Question 1 – display_as_list

Create a function called **display_as_list**

display_as_list has **three parameters**

- **display_items**
- **message**
- **counter_reqd**

display_items is **mandatory** and both message and counter_reqd are **optional**.

message is a string and **defaults** to 'Item'.

counter_reqd is boolean and **defaults** to True.

If there are items in the list, the function displays them as required and **returns True**. See the examples below.
If there are no items in the list, the function informs the user accordingly and **returns False**. See the example below.

If counter_reqd is True then on each line, display both the message and the counter. If counter_reqd is False then on each line, only display the item, not the message or the counter.

display_list prints a blank line before printing the individual elements of the list on separate lines as per the examples below.

Your solution needs to pass all the tests in order to score the marks for this question.

Hints.

- The function is identical to question 1 in the lists lab. Re-using that function is the preferred solution. It is a good reminder of why we write code as functions.
- Not all the test data used is shown.

For example:

Test	Result

recipe_instructions = ['Toast bread', 'Spread butter', 'Spread marmite']
shopping_message = 'Shopping item'
purchase_message = 'Purchase item'
 	
# Use passing by position. 	
# Leave message and counter_reqd as their defaults 	
return_value = display_as_list(recipe_instructions)
 	
# Use passing by name 	
# Pass value by name for display_items 	
# Leave message and counter_reqd as their defaults 	
return_value = display_as_list(display_items=recipe_instructions)
 	
# Use passing by name 	
# Pass value by name for display_items and message 	
# counter_reqd as the default 	
return_value = display_as_list(display_items=recipe_instructions, message=shopping_message)
print(f'The function returned {return_value}')
 	
# Use passing by name 	
# Pass value by name for display_items, message and counter_reqd (explicit as True) 	
return_value = display_as_list(display_items=recipe_instructions, message=purchase_message, counter_reqd=True)
print(f'The function returned {return_value}')
 	
# Use passing by name 	
# Pass value by name for display_items, message and counter_reqd 	
# No counter displayed 	
return_value = display_as_list(display_items=recipe_instructions, message=purchase_message, counter_reqd=False)
print(f'The function returned {return_value}')
	Item 1: Toast bread
	Item 2: Spread butter
	Item 3: Spread marmite
	
	Item 1: Toast bread
	Item 2: Spread butter
	Item 3: Spread marmite
	
	Shopping item 1: Toast bread
	Shopping item 2: Spread butter
	Shopping item 3: Spread marmite
	The function returned True
	
	Purchase item 1: Toast bread
	Purchase item 2: Spread butter
	Purchase item 3: Spread marmite
	The function returned True
	
	Toast bread
	Spread butter
	Spread marmite
	The function returned True

Test	Result
# Test for empty list and parameter names # Pass value by name for display_items and counter_reqd # Leave message as default recipe_instructions = [] return_value = display_as_list(display_items = recipe_instructions, counter_reqd = True) instructions = [] return_value = display_as_list(display_items = recipe_instructions, message = "Don't display this", counter_reqd = False) instructions = [] return_value = display_as_list(display_items = instructions, counter_reqd = False)	Sorry, the list is empty. Sorry, the list is empty. Sorry, the list is empty.

telephone_menu = [
'Telephone contact options.',
'A) Add/modify a contact.',
'R) Remove a contact.',
'S) Show/find a contact.',
'D) Display all contacts.',
'E) Empty all contacts.',
'Q) Quit.'
]
# Pass by both position and name 	
return_value = display_as_list(telephone_menu, counter_reqd=False)
return_value = display_as_list(display_items=telephone_menu, message="Don't display this message", counter_reqd=False)
return_value = display_as_list(display_items=telephone_menu, message='Do display this message:', counter_reqd=True)
	Telephone contact options.
	A) Add/modify a contact.
	R) Remove a contact.
	S) Show/find a contact.
	D) Display all contacts.
	E) Empty all contacts.
	Q) Quit.
	
	Telephone contact options.
	A) Add/modify a contact.
	R) Remove a contact.
	S) Show/find a contact.
	D) Display all contacts.
	E) Empty all contacts.
	Q) Quit.
	
	Do display this message: 1:
	Telephone contact options.
	Do display this message: 2: A)
	Add/modify a contact.
	Do display this message: 3: R)

Test	Result
	Remove a contact. Do display this message: 4: S) Show/find a contact. Do display this message: 5: D) Display all contacts. Do display this message: 6: E) Empty all contacts. Do display this message: 7: Q) Quit.

Question 2 – get_option

Create a function called **get_option**

get_option has **one parameter**.

- **prompt_msg**

prompt_msg is **optional** and **defaults** to 'Option: '

get_option asks the user to enter an option, removes any extra spaces, converts the entered text into lower case and **returns** it as a **string**.

Your solution needs to pass all the tests in order to score the mark for this question.

Hints.

- The function is identical to the shopping lists lab get_option function. Re-using that function is the preferred solution. It is a good reminder of why we write code as functions.
- Not all the test code used is shown.

For example:

Test	Input	Result

# Test for removal of spaces, case and return object type 	
# Use default value for prompt_msg 	
option = get_option()
print(f'Function returned: {option}')
print(f'Function returned: {len(option)} chrs.')
print(f'Function returned: {type(option)} chrs.')
print()
option = get_option()
print(f'Function returned: {option}')
print(f'Function returned: {len(option)} chrs.')
	Add_contact
	remove contact
		Option: Add_contact
		Function returned: add_contact
		Function returned: 11 chrs.
		Function returned: <class 'str'> chrs.
		
		Option: remove contact
		Function returned: remove contact
		Function returned: 14 chrs.
		Function returned: <class 'str'> chrs.

Test	Input	Result
print(f'Function returned: {type(option)} chrs.')		

# Test for default value of prompt_msg 	
reqd_option = get_option()
print(reqd_option)
 	
print()
# Test for default value of prompt_msg 	
# Shorter version 	
print(get_option())
 	
print()
user_prompt = 'Enter yes or no :'
print(get_option(prompt_msg = user_prompt))
 	
print()
# Test parameter name 	
get_option(prompt_msg = 'Enter y or n :')
 	
print()
# Short prompt 	
get_option(prompt_msg = ':')
	QUIT
	exit!
	Yes
	n
	0
		Option: QUIT
		quit
		
		Option: exit!
		exit!
		
		Enter yes or no :Yes
		yes
		
		Enter y or n :n
		
		:0

Question 3 – get_name

Create a function called **get_name**
get_name has **one parameter**.

- **message**

message is the message to be **displayed** as a prompt for the user.
Any leading and trailing spaces need to be removed from the entered name. The entered name needs to be correctly titled.
get_name **returns** the entered name as a **string**.
Your solution needs to pass all the tests in order to score the mark for this question.

Hint.

- Ignoring the docstring, the solution can be 2 or 3 lines of code.

For example:

Test	Input	Result
contact_msg = 'Contact: ' print(get_name(contact_msg))	dave bracken	Contact: dave bracken Dave Bracken

# Leading spaces and a blank name entered 	
contact_msg = 'Contact name: '
contact = get_name(contact_msg)
print(contact)
	leading spaces	Contact name: leading spaces
		Leading Spaces
		Contact name:

Test	Input	Result
contact = get_name(contact_msg) print(contact)		
print(get_name('First name: '))	trailing spaces	First name: trailing spaces Trailing Spaces

Data used for code answer:

'{}'

'{}'
is only shown to indicate output is required. Unless you want to, there is no need for an f-string, simply displaying the object is sufficient.

Question 4 – get_number_as_string

Create a function called **get_number_as_string**

get_number_as_string has two parameters

- **message**
- **min_length**

message is the message to display when asking the user for input.

min_length is the minimum number of digits needed.

get_number_as_string asks the user to a number using the message it received as a parameter. All spaces must be removed from the entered value.

The number is valid if it contains only digits and is at least min_length in length.

If the value entered is comprised of not only digits, inform the user and ask for the value to be entered again. See examples below.

If the value entered is comprised of only digits but is less than the minimum required length, inform the user and ask for the value to be entered again. See examples below.

get_number_as_string **returns** the entered number as a **string**.

Hints.

- You need to remove all the spaces from the data entered, not just the leading and trailing ones. You only need to use one string method for this.
- The solution involves using a loop. See the loops presentations for examples that can be used as the basis of a solution for this.
- Ignoring blanks lines and docstring, the solution is about 9 lines of code.
- Not all the test data used is shown.

For example:

Test	Input	Result

# Test parameter names 	
pin_message = 'Please enter your pin number: '
min_digits = 4
entered_number = get_number_as_string(message = pin_message, min_length = min_digits)
print(f'The number is {entered_number}')
	one two three four
	one 2 3 four
	
	0.3 9.4.0 8.1.6.1
	0.3.9.4.0.8.1.6.1.
	03 940 8161
		Please enter your pin number: one two three four
		Digits only please.
		Please enter your pin number: one 2 3 four
		Digits only please.
		Please enter your pin number:
		Digits only please.
		Please enter your pin number: 0.3 9.4.0 8.1.6.1
		Digits only please.
		Please enter your pin number: 0.3.9.4.0.8.1.6.1.
		Digits only please.
		Please enter your pin number: 03 940 8161
		The number is 039408161

landline_message = 'Please enter your landline number: ' minimum_digits = 7 mobile_number = get_number_as_string(landline_message, minimum_digits) print(f'The number is {mobile_number}')	9 4 o 8 i 6 i 9 4 0 8 1 6 1	Please enter your landline number: 9 4 o 8 i 6 i Digits only please. Please enter your landline number: 9 4 0 8 1 6 1 The number is 9408161
ext_message = 'Please enter your extension number: ' min_dgts = 4 extension_number = get_number_as_string(ext_message, min_dgts) print(f'The number is {extension_number}')	four three two one four three two 1 four three 2 one four three 2 1 four 3 two one	Please enter your extension number: four three two one Digits only please. Please enter your extension number: four three two 1

Test	Input	Result
	four 3 two 1 four 3 2 one four 3 2 1 4 three two one 4 three two 1 4 three 2 one 4 three 2 1 4 3 two one 4 3 two 1 4 3 2 0ne 0 1 2 o i 2 3 4 0 i 2 3 4 5 0 1 2 3 4	Digits only please. Please enter your extension number: four three 2 one Digits only please. Please enter your extension number: four three 2 1 Digits only please. Please enter your extension number: four 3 two one Digits only please. Please enter your extension number: four 3 two 1 Digits only please. Please enter your extension number: four 3 2 one Digits only please. Please enter your extension number: four 3 2 1 Digits only please. Please enter your extension number: 4 three two one Digits only please. Please enter your extension number: 4 three two 1 Digits only please. Please enter your extension number: 4 three 2 one Digits only please. Please enter your extension number: 4 three 2 1 Digits only please. Please enter your extension number: 4 3 two one Digits only please.

Test	Input	Result
		Please enter your extension number: 4 3 two 1 Digits only please. Please enter your extension number: 4 3 2 0ne Digits only please. Please enter your extension number: 0 1 2 The number must be at least 4 digits long. Please enter your extension number: o i 2 3 4 Digits only please. Please enter your extension number: 0 i 2 3 4 5 Digits only please. Please enter your extension number: 0 1 2 3 4 The number is 01234

Data used for code answer:

```
# 'Digits only please.'
# 'The number must be at least {} digits long.'
```

Question 5 – add_edit_contact

Create a function called **add_edit_contact**

add_edit_contact has one parameter

- contacts

contacts is a dictionary of contacts.

add_edit_contact must obtain the name of a contact by **calling** the **get_name** function.

If the entered contact name returned by get_name is not in the contacts dictionary then add_edit_contact must obtain the phone number of the contact by **calling** the **get_number_as_string** function and **passing** it the 'Contact number: ' message as the prompt. The minimum number of digits for the number (required by get_number_as_string) is 3. The add_edit_contact function then adds the contact and their phone number that was returned by get_number_as_string to the contacts dictionary and informs the user that it has been added. The phone number in the dictionary is a string.

If the entered contact returned by get_name is in the contacts dictionary then add_edit_contact informs the user that the contact is present and asks if the user would like to change the number. The add_edit_contact function **calls** the **get_option** function **passing** it 'Would you like to change the phone number? y/n: ' as the prompt to be displayed. If the user enters 'y' (case insensitive) then add_edit_contact asks the user to enter the new number for the contact by **calling** the **get_number_as_string**function and **passing** it the "New number: " message as the prompt. If the new number is different from the existing number, the add_edit_contact function then changes the phone number for the contact and informs the user that it has been changed. If the new number is the same as the existing number of the contact, add_edit_contact informs the user that the number is the same and so it won't be changed. If the user did not enter 'y' (case insensitive) then add_edit_contact informs them that the contact has not been updated.

add_edit_contact returns boolean True if the contact and or their number was added or changed, otherwise it returns False.

Hints.

- The solution is about 21/22 lines of code.
- The add_edit_contact function does not call the input function itself, that's the job of the other functions.
- The message 'Digits only please.' is generated by the get_number_as_string function.
- Don't forget to check the return value.
- Not all of the test data used is shown.

Only post the add_edit_contact function below. Do **not** include the get_name or get_number_as_string or get_option functions.

For example:

Test	Input	Result

# Add new contact 	
# Test parameter name 	
personal_contacts = {'Ms One':'101',
'Mr Two':'102',
'Ms Tree':'103'
}
modification_status = add_edit_contact(contacts=personal_contacts)
print(personal_contacts)
	Master
	Four
	104
		Contact: Master Four
		Contact number: 104
		Master Four has been added with the number: 104.
		{'Ms One': '101', 'Mr Two': '102', 'Ms Tree': '103',
		'Master Four': '104'}

# Attempt to add an existing contact 	
my_contacts = {'Ms One':101,
'Mr Two':'102',
'Ms Tree':'103'
}
modification_status = add_edit_contact(my_contacts)
print(my_contacts)
	Ms Tree
	No
		Contact: Ms Tree
		Ms Tree is already in the phone system.
		Would you like to change the phone number? y/n:
		No
		Ms Tree has not been updated.
		{'Ms One': 101, 'Mr Two': '102', 'Ms Tree': '103'}

# Modify an existing contact 	
all_contacts = {'Ms One':101,
'Mr Two':'102',
'Ms Tree':'103'
}
modification_status = add_edit_contact(all_contacts)
print(all_contacts)
print()
# Enter the same number when attempting to modify an existing contact 	
	Ms Tree
	Y
	ii3
	113
	ms tree
	y
	113
		Contact: Ms Tree
		Ms Tree is already in the phone system.
		Would you like to change the phone number? y/n: Y
		New number: ii3
		Digits only please.
		New number: 113
		Ms Tree has been updated with the number: 113.
		{'Ms One': 101, 'Mr Two': '102', 'Ms Tree': '113'}
		
		Contact: ms tree

Test	Input	Result
modification_status = add_edit_contact(all_contacts) print(all_contacts)		Ms Tree is already in the phone system. Would you like to change the phone number? y/n: y New number: 113 Ms Tree already has the number 113. The number has not been changed. {'Ms One': 101, 'Mr Two': '102', 'Ms Tree': '113'}

Data used for code answer:

```
# 'Contact: '
# '{} is already in the phone system.'
# 'Would you like to change the phone number? y/n: '
# 'New number: '
# '{} already has the number {}. The number has not been changed.'
# '{} has been updated with the number: {}.'
# '{} has not been updated.'
# 'Contact number: '
# '{} has been added with the number: {}.'
```

Question 6 – remove_contact

Create a function called **remove_contact**

remove_contact has one parameter

- contacts

contacts is a dictionary of contacts.

If there are no contacts in the contact dictionary, remove_contact informs the user that the contacts dictionary is empty.

If there is a contact in the contacts dictionary, remove_contact asks the user to enter the name of a contact
by **calling** the **get_name** function passing it the prompt 'Enter the contact to be removed (or press Enter to cancel): '.

If the user just presses the Enter key when asked for the contact that is to be removed, inform the user that the request is being cancelled.

If the entered contact is in the contacts dictionary then remove_contact removes the user from the contacts dictionary and informs the user that the contact has been removed.

If the entered contact (something was entered as the contact name other than the Enter key) is not in the contacts dictionary then remove_contact informs the user that the entered contact is not present.

remove_contact returns boolean True if the contact was removed, otherwise it returns False.

Hints.

- Ignoring the docstring and any comments, the solution is about 13 lines of code.
- The remove_contact function does not call the input function itself, that's the job of the get_name function.
- Not all the test data used is shown.

Only post the remove_contact function below. Do not include the get_name function.

For example:

Test	Input	Result

# Remove Ms One 	
# Test parameter name 	
personal_contacts = {'Ms One':'101',
'Mr Two':'102',
'Mrs Three':'103'
}
return_status = remove_contact(contacts=personal_contacts)
print(personal_contacts)
print()
# Cancel the request 	
return_status = remove_contact(contacts=personal_contacts)
print(personal_contacts)
	ms one	Enter the contact to be removed (or press Enter to cancel): ms one
		Ms One has been removed from the contacts.
		{'Mr Two': '102', 'Mrs Three': '103'}
		
		Enter the contact to be removed (or press Enter to cancel): Cancelling the remove request.
		{'Mr Two': '102', 'Mrs Three': '103'}

# No Johnny Five 	
sales_contacts = {'Ms One':'101',
'Mr Two':'102',
'Mrs Three':'103'
}
return_status = remove_contact(sales_contacts)
print(sales_contacts)
	johnny five	Enter the contact to be removed (or press Enter to cancel): johnny five
		Sorry, Johnny Five is not a contact.
		{'Ms One': '101', 'Mr Two': '102', 'Mrs Three': '103'}

# Test for empty contact dictionary 	
# Test for correct parameter name 	
# Billy no mates 	
job_contacts = {}
return_status = remove_contact(contacts = job_contacts)
print(job_contacts)
		Sorry, there are no contacts.
		{}

Data used for code answer:

'Enter the contact to be removed (or press Enter to cancel): '

```
# '{} has been removed from the contacts.'
# 'Sorry, {} is not a contact.'
# 'Sorry, there are no contacts.'
# 'Cancelling the remove request.'
```

Question 7 – show_contact

Create a function called **show_contact**

show_contact has one parameter

- contacts

contacts is a dictionary of contacts.

If the contacts dictionary is not empty, show_contact asks the user to enter the name of a contact by **calling** the **get_name**function.

If the entered contact returned by get_name is in the contacts dictionary then show_contact displays the phone number of the contact.

If the entered contact returned by get_name is not in the contacts dictionary then show_contact informs the user that the contact is unknown.

If the contacts dictionary is empty, show_contact informs the user that the contacts are empty.

show_contact returns boolean True if the contact was present, otherwise it returns False.

Hints.

- Ignoring the docstring and any comments, the solution is about 12 lines of code.
- The show_contact function does not call the input function itself, that's the job of the get_name function.
- Not all the test data used is shown.

Only post the show_contact function below. Do not include the get_name function.

For example:

Test	Input	Result
# Find Mr Two	mr two	Contact: mr two

Test	Input	Result
all_contacts = {'Ms One':'101', 'Mr Two':'102', 'Mrs Three':'103' } return_status = show_contact(all_contacts) print(all_contacts)		Mr Two: 102 {'Ms One': '101', 'Mr Two': '102', 'Mrs Three': '103'}

# No Johnny Five 	
dept_contacts = {'Ms One':'101',
'Mr Two':'102',
'Mrs Three':'103'
}
return_status = show_contact(dept_contacts)
print(dept_contacts)
	johnny five	Contact: johnny five
		Johnny Five is unknown.
		{'Ms One': '101', 'Mr Two': '102', 'Mrs Three': '103'}

# Test for correct parameter name 	
# Billy no mates 	
personal_contacts = {}
return_status = show_contact(contacts = personal_contacts)
print(personal_contacts)
		Sorry, there are no contacts.
		{}

Data used for code answer:

```
# '{}: {}'
# '{} is unknown.'
# 'Sorry, there are no contacts.'
# 'Contact: '
```

Question 8 – display_contacts

Create a function called **display_contacts**

display_contacts has one parameter

- contacts

contacts is a dictionary of contacts.

If the contacts dictionary is not empty then display_contacts displays the contacts and associated phone numbers as per the examples below.

If the contacts dictionary is empty then display_contacts informs the user that there are no contacts to be displayed.

list_contacts returns boolean True if the contacts were displayed, otherwise it returns False.

Hints.

- The solution is similar to a question in the lists lab. Use the code from that function as the basis (but remembering that question was referencing a list) for this one.
- Ignoring the docstring and comments, the solution is about 8 lines of code.
- Not all the test data used is shown.

For example:

Test	**Result**
onsite_contacts = {'Ms One':'101', 'Mr Two':'102', 'Mrs Three':'103' }	Ms One : 101 Mr Two : 102 Mrs Three : 103

Test	Result
return_status = display_contacts(onsite_contacts)	
current_contacts = {'Ms One':'101'} return_status = display_contacts(current_contacts)	Ms One : 101

# Test for correct parameter name 	
personal_contacts = {}
return_status = display_contacts(contacts = personal_contacts)
	Sorry, there are no contacts.

Data used for code answer:

```
# '{:<15}: {}'
# 'Sorry, there are no contacts.'
```

Question 9 – empty_contacts

Create a function called **empty_contacts**

empty_contacts has one parameter

- contacts

contacts is a dictionary of contacts.

If the contacts dictionary is not empty and the user confirms they want to empty the contacts dictionary (by entering 'y' in either case) then empty_contacts removes all the contacts from the dictionary and **returns True**.

If the contacts dictionary is not empty and the user does not confirm they want to empty the contacts dictionary then empty_contacts informs the user accordingly and **returns False**.

If the contacts dictionary is empty then empty_contacts informs the user and **returns False**.

The empty_contacts function **must call** or invoke the **get_option** function to ascertain if the user wants to empty the contacts.

Hints.

- The solution is similar to the question raised in the lists lab. Use the code from that function as the basis for this one.
- The empty_contacts function does not call the input function itself, that's the job of the get_option function.
- Ignoring the docstring and comments, the solution is about 11 lines of code.
- Not all the test data used is shown.

Paste the code for **empty_contacts** below. Do **not** include the code for get_option.

For example:

Test	Input	Result

# Empty the contacts 	
# Test parameter name 	
current_contacts = {'Ms One':'101',
'Mr Two':'102',
'Mrs Three':'103'
}
return_status =
empty_contacts(contacts=current_contacts)
print(return_status)
print(f'The contacts dictionary is now:
{current_contacts}')
 	
# Don't empty the contacts 	
current_contacts = {'Ms One':'101',
'Mr Two':'102',
'Mrs Three':'103'
}
return_status =
empty_contacts(contacts=current_contacts)
print(return_status)
print(f'The contacts dictionary is now:
{current_contacts}')
 	
# Don't empty the contacts 	
current_contacts = {'Ms One':'101',
'Mr Two':'102',
'Mrs Three':'103'
}
return_status =
empty_contacts(contacts=current_contacts)
print(return_status)
print(f'The contacts dictionary is now:
{current_contacts}')
	Y
	n
	Yikes, please don't
	empty the contacts!
		Please confirm that you want remove all the
		contacts. y/n: Y
		All the contacts have been removed.
		True
		The contacts dictionary is now: {}
		Please confirm that you want remove all the
		contacts. y/n: n
		The contacts have not been removed.
		False
		The contacts dictionary is now: {'Ms One': '101',
		'Mr Two': '102', 'Mrs Three': '103'}
		Please confirm that you want remove all the
		contacts. y/n: Yikes, please don't empty the
		contacts!
		The contacts have not been removed.
		False
		The contacts dictionary is now: {'Ms One': '101',
		'Mr Two': '102', 'Mrs Three': '103'}

Test	Input	Result

# Empty contacts 	
offsite_contacts = {'Ms One':'101'}
return_status =
empty_contacts(contacts=offsite_contacts)
print(f'The contacts dictionary is now:
{offsite_contacts}')
	y	Please confirm that you want remove all the contacts. y/n: y
		All the contacts have been removed.
		The contacts dictionary is now: {}

# Test on empty contacts 	
# Test for correct parameter name 	
dept_contacts = {}
return_status =
empty_contacts(contacts=dept_contacts)
print(f'The contacts dictionary is now: {dept_contacts}')
		Sorry, there are no contacts.
		The contacts dictionary is now: {}

Data used for code answer:

'Please confirm that you want remove all the contacts. y/n: '
'All the contacts have been removed.'
'The contacts have not been removed.'
'Sorry, there are no contacts.'