SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

Mixing Python & COBOL

Roland Koo, IBM
Program Director, Product Management, Compilation Technologies, and Enterprise Products
rkoo@ca.ibm.com

Steven Pitman, IBM
Technical lead for IBM Open Enterprise SDK for Python
pitman@ca.ibm.com

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Agenda

SHARE
EDUCATE • NETWORK • INFLUENCE

- Motivation
- Python on z/OS Overview
- COBOL on z/OS Overview
- Types of Interoperability
- New address spaces
- Same address spaces

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Motivation – Why am I giving this talk?

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

- Clients have their already existing COBOL code

- Python makes new features easy to get access to
  - Ex: ML/AI Related functionality, automation, data analysis, web hosting, …
  - Big benefit of Python is the large amount of open source packages written by the community

- Clients do not want to rewrite any of their COBOL
  - Just use both – get the best of both worlds
  - Integrate Python & COBOL

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

IBM Open Enetperise SDK for Python

SHARE
EDUCATE • NETWORK • INFLUENCE

- Python interpreter that runs under z/OS UNIX
  - Contains the Python standard library
  - Has a few extra packages included

- Has extra features for z/OS specific tasks
  - Auto conversion for EBCDIC <-> ASCII inputs
  - Additional z/OS specific encodings
  - zEDC Offloading support

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

IBM Open Enterprise SDK for Python

SHARE
EDUCATE • NETWORK • INFLUENCE

- 3 Currently supported releases
  - Python 3.12, 3.13, 3.14
    - Note: 3.12 will be out of service soon in April 2026
  - Runs on all currently supported z/OS versions & CPU generations

- All Python versions are zIIP offloadable
  - Up to 70% of the runtime can be offloaded to the zIIP
  - Not everything is eligible for offloading – see our docs for details

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

IBM Open Enterprise SDK for Python

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

- Easy to obtain
  - Shop Z – SMP/E
    - Bypassable requisite for z/OS
  - Pax
  - z/OS Container Platform

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

IBM Enterprise COBOL for z/OS

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

- COBOL Compiler for z/OS
  - Two supported versions, 6.3 and 6.4
  - Runs on all supported versions of z/OS

- Strengths
  - English-like syntax makes the language easy to learn and business logic readable
  - Designed for business data processing (efficient in handling large-scale batch jobs, business transactions, large datasets…)
  - Ensures financial accuracy with precise decimal arithmetic with no rounding errors

- New Enterprise COBOL introduces features that make the language more approachable for new developers.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

IBM Enterprise COBOL for z/OS

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

- Runs in multiple environments
  - z/OS UNIX, Batch, …
- Has interoperability with other high-level languages
  - Ex: Java, C/C++, PL/I
- Available via ShopZ

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

What's an address space?

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

A range of virtual memory assigned by the operating system to a job, user, or running program. It provides an isolated, protected area in which that work executes

- Stores instructions for execution
  - Ex: Actual machine code
- Stores data used during the execution
  - Ex: Strings used for printing
- A close Unix equivalent is a process

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Types of Interoperability

- New address space
  - Spawns a new process
- Same address space
  - Directly call between the languages
  - Going to be focusing on this
- But COBOL is a compiled language, Python is interpreted
  - How’s this work then?

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Easy: New address space

SHARE
EDUCATE • NETWORK • INFLUENCE

- Python calling COBOL
  - Mulitple ways using standard library packages
  - Ex: subprocess, os
  - Can use Z Open Automation Utilities if a batch job is needed

- COBOL calling Python
  - Ex: Fork, execl

- Doesn’t matter if COBOL is 31 or 64-bit
  - A new process is being spawned, no direct interaction

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Example: Python calling COBOL

```python
import subprocess

if __name__ == '__main__':
    # Run our COBOL subprocess, get its output
    output = subprocess.check_output(['/path/to/cobol.binary'])

    # check_output returns bytes - so convert it to a usable string to print it out
    print(output.decode("utf-8"))
```

Python spawns a new subprocess with **subprocess.check_output**

```cobol
IDENTIFICATION DIVISION.
PROGRAM-ID. "RUNCBL".
DATA DIVISION.
WORKING-STORAGE SECTION.
PROCEDURE DIVISION.
    DISPLAY "HELLO, WORLD!"
    STOP RUN.
```

