Data Structures

Data structures are basically just that - they are structures which can hold some data together. In other words, they are used to store a collection of related data. There are four built-in data structures in Python - list, tuple, dictionary and set. We will see how to use each of them and how they make life easier for us.

Tuples

Python tuple is a sequence, which can store hetrogeneous data types such as integers, float, strings, lists and dictionary. Like strings, tuple is immutable.

In [ ]:

```
# Create an empty tuple with parentheses:

t = ()
type(t)
```

Out[ ]:

```
tuple
```

In [ ]:

```
# Note that a single value in parentheses is not a tuple:

t1 = ('s')
type(t1)
```

Out[ ]:

```
str
```

In [ ]:

```
# To create a singleton tuple it is necessary to have a trailing comma.

t2 = ('a',)
type(t2)
```

Out[ ]:

```
tuple
```

Creating tuple with elements

To create a tuple, fill the values in tuple separated by commas:

In [ ]:

```
tup = (2,3,4, "hello", 'python')
```

In [ ]:

```
tup
```

Out[ ]:

```
(2, 3, 4, 'hello', 'python')
```

Indexing tuple

In order to access a particular value of tuple, specify a position number, in brackets. Let's discuss with an example

example.

In [ ]:

```
tup[3]
```

Out[ ]:

```
'hello'
```

In [ ]:

```
tup[-1]
```

Out[ ]:

```
'python'
```

Slicing of tuple

In order to do slicing, use the square brackets with the index or indices.

In [ ]:

```
tup[1:3]
```

Out[ ]:

```
(3, 4)
```

In [ ]:

```
tup[1:6:2]
```

Out[ ]:

```
(3, 'hello')
```

In [ ]:

```
tup[1:]
```

Out[ ]:

```
(3, 4, 'hello', 'python')
```

In [ ]:

```
tup[::-1]
```

Out[ ]:

```
('python', 'hello', 4, 3, 2)
```

In [ ]:

```
# Tuples are immutable, that is, one cannot add or modify items once the tuple is initial
ized.

tup1 = (1, 4, 9, 'ITSkills', 'Solution')
tup1[3] = 'Python'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-c45aff0390bf> in <module>
      2
      3 tup1 = (1, 4, 9, 'ITSkills', 'Solution')
----> 4 tup1[3] = 'Python'

TypeError: 'tuple' object does not support item assignment
```

Unpacking the items of tuples

In [ ]:

```
tup2 = (1,2,3)
a,b,c = tup2
```

In [ ]:

```
a, b, c
```

Out[ ]:

```
(1, 2, 3)
```

Built-in Tuple Functions

In [ ]:

```
# The function len returns the total length of the tuple

tuple1 = ("C", "Python", "java","html", 'C++')
len(tuple1)
```

Out[ ]:

```
5
```

In [ ]:

```
# The function max returns item from the tuple with the max value

max(tuple1)
```

Out[ ]:

```
'java'
```

In [ ]:

```
# The function min returns the item from the tuple with the min value

min(tuple1)
```

Out[ ]:

```
'C'
```

In [ ]:

```
# count function

a = (2,4,'shivam',3,2,4,2,6,9)
a.count(2)
```

Out[ ]:

```
3
```

In [ ]:

```
# index function

a.index('shivam')
```

Out[ ]:

```
2
```

In [ ]:

```
# By using the + operator, two tuples can be added
```

```
tuple2 = ('1','2','3')
tuple3 = ('a', 'b', 'c', 'd', 'e')

tuple2 + tuple3
```

Out[ ]:

```
('1', '2', '3', 'a', 'b', 'c', 'd', 'e')
```

In [ ]:

```
# By using the * operator, you can perform multiplication

tuple2*2
```

Out[ ]:

```
('1', '2', '3', '1', '2', '3')
```

List

A list is a data structure in Python that is a mutable, or changeable, ordered sequence of elements. Each element or value that is inside of a list is called an item. Just as strings are defined as characters between quotes, lists are defined by having values between square brackets [ ].

In [ ]:

```
a = [1, 2, 3, 4, 5, 'Python', 'Tutorial']
a
```

Out[ ]:

```
[1, 2, 3, 4, 5, 'Python', 'Tutorial']
```

In [ ]:

```
type(a)
```

Out[ ]:

```
list
```

Accessing list values

In order to access list values, use list names with positional index in square brackets.

