Problem Solving with Python

Priyansh Singh

Dictionary

- There is currently only one standard mapping type, the dictionary. Dictionaries are an unordered collection of mappings between hashable values and arbitrary objects. Dictionaries are mutable objects.

- Keys are unique within a dictionary while values may not be. The values of a dictionary can be any arbitrary object (built-in or user defined) but the keys must be hashable.

- All immutable built-in objects are hashable, while no mutable containers (such as lists or dictionaries) are.

Dictionary

- Dictionaries can be created by placing a comma-separated list of key:value pairs within curly braces. See the example below.

```
# Keys should be unique and hashable (strings mostly). Look carefully
# the keys are given within quotes.
my_dict = {'name':'quick fox', 'age':25, 'weight':56}
```

Dictionary

- Dictionaries can also be created by making use of built-in type constructor - dict(). It can take keyword arguments, iterables (an iterable of tuples or lists each having exactly two objects, the first object becomes the key and the second object becomes the value) or mappings.

- Following is the syntax of this function.

```
dict([**kwargs])

dict(mapping [, **kwargs])

dict(iterable [, **kwargs])
```

- This function expects only one mapping (mapping) or one iterable (iterable). Extra key value pairs can be provided in the form of keyword arguments (**kwargs). Note that keyword arguments are optional. Below are some examples of the usage of this function.

```
# Using keyword arguments. The keywords become the keys of the dictionary. Note that the
keywords are not given within quotes.
my_dict = dict(name = 'quick fox', age = 25, weight = 56)

# Using key value value pairs in the form of mapping. Note that the keywords are given in
quotes.
my_dict = dict({'name':'quick fox', 'age':25, 'weight':56})

# Using key value pairs in the form of lists or tuples. Each tuple or list should have
exactly 2 elements. The first one becomes the key and the second one becomes the value.
Note that the keywords are given in quotes.
my_dict = dict([['name', 'quick fox'], ['age', 25], ['weight', 56]])
my_dict = dict([('name', 'quick fox'), ('age', 25), ('weight', 56)])

# Additional keyword arguments can be provided along with the mapping or iterable.
my_dict = dict({'name':'fox', 'age':25, 'weight':56}, height = 1.5)
my_dict = dict([['name', 'quick fox'], ['age', 25]], weight = 56)
```

```
# Creating the dictionary object
my_dict = {‘name’: ‘Quick Fox’, ‘area’: 24, ‘age’: 23.9}

# Value can be accessed with a key and square brackets
print(my_dict[‘name’])
print(my_dict[‘area’])

# Specific values can be modified.
my_dict['name'] = 'Lazy Fox'
print(my_dict)

# Adding new key-value pairs.
my_dict['height'] = 1.2
print(my_dict)
```

```
Quick Fox
24
{'name': 'Lazy Fox', 'area': 24, 'age': 23.9}
{'name': 'Lazy Fox', 'area': 24, 'age': 23.9, 'height': 1.2}
```

```python
# Delete specific value
del my_dict['height']

# Delete the complete dictionary.
del my_dict

# Checks for availability of a key in the dictionary
'name' in my_dict

# Checks for non-availability of a key in the dictionary
'name' not in my_dict

# Returns the number of elements in the dictionary
len(my_dict)
```

```
True
False
4
```

Dictionary Functions

- clear() - Removes all the elements from the dictionary.

- items(), keys() and values() - Return a new view object of dictionary’s key and value (in the form of tuples), keys and values respectively. View objects provide a dynamic view on the dictionary’s entries, which means that when the dictionary changes, the view reflects these changes.

- pop(key [, default]) - If key is in the dictionary, remove it and return its value, else return default. If defaul*t* is not given and key is not in the dictionary, a KeyError is raised.

Dictionary Functions

- popitem() - Remove and return a (key, value) tuple from the dictionary. Pairs are returned in LIFO order.

- update([**kwargs]) or update(mapping [, **kwargs]) or update(iterable [, **kwargs]) - Updates the dictionary with the supplied key-value pairs. If the key already exists, its value is updated, otherwise the key-value pair is added. The usage of this function is similar to the usage of dict() function.

Practice Questions

- Write a Python script to add a key to a dictionary.

Practice Questions

- Write a Python program to iterate over dictionaries using for loops

Dictionary Functions

```
d = {'x': 10, 'y': 20, 'z': 30}
for dict_key, dict_value in d.items():
      print(dict_key,'->',dict_value)
```

Practice Questions

- Write a Python script to concatenate following dictionaries to create a new one.

