Exercises

Exercise 1: Variables, Strings, Lists

Write a program that:

- Takes the first two command line arguments (excluding the filename). Both of them will be strings, and both of them will be long enough that you will not get index errors.
- Print those two arguments, separated by a space.
- Prints a string consisting of the first character of the first argument, followed by the third character of the second argument, followed by the last character of the second argument, followed by the length of the first argument.
- Prints two empty lines.
- Prints the total number of arguments (including file name).
- Print the 2nd - 4th characters of the first argument.
- Prints the phrase "use of 'quotation' marks".
- Takes input from the user with the prompt "please enter: " and prints the second character of the user input.

```
william@pricard:~/Master/python-projectv1/solutions$ python3 exercise1.py Hello Pythonistas Welcome to Class
Hello Pythonistas
Hts5



6
ell
use of 'quatation' marks
Please enter: This is a statement
h
william@pricard:~/Master/python-projectv1/solutions$
```

Exercise 2: Attention to Detail

Create a program that takes (at least) three command line arguments. The first two will be integers and the third will be a float.

- On one line, separated by spaces, print the sum of the first two arguments, the product of the first and third arguments, the first argument modulo the second, and the integer quotient of the first by the third.
- Add 1 to all three arguments.
- On a new line, print the first argument bitwise left shift 3, the second argument divided by 2 (not integer division), and the bitwise OR (operator) of the first and second arguments. (all separated by spaces.)
- On the last line, print the sum of the first argument (after the addition) and the total number of arguments, excluding the program name.
- Make sure the output numbers that are floats are at two digits of precision.

```
william@pricard:~/Master/python-projectv1/solutions$ python3 exercise2.py 5 10 3.14
15 15.70 5 1
48 5.5 15
9
william@pricard:~/Master/python-projectv1/solutions$
```

Exercise 3: Conditional Statements

Create a program that takes (at least) three command line arguments. The first two will be integers and the third will be a string.

- Prints the larger of the two integers. If they are equal, do not print either one.
- If the word "time" appears in the string, print the sum of the integers.
- If both the integers are odd, or one of them is a multiple of 3, print the string.
- If there are more than three command line arguments (excluding the filename), print the word "error".

```
william@pricard:~/Master/python-projectv1/solutions$ python3 exercise3.py 5 15 sometime
15
20
sometime
william@pricard:~/Master/python-projectv1/solutions$ python3 exercise3.py 5 15 sometimes there are more
15
20
sometimes
error
william@pricard:~/Master/python-projectv1/solutions$
```

Exercise 4: Can You Float This For Me?

Write a program that has a function named **float_total** that:

- Will take a comma separated string of floats
- Return a list of the floats

Exercise 5: Can You Change That To Numbers

Write a program that has a function named **numbers** that:

- Takes a string argument
- Returns a list of the ordinal numbers for each charater of the string

Exercise 6: Loops

Create a program that takes one integer command line argument.

- Ensure the argument is an integer, if not the program doesn't do anything.
- For each number between 1 and the argument (inclusive), print the word "triangle" if the number is a multiple of three.
- Print "square" if it is a multiple of four.
- If it is a multiple of 12, do not print triangle or square. Instead, print "x dozen" where x is the numeral representing how many dozen you're at("1 dozen" for 12, "2 dozen" for 24, etc.)
- Lastly, print the sum of the numbers between 1 and the argument (inclusive).

```
william@pricard:~/Master/python-projectv1/solutions$ python3 exercise6.py 26
triangle
square
triangle
square
triangle
1 dozen
triangle
square
triangle
square
triangle
2 dozen
351
william@pricard:~/Master/python-projectv1/solutions$
```

Exercise 7: Functions

Create a program that has a function called **factorial** that takes one integer argument.