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Example: Python calling COBOL

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

```
# Build the COBOL binary
/c390/IGY/v6r3m0/usr/lpp/IBM/cobol/igyv6r3/bin/cob2 -o runcbl run.cbl

# Environment variables required to run IBM Open Enterprise SDK for Python
export PATH=/home/pitman/python3.13/published/GA-8/usr/lpp/IBM/cyp/v3r13/pyz/bin:/bin
export LIBPATH=/home/pitman/python3.13/published/GA-8/usr/lpp/IBM/cyp/v3r13/pyz/lib:/lib
export _BPXK_AUTOCVT=ON
export _CEE_RUNOPTS='FILETAG(AUTOCVT,AUTOTAG) POSIX(ON)'

# Run Python - which will spawn a subprocess that calls the COBOL binary
python3 run.py
```

Shell script to compile the COBOL, set up environment for Python, then run Python code

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Example: COBOL Calling Python

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

```
PROCEDURE DIVISION.
    CALL "fork" returning PID

    If PID < 0 then
      Display "Fork failed"
    Else
      If PID = 0 then
        DISPLAY "From child"

        Move Z"/home/pitman/python3.13/published/GA-8/usr/lpp/IBM
              "/cyp/v3r13/pyz/bin/python3" to exec-path
        Set exec-path-ptr to address of exec-path

        Move Z"/home/pitman/python3.13/shar/new/1.coboltopython/r
              "un.py" to exec-param1
        Set exec-param1-ptr to address of exec-param1

        CALL "execl" using by value exec-path-ptr
                           by value exec-param1-ptr
                           by value null-ptr

        DISPLAY "execl failed"
      Else
        DISPLAY "From parent"
        CALL "wait"
      End-if
    End-if

    STOP RUN.
```

```
if __name__ == '__main__':
    print('Hello, world')
```

COBOL Calls fork & the child exec's to spawn the Python process

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Example: COBOL Calling Python

SHARE
EDUCATE • NETWORK • INFLUENCE

```
# Build the COBOL binary
/c390/IGY/v6r3m0/usr/lpp/IBM/cobol/igyv6r3/bin/cob2 -o runcbl run.cbl

# Environment variables required to run IBM Open Enterprise SDK for Python
export PATH=/home/pitman/python3.13/published/GA-8/usr/lpp/IBM/cyp/v3r13/pyz/bin:/bin
export LIBPATH=/home/pitman/python3.13/published/GA-8/usr/lpp/IBM/cyp/v3r13/pyz/lib:/lib
export PYTHONHOME=/home/pitman/python3.13/published/GA-8/usr/lpp/IBM/cyp/v3r13/pyz
export _BPXK_AUTOCVT=ON
export _CEE_RUNOPTS='FILETAG(AUTOCVT,AUTOTAG) POSIX(ON)'

# Run our COBOL binary, which will spawn a Python process
./runcbl
```

Shell script which compiles the COBOL code, sets up the Python environment, then runs the COBOL

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Harder: Same address space

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

- Calling a function within the same address space has potential issues
  - Is it the same AMODE? 31-bit or 64-bit

- If the COBOL is 64-bit
  - Can just call Pythons C API directly

- If the COBOL is 31-bit
  - It requires an AMODE change, can call LE's API's to do this
  - CEL4RO31 & CEL4RO64

- I'll show an example of both

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Same address space – 64-bit COBOL

- Several different for Python to call 64-bit COBOL
  - Load up the shared library with cffi or ctypes
  - Write a Python package in C, call the COBOL from C
  - In both cases, the COBOL must be a DLL

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Example: Calling 64-bit COBOL from Python

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

```python
import ctypes

# Load up the COBOL DLL
mydll = ctypes.CDLL("./cobtest.so")

# Set up the function argument and return types
mydll.COBTEST.argtypes = [ctypes.c_int, ctypes.c_int]
mydll.COBTEST.restype = ctypes.c_int

# Call our function that adds two numbers together
output = mydll.COBTEST(5,6)

# Result should print out 11
print(output)
```

