string interview questions

1.

What are the methods to check if a string contains only alphabets, only digits, or both in Python?

- **Answer**:
    - `isalpha()`: Returns `True` if all characters in the string are alphabets.
    - `isdigit()`: Returns `True` if all characters in the string are digits.
    - `isalnum()`: Returns `True` if all characters in the string are either alphabets or digits.
    - Example:

    ```
    s1 = "Hello"
    s2 = "12345"
    s3 = "Hello123"
    print(s1.isalpha())  # Output: True
    print(s2.isdigit())  # Output: True
    print(s3.isalnum())  # Output: True
    ```

2. **How do string formatting methods `format()`, `f-strings` operator differ in Python?**

- **Answer**:
    - `format()`: Uses curly braces `{}` as placeholders which are replaced with values.

    ```
    pythonCopy code
    name = "Alice"
    age = 30
    print("My name is {} and I am {} years old.".format(nam
    e, age))
    ```

```
# Output: My name is Alice and I am 30 years old.
```

- `f-strings`: Prefixed with `f` or `F` and allows expressions inside curly braces `{}`.

  ```
  pythonCopy code
  name = "Alice"
  age = 30
  print(f"My name is {name} and I am {age} years old.")
  # Output: My name is Alice and I am 30 years old.
  ```

3. **What are raw strings in Python and how are they different from regular strings?**

- **Answer**: Raw strings are prefixed with `r` or `R` and treat backslashes (`\`) as literal characters. This is useful for dealing with regular expressions or file paths where backslashes are common.
  - Example:

    ```
    pythonCopy code
    regular_string = "C:\\Users\\Name\\Documents"
    raw_string = r"C:\Users\Name\Documents"
    print(regular_string)  # Output: C:\Users\Name\Docum
    ents
    print(raw_string)  # Output: C:\Users\Name\Documents
    ```

4. **What is the difference between `find()` and `index()` methods in Python?**

- **Answer**: Both `find()` and `index()` methods are used to locate the position of a substring within a string.
  - `find()`: Returns the lowest index of the substring if it is found, otherwise it returns `1`.

    - `index()`: Returns the lowest index of the substring if it is found, but raises a `ValueError` if the substring is not found.
    - Example:

      ```
      pythonCopy code
      s = "hello world"
      print(s.find("world"))  # Output: 6
      print(s.index("world"))  # Output: 6
      print(s.find("python"))  # Output: -1
      print(s.index("python"))  # Raises ValueError
      ```

5. **Explain the use of escape sequences in strings with examples.**
   - **Answer**: Escape sequences are special characters used in strings to represent certain whitespace characters, quotes, or other characters that are difficult to type directly. Common escape sequences include:
     - `\n` for newline
     - `\t` for tab
     - `\\` for backslash
     - `\'` for single quote
     - `\"` for double quote
     - Example:

       ```
       pythonCopy code
       s = "Hello,\nWorld!\tIt's a beautiful day."
       print(s)
       # Output:
       # Hello,
       # World!    It's a beautiful day.
       ```

6. **How can you check if a string starts with or ends with a particular substring in Python? Provide examples.**

- **Answer**: You can use the `startswith()` and `endswith()` methods. For example, `'Hello, World!'.startswith('Hello')` returns `True`, and `'Hello, World!'.endswith('World!')` returns `True`.

7. **Explain the use of regular expressions in Python for string manipulation. Provide an example of finding all email addresses in a given text.**
   - **Answer**: Regular expressions (regex) are patterns used for matching and manipulating strings. The `re` module in Python provides regex support. For example, to find all email addresses in a text:

```
import re
text = "Contact us at support@example.com or sales@example.com"
emails = re.findall(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b', text)
print(emails)  # Output: ['support@example.com', 'sales@example.com']
```

8. **Explain the concept of string immutability in Python. How does it affect string operations?**

- **Answer**: Strings in Python are immutable, meaning once they are created, their contents cannot be changed. Any operation that modifies a string returns a new string instead of altering the original. For example, `s = 'hello'` and `s.upper()` returns `'HELLO'`, but `s` remains `'hello'`.

9. **Explain the difference between single quotes, double quotes, and triple quotes in Python.**

- **Answer**: Single and double quotes are used interchangeably for creating strings. Triple quotes (`'''` or `"""`) are used for creating multi-line strings or strings that contain both single and double quotes.

10. **What is the difference between `replace()` and `re.sub()` for string replacement in Python?**

- **Answer**:
    - `replace()`: Replaces all occurrences of a specified substring with another substring.
    - `re.sub()`: Uses regular expressions to replace matching patterns in a string.
    - Example:

```
pythonCopy code
import re
s = "Hello, World! Hello, Python!"
# Using replace()
print(s.replace("Hello", "Hi"))  # Output: "Hi, World!
Hi, Python!"
# Using re.sub()
print(re.sub(r"Hello", "Hi", s))  # Output: "Hi, World!
Hi, Python!"
```

Explain the difference between match() and search() in regular expressions.

