BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 5

DCA3104

PYTHON PROGRAMMING

Unit 9
Tuple, Sets, and Dictionary

Table of Contents

1. INTRODUCTION

In the previous chapters, you learned about the specific characteristics and distinctiveness of different features in the Python programming language like various operators, data types, etc. Python allows programmers to use basic operators that are similar to mathematical operators such as addition (+), multiplication (*), exponentiation (**), etc. Variables are also a key component in Python coding for memory storage and to execute tasks. As you now know, there are also many types of data types that the variables can store, such as string, integer, boolean, floating numbers, etc.

You also learned about branching and looping in Python. Conditional statements and loops are used by a programmer when they want to change the program direction. They do this by setting conditions that direct the program execution. The previous chapters also covered data structures and some of the types of data structures.

Data structures provide a fundamental construct to your program based on which you can organize data and retrieve them more efficiently. You have already learned how to use some data structures/sequence/iterables like list and string. Iterators are objects that store a given number of values that can iterate. Lists, tuples, sets, and dictionaries all act are iterable containers from which you can get an iterator.

In this chapter, you will learn about other data structures like tuple, sets, and dictionaries. You will learn how to create, add, update, delete and retrieve data from these data structures in your program. This unit's activities will help you familiarise yourself with the given data structures and assess how to utilize them efficiently.

1.1 Learning Objectives

After studying this chapter, you will be able to:

- ❖ *Create, delete and use the basic operations in a tuple. You will also be able to index and slice in a tuple.*
- ❖ *Create and use different functions in Sets like set(), update(), add(), and discard().*
- ❖ *Create, add and delete data in a Dictionary. You will also know how to access data, perform iteration, and use the built-in functions of a Dictionary.*

2. TUPLE

A tuple is an ordered collection of items and is a part of the standard Python language. Programmers can use a tuple to store multiple items or objects in a single variable.

2.1 Introduction

The tuple is like a list in many aspects and is one of Python's built-in data types. However, unlike a list, tuples are immutable. That means we cannot change the elements once we assign them to a tuple, but we can reassign or delete an entire tuple. The immutable nature of tuples helps programmers manipulate them faster when compared to a list. As the elements are intended to stay constant throughout the program, using tuple prevents any accidental data collection changes.

STUDY NOTE

If the individual elements in a tuple are mutable, we can change their values within the tuple.

Another feature that differentiates a list and a tuple is how they are enclosed. We enclose the items of a list within square brackets ("[...]"). On the other hand, we use parenthesis or round bracket ("(...)") to enclose the elements. Although we use parenthesis in a tuple, we refer to the index number of the elements (enclosed in square brackets) to access them, similar to lists and strings.

2.2 Creating Tuple

Tuples collect and hold their elements inside a parenthesis. They are characterized by immutable nature . Tuple has the capacity to hold multiple elements under a single variable name.

```
# tuple with parenthesis
l = (a, c, e)
```

You can also make tuples without using the parenthesis, but it a good practice to use them.

```
# tuple without parenthesis
l = q, f
print(l)
(q, f)
```

For the most part, tuples include more than one item. However, the comma must be used even when creating tuples with single elements.

```
# tuple with a single element
single = (a, )
```

As a tuple is indexed, it allows programmers to use a single element multiple times.

```
# tuple with duplicated values
my_tup = ('pineapple', 'apple', 'banana', 'banana', 'apple')
print (my_tup)
```

You can also create tuples with a mix of various data types or nested elements.

```
# tuple with mixed data types
a=12
my_tup = (a, 'Hi', 3.4)
print(my_tup)
# nested tuple
my_tup = ('horse', [2, 7, 6], (1, 4, 2))
print(my_tup)
```

2.3 Indexing and Slicing

In Python, we use indexing and slicing to access elements in sequence data types like a tuple. Indexing is used to access individual items by using their index numbers; whereas, slicing is used to access a sequence of elements.

STUDY NOTE

Python is a zero-index programming language. That means a sequence with four items will have an index: 0, 1, 2. 3.

In both indexing and slicing, the index operator "[]" is incorporated to access an element. The program then fetches the respective item from the given tuple (counting from the left). Indexing is limited only to the number of items in the tuple. If you try to retrieve an element beyond the index range: the program will raise an IndexError.

Using float or other data types will result in a TypeError. Hence, only integers are allowed in indexing.

```
my_tuple = ('my', 'name', 'is')
print(my_tuple[1])
```

Similarly, if you wish to retrieve items from a nested tuple, you use nested indexing.

```
# nested tuple
nes_tuple = ("horse", [2, 7, 6], (1, 5, 8))
# nested index
print(nes_tuple[0][3]) # s is retrieved
print(nes_tuple[2][2]) # 8 is retrieved
```

In slicing, we use the slicing operator (:) to retrieve a range of elements. To understand how the range works, imagine that the index is present between the elements instead. To access a range, we use the index that will slice that particular range.