- Ensure the argument is an integer, if not the program doesn't do anything.
- Calculate the factorial of the number. A factorial is the product of a whole number and all the positive whole numbers beneath it.
- For examp[illegible] 4x3x2x1 = 24.
- The functic[illegible]2x1 = 24"

```
william@pricard:~/Master/python-projectv1/solutions$ python3 exercise7.py 15
15x14x13x12x11x10x9x8x7x6x5x4x3x2x1 = 1307674368000
william@pricard:~/Master/python-projectv1/solutions$ python3 exercise7.py 16
16x15x14x13x12x11x10x9x8x7x6x5x4x3x2x1 = 20922789888000
william@pricard:~/Master/python-projectv1/solutions$
```

Exercise 8: Better Than Average

Write a program that has a function named **average** that:

- Takes a variable length argument list
- Returns the average for the arguments as a float

Exercise 9: It's The End That Is Important

Write a program that has a function named **end** that:

- Takes two arguments; a list of items and an integer number
- Return a new list containing the last number of items in the list

Exercise 10: Say It A Little Slower, In Sorted Order

Write a program that has a function named **slower** that:

- Takes a string argument
- Returns a tuple with each element in the string as an element in the tuple.
- The elements should be sorted.

Exercise 11: Fibonacci Functions

Create a program that has two functions, **which_fibonacci** and **next_fibonacci**.

- **which_fibonacci** takes a non-negative integer as an argument, which may or may not be a Fibonacci number. If it is, then it returns its position in the sequence. For example, 2 is the 4th Fibonacci number. If 1 is given as the argument, the function should return 2. If the argument is not a Fibonacci number, the function returns 0.
- **next_fibonacci** takes a non-negative integer and returns the next largest Fibonacci number. If the argument is itself a Fibonacci number, it should return the next Fibonacci number, not the argument itself. If 1 is received, **next_fibonacci** should return 2. If the argument is not a fibonacci number, it should return the next fibonacci number after the argument.

```
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise11.py 610
15
987
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise11.py 609
0
610
```

Exercise 12: How Many Characters?

Create a program that has a function called counts that:

- counts all of the occurring characters in a string (UTF-8)
- if you have a string like "aba", the result should be {"a":2, "b":1}
- If the string is empty, return {}

```
william@pricard:~/Master/python-projectv1/solutions$ python3 exercise12.py aba
{'a': 2, 'b': 1}
william@pricard:~/Master/python-projectv1/solutions$ python3 exercise12.py
{}
william@pricard:~/Master/python-projectv1/solutions$ python3 exercise12.py "Four score and seven years ago our fathers brought forth"
{'F': 1, 'o': 6, 'u': 3, 'r': 7, ' ': 9, 's': 4, 'c': 1, 'e': 5, 'a': 4, 'n': 2, 'd': 1, 'v': 1, 'y': 1, 'g': 2, 'f': 2, 't': 3, 'h': 3, 'b
': 1}
william@pricard:~/Master/python-projectv1/solutions$
```

Exercise 13: File & Dictionary Manipulation

reate a program that:

- Reads a filename from the command line.
- Determines the most common letter in the file that is not the white space characters, and prints what it is and how many times it occurs. Use the format " is the most common letter. It occurs _ times." Replace _ with the appropriate letter (uppercase) and number.
- Determines what percentage of the number of words in the file is the word "the"; print the integer that is closest to this percentage, rounding down. Ignore capitalization: "The" and "the" are the same word.
- Writes the first ten words of the file (as determined by whitespace) to a new file named "Exercise_8_output.txt". Assume the file will be written to the same directory where exercise8.py is located.

```
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise13.py large.txt
e is the most common letter.  It occurs 74433 times.
'The' is 9889 of 144833 words or 6.83%.
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise13.py small.txt
e is the most common letter.  It occurs 308 times.
'The' is 25 of 647 words or 3.86%.
```

Exercise 14: What Is The Total?

Write a program that has a function named **total** that:

- Takes two arguments; a dictionary and a list of tuples
- The dictionary will contain items as keys and the price as the values
- Each tuple will contain the item name and quantity requested
- Return the t

Exercise 15: Which Line?

Write a program that has a function named **longest** that:

- Takes a filename as the argument
- Returns a tuple containing the longest line and its length

Exercise 16: Can You Save This For Me?

