Python Data Structures

Prepared by Fahad Siddiqui on 26 Feb 2020

Outline

- Python List
- Slicing Operation
- Tuples
- Dictionaries
- Sets
- Looping Through Values
- Nested Lists and Dictionaries
- List Comprehension
- Performance Comparison
- Lambda Expressions
- Built-in functions
- Generators

List

In [15]:

```
list = ["This is string",1,True,2.5]

print("List content: ", list)
print()

print("Type of list : ",type(list))
print()

print("Type of '", list[0] ,"' is " ,type(list[0]))
print()

print("Type of '", list[1] ,"' is " ,type(list[1]))
print()

print("Type of '", list[2] ,"' is " ,type(list[2]))
print()

print("Type of '", list[3] ,"'' is " ,type(list[3]))
```

```
List content:  ['This is string', 1, True, 2.5]

Type of list :  <class 'list'>

Type of ' This is string ' is  <class 'str'>

Type of ' 1 ' is  <class 'int'>

Type of ' True ' is  <class 'bool'>

Type of ' 2.5 '' is  <class 'float'>
```

In [43]:

```
cities = ["Khairpur","Sukkur","Lahore"]
print(cities)
```

```
['Khairpur', 'Sukkur', 'Lahore']
```

In [44]:

```
# Insertion in list

cities.append("Islamabad")

print(cities , end = "\n\n")

# Insert at specific index in a list
cities.insert(0,"Karachi")

print(cities , end = "\n\n")

cities.insert(1,"Hyderabad")

print(cities , end = "\n\n")

cities[2] = "Nawabshah"

print(cities , end = "\n\n")
```

```
['Khairpur', 'Sukkur', 'Lahore', 'Islamabad']

['Karachi', 'Khairpur', 'Sukkur', 'Lahore', 'Islamabad']

['Karachi', 'Hyderabad', 'Khairpur', 'Sukkur', 'Lahore', 'Islamabad']

['Karachi', 'Hyderabad', 'Nawabshah', 'Sukkur', 'Lahore', 'Islamabad']
```

In [45]:

```
# Deleting and Removing elements of list

del cities[2]

# Nawabshah deleted
print(cities , end = "\n\n")


cities.remove("Sukkur")
# Sukkur removed
print(cities , end = "\n\n")


cities.pop(1)
# pop Hyderabad
print(cities , end = "\n\n")


cities.pop(2)
# pop Islamabad
print(cities , end = "\n\n")
```

```
['Karachi', 'Hyderabad', 'Sukkur', 'Lahore', 'Islamabad']

['Karachi', 'Hyderabad', 'Lahore', 'Islamabad']

['Karachi', 'Lahore', 'Islamabad']

['Karachi', 'Lahore']
```

Slicing

In [57]:

```
# Slicing in string

str = "I am OK!"

print("Length of str is : ",len(str))
print()

print("print 'am' from str: '", str[2:4] ,"'" )
print()

print("print 'I am OK' from str: '", str[:-1] ,"'" )
print()

print("Reverse of str: '", str[::-1] ,"'" )
print()
```

```
Length of str is :  8

print 'am' from str: ' am '

print 'I am OK' from str: ' I am OK '

Reverse of str: ' !KO ma I '
```

In [73]:

```
# Slicing in list

cars = ['Toyota Corolla','Suzuki Mehran','Suzuki Bolan', 'Suzuki Cultus','Honda City','Hond

print("All cars             " ,cars, end="\n\n")

print("First 3 cars:        ", cars[:3] , end="\n\n")

print("Last 3 cars:         ", cars[-3:] , end="\n\n")

print("Two cars from center:", cars[2:4] , end="\n\n")


print("All cars in reverse: ", cars[::-1] , end="\n\n")
```

```
All cars             ['Toyota Corolla', 'Suzuki Mehran', 'Suzuki Bolan', 'S
uzuki Cultus', 'Honda City', 'Honda Civic']

First 3 cars:        ['Toyota Corolla', 'Suzuki Mehran', 'Suzuki Bolan']

Last 3 cars:         ['Suzuki Cultus', 'Honda City', 'Honda Civic']

Two cars from center: ['Suzuki Bolan', 'Suzuki Cultus']

All cars in reverse:  ['Honda Civic', 'Honda City', 'Suzuki Cultus', 'Suzuki
Bolan', 'Suzuki Mehran', 'Toyota Corolla']
```