Example:

```
slice_tuple = ('l', 'o', 'n', 'g', 'i', 't', 'u', 'd', 'e')
print(slice_tuple[1:4])
# elements 2nd to 4th are sliced
# output will be ( 'o', 'n', 'g' )
```

You can also access a range from the beginning or end to a particular.

```
slice_tuple = ('l', 'o', 'n', 'g', 'i', 't', 'u', 'd', 'e')
print(slice_tuple[6:])
# elements from 7th to end are sliced
# output will be ( 'u', 'd', 'e' )
slice_tuple = ('l', 'o', 'n', 'g', 'i', 't', 'u', 'd', 'e')
print(slice_tuple[:6])
# elements from beginning to 5th are sliced
# output will be ( 'l', 'o', 'n', 'g', 'i', 't' )
```

2.4 Negative Indexing

In Python, the sequence can also have negative indexing. The last item in the sequence is assigned -1 index, the second to last has -2 index, etc.

```
# Using negative indexing
my_tuple = ('l', 'o', 'n', 'g', 'i', 't', 'u', 'd', 'e')
# For output: 'i'
print(my_tuple[-5])
```

Slicing using the negative index:

```
my_tuple = ('l', 'o', 'n', 'g', 'i', 't', 'u', 'd', 'e')
# elements 3rd to 5th
print(my_tuple[-3:-5])
# Output: ( 'n', 'g', 'i' )
```

2.5 Delete a Tuple

As discussed earlier, tuples are unchangeable sequences. It explains the unvarying nature of the elements in the tuple. However, we can delete a tuple entirely by using the "del" keyword.

```
# Deleting a tuple
my_tuple = ('l', 'o', 'n', 'g', 'i', 't', 'u', 'd', 'e')
del my_tuple
```

In the above example, if you try to print the tuple "my_tuple," the program will throw a Name Error.

2.6 Basic Operations of Tuple

We can perform some basic operations in a tuple as follows:

A. **Addition:** Using the addition operator (+), we can concatenate two or more tuples.

```
# '+' will join the three tuples given
t = (2, 5, 0) + (1, 3) + (4, 5, 6)
print (t)
# Output: (2, 5, 0, 1, 3, 4, 5, 6)
```

B. **Multiplication:** We use the multiplication operator to (*) to create a repetitive tuple sequence. When a tuple is multiplied with an integer 'x', it creates a new tuple with all the initial tuple elements repeated 'x' times.

```
# Multiplying a tuple with an integer
t = ('l', 'a')
print (t*3)
# Output: ( 'l', 'a', 'l', 'a', 'l', 'a' )
```

STUDY NOTE

You can use the len() function to determine the number of items in a tuple.
my_tup=(“pineapple”, “banana”, “apple”)
print(len(my_tup))

C. **Using "in" Keyword:** We can check the existence of an element in a tuple using the "in" keyword. If the element exists, the program will return 'True'; otherwise, it returns 'False'.

```
#Using "in" keyword
my_tuple = ('l', 'o', 'n', 'g', 'i', 't', 'u', 'd', 'e')
print('n' in my_tuple) # Output: True
print('a' in my_tuple) # Output: False
```

D. **min() and max() Function:** Using the min() and max() function, we can determine the tuple's minimum and maximum values.

```
#Using min() and max() Function
my_tuple = (1,2,3,4,5,6)
print (max(my_tuple)) # Output: 6
print (min(my_tuple)) # Output: 1
```

E. **Iterating in a Tuple:** We can use the for loop to iterate through a tuple as follows:

```
# Using for loop to iterate through a tuple
for name in ('Joshua', 'Mike'):
  print("Hello", name)
# Output
Hello Joshua
Hello Mike
```

Activity 1

Create a program that calculated the element-wise sum of three tuples. Assign the same number of integer data type elements to three tuples. Your output should print the three tuples used along with the sum of the elements that belong to the same index in each tuple.

SELF-ASSESSMENT QUESTIONS – 1

1. ____________ allows programmers to store multiple items or objects in a single variable.
2. Tuple items are separated using the ____________.
3. ____________ brackets are used in indexing and slicing.
4. ____________ is used to find create a tuple with repetitive elements.
5. A tuple can store only one data type and does not allow duplication. [True/**False**]
6. Negative indexing starts from the first element in the tuple. [True/**False**]
7. Suppose t = (a, b, r, d), which of the following is incorrect?
   a) print(t[r])
   b) t[r] = 45
   c) print(max(t))
   d) print(len(t))
8. The output of the following code:
   t = (a,d,d,i,c,t,i,o,n)
   print(t[:-3])
   a) (a,d,d,i,c,t,i,o,n)
   b) (a,d,d,i,c,t)
   c) (a,d,d,i)
   d) (a,d,d,i,c,t,i)

3. SETS

Unlike a tuple, a Set is an unorganized and unindexed collection of objects. The items in a set do not have a defined order and may appear in a different order every time we use them. Hence, they cannot be accessed or referred to using a key or index.