Write a program that has a function named **ending** that:

- Takes two arguments; a filename and a list
- Write each item in the list to the file, one item per line until the word "end"
- If "end" is not in the list, write the entire list to the file

Exercise 17: First Letter

Create a program that has a function named **first** that:

- Takes a string as an argument.
- Returns the first character that is not repeated anywhere in the string.
- If a string contains all repeating characters, it should return an empty string "".
- Upper and lowercase letters are considered the same character, but the function should return the correct case for the initial letter. For example, the input "sTreSS" should return "T".

```
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise17.py sTress
T
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise17.py abba

william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise17.py Hooahhh
a
william.d.howard36@workstation21:~/python-projectv1/solutions$
```

Exercise 18: Phone Number

Create a program that has a function named **phone** that:

- Accepts an array of 10 integers (between 0 and 9).
- Returns a string of those numbers in the form of a phone number.
- The input could be (1,2,3,4,5,6,7,8,9,0), [1,2,3,4,5,6,7,8,9,0], 1234567890, or 1,2,3,4,5,6,7,8,9,0.

```
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise18.py 1234567890
(123) 456-7890
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise18.py '[1234567890]'
(123) 456-7890
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise18.py '(1234567890)'
(123) 456-7890
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise18.py 1,2,3,4,5,6,7,8,9,0
(123) 456-7890
william.d.howard36@workstation21:~/python-projectv1/solutions$
```

Exercise 19: Classes

Create a program that has a class named **Sphere** that:

- Takes a numeric value (float or int) representing the radius when it is declared.
- **Sphere** should have at least three methods
  - **get_surface_area**, which returns the surface area of the sphere
  - **get_volume**, which returns the volume of the sphere
  - **set_radius**, which changes the radius of the sphere. It should result in new values being returned from the other two methods.
- Feel free to set __init__ or other methods as necessary to accomplish the task.
- To ensure you match the test cases exactly, use 3.14 as your approximation for π. The surface area of a sphere is 4*π*r^2; the volume of a sphere is 4/3*π*r^3.
- The program will accept multiple command line arguments.
- When the program runs, it calculates the surface area and volume for the provided numeric value (radius). Then the program adds 1 to the numeric value (radius), sets the new radius, and computes the surface area and volume for the new radius value.
- The program prints out the radius, surface area, and volume for the initial radius and the new radius.

```
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise19.py 1 3.2 4 9
Radius: 1.0     Surface Area: 12.5600   Volume: 4.1867
Radius: 2.0     Surface Area: 50.2400   Volume: 33.4933

Radius: 3.2     Surface Area: 128.6144  Volume: 137.1887
Radius: 4.2     Surface Area: 221.5584  Volume: 310.1818

Radius: 4.0     Surface Area: 200.9600  Volume: 267.9467
Radius: 5.0     Surface Area: 314.0000  Volume: 523.3333

Radius: 9.0     Surface Area: 1017.3600 Volume: 3052.0800
Radius: 10.0    Surface Area: 1256.0000 Volume: 4186.6667
```

Exercise 20: Positive Properties

Create a program that has a class named **Number**, which is initialized with a positive integer **n** as its sole argument. It should have at least four methods:

- **is_prime** returns True if **n** is prime, and False if not. 1 is not prime.
- **get_divisors** returns a list of positive divisors of **n**, in ascending order.
- **get_gcd** takes a positive integer argument, and returns the greatest common divisor of **n** and the argument.
- **get_lcm** takes a positive integer argument, and returns the least common (positive) multiple of n and the argument.

```
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise20.py 5 4
Prime? True
Factors: [1, 5]
GCD 5 and 4: 1
LCM 5 and 4: 20
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise20.py 4 6
Prime? False
Factors: [1, 2, 4]
GCD 4 and 6: 2
LCM 4 and 6: 12
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 exercise20.py 9 15
Prime? False
Factors: [1, 3, 9]
GCD 9 and 15: 3
LCM 9 and 15: 45
```

Exercise 21: Password Checker

Create a program named **password_checker** with a function named **password_checker** that:

- Takes one string argument (password)
- Checks the password for the following rules:
  - Password must be at least 14 characters in length
  - Password must contain at least one character from each of the following four character sets:
    - Uppercase characters (string.ascii_uppercase)
    - Lowercase characters (string.ascii_lowercase)
    - Numerical digits (string.digits)
    - Special characters (string.punctuation)
- Password cannot contain more than three consecutive characters from the same character set.
- Password cannot contain whitespace characters (string.whitespace)
- Returns True if a valid password. False, otherwise.