Sample Dictionary :
dic1={1:10, 2:20}
dic2={3:30, 4:40}
dic3={5:50,6:60}
Expected Result : {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

Practice Questions

- Write a Python script to check if a given key already exists in a dictionary.

Practice Questions

- Write a Python program to remove a key from a dictionary
- Write a Python program to remove duplicates from Dictionary.

Practice Questions

- Write a Python script to generate and print a dictionary that contains a number (between 1 and n) in the form (x, x*x).

Practice Questions

- Write a Python script to concatenate following dictionaries to create a new one.

Sample Dictionary :
dic1={1:10, 2:20}
dic2={3:30, 4:40}
dic3={5:50,6:60}
Expected Result : {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

Practice Questions

- Write a Python script to generate and print a dictionary that contains a number (between 1 and n) in the form (x, x*x).

Practice Questions

- Write a Python program to combine two dictionary adding values for common keys.

d1 = {'a': 100, 'b': 200, 'c':300}
d2 = {'a': 300, 'b': 200, 'd':400}
Sample output: Counter({'a': 400, 'b': 400, 'd': 400, 'c': 300})

Practice Questions

- Write a Python program to print all unique values in a dictionary.

Sample Data : [{"V":"S001"}, {"V": "S002"}, {"VI": "S001"}, {"VI": "S005"}, {"VII":"S005"}, {"V":"S009"},{"VIII":"S007"}]
Expected Output : Unique Values: {'S005', 'S002', 'S007', 'S001', 'S009'}

Practice Questions

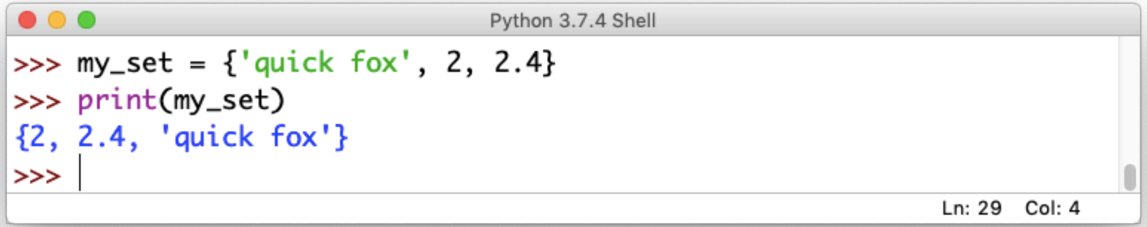
Write a Python program to create a dictionary from a string.
Note: Track the count of the letters from the string.

Sets

- A set object is an unordered collection of distinct hashable objects.
- This means only immutable types can become part of a set.
- Common uses include membership testing, removing duplicates from a sequence, and computing mathematical operations such as intersection, union, difference, and symmetric difference.

Sets

- Set objects can be created by passing a set of hashable objects within curly braces. See example below.



```
>>> my_set = {'quick fox', 2, 2.4}
>>> print(my_set)
{2, 2.4, 'quick fox'}
>>> 
```

