1. **What is a dictionary in Python ?**

   Dictionary is a heterogeneous , dynamic data structure which can hold data in the form of key value pairs. Here the key has to be unique but the value can hold duplicate values.

   In Python, a dictionary is a collection of key-value pairs. It is also known as an associative array, a hash table, or a map in other programming languages.

   In a dictionary, each key is unique and is associated with a value. The keys are used to index and access the values, similar to the way you would use an index to access an element in a list.

   And you can remove a key-value pair like this:

   del my_dict["apple"]

2. **Explain the difference between a list and a tuple in Python ?**

   List is a heterogeneous , dynamic data structure which can hold duplicate data but a tuple is similar to list but it is not dynamic.

   There are many methods supported by list but tuple only supports 2 methods.

   Tuple takes less memory and is faster to iterate compared to a list.

   Tuple is formed by () while list is formed by []

3. **How can you add an element to a list in Python ?**

   Now we can add element to a list via two ways:

   1) list_name.appned(new_element)

   2) list_name.extend(new_list)

   3) list_name + [new_element]

   4) list_name.insert(position , element)

4. **What is a set in Python ?**

   Set is a heterogeneous, unordered form of data structure which is dynamic but cannot hold duplicate value.

   We can create a set using the {}.

   To add a element to a set we can use
   > set_name.add(element)

   To remove a element from a set:
   > set_name.remove(element)

   We can perform union , intersection , and difference on set operations.

5. **How can you iterate over the keys and values of a dictionary in Python ?**

   We can use a for loop or a while to iterate over a dictionary

   ⇒

   ```
   for i in dict_name.keys() / dict_name:
       print(dict_name[i])
   ```

   ⇒

   ```
   for i in dict_name.values():
       print(dict_name[i])
   ```

   ⇒

   ```
   for i in dict_name.items():
       print(i) → Will print the dict in the form of (key , value)
   ```

6. **How can you sort a list in Python?**

   We can sort a list by either using built-in methods or by creating our own algorithm to sort a list.

   list_name.sort() / list_name.sort(reverse = True)

   sorted(list_name) ⇒ returns a sorted list in ascending order

   sorted(list_name , reverse = True) ⇒ returns in sorted list in descending order

   sorted(list_name , key = function) ⇒ will sort according to a specific conditions

7. **What is a generator in Python?**
   A generator in Python is a special type of function that allows you to generate a series of values over time, rather than computing and returning them all at once. Generators are used to create iterators, which are objects that produce a sequence of values on demand.

   Generators use the yield keyword to produce values one at a time.'

```python
def square_generator(limit):
    for i in range(limit):
        yield i ** 2

# Use the generator in a for loop
for x in square_generator(5):
    print(x)
```

8. **What is the difference between a shallow copy and a deep copy in Python?**
   A shallow copy creates a new object that references the same memory locations as the original object. In other words, the new object is a separate object, but it still points to the same underlying data as the original object. Changes made to the data in the new object will be reflected in the original object, and vice versa.
   On the other hand, a deep copy creates a new object with a new memory address, and all the objects and data contained within it are also copied. The new object is completely

independent of the original object, and changes made to the data in one object will not affect the other.

```python
import copy

# Original list
original_list = [1, 2, [3, 4]]

# Shallow copy
shallow_copy = copy.copy(original_list)

# Deep copy
deep_copy = copy.deepcopy(original_list)

# Modify the sublist in the shallow copy
shallow_copy[2][0] = 99

# Modify the sublist in the deep copy
deep_copy[2][0] = 88

# Print the original list and the copies
print(original_list) # Output: [1, 2, [99, 4]]
print(shallow_copy) # Output: [1, 2, [99, 4]]
print(deep_copy) # Output: [1, 2, [88, 4]]
```

9. **What is a namedtuple in Python?**

   A namedtuple in Python is a subclass of the built-in tuple class. It is a lightweight data structure that behaves like a tuple, but also has named fields that can be accessed using dot notation, just like object attributes.

   namedtuple provides a way to create small, immutable, and easily-readable objects. It's particularly useful when you need to represent simple data objects with a fixed number of fields.

```python
from collections import namedtuple

# Define a named tuple
Person = namedtuple('Person', ['name', 'age', 'gender'])

# Create an instance of the named tuple
person1 = Person(name='Alice', age=25, gender='Female')

# Access the fields using dot notation
print(person1.name) # Output: 'Alice'
print(person1.age) # Output: 25
print(person1.gender) # Output: 'Female'
```

One of the main advantages of using a namedtuple instead of a regular tuple is that the fields have names, which makes the code more readable and self-documenting. Another advantage is that namedtuple instances can be accessed by name or by index, just like regular tuples.