`match()` checks for a match only at the beginning of a string, while `search()` looks for a match anywhere in the string.

What is string interpolation in Python?

String interpolation is the process of evaluating a string containing placeholders to produce a new string with the placeholders replaced.

What is the difference between str() and repr()?

`str()` is used for creating output for end-users, while `repr()` is mainly used for development and debugging, providing a more unambiguous representation.

What is the purpose of the findall() method in regular expressions?

The `findall()` method returns all non-overlapping matches as a list.

What is Unicode?

Unicode is a standardized character encoding system that assigns unique code points to characters.

Coding questions :

1. Reverse a String

Code:

```
pythonCopy code
def reverse_string(input_string):
    return input_string[::-1]

# Example usage
print(reverse_string("Python"))  # Output: nohtyP
```

Explanation: This function uses slicing to reverse the string.

2. Check if a String is a Palindrome

Code:

```
pythonCopy code
def is_palindrome(input_string):
    return input_string == input_string[::-1]

# Example usage
print(is_palindrome("madam"))  # Output: True
print(is_palindrome("python"))  # Output: False
```

Explanation: The function checks if the string reads the same forwards and backwards.

3. Count the Number of Vowels in a String

Code:

```
pythonCopy code
def count_vowels(input_string):
    vowels = "aeiouAEIOU"
    return sum(1 for char in input_string if char in vowels)

# Example usage
print(count_vowels("Python"))  # Output: 1
```

Explanation: This function counts the vowels in the string by iterating over each character and checking if it's in the `vowels` set.

4. Find the Most Frequent Character in a String

Code:

```
pythonCopy code
def most_frequent_char(input_string):
    char_count = {}
    for char in input_string:
        char_count[char] = char_count.get(char, 0) + 1
    return max(char_count, key=char_count.get)

# Example usage
print(most_frequent_char("Python"))  # Output: 'P' (or any ch
aracter that appears most frequently)
```

Explanation: The function counts the frequency of each character and returns the character with the highest count.

---

5. Remove All Duplicates from a String

Code:

```
pythonCopy code
def remove_duplicates(input_string):
    return "".join(sorted(set(input_string), key=input_strin
g.index))

# Example usage
print(remove_duplicates("Python"))  # Output: "Pytho"
```

Explanation: This function uses a set to remove duplicates while preserving the order of characters.

---

6. Check if a String Contains Only Digits

Code:

```
pythonCopy code
def contains_only_digits(input_string):
    return input_string.isdigit()

# Example usage
print(contains_only_digits("12345"))  # Output: True
print(contains_only_digits("123a45"))  # Output: False
```

Explanation: The function uses `isdigit()` to check if all characters in the string are digits.

---

7. Count the Occurrences of Each Word in a Sentence

Code:

```
pythonCopy code
def count_word_occurrences(input_string):
    words = input_string.split()
    word_count = {}
    for word in words:
        word_count[word] = word_count.get(word, 0) + 1
    return word_count

# Example usage
print(count_word_occurrences("hello world hello"))  # Output:
{'hello': 2, 'world': 1}
```

Explanation: This function counts the occurrences of each word in the sentence and returns a dictionary with the counts.

---

8. Capitalize the First Letter of Each Word in a Sentence

Code:

```
pythonCopy code
def capitalize_first_letter(input_string):
    return " ".join(word.capitalize() for word in input_strin
g.split())

# Example usage
print(capitalize_first_letter("hello world"))  # Output: "Hel
lo World"
```

Explanation: The function capitalizes the first letter of each word and joins them back into a string.

9. Find the Longest Word in a Sentence

Code:

```
pythonCopy code
def longest_word(input_string):
    words = input_string.split()
    return max(words, key=len)

# Example usage
print(longest_word("Find the longest word here"))  # Output:
"longest"
```

Explanation: The function splits the sentence into words and returns the longest one.

10. Check if Two Strings are Anagrams

Code:

```
pythonCopy code
def are_anagrams(str1, str2):
    return sorted(str1) == sorted(str2)

# Example usage
print(are_anagrams("listen", "silent"))  # Output: True
print(are_anagrams("hello", "world"))    # Output: False
```

Explanation: This function checks if two strings have the same characters in any order.

11. Count the Number of Occurrences of a Specific Substring in a String

Code:

```
pythonCopy code
def count_substring_occurrences(input_string, substring):
    count = 0
    index = 0
    while True:
        index = input_string.find(substring, index)
        if index == -1:
            break
        count += 1
        index += len(substring)  # Move past the last occurre
nce
    return count

# Example usage
print(count_substring_occurrences("ababab", "ab"))  # Output:
3
```

Explanation: The function counts the occurrences of a substring in the string, ensuring overlapping matches are counted.

