ZOAU2

Power though Python

Using Python to create a data processing app

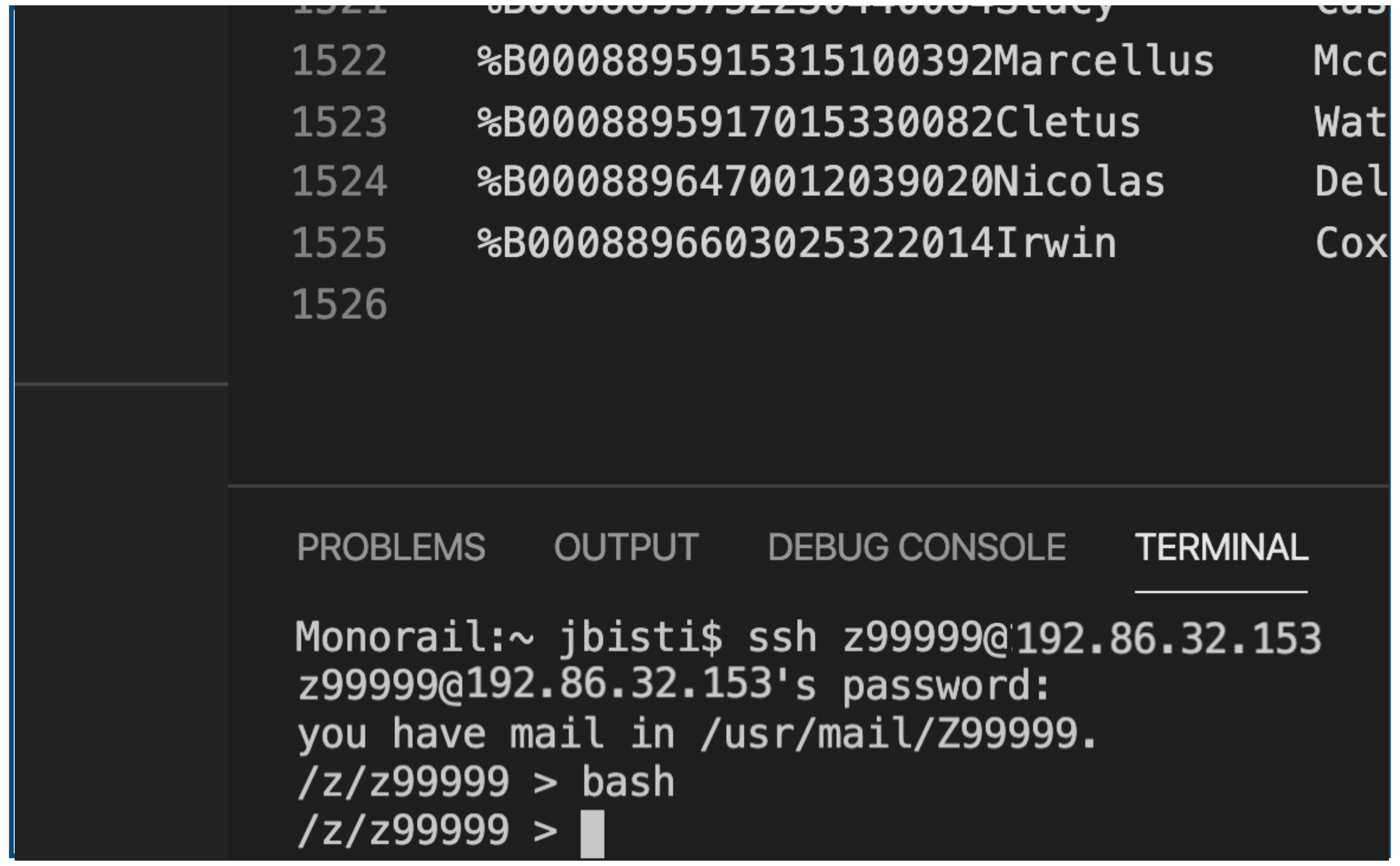
9 steps   4 Hours

THE CHALLENGE

The Z Open Automation Utilities support shell scripts, Python and Node.js. You previously saw how easy it was to stick ZOAU commands into a shell script, but when you start dealing with more complicated data and processes, you'll want to use a language with greater capabilities and flexibility. In this challenge, we'll build an app to validate credit card data using Z Open Automation Utilities and Python. Don't be afraid if you're not the programming type, this is about logic more than anything.

BEFORE YOU BEGIN

Definitely complete the first ZOAU challenge before attempting this one. You will need to write some Python code to complete this, which will require some research and thinking, but it's nothing too tricky for even a Python newbie.