In [ ]:

```
a[4]
```

Out[ ]:

```
5
```

In [ ]:

```
a[6]
```

Out[ ]:

```
'Tutorial'
```

In [ ]:

```
a[-2]
```

Out[ ]:

```
'Python'
```

In [ ]:

```
a[7]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-46-9cf13ba20553> in <module>
----> 1 a[7]

IndexError: list index out of range
```

If the desired index is not found in the list, then the interpreter throws IndexError.

Slicing of List

The slicing of a list is the same as we did in tuples.

In [ ]:

```
a[1:5]
```

Out[ ]:

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

In [ ]:

```
a[:6]
```

Out[ ]:

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

In [ ]:

```
a[:]
```

Out[ ]:

```
[1, 2, 3, 4, 5, 'Python', 'Tutorial']
```

In [ ]:

```
a[1:7:2]
```

Out[ ]:

```
[2, 4, 'Python']
```

The step means the amount by which the index increases. If you don't define it, then it takes 1 step by default.

Updating the list

Lists are mutable, so the values of a list can be updated.

In [ ]:

```
l = ['Learn', 'Python', 'in', 8, 'months']
```

In [ ]:

```
l[3] = 7
```

In [ ]:

```
l
```

Out[ ]:

```
['Learn', 'Python', 'in', 7, 'months']
```

In [ ]:

```
l[4] = 'days'
```

In [ ]:

```
l
```

Out[ ]:

```
['Learn', 'Python', 'in', 7, 'days']
```

List functions

In [ ]:

```
# The len() function returns the number of elements or values in the list

len(l)
```

Out[ ]:

```
5
```

In [ ]:

```
# The max() function returns the element of the list with the maximum value

l1 = [2,5,1,6,3,9,7]
max(l1)
```

Out[ ]:

```
9
```

In [ ]:

```
# The min() function returns the element of the list with the maximum value

min(l1)
```

Out[ ]:

```
1
```

In [ ]:

```
# The sorted () function returns a new sorted list from the values in iterable.

sorted(l1)
```

Out[ ]:

```
[1, 2, 3, 5, 6, 7, 9]
```

List methods

1. **append(value) – appends a new element to the end of the list.**

In [ ]:

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

# Append values 6, 7, and 7 to the list
a.append(6)
a.append(7)
a.append(7)
```

```
a
```

Out[ ]:

```
[1, 2, 3, 4, 5, 6, 7, 7]
```

In [ ]:

```
# Append another list

b = [8, 9]
a.append(b)
a
```

Out[ ]:

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

In [ ]:

```
# Append an element of a different type, as list elements do not need to have the same ty
pe

my_string = "Hello Python"
a.append(my_string)
a
```

Out[ ]:

```
[1, 2, 3, 4, 5, 6, 7, 7, [8, 9], 'Hello Python']
```

Note that the append() method only appends one new element to the end of the list. If you append a list to another list, the list that you append becomes a single element at the end of the first list.

1. **extend(enumerable) – extends the list by appending elements from another enumerable.**

In [ ]:

```
a = [1, 2, 3, 4, 5, 6, 7, 7]
b = [8, 9, 10]

# Extend list by appending all elements from b
a.extend(b)
```

In [ ]:

```
a
```

Out[ ]:

```
[1, 2, 3, 4, 5, 6, 7, 7, 8, 9, 10]
```

In [ ]:

```
# Extend list with elements from a non-list enumerable:

a.extend(range(3))
a
```

Out[ ]:

```
[1, 2, 3, 4, 5, 6, 7, 7, 8, 9, 10, 0, 1, 2]
```

In [ ]:

```
# Lists can also be concatenated with the + operator. Note that this does not modify any
of the original lists:
a = [1, 2, 3, 4, 5, 6] + [7, 7] + b
a
```

Out[ ]:

```
[1, 2, 3, 4, 5, 6, 7, 7, 8, 9, 10]
```

Difference between append and extend.

If you are confused between the append and extend methods, the following example will clear your doubts:

In [ ]:

```
Linux = ["kali", "Ubuntu", "debian"]
Linux2 = ["RHEL", "Centos"]

Linux.extend(Linux2)
Linux
```

Out[ ]:

```
['kali', 'Ubuntu', 'debian', 'RHEL', 'Centos']
```

In [ ]:

```
Linux = ["kali", "Ubuntu", "debian"]
Linux2 = ["RHEL", "Centos"]

Linux.append(Linux2)
Linux
```

Out[ ]:

