Built-in Data Structures

#1. Lists

In [1]:
```python
values = [1,2,3,4,5]
print(values)
```
```
[1, 2, 3, 4, 5]
```

1.1. Indices

In [2]:
```python
values = [23,7,18,0.23,91]
print(values[0])
```
```
23
```

In [3]:
```python
print(values[2])
```
```
18
```

In [4]:
```python
print(values[3])
```
```
0.23
```

In [5]:
```python
values[3] = 0.7
```

In [6]:
```python
print(values)
```
```
[23, 7, 18, 0.7, 91]
```

In [7]:
```python
values[0] = values[0]+values[2]
print(values)
```
```
[41, 7, 18, 0.7, 91]
```

_

In [8]:
```python
print(values[-1])
```
```
91
```

In [9]:
```python
print(values[-4])
```
```
7
```

1.2. Values in Lists

In [10]:
```python
x = 35
k = 19
y = 5
values = [x,k,y]
print(values)
```
```
[35, 19, 5]
```

In [11]:
```python
x = 1
print(values)
```
```
[35, 19, 5]
```

In [12]:
```python
values = [2,'value', 3.2]
print(values)
```
```
[2, 'value', 3.2]
```

In [13]:
```python
more_values=[5,'test',values]
print(more_values)
```
```
[5, 'test', [2, 'value', 3.2]]
```

In [14]:
```python
values[2] = 7
print(more_values)
```
```
[5, 'test', [2, 'value', 7]]
```

In [15]:
```python
values = [1,2]
print(more_values)
```
```
[5, 'test', [2, 'value', 7]]
```

1.3. Testing List Contents

In [16]:
```python
values = [1, 'abc',2,3]
print(1 in values)
```
```
True
```

In [17]:
```python
print(20 in values)
```
```
False
```

In [18]:
```python
print(20 not in values)
```
```
True
```

In [19]:
```python
# Application
passwords = ['pass1','pass2','pass4']
user =input('pleas enter your password: ')
if user in passwords:
    print('Welcome!')
else:
    print('Try again')
```
```
pleas enter your password: pass1
Welcome!
```

1.4.Slicing

In [20]:
```python
L = [1,2,3,4,5]
print(L[1:3])
```
```
[2, 3]
```

In [21]:
```python
print(L[3])
```
```
4
```

In [22]:
```python
L[:3]
```
Out[22]:
```
[1, 2, 3]
```

In [23]:
```python
L[1:]
```
Out[23]:
```
[2, 3, 4, 5]
```

In [24]:
```python
L[:]
```
Out[24]:
```
[1, 2, 3, 4, 5]
```

In [25]:
```python
#Using slicing allows us to add or delete elements
```

In [26]:
```python
L[2:4] = [100,101,102,103]
L
```
Out[26]:
```
[1, 2, 100, 101, 102, 103, 5]
```

In [27]:
```python
L[3:6]=[]
L
```
Out[27]:
```
[1, 2, 100, 5]
```

In [28]:
```python
values = [1,2,3,4,5]
values[::2]
```
Out[28]:
```
[1, 3, 5]
```

In [29]:
```python
values[::-2]
```
Out[29]:
```
[5, 3, 1]
```

In [30]:
```python
values[1:4:2]
```
Out[30]:
```
[2, 4]
```

In [31]:
```python
values[4:1:-1]
```
Out[31]:
```
[5, 4, 3]
```

1.5.List Methods and Functions

In [32]:
```python
#.APPEND()
values = [1,2,3]
values.append(12)
values.append(111)
print(values)
```
```
[1, 2, 3, 12, 111]
```

In [33]:
```python
#.INSERT()
values = [1,2,3]
values.insert(1,13)
print(values)
```
```
[1, 13, 2, 3]
```

In [34]:
```python
#.POP()
values = [1,2,3,4,3456]
print(values.pop())
```
```
3456
```

