Daily Tasks - Interview Questions 📝

List Methods

`Date: 2024-11-11`

append(item)

- Adds the item at the very end of the list
- Doesn't return anything, it directly updates the list itself

```
expenses = [200, 150, 300]
expenses.append(100)  #today's expense
print(expenses)
```

```
[200, 150, 300, 100]
```

clear()

This method wipes out all the items in a list, leaving it completely empty but still ready to be reused.

- Removes every item from the list but doesn't delete the list itself
- Great when you want to start fresh with the same list for new data

```
inventory = ['Apples', 'Oranges', 'Bananas']  # Last month's stock
inventory.clear()  # Clearing out the old stock details
print(inventory)
```

```
[]
```

copy()

- Creates a new list, leaving the original list unchanged
- Useful when you need to create a duplicate list and keep the original safe from unintended changes

```
recipe = ['flour', 'sugar', 'eggs']
new_recipe = recipe.copy()
new_recipe.append('chocolate')  # Adding a new ingredient
print(recipe)
print(new_recipe)
```

```
['flour', 'sugar', 'eggs']
['flour', 'sugar', 'eggs', 'chocolate']
```

Reassigning the copied list to the original list reference makes them point to the same object.

```
recipe = ['flour', 'sugar', 'eggs']
new_recipe = recipe.copy()
recipe = new_recipe
new_recipe.append('chocolate')  # Adding a new ingredient
print(recipe)
print(new_recipe)
```

```
['flour', 'sugar', 'eggs', 'chocolate']
['flour', 'sugar', 'eggs', 'chocolate']
```

count(value)

**This method helps you count how many times a specific value appears in a list **

```
#Say you're keeping track of how often you visit different coffee shops in a week
visits = ['Starbucks', 'Banyan Tree Cafe, Mumbai', 'Araku Coffee, Bengaluru', 'The Daily Cafe, Kolkata', 'Starbucks' ]
print(visits.count('Starbucks'))
```

```
2
```

extend(iterable)

- Adds all items from another iterable to the end of your list
- It's faster and more efficient than using + for list concatenation

```
groceries = ['milk', 'bread']
extras = ['eggs', 'butter']
groceries.extend(extras) # will adds up items to the groceries
print(groceries)
```

```
['milk', 'bread', 'eggs', 'butter']
```

index(value, start=0, end=len(list))

- Returns the index of the first occurrence of the specified value
- If the value is not found, it will raise ValueError
- You can specify a starting and ending point to limit the search

```
books = ['Python', 'C++', 'Java', 'Python']
print(books.index('Python'))
```

```
0
```

```
books = ['Python', 'C++', 'Java', 'Python']
print(books.index('PHP'))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-12-6cfe02cb14d7> in <cell line: 2>()
      1 books = ['Python', 'C++', 'Java', 'Python']
----> 2 print(books.index('PHP'))

ValueError: 'PHP' is not in list
```

insert(index, item)

- Adds the item at the specified index, and shifts the other items to the right
- This is handy when you want to maintain an ordered list but need to insert something in the middle

```
#Say you're scheduling meetings and want to add a new meeting in the middle >>>

schedule = ["9 AM", '11 AM', '2 PM']
schedule.insert(1, '10 AM')
print(schedule)
```

```
['9 AM', '10 AM', '11 AM', '2 PM']
```

pop(index=-1)

The pop() method removes and returns the element at the specified index

- By default, removes the last item if no index is specified

```
#Say you've just completed a Coding task and want to remove it from your to-do list
tasks = ['Email', 'Meeting', 'Coding']
completed_task = tasks.pop()
print(completed_task)
print(f"Remaining tasks: {tasks}")
```

```
Coding
Remaining tasks: ['Email', 'Meeting']
```

remove(value)

- Removes the first instance of the given value in the list
- If the value isn't found, it will raise a ValueError

```
#Imagine you need to remove an expired coupon from your list of saved coupons
coupons = ['SAVE10', 'NEWUSER24', 'SAVE10', 'Diwali40']
coupons.remove('SAVE10')
print(coupons)
```

```
['NEWUSER24', 'SAVE10', 'Diwali40']
```

reverse()

The reverse() method reverses the order of the elements in the list in place, meaning it changes the list directly and doesn't create a new one.

- It's useful when you want to flip the order without sorting the items

```
#If you want to view a list of dates in reverse chronological order >>>

dates = ['2024-11-10', '2024-11-11', '2024-11-12']
dates.reverse()
print(dates)
```

```
['2024-11-12', '2024-11-11', '2024-11-10']
```

sort(key=None, reverse=False)

- sorts the list in ascending order(default)
- You can specify a custom sorting rule using the `key` argument
- The `reverse argument` allows you to sort in descending order if set to `True`

```
#Let's say you want to sort names alphabetically

names = ['John', 'Alice', 'Bob']
names.sort()
print(names)
```

```
['Alice', 'Bob', 'John']
```

A Function?

***A function is a standalone block of code designed to perform a specific

- It is called directly using its name
- Operates on the arguments passed to it
- Functions are not bound to any object
- They can be used on various data types, depending on their implementation
- deepcopy() works on any compatible data structure, not just lists
- It is called directly by its name because it's a function

Example >>> deepcopy()

The deepcopy() function from the copy module creates a deep copy of an object.

```
from copy import deepcopy

Nested_list = [[1, 2], [3, 4]]
deep_copied = deepcopy(Nested_list)

#Copied object won't affect the original
deep_copied[0][0] = 99
print(Nested_list)
print(deep_copied)
```

```
[[1, 2], [3, 4]]
[[99, 2], [3, 4]]
```

A Method?

A method is a function that is associated with an object and operates on that object

- It is called on the object using dot notation >> (original.copy())
- Methods are designed to manipulate or interact with the specific object they are bound to

Example: copy()

The `copy() method` creates a `shallow copy` of a list object

```
original = [[1, 2], [3, 4]]
shallow_copied = original.copy()

#nested list affects both copies
shallow_copied[0][0] = 99
print(original)
print(shallow_copied)
```

```
[[99, 2], [3, 4]]
[[99, 2], [3, 4]]
```

`Difference Between sort() and sorted()`

**sort() >>> `Modifies` the `original list` and `returns None`

sorted() >>> `Returns a new sorted list`, leaving the original `list unchanged` **

```
guests = ['Daksha', 'Ridhi', 'Navya']
sorted_list_of_guests = sorted(guests)
print(guests)
print(sorted_list_of_guests)
```

```
['Daksha', 'Ridhi', 'Navya']
['Daksha', 'Navya', 'Ridhi']
```