INTRODUCTION

https://cryptohack.org/challenges/introduction/

Challenge: Great Snakes

```
#!/usr/bin/env python3

import sys
# import this

if sys.version_info.major == 2:
    print("You are running Python 2, which is no longer supported. Please update to Pytho

ords = [81, 64, 75, 66, 70, 93, 73, 72, 1, 92, 109, 2, 84, 109, 66, 75, 70, 90, 2, 92, 79

print("Here is your flag:")
print("".join(chr(o ^ 0x32) for o in ords))
```

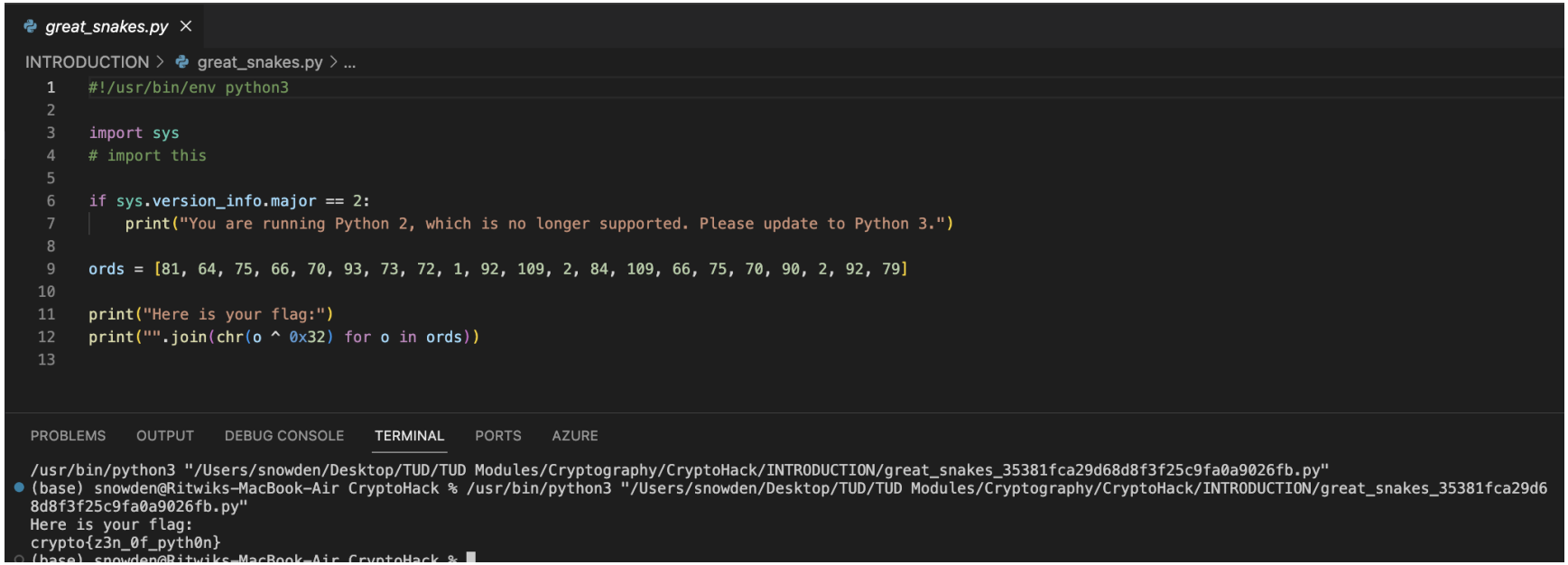
Explanation:

Certainly! This Python code appears to be a simple script that prints out a flag by performing a bitwise XOR operation on a list of integer values and converting the result to characters. Let's break down the code step by step:

1. `#!/usr/bin/env python3`: This line is called a shebang line and is used to specify the interpreter that should be used to execute the script. In this case, it specifies that the script should be executed using Python 3.
2. `import sys`: This line imports the `sys` module, which provides access to various system-specific parameters and functions.
3. `if sys.version_info.major == 2:`: This conditional statement checks the major version of Python that is running the script. If it detects that the major version is 2 (i.e., the script is running on Python 2), it prints a message informing the user that Python 2 is no longer supported and advises them to update to Python 3.
4. `ords = [81, 64, 75, 66, 70, 93, 73, 72, 1, 92, 109, 2, 84, 109, 66, 75, 70, 90, 2, 92, 79]`: This line defines a list called `ords` that contains a series of integer values. These integers will be used in the next step to generate the flag.
5. `print("Here is your flag:")`: This line simply prints a message to inform the user that the flag is about to be displayed.
6. `print("".join(chr(o ^ 0x32) for o in ords))`: This line performs the main operation to generate the flag. It uses a list comprehension to iterate over each integer `o` in the `ords` list, and for each integer, it calculates the result of performing a bitwise XOR operation (`^`) with the integer `0x32` - a hexadecimal number (which is equivalent to the decimal value 50).
    - `chr(o ^ 0x32)` converts the XOR result back to a character.
    - `" ".join(...)` joins all the characters generated from the XOR operation into a single string with no spaces between them.