In [35]:
```python
print(values.pop(0))
```
```
1
```

In [36]:
```python
print(values)
```
```
[2, 3, 4]
```

In [37]:
```python
values = [1,2,3,12]
print(len(values))
```
```
4
```

In [38]:
```python
print(sum(values))
```
```
18
```

In [39]:
```python
values.append('qwe')
print(sum(values))
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-210c0add9c75> in <module>
      1 values.append('qwe')
----> 2 print(sum(values))

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

1.6.Combining Lists

In [41]:
```python
values = [1,2,3]
more_values = [3,2,1]
combined_values = values+more_values
print(combined_values)
```
```
[1, 2, 3, 3, 2, 1]
```

#2.Tuples

In [42]:
```python
food = ('eggs', 'bananas', 'lemons')
food[1]
```
Out[42]:
```
'bananas'
```

In [43]:
```python
food[1:3]
```
Out[43]:
```
('bananas', 'lemons')
```

In [44]:
```python
food[1]  = 'meat'
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-44-ff922d1671d6> in <module>
----> 1 food[1]  = 'meat'

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

In [45]:
```python
food = 'milk','coke'
food[0]
```
Out[45]:
```
'milk'
```

In [46]:
```python
foo = ([1,2,3],[4,5,(6,7,8)],(9,10))
foo
```
Out[46]:
```
([1, 2, 3], [4, 5, (6, 7, 8)], (9, 10))
```

In [47]:
```python
foo = ()
len(foo)
```
Out[47]:
```
0
```

In [48]:
```python
#Unpacking tuples
food = 'milk', 'coke'
benfood, gwenfood = food
# is equivalent to benfood=food[0] and gwenfood=food[1]
benfood
```
Out[48]:
```
'milk'
```

Converting between Sequence Types

In [49]:
```python
food = ('nothing','cereal','lime')
list(food)
```
Out[49]:
```
['nothing', 'cereal', 'lime']
```

In [50]:
```python
str(food)
```
Out[50]:
```
"('nothing', 'cereal', 'lime')"
```

In [51]:
```python
s = "The cat in the hat"
list(s)
```
Out[51]:
```
['T',
 'h',
 'e',
 ' ',
 'c',
 'a',
 't',
 ' ',
 'i',
 'n',
 ' ',
 't',
 'h',
 'e',
 ' ',
 'h',
 'a',
 't']
```

In [52]:
```python
tuple(s)
```
Out[52]:
```
('T',
 'h',
 'e',
 ' ',
 'c',
 'a',
 't',
 ' ',
 'i',
 'n',
 ' ',
 't',
 'h',
 'e',
 ' ',
 'h',
 'a',
 't')
```

for Loops

In [53]:
```python
values = [1,2,3,4,5]
for i in values:
    print(i)
```
```
1
2
3
4
5
```

In [54]:
```python
values = [1,2,3,4,5,6,7,8,9,10]
squares = []
for i in values:
    squares.append(i**2)

print('Initial values: {} '.format(values))
print('Squares: {}'.format(squares))
```
```
Initial values: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Range

In [55]:
```python
print(range(5))
```
```
range(0, 5)
```

In [58]:
```python
print(list(range(5)))
```
```
[0, 1, 2, 3, 4]
```

In [59]:
```python
print(list(range(1,5)))
```
```
[1, 2, 3, 4]
```

In [60]:
```python
print(list(range(1,7,2)))
```
```
[1, 3, 5]
```

In [61]:
```python
values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
squares = []
for i in range(len(values)):
    squares.append(values[i] ** 2)

print("Initial values: {}".format(values))
print("Squares: {}".format(squares))
```
```
Initial values: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In [62]:
```python
values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
adjacent_sums = []

for i in range(len(values)):
    if i != 0:
        adjacent_sums.append(values[i] + values[i - 1])

print("Initial values: {}".format(values))
print("Sums: {}".format(adjacent_sums))
```
```
Initial values: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sums: [3, 5, 7, 9, 11, 13, 15, 17, 19]
```

In [63]:
```python
#Enumerate()
food = ['eggs','burgers','potatoes']
for count, item in enumerate(food):
    print('{} is the {}th item'.format(item,count))
```
```
eggs is the 0th item
burgers is the 1th item
potatoes is the 2th item
```

Exercise 1

Any fraction can be written as the division of two integers. You could express this in Python as a tuple – (numerator, denominator). Write functions for each of the following. They must use the tuple representation to return fractions.

