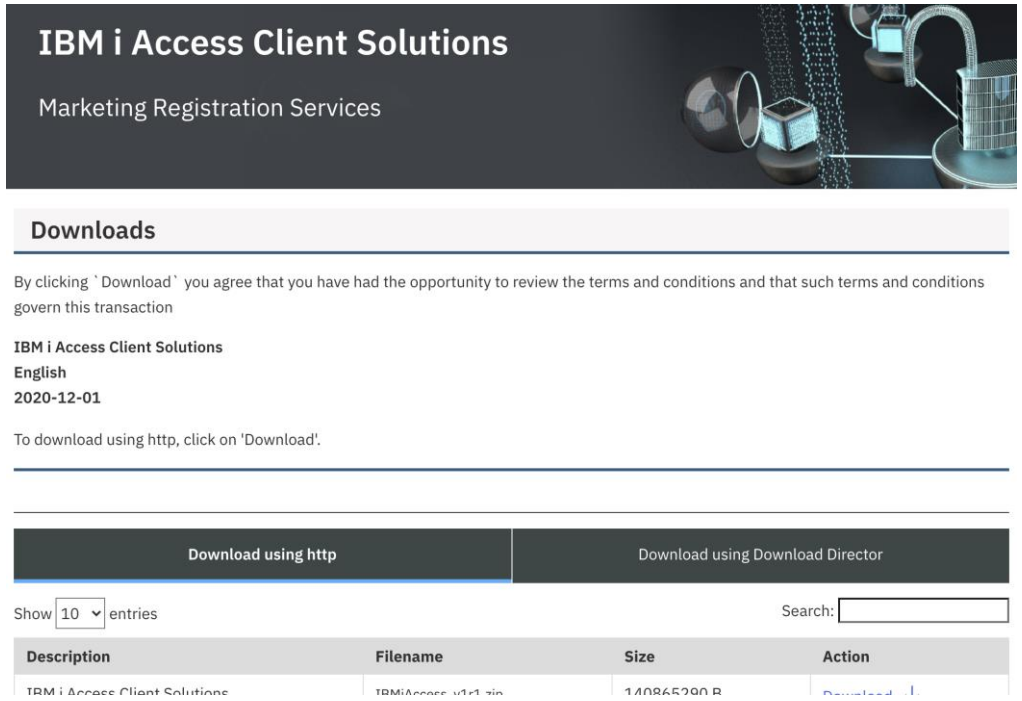
3. IBMid でログイン


4. ライセンス条項に同意し I confirm

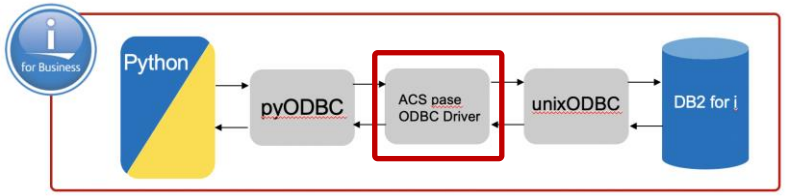



# データベース・アクセス２：pyodbc

- ACS ODBC ドライバー のインストール（続き）
  5. Downloads ページから “**ACS PASE App Pkg**” を Downlaod