You will test your program with the testpasswords.pyc:

>>> python3 testpasswords.pyc --module password_checker.py --checker password_checker

```
william.d.howard36@workstation21:~/python-projectv1/solutions$ python3 testpasswords.cpython-38.pyc --module password_checker.py --checker password_checker.py
.....................................sssssssssssssss
----------------------------------------------------------------------
Ran 55 tests in 0.002s

OK (skipped=16)
```

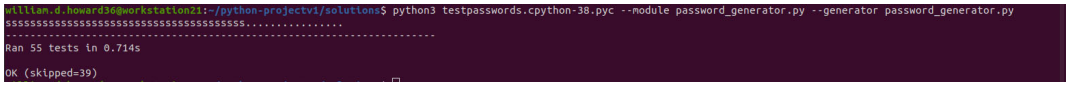
Exercise 22: Password Generator

Create a program named **password_generator** with a function named **password_generator** that:

- The function should have one optional argument for length. The default is 14.
- Generate a password (of the given length) that meets the password requirement from the Password Checker.

Test your program with testpasswords.pyc:

>>> python3 testpasswords.pyc --module password_generator.py --generator password_generator



Exercise 23: Blackjack, Guessing Game, Etc.

Blackjack:

Design an object-oriented program for a simple blackjack game that provides for one player and a dealer (the computer).

- The dealer must continue to take cards until the dealer's hand has at least 17 points.
- Don't allow the player to split the hand.
- Aces can be one or eleven depending on logic in the code.
- If the user keeps asking for a card (hit) and the total score goes over 21, the user busts.
- If the dealer goes over 21, the dealer busts.
- If the dealer and user have the same score, 21 or under, they tie.
- The highest hand without going over 21 wins.
- It is up to the programmer to develop a player class, or not.

Below is a suggested Card and Deck class.

```
class Card():
    def __init__(self, suit, number):
        self.suit = suit
        self.number = number

    def __str__(self):
        return f"{self.number} of {self.suit}"
```

```
class Deck():
    def __init__(self):
        self.cards = []
        self.card_range = [2,3,4,5,6,7,8,9,10,"Jack","Queen","King","Ace"]
        self.suits = ["Diamonds", "Spades", "Hearts", "Clubs"]
        self.__build()

    def __build(self):
        for s in self.suits:
            for c in self.card_range:
                self.cards.append(Card(s,c))

    def shuffle(self):
        random.shuffle(self.cards)

    def deal_a_card(self):
        return self.cards.pop()
```

99 BOTTLES:

Design a program that mimics the "99 bottles of beer" nursery rhyme. The program will iterate through the rhyme printing the verses and iterating from 99 to zero.

- add the --w option that forces the program to print out the numbers as words. EX: 99 --> ninety-nine >>> python3 bottles.py --w
- add the --n option to allow the user to select a different number between 99 and 1. >>> python3 bottles.py --n 50
- add the --b option that allows the user to choose a different beverage than beer. EX: coke >>> python3 bottles.py --b coke

NORMAL OUTPUT

```
4 bottles of beer on the wall!
4 bottles of beer
Take one down
And pass it around
3 bottles of beer on the wall!

3 bottles of beer on the wall!
3 bottles of beer
Take one down
And pass it around
2 bottles of beer on the wall!

2 bottles of beer on the wall!
2 bottles of beer
Take one down
And pass it around
1 bottles of beer on the wall!

1 bottle of beer on the wall!
1 bottle of beer
Take one down
And pass it around
No more bottles of beer on the wall!
```

WORDS OUTPUT

```
Six bottles of beer on the wall!
Six bottles of beer
Take one down
And pass it around
Five bottles of beer on the wall!

Five bottles of beer on the wall!
Five bottles of beer
Take one down
And pass it around
Four bottles of beer on the wall!

Four bottles of beer on the wall!
Four bottles of beer
Take one down
And pass it around
Three bottles of beer on the wall!

Three bottles of beer on the wall!
Three bottles of beer
Take one down
And pass it around
Two bottles of beer on the wall!
```

