Assignment 8 Solution

Submitted by : Yameen Ali

Go to Github repository

Question 1: Scenario (startswith())

You are developing a file management system for a large organization. Each department has its own set of files stored in a central repository. The system needs to identify files belonging to the HR department based on their filenames, which all start with the prefix "HR_". Question: How would you use the startswith() method to efficiently filter out files belonging to the HR department from a list of filenames?

In [21]:
```
filenames = ["report.txt", "HR_payroll.xls", "meeting_minutes.docx", "HR_benefits.pdf", "invoice.csv"]
hr_files = [filename for filename in filenames if filename.startswith("HR_")]
print(hr_files)
```

```
['HR_payroll.xls', 'HR_benefits.pdf']
```

Question2: Scenario (endswith())

You are working on an e-commerce platform that sells various products. The platform needs to categorize products based on their file extensions to ensure they are displayed correctly on the website. For example, images should end with ".jpg" or ".png", while documents should end with ".pdf" or ".docx". Question: Describe how you would utilize the endwith() method to categorize products based on their file extensions, ensuring accurate display on the ecommerce platform.

In [42]:
```
products = {
    "product1.jpg":{"name":"Beach Ball", "Category": None},
    "report.docx": {"name":"Monthly sales report","category":None},
    "summer_dress.png": {"name": "Summer Dress", "category": None},
    "instruction_manual.pdf": {"name": "Instruction Manual", "category": None}
}
for filename,product_info in products.items():
    if filename.endswith(".jpg") or filename.endswith(".png"):
        product_info["category"] = "Image"
        print(product_info)

    elif filename.endswith(".docx") or filename.endswith(".pdf"):
        product_info["category"] = "Document"
        print(product_info)
```

```
{'name': 'Beach Ball', 'Category': None, 'category': 'Image'}
{'name': 'Monthly sales report', 'category': 'Document'}
{'name': 'Summer Dress', 'category': 'Image'}
{'name': 'Instruction Manual', 'category': 'Document'}
```

Question3: Scenario (find())

You are developing a search engine for a large online library containing millions of documents. Users should be able to search for specific keywords within the documents. To optimize search performance, the system needs to efficiently locate the position of keywords within the text. Question: How can you use the find() method to quickly locate the position of a keyword within a document, allowing users to search through millions of documents efficiently?

In [54]:
```
document = "Yameen is a Pyhton Developer and Data Scientist."
keyword = "Yameen"

# Find the first occurrence (case-sensitive)
position = document.find(keyword)

if position != -1:
    print(f"Keyword '{keyword}' found at index {position}")
else:
    print(f"Keyword '{keyword}' not found in document")
```

```
Keyword 'Yameen' found at index 0
```

Question4: Scenario (isalnum()):

You are implementing a user registration system for an online platform. Usernames must meet certain criteria, including being alphanumeric and not containing special characters. The system needs to validate usernames entered by users to ensure they meet these criteria. Question: Explain how you would use the isalnum() method to validate usernames entered by users during the registration process, ensuring they are alphanumeric and do not contain special characters.

In [66]:
```
# Checks if a username is alphanumeric and does not contain special characters.
def is_valid_username(username):
    # check if the username is empty
    if not username:
        return False

    return username.isalnum()

username = input("Enter Username: ")
if is_valid_username(username):
    print(f"{username} is Valid.")

else:
    print(f"{username} is invalid. Please use only alphanumeric characters.")
```

```
Yameenali012 is Valid.
```

Question 5: Scenario (isalpha()):

You are developing a language processing application that analyzes text data from various sources. The application needs to distinguish between text data containing only alphabetic characters and data containing alphanumeric characters or symbols. Question: How would you use the isalpha() method to differentiate between text data containing only alphabetic characters and data containing alphanumeric characters or symbols in a language processing application?

In [73]:
```
def is_alphabetic(text):
    if not text:
        return False

    return text.isalpha()

text = input("Enter Text: ")
if is_alphabetic(text):
    print(f"{text} is Alphabetic.")

else:
    print("Text is not Alphabetic")

# note : white spaces is also non-alphabetic characters.
```

```
Yameen is Alphabetic.
```

Question 6: Scenario (join()):

You are developing a script to generate CSV (Comma-Separated Values) files from structured data. The script needs to concatenate individual data fields into CSV rows, separating each field with a comma. Question: How can you use the join() method to concatenate individual data fields into CSV rows, ensuring proper formatting for CSV files?

In [72]:
```
# Sample Data
data = [
    ["Name", "Age", "City", "Country"],
    ["Yameen", 22, "Multan", "Pakistan"],
    ["Rafay", 21, "Multan", "Pakistan"],
]

#  Function to generate CSV row
def generate_csv_row(data_row):
    return ",".join(str(field) for field in data_row)

# Generate CSV rows from data
csv_rows = [generate_csv_row(row) for row in data]
for rows in csv_rows:
    print(rows)
```

```
Name,Age,City,Country
Yameen,22,Multan,Pakistan
Rafay,21,Multan,Pakistan
```

Question 7: Scenario (strip()):

You are working with textual data extracted from web pages, which often contain leading and trailing whitespace characters. Before processing the text, you need to remove any leading or trailing whitespace to ensure accurate analysis. Question: Explain how you would utilize the strip() method to remove leading and trailing whitespace characters from textual data extracted from web pages, improving the accuracy of subsequent text analysis.

In [84]:
```
text_data = [
    "  Extra spaces before and after.  ",
    "\tLeading tab and trailing space\t",
    " No whitespace at the edges. ",
]

cleaned_data = [text.strip() for text in text_data]
for text in cleaned_data:
    print(text)
```

```
Extra spaces before and after.
Leading tab and trailing space
No whitespace at the edges.
```

Question 8: Scenario (replace()):

In a text processing pipeline, you need to sanitize user input by replacing certain substrings with sanitized versions. For example, you may need to replace sensitive information such as email addresses or phone numbers with placeholders. Question: Describe how you would employ the replace() method to sanitize user input by replacing sensitive substrings with placeholders, ensuring the privacy and security of user data.