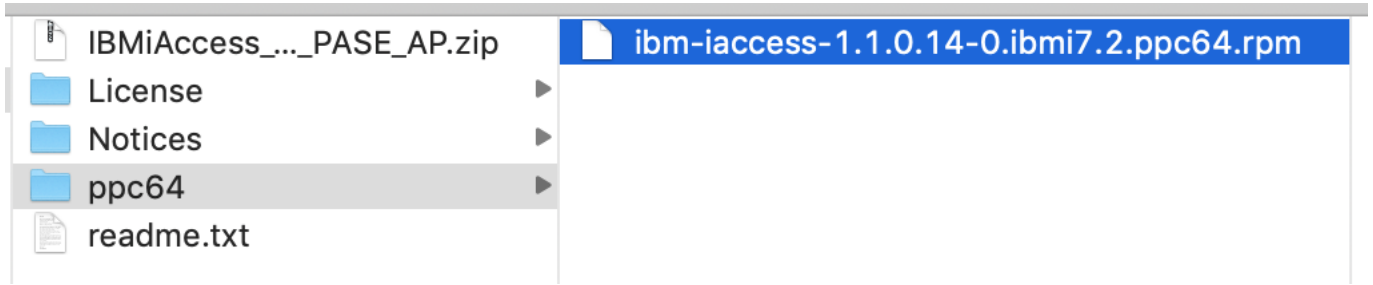




# データベース・アクセス２：pyodbc

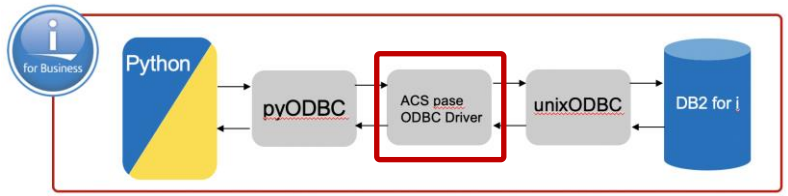
- ACS ODBC ドライバー のインストール（続き）

6. ダウンロードした以下を解凍
   - IBMiAccess_v1r1_PASE_AP.zip




7. 解凍した以下を FTP で IBM i に転送
   - ibm-iaccess-1.1.0.14-0.ibmi7.2.ppc64.rpm

     ※ bin (binary) で転送すること



# データベース・アクセス２：pyodbc

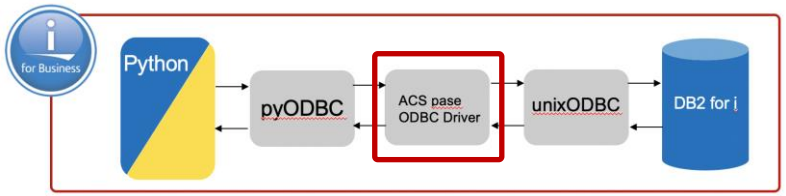
- ACS ODBC ドライバー のインストール（続き）

## 8. yum で　ACS PASE ODBC Driver をインストール

```
yum install /home/KIT/acs_odbc_driver/ibm-iaccess-1.1.0.14-0.ibmi7.2.ppc64.rpm
```

※ /home/KIT/acs_odbc_driver は ibm-iaccess-1.1.0.14-0.ibmi.7.2.ppc64.rpm の配置例

```
bash-4.4$ yum install /home/KIT/acs_odbc_driver/ibm-iaccess-1.1.0.14-0.ibmi7.2.ppc64.rpm
Setting up Install Process
Examining /home/KIT/acs_odbc_driver/ibm-iaccess-1.1.0.14-0.ibmi7.2.ppc64.rpm: ibm-iaccess-1.1.0.14-0.ppc64
Marking /home/KIT/acs_odbc_driver/ibm-iaccess-1.1.0.14-0.ibmi7.2.ppc64.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package ibm-iaccess.ppc64 0:1.1.0.14-0 will be installed
--> Processing Dependency: /QOpenSys/pkgs/bin/odbcinst for package: ibm-iaccess-1.1.0.14-0.ppc64
--> Running transaction check
---> Package unixODBC.ppc64 0:2.3.4-3 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package                Arch              Version                 Repository                                  Size
================================================================================
Installing:
 ibm-iaccess            ppc64             1.1.0.14-0              /ibm-iaccess-1.1.0.14-0.ibmi7.2.ppc64       41 M
Installing for dependencies:
 unixODBC               ppc64             2.3.4-3                 ibm                                        204 k

Transaction Summary
================================================================================
Install       2 Packages

Total size: 41 M
Installed size: 42 M
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : unixODBC-2.3.4-3.ppc64                                      1/2
  Installing : ibm-iaccess-1.1.0.14-0.ppc64                                2/2
grep: can't open /QOpenSys/etc/odbc.ini

Installed:
  ibm-iaccess.ppc64 0:1.1.0.14-0

Dependency Installed:
  unixODBC.ppc64 0:2.3.4-3

Complete!
```



# データベース・アクセス２：pyODBC shell で実行してみた

## pyODBC の利用

exdb02_app_cli.py

```
1  import pyodbc as db
2  import io,sys
3  sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
4
5  conn = db.connect('DRIVER={IBM i Access ODBC Driver};SYSTEM=172.23.0.234;UID=KIT;PWD=<password>')
6  cursor = conn.cursor()
7  cursor.execute("SELECT * FROM QEOL.TOKMSP")
8  for r in cursor.fetchall():
9      print(r[0], end=' ')
10     print(r[1], end=' ')
11     print(r[2])
```

$ python exdb02_app_cli.py

# 参考：ODBC の設定

- ODBC の設定をプログラム外で行う場合
  - odbcinst -j コマンドで参照ファイル、パスの確認が可能

```
bash-4.4$ odbcinst -j
unixODBC 2.3.4
DRIVERS............: /QOpenSys/etc/odbcinst.ini
SYSTEM DATA SOURCES: /QOpenSys/etc/odbc.ini
FILE DATA SOURCES..: /QOpenSys/etc/ODBCDataSources
USER DATA SOURCES..: /home/KIT/.odbc.ini
SQLULEN Size.......: 8
SQLLEN Size........: 8
SQLSETPOSIROW Size.: 8
```