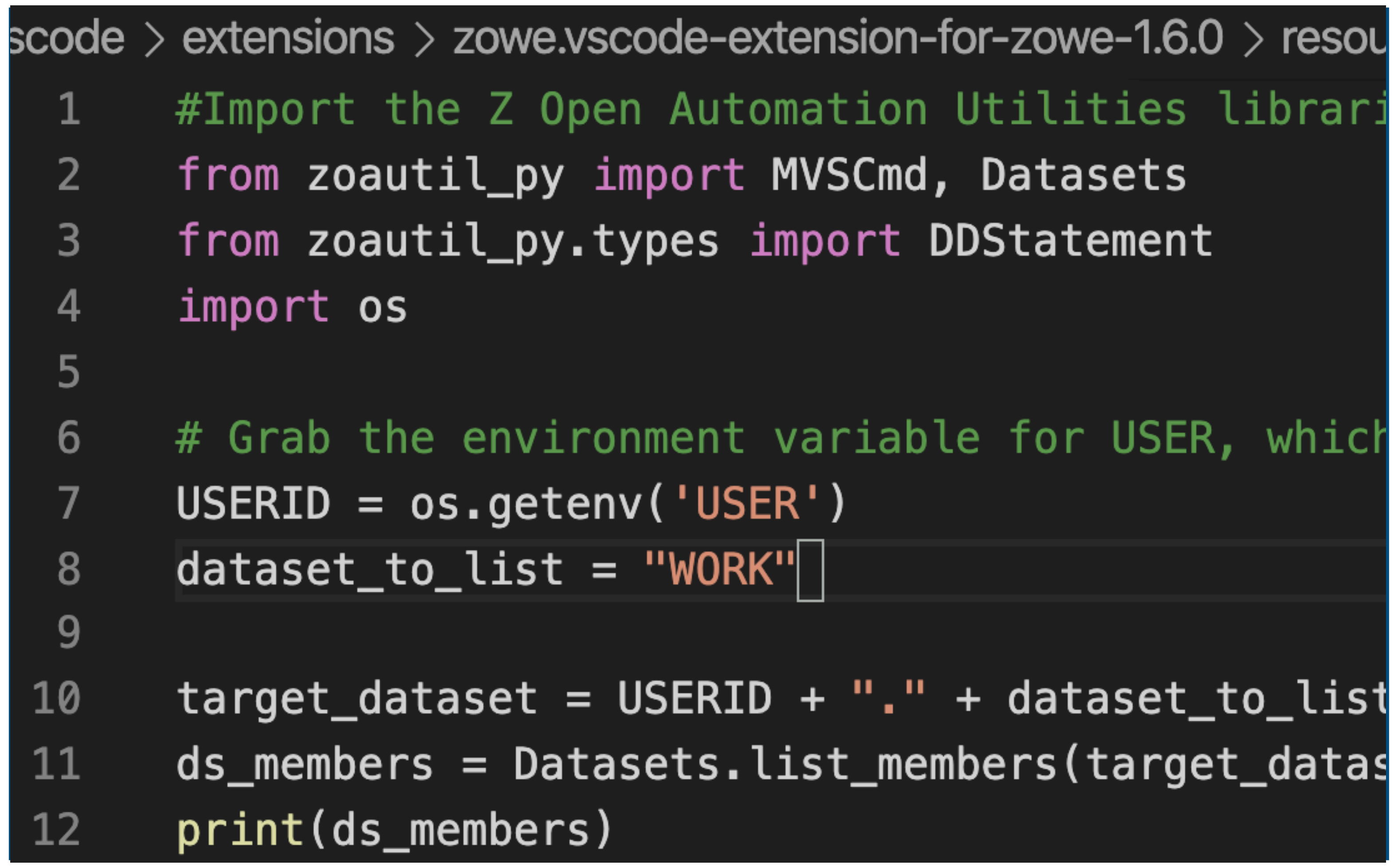
```
%B0008895915315100392Marcellus    Mcc
%B0008895917015330082Cletus       Wat
%B0008896470012039020Nicolas      Del
%B0008896603025322014Irwin        Cox
```

```
Monorail:~ jbisti$ ssh z99999@192.86.32.153
z99999@192.86.32.153's password:
you have mail in /usr/mail/Z99999.
/z/z99999 > bash
/z/z99999 > 
```

1. SSH INTO WITH YOUR ID

Using whatever method you prefer, connect to your system via SSH. We recommend using the Bash shell once you're there, but there's no hard requirement there.

If you breezed through the first bunch of challenges quickly, don't be surprised if these take you quite a bit longer. They were designed to familiarize you with the fundamentals. It all comes together here, and you're expected to struggle a little bit.