12. Find the Index of the First Occurrence of a Substring in a String

Code:

```
pythonCopy code
def find_substring(input_string, substring):
    return input_string.find(substring)

# Example usage
print(find_substring("hello world", "world"))  # Output: 6
```

Explanation: The function uses `find()` to locate the first occurrence of the substring.

---

13. Replace All Occurrences of a Substring with Another Substring

Code:

```
pythonCopy code
def replace_substring(input_string, old_substring, new_substr
ing):
    return input_string.replace(old_substring, new_substring)

# Example usage
print(replace_substring("hello world", "world", "there"))  #
Output: "hello there"
```

Explanation: The function replaces all instances of `old_substring` with `new_substring`.

---

14. Find the Length of the Last Word in a Sentence

Code:

```
pythonCopy code
def length_of_last_word(input_string):
    words = input_string.split()
    if words:
        return len(words[-1])
    return 0

# Example usage
print(length_of_last_word("hello world"))  # Output: 5
```

Explanation: This function finds the length of the last word by splitting the sentence and checking the last word's length.

15. Reverse the Order of Words in a Sentence

Code:

```
pythonCopy code
def reverse_words_order(input_string):
    words = input_string.split()
    return " ".join(reversed(words))

# Example usage
print(reverse_words_order("hello world"))  # Output: "world h
ello"
```

Explanation: The function reverses the order of words in the sentence and joins them into a new string.

16. Check if a String is a Valid Email Address

Code:

```
pythonCopy code
import re

def is_valid_email(email):
    pattern = r'^[\w\.-]+@[a-zA-Z\d\.-]+\.[a-zA-Z]{2,}$'
    return bool(re.match(pattern, email))

# Example usage
print(is_valid_email("example@example.com"))  # Output: True
print(is_valid_email("invalid-email"))  # Output: False
```

Explanation: The function uses a regular expression to validate the email format.

17. Check if a String is a Valid URL

Code:

```
pythonCopy code
import re

def is_valid_url(url):
    pattern = r'^https?://(?:[-\w.]|(?:%[\da-fA-F]{2}))+'
    return bool(re.match(pattern, url))

# Example usage
print(is_valid_url("https://www.example.com"))  # Output: Tru
e
print(is_valid_url("invalid-url"))  # Output: False
```

Explanation: The function checks if the URL matches a regular expression pattern for valid URLs.

18. Find the First Non-Repeated Character in a String

Code:

```
pythonCopy code
from collections import Counter

def first_non_repeated_char(input_string):
    char_count = Counter(input_string)
    for char in input_string:
        if char_count[char] == 1:
            return char
    return None

# Example usage
print(first_non_repeated_char("swiss"))  # Output: 'w'
```

Explanation: The function uses `Counter` to count characters and returns the first character with a count of 1.

---

19. Remove All Leading and Trailing Whitespaces from a String

Code:

```
pythonCopy code
def trim_whitespace(input_string):
    return input_string.strip()

# Example usage
print(trim_whitespace("  hello world  "))  # Output: "hello w
orld"
```

Explanation: The function uses `strip()` to remove leading and trailing whitespace from the string.

---

20. Find the Common Characters Between Two Strings

Code:

```
pythonCopy code
def common_characters(str1, str2):
    return "".join(char for char in set(str1) if char in str
2)

# Example usage
print(common_characters("hello", "world"))  # Output: "lo"
```

Explanation: The function finds and returns characters that are present in both strings.

---

21. Find the Second Most Frequent Character in a String

Code:

```
pythonCopy code
def second_most_frequent_char(input_string):
    char_count = {}
    for char in input_string:
        char_count[char] = char_count.get(char, 0) + 1
    sorted_char_count = sorted(char_count.items(), key=lambda
x: x[1], reverse=True)
    return sorted_char_count[1][0] if len(sorted_char_count)
>= 2 else None

# Example usage
print(second_most_frequent_char("aabbbc"))  # Output: 'b'
```

Explanation: The function sorts characters by frequency and returns the second most frequent character.

---

22. Check if a String Contains Only Unique Characters

Code:

```
pythonCopy code
def has_unique_characters(input_string):
    return len(set(input_string)) == len(input_string)

# Example usage
print(has_unique_characters("abcdef"))  # Output: True
print(has_unique_characters("aabbcc"))  # Output: False
```

Explanation: The function checks if the length of the set (unique characters) is equal to the length of the string.