  - デフォルト

**odbc.ini** (接続情報)

```
bash-4.4$ cat odbc.ini

### IBM provided DSN - do not remove this line ###
[*LOCAL]
Description = Default IBM i local database
Driver      = IBM i Access ODBC Driver
System      = localhost
UserID      = *CURRENT
### Start of DSN customization
### End of DSN customization
### IBM provided DSN - do not remove this line ###
```

**odbcinst.ini** (ドライバー情報)

```
bash-4.4$ cat odbcinst.ini
[IBM i Access ODBC Driver]
Description=IBM i Access ODBC Driver
Driver=/QOpenSys/pkgs/lib/libcwbodbc.so
Threading=0
DontDLClose=1
UsageCount=1
```

python での利用

```
#conn = db.connect('DRIVER={IBM i Access ODBC Driver};SY
conn = db.connect('DSN=*LOCAL')
```

# データベース・アクセス２：pyODBC shell で実行してみた

## ■ 解説

PEP249 への準拠

pyODBC
モジュールの読み込み

```
1   import pyodbc as db
2   import io,sys
3   sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
4
5   conn = db.connect('DRIVER={IBM i Access ODBC Driver};SYSTEM=172.23.0.234;UID=KIT;PWD=<password>')
6   cursor = conn.cursor()
7   cursor.execute("SELECT * FROM QEOL.TOKMSP")
8   for r in cursor.fetchall():
9       print(r[0], end=' ')
10      print(r[1], end=' ')
11      print(r[2])
```

IBM i との接続
ODBC DNS文字列を使用

カーソル作成と
SQLの実行

フェッチし結果セット取得
※fetchall() の実行で全件が list で取り出される
ループ毎にレコードを取得
レコード1件毎に出力

# データベース・アクセス２：ibm_db_dbi と pyODBC の比較

- レコードの**読み込み速度**を比較
  - 10万件
  - 100フィールド（10A　X 100）
- 利用したソース

結果は環境に依存します

**ibm_db_dbi**

```
1  import ibm_db_dbi as db
2  import time
3  import io,sys
4  sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
5  start = time.time()
6
7  conn = db.connect("DATABASE=*LOCAL;", "KIT", "<password>")
8  cursor = conn.cursor()
9  cursor.execute("SELECT * FROM KIT.TESTPF")
10 rs = cursor.fetchone()
11 mid_time = time.time() - start
12 print ("mid_time:{0}".format(mid_time) + "[sec]")
13 while(rs):
14     rs = cursor.fetchone()
15
16 elapsed_time = time.time() - start
17 print ("elapsed_time:{0}".format(elapsed_time) + "[sec]")
```

※　ロジックを合わせるため、fetchone() を利用

**pyODBC**

```
1  import pyodbc as db
2  import time
3  import io,sys
4  sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
5  start = time.time()
6
7  conn = db.connect('DRIVER={IBM i Access ODBC Driver};SYSTEM=172.23.0.234;UID=KIT;PWD=<password>')
8  cursor = conn.cursor()
9  cursor.execute("SELECT * FROM KIT.TESTPF")
10 rs = cursor.fetchone()
11 mid_time = time.time() - start
12 print ("mid_time:{0}".format(mid_time) + "[sec]")
13 while(rs):
14     rs = cursor.fetchone()
15
16 elapsed_time = time.time() - start
17 print ("elapsed_time:{0}".format(elapsed_time) + "[sec]")
18
```

※　ロジックを合わせるため、fetchone() を利用

# データベース・アクセス２：ibm_db_dbi と pyODBC の比較

- レコードの読み込み速度を比較：結果

結果は環境に依存します

| | ibm_db_dbi | pyODBC |
|---|---|---|
| 1回目 | 297.63 sec | 447.38 sec |
| 2回目 | 309.14 sec | 443.40 sec |
| 3回目 | 307.51 sec | 439.19 sec |
| 3回の平均 | 304.76 sec | 443.32 sec |

- pyODBC の存在意義は？
  - ODBC の利用で他言語との移植性を高める
    - 他プラットフォームとの共通性
      - 例えば、「Windows, Linux で開発を行い IBM i へデプロイする」など
      - PEP249準拠
    - 他言語との共通性
      - java, php, node.js