Use ctypes to load the COBOL DLL and call the function

```cobol
IDENTIFICATION DIVISION.
PROGRAM-ID. 'COBTEST'.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
2 references
01 PARAM1                    PIC S9(9) USAGE IS BINARY.
2 references
01 PARAM2                    PIC S9(9) USAGE IS BINARY.
3 references
01 RETCODE                   PIC S9(9) USAGE IS BINARY.
PROCEDURE DIVISION USING BY VALUE PARAM1
                         BY VALUE PARAM2
                         RETURNING RETCODE.
    ADD PARAM1 to RETCODE
    ADD PARAM2 to RETCODE
    GOBACK.
```

COBOL – defines a function that has 2 Integer parameters and returns the addition of the 2

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Example: Calling 64-bit COBOL from Python

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

```
# Environment variables required to run IBM Open Enterprise SDK for Python
export PATH=/home/pitman/python3.13/published/GA-8/usr/lpp/IBM/cyp/v3r13/pyz/bin:/bin
export LIBPATH=/home/pitman/python3.13/published/GA-8/usr/lpp/IBM/cyp/v3r13/pyz/lib:/lib
export _BPXK_AUTOCVT=ON
export _CEE_RUNOPTS='FILETAG(AUTOCVT,AUTOTAG) POSIX(ON)'

# Build COBOL shared library
/c390/IGY/v6r3m0/usr/lpp/IBM/cobol/igyv6r3/bin/cob2 -q64 -q"dll" -q"pgmname(longmixed)" -bdll -qexportall cobtest.cbl -o cobtest.so

# Run Python to call the 64bit COBOL
python3 call_cobtest.py
```

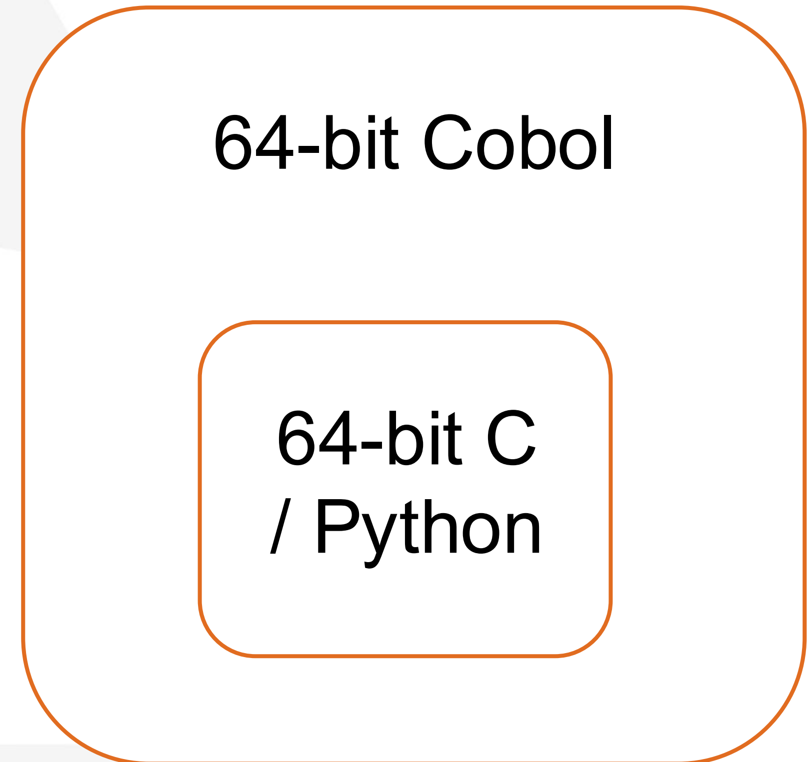
Shell script which sets up the Python environment, compiles the COBOL code, and runs Python

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Same address space – 64-bit COBOL

- COBOL calling 64-bit Python
  - Call Pythons C API directly from COBOL

- Python has a well documented API



Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Example: Calling Python from 64-bit COBOL

SHARE
EDUCATE • NETWORK • INFLUENCE

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "COBTEST".
DATA DIVISION.
WORKING-STORAGE SECTION.
1 reference
01 pyrun PIC u(80) VALUE z'print("Hello, world")'.
PROCEDURE DIVISION.
*    Initialize the Python interpreter
     CALL "Py_Initialize"

