Introduction to Python Data Structures

Python data structures are fundamental building blocks for organizing and manipulating data. They provide efficient ways to store, access, and manage information within your programs.

MH

by Md. Mosarraf Hossen

List: Definition and Example

Lists are used to store multiple items in a single variable.

1 Definition

A list is a sequence of elements, ordered, mutable and Allowed duplicate. Elements can be of different types, including numbers, strings, or even other lists.

2 Example

In Python, lists are defined using square brackets `[]`. For instance, `list1 = ["abc", 34, True, 40, "male“, 3.14]` creates a list with an integer, a string, Boolean, and a float.

3 Flexibility

You can add, remove, or modify elements within a list dynamically, making it a versatile data structure.

4 Visualization

Imagine a shopping list with items arranged in order, like a grocery store aisle, where you can add or remove items as needed.

Tuple: Definition and Example

Ordered Sequence

A tuple is a collection of elements, similar to a list, but immutable, meaning the elements cannot be changed after creation.

Immutable Nature

Once a tuple is created, its elements remain fixed, providing data integrity and ensuring that values are not accidentally altered.

Efficiency

Tuples are generally more efficient than lists for storing and accessing data due to their fixed nature, making them suitable for situations where data should not be modified.

Set: Definition and Example

Unordered Collection

A set is an unordered collection of unique elements. It doesn't allow duplicates. Sets are mutable and can be modified after creation.

Membership Test

Sets are efficient for checking if an element exists within the collection, using the `in` operator.

Mathematical Operations

Sets support operations like union, intersection, and difference, making them useful for data analysis and logical operations.

Dictionary: Definition and Example

Key-Value Pairs

A dictionary is a collection of key-value pairs, where each key is unique and maps to a corresponding value.

Mutable and Ordered

Dictionaries are mutable, allowing you to add, remove, or modify key-value pairs. Python 3.7 and later versions maintain insertion order.

Example

You can define a dictionary in Python using curly braces `{}`. For example, `my_dict = {"name": "Alice", "age": 30}` stores a person's name and age.

Efficient Lookup

Dictionaries are optimized for retrieving values based on their associated keys, making them ideal for storing and accessing data efficiently.

Accessing and Manipulating Lists

1

Indexing

Access elements by position, starting from 0. For Negative indexing means start from the end

2

Slicing

Extract sub-sequences using start and end indices.

3

Methods

Use built-in functions like append, insert, and remove.

`append()`

`insert()`

`remove()`

4

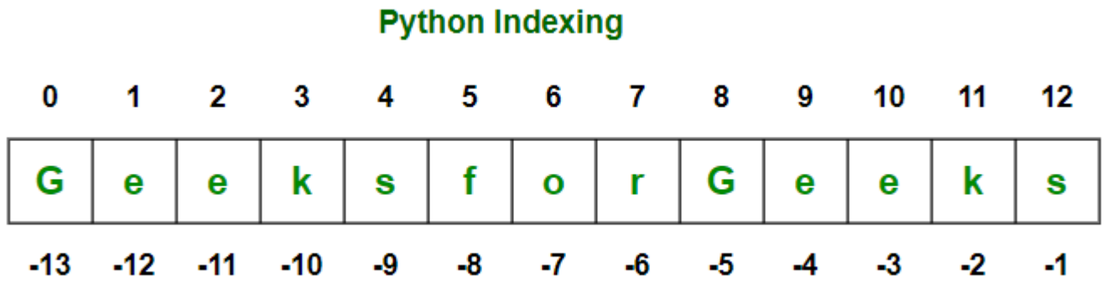
Iteration

Loop through elements using `for` loop.

sequence[Start : End : Step]

Parameters:

- **Start:** It is the starting point of the slice or substring.
- **End:** It is the ending point of the slice or substring but it does not include the last index.
- **Step:** It is number of steps it takes.




```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

Source: https://www.geeksforgeeks.org/

Operation in List with example

```
thislist = ["Mango", " Dates", " Blueberry"]
print(len(thislist))
```

List[start:stop:step]

```
thislist = list((“Mango", " Dates", " Orange"))
print(thislist)
```

```
thislist[1] = "blackcurrant“ # Change an item
```

```
thislist = [“Mango", “Dates", “Blueberry“, “Orange”]
print(thislist[1])
```

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
```

```
thislist.insert(2, "watermelon")
```

```
thislist = [“Watermelon", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

myListOpn.py

Source: https://www.w3schools.com/

Operation in List with example

Write a program that accepts a list from user and print the alternate element of list.

```
mylist = []
size = int(input('How many elements you want to enter? '))
print('Enter',str(size),'elements') for i in range(size):
data = int(input()) mylist.append(data)
print('Alternate elements are:')
for i in range(0,size,2):
        print(mylist[i])
```

myListOpn.py

Source: https://www.w3schools.com/

Accessing and Manipulating Tuples

1

Indexing

You can access individual elements in a tuple using their position (index), starting from 0. The index is enclosed in square brackets after the tuple name.

2

Slicing

To extract a subsequence from a tuple, use slicing. Specify the starting and ending indices, separated by a colon, within square brackets.

3

Immutability

Tuples are immutable, meaning you cannot directly modify elements once the tuple is created. You must create a new tuple with the desired changes.

Operation in Tuple with example