# データベース・アクセス２：アクセス方法による違い

■ パフォーマンスの観点（※レコードの読み込み速度）

結果は環境に依存します


■ オープン性の観点


・IBM i only
・python only

・python only

# データの操作など

# データの操作など：ibm_db データの操作の概要

## ■前提

- データ操作対象のファイルがジャーナル開始されている事

## ■データ操作（登録,変更,削除）の流れ

1. IBM i への接続
2. SQLの事前コンパイル
3. 値の設定
4. SQLの実行
5. リソースの解放

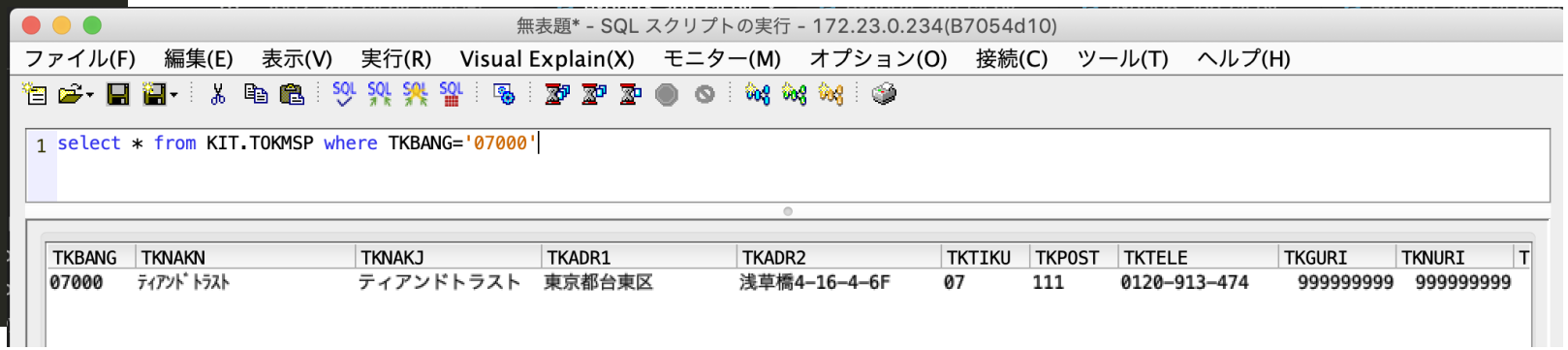
# データの操作など：ibm_db データの登録

## ■データの登録：コーディングと shell からの実行

exdb03_app_cli.py

```
1  import ibm_db
2
3  conn = ibm_db.connect("*LOCAL","KIT","<password>")
4  sql = "insert into KIT.TOKMSP values( \r\n"\
5      "?, ?, ?, ?, ?, ?, ?, ?, ?, ?, \r\n"\
6      "?, ?, ?, ?, ? \r\n"\
7      ")"
8  stmt = ibm_db.prepare(conn, sql)
9
10 ibm_db.bind_param(stmt, 1, '07000')
11 ibm_db.bind_param(stmt, 2, 'ﾃｨｱﾝﾄﾞﾄﾗｽﾄ')
12 ibm_db.bind_param(stmt, 3, 'ティアンドトラスト')
13 ibm_db.bind_param(stmt, 4, '東京都台東区')
14 ibm_db.bind_param(stmt, 5, '浅草橋4-16-4-6F')
15 ibm_db.bind_param(stmt, 6, '07')
16 ibm_db.bind_param(stmt, 7, '111')
17 ibm_db.bind_param(stmt, 8, '0120-913-474')
18 ibm_db.bind_param(stmt, 9, 999999999)
19 ibm_db.bind_param(stmt, 10, 999999999)
20 ibm_db.bind_param(stmt, 11, 999999999)
21 ibm_db.bind_param(stmt, 12, 999999999)
22 ibm_db.bind_param(stmt, 13, 999999999)
23 ibm_db.bind_param(stmt, 14, 999999999)
24 ibm_db.bind_param(stmt, 15, '9')
25
26 ibm_db.execute(stmt)
27
28 ibm_db.free_result(stmt)
29 ibm_db.close(conn)
```