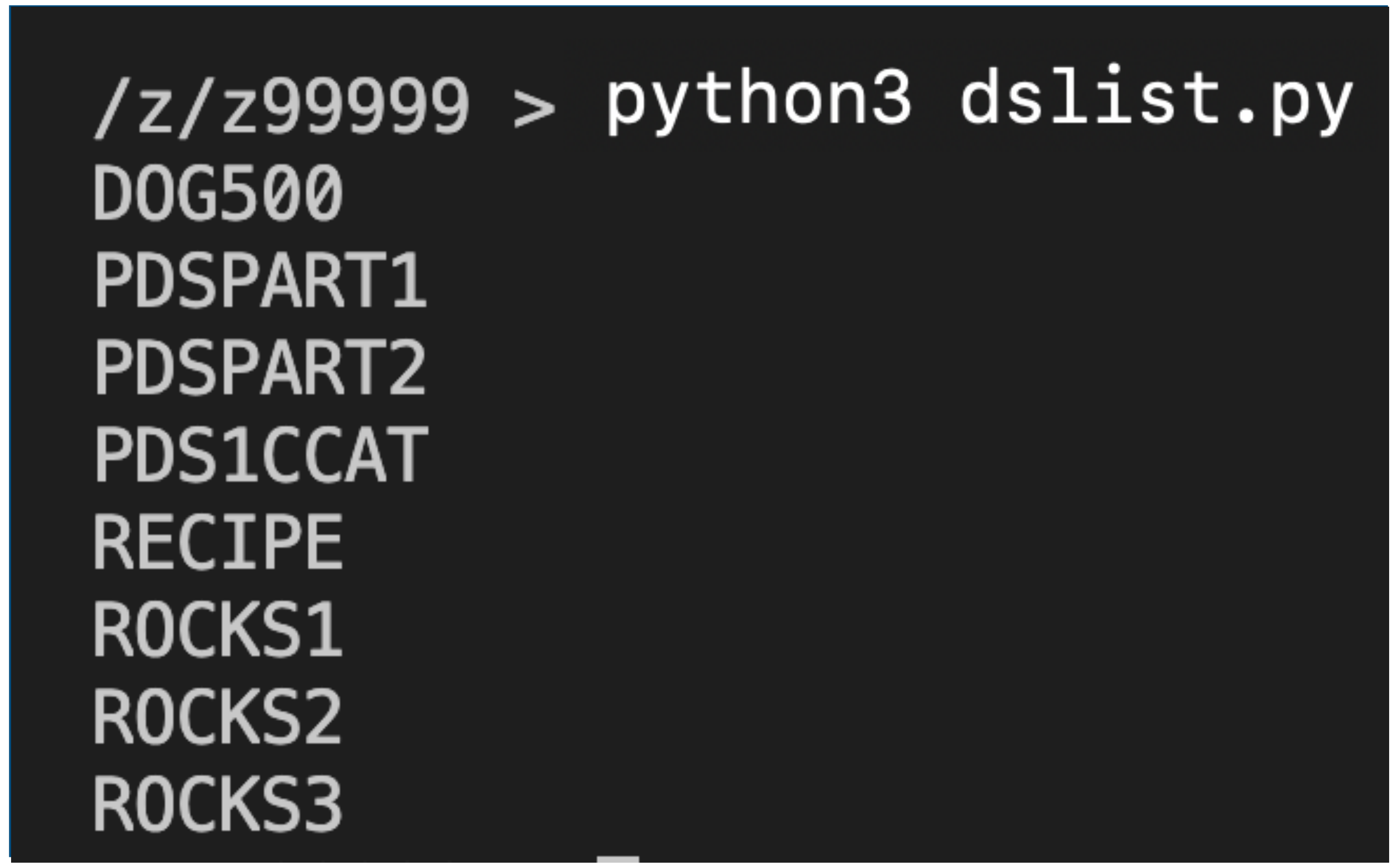


scode > extensions > zowe.vscode-extension-for-zowe-1.6.0 > resou

```
#Import the Z Open Automation Utilities librari
from zoautil_py import MVSCmd, Datasets
from zoautil_py.types import DDStatement
import os

# Grab the environment variable for USER, which
USERID = os.getenv('USER')
dataset_to_list = "WORK"

target_dataset = USERID + "." + dataset_to_list
ds_members = Datasets.list_members(target_datas
print(ds_members)
```

2. LOOK AT dslist.py

Using whatever method you like, check out the dslist.py file in your home directory on the mainframe system. The py suffix tells you this is a Python file, and VS Code may ask if you'd like to install some specialized Python plugins. It's up to you, but we will proceed without requiring the installation of anything additional. Read the code and notice how we import the zoautil packages and set the USERID. Lots of new tricks happening in here.



```
/z/z99999 > python3 dslist.py
DOG500
PDSPART1
PDSPART2
PDS1CCAT
RECIPE
ROCKS1
ROCKS2
ROCKS3
```

3. RUN THE PROGRAM

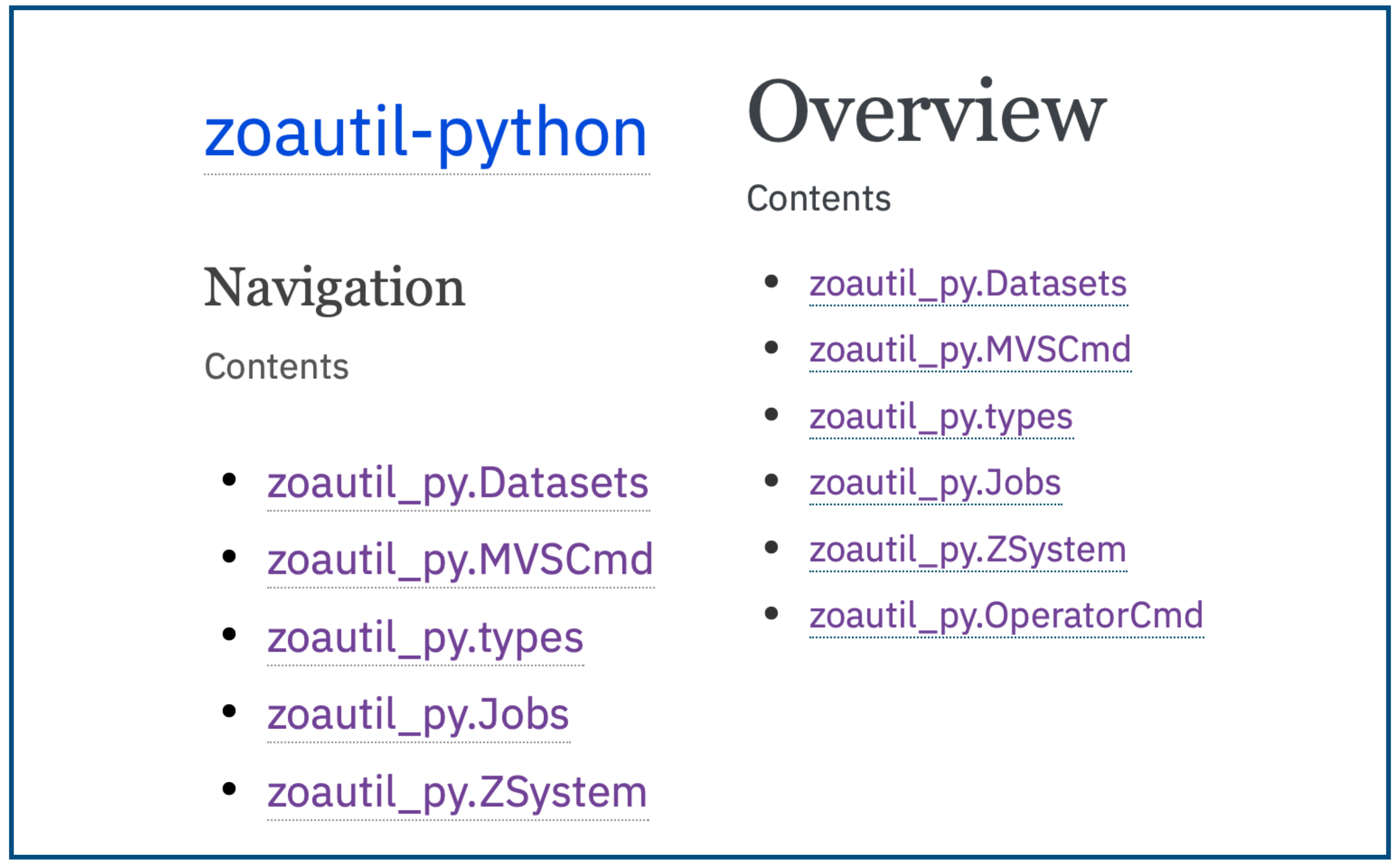
Run the program with **python3 dslist.py**

It will think for a bit so be patient. The output should look similar to what you see above.