Tuples

In [84]:

```
# tuple—a list that's written in stone
weathers = ("Sunny", "Cloudy", "Rainy", "Windy","Snowy")

print(weathers)

# Accessing elements of tuple

sunny , cloudy, rainy, windy, snowy = weathers

print("\nAll elements of tuple ")

print(sunny)

print(cloudy)

print(rainy)

print(windy)

print(snowy)
```

```
('Sunny', 'Cloudy', 'Rainy', 'Windy', 'Snowy')

All elements of tuple
Sunny
Cloudy
Rainy
Windy
Snowy
```

Dictionaries

In [87]:

```
# Customer information

customer_1 = {
    "first_name" : "David",
    "last_name"  : "Elliott",
    "address"    : "4803 Wellesley St."
}

print(customer_1 , end = "\n\n")

print("First Name: ",customer_1['first_name'] , end = "\n\n")

print("Last Name: ",customer_1['last_name'] , end = "\n\n")
```

```
{'first_name': 'David', 'last_name': 'Elliott', 'address': '4803 Wellesley S
t.'}

First Name:  David

Last Name:  Elliott
```

In [93]:

```
print("Keys of dictionary: ", customer_1.keys() , end = "\n\n")

print("Values of dictionary: ",customer_1.values())
```

```
Keys of dictionary:  dict_keys(['first_name', 'last_name', 'address'])

Values of dictionary:  dict_values(['David', 'Elliott', '4803 Wellesley S
t.'])
```

In [106]:

```
# Insertion in dictionary

my_dict = {'apple': 'fruit', 'beetroot': 'vegetable'}

my_dict['doughnut'] = 'snack'

print(my_dict , end = "\n\n")

# dict.update(Iterable_Sequence of key:value)

my_dict.update({'cake':'dessert'})

print(my_dict , end = "\n\n")

# updating an existing value
my_dict.update( apple ='FRUIT')

print(my_dict , end = "\n\n")

# Inserting multiple values at one time

my_dict.update([ ('banana', 'snack') , ('tomato', 'vegetable')] )

print(my_dict , end = "\n\n")
```

```
{'apple': 'fruit', 'beetroot': 'vegetable', 'doughnut': 'snack'}

{'apple': 'fruit', 'beetroot': 'vegetable', 'doughnut': 'snack', 'cake': 'de
ssert'}

{'apple': 'FRUIT', 'beetroot': 'vegetable', 'doughnut': 'snack', 'cake': 'de
ssert'}

{'apple': 'FRUIT', 'beetroot': 'vegetable', 'doughnut': 'snack', 'cake': 'de
ssert', 'banana': 'snack', 'tomato': 'vegetable'}
```

In [108]:

```
# Removing and changing items

my_dict ['banana'] = "fruit"

print(my_dict , end = "\n\n")

del my_dict ['apple']

print(my_dict , end = "\n\n")
```

```
{'apple': 'FRUIT', 'beetroot': 'vegetable', 'doughnut': 'snack', 'cake': 'de
ssert', 'banana': 'fruit', 'tomato': 'vegetable'}

{'beetroot': 'vegetable', 'doughnut': 'snack', 'cake': 'dessert', 'banana':
'fruit', 'tomato': 'vegetable'}
```