$ python exdb02_app_cli.py



# データの操作など：ibm_db データの登録

## ■データの登録：コードの解説

```
1  import ibm_db
2
3  conn = ibm_db.connect("*LOCAL","KIT","<password>")
4  sql = "insert into KIT.TOKMSP values( \r\n"\
5      "?, ?, ?, ?, ?, ?, ?, ?, ?, ?, \r\n"\
6      "?, ?, ?, ?, ? \r\n"\
7      ")"
8  stmt = ibm_db.prepare(conn, sql)
9
10 ibm_db.bind_param(stmt, 1, '07000')
11 ibm_db.bind_param(stmt, 2, 'ﾃｨｱﾝﾄﾞﾄﾗｽﾄ')
12 ibm_db.bind_param(stmt, 3, 'ティアンドトラスト')
13 ibm_db.bind_param(stmt, 4, '東京都台東区')
14 ibm_db.bind_param(stmt, 5, '浅草橋4-16-4-6F')
15 ibm_db.bind_param(stmt, 6, '07')
16 ibm_db.bind_param(stmt, 7, '111')
17 ibm_db.bind_param(stmt, 8, '0120-913-474')
18 ibm_db.bind_param(stmt, 9, 999999999)
19 ibm_db.bind_param(stmt, 10, 999999999)
20 ibm_db.bind_param(stmt, 11, 999999999)
21 ibm_db.bind_param(stmt, 12, 999999999)
22 ibm_db.bind_param(stmt, 13, 999999999)
23 ibm_db.bind_param(stmt, 14, 999999999)
24 ibm_db.bind_param(stmt, 15, '9')
25
26 ibm_db.execute(stmt)
27
28 ibm_db.free_result(stmt)
29 ibm_db.close(conn)
```

SQL文（INSERT）の作成
※ パラメータマーカーを設定

SQL文の事前コンパイル（prepare）

各項目の登録する値を設定
(bind_parm)

SQLの実行

リソースの解放(free_result, close)

# データの操作など：ibm_db データの更新、削除

## ■データの更新

```
1  import ibm_db
2
3  conn = ibm_db.connect("*LOCAL","KIT","<password>")
4  sql = "update KIT.TOKMSP set TKTELE=? where TKBANG=?"
5  stmt = ibm_db.prepare(conn, sql)
6
7  ibm_db.bind_param(stmt, 1, '03-5821-3666')
8  ibm_db.bind_param(stmt, 2, '07000')
9  ibm_db.execute(stmt)
10
11 ibm_db.free_result(stmt)
12 ibm_db.close(conn)
13
```

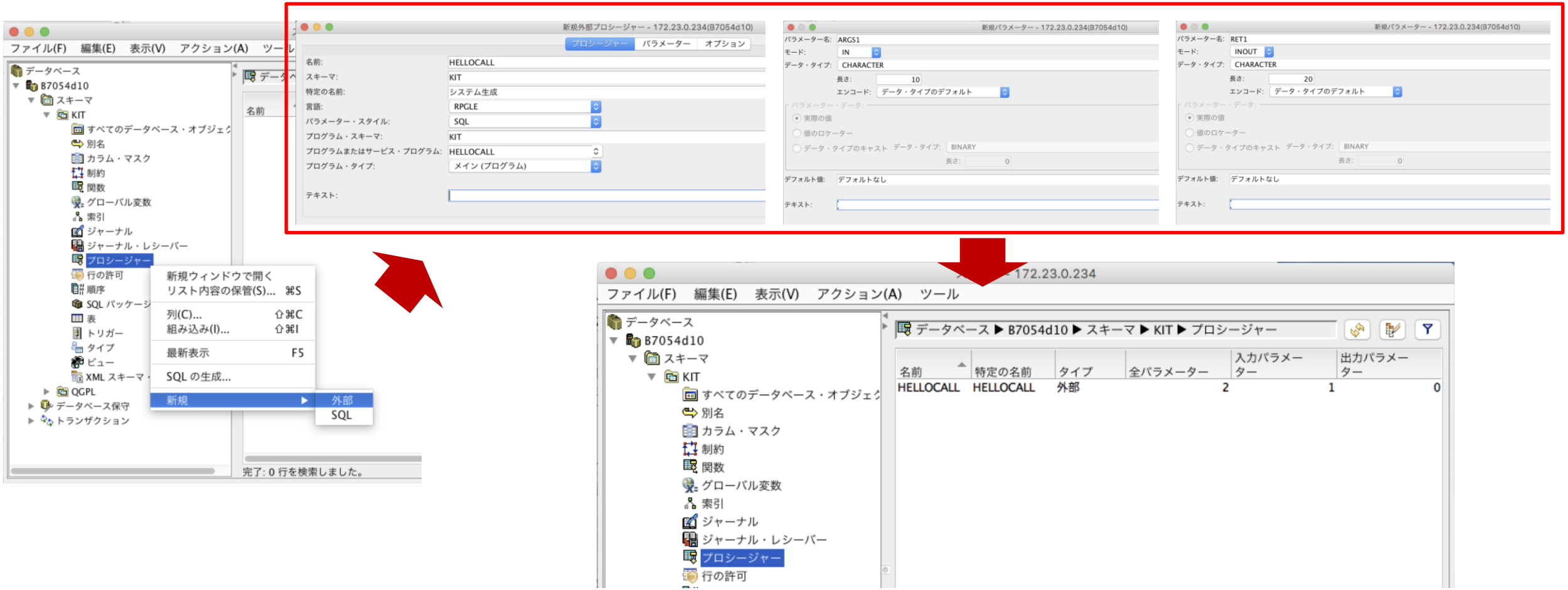
## ■データの削除

```
1  import ibm_db
2
3  conn = ibm_db.connect("*LOCAL","KIT","<password>")
4  sql = "delete from KIT.TOKMSP where TKBANG=?"
5  stmt = ibm_db.prepare(conn, sql)
6
7  ibm_db.bind_param(stmt, 1, '07000')
8  ibm_db.execute(stmt)
9
10 ibm_db.free_result(stmt)
11 ibm_db.close(conn)
12
```