    The result of this operation is the flag, which is printed to the console.

To summarize, this code checks the Python version and warns the user if they are using Python 2. Then, it performs a bitwise XOR operation on a list of integers to generate a flag and prints the flag to the console.



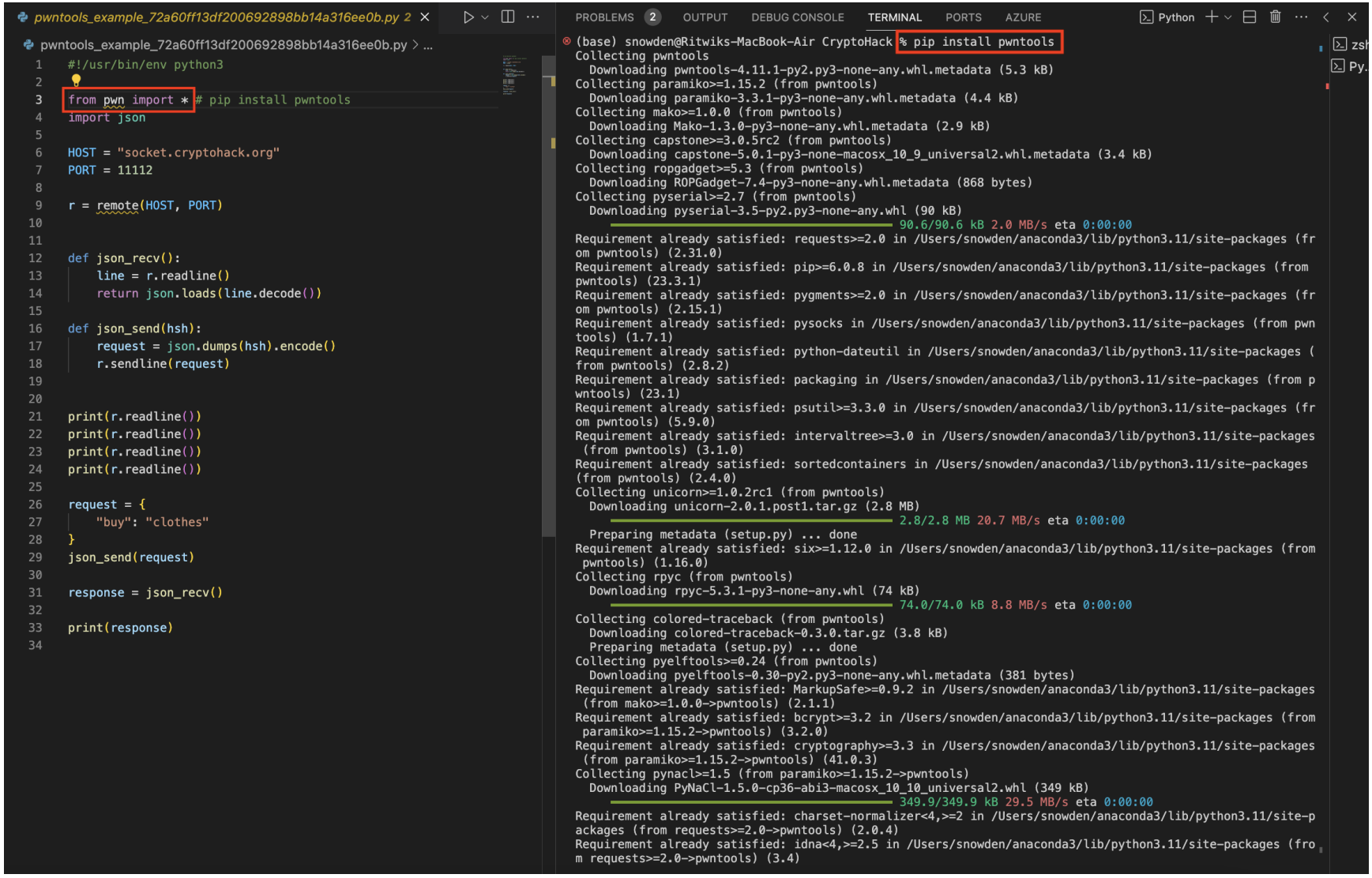
FLAG: **crypto{z3n_0f_pyth0n}**

Challenge: Network Attacks

Several of the challenges are dynamic and require you to talk to our challenge servers over the network. This allows you to perform man-in-the-middle attacks on people trying to communicate, or directly attac a vulnerable service. To keep things consistent, our interactive servers always send and receive JSON objects.