This is a very simple example of how the Z Open Automation Utilities can be used in Python on IBM Z.

Want to talk? Join our Slack
ibm.biz/mtm_slack

Tweet about it!
#MasterTheMainframe



zoautil-python

Navigation

Contents

- zoautil_py.Datasets
- zoautil_py.MVSCmd
- zoautil_py.types
- zoautil_py.Jobs
- zoautil_py.ZSystem

Overview

Contents

- zoautil_py.Datasets
- zoautil_py.MVSCmd
- zoautil_py.types
- zoautil_py.Jobs
- zoautil_py.ZSystem
- zoautil_py.OperatorCmd

4. LOOK AT THE COMMANDS

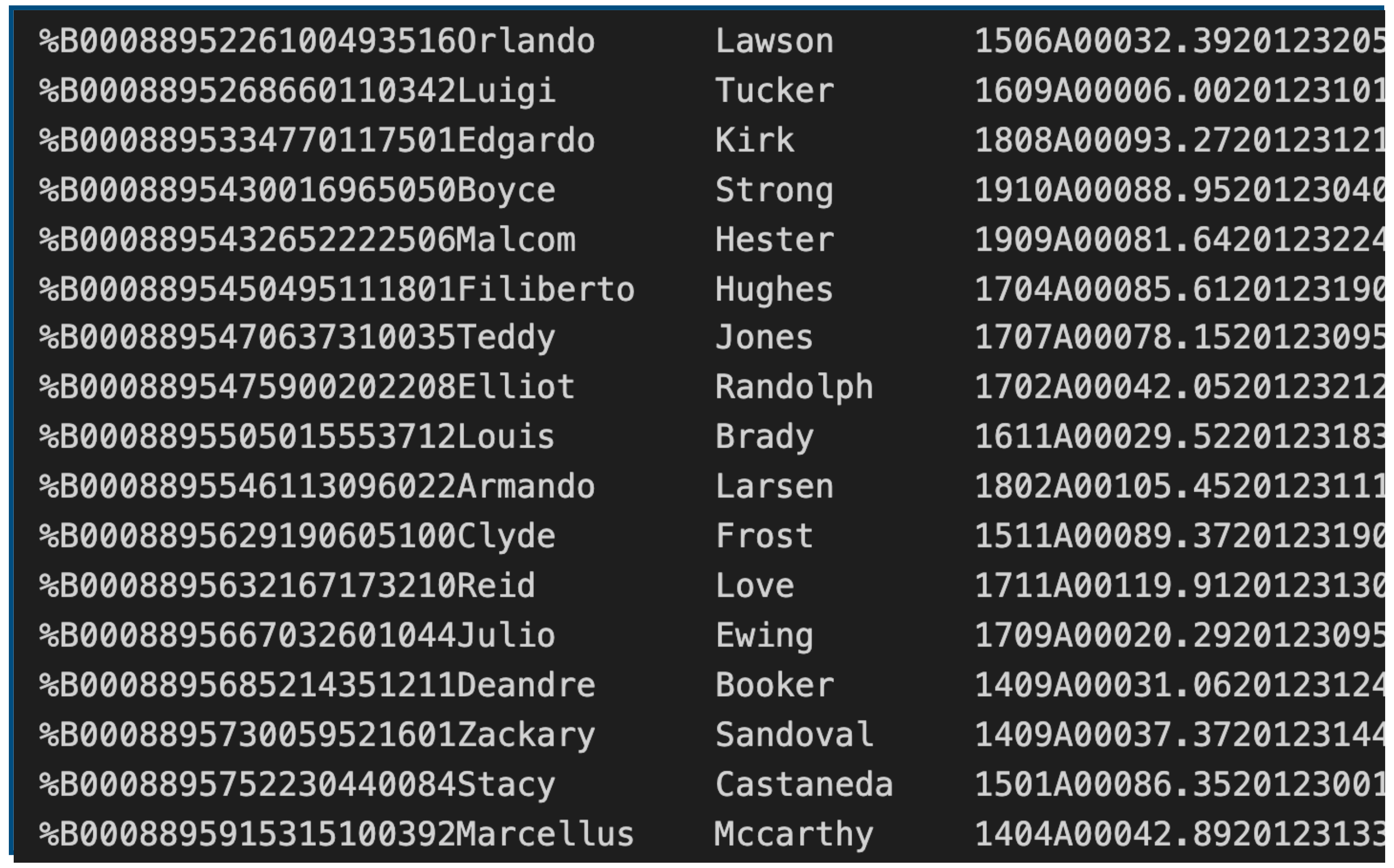
Read more about the commands available through Z Open Automation Utilities in Python here: https://www.ibm.com/support/knowledgecenter/SSKFYE_1.0.3/python_doc_zoautil/index.html?view=embed

If you've never seen Python before, and want to get a handle on the basics, https://pythonbasics.org has great instructions and examples. For this challenge, you'll only really need from the beginning to the end of the "Data and Operations" section.

"HOW DO Z OPEN AUTOMATION UTILITIES WORK ON Z?