# データの操作など：ibm_db RPG ストアドプロシージャの実行

## ■前提

- 実行対象のプログラムがストアドプロシージャとして登録されている事

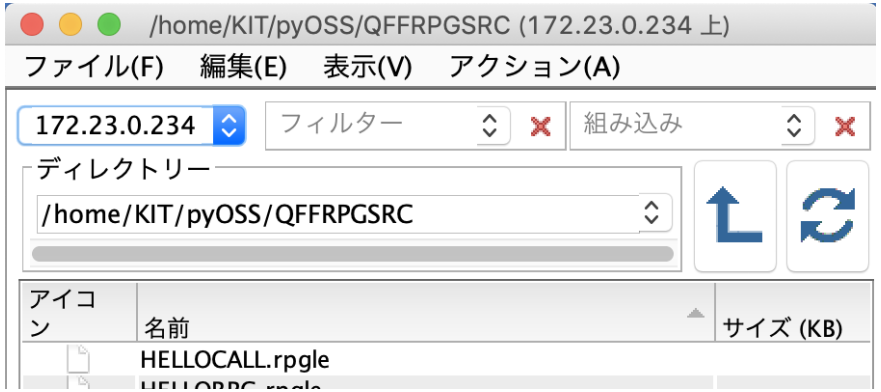


# データの操作など：ibm_db RPG ストアドプロシージャの実行

## ■ 参考：RPGソース

HELLOCALL.rpgle

```
QFFRPGSRC > ≡ HELLOCALL.rpgle
  1    **free
  2    // Parameters
  3    dcl-pi hellocall;
  4      dcl-parm args1 char(10);
  5      dcl-parm ret1 char(20);
  6    end-pi;
  7
  8    // Main Routine
  9    ret1 = 'CALL is ' + %trim(args1);
 10
 11    *inlr = *on;
 12    return;
```




IFS のファイルを指定可能

コンパイルで利用する CCSID を設定
※ コンパイラーがCCSID 1208
を認識できないため、5035を指定

CRTBNDRPG PGM(KIT/HELLORPG) SRCSTMF('/home/KIT/PYOSS/QFFRPGSRC/HELLORPG.rpgle') TGTCCSID(5035)

# データの操作など：ibm_db RPG ストアドプロシージャの実行

## ■プログラムの実行：コーディングと shell からの実行

exdb06_app_cli.py

```
1   import ibm_db
2   import io,sys
3   sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
4
5   conn = ibm_db.connect("*LOCAL","KIT","<password>")
6   parm1 = "北原"
7   ret1 = ""
8   stmt, parm1, ret1 = ibm_db.callproc(conn, "KIT.HELLOCALL", (parm1, ret1))
9
10  print("ret1 = " + ret1)
11
12  ibm_db.free_result(stmt)
13  ibm_db.close(conn)
```

$ python exdb06_app_cli.py

```
bash-4.4$ python exdb06_app_cli.py
parm2 = CALL is 北原
bash-4.4$
```

# データの操作など：ibm_db RPG ストアドプロシージャの実行

## ■プログラムの実行：コードの解説

```
1   import ibm_db
2   import io,sys
3   sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
4
5   conn = ibm_db.connect("*LOCAL","KIT","<pa
6   parm1 = "北原"
7   ret1 = ""
8   stmt, parm1, ret1 = ibm_db.callproc(conn, "KIT.HELLOCALL", (parm1, ret1))
9
10  print("ret1 = " + ret1)
11
12  ibm_db.free_result(stmt)
13  ibm_db.close(conn)
```

パラメータの定義

ストアドプロシージャの実行（callproc）
※ パラメータは tuple で指定
※ 実行後の戻り値も tuple で設定

# 今後の Python 分科会活動

# Python 分科会の活動候補


- 今後の Python 検証
  - 今回の続き
    - itoolkit
  - Web アプリケーション
    - フレームワーク(Django/Flask), パフォーマンス, Rest API, PHPとの比較
  - パッケージの利用： IBM i だけでどこまでできるのか？
    - データ解析、画像解析、機械学習