In [115…
```
sensitive_data = {
    "email": "yameenali012@gmail.com",  # Replace with actual email if testing
    "phone_number": "03166491000"        # Replace with actual phone number if testing
}

placeholders = {
    "email": "[EMAIL_ADDRESS]",
    "phone_number": "[PHONE_NUMBER]"
}
def sanitize_input(text):
  for data_key, data_value in sensitive_data.items():
      if data_value in text:  # Check for exact match
          text = text.replace(data_value, placeholders[data_key])
  return text

user_input = "My email is yameenali012@gmail.com and phone number is 03166491000."
sanitized_text = sanitize_input(user_input)

print(f"Original Value: {user_input}")
print(f"Sanitized Value: {sanitized_text}")
```

```
Original Value: My email is yameenali012@gmail.com and phone number is 03166491000.
Sanitized Value: My email is [EMAIL_ADDRESS] and phone number is [PHONE_NUMBER].
```

Question 9: Scenario (split()):

You are developing a script to parse log files containing timestamped events. Each line in the log file represents a single event, with timestamp and event details separated by a delimiter. You need to split each line into timestamp and event details for further analysis. Question: How would you utilize the split() method to parse log file lines into timestamp and event details, extracting relevant information for further analysis in a script?

In [139…
```
def parse_log_line(line):
  # Define delimiter (assuming space between timestamp and message)
  delimiter = " "
  # Split the line using the delimiter (get maximum of 1 split)
  parts = line.split(delimiter, maxsplit=1)

  if len(parts) != 2:
      return None  # Handle invalid lines (not enough parts)

  timestamp, message = parts
  return timestamp, message

# Example usage:
```

```
log_lines = [
    "2024-03-24 10:00:00 INFO: User login successful.",
    "2024-03-24 12:15:17 WARNING: Application error occurred.",
    "Invalid log line",  # Test case for invalid line
]

for line in log_lines:
  parsed_data = parse_log_line(line)
  if parsed_data:
      timestamp, message = parsed_data
      print(f"Timestamp: {timestamp}")
      print(f"Message: {message}")
      print("-" * 20)  # Separator for readability
  else:
      print(f"Invalid log line format: {line}")
```

```
Timestamp: 2024-03-24
Message: 10:00:00 INFO: User login successful.
--------------------
Timestamp: 2024-03-24
Message: 12:15:17 WARNING: Application error occurred.
--------------------
Timestamp: Invalid
Message: log line
--------------------
```

Question 10: Scenario (lower()):

You are implementing a search functionality in an application where search queries should be case-insensitive. The application needs to convert all search queries to lowercase before performing the search operation to ensure consistent results. Question: Explain how you would apply the lower() method to convert search queries to lowercase, enabling case-insensitive search functionality in an application.

In [159…

```
def perform_search(query,data):
    # Convert the query to lowercase for case-insensitive search
    lowercase_query = query.lower()
    matching_items = [item for item in data if lowercase_query in item.lower()]
    return matching_items


data = ["This is some data by Yameen", "Another item to search", "This One Has Mixed Case"]
search_query = input("Enter Query: ")

results = perform_search(search_query,data)
print(f"Search Results for '{search_query}': ")
for item in results:
    print(item)
```

```
Search Results for 'YameEn':
This is some data by Yameen
```

Question 11: Scenario (islower()) & (isupper()):

You are developing a text analysis tool that categorizes text data based on its case sensitivity. The tool needs to differentiate between text data containing only lowercase letters, only uppercase letters, or a combination of both. Question: How can you use the islower() and isupper() methods to categorize text data based on its case sensitivity in a text analysis tool?

In [167…

```
def analyze(text):
    if text.islower():
        return "lowercase"
    elif text.isupper():
        return "uppercase"
    else:
        return "mixed"

text_data = [
    "this is all lowercase text.",
    "THIS IS ALL UPPERCASE TEXT.",
    "This Text Has Mixed Case."
]
for text in text_data:
    categorize_text = analyze(text)
    print(f"'{text}' is categorized as: {categorize_text}")
```

```
'this is all lowercase text.' is categorized as: lowercase
'THIS IS ALL UPPERCASE TEXT.' is categorized as: uppercase
'This Text Has Mixed Case.' is categorized as: mixed
```

Question 12: Scenario (isdigit()):

In a text processing pipeline, you need to identify and extract numerical values from a large dataset containing mixed alphanumeric text. These numerical values can represent various metrics, such as product quantities, temperatures, or timestamps. Question: Describe how you would use the isdigit() method to extract numerical values from a dataset containing mixed alphanumeric text, ensuring that only valid numerical values are captured.

In [175…

```
def extract_numericals(data):
  numerical_values = []
  current_number = ""

  for char in data:
    if char.isdigit():
      current_number += char  # Append digit to current number
    elif current_number:  # If a number was building and encountered a non-digit
      numerical_values.append(current_number)
      current_number = ""  # Reset for next potential number

  # Handle the last number if it reaches the end of the string
  if current_number:
    numerical_values.append(current_number)

  return numerical_values

# Sample data
data = "Yameen is 23 Years Old, The body temperature is 37 C, Height is 5.8"


# Extract numerical values
extracted_numbers = extract_numericals(data)

if extracted_numbers:
  print("Extracted numerical values:", extracted_numbers)
else:
  print("No numerical values found in the data.")
```

```
Extracted numerical values: ['23', '37', '5', '8']
```