3.1 Introduction

Every value in a set is unique and unchangeable. Set elements are unordered. That is, there is no specific order in which the elements appear. Set features restrict duplicate values. In addition, we can add additional elements to the existing set. Set hold its elements in curly brackets "{...}".

3.2 Creating a Set

As mentioned above, we can create a set by enclosing elements in curly brackets separated by a comma.

A set can store any number of items, and they can be of different data types (boolean, integer, string, etc.). However, a set cannot have a mutable element like lists or dictionaries.

STUDY NOTE

We cannot create an empty set with ("{}") as this would create an empty dictionary.

You can check the datatype of a variable using print(type(<variable>)).

```
# Set with mixed data types
my_set = {5.0, "H1", (1, 2, 3)}
print(my_set)
```

3.3 Set() Method

The set() method or set construct is used to convert an iterable sequence into a set. We can also create sets using the built-in set() function in Python. It follows the following syntax.

set<iterable>

The "iterable" in the above syntax may be a list, a dictionary, a string, or a tuple. We can create an empty set using the set() method if we do not specify any argument/iterable in it.

```
# Creating an empty set
a = set()
```

We can create a set from a string in the following manner.

```
#Set from a string
my_set = (set('Language'))
print(my_set)
#Output:
{ 'L', 'n', 'u', 'a', 'g', 'e' }
```

We can create a set from a tuple in the following manner.

```
my_set = (set(('a', 'b', 'o', 'a', 'r', 'd')))
print(my_set)
#Output:
{'a', 'b', 'd', 'o', 'r'}
```

We can create a set from a list in the following manner.

```
#Set from a List
my_set = set([1, 2, 3, 2])
print(my_set)
#Output:
{1,2,3}
# The last '2' is not included as a set does not allow duplicates.
```

3.4 Add()

Although sets are mutable, they are unorganized. Hence, indexing has no meaning, and we cannot use indexing or slicing to access elements in sets.

STUDY NOTE

The add() will become invalid if the element subjected to add is already present in the set.

We can add a new item in a set using the add() method. The add() method is used with the following syntax:

set.add(elem)

Where "elem" is the element that we want to add. We cannot get back a set if we use the add() method while creating a new set.

noneValue = set().add(elem)

The above expression will return "none" as that is the return type of add().

Let us look at an example of how to add an element to a set.

```
letter = {'c', 'o', 'p', 'p', 'e'}
# adding 'r'
letter.add( 'r' )
print('The letters are:', letter)

#Output:
The letters are: {'p', 'c', 'r', 'e', 'o'}
# The order of the letters may be different.
```

STUDY NOTE

Add() function will add an only individual element, unless they are other sequences like list, tuple, etc.

We can add a tuple to a set as:

```
# adding a tuple to a set
letter = {'c', 'o', 'p'}
# the tuple
tup = ('p', 'e', 'r')
# using add()
letter.add(tup)
print('The letters are:', letter)
#Output:
The letters are: { ('p', 'e', 'r'), 'c', 'o', 'p'}
```

3.5 Update()

The update() method is similar to the add() method. However, we can add multiple elements at the same time using the update() method. The update() method can accept any iterable like tuples, lists, strings, or other sets. It has the following syntax:

set.update(iterable1, iterable2, iterable3)

Similar to the add() method, update() does not return any value. Let us see how the update() is used to add another set to an existing one.

```
letter = {'p', 'y', 't'}
let = {'t', 'h', 'o', 'n'}
# using update()
```

```
letter.update(let)
print ("The word is ", letter)
Output:
The word is {'p', 'y', 't', 'h', 'o', 'n' }
```

3.6 Discard()

In Python, the library function discard() is programmed to eliminate specific elements from the set, given the elements are available in the set. Discard() function will not display error if the element specified in the discard() function is not available in the iterable unlike remove() function.

Syntax:
set.discard(x)

Where "x" is the element to be removed, the discard() method does not have a return value.

We can use discard() in the following ways:

Method 1:

```
num = {2, 6, 4, 5, 3}
num.discard(3)
print('numbers = ', num)
# Output: numbers = {2, 6, 4, 5}
```

The output remains the same even if we use the code below:

```
num.discard(15)
print('numbers = ', num)
# Output: numbers = {2, 6, 4, 5}
```

Method 2:

```
num = {2, 6, 4, 5, 3}
print(num.discard(3))
# The above expression returns "None"
print('numbers = ', num)
# Output: numbers = {2, 6, 4, 5}
```

Activity 2

Create a set and a tuple with the names of your peers. Write a Python program to add the tuple to the set to create a new set. Write a program to iterate through all the elements of the set using the *for* a loop.