The whole point of Z Open Automation Utilities is making it so we can issue z/OS commands, which are typically done through JCL or a command prompt, in a new, more easily scriptable format. Prior to the Z Open Automation Utilities, a System Programmer might write their own programs just to gather information from z/OS, and then relay it back to USS. These utilities offer official, more efficient methods of performing the same tasks and getting the same information.

Additionally, they also provide a very useful interface for automation from utilities like Ansible, which we cover in much greater detail in Part 3. So the utilities are not so much about what you can do, but about the new ways in which you can do it.

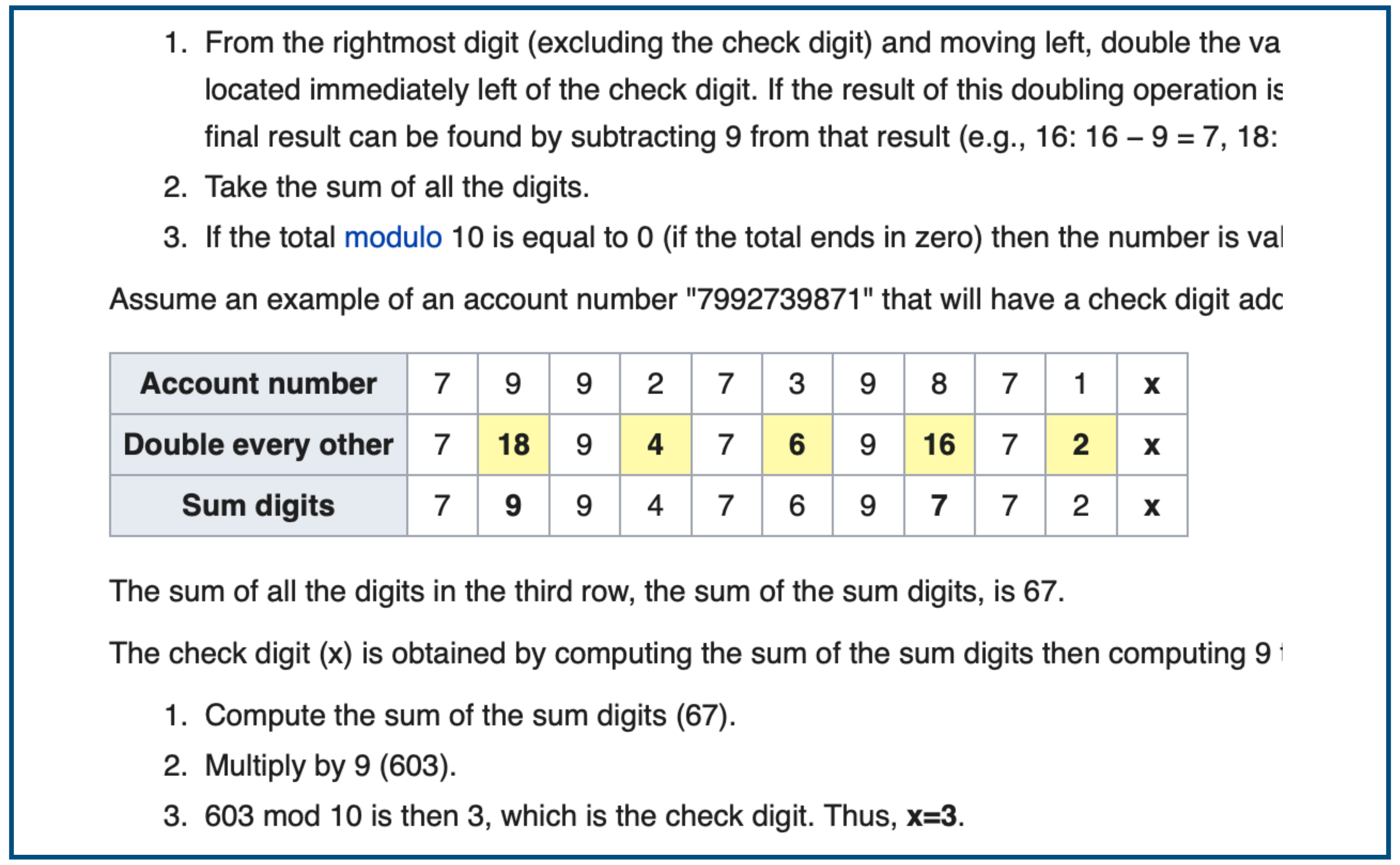


```
%B00088952261004935160rlando      Lawson      1506A00032.3920123205
%B0008895268660110342Luigi        Tucker      1609A00006.0020123101
%B0008895334770117501Edgardo      Kirk        1808A00093.2720123121
%B0008895430016965050Boyce        Strong      1910A00088.9520123040
%B0008895432652222506Malcom       Hester      1909A00081.6420123224
%B0008895450495111801Filiberto    Hughes      1704A00085.6120123190
%B0008895470637310035Teddy        Jones       1707A00078.1520123095
%B0008895475900202208Elliot       Randolph    1702A00042.0520123212
%B0008895505015553712Louis        Brady       1611A00029.5220123183
%B0008895546113096022Armando      Larsen      1802A00105.4520123111
%B0008895629190605100Clyde        Frost       1511A00089.3720123190
%B0008895632167173210Reid         Love        1711A00119.9120123130
%B0008895667032601044Julio        Ewing       1709A00020.2920123095
%B0008895685214351211Deandre      Booker      1409A00031.0620123124
%B0008895730059521601Zackary      Sandoval    1409A00037.3720123144
%B0008895752230440084Stacy        Castaneda   1501A00086.3520123001
%B0008895915315100392Marcellus    Mccarthy    1404A00042.8920123133
```

5. TAKE A SWIPE AT THIS FILE

Look at **MTM2020.PUBLIC.CUST16**

It's full of data, but what exactly? This is generated (not real) magnetic stripe data that you might find on a credit card, if it doesn't have a chip. Want some help decoding it? Take a look at Track 1, Format B in the ISO/IEC 7813 format, spelled out here: https://en.wikipedia.org/wiki/ISO/IEC_7813



1. From the rightmost digit (excluding the check digit) and moving left, double the va located immediately left of the check digit. If the result of this doubling operation is final result can be found by subtracting 9 from that result (e.g., 16: 16 – 9 = 7, 18:
2. Take the sum of all the digits.
3. If the total modulo 10 is equal to 0 (if the total ends in zero) then the number is val

Assume an example of an account number "7992739871" that will have a check digit adc

Account number	7	9	9	2	7	3	9	8	7	1	**x**
Double every other	7	**18**	9	**4**	7	**6**	9	**16**	7	**2**	**x**
Sum digits	7	**9**	9	4	7	6	9	**7**	7	2	**x**

The sum of all the digits in the third row, the sum of the sum digits, is 67.