  - **皆さまへのお願い**
    - **RFE に 動画解析パッケージ (OpenCV) をリクエスト中です。**
      **ご興味のある方、ぜひご投票（volte）をお願いします**
      - https://www.ibm.com/developerworks/rfe/execute?use_case=viewRfe&CR_ID=140124

※その他、検証して欲しい事があればアンケートにご記入ください

# おまけ

IBM i の オープンソース環境をブラウザ(Chrome)で利用

# 参考：Chrome secure shell

# 参考：Chrome Secure Shell

## Chrome Secure Shell の利用

1. IBM i にユーザーのホームディレクトリを作成（作成済みの場合は不要、 MKDIR DIR('/home/<ユーザー>') ）
2. IBM i で SSHD を開始（開始済みの場合は不要）
3. Chrome ブラウザで「chrome ウェブストア」を表示
   - https://chrome.google.com/webstore
4. 「ストアを検索」に “secure shell app” と入力し Enter
5. 一覧から “Secure Shell” (以下のアイコン)をクリック
6. 「Chrome に追加」をクリック
7. 確認画面で「拡張機能を追加」をクリック
8. Chrome の右上にある「拡張機能アイコン」をクリック
9. 表示された「接続ダイアログ」をクリック
10. ユーザー、IBM I の IPアドレス、ポート(22)を入力し [ENTER]接続
11. 接続後、パスワードを入力（初回のみ接続確認があるので yes で回答）
12. 環境の作成（初回のみ）

# 参考：Chrome Secure Shell

## [補足] 10. ユーザー、IBM I の IPアドレス、ポート(22)を入力し [ENTER]接続


# 参考：Chrome Secure Shell

## [補足] 12. 環境の作成(初回のみ)

事前に以下を対応しておくと使いやすいです。
※ Secure shell にログイン後以下を実行

```
$ pwd
/home/KIT3
$ echo bash > .profile
$ echo alias python=python3 > .bashrc
$ echo alias pip=pip3 >> .bashrc
$ echo PATH=/QOpenSys/pkgs/bin:/QOpenSys/pkgs/sbin:$PATH >> .bashrc
```

作成したホームディレクトリになっている事を確認

bash シェルを利用するための設定（.profile）

以下、pyhton を利用するための設定（.bashrc）


実行後、以下の様に cat コマンドで作成したファイル（.profile, .bashrc）を確認

```
$ cat .profile
bash
$ cat .bashrc
alias python=python3
alias pip=pip3
PATH=/QOpenSys/pkgs/bin:/QOpenSys/pkgs/sbin:/QOpenSys/usr/bin:/usr/ccs/bin:/QOpenSys/usr/bin/X11:
/usr/sbin:.:/usr/bin
```

exit コマンドで一度終了し、再度接続する事で、作成した環境設定が反映されます。
環境設定の適用後は終了する際、exit を2回実行します(bashの終了とシステムからのログアウト)

IBM i の オープンソース環境を 無料で利用できる Microsoft Visual Studio Code で利用

# 参考：VSCode

# 参考：VSCode (Microsoft Visual Studio Code)

- VSCode の利用
  1. VSCode のインストール（無償）
     - ダウンロード： https://code.visualstudio.com/
  2. 拡張機能を導入
     - python
     - rpgle
     - sftp

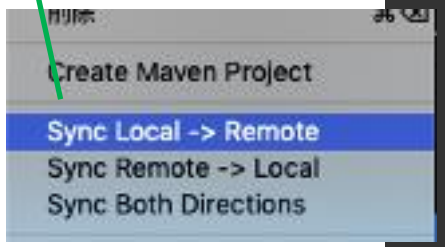
# 参考：VSCode (Microsoft Visual Studio Code)