*    Run a Python function defined in pyrun
     CALL "PyRun_SimpleString" USING
     BY REFERENCE pyrun
     END-CALL

*    Clean up the Python interpreter
     CALL "Py_Finalize"

     STOP RUN.
```

COBOL Calling Pythons C API to run a Python function

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Example: Calling Python from 64-bit COBOL

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

```
# Environment variables required to run IBM Open Enterprise SDK for Python
export PATH=/home/pitman/python3.13/published/GA-8/usr/lpp/IBM/cyp/v3r13/pyz/bin:/bin
export LIBPATH=/home/pitman/python3.13/published/GA-8/usr/lpp/IBM/cyp/v3r13/pyz/lib:/lib
export _BPXK_AUTOCVT=ON
export _CEE_RUNOPTS='FILETAG(AUTOCVT,AUTOTAG) POSIX(ON)'

# Compile COBOL which embeds Python
/c390/IGY/v6r3m0/usr/lpp/IBM/cobol/igyv6r3/bin/cob2 -q64 -q"pgmname(longmixed)" cobtest.cbl \
    -o cobtest /home/pitman/python3.13/published/GA-8/usr/lpp/IBM/cyp/v3r13/pyz/lib/libpython3.13.x

# Run the COBOL program - expected outout is a message 'Hello world'
./cobtest
```

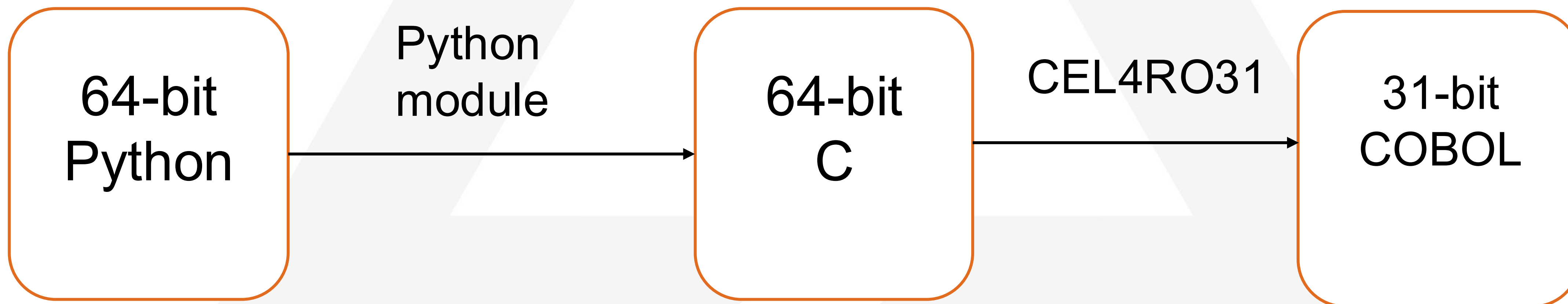
Shell script to compile the COBOL program which calls Pythons C API, and then runs the COBOL program

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Python calling 31-bit COBOL

- Python allows you to write packages in other languages
  - Commonly using C or C++

- Python is a 64-bit application
  - But from C, we can call the LE service CEL4RO31
  - This can run a AMODE31 function from AMODE64



Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Example: Calling 31-bit COBOL from Python

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

```c
/* Setting up iconv function name conversion from utf to ebcdic */
char outbuf_func[FLENGTH];
char *inptr_func = "COBTEST";
char *outptr_func = outbuf_func;
size_t insize_func = FLENGTH;
size_t outsize_func = FLENGTH;

size_t func_res = iconv(cd, &inptr_func, &insize_func, &outptr_func, &outsize_func);
if (func_res == (size_t)-1) {
    printf("iconv had an unsuccesful attempt to convert function\n");
    return 0;
}
strcpy((*R031_func_p).function_name, outbuf_func);

iconv_close(cd);

printf("calling\n");
/* Call CEL4R031()       */
CEL4R031((void*)R031_info);
```

C code which uses CEL4RO31 to call the COBOL Program COBTEST. This code is contained within a Python Package.

Full sample is large - see full example on Github here

