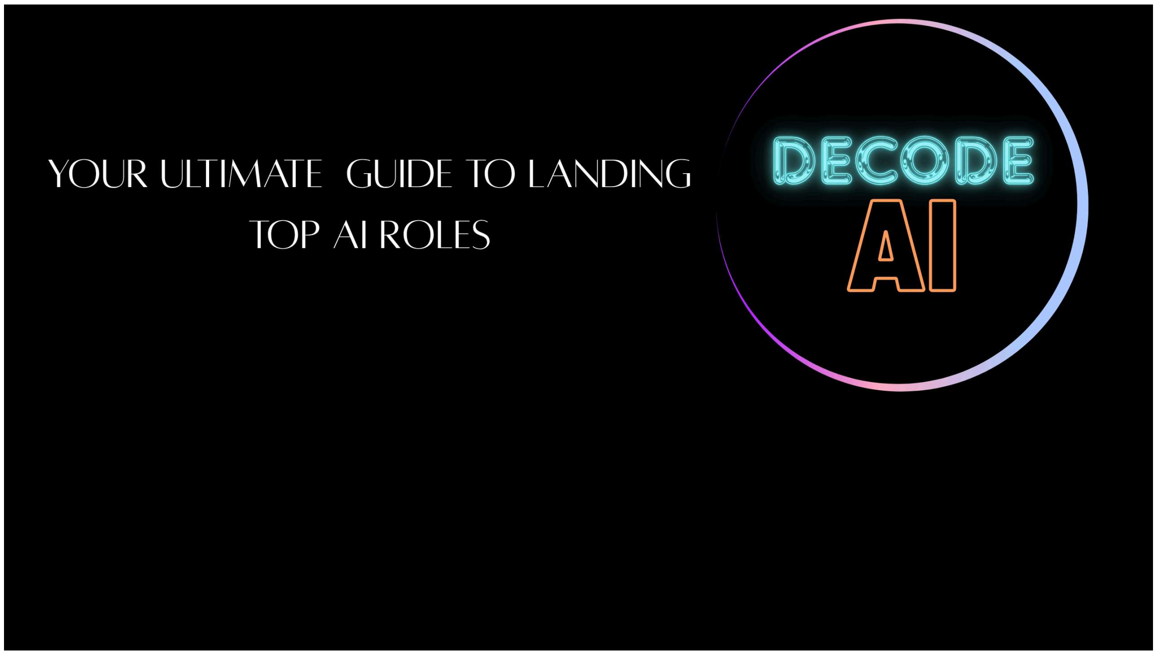

Lists in Python

In Python, Lists can be considered as the most general version of a "sequence". Unlike strings, they are mutable which means the elements inside a list can be changed!

Lists are constructed with brackets `[]` and commas separating every element in the list.

✅ **Key Features of Lists:**

- **Ordered**: Items have a defined order and can be accessed by index.
- **Mutable**: You can change, add, or remove items after the list is created.
- **Allows duplicates**: Lists can have repeated values.
- **Heterogeneous**: Can store different data types together.

Hands on Time

Examples of list

In [5]:
```python
# Examples of list

# 1. Homogenous Data
numbers = [10, 20, 30, 40, 50] # List of Int
fruits = ["apple", "banana", "cherry"] # List of strings
```

```
bools = [True, False, True, False] # List of boolean
temperatures = [36.6, 37.0, 36.4] # List of float
```

In [6]:
```
# 2. Hetrogenous Data
mixed = [101, "Alice", True, 99.5] # contains only primitive data
employee = ["John", 32, 5.8, True, ["Python", "C++"]] # nested - List of List
```

In [7]:
```
my_list = ['A string',23,100.232,'o']
```

In [8]:
```
len(my_list)
```

Out[8]:
```
4
```

Indexing and Slicing

Slicing allows you to extract a portion of a list using a clean, flexible syntax.

`list[start : stop : step]`

Indexing and slicing of lists works just like in Strings. Let's make a new list to remind ourselves of how this works:

In [9]:
```
my_list = ['one','two','three', 4, 5]
```

In [10]:
```
# Grab element at index 0
my_list[0]
```

Out[10]:
```
'one'
```

In [11]:
```
# Grab index 1 and everything past it
my_list[1:]
```

Out[11]:
```
['two', 'three', 4, 5]
```

In [12]:
```
# Grab everything UP TO index 3
my_list[:3]
```

Out[12]:
```
['one', 'two', 'three']
```

We can also use "+" to concatenate lists, just like we did for Strings.