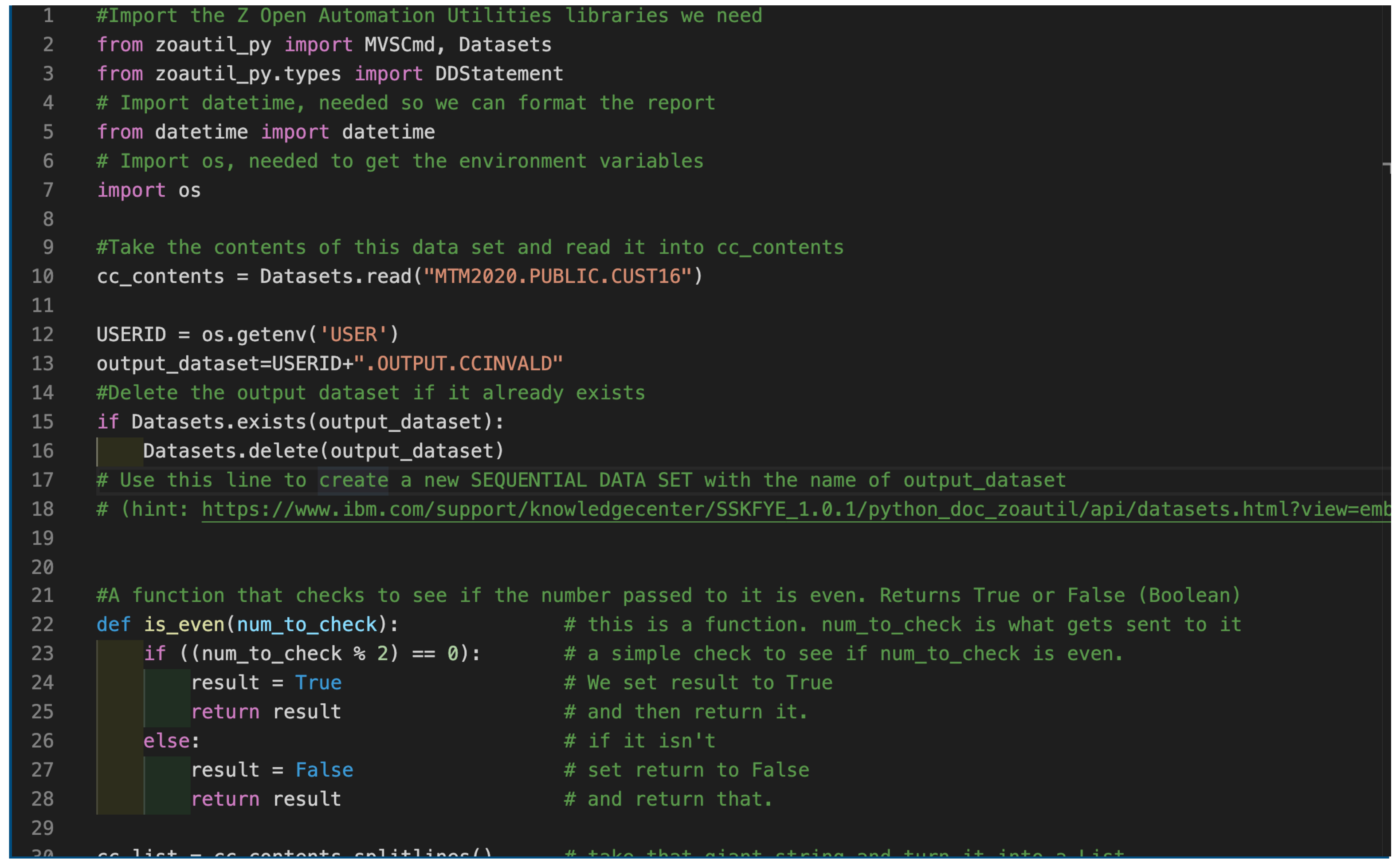
The check digit (x) is obtained by computing the sum of the sum digits then computing 9

1. Compute the sum of the sum digits (67).
2. Multiply by 9 (603).
3. 603 mod 10 is then 3, which is the check digit. Thus, **x=3**.