```cobol
IDENTIFICATION DIVISION.
PROGRAM-ID. 'COBTEST'.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
2 references
01 PARAM1                   PIC S9(9) USAGE IS BINARY.
2 references
01 PARAM2                   PIC S9(9) USAGE IS BINARY.
3 references
01 RETCODE                  PIC S9(9) USAGE IS BINARY.
PROCEDURE DIVISION USING BY VALUE PARAM1
                         BY VALUE PARAM2
                         RETURNING RETCODE.
    ADD PARAM1 to RETCODE
    ADD PARAM2 to RETCODE
    GOBACK.
```

The COBOL program COBTEST

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Example: Calling 31-bit COBOL from Python

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

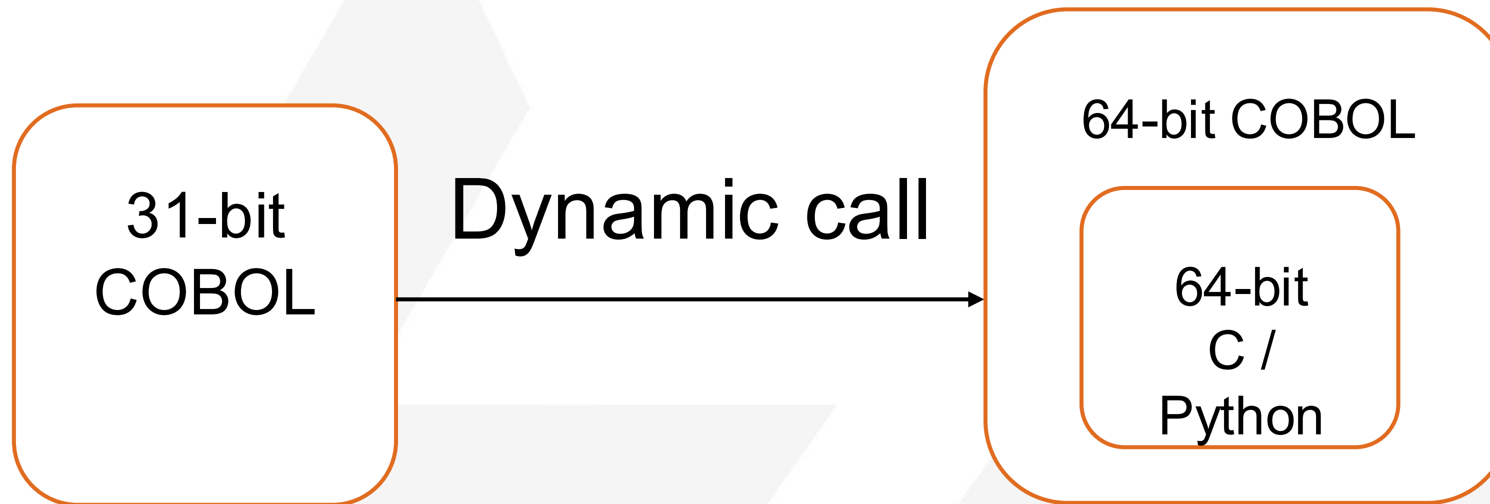
```
# Environment variables required to run IBM Open Enterprise SDK for Python
export PATH=/home/pitman/python3.13/published/GA-8/usr/lpp/IBM/cyp/v3r13/pyz/bin:/bin
export LIBPATH=/home/pitman/python3.13/published/GA-8/usr/lpp/IBM/cyp/v3r13/pyz/lib:/home/pyzbld/zEDC/hzc/lib:/lib
export _BPXK_AUTOCVT=ON
export _CEE_RUNOPTS='FILETAG(AUTOCVT,AUTOTAG) POSIX(ON)'

python3 -m venv venv
source ./venv/bin/activate

# Build our COBOL module
/c390/IGY/v6r3m0/usr/lpp/IBM/cobol/igyv6r3/bin/cob2 -q"dll" -q"pgmname(longmixed)" -bdll -qexportall cobtest.cbl -o cobtest.so