Set

In [139]:

```
# set is a well-defined collection of distinct objects

'''
Set in Python is a data structure equivalent to sets in mathematics.
It may consist of various elements; the order of elements in a set is undefined.
You can add and delete elements of a set, you can iterate the elements of the set,
you can perform standard operations on sets (union, intersection, difference).
Besides that, you can check if an element belongs to a set.

'''


A = {1,2,3,4,5}

B = {3,1,5,2,4}

C = {'a','e','i','o','u'}


print("A = ", A , end="\n\n")

print("B = ",B , end="\n\n")

print("C = ",C , end="\n\n")

print( "A == B : ", A==B ,end="\n\n")

print("A == C : ", A==C ,end="\n\n")

# Insertion
A.add(6)

print("Updated A = ", A , end="\n\n")

print( "A == B : ", A==B ,end="\n\n")
```

```
A =  {1, 2, 3, 4, 5}

B =  {1, 2, 3, 4, 5}

C =  {'o', 'e', 'i', 'a', 'u'}

A == B :  True

A == C :  False

Updated A =  {1, 2, 3, 4, 5, 6}

A == B :  False
```

In [140]:

```
# Operations on sets

# B.add(7)

print("A = ", A , end="\n\n")

print("B = ",B , end="\n\n")

print( "A U B : ", A.union(B) ,end="\n\n")


# Intersection symbol unicode "\u2229"
print( "A \u2229 B : ", A.intersection(B) ,end="\n\n")


print( "A - B : ", A.difference(B) ,end="\n\n")

# Symmetric difference
# element belongs to A or B but not both
print( "A ^ B : ", A.symmetric_difference(B) ,end="\n\n")

print( "A \u2286 B : ", A.issubset(B) ,end="\n\n")

print( "A \u2287 B : ", A.issuperset(B) ,end="\n\n")

print( "A < B : ", A < B ,end="\n\n")

print( "A > B : ", A > B ,end="\n\n")
```

```
A =  {1, 2, 3, 4, 5, 6}

B =  {1, 2, 3, 4, 5}

A U B :  {1, 2, 3, 4, 5, 6}

A ∩ B :  {1, 2, 3, 4, 5}

A - B :  {6}

A ^ B :  {6}

A ⊆ B :  False

A ⊇ B :  True

A < B :  False

A > B :  True
```

In [147]:

```
print("A = ", A , end="\n\n")

print("C = ",C , end="\n\n")

A.discard(2)

C.discard('e')

print("A = ", A , end="\n\n")

print("C = ",C , end="\n\n")

C.remove('o')

print("C = ",C , end="\n\n")
```

```
A =  {1, 3, 4, 5, 6}

C =  {'o', 'i', 'a', 'u'}

A =  {1, 3, 4, 5, 6}

C =  {'o', 'i', 'a', 'u'}

C =  {'i', 'a', 'u'}
```

Looping through values

In [157]:

```
# List
cities = ["Khairpur","Sukkur","Lahore"]

# Dictionary
my_dict = {'apple': 'fruit', 'beetroot': 'vegetable', 'doughnut': 'snack', 'cake': 'dessert

# Tuple
weathers = ("Sunny", "Cloudy", "Rainy", "Windy","Snowy")

# Set
C = {'a','e','i','o','u'}


print("****************** All Cities ****************\n")

for city in cities:
    print(city , end = ", ")

print("\n\n")

print("****************** All Set Elements ****************\n")

for elem in C:
    print(elem , end= ", ")


print("\n\n")

print("****************** All Weathers ****************\n")

for weather in weathers:
    print(weather , end= ", ")


print("\n\n")

print("****************** All Dictionary Elements ****************\n")

for key,value in my_dict.items():
    print("{}  :  {}".format(key,value))
```

```
****************** All Cities ****************

Khairpur, Sukkur, Lahore,


****************** All Set Elements ****************

o, e, i, a, u,


****************** All Weathers ****************

Sunny, Cloudy, Rainy, Windy, Snowy,


****************** All Dictionary Elements ****************

apple  :  fruit
```

```
beetroot  :  vegetable
doughnut  :  snack
cake  :  dessert
```

Nested

In [162]:

```
# list of list
list1 = [1,2,3]
list2 = [4,5,6]

list = [list1,list2]

print(list ,end = "\n\n")

for one in list:
     for elem in one:
            print(elem, end = ", ")
```

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

1, 2, 3, 4, 5, 6,
```

In [167]:

```
# list of dictionaries

dict_1 = {
    0:"A",
    1:"B",
    2:"C"
}

dict_2 = {
    0:"D",
    1:"E",
    2:"F"
}

list = [dict_1,dict_2]

print(list ,end = "\n\n")

for dic in list:
    for key,val in dic.items():
            print(key," : ",val)
    print()
```

```
[{0: 'A', 1: 'B', 2: 'C'}, {0: 'D', 1: 'E', 2: 'F'}]

0  :  A
1  :  B
2  :  C

0  :  D
1  :  E
2  :  F
```

In [169]:

```
# Dictionary of Lists

list1 = [1,2,3]
list2 = [4,5,6]

dict = {
    "list1":list1,
    "list2":list2
}



print(dict ,end = "\n\n")


for key,val in dict.items():
    print(key , end = " : ")
    for i in val:
        print(i, end = ", ")
    print()
```

```
{'list1': [1, 2, 3], 'list2': [4, 5, 6]}

list1 : 1, 2, 3,
list2 : 4, 5, 6,
```

In [171]:

```
# Dictionary of Dictionaries


dict_1 = {
    0:"A",
    1:"B",
    2:"C"
}

dict_2 = {
    0:"D",
    1:"E",
    2:"F"
}

dict = {
    "dict_1":dict_1,
    "dict_2":dict_2
}

print(dict ,end = "\n\n")


for key,val in dict.items():
    print(key)
    for k,v in val.items():
        print(k," : ",v)
    print()
```

```
{'dict_1': {0: 'A', 1: 'B', 2: 'C'}, 'dict_2': {0: 'D', 1: 'E', 2: 'F'}}

dict_1
0  :  A
1  :  B
2  :  C

dict_2
0  :  D
1  :  E
2  :  F
```

List Comprehension

In [180]:

```
# list_variable = [x for x in iterable]

my_list = [i for i in range(10)]

print(my_list)
```

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

In [174]:

```
# S = {x² : x in {0 ... 9}}

S = [x**2 for x in range(10)]

print(S)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [179]:

```
# List contains even squared numbers between 0 to 9 inclusive

'''
numbers = range(10)

new_list = []

# Add values to `new_list`
for n in numbers:
    if n%2==0:
        new_list.append(n**2)


output : [0, 4, 16, 36, 64]

'''

numbers = range(10)

new_list = [n**2 for n in numbers if n%2==0]

print(new_list)
```

```
[0, 4, 16, 36, 64]
```

Performance Comparison of Loop and List Comprehension

In [19]:

```
# Let's study the difference in performance between the list comprehension and the for loop

# Timeit module provides a simple way to time small bits of Python code.


# Import `timeit`
import timeit

print("Execution time of list comprehension : ", end = " ")

# Print the execution time
print(timeit.timeit('[n**2 for n in range(10) if n%2==0]', number=10000))
```

```
Execution time of list comprehension :  0.044358099999996625
```

In [68]:

```
# Import `timeit`
import timeit

numbers = range(10)

new_list = []

# Define `power_two()`
def power_two(numbers):
    for n in numbers:
        if n%2==0:
            new_list.append(n**2)
    return new_list


print("Execution time of simple looping : ", end = " ")

# Print the execution time
print(timeit.timeit('power_two(numbers)', globals=globals(), number=10000))
```

```
Execution time of simple looping :  0.056625199999999154
```

In [38]:

```
# Conditions in list comprehension

nums = range(10)

# 0, 1,2,3,4,5,6,7,8,9

print(type(nums))

nums = list(nums)

print(type(nums), end="\n\n")

del nums[4]


updated_list = [ str(x) + " is greater than 5" if x > 5  else str(x) + " is less than 5" fo

for i in updated_list:
    print(i)
```

```
<class 'range'>
<class 'list'>

0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
5 is less than 5
6 is greater than 5
7 is greater than 5
8 is greater than 5
9 is greater than 5
```