1. Given two fractions as tuples, multiply them.
2. Given two fractions as tuples, divide them.
3. Given a list of fractions as a tuple, return the one that is smallest in value. Also write a small command-line interface such that the user running your script sees something like this:

```
== Multiplication and Division ==
Enter a fraction >>> 5/3
Enter a fraction >>> 10/3
Multiplication of the fractions: 50/9
Division of the first by the second: 5/10

== Smallest fraction ==
Enter a fraction >>> 1/3
Enter a fraction >>> 10/3
Enter a fraction >>> 6/4
Enter a fraction >>> stop
Smallest fraction: 1/3
```

Exercise 2

Take in numbers as input until “stop” is entered. Then split the numbers into three lists: one containing all the numbers, one containing all even values, and one containing all odd. Print out all three lists, as well as each list’s sum and average. Assume all input values are integers. Sample:

```
Input a number >>> 1
Input a number >>> 5
Input a number >>> 8
Input a number >>> 2
Input a number >>> 8
Input a number >>> 100
Input a number >>> 3
Input a number >>> 7
Input a number >>> 27
Input a number >>> 5
Input a number >>> stop
All numbers: [1, 5, 8, 2, 8, 100, 3, 7, 27, 5]
Average of all numbers: 16.6
Sum of all numbers: 166
Even numbers: [8, 2, 8, 100]
Average of even numbers: 29.5
Sum of even numbers: 118
Odd numbers: [1, 5, 3, 7, 27, 5]
Average of odd numbers: 8.0
Sum of odd numbers: 48
```

Exercise 3

Take in numbers as input until “stop” is entered. As you take in each number, insert it into a list so that the list is sorted in ascending order. That is, look through the list until you find the place where the new element belongs, then use .insert() to place it there. If the number isalready in the list, do not add it again. After “stop” is entered, print out the list. Do not use any of Python’s built-in sorting functions. You cannot use .sort() for this exercise. Sample:

```
Input a number >>> 12
Input a number >>> 5.2
Input a number >>> 73
Input a number >>> 45
Input a number >>> 100
Input a number >>> -5
Input a number >>> 2.3
Input a number >>> stop
[-5.0, 2.3, 5.2, 12.0, 45.0, 73.0, 100.0]
```

What happens in the background:

```
Input a number >>> 12
List contains [12.0]
Input a number >>> 5.2
List contains [5.2, 12.0]
Input a number >>> 73
List contains [5.2, 12.0, 73.0]
Input a number >>> 45
List contains [5.2, 12.0, 45.0, 73.0]
Input a number >>> 100
List contains [5.2, 12.0, 45.0, 73.0, 100.0]
Input a number >>> -5
List contains [-5.0, 5.2, 12.0, 45.0, 73.0, 100.0]
Input a number >>> 2.3
List contains [-5.0, 2.3, 5.2, 12.0, 45.0, 73.0, 100.0]
Input a number >>> stop
[-5.0, 2.3, 5.2, 12.0, 45.0, 73.0, 100.0]
```

Option 1.

Task 1. Find the number of positive (>0) items in this list.

Task 2. A list of numbers is given. Print the value of the largest item in the list, and then the index of that item in the list. If there are several largest elements, output the index of the first one.

Task 3. Print the value of the smallest odd element in the list, and if there are no odd elements in the list, print the number 0.

Option 2.

Task 1.A list of numbers is given. Print all the list items that are larger than the previous item.

Task 2. A list of numbers is given. Determine how many elements in this list that are greater than two of their neighbors and output the number of such elements.

Task 3. Print the value of the smallest of all the positive elements in the list. It is known that there is at least one positive element in the list, and the values of all the elements of the list modulo do not exceed 1000.

In [ ]:
```python

```