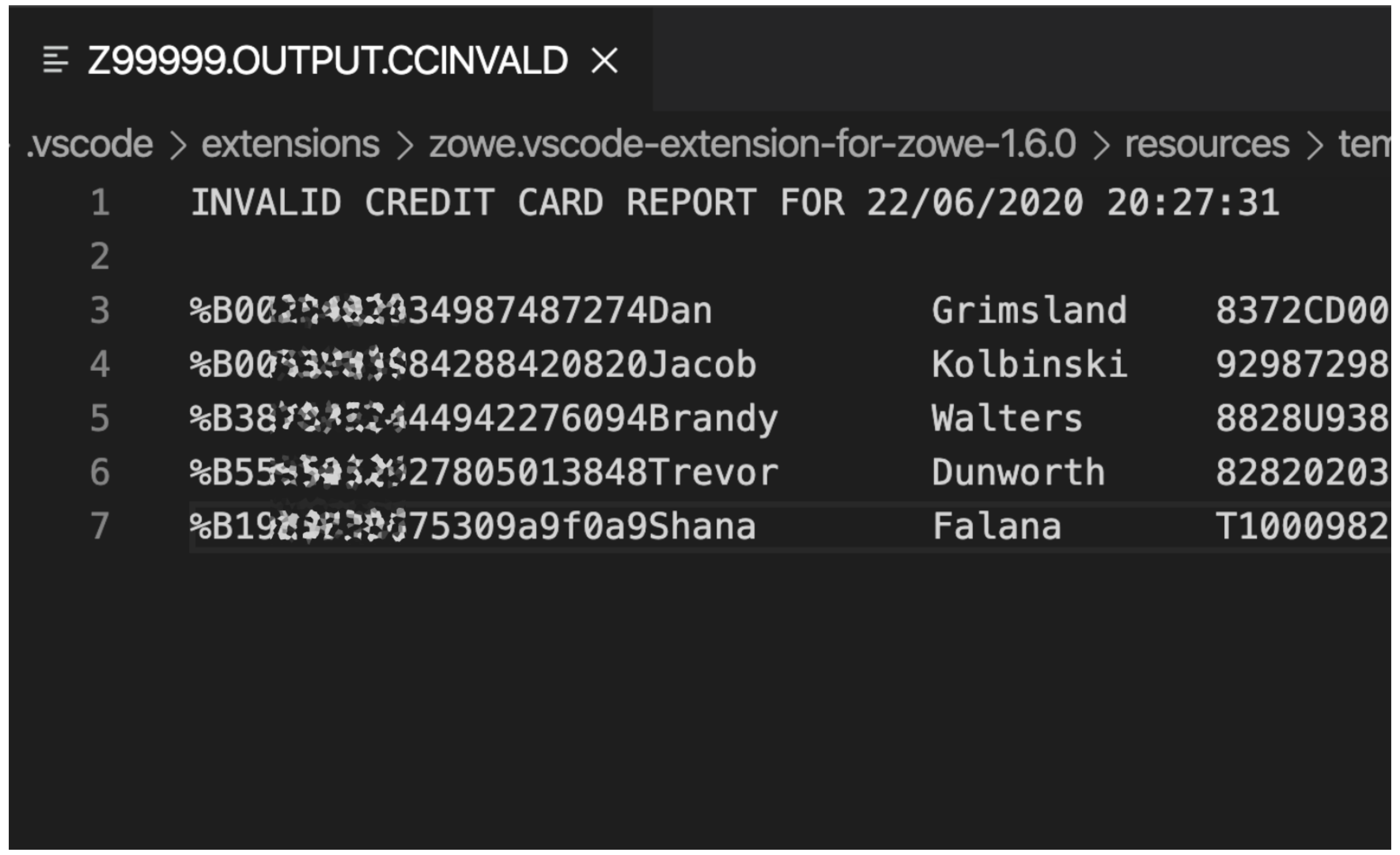
6. LEARN THE LUHN

The Luhn Algorithm is an efficient method of checking if a credit card number is valid, locally, without needing to have the bank or financial institute process it. This way, cards can be checked directly on a web page for mistakes in typing or copying digits. Basically, it all comes down to the last digit, known as the check digit. If it doesn't match what the algorithm calculates, it's an invalid number.

It's a fairly simple check, and has many implementations.