In [13]:
```
my_list + ['new item']
```

Out[13]:
```
['one', 'two', 'three', 4, 5, 'new item']
```

In [14]:
```
# The original list doesnot change
my_list
```

Out[14]:
```
['one', 'two', 'three', 4, 5]
```

In this case, you have to reassign the list to make the permanent change.

In [15]:
```
# Reassign
my_list = my_list + ['add new item permanently']
```

In [16]:
```
my_list
```

Out[16]:
```
['one', 'two', 'three', 4, 5, 'add new item permanently']
```

In [17]:
```
my_list * 2
```

Out[17]:
```
['one',
 'two',
 'three',
 4,
 5,
 'add new item permanently',
 'one',
 'two',
 'three',
 4,
 5,
 'add new item permanently']
```

Basic List Methods

In [18]:
```
# Create a new list
l = [1,2,3]
```

In [19]:
```
# Append
l.append('append me!')
```

In [20]:
```
# Show
l
```

Out[20]:
```
[1, 2, 3, 'append me!']
```

In [22]:
```
# list.pop([index]) -> Pop off the 0 indexed item
l.pop(0)
```

Out[22]:
```
2
```

In [23]:
```
# Show
l
```

Out[23]:
```
[3, 'append me!']
```

In [24]:
```
# Assign the popped element, remember default popped index is -1
popped_item = l.pop()
```

Sort and Reverse in Lists

In [25]:
```
new_list = ['a','e','x','b','c']
```

In [26]:
```
#Show
new_list
```

Out[26]:
```
['a', 'e', 'x', 'b', 'c']
```

In [27]:
```
# list.reverse() -> Use reverse to reverse order (this is permanent!)
new_list.reverse()
```

In [28]:
```
new_list
```

Out[28]:
```
['c', 'b', 'x', 'e', 'a']
```

In [29]:
```
# list.sort(key=None, reverse=False) => Use sort to sort the list (in this case alp
new_list.sort()
```

In [30]:
```
new_list
```

Out[30]:
```
['a', 'b', 'c', 'e', 'x']
```

In [ ]:
```

```

Nesting Lists

Nesting Lists is one of the great features in Python data structures. Nesting Lists means we can have data structures within data structures.

For example: A list inside a list.

In [31]:
```
# Let's make three lists
lst_1=[1,2,3]
lst_2=[4,5,6]
lst_3=[7,8,9]

# Make a list of lists to form a matrix
matrix = [lst_1,lst_2,lst_3]
```

In [32]:
```
# Show
matrix
```

Out[32]:
```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

In [33]:
```
# Grab first item in matrix object
matrix[0]
```

```
Out[33]: [1, 2, 3]
```

List Comprehensions

Python has an advanced feature called list comprehensions which allows for quick construction of lists.

Here are few of oue examples which helps you to understand list comprehensions.

```
In [34]: matrix
```

```
Out[34]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
In [35]: # Problem: from matrix list extract the first column data.
```

```
In [36]: # Build a list comprehension by deconstructing a for loop within a []
         first_col = [row[0] for row in matrix]
```

```
In [37]: first_col
```

```
Out[37]: [1, 4, 7]
```

some miscellaneous List methods

```
In [38]: l = [1,2,3]
```

```
In [39]: # list.append(element) -> append in list
         l.append(4)
```

```
In [40]: # list.count(element) -> count in list
         l.count(3)
```

```
Out[40]: 1
```

```
In [42]: # list.append(element) -> append in list

         x = [1, 2, 3]
         x.append([4, 5])
         print(x)
```

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

```
In [43]: # list.extend(iterable) -> extend in list

         x = [1, 2, 3]
         x.extend([4, 5])
         print(x)
```

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

```
In [45]: # list.index(element, start=0, end=len(list)) ->index in list - Make a note that if
         l.index(2)

Out[45]: 1

In [46]: # insert(index,object) in list
         l = [1,2,3,4]

         # Place a letter at the index 2
         l.insert(2,'inserted')
         print(l)

[1, 2, 'inserted', 3, 4]

In [47]: # pop() in list
         l = [1,2,3,4]
         l.pop()

Out[47]: 4

In [48]: print(l)

[1, 2, 3]

In [49]: # list.remove(element) - The remove() method removes the first occurrence of a valu
         l = [1, 2, 'inserted', 3, 4]
         l.remove('inserted')

In [50]: print(l)

[1, 2, 3, 4]

In [ ]:
```

📚 References

1. https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str
2. https://arpitbhayani.me/blogs/string-interning-python/

📘 Curated by Team Decode-AI

Decoding AI Interviews at Top-Tech Companies