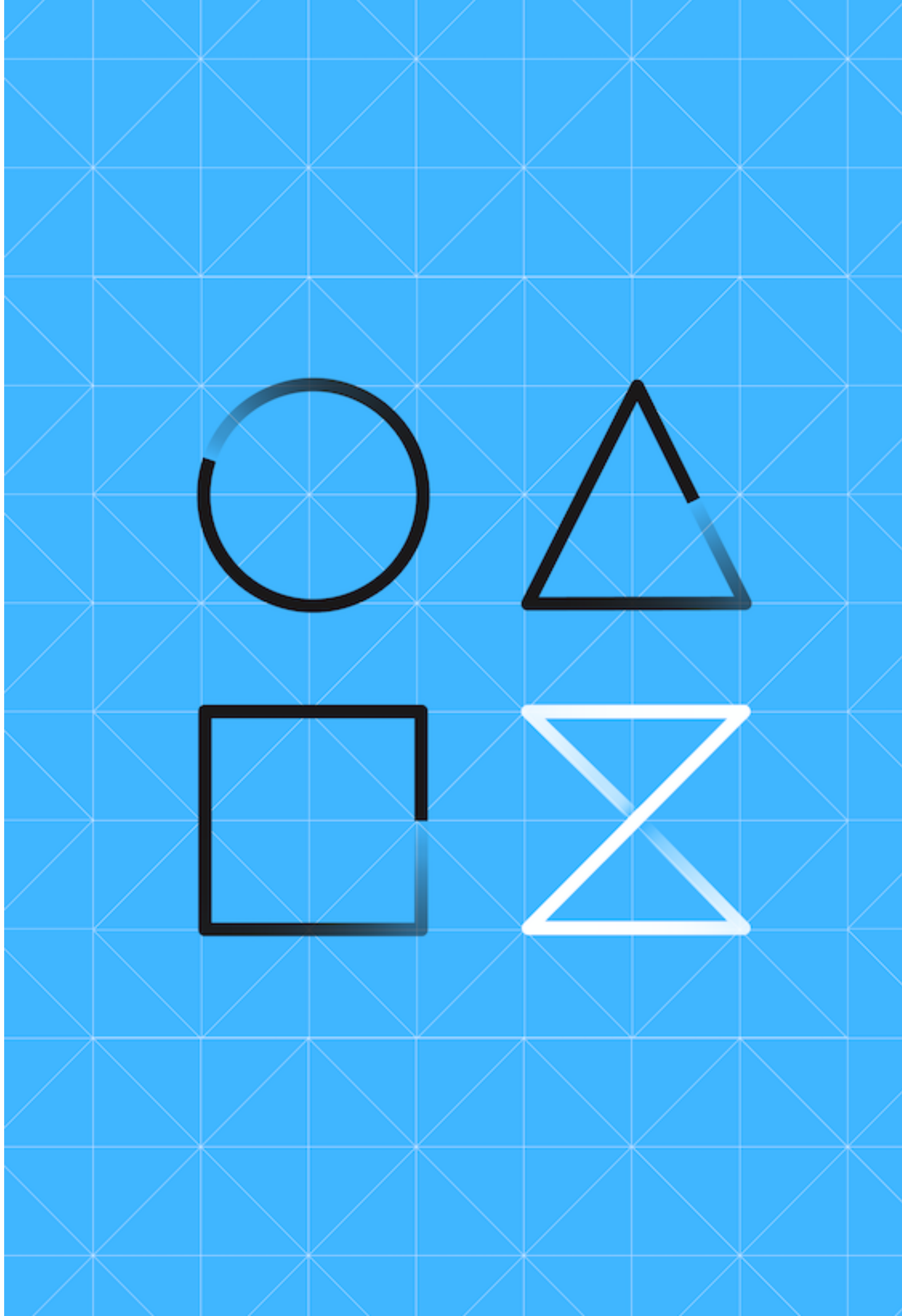
Fundamentals

230815-2310

WRAPUP

Work Smarter



Z Xplore

TIME TO BRING IN Z OPEN AUTOMATION UTILITIES

The Challenge

Now that you know some of the core z/OS fundamentals, and you have also explored a bit into scripting and Python, let's bring it all together with something called the 'Z Open Automation Utilities' or, to save some space from now on, *ZOAU*.

Combine the core IBM Z skills you have acquired, from learning about data sets and JCL, with the scripting abilities found in Python, to automate some tasks a System Programmer might perform on a day-to-day basis.

