Python List Data Type:

Definition: A list in Python is an ordered, mutable collection that can store elements of different data types (integers, strings, floats, etc.).

Lists are defined using square brackets [ ] and elements are separated by commas.

```
my_list = [1, 2, 3, "apple", True]
```

Indexing: Lists are indexed, meaning elements can be accessed using their position (starting from 0).

```
my_list[0]  # returns 1
my_list[3]  # returns 'apple'
```

Slicing: You can retrieve a *subset* of a list using slicing.

```
my_list[1:3]  # returns [2, 3]
my_list[:3]   # returns [1, 2, 3]
my_list[-1]   # returns True (last element)
```

Mutability: Lists are mutable, meaning you can change their contents after creation.

```
my_list[2] = "orange"  # changes third element to 'orange'
```

Working with Lists:

Adding Elements:

append(): Adds an element to the end of the list.

```
my_list.append("banana")
# my_list becomes [1, 2, "orange", "apple", True, "banana"]
```

insert(): Inserts an element at a specified index.

```
my_list.insert(1, "grape")
# my_list becomes [1, "grape", 2, "orange", "apple", True, "banana"]
```

extend(): Extends the list by appending all elements from an iterable (like another list).

```
my_list.extend([7, 8])
# my_list becomes [1, "grape", 2, "orange", "apple", True, "banana", 7, 8]
```

Removing Elements:

remove(): Removes the first instance of a specified element.

```
my_list.remove("apple")
# my_list becomes [1, "grape", 2, "orange", True, "banana", 7, 8]
```

pop(): Removes and returns the element at a specified index (default is the last item).

```
my_list.pop(0)  # removes and returns 1
# my_list becomes ["grape", 2, "orange", True, "banana", 7, 8]
```

clear(): Removes all elements from the list.

```
my_list.clear()  # my_list becomes []
```

Other List Operations:

len(): Returns the length of the list.

```
len(my_list)  # returns 7
```

sort(): Sorts the list in ascending order (in-place).

```
my_list.sort()
```

reverse(): Reverses the order of elements in the list.

```
my_list.reverse()
```

index(): Returns the index of the first occurrence of a specified value.

my_list.index("banana") # returns 4

count(): Returns the number of occurrences of a value in the list.

my_list.count(7) # returns 1

Augmented Assignment Operators with Lists:

Augmented assignment operators are shorthand operators that perform an operation and assignment in one step. They modify the list in place.

+=: Extends the list with another list **(concatenation).**

my_list += [9, 10]

my_list becomes ["grape", 2, "orange", True, "banana", 7, 8, 9, 10]

*****=: Repeats the elements of the list a specified number of times.

my_list *= 2

my_list becomes ["grape", 2, "orange", True, "banana", 7, 8, 9, 10, "grape", 2, "orange", True, "banana", 7, 8, 9, 10]

List Methods Recap:

Adding: append(), insert(), extend()

Removing: remove(), pop(), clear()

Other: sort(), reverse(), index(), count()

Lists are a versatile data structure in Python with multiple operations that help in manipulating data.

Python Dictionary Data Type:

Definition: A dictionary in Python is an unordered, mutable collection of key-value pairs. Each key in a dictionary must be unique and immutable (like strings, numbers, or tuples), while values can be of any data type (including lists or other dictionaries). Dictionaries are defined using curly braces {}.

```
my_dict = {"name": "John", "age": 25, "city": "New York"}
```

Key-Value Pairing: Each element in a dictionary consists of a key and a corresponding value. You can access values by referencing their keys.

```
my_dict["name"]  # returns 'John'
my_dict["age"]   # returns 25
```

Mutability: Dictionaries are mutable, meaning you can change, add, or remove key-value pairs after creation.

```
my_dict["age"] = 26  # updates the value of the 'age' key
my_dict["country"] = "USA"  # adds a new key-value pair
```

Dictionary Operations:

Accessing Data:

Using []:

```
value = my_dict["name"]  # returns 'John'
```

Using get() (returns None if the key doesn't exist, instead of raising an error):

```
value = my_dict.get("address")  # returns None
```

Adding or Updating Elements:

my_dict["job"] = "Engineer" # adds a new key-value pair

my_dict["age"] = 27 # updates the value for 'age'

Removing Elements:

pop(): Removes a key-value pair by key and returns the value.

my_dict.pop("city") # removes 'city' and returns 'New York'

del: Deletes a key-value pair.

del my_dict["job"]

clear(): Removes all key-value pairs.

my_dict.clear() # my_dict becomes {}

Dictionary Methods:

keys(): Returns a view object of the dictionary's keys.

my_dict.keys() # returns dict_keys(['name', 'age'])

values(): Returns a view object of the dictionary's values.

my_dict.values() # returns dict_values(['John', 27])

items(): Returns a view object of the dictionary's key-value pairs.

my_dict.items() # returns dict_items([('name', 'John'), ('age', 27)])

update(): Updates the dictionary with key-value pairs from another dictionary or iterable.

my_dict.update({"email": "john@example.com", "age": 28}) # updates 'age' and adds 'email'

Pretty Printing in Python:

Sometimes dictionaries (or other data structures) contain nested structures or a lot of information that can be hard to read. Pretty printing is a way of displaying data in a more readable format.

Using the pprint module:

```
import pprint

my_dict = {
    "name": "John",
    "age": 28,
    "job": "Engineer",
    "skills": ["Python", "C++", "Machine Learning"],
    "address": {
        "city": "New York",
        "zipcode": 10001,
        "country": "USA"
    }
}

pprint.pprint(my_dict)
```

Output:

```
{'address': {'city': 'New York', 'country': 'USA', 'zipcode': 10001},
 'age': 28,
 'job': 'Engineer',
 'name': 'John',
```

'skills': ['Python', 'C++', 'Machine Learning']}

Customizing Pretty Printing: You can control parameters such as indentation and width.

Ex: pprint.pprint(my_dict, indent=4, width=40

Example-2

```
import pprint
message = 'It was a bright cold day in April, and the clocks were striking thirteen.'
count = {}
for character in message:
count.setdefault(character, 0)
count[character] = count[character] + 1
pprint.pprint(count)
```

output

```
{' ': 13,
',': 1,
'.': 1,
'A': 1,
'I': 1,
'a': 4,
'b': 1,
'c': 3,
'd': 3,
'e': 5,
'g': 2,
'h': 3,
'i': 6,

'k': 2,
'l': 3,
'n': 4,
'o': 2,
'p': 1,
'r': 5,
's': 3,
't': 6,
'w': 2,
'y': 1}
```