In [49]:

```
# Nested List Comprehension

# 2D list
list_of_list = [[1,2,3],[4,5,6],[7,8,9]]

print("List of List")

for lst in list_of_list:
    print(lst)



# flatten this list of list by using list comprehension

flattened_list = [j for i in list_of_list for j in i]

# outer loop --> for i in list_of_list:
# inner loop --> for j in i:
# Now j contains elements one by one

print("\nFlattened List : ",flattened_list)
```

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

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

Lambda Expressions

In [51]:

```
'''

map() function returns a map object(which is an iterator) of the results
after applying the given function to each item of a given iterable

map(fun, iter)

'''

'''


# Return double of n
def addition(n):
    return n + n

# We double all numbers using map()
numbers = (1, 2, 3, 4)

result = map(addition, numbers)

print(list(result))


'''

numbers = (1, 2, 3, 4)

result = map(lambda x: x + x, numbers)

print(list(result))
```

```
[2, 4, 6, 8]
```

In [7]:

```
'''

1 Km = 3280.8399 feet

feet = [float(3280.8399)*x for x in kilometer]

print(feet)

'''


# Initialize the `kilometer` list
kilometer = [39.2, 36.5, 37.3, 37.8]

# Construct `feet` with `map()`
feet = map(lambda x: float(3280.8399)*x, kilometer)

# Print `feet` as a list
print(list(feet))
```

```
[128608.92408000001, 119750.65635, 122375.32826999998, 124015.74822]
```

In [17]:

```
# filter()

# This method apply filtering based on condition based on lambda
# Map the values of `feet` to integers


'''

feet = [int(x) for x in feet]
    print(feet)

uneven = [x%2 for x in feet]

print(uneven)

'''


# Convert `kilometer` to `feet`
feet = [float(3280.8399)*x for x in kilometer]

print(feet)

feet = list(map(int, feet))

# Filter `feet` to only include uneven distances
uneven = filter(lambda x: x % 2, feet)

# Check the type of `uneven`
print("\nType of uneven: ",type(uneven))

# Print `uneven` as a list or odd distances
print("\n Uneven List: ",list(uneven))
```

```
[128608.92408000001, 119750.65635, 122375.32826999998, 124015.74822]

Type of uneven:  <class 'filter'>

 Uneven List:  [122375, 124015]
```

In [20]:

```
# In Python 3 reduce() function is moved to functools

from functools import reduce

# It reduces the syntax

'''
Calculate sum of all elements in list

reduced_feet = sum([x for x in feet])

print(reduced_feet)

output : 494748

'''

#Calculate sum of all elements in list with reduce function
# Reduce `feet` to `reduced_feet`
reduced_feet = reduce(lambda x,y: x+y, feet)

# Print `reduced_feet`
print(reduced_feet)
```

```
494748
```

Generators

Python generators are a simple way of creating iterators. All the overhead we mentioned above are automatically handled by generators in Python. Simply speaking, a generator is a function that returns an object (iterator) which we can iterate over (one value at a time).

In [56]:

```
my_list = (i for i in range(10))
print(my_list)
```

```
<generator object <genexpr> at 0x000002726847A0C8>
```

List Comprehension returns a list whereas, Generators return generator object. Both can be iterated over

In [57]:

```
for i in my_list:
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

Generator Functions

generates a value with **yield** keyword

In [68]:

```
# This generator function generates values from 0 to given number n-1

def num_sequence(n):
    i = 0
    while i < n:
        yield i
        i += 1

result = num_sequence(5)

print(type(result))
```

```
<class 'generator'>
```

In [69]:

```
# next() method is used to iterate any iterable one value at a time

print(next(result))

print(next(result))

print(next(result))

print(next(result))

print(next(result))
```

```
0
1
2
3
4
```

In [70]:

```
# Intialize the list
my_list = [1, 3, 6, 10]

a = (x**2 for x in my_list)
# Output: 1
print(next(a))

# Output: 9
print(next(a))

# Output: 36
print(next(a))

# Output: 100
print(next(a))

# Output: StopIteration
next(a)
```

```
1
9
36
100


---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-70-ec1761f1e908> in <module>
     16
     17 # Output: StopIteration
---> 18 next(a)

StopIteration:
```

THE END!