# Build our Python package
pip3 install -v ./callcobol

# Call the 31bit COBOL from Python
export LIBPATH=.:$LIBPATH
python3 -c "import callcobol; print(callcobol.call_cobtest(5,6))"
```

Shell script to compile the COBOL application, install the Python package which contains the C code, and then runs Python

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

31-bit COBOL calling Python

- 31-bit COBOL can dynamically call 64-bit programs. This program can then DLL call the Python C API
- Restriction: The callee of the dynamic call must be in a z/OS dataset



Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Example: Calling Python from 31-bit COBOL

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "COBTEST".
DATA DIVISION.
WORKING-STORAGE SECTION.
2 references
77   PGM-NAME                  PICTURE X(13).
LINKAGE SECTION.
PROCEDURE DIVISION.
*Dynamically call our 64-bit COBOL program
     MOVE "DYCALLEE" to PGM-NAME.
     CALL PGM-NAME.
     STOP RUN.
```

31-bit COBOL application which calls the 64-bit COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "DYCALLEE".
DATA DIVISION.
WORKING-STORAGE SECTION.
1 reference
01 pyrun PIC u(80) VALUE z'print("hello world")'.
LINKAGE SECTION.
PROCEDURE DIVISION.
    CALL "Py_Initialize"
    CALL "PyRun_SimpleString" USING
    BY REFERENCE pyrun
    END-CALL
    CALL "Py_Finalize"
    GOBACK.
```

64-bit COBOL application which can then call Pythons C API

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Summary

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

- Learned about interoperability with Python & COBOL
  - New address space
  - Same address space

  - Learned about the problems with differing AMODE

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Resources

- IBM Open Enterprise SDK for Python
- Z Open Automation Utilities
- Blog: Using Python with JCL and REXX
- Blog: Python & 64-bit COBOL Interoperability
- Blog: Python & 31-bit COBOL Interoperability
- Blog: How to call existing COBOL modules from Python

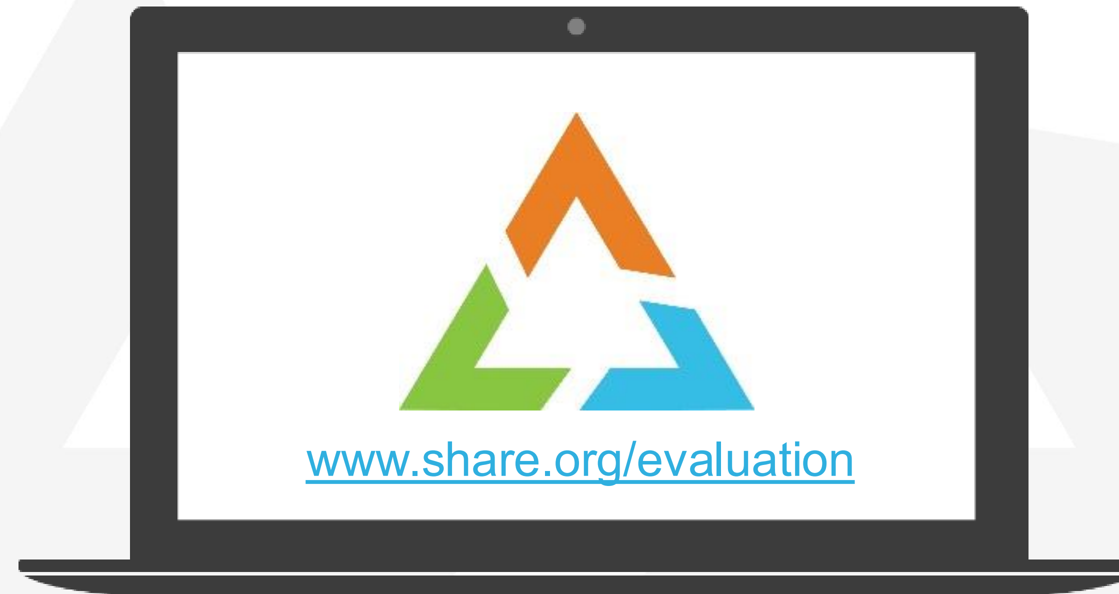
Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Your feedback is important!

SHARE
EDUCATE ‣ NETWORK ‣ INFLUENCE

Submit a session evaluation for each session you attend:

www.share.org/evaluation



Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/