Before You Begin

At this point, you know how to work with data sets, submit jobs, and navigate around USS.

You'll put all that to use here and wrap up your fundamental understanding of z/OS.

Investment

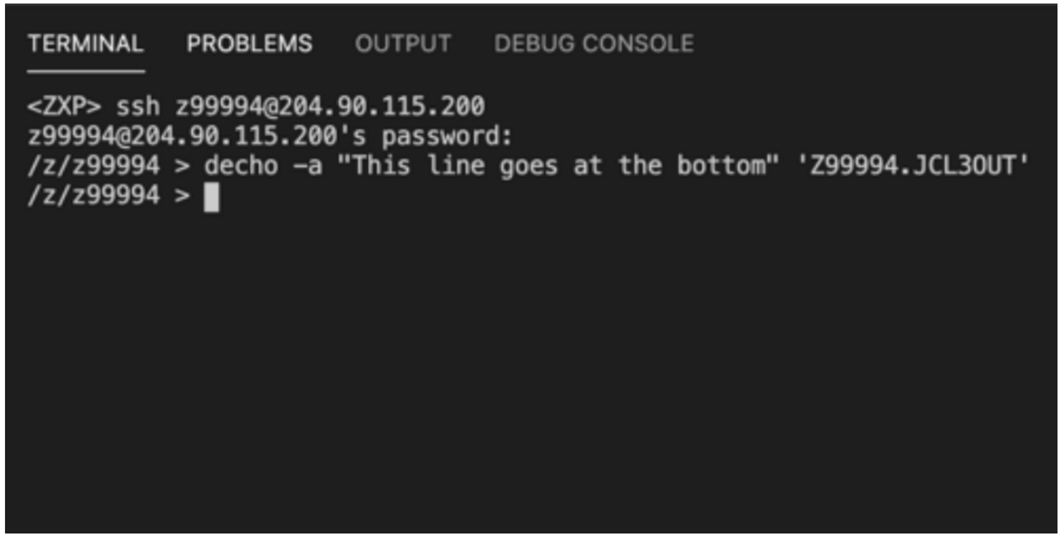
Steps	Duration
12	90 minutes

1 DECHO … DECHO … DECHO

You will be using a sequential data set for this challenge.

If you still have your JCL3OUT sequential data set, use that; otherwise, make new one by Right-clicking on your VSCode connection profile in ‘Data Sets’, selecting “Create New Data Set” and following the prompts.

All set?



```
<ZXP> ssh z99994@204.90.115.200
z99994@204.90.115.200's password:
/z/z99994 > decho -a "This line goes at the bottom" 'Z99994.JCL3OUT'
/z/z99994 >
```

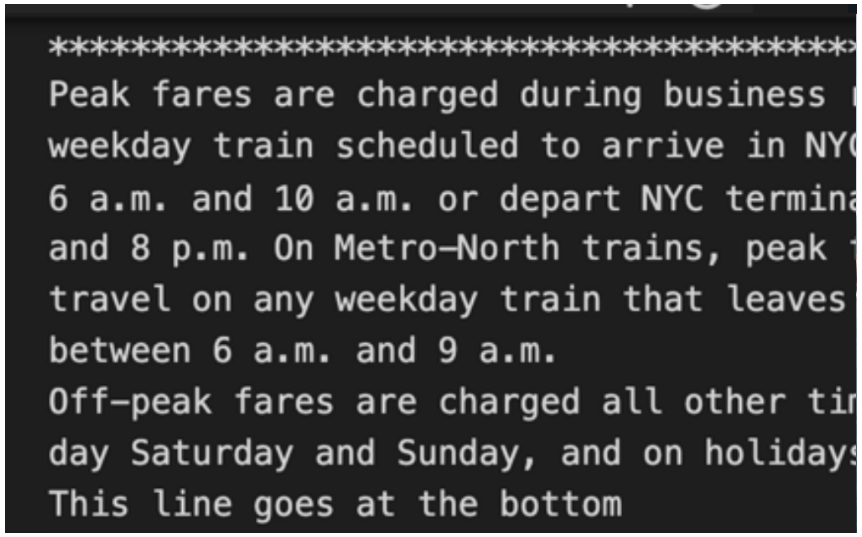
Next - SSH into the mainframe system again so you get a USS shell and then enter the correct version of the following command, substituting in your own userid and the data set you want to use.

```
decho -a "This line goes at the bottom" 'Zxxxxx.JCL3OUT'
```

This may take a few seconds to fully execute, so be patient.