Such network communication can be made easy in Python with the `pwntools` module. This is not part of the Python standard library, so needs to be installed with pip using the command line `pip install pwntools`.

For this challenge, connect to socket.cryptohack.org on port 11112. Send a JSON object with the key ‘buy’ and value ‘flag’.



ERROR! while `pip install pwntools`

To fix this,

Python makes such network communication easy with the ***telnetlib*** module. Conveniently, it's part of Python's standard library, so let's use it for now.

```
#!/usr/bin/env python3

import telnetlib
import json

HOST = "socket.cryptohack.org"
PORT = 11112
tn = telnetlib.Telnet(HOST, PORT)

def readline():
    return tn.read_until(b"\n")

def json_recv():
    line = readline()
    return json.loads(line.decode())

def json_send(hsh):
    request = json.dumps(hsh).encode()
    tn.write(request)

print(readline())
```

```
print(readline())
print(readline())
print(readline())

request = {
    "buy": "flag"
}

json_send(request)
response = json_recv()
print(response)
```

This Python code appears to be a simple client that communicates with a remote server over Telnet using the `telnetlib` library. It sends and receives JSON data to and from the server. Let's break down the code step by step, along with explanations of the functions used:

1. `import telnetlib`: This line imports the `telnetlib` library, which provides functionality for interacting with Telnet servers.
2. `import json`: This line imports the `json` library, which is used for encoding and decoding JSON data.
3. `HOST = "socket.cryptohack.org"` and `PORT = 11112`: These lines define constants `HOST` and `PORT`, which specify the hostname and port number of the Telnet server to connect to.
4. `tn = telnetlib.Telnet(HOST, PORT)`: This line establishes a Telnet connection to the specified `HOST` and `PORT` using the `Telnet` class from the `telnetlib` library. It assigns the connection to the `tn` variable.
5. `def readline():`: This defines a custom function named `readline`. When called, this function reads data from the Telnet connection until it encounters a newline character (`b"\n"`), and then it returns the received line.

6. `def json_recv():`: This defines another custom function named `json_recv`. This function reads a line from the Telnet connection using the `readline` function and then decodes the received line as JSON data using the `json.loads` function. It returns the decoded JSON object.
7. `def json_send(hsh):`: This function takes a dictionary `hsh` as input. It converts the dictionary to a JSON-formatted string using `json.dumps`, encodes the string to bytes, and sends it over the Telnet connection using `tn.write`.
8. The following lines of code call the `readline` function four times to read and print four lines of text from the Telnet server. These lines serve as initial communication with the server.
9. `request = {"buy": "clothes"}`: This line defines a Python dictionary named `request` with a single key-value pair. This dictionary will be sent to the server as a JSON request.
10. `json_send(request)`: This line sends the `request` dictionary to the server by calling the `json_send` function, which serializes the dictionary to JSON and sends it over the Telnet connection.
11. `response = json_recv()`: This line receives a response from the server by calling the `json_recv` function, which reads a line from the server, decodes it as JSON, and assigns the resulting JSON object to the `response` variable.
12. `print(response)`: Finally, the code prints the received JSON response from the server, which should contain data related to the request made in step 10.

In summary, this Python script establishes a Telnet connection to a remote server, exchanges JSON data with the server, and prints the server's responses. The `readline`, `json_recv`, and `json_send` functions are used to simplify the process of reading and sending JSON data over the Telnet connection.

Note:

In Python, `b"\n"` represents a newline character encoded as bytes. The `b` prefix before a string in Python indicates that it is a bytes literal. The `"\n"` within the bytes literal represents the newline character.

The newline character (`"\n"`) is a special character used to represent the end of a line in text files and strings. When it is encountered in a string, it typically signals the beginning of a new line in a multiline text.

