

Python Comprehensions Interview Questions

What are list comprehensions in Python?

- List comprehensions provide a concise way to create lists in Python. It consists of brackets **containing** an expression followed by a `for` clause, then zero or more `for` or `if` clauses.
- The result will be a new list resulting from evaluating the expression in the context of the `for` and `if` clauses.

Example:

Without list comprehension

```
squares = []  
for x in range(10):  
    squares.append(x**2)
```

With list comprehension

```
squares = [x**2 for x in range(10)]  
print(squares)  # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

How do you use list comprehensions to filter elements?

- You can include an `if` statement in the list comprehension to filter elements based on a condition.

Example:

Filtering even numbers

```
even_numbers = [x for x in range(10) if x % 2 == 0]
```

How to Find the squares of numbers from 1 to 10:

```
squares = [x**2 for x in range(1, 11)]  
print(squares)
```

How to Filter even numbers from a list:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
even_numbers = [x for x in numbers if x % 2 == 0]  
print(even_numbers)  # [2, 4, 6, 8, 10]
```

What are dictionary comprehensions in Python?

- Dictionary comprehensions are similar to list comprehensions but create dictionaries instead.
- They use curly braces `{}` and key-value pairs.

Example:

Creating a dictionary of squares

```
squares_dict = {x: x**2 for x in range(5)}
```

How to Create a dictionary mapping numbers to their squares:

```
numbers = [1, 2, 3, 4, 5]
```

```
squares_dict = {x: x**2 for x in numbers}
```

```
print(squares_dict)
```

Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

How to Count the frequency of characters in a string:

```
string = "hello world"
```

```
char_freq = {char: string.count(char) for char in string}
```

```
print(char_freq)
```

Output: {'h': 1, 'e': 1, 'l': 3, 'o': 2, ' ': 1, 'w': 1, 'r': 1, 'd': 1}

How to Extract vowels from a string:

```
string = "hello world"
```

```
vowels = [char for char in string if char in 'aeiouAEIOU']
```

```
print(vowels)
```

Output : ['e', 'o', 'o']

Flatten a nested list:

```
nested_list = [[1, 2, 3], [4, 5], [6, 7, 8]]
```

```
flattened_list = [item for sublist in nested_list for item in sublist]
```

```
print(flattened_list)
```

Output : [1, 2, 3, 4, 5, 6, 7, 8]

Generate a list of prime numbers:

```
def is_prime(n):  
    if n <= 1:  
        return False  
    for i in range(2, int(n**0.5) + 1):  
        if n % i == 0:  
            return False  
    return True  
  
primes = [x for x in range(2, 101) if is_prime(x)]  
  
print(primes)
```

Output: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

Explain the syntax of set comprehensions.

- Set comprehensions are similar to list comprehensions, but they produce sets instead of lists.
- They use curly braces `{}` like dictionary comprehensions but without key-value pairs.

Example:

Creating a set of unique characters from a string

```
unique_chars = {char for char in 'hello'}
```

When would you use generator expressions instead of list comprehensions?

- Generator expressions are used when you want to iterate over a sequence but do not need to store the entire sequence in memory at once.
- They are memory efficient compared to list comprehensions. Generator expressions use parentheses `()` instead of square brackets `[]`.

Example:

Using generator expression

```
even_numbers_gen = (x for x in range(10) if x % 2 == 0)
```

Generate squares of numbers from 1 to 10:

```
squares = (x**2 for x in range(1, 11))  
for square in squares:  
    print(square)
```

Output: 1 4 9 16 25 36 49 64 81 100

Filter even numbers from a list using generator comprehensions :

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = (x for x in numbers if x % 2 == 0)
for even_number in even_numbers:
    print(even_number, end=" ")
```

Output: 2 4 6 8 10

Generate a list of powers of 2:

```
powers_of_2 = (2**x for x in range(10))
for power in powers_of_2:
    print(power)
```

Generate uppercase versions of strings in a list:

```
strings = ["hello", "world", "python"]
uppercase_strings = (s.upper() for s in strings)
for uppercase_string in uppercase_strings:
    print(uppercase_string, end=" ")
```

Output: HELLO, WORLD, PYTHON,

What is the difference between a list comprehension and a generator expression?

- List comprehensions return a list containing the results, while generator expressions return a generator object, which generates values lazily as they are needed.
- Generator expressions are more memory efficient, especially for large sequences, as they produce values on-the-fly rather than storing them all in memory at once.

Explain how you can use compression techniques to handle large data sets efficiently in Python.

- Compression techniques such as `gzip`, `zlib`, or `bz2` modules in Python can be used to compress and decompress large data sets efficiently.
- For instance, you can compress data before storing it and decompress it when needed, which helps reduce storage space and transfer time, especially when dealing with large volumes of data.
- These modules provide functions like `gzip.compress()` and `gzip.decompress()` to perform compression and decompression operations.