2 HELLO DOWN THERE

Edit the JCL3OUT dataset in VSCode. If you created it again, you may need to Right-click on it and select “Pull From Mainframe”; this makes sure you have absolute latest version in your edit session.



```
****************************************
Peak fares are charged during business
weekday train scheduled to arrive in NY
6 a.m. and 10 a.m. or depart NYC termina
and 8 p.m. On Metro-North trains, peak
travel on any weekday train that leaves
between 6 a.m. and 9 a.m.
Off-peak fares are charged all other ti
day Saturday and Sunday, and on holiday
This line goes at the bottom
```

You should see the line you just `decho`’d appended to the bottom of the data set.

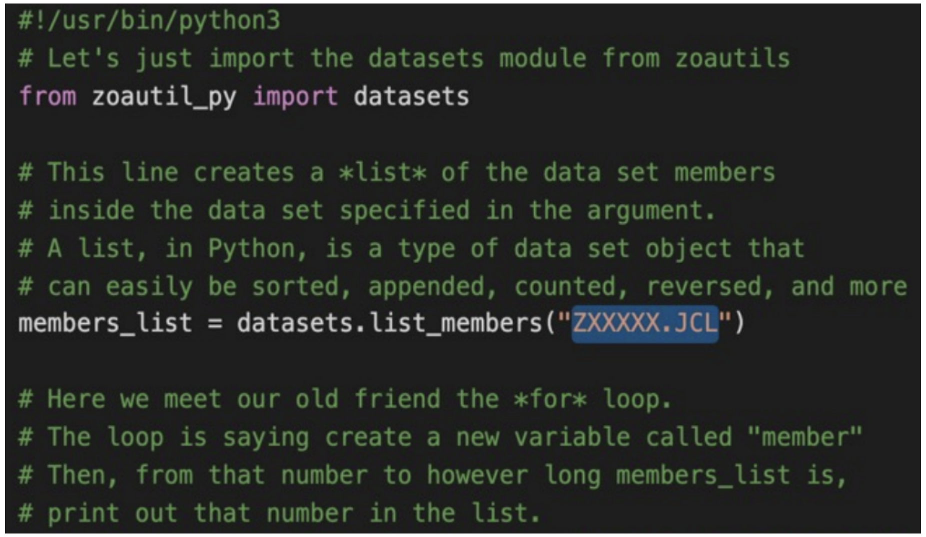
A neat trick, though probably not easier than just opening it up and typing it in there manually.

The real power of these tools comes from being able to integrate z/OS actions into new and existing Python programs and shell scripts.

3 ALL SET WITH DATASETS

Make sure you have a copy of **dslist.py** in your home directory in USS.

You should take a quick peek at the code to figure out what it will be doing.



```
#!/usr/bin/python3
# Let's just import the datasets module from zoautils
from zoautil_py import datasets

# This line creates a *list* of the data set members
# inside the data set specified in the argument.
# A list, in Python, is a type of data set object that
# can easily be sorted, appended, counted, reversed, and more
members_list = datasets.list_members("ZXXXXX.JCL")

# Here we meet our old friend the *for* loop.
# The loop is saying create a new variable called "member"
# Then, from that number to however long members_list is,
# print out that number in the list.
```

It's a Python script (if you didn't guess by the .py suffix) and you can see it gets a listing of all the members in a particular data set, then uses a 'for loop' to print out each member name in it.

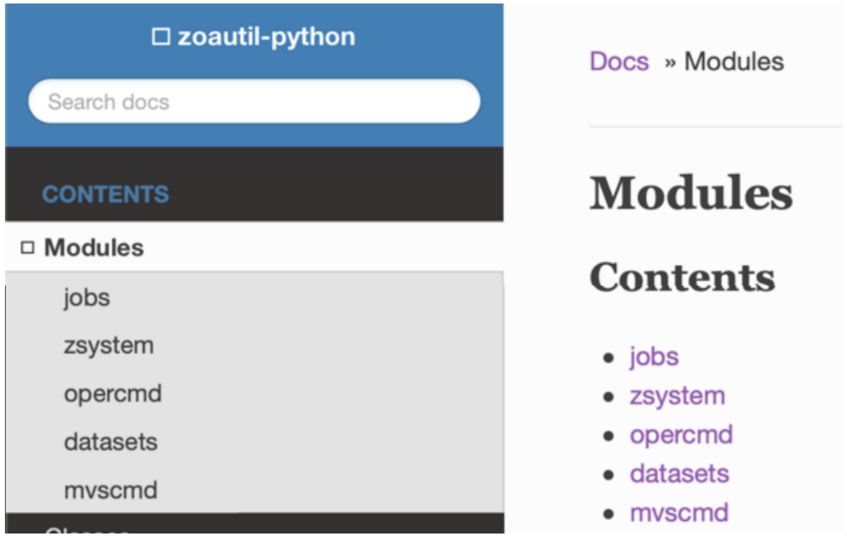
At least, it will, once you have edited the script to point to one of your partitioned datasets on line number 9 (the line beginning `members_list =`). Try now by specifying your JCL or OUTPUT dataset.