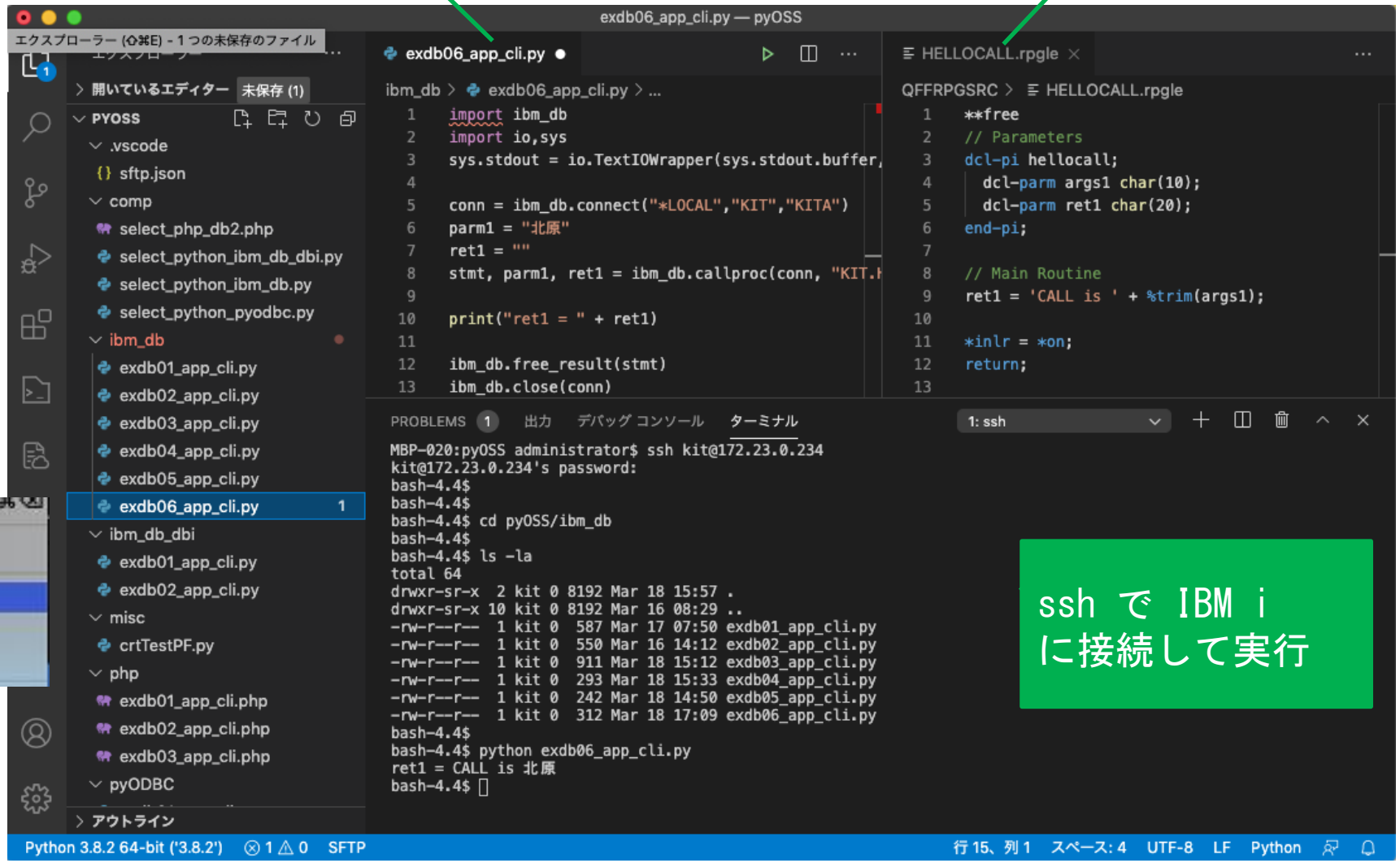
## ■ 利用のイメージ

python のコーディング

free form RPG のコーディング

右クリックで
IFS に配置
（sftp）




ssh で IBM i
に接続して実行



T&Trust Tenderness and Trust

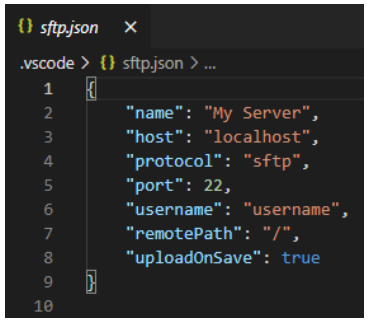
# 参考：VSCode (Microsoft Visual Studio Code)

- sftp の設定例

1. 管理→コマンドパレット


2. 画面上部に表示された入力域に “sftp” と入力し、sftp:config をクリック


3. 設定画面が表示される



# 参考：VSCode (Microsoft Visual Studio Code)

- sftp の設定例

```
{
    "name": "OSS Python Subcommittee",
    "host": "172.23.0.234",
    "protocol": "sftp",
    "port": 22,
    "username": "KIT",
    "password": "<password>",
    "remotePath": "/home/KIT/pyOSS/",
    "ignore": [
        ".vscode",
        ".git",
        ".DS_Store",
        ".svn"
    ],
    "syncOption": {
        "delete": false,
        "skipCreate": false,
        "ignoreExisting": false,
        "update": true
    },
    "uploadOnSave": false,
    "watcher": {
        "files": "/Users/kitahara/Documents/pyOSS/*",
        "autoUpload": false,
        "autoDelete": false
    }
}
```

IBM i の情報

ローカルPCの情報