---

23. Find the Longest Common Prefix Among a List of Strings

Code:

```
pythonCopy code
def longest_common_prefix(strings):
    if not strings:
        return ""
    prefix = strings[0]
    for string in strings[1:]:
        i = 0
        while i < len(prefix) and i < len(string) and prefix
[i] == string[i]:
            i += 1
        prefix = prefix[:i]
    return prefix

# Example usage
print(longest_common_prefix(["flower", "flow", "flight"]))  #
Output: "fl"
```

Explanation: The function finds the common prefix among all strings by comparing characters.

24. Check if a String is a Valid IPv4 Address

Code:

```
pythonCopy code
import socket

def is_valid_ipv4(ipv4):
    try:
        socket.inet_pton(socket.AF_INET, ipv4)
        return True
    except socket.error:
        return False

# Example usage
print(is_valid_ipv4("192.168.1.1"))  # Output: True
print(is_valid_ipv4("999.999.999.999"))  # Output: False
```

Explanation: The function uses `socket.inet_pton` to validate the IPv4 address format.

25. Find the First Non-Repeated Character in a String Using OrderedDict

Code:

```
pythonCopy code
from collections import OrderedDict

def first_non_repeated_char_ordered(input_string):
    char_count = OrderedDict()
    for char in input_string:
        char_count[char] = char_count.get(char, 0) + 1
    for char, count in char_count.items():
        if count == 1:
            return char
    return None

# Example usage

print(first_non_repeated_char_ordered("swiss"))  # Output:
'w'
```

26. Implement Regular Expression Matching with Support for `'.'` and `'*'`

Code:

```
pythonCopy code
import re

def is_match(s: str, p: str) -> bool:
    return re.fullmatch(p, s) is not None

# Example usage
print(is_match("aa", "a*"))  # Output: True
print(is_match("aab", "c*a*b"))  # Output: True
print(is_match("mississippi", "mis*is*p*."))  # Output: False
```

Explanation: This function uses the `re.fullmatch()` method to check if the entire string `s` matches the pattern `p`. The `'.'` matches any character, and `'*'` means zero or more of the preceding element.

---

27. Check if a String Contains Only Alphabets, Only Digits, or Both

Answer:

- `isalpha()`: Returns `True` if all characters in the string are alphabets.
- `isdigit()`: Returns `True` if all characters in the string are digits.
- `isalnum()`: Returns `True` if all characters in the string are either alphabets or digits.

Code Example:

```
pythonCopy code
s1 = "Hello"
s2 = "12345"
s3 = "Hello123"

print(s1.isalpha())  # Output: True
print(s2.isdigit())  # Output: True
print(s3.isalnum())  # Output: True
```

Explanation: `isalpha()` checks for alphabetic characters only, `isdigit()` checks for numeric characters only, and `isalnum()` checks for alphanumeric characters.

---

28. Remove All Duplicate Characters from a String While Maintaining the Order

Answer:
You can use a set to keep track of characters that have been seen and a list to maintain the order.

Code Example:

```
pythonCopy code
def remove_duplicates(s):
    seen = set()
    result = []
    for char in s:
        if char not in seen:
            seen.add(char)
            result.append(char)
    return ''.join(result)

# Example usage
s = "abracadabra"
print(remove_duplicates(s))  # Output: "abracd"
```

Explanation: The function iterates through the string, adds unseen characters to the result, and maintains their order.

---

29. Reverse a String in Python

Answer:
There are several methods to reverse a string:

- **Slicing:**

  ```
  pythonCopy code
  s = "Python"
  reversed_string = s[::-1]
  print(reversed_string)  # Output: "nohtyP"
  ```

- **Loop and Concatenation:**

  ```
  pythonCopy code
  s = "Python"
  reversed_string = ''
  for char in s:
      reversed_string = char + reversed_string
  print(reversed_string)  # Output: "nohtyP"
  ```

- **Using `reversed()` and `join()`:**

```
pythonCopy code
s = "Python"
reversed_string = ''.join(reversed(s))
print(reversed_string)  # Output: "nohtyP"
```

Explanation: Slicing is the most concise method, while looping and using `reversed()` with `join()` are more explicit.

---

30. Difference Between `split()` and `splitlines()` Methods

Answer:

- `split()`: Splits a string into a list using a specified delimiter (default is whitespace).

  Code Example:

  ```
  pythonCopy code
  text = "Hello, World!"
  words = text.split()
  print(words)  # Output: ['Hello,', 'World!']
  ```

- `splitlines()`: Splits a string at line breaks (`\n`, `\r`, or `\r\n`) and returns a list of lines.

  Code Example:

  ```
  pythonCopy code
  text = "Hello\nWorld!\nPython"
  lines = text.splitlines()
  print(lines)  # Output: ['Hello', 'World!', 'Python']
  ```

Explanation: `split()` is used for breaking a string into parts based on a delimiter, while `splitlines()` is specifically for handling multiline strings by breaking them at line breaks.