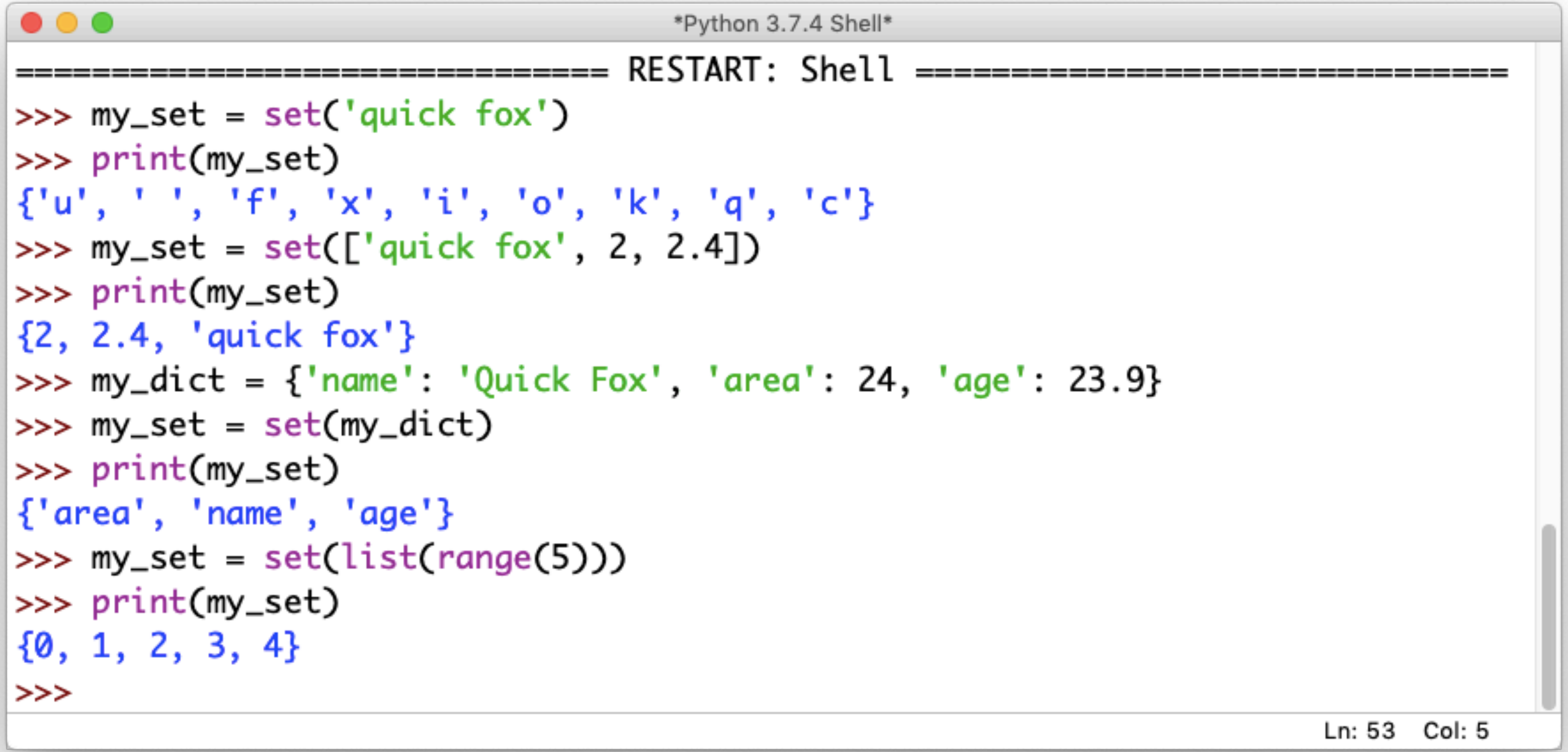
- Set objects can also be created by using the built in type constructor. The following is the syntax:

my_set = set(iterable)

Sets

- List, tuple, string, range, dictionary and set are all examples of iterables. The function can accept any one of these and create set from it. See below for sample usage of this function.

Sets


Python 3.7.4 Shell

```
================================ RESTART: Shell ================================
>>> my_set = set('quick fox')
>>> print(my_set)
{'u', ' ', 'f', 'x', 'i', 'o', 'k', 'q', 'c'}
>>> my_set = set(['quick fox', 2, 2.4])
>>> print(my_set)
{2, 2.4, 'quick fox'}
>>> my_dict = {'name': 'Quick Fox', 'area': 24, 'age': 23.9}
>>> my_set = set(my_dict)
>>> print(my_set)
{'area', 'name', 'age'}
>>> my_set = set(list(range(5)))
>>> print(my_set)
{0, 1, 2, 3, 4}
>>>
```

Ln: 53 Col: 5

Set Functions

Sets support the following operations:

- Inclusion check: obj in set returns True if obj is part of the set.
- Size of set: len(set) returns the number of elements in the set.

Following functions are available in the Set class:

- add(item) adds single element (item) to the set.
- update(iterable) adds all the elements of the iterable. Could be any iterable including list, tuple, string, range, dictionary, and another set or a set of objects within curly braces. Behaves exactly as set().

Set Functions

Following functions are available in the Set class:

- remove(element) removes element from the set. If the element is not found KeyError is raised.

- discard(element) removes element from the set. Will not raise any error, if the element is not found.

- pop() removes the last element. Thought you will never know which is the last element (set is an unordered collection).

Set Functions

Following functions are available in the Set class:

- clear() removes all the elements of the set.
- difference(other_set), intersection(other_set) or union(other_set) returns another set which is the difference, intersection or union with the other_set.
- issubset(other_set) and isuperset(other_set) returns True if the host set is a subset or superset of other_set.
- isdisjoint(other_set) returns True if the host set and the other_set has an intersection.