Save the file, and execute it from a USS shell login.

NOTE: remember in the **CODE** challenge that every time you executed a python script you used the `python3` command, *not* plain old `python`.

Nice and simple, right?

4 WAITER - THERE'S A PYTHON IN MY Z



Writing a Python script just to list what's in a data set might seem like overkill, and in this case … yes, it is.

However, with a very simple program, you can see just how it can be done.

And once you see and understand it, maybe it will give you some ideas.

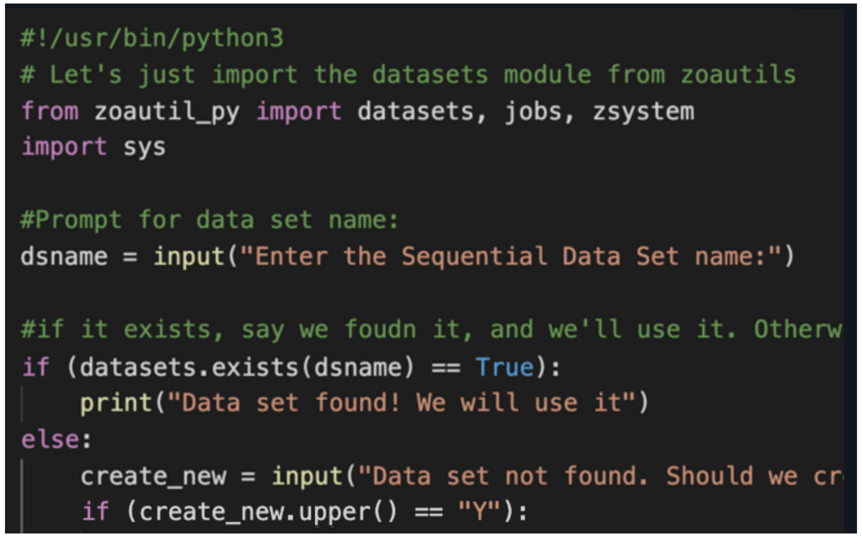
Now that you know how you can perform core z/OS tasks within Python code. This example only involved datasets. You can also work with jobs, the system itself, and two types of commands.

Take a look at the ZOAU Python APIs page and read more about these modules at https://www.ibm.com/docs/en/zoau/1.2.0?topic=reference-classes

(this location changes - if you need to find a different version, use an internet search for "*zoau python reference classes*")

5 INSERT SOME EXTRA LOGIC

Now you can work with a script that does a bit more, including creating a new sequential dataset, gathering some data, and then writing that data out to your newly-created sequential dataset.



```
#!/usr/bin/python3
# Let's just import the datasets module from zoautils
from zoautil_py import datasets, jobs, zsystem
import sys

#Prompt for data set name:
dsname = input("Enter the Sequential Data Set name:")

#if it exists, say we foudn it, and we'll use it. Otherw
if (datasets.exists(dsname) == True):
    print("Data set found! We will use it")
else:
    create_new = input("Data set not found. Should we cr
    if (create_new.upper() == "Y"):
```

You can get started on a couple of these functions that already work, and your job will be to get the rest of them working smoothly.

Open up your ‘members.py’ file in your home directory and take a look at the source.

6 GET YOUR HACK ON!

At this point, you might be seeing a lot of code and getting nervous.

Here's the deal … even if you don't want to be a programmer, if you work with systems, you'll have to spend some time looking at code and making small adjustments to other people's programmes.

That is the very spirit of hacking.

A lot of coding involves figuring out where to start, pulling together all of the resources you need, and creating something from scratch.

Hacking is about taking something that does something useful already and making small changes to add a feature, fix problems, or to just leave your own little mark on it.

```
# These are comments
# They are here to help
```

Whenever there's code involved, there are LOTS of comments to help you figure out what's going on, and know that (at least for this challenge) you won't never need to write more than a few lines of new code.