10. **How do you check if a given value exists in a list in Python?**

    To check if a given value exists in a list in Python, you can use the in operator. The in operator returns a boolean value True if the given value is present in the list, and False otherwise.

    To check if a given value exists in a list in Python, you can use the in operator. The in operator returns a boolean value True if the given value is present in the list, and False otherwise.

11. **What is the difference between the append() and extend() methods for lists in Python?**

    append appends a specified object at the end of the list.

    extend extends the list by appending elements from the specified iterable.

12. **What is a dictionary in Python?**

Dictionary is a heterogeneous , dynamic data structure which can hold data in the form of key value pairs. Here the key has to be unique but the value can hold duplicate values.

In Python, a dictionary is a collection of key-value pairs. It is also known as an associative array, a hash table, or a map in other programming languages.

In a dictionary, each key is unique and is associated with a value. The keys are used to index and access the values, similar to the way you would use an index to access an element in a list.

And you can remove a key-value pair like this:

del my_dict["apple"]

13. What is the difference between sort() and sorted() in Python ?

sort() and sorted() are both used to sort elements in a list in Python. The main difference between them is that sort() is a method of a list object and sorts the list in place, while sorted() is a built-in function that creates a new sorted list from an iterable.

Here's an example:

```
my_list = [3, 1, 4, 2]
my_list.sort()
print(my_list)
```

This code uses the sort() method to sort the elements of my_list in ascending order. The output would be [1, 2, 3, 4].

```
my_list = [3, 1, 4, 2]
new_list = sorted(my_list)
```

14. What is a slice in Python?

In Python, a slice is a way to extract a portion of a sequence such as a string or list. You can specify the start and end indices as well as the step size to create a new sequence that contains the desired elements.

List_name[start : stop : step]

15. What happens if the start index is greater than the stop index in a slice ?

If the start index is greater than the stop index in a slice, the resulting slice will be an empty sequence of the same type as the original sequence.

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

# Slice with start index greater than stop index
my_slice = my_list[3:1]

print(my_slice) # Output: []
```

If we slice the list with a start index of 3, a stop index of 1, and a step value of -1 then it won't be an empty list.

16. Can you use slices to modify a sequence in-place ?

Yes, you can use slices to modify a sequence in-place in Python. When you modify a slice of a sequence, the changes are reflected in the original sequence as well.

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

# Modify slice of the list in-place
my_list[2:4] = [6, 7, 8]

print(my_list) # Output: [1, 2, 6, 7, 8, 5]
```

This behavior is consistent across all mutable sequence types in Python, including lists and byte arrays.

Note that when you use a slice to modify a sequence in-place, the length of the sequence may change, depending on the length of the replacement sequence. If the replacement sequence is shorter than the slice, the original sequence will shrink. If the replacement sequence is longer than the slice, the original sequence will grow.

17. **Can you use slices to reverse a sequence in Python?**

    Yes! We can use slicing to reverse a sequence in Python.

    list_name[::-1]

18. **What happens if you use a step of zero in a slice?**

    If you use a step of zero in a slice, you will get a ValueError with the message "slice step cannot be zero".

19. **Can you use slices with strings in Python?**

    Yes, you can use slices with strings in Python.
    Strings are sequence types in Python, which means that they support slicing operations. You can use slicing to extract a portion of a string, or to create a new string that is a subset of the original.

20. **Can you use slices with sets or dictionaries in Python?**
    Slicing is an operation that is only supported by sequence types in Python, such as lists, tuples, and strings. Sets and dictionaries are not sequences, but rather mappings or collections of key-value pairs. As such, they do not support slicing operations.