Practice Questions

Write a Python program to add member(s) in a set

Practice Questions

Write a Python program to remove an item from a set if it is present in the set.

Practice Questions

Write a Python program to create a union of sets

Practice Questions

Write a Python program to create a symmetric difference

Practice Questions

Write a Python program to find maximum and the minimum value in a set

Practice Questions

Determines if all the numbers from a list are different from each other. Attempt the problem using loops and sets and compare the performances of the two approaches.

Practice Questions

You have been given a list which may have duplicate entries. Create another list by picking unique elements from the given list. Try solving the problems using loops and sets and compare the time of execution of the two approaches.

Practice Questions

Create and print list of all unique values in a given dictionary. Implement the solution using sets and loops and compare the time of execution of the two approaches.

Practice Questions

Write a program to check if the string contains all the letters of the English alphabet. Attempt the problem using loops and sets and compare the performance of the two approaches.

Functions

- Function is a piece of code written to carry out a specified task. To carry out that specific task, the function might or might not need multiple inputs. When the task is carried out, the function can or cannot return one or more values.

Functions

- Function is a piece of code written to carry out a specified task. To carry out that specific task, the function might or might not need multiple inputs. When the task is carried out, the function can or cannot return one or more values.

- A **method** refers to a **function** which is part of a class. You access it with an instance or object of the class.

Functions

```
# Defining the function.
def sum_upto(n):
    sum = 0
    for i in range(n):
        sum += i
    return sum
# Calling the function
sum_upto(5)
```

```
10
```

Return Statement

Every function returns a value either explicitly using the return statement or implicitly (line number 9). Even if the function does not appear to return a value using the return statement the function actually returns None value (NoneType).

Return Statement

```
def nothing_returns(message):

        print(message)



print(nothing_returns(‘hello’))
```

```
hello

None
```

```
def get_data():

        return (‘Bheem’, 14, 32.5)



print(get_data())
```

```
(‘Bheem’, 14, 32.5)
```

Arguments

There are four types of arguments that Python uses:

- Required Arguments
- Keyword Arguments
- Default Arguments
- Variable Number of Arguments

Variable Length Arguments

Variable length arguments can be passed to a function using two special operators - asterisk (*) for non keyword arguments and double asterisk (**) for keyword arguments.

```
def does_nothing(*args):
        print(type(args))
        print(len(args))


does_nothing(45, 2.56, 'quick')
does_nothing(290, 'fox')


def still_does_nothing(param, *args):
        print(type(args))
        print(len(args))
        print(param)


still_does_nothing([1, 2, 3], 290, 'fox’)a
```

```
<class 'tuple'>
3
<class 'tuple'>
2
<class 'tuple'>
2
[1, 2, 3]
```

```python
def does_nothing(**kwargs):
      print(type(kwargs))
      print(len(kwargs))
      for key in kwargs.keys():
            print(key)

does_nothing(name = 'quick fox', age = 12)
```

```
<class 'dict'>
2
name
age
```

Lambda Functions

- While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword. The lambda function has the following syntax:

```
lambda arguments: expression
```