DIFFERENT BEVER

```
6 bottles of coke on the wall!
6 bottles of coke
Take one down
And pass it around
5 bottles of coke on the wall!

5 bottles of coke on the wall!
5 bottles of coke
Take one down
And pass it around
4 bottles of coke on the wall!

4 bottles of coke on the wall!
4 bottles of coke
Take one down
And pass it around
3 bottles of coke on the wall!

3 bottles of coke on the wall!
3 bottles of coke
Take one down
And pass it around
2 bottles of coke on the wall!
```

Extra Future Content: Command Line Argument Processing

For this challenge, you need to create one function: **return_parsed_args**. It operates as follows:

- Takes no parameters.
- Creates an ArgumentParser object with a one-line description of what we are doing (that is, examining a pcap).
- Defines five optional arguments: **--start**, **--end**, **--iplist**, **--protocols**, and **--follow-stream**.
- Defines one positional argument, capture.
- Returns an argparse.Namespace instance with those six optional arguments as attributes. (That is, return the equivalent of args from step 3, above--the object that results when you run parse_args on an ArgumentParser instance.)

The **--start** optional argument:

- Is an integer.
- Has a default value of 0.
- Should have a help field indicating this is the first packet processed chronologiclexed.

- If you get stuck, try copying the example declaration of --foo, shown above.

The **--end** optional argument:

- Is an integer.
- Has a default value of -1.
- Should have a help field indicating that this is the first packet NOT processed chronologically and that the packets are zero-indexed.
- If you get stuck, try copying the example declaration of --foo, shown above.

The **--iplist** optional argument:

- Is Boolean.
- Has a default value of False.
- Should have a help field. We will use this parameter to list IPs in the capture.
- If you get stuck, try copying the example declaration of --true, shown above.

The **--protocols** optional argument:

- Is Boolean.
- Has a default value of False.
- Should have a help field. We will use this parameter to print a list of protocols used in the capture.
- If you get stuck, try copying the example declaration of --true, shown above.

The **--follow-stream** optional argument:

- Is a list of 2 strings (we will check for correct string formatting later.)
- Should have a help field specifying that the proper form for inputs is ip1:port1 ip2:port2.
- If you get stuck, try copying the example declaration of --bar, shown above.

The capture positional argument:

- Is a string.
- Should have a help field specifying that it is expected to be the file name for a pcap.

Exercise 24

The purpose of t Controlled Unclassified Information (CUI) t is the beginning of a custom library to r d by another program in

order for the function(s) within it can be called.In this function:

- create a function called get_date_time
- return the current date from the now() method of datetime
- will not perform any calculations or print statements when this program is executed by its name.

Exercise Notes:

- The program does not need a __main__ because it is meant to be imported
- Functions must be named exactly as listed above

Grading: Grading will be based upon whether the requirements are met and if the code executes without errors or warnings to complete the requirements. Partial credit may be granted, if code is there to support it when a requirement has not been met. The submitted program must work with python3.8 and on the classroom machine.

```
# Modules to import

def get_date_time():    pass
```

Exercise 25

The purpose of this program is to create a program that is the beginning of a custom library to test file attributes that will be imported by another program in order for the function(s) within it can be called. In this function:

- create a function called is_file_readable
  - uses os.access
  - return True if the passed file is readable
  - return False if the passed file is not readable
- create a function called is_file_executable
  - uses os.access
  - return True if the passed file is readable
  - return False if the passed file is not readable
- will not perform any calculations or print statements when this program is executed by its name

Exercise Notes:

- The program does not need a __main__ because it is meant to be imported
- Functions must be named exactly as listed above

Grading: Grading will be based upon whether the requirements are met and if the code executes without errors or warnings to complete the requirements. Partial credit may be granted, if code is there to support it when a requirement has not been met. The submitted program must work with python3.8 and on the classroom machine.

```
# Module to impor
```

def is_file_readable(): pass

def is_file_executable(): pass

Exercise 26

The purpose of this program is to retrieve, format, and print the shell environmental variables for the user running the program. The program will:

- only execute commands if execution occurred as a result of this program name being called and not when imported
- when imported, functions will only execute when called
- create a function called main that will call the other functions to perform the required actions
- create a function called retrieve_env_variables
  - retrieve the environmental variables with associated values
  - return the variable name with associated value
- create a function called format_env_variables
  - formats the key/value pairs
  - environmental variable name left justified within a 30 character string
  - value formatting does not change
- create a function called print_env_variables
  - prints the formatted environmental variable
  - followed directly by a colon and a space
  - followed directly by the value that the environmental variable holds
  - end the line with one new line

Output (partial - data will change for the user):

exercise3_format_environment.py

```
Output (partial - data will change for the user):
exercise3_format_environment.py
SHELL                          : /bin/bash
SESSION_MANAGER                : local/thekeeper:@/tmp/.ICE-unix/2012,
unix/thekeeper:/tmp/.ICE-unix/2012
QT_ACCESSIBILITY               : 1
COLORTERM                      : truecolor
XDG_CONFIG_DIRS                : /etc/xdg/xdg-ubuntu-wayland:/etc/xdg
SSH_AGENT_LAUNCHER             : gnome-keyring
XDG_MENU_PREFIX                : gnome-
GNOME_DESKTOP_SESSION_ID       : this-is-deprecated
LANGUAGE                       : en_US:
GNOME_SHELL_SESSION_MODE       : ubuntu
SSH_AUTH_SOCK                  : /run/user/1006000561/keyring/ssh
XMODIFIERS                     : @im=ibus
DESKTOP_SESSION                : ubuntu-wayland
GTK_MODULES                    : gail:atk-bridge
```

Exercise Notes:

- The program should have the main test with a main() function that executes the

- There is no requirement to retrieve anything from the command line
- Functions must be named exactly as listed above
- Output must be identically formatted as to what is shown here

Grading: Grading will be based upon whether the requirements are met and if the code executes without errors or warnings to complete the requirements. Partial credit may be granted, if code is there to support it when a requirement has not been met.
The submitted program must work with python3.8 and on the classroom machine.

```
# Import required modules

def retrieve_env_variables():

   pass

def format_env_variables():

   pass

def print_env_variables():

   pass


def main():

   pass


if __name__ == '__main__':

    pass
```

Exercise 27

The purpose of this program is to retrieve the MAC address for all interfaces found on the host system, retrieve the vendor that supplied the interface, and prints the results in a formatted way. The program will:

- only execute commands if execution occurred as a result of this program name being called and not when imported
- when imported, functions will only execute when called
- create a function called main that will call the other functions to perform the required actions
- create a function called get_if_name_list
    - is passed in the file name to retrieve the interface names
        - /proc/net/dev
- read the file
- look for colons (:), skipping lines with colons
- retrieve the name before the colon (:)
- return the list of interface names found

- create a function called get_mac_addresses
  - is passed in a list of interface names
  - for each interface name
  - opens an empty socket
    - sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    - retrieves the MAC address associated with the socket for the specified interface name
    - retrieve the information associated with the interface name's socket
    - fcntl.ioctl(sock.fileno(), 0x8927, struct.pack('256s', bytes(**<if_name_variable>**<if_name_variable>,'utf-8')[:15]))
    - <if_name_variable>get the MAC address from the information recovered
    - <if_name_variable>':'.join('%02x' % b for b in <var_name>[18:24])
    - <if_name_variable><var_name>close the socket
    - <if_name_variable><var_name>keep the recovered MAC address if it is not '00:00:00:00:00:00'
    - <if_name_variable><var_name>return the recovered list of MAC addresses found

- create a function called get_mac_details
  - is passed in a list of mac_addresses
  - to retrieve MAC address vendor, use the API url of:https://api.macvendors.com/
  - step through each MAC address found
  - append the MAC address to the API url
  - sleep for 1 second, required for API to ensure that it works
  - use the requests library to read data from the API
  - requests.get
  - requests.status_code (200 is a valid response)
  - requests.content.decode
  - if a valid response was received, store the MAC address with the vender decoded from the response
  - if a valid response was not received, store the MAC address with the phrase "[!] Invalid MAC Address"
  - return dictionary of the the MAC address with the associated information stored

- create a function called print_mac_address_info
  - is passed in the dictionary of MAC address with associated information stored
  - step through each MAC address found
  - if the MAC address was valid, print the message formatted:
  - Device vendor for MAC address is .
  - if the MAC address was invalid, print the message formated:
    - [!] Invalid MAC Address:

Example Output (extra newline characters are for display purposes only):

exercise4_mac_information.py

Device vendor for

Device vendor for MAC address 48:5f:99:ce:f0:c3 is Cloud NetworkTechnology (Samoa) Limited.