```
thistuple =
("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

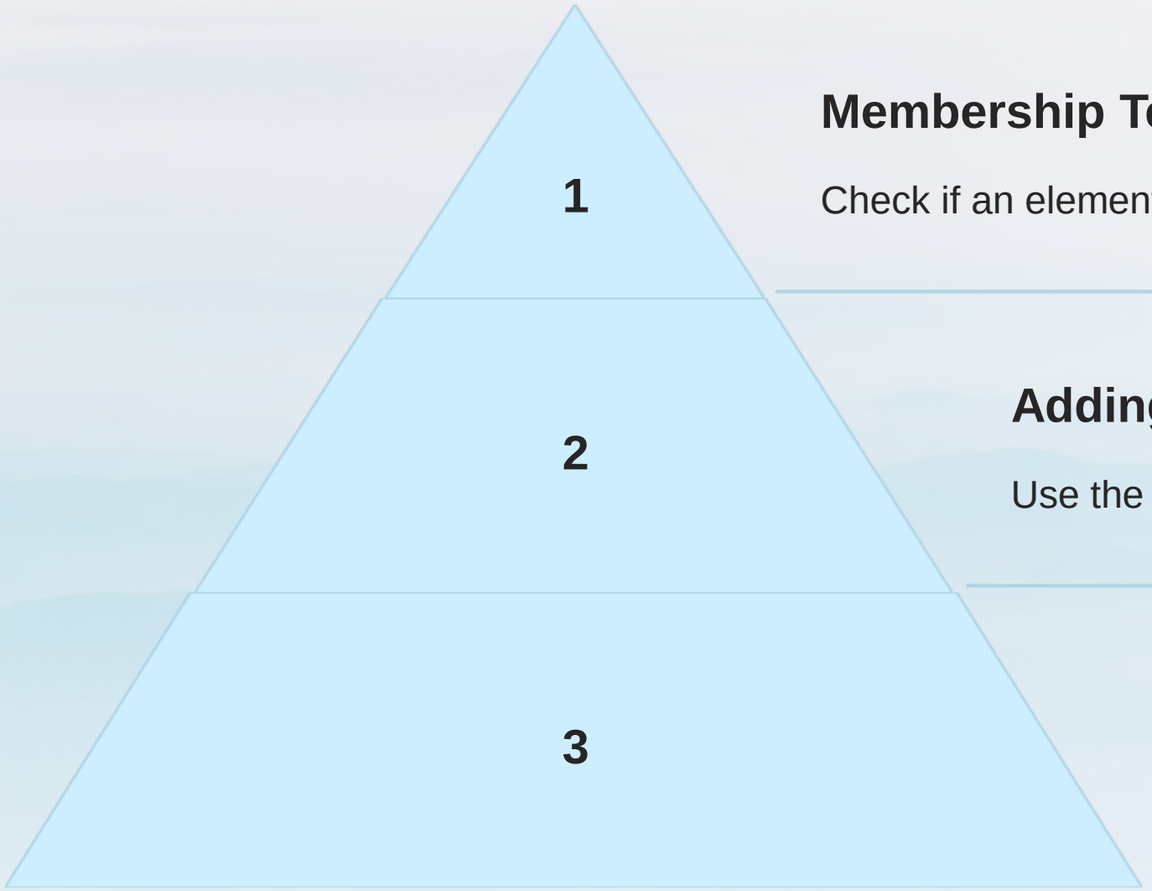
```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])
```

Source: https://www.w3schools.com/

Accessing and Manipulating Sets



Membership Test

Check if an element exists in the set using the `in` operator.

Adding Elements

Use the `add()` method to insert a new element into the set.

Removing Elements

Use the `remove()` or `discard()` methods to remove an element from the set.

Sets are mutable, meaning they can be modified after creation. You can add or remove elements from a set using specific methods.

Operation in Set with example

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

```
thisset = {"apple", "banana", "cherry", False, True, 0}

print(thisset)
```

```
  thisset = {"apple", "banana", "cherry"}

  for x in thisset:
    print(x)
```

Source: https://www.w3schools.com/

Accessing and Manipulating Dictionaries

1

Key-Based Access

Retrieve values using their corresponding keys, enclosed in square brackets.

2

Modifying Values

Change the value associated with a key by assigning a new value to it.

3

Adding Entries

Insert a new key-value pair into the dictionary.

4

Deleting Entries

Remove a specific key-value pair using the `del` keyword.

Dictionaries are written with curly brackets, and have keys and values

Operation in Dictionary with example

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

```
print("Loop in dictionary")
for x in newDict:
   print(x)
```

```
print("all values in a dictionary using loop")
for x in myDict:
   print(myDict[x])

print("values()")
for x in myDict.values():
   print(x)
print("keys()")
for x in myDict.keys():
   print(x)
print("Both key and values")
for x, y in myDict.items():
   print(x, y)
```

```
myDict={
   "brand": "Fold",
   "Country": "AUS",
   "Model": "Fold40",
   "Manufacture": "1964",
   "Color": ["red", "green", "blue"]
}
print(myDict)
print("Length: ", len(myDict))
print(myDict["Country"])
print(myDict["Manufacture"])
print(myDict["Color"])
print(type(myDict))
```

myDictOpn.py

Operation in Nested Dictionary with example..

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

```
Accessing Nested Dictionary Item
print(myfamily["child2"]["name"])
```

Source: https://www.w3schools.com/

Comparison and Use Cases of Data Structures

List

A list is ordered and mutable, allowing elements to be added, removed, or modified. Lists are ideal for storing sequences of data, such as lists of items, instructions, or steps in a process.

Tuple

A tuple is an immutable sequence, designed for storing data that should not be changed. Tuples are often used for representing fixed data, like coordinates, database records, or function return values.

Set

A set is an unordered collection of unique elements, useful for membership testing and mathematical operations like union, intersection, and difference. They are commonly used for removing duplicates from a list or for performing set-based logic.

Dictionary

A dictionary is a key-value store, allowing efficient data retrieval based on a unique key. Dictionaries are widely used for storing configurations, lookups, and mapping data, where each key represents a distinct item and its value provides associated information.

“Skill is only developed by hours and hours of work.”
– Lewis Hamilton

Thank you