Lambda Functions

```
mod = lambda x, y: x**2 + y**2

print(mod(5, 2))
```

```
29
```

```
# Below line creates a list of the even numbers between 0 and 9.

my_list = list(filter(lambda x: x%2 == 0, range(10)))

print(my_list)


# Below line creates a list of squares of numbers from 0 to 9.

my_list = list(map(lambda x: x**2, range(10)))

print(my_list)
```

```
[0, 2, 4, 6, 8]

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

Built-In Functions

Conversion Functions

- float([number|string]): Convert a string or a number to a floating point. The argument may also be '[+|-]nan' or '[+|-]inf'. The string can optionally be preceded by a + or – (with no space in between) and optionally surrounded by whitespace.
- bool([object]): Converts the object to its corresponding truth value. Without argument it returns False.
- int([number|string): Convert a number or string to an integer. If no arguments are given, return 0. The argument may also be '[+|-]nan' or '[+|-]inf'. The string can optionally be preceded by a + or – (with no space in between) and optionally surrounded by whitespace.

Built-In Functions

- bin(x): Convert an integer number to a binary string.
- hex(x): Convert an integer number to a hexadecimal string.
- oct(x): Convert an integer number to an octal string.
- str(object): Converts the object into its string representation.
- list([object]): Return a list whose items are the same and in the same order as the items of object. object may be a sequence, a container that supports iteration, or an iterator object. If object is already a list, a copy is made and returned. For instance, list('abc') returns ['a', 'b', 'c'] and list((1, 2, 3)) returns [1, 2, 3]. If no argument is given, returns an empty list.

Built-In Functions

- tuple([iterable]): Return a tuple whose items are the same and in the same order as the iterable’s items. iterable may be a sequence, a container that supports iteration, or an iterator object.

- set([iterable]): Return a new set, optionally with elements taken from the iterable. If no argument is given returns an empty set.

- dict([arg]): Return a dictionary initialized with the optional argument. See dictionary section for details of usage of this function.

Built-In Functions

- abs(x): Return the absolute value of a number. The argument may be an integer or a floating point number. If the argument is a complex number, its magnitude is returned.

- all(iterable) or any(iterable): Return True if all elements of the iterable are true and the other function returns True if any element is true.

- divmod(a, b): Take two (non complex) numbers as arguments and return a pair of numbers consisting of their quotient and remainder when using integer division.

Built-In Functions

Other Functions

- abs(x): Return the absolute value of a number. The argument may be an integer or a floating point number. If the argument is a complex number, its magnitude is returned.

- all(iterable) or any(iterable): Return True if all elements of the iterable are true and the other function returns True if any element is true.

- divmod(a, b): Take two (non complex) numbers as arguments and return a pair of numbers consisting of their quotient and remainder when using integer division.

Built-In Functions

- id(object): Returns the identity of an object. This is an integer which is guaranteed to be unique and constant for this object during its lifetime. Two objects with non-overlapping lifetimes may have the same id.

- isinstance(object, classinfo): Return True if the object argument is an instance of the classinfo argument, or of a (direct or indirect) subclass thereof.

- iter(iterable|sequence): Return an iterator object.

Built-In Functions

- map(function, iterable, ...): Return an iterator after applying the given function to every item of iterable. If additional iterable arguments are passed, the function must take that many arguments and is applied to the items from all iterables in parallel. With multiple iterables, the iterator stops when the shortest iterable is exhausted.

Built-In Functions

```
num_one = [1, 2, 3]

num_two = [6, 7, 8]

def addition(x, y):

        return x + y


print(list(map(addition, num_one, num_two)))
```

```
[7, 9, 11]
```

Built-In Functions

- filter(function, iterable): Returns an iterator were the items are filtered through a function to test if the item is accepted or not. In simple words, the filter() method filters the given iterable with the help of a function that tests each element in the iterable to be true or not.

Built-In Functions

```
# Below line creates a list of the even numbers between 0 and 9.
my_list = list(filter(lambda x: x%2 == 0, range(10)))
print(my_list)
```

```
[0, 2, 4, 6, 8]
```

Built-In Functions

- min(iterable) or max(iterable) or min(args...) or max(args...): Returns the minimum or maximum value from the given iterable or the minimum or maximum of all the arguments.

- round(x [, n]): Return the floating point value x rounded to n digits after the decimal point. If n is omitted, it defaults to zero. The return value is an integer if called with one argument, otherwise of the same type as x.

- sum(iterable [, start]): Sums the items of the iterable from left to right starting from start. start defaults to 0. The items of the iterable are normally numbers, and are not allowed to be strings.

Built-In Functions

- zip(*iterables): Make an iterator that aggregates elements from each of the iterables. Returns an iterator of tuples, where the ith tuple contains the ith element from each of the argument sequences or iterables. The iterator stops when the shortest input iterable is exhausted. With no arguments, it returns an empty iterator.

- reversed(iterable): Returns an iterable that accesses the iterable in reverse order.

Built-In Functions

- sorted(iterable [, key] [, reverse]): Returns a new sorted iterable.

- The sorting will be in descending order if reverse is set to True.
- A comparison function can be provided through the optional key parameter.
- Any built-in or custom functions can be specified for comparison, **as long as it takes one argument and returns one value**. The sorted function applies the key function only once to every element of the iterable and uses the returned value to sort the iterable.

Built-In Functions

```
my_list = ['quick', 'brown', 'fox', 'jumps']


# Sorts the list lexicographically.

sorted(my_list)


def third(st):

	return st[2]


# Sorts the list in descending order using the third character.

sorted(my_list, key = third, reverse = True)
```

```
['brown', 'fox', 'jumps', 'quick']

['fox', 'brown', 'jumps', 'quick']
```

Built-In Functions

- sort([, key] [, reverse]) method in the list class can also be used for sorting.
- It is to be noted that this method does sorting in-place. In other words instead of returning a new list it modifies the list in-place.