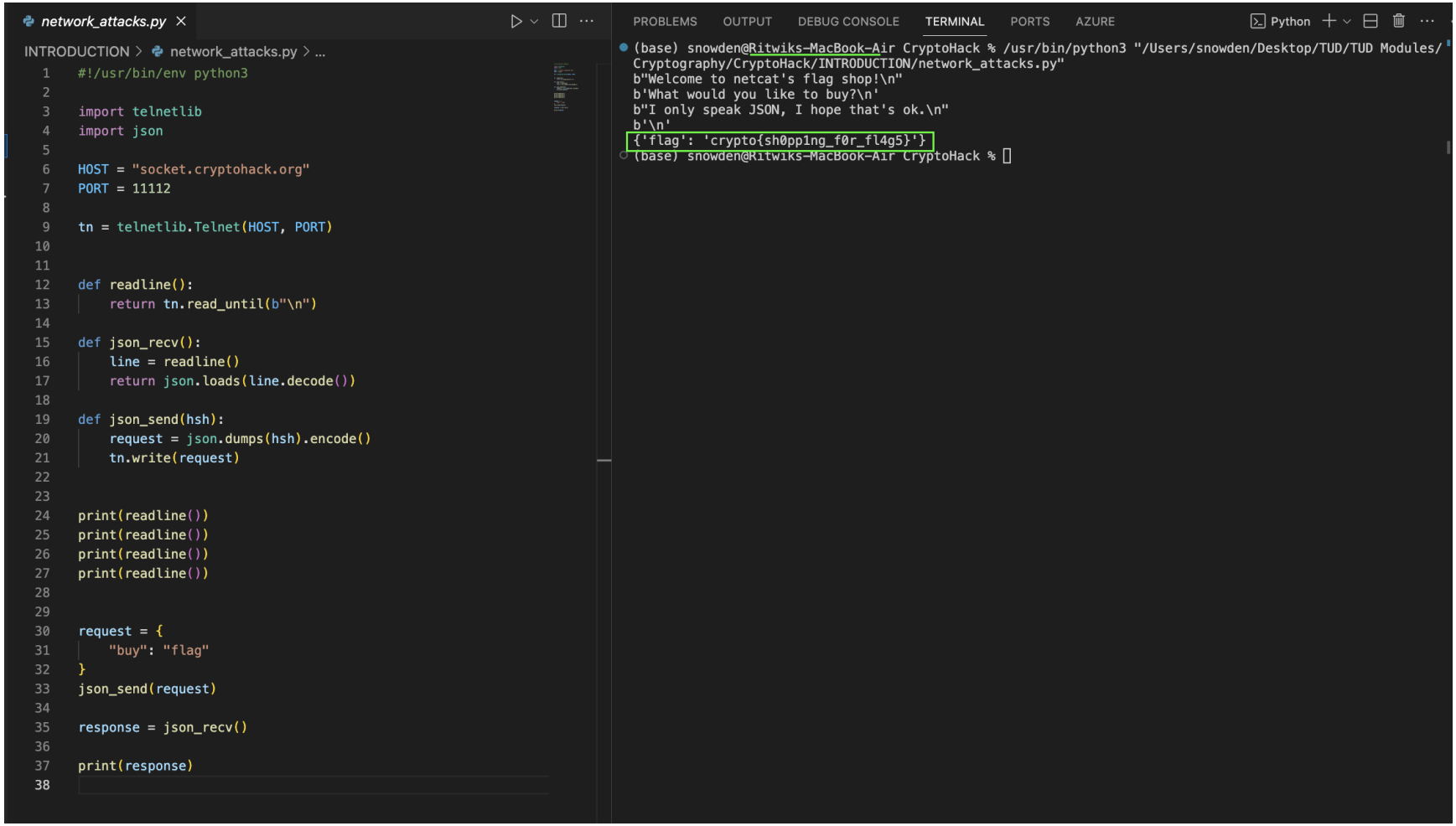
Here's how it works:

- In regular string literals (e.g., `"Hello\nWorld"`), `"\n"` is interpreted as a newline character, and it causes the text to break to a new line. So, if you print this string, it will be displayed as:

```
Hello
World
```

- In bytes literals (e.g., `b"Hello\nWorld"`), `"\n"` is also a newline character, but it's represented as bytes rather than a regular string. Bytes literals are used when dealing with binary data or when you need to represent characters using their raw byte values. For example, if you were working with binary data and wanted to include a newline character, you would use `b"\n"`.

In the context of the code you provided, `b"\n"` is used to specify the newline character as bytes because the `telnetlib` library deals with data transmission over a network connection, and network data is typically represented as bytes. When reading data from a network connection, `b"\n"` is used to detect the end of a line or a message.



FLAG: **crypto{sh0pp1ng_f0r_fl4g5}**