Q1 Unicode Codes

Complete the given "scaffold" to write a Python script called **unicode.py** that takes one command line argument. The argument is a short string representing **one of five possible Unicode characters**, and the script should print the corresponding **Unicode code** as per the table below:

**Unicode character Unicode code**

NUL U+0000

LF U+000A

CR U+000D

PAD U+0080

ESC U+001B

The Unicode characters and Unicode codes are simple strings, that can be printed with the normal **print** function.

The argument can be provided in **any combination of lower cases and upper case letters**. If any other argument is given, the script should output the string **error**. Below are some examples:

python unicode.py LF

U+000A

python unicode.py eSc

U+001B

python unicode.py DEL

error

Your script will always be tested with one argument, so there is no need to test the number of arguments.

Q2 Celsius and Fahrenheit

Complete the scaffold script to provide a program that takes two command line arguments.

The first argument is a string that represents an option and can only have values "**-c**" or "**-f**".

The second argument must be a numerical value. If the option is "**-c**", the value is meant to represent a temperature in degrees Fahrenheit, and your program will convert it to degrees Celsius using the following formula (no knowledge of temperature scales is required):

(temperature in degrees Fahrenheit - 32)/1.8

Conversely, if the option is "**-f**", the value is meant to represent a temperature in degrees Celsius, and your program will convert it to degrees Fahrenheit using the following formula:

temperature in degrees Celsius \* 1.8 + 32

The script must **print** the result using the normal Python **print** function (without any special formatting)

Two examples:

python convert.py -c 81.5

27.5

python convert.py -f 28

82.4

NB: The script will always be tested with two correct inputs. There is no need for your script to check this, and there is no need for your script to test that the temperature values are in a realistic range.

Q3 Replace Last Item in a List, and Sort the List

Modify the scaffold code so that it reads one command line argument, a string.

The script must then **replace** that string to the last element of list **test\_list**, **sort** the list and **print** the resulting list.

Example:

python list\_item.py Great

['Any', 'Great', 'Here', 'Some', 'There']

(as you can see in the scaffold, the list was originally: ['Here', 'Some', 'Any', 'There', 'Good']).

To print, use the normal Python **print** function with only list **test\_list** as its argument. The script should produce no other output.

The script will always be tested with one argument so **there is no need to test for the correct number of arguments**.

## Q4 String First and Last Character

Modify the scaffold script so that it reads one command line argument, a string.

The script must then print the **first and last** character of the string separated by a space, and nothing else.

Example:

python show\_char.py Spring2021

S 1

(as you can see in the scaffold, the list was originally: ['Here', 'Some', 'Any', 'There', 'Good']).

The script will always be tested with one input so **there is no need to test for the correct number of arguments**.

*Hints:*

1. *By default the normal Python print function will produce the correct format if it's given the two arguments.*
2. *In Python, index -1 means: last.*

## Q5 Split and Print Each Line

Modify the scaffold code so that: it removes the newline from each line; splits the line using '**,**' as the separator; saves the split parts in a list; and eventually prints the list.

As an example, if the argument file contains these two lines:

a,b,c,d

hi, how are you?

The program should be invoked like this and produce this output:

python split.py

['a', 'b', 'c', 'd']

['hi', ' how are you?']

*Hint: This problem is most easily solved by using the methods in the re module.*

## Q6 Function Showing Simple Bar Plot

Modify the scaffold code so that it displays a simple sideways bar plot (example below).

To do this you must define a function called **show\_bars\_func** that has **1 parameter**, an integer.

**show\_bars\_func** must use the normal Python **print** function to output a **string** of '-' (i.e., dash) characters. The number of dash characters in output must be equal to the value of the integer argument passed to **show\_bars\_func**.

Scaffold code is provided to loop over the arguments received by your script from the command line. You must call **show\_bars\_func** within this loop block to show a bar the of the correct length.

Example output:

python show\_bars.py 10 3 14 5 15

----------

---

--------------

-----

---------------

*Hint: To show* ***n*** *repeats of a string* ***s*** *you can use the string repetition operator* ***s \* n***

## Q7 Split at Character Argument

Modify the scaffold code so that it takes two command line arguments, a **string** and a **character**, in this order. The script must **split** the string to form a Python **list** of parts. The **split** must be at each point in the **string** where the input character occurs.

The script must print the resulting list with the normal Python **print** function. There must be no other output.

Example input and result:

python split\_at\_char.py 'turn on : tune in : chop out' :

['turn on ', ' tune in ', ' chop out']

The script will always be tested with two inputs so **there is no need to test for the correct number of arguments**.

*Hint: This problem is most easily solved by using one of the main methods in the re module.*

## Q8 Simple Dictionary Manipulation

The scaffold code provides a short **dict** of chemical element names as keys, with their approximate atomic masses as values. Your task is to extend the code so that it reads **two command line arguments** (an element name and its atomic mass). It must then test if the element name already exists as a key in the **dict**, check that it has the atomic mass received as argument, and, if so, print a message. If the key exists, but its value is different, it prints a message. If the key doesn't exist, it adds it to the dictionary with the given value and prints a message.

In detail:

If the element name exists as a key in the **dict** and has the atomic mass received as argument as value, your code must **print** string: "Element already present". For example (with reference to the scaffold):

python dict\_item.py Neon 20

Element already present

If the element name exists as a key in the **dict**, but its value is different from the atomic mass received as argument, your code must **print** string: "Element present, but the value is different". For example (with reference to the scaffold):

python dict\_item.py Helium 5

Element present, but the value is different

If the element name is not a key in the **dict**, your code must **print** string: "Element added. Updated dictionary:", followed by the updated dictionary in the following line. For example (with reference to the scaffold):

python dict\_item.py Zinc 65

Element added. Updated dictionary:

{'Hydrogen': 1, 'Helium': 4, 'Lithium': 7, 'Beryllium': 9, 'Boron'

: 11, 'Carbon': 12, 'Nitrogen': 14, 'Oxygen': 16, 'Fluorine': 19,

'Neon': 20, 'Zinc': '65'}

Use the normal Python **print** function to print all results. Do not print anything extra.

The script will always be tested with **correct** arguments, so **there is no need to test for the correct number of arguments** **and/or their values**.