Working out the flow and structure of what is already there will probably be the hardest thing.

But don’t worry - You’ve got this!

WHY LEARN ANOTHER WAY? BECAUSE IT'S GOOD TO HAVE OPTIONS.

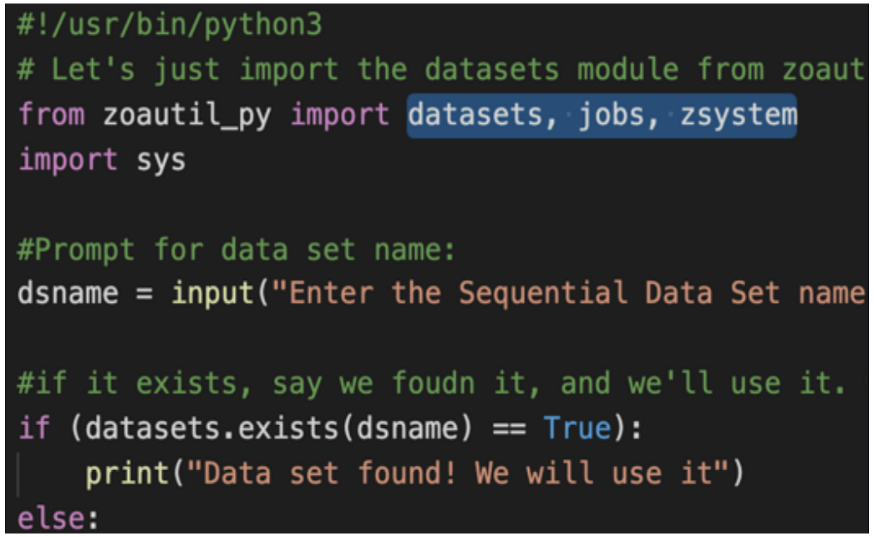
If you started working on z/OS years ago, the chances are you're very comfortable with JCL and the many methods already built into the operating system for getting things done.

However, if you're used to working with Linux and Python, you may want to keep writing code the same way, while still having access to the core functionality of z/OS. That's where the ZOAU library modules come in handy.

Learning new methods in addition to what we already know enables more options, and more opportunity to make efficient choices.

These exercises are here to give you a few ideas of what is possible when you combine the scripting and automation ability of Python with the backend capabilities of z/OS.

7 LOADING MODULES



```
#!/usr/bin/python3
# Let's just import the datasets module from zoaut
from zoautil_py import datasets, jobs, zsystem
import sys

#Prompt for data set name:
dsname = input("Enter the Sequential Data Set name

#if it exists, say we foudn it, and we'll use it.
if (datasets.exists(dsname) == True):
    print("Data set found! We will use it")
else:
```

Look at line #3 of **members.py**.

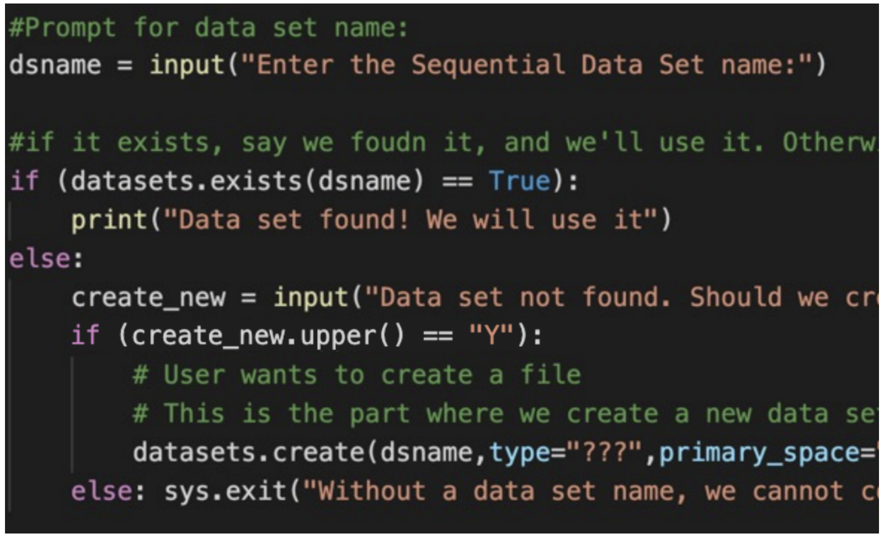
Three modules are being imported from the ‘zoautil.py’ library: datasets, jobs, and zsystem.

Each of these modules provides a set of functions which you can now use in your code.

You can read all about the modules and what they do here:

https://www.ibm.com/docs/en/zoau/1.2.0?topic=python-api-reference

8 GET THAT DATASET



```
#Prompt for data set name:
dsname = input("Enter the Sequential Data Set name:")

#if it exists, say we foudn it, and we'll use it. Otherw
if (datasets.exists(dsname) == True):
    print("Data set found! We will use it")
else:
    create_new = input("Data set not found. Should we cr
    if (create_new.upper() == "Y"):
        # User wants to create a file
        # This is the part where we create a new data se
        datasets.create(dsname,type="???",primary_space=
    else: sys.exit("Without a data set name, we cannot c
```

Lines 6-18 of **members.py** ask the user for a sequential dataset name and assigns that value to the variable ‘dsname’.

If the dataset already exists, the program will use that and continue on.

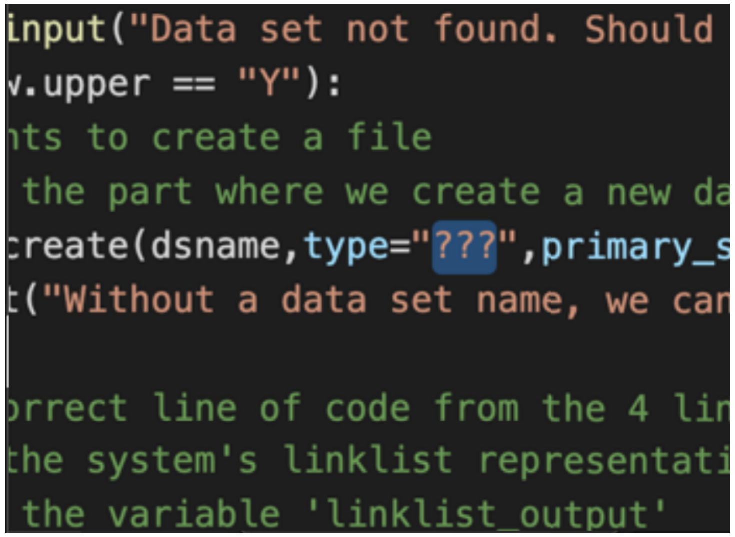
If the data set does *not* exist, you can create that for the user (which you will do in the next step).

If the dataset does NOT exist, and the user says they don’t want to create it, then there isn’t much sense in continuing the rest of the code, so ‘sys.exit( )’ is used to quit the program.

So far, so good? Alright, let’s start hacking some code …

9 IN THE RIGHT SEQUENCE

For this challenge, you'll be using the script (**members.py**) to gather data and write it out to a sequential dataset.



```
input("Data set not found. Should
v.upper == "Y"):
nts to create a file
 the part where we create a new da
create(dsname,type="???",primary_s
t("Without a data set name, we can

orrect line of code from the 4 lin
the system's linklist representati
 the variable 'linklist_output'
```

Line #17 is using the create function from the zoau 'datasets' module to attempt to create this sequential dataset. But as it is right now, it will not work until you make one small adjustment. Take a look at the *datasets.create( )* function and work out what needs to replace the **???** in order to create a sequential dataset.

Use **Zxxxxx.COMPLETE** as the sequential dataset name when prompted by the script. That's where the validation job will be looking to validate your work, as well as executing your 'members.py'.

Hint: A sequential dataset is a particular type of dataset. Other types of datasets are KSDS, PDS, ESDS, but in this case, the type you need to create is a sequential dataset. Check the documentation and comments for help.

KSDS? ESDS? I THOUGHT IT WAS JUST A PARTITIONED AND SEQUENTIAL?

Sometimes a dataset is little more than a simple file; a place to hold some data that you want to read or process from top to bottom. Partitioned Dataset members and sequential datasets work well for these cases.

However, sometimes applications need fast access to a specific record, and they need a better way of getting to that record than reading the whole dataset top to bottom. In these situations, you can use a *Key Sequenced Data Set* (KSDS), *Entry Sequenced Data Set* (ESDS) or Relative Record Data Set (RRDS). These are all examples of Virtual Storage Access Method (VSAM) data sets, and these will be covered in later challenges.

Having options when it comes to data access means the developer or system programmer can choose the best, fastest, most efficient way of working with all those bits and bytes.

Understanding the options and how to make it all work makes you a valuable z/OS professional, and employers definitely like that sort of thing.