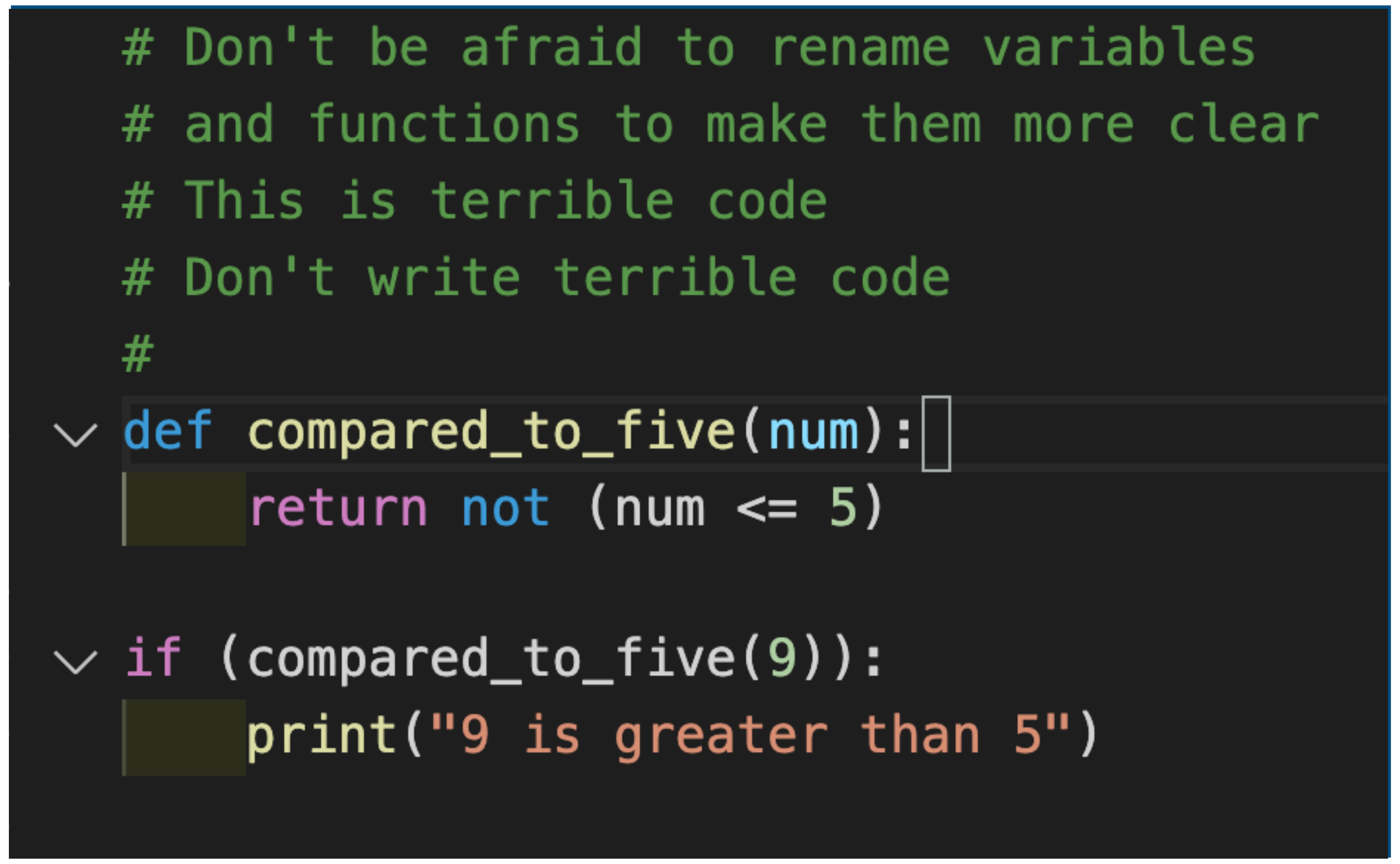


```python
#Import the Z Open Automation Utilities libraries we need
from zoautil_py import MVSCmd, Datasets
from zoautil_py.types import DDStatement
# Import datetime, needed so we can format the report
from datetime import datetime
# Import os, needed to get the environment variables
import os

#Take the contents of this data set and read it into cc_contents
cc_contents = Datasets.read("MTM2020.PUBLIC.CUST16")

USERID = os.getenv('USER')
output_dataset=USERID+".OUTPUT.CCINVALD"
#Delete the output dataset if it already exists
if Datasets.exists(output_dataset):
    Datasets.delete(output_dataset)
# Use this line to create a new SEQUENTIAL DATA SET with the name of output_dataset
# (hint: https://www.ibm.com/support/knowledgecenter/SSKFYE_1.0.1/python_doc_zoautil/api/datasets.html?view=emb


#A function that checks to see if the number passed to it is even. Returns True or False (Boolean)
def is_even(num_to_check):              # this is a function. num_to_check is what gets sent to it
    if ((num_to_check % 2) == 0):       # a simple check to see if num_to_check is even.
        result = True                   # We set result to True
        return result                   # and then return it.
    else:                               # if it isn't
        result = False                  # set return to False
        return result                   # and return that.

cc_list = cc_contents.splitlines()      # take that giant string and turn it into a list
```



Z99999.OUTPUT.CCINVALD ×
.vscode > extensions > zowe.vscode-extension-for-zowe-1.6.0 > resources > tem

```
INVALID CREDIT CARD REPORT FOR 22/06/2020 20:27:31

%B00[redacted]34987487274Dan          Grimsland     8372CD00
%B00[redacted]84288420820Jacob        Kolbinski     92987298
%B38[redacted]44942276094Brandy       Walters       8828U938
%B55[redacted]27805013848Trevor       Dunworth      82820203
%B19[redacted]75309a9f0a9Shana        Falana        T1000982
```



```python
# Don't be afraid to rename variables
# and functions to make them more clear
# This is terrible code
# Don't write terrible code
#
def compared_to_five(num):
    return not (num <= 5)

if (compared_to_five(9)):
    print("9 is greater than 5")
```

7. MODIFY cc_check.py

Look at **cc_check.py**. It was supposed to be a program that finds invalid credit card numbers in a big list, but the programmer got busy and never finished it. Oh no! Right now, all it does is check to see if a portion of the credit card number is even or odd, which is nice, but not really all that helpful.

Edit this program so that instead of locating the odd numbers, it implements the Luhn algorithm to find the six invalid entries. **Note: Use the sample code in luhn.py we provide you to perform the checking logic.** You'll need to implement that logic in your cc_check.py program. There are other implementations out there that might not give you the correct answers.

This challenge is noticeably more difficult than others you've solved, but it gives you a taste of what's to come in Level 3, so you'll know what you're up against in the next challenges.

8. MIND YOUR OUTPUT

Your program needs to output the invalid entries to a data set member, written out using the Z Open Automation Utility methods. There are LOTS of comments to help you out.

When completed, your output will look very somewhat similar to what's in the above screenshot, but with different names and numbers. (We made this screenshot just as an example)

9. SOME HINTS

1) We're using fixed-record-length data. Make sure you're looking at the right span of digits.
2) Figure out if the algorithm is looking for valid or invalid numbers. You can flip logic easily by putting **not** before a boolean. For example: **Return not True** will return **False**
3) Take your time, use the internet, perform sanity checks. All the pieces are here, you just have to put them together.

We're going to use the **CHK2** job for the ZOAU2 challenges, so submit *that* job in MTM2020.PUBLIC.JCL to check your work.

NICE JOB! LET'S RECAP

Way to finish strong! You've accomplished incredible feats, and leveraged some fairly new IBM Z features to get here. Finishing this task shows that you've got what it takes to couple z/OS data set commands with Python code by way of Z Open Automation Utilities. After submitting your work and verifying completion, take a moment to reflect, brag, or even just take a victory lap around the room. You're on track to make IBM Z an everyday skill.

NEXT UP…

With Part 2 complete, you can dive right into Part 3, or if anything here caught your eye, take this time to go back and explore it further. Don't worry about any changes marking completed tasks incomplete, that's a one-way toggle. Now go do great things.

Want to talk? Join our Slack
ibm.biz/mtm_slack

Tweet about it!
#MasterTheMainframe