SELF-ASSESSMENT QUESTIONS – 2

9. ___________ method does not give an error on removing an element that does not exist.
10. Set elements are enclose within ____________ brackets.
11. We can add multiple elements to a set using the __________ method.
12. Sets are mutable, unorganized, and non-indexed collections. [**True**/False]
13. We cannot convert iterable elements into a set. [True/**False**]
14. We cannot use the add() method while creating a new set. [**True**/False]
15. The Python code "s = {2, 3, 4, [5, 6]}" will throw an error. [**True**/False]
16. What will be the result of the following code:

```
lett = set(['a','a','b','b','c','c','c','c'])
print(len(lett))
```

   a) 7
   b) Error, invalid syntax for formation of set
   c) 3
   d) 8

4. DICTIONARY

Dictionaries are central data structures in Python that store an ordered collection of items. Each item in a dictionary is mutable and identified using a key name (or key) with an associated value. Unlike an index, a key can be of any data type, such as string, float, integer, etc., making it more flexible.

Dictionaries can store any number of items and are also referred to as maps, hashmaps, lookup tables, or associative arrays. In the real world perspective, dictionaries can be associated with phonebooks. They help you quickly retrieve information, such as a phone number, with a specific key (person's name).

4.1 Creating a Dictionary

As mentioned above, each element in a dictionary is tied to a key. The corresponding key is used to get/view that particular element in the dictionary. Compared to conventional programming languages, Python offers the flexibility to select the desired key for elements. However, the key must be unique and immutable data types such as string, number, or tuple. On the other hand, the item associated with the key may be of any data type.

STUDY NOTE

Dictionaries are created with curly brackets ({}), containing key-value pairs separated by commas (,).

A dictionary has the following syntax:
variable = {key1 : item1, key2 : item2...}

The key and its value are separated by a colon (:), the unit items are separated by commas, and the whole sequence is enclosed within curly brackets.

We can create an empty dictionary in the following manner:
empty = {}

4.2 Accessing Values of Dictionary

Unlike other ordered data structures that use indexing, dictionaries use keys to access or retrieve items. We can access the values in the dictionary by using the square bracket ("[...]") or the get() method.

If a value does not exist in the dictionary and we use the square brackets, the program raises a KeyError. In contrast, the get() method returns a "None" value.

Example:

```
# retrieving elements
my_dict = {'name': 'Phil', 'age': 50, 'country': 'America'}
# Accessing value using []
print(my_dict['name']) #Output: Phil
# Accessing value using get()
print(my_dict.get('country'))

#Output: America
```

4.3 Adding and Deleting Values in Dictionary

In Python, we can append a new element/item or alter the existing value using the assignment operator "=". To add or append a new item, a new key and associated value should be constructed in the syntax. In order to alter the existing element describes the new value for the existing key of the dictionary in the syntax. .

```
# Adding and updating values in a dictionary
my_dict = {'name': 'Phil', 'age': 50}
# Adding new key : value pair
my_dict['country'] = 'America'
# "country" is the key and "America" is its value
print(my_dict)
# Output: {'name': 'Phil', 'age': 50, 'country': 'America'}
# Changing an existing value
my_dict['age'] = 53
print(my_dict)
# Output: {'name': 'Phil', 'age': 53, 'country': 'America'}
```

We can delete the items from a dictionary using the del keyword with the key enclosed in square brackets.

```
my_dict = {10 : 'rose', 20 : 'lily', 30 : 'tulip', 40 : 'lotus', 50 : 'marigold'}
# Deleting item using del keyword
```

```
del my_dict [30]
print(my_dict)
#Output:
{10: 'rose', 20: 'lily', 40: 'lotus', 50: 'marigold'}
```

If you wish to remove an item and return the value associated with it, you can use the pop() method.

```
my_dict = {10 : 'rose', 20 : 'lily', 30 : 'tulip', 40 : 'lotus', 50 : 'marigold'}
# Deleting item using pop() method
print(my_dict.pop(40))
#Output:
lotus
```

The popitem() method is used to delete the last item in the dictionary and return its key and value. Using the clear() method, delete all the items in a dictionary.

```
my_dict = {10 : 'rose', 20 : 'lily', 30 : 'tulip', 40 : 'lotus', 50 : 'marigold'}
# Deleting item using popitem() method
print(my_dict.popitem())
Output:
(50, 'marigold')
# Deleting all the items
print(my_dict.clear())
#Output:
None
```

4.4 Iteration in Dictionary

Similar to lists, we can retrieve multiