Project List :

**Create a class that will accept input in either a string or a list type, if it’s a list then call another function in the class to check they’re all strings (if not, return an error)  
  
Create another function inside the class which can output the list as a file, and finally create an append function that will allow you to append to the list. Once complete you will have a function that accepts and appends lists on demand but only of the str type.  
  
Add additional checks to ensure they’re ASCII-compliant (eg no #%$%^ allowed)  
  
This documentation may help: https://docs.python.org/3/tutorial/errors.html  
https://stackoverflow.com/questions/196345/how-to-check-if-a-string-in-python-is-in-ascii**

**Project Link :**

Question a

Create a class that will accept input in either a string or a list type, if it’s a list then call another function in the class to check they’re all strings (if not, return an error)

Here's a Python class that accepts input in either a string or a list type. If the input is a list, it calls another function to check if all elements in the list are strings. If not, it returns an error:

class InputValidator:

def \_\_init\_\_(self, input\_data):

self.input\_data = input\_data

def validate\_input(self):

if isinstance(self.input\_data, str):

# Input is a string

return self.input\_data

elif isinstance(self.input\_data, list):

# Input is a list, so check if all elements are strings

if all(isinstance(item, str) for item in self.input\_data):

return self.input\_data

else:

return "Error: All elements in the list must be strings."

else:

return "Error: Input must be either a string or a list."

# Example usage:

input\_str = "This is a string."

input\_list = ["Hello", "World"]

input\_mixed\_list = ["Hello", 42, "World"]

validator1 = InputValidator(input\_str)

result1 = validator1.validate\_input()

print(result1) # Output: "This is a string."

validator2 = InputValidator(input\_list)

result2 = validator2.validate\_input()

print(result2) # Output: ["Hello", "World"]

validator3 = InputValidator(input\_mixed\_list)

result3 = validator3.validate\_input()

print(result3) # Output: "Error: All elements in the list must be strings."

In this code, the InputValidator class takes the input data in its constructor and then uses the validate\_input method to determine the type of input and validate it accordingly.

Question b.

Create another function inside the class which can output the list as a file, and finally create an append function that will allow you to append to the list. Once complete you will have a function that accepts and appends lists on demand but only of the str type.

Sure, I can help you extend the InputValidator class to include a function for outputting the list as a file and another function for appending to the list. Here's the updated class:

class InputValidator:

def \_\_init\_\_(self, input\_data):

self.input\_data = input\_data

def validate\_input(self):

if isinstance(self.input\_data, str):

# Input is a string

return self.input\_data

elif isinstance(self.input\_data, list):

# Input is a list, so check if all elements are strings

if all(isinstance(item, str) for item in self.input\_data):

return self.input\_data

else:

return "Error: All elements in the list must be strings."

else:

return "Error: Input must be either a string or a list."

def output\_to\_file(self, filename):

if isinstance(self.input\_data, list):

with open(filename, 'w') as file:

for item in self.input\_data:

file.write(item + '\n')

else:

return "Error: Input must be a list to output to a file."

def append\_to\_list(self, new\_data):

if isinstance(new\_data, str):

if isinstance(self.input\_data, list):

self.input\_data.append(new\_data)

else:

return "Error: Cannot append to a non-list input."

else:

return "Error: Appended data must be a string."

# Example usage:

input\_list = ["Hello", "World"]

validator = InputValidator(input\_list)

result = validator.validate\_input()

print(result) # Output: ["Hello", "World"]

# Output the list to a file

validator.output\_to\_file("output.txt")

# Append to the list

validator.append\_to\_list("Appended String")

print(validator.validate\_input()) # Output: ["Hello", "World", "Appended String"]

In this updated class, we've added two new methods: output\_to\_file for writing the list to a file and append\_to\_list for appending new strings to the list. These functions are designed to work with lists of strings.

Question c

Add additional checks to ensure they’re ASCII-compliant (eg no #%$%^ allowed)

Certainly, you can add additional checks to ensure that the strings are ASCII-compliant and do not contain certain special characters. Here's the updated InputValidator class with these checks included:

class InputValidator:

def \_\_init\_\_(self, input\_data):

self.input\_data = input\_data

def is\_ascii\_compliant(self, text):

try:

text.encode('ascii')

return True

except UnicodeEncodeError:

return False

def validate\_input(self):

if isinstance(self.input\_data, str):

# Input is a string

if self.is\_ascii\_compliant(self.input\_data) and not any(char in "#%$%^" for char in self.input\_data):

return self.input\_data

else:

return "Error: Input contains non-ASCII characters or disallowed special characters."

elif isinstance(self.input\_data, list):

# Input is a list, so check if all elements are strings

if all(isinstance(item, str) for item in self.input\_data):

if all(self.is\_ascii\_compliant(item) and not any(char in "#%$%^" for char in item) for item in self.input\_data):

return self.input\_data

else:

return "Error: List contains non-ASCII characters or disallowed special characters."

else:

return "Error: All elements in the list must be strings."

else:

return "Error: Input must be either a string or a list."

def output\_to\_file(self, filename):

if isinstance(self.input\_data, list):

with open(filename, 'w') as file:

for item in self.input\_data:

file.write(item + '\n')

else:

return "Error: Input must be a list to output to a file."

def append\_to\_list(self, new\_data):

if isinstance(new\_data, str):

if isinstance(self.input\_data, list):

if self.is\_ascii\_compliant(new\_data) and not any(char in "#%$%^" for char in new\_data):

self.input\_data.append(new\_data)

else:

return "Error: Appended string contains non-ASCII characters or disallowed special characters."

else:

return "Error: Cannot append to a non-list input."

else:

return "Error: Appended data must be a string."

# Example usage:

input\_list = ["Hello", "World"]

validator = InputValidator(input\_list)

result = validator.validate\_input()

print(result) # Output: ["Hello", "World"]

# Output the list to a file

validator.output\_to\_file("output.txt")

# Append to the list

validator.append\_to\_list("AppendedString")

print(validator.validate\_input()) # Output: ["Hello", "World", "AppendedString"]

In this updated code, the is\_ascii\_compliant method checks whether a string is ASCII-compliant, and both validate\_input and append\_to\_list methods use this check to ensure that the input is ASCII-compliant and does not contain disallowed special characters.