Exercise Notes:

- The program should have the __main__ test with a main() function that executes the code
- There is no requirement to retrieve anything from the command line
- Functions must be named exactly as listed above
- Output must be identically formatted as to what is shown here

Grading: Grading will be based upon whether the requirements are met and if the code executes without errors or warnings to complete the requirements. Partial credit may be granted, if code is there to support it when a requirement has not been met. The submitted program must work with python3.8 and on the classroom machine.

```
# Imports the modules

def get_if_name_list():

    pass

def get_mac_addresses():

    pass

def get_mac_details():

    pass

def print_mac_address_info():

    pass

def main():

    pass

if __name__ == '__main__':

    pass
```

Exercise 28

The purpose of this program is to retrieve the host system informationand the results in a formatted way.

The program will:

- only execute commands if execution occurred as a result of thisprogram name being called and not when imported
- when imported, functions will only execute when called
- create a func[redacted]unctions toperform the required acti[redacted]

- create a function called retrieve_host_information
  - is not passed anything
  - returns the result of the os.uname method

- create a function called print_host_information
  - is passed the result of the os.uname method
  - prints the information os.uname results (one per line) with the fieldname (attribute) and its value separated by a " => "
  - the fields (attributes) to recover from os.uname are:
    - sysname
    - nodename
    - release
    - version
    - machine
  - the values associated with each field can be retrieved using thegetattr function
  - does not return anything

```
Example Output:
exercise5_get_host_information.py
sysname => Linux
nodename => thekeeper
release => 5.4.0-139-generic
version => #156-Ubuntu SMP Fri Jan 20 17:27:18 UTC 2023
machine => x86_64
```

Exercise Notes:

- The program should have the __main__ test with a main() function that executes the code
- There is no requirement to retrieve anything from the command line
- Functions must be named exactly as listed above
- Output must be identically formatted as to what is shown here

Grading: Grading will be based upon whether the requirements are met and if the code executes without errors or warnings to complete the requirements. Partial credit may be granted, if code is there to support it when a requirement has not been met. The submitted program must work with python3.8 and on the classroom machine.

```
# Import the libraries

def retrieve_host_information():

   pass

def print_host_information():

   pass

def main():
```

```
    pass

if __name__ == '__main__':

    pass
```

Exercise 29

The purpose of this program is to retrieve the host system platform information and the results in a formatted way.

The program will:

- only execute commands if execution occurred as a result of this program name being called and not when imported
- when imported, functions will only execute when called
- create a function called main that will call the other functions to perform the required actions
- create a function called retrieve_platform_information
  - is not passed anything
  - returns the result of the platform.uname method
- create a function called print_platform_information
  - is passed the result of the platform.uname method
  - prints the information platform.uname results (one per line) with the field name (attribute) and its value separated by a " => "
  - the fields (attributes) to recover from platform.uname are:
    - system
    - node
    - release
    - version
    - machine
    - processor
  - when the field name (attributes) is printed, it should be printed as title case
  - the values associated with each field can be retrieved using the getattr function
  - does not return anything

```
Example Output:
exercise6_get_platform_information.py
System => Linux
Node => thekeeper
Release => 5.4.0-139-generic
Version => #156-Ubuntu SMP Fri Jan 20 17:27:18 UTC 2023
Machine => x86_64
Processor => x86_64
```

Exercise Notes:

- The program should have the __main__ test with a main() function that executes the code
- There is no r Controlled Unclassified Information :he command line
- Functions m (CUI)

- Output must be identically formatted as to what is shown here

Grading: Grading will be based upon whether the requirements are met and if the code executes without errors or warnings to complete the requirements. Partial credit may be granted, if code is there to support it when a requirement has not been met. The submitted program must work with python3.8 and on the classroom machine.

```
# Import the libraries

def retrieve_platform_information():

    pass

def print_platform_information():

    pass

def main():

    pass

if __name__ == '__main__':

    pass
```