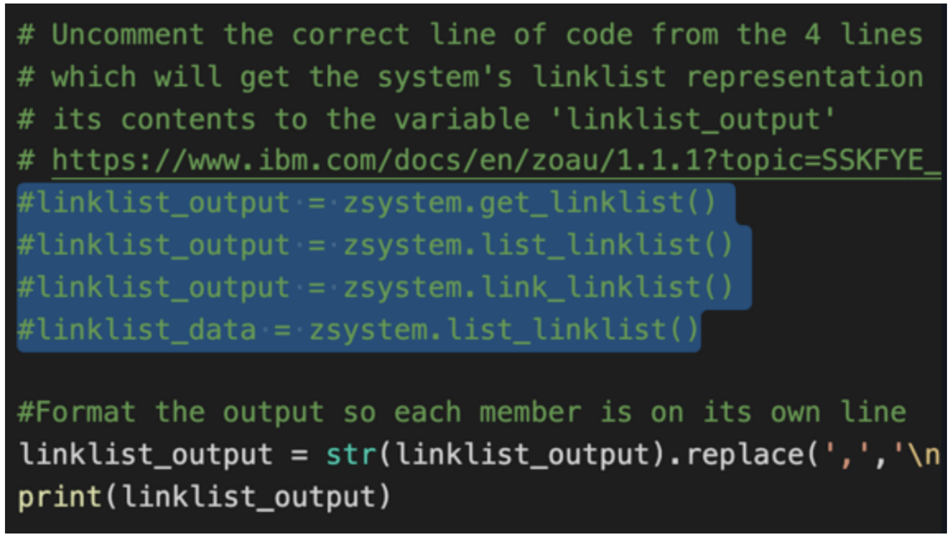
10 GRAB THE LINKLIST

Remember how you imported some modules at the beginning of this script? When the z/OS operating system starts up, it sort of does the same thing, loading modules and libraries full of the types of capabilities the user might need to use.

The full list of loaded libraries is known as the linklist, and it’s what lets a user just type out a single command instead of having to reference it by its full path every time.

Very handy!

Anyway, knowing what the full linklist looks like is kind of important, and it is available right within the zsystem module of zoautil.py.



```
# Uncomment the correct line of code from the 4 lines
# which will get the system's linklist representation
# its contents to the variable 'linklist_output'
# https://www.ibm.com/docs/en/zoau/1.1.1?topic=SSKFYE_
#linklist_output = zsystem.get_linklist()
#linklist_output = zsystem.list_linklist()
#linklist_output = zsystem.link_linklist()
#linklist_data = zsystem.list_linklist()

#Format the output so each member is on its own line
linklist_output = str(linklist_output).replace(',','\n
print(linklist_output)
```

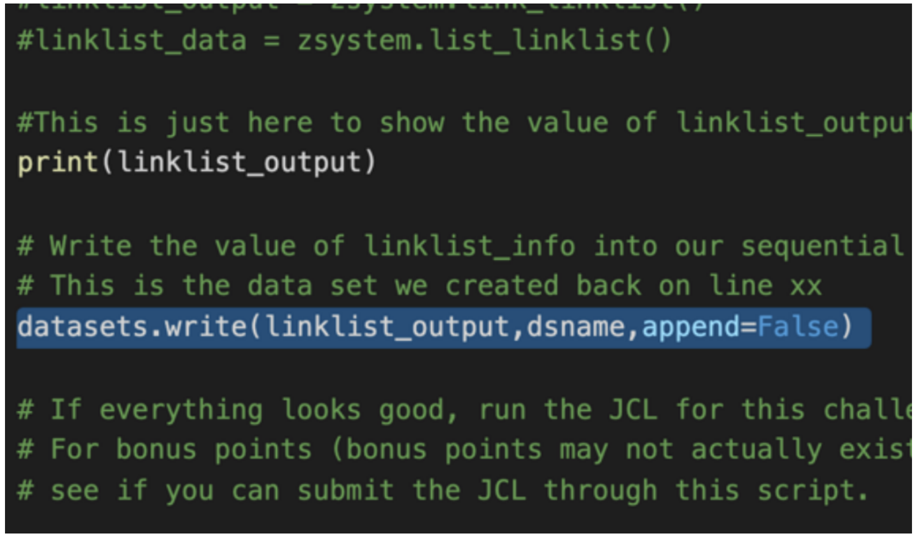
Look at lines 20-27. As you can see, there are four lines of code in there (and a lot of comments which you probably should read).