```
['kali', 'Ubuntu', 'debian', ['RHEL', 'Centos']]
```

The append method gives a list within the list. The list Linux2 = ["RHEL", "Centos"]has been taken as one list.

1. **index(value, [startIndex]) – gets the index of the first occurrence of the input value. If the input value is not in the list a ValueError exception is raised. If a second argument is provided, the search is started at that specified index.**

In [ ]:

```
a.index(7)
```

Out[ ]:

```
6
```

In [ ]:

```
a.index(7, 7)
```

Out[ ]:

```
7
```

1. **insert(index, value) – inserts value just before the specified index. Thus after the insertion the new element occupies position index.**

In [ ]:

```
a.insert(0, 0) # insert 0 at position 0
a.insert(2, 5) # insert 5 at position 2
a
```

Out[ ]:

```
[0, 1, 5, 2, 3, 4, 5, 6, 7, 7, 8, 9, 10]
```

1. **pop([index]) – removes and returns the item at index. With no argument it removes and returns the last element of the list.**

In [ ]:

```
a.pop(2)
```

Out[ ]:

```
5
```

In [ ]:

```
a
```

Out[ ]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 7, 8, 9, 10]
```

In [ ]:

```
# With no argument:
a.pop()
```

Out[ ]:

```
10
```

In [ ]:

```
a
```

Out[ ]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 7, 8, 9]
```

1. **remove(value) – removes the first occurrence of the specified value. If the provided value cannot be found, a ValueError is raised.**

In [ ]:

```
a.remove(0)
a.remove(9)
a
```

Out[ ]:

```
[1, 2, 3, 4, 5, 6, 7, 7, 8]
```

1. **reverse() – reverses the list in-place and returns None**

In [ ]:

```
a.reverse()
```

In [ ]:

```
a
```

Out[ ]:

```
[8, 7, 7, 6, 5, 4, 3, 2, 1]
```

1. **count(value) – counts the number of occurrences of some value in the list.**

In [ ]:

```
a.count(7)
```

Out[ ]:

```
2
```

In [ ]:

```
a="ritik"
```

In [ ]:

```
a.reverse()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-2-d48a15021150> in <module>()
----> 1 a.reverse()

AttributeError: 'str' object has no attribute 'reverse'
```

1. **sort() – sorts the list in numerical and lexicographical order and returns None.**

In [ ]:

```
c = [7,2,9,3,4,5,10,8]
c.sort()
```

In [ ]:

```
c
```

Out[ ]:

```
[2, 3, 4, 5, 7, 8, 9, 10]
```

In [ ]:

```
# Lists can also be reversed when sorted using the reverse=True flag in the sort() method
.

c.sort(reverse=True)
c
```

Out[ ]:

```
[10, 9, 8, 7, 5, 4, 3, 2]
```

1. **clear() – removes all items from the list**

In [ ]:

```
c.clear()
c
```

Out[ ]:

```
[]
```

1. **Replication – multiplying an existing list by an integer will produce a larger list consisting of that many copies of the original. This can be useful for example for list initialization:**

In [ ]:

```
p = ['python', 'class']*3

p
```

Out[ ]:

```
['python', 'class', 'python', 'class', 'python', 'class']
```

In [ ]:

```
d = [1, 3, 5] * 5
```

```
d
```

Out[ ]:

```
[1, 3, 5, 1, 3, 5, 1, 3, 5, 1, 3, 5, 1, 3, 5]
```

1. **Element deletion – it is possible to delete multiple elements in the list using the del keyword and slice notation:**

In [ ]:

```
a
```

Out[ ]:

```
[8, 7, 7, 6, 5, 4, 3, 2, 1]
```

In [ ]:

```
del a[2]
```

In [ ]:

```
a
```

Out[ ]:

```
[8, 7, 6, 5, 4, 3, 2, 1]
```

In [ ]:

```
del a[1:4]
```

In [ ]:

```
a
```

Out[ ]:

```
[8, 4, 3, 2, 1]
```

Accessing values in nested list

In [ ]:

```
alist = [[[1,2],[3,4]], [[5,6,7],[8,9,10], [12, 13, 14]]]
```

In [ ]:

```
# Accesses second element in the first list in the first list
alist[0][0][1]
```

Out[ ]:

```
2
```

In [ ]:

```
# #Accesses the third element in the second list in the second list
alist[1][1][2]
```

Out[ ]:

```
10
```

Dictionary

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values

curly brackets, and they have keys and values.

Creating a Dictionary

In [ ]:

```
d = {} # empty dict
d = {'key': 'value'} # dict with initial values
d
```

Out[ ]:

```
{'key': 'value'}
```

In [ ]:

```
d = {'name': 'shivam', 'score': 77, 'rating': 4.5}
d
```

Out[ ]:

```
{'name': 'shivam', 'rating': 4.5, 'score': 77}
```

Accessing the values of dictionary

In [ ]:

```
d['score']
```

Out[ ]:

```
77
```

In [ ]:

```
d['rating']
```

Out[ ]:

```
4.5
```

In [ ]:

```
d['rate']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-7-5ae4a336d849> in <module>()
----> 1 d['rate']

KeyError: 'rate'
```

If the key is not found, then the interpreter shows the preceding error.

Deleting an item from the dictionary

In [ ]:

```
del d['score']
```

In [ ]:

```
d
```

Updating the values of the dictionary

Updating the dictionary is pretty simple: just specify the key in the square bracket along with the dictionary

Updating the dictionary is pretty simple, just specify the key in the square bracket along with the dictionary name.

In [ ]:

```
info = {'course': 'Python', 'mode': 'online', 'rating': 4.5}
```

In [ ]:

```
info['rating'] = 5
info
```

Adding an item to the dictionary

Adding an item to the dictionary is very simple; just specify a new key in the square brackets along with the dictionary.

In [ ]:

```
info = {'course': 'Python', 'mode': 'online', 'rating': 4.5}
```

In [ ]:

```
info['place'] = 'Varanasi'
```

In [ ]:

```
info
```

Dictionary functions

In [ ]:

```
# In order to find the number of items that are present in a dictionary, you can use the
len() function

d = {'name': 'shivam', 'score': 77, 'rating': 4.5}
len(d)
```

In [ ]:

```
# Consider a situation where you want to convert a dictionary into a string; here you can
use the str() function.

str(d)
```

In [ ]:

```
# copy() method

d1 = {'sr': 56, 'tt':78, 'yu':89}
d2 = d1.copy()
d2
```

In [ ]:

```
# The get() method is used to get the value of a given key from the dictionary.

d2.get('sr')
```

In [ ]:

```
# Consider a situation where you want to do some operation on a dictionary's keys and wan
t to get all the keys in different lists. In this situation, you can use the keys() metho
d.

d2.keys()
```

In [ ]:

```
# Similarly, if we want all the values in a separate list, we can use the values() method
.

d2.values()
```

In [ ]:

```
# update() method

port1 = {22: "SSH", 23: "telnet", 80: "Http" }
port2 = {53 :"DNS", 443 : "https"}

port1.update(port2)
port1
```

In [ ]:

```
# The items() method returns the list of dictionary's (key, value) tuple pairs:

dict1 = d={1:'one',2:'two',3:'three'}
dict1.items()
```

In [ ]:

```
# clear() method

dict1.clear()
dict1
```

Sets

A Set is an unordered collection data type that is iterable, mutable, and has no duplicate elements. Python's set class represents the mathematical notion of a set.This is based on a data structure known as a hash table.

In [ ]:

```
d = {5,6}
type(d)
```

Out[ ]:

```
set
```

In [ ]:

```
s = {1,4,4,4,2,5,2,7,8}
type(s)
```

Out[ ]:

```
set
```

In [ ]:

```
s.add(9) #add method
```

In [ ]:

```
s
```

Out[ ]:

```
{1, 2, 4, 5, 7, 8, 9}
```

In [ ]:

```
# difference method
```

```
s1 = {'roh', 'io', 'py', 'tw'}
s2 = {'io', 'py', 'te'}
```

In [ ]:

```
s1.difference(s2)
```

Out[ ]:

```
{'roh', 'tw'}
```

In [ ]:

```
s1
```

Out[ ]:

```
{'io', 'py', 'roh', 'tw'}
```

In [ ]:

```
s1.difference_update(s2) #for permanent change
```

In [ ]:

```
s1
```

Out[ ]:

```
{'roh', 'tw'}
```

Accessing elements in sets

In [ ]:

```
c1 = {'audi': 1970}
c2 = {'marceddes': 1960}

ct = {'cm': c1, 'cm2': c2}
```

In [ ]:

```
c2['marceddes']
```

Out[ ]:

```
1960
```

In [ ]:

```
ct
```

Out[ ]:

```
{'cm': {'audi': 1970}, 'cm2': {'marceddes': 1960}}
```

In [ ]:

```
ct['cm2']['marceddes']
```

Out[ ]:

```
1960
```

In [ ]:

```

```