Your task is to identity and uncomment the one line of code (from 24-27) that is the correct line that will capture the linklist representation and assign it to the ‘linklist_output’ variable.

Note: the link to online documentation in the code may be out of date; you will encounter this quite a lot - comments get out of date, but do not always get updated with code changes.

11 LAST STOP: WRITING OUT

So far, you have created (or at least re-used) a sequential dataset and gathered some data. Now you need to write that data out into the dataset at end of your script.

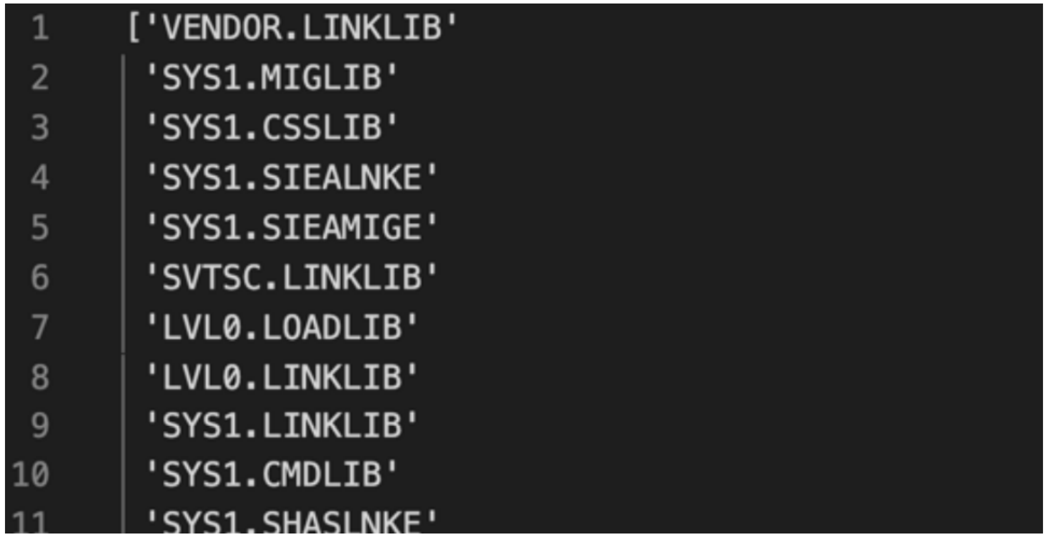


```
#linklist_data = zsystem.list_linklist()

#This is just here to show the value of linklist_output
print(linklist_output)

# Write the value of linklist_info into our sequential
# This is the data set we created back on line xx
datasets.write(linklist_output,dsname,append=False)

# If everything looks good, run the JCL for this challe
# For bonus points (bonus points may not actually exist
# see if you can submit the JCL through this script.
```

All the information is there, but something is wrong and you’ll need to take a look at the ‘datasets.write( )’ method and its documentation, to figure out what needs fixed.

Once you’ve got it all straightened out and in order, run the program and make sure the script works correctly for you.

12 CHECK IT; FINISH IT



```
['VENDOR.LINKLIB'
 'SYS1.MIGLIB'
 'SYS1.CSSLIB'
 'SYS1.SIEALNKE'
 'SYS1.SIEAMIGE'
 'SVTSC.LINKLIB'
 'LVL0.LOADLIB'
 'LVL0.LINKLIB'
 'SYS1.LINKLIB'
 'SYS1.CMDLIB'
 'SYS1.SHASLNKE'
```

When you completed all of the necessary steps, you should have a Zxxxxx.COMPLETE sequential dataset that contains a listing of the system's linklist. Refer to the screenshot above if you need clarification.

Find and submit the **CHKAUTO** job to mark this challenge complete.

You can do that the way you've been doing it up until now through VSCode, but I'll bet there's a Python function for submitting JCL that might work nicely, too.

Either way, congratulations on completing all the Fundamentals challenges!

You're all set to go bigger and further.

Nice job - let’s recap	Next up …
You scripted a series of deep fundamental z/OS actions right from a Python script! We kept it fairly simple in this example, but now that you know how to connect the dots, you should be able to create any number of new utilities that may be helpful in processing text, checking job output, verifying system changes, and working with data sets. You knew how to do most of this before, and now you know another way. Like we said, more options are good to have.	You’ve completed all the Fundamental challenges! You’ve got what it takes to move on to the Advanced challenges. By now, you probably have some idea of what you’re interested in, and you’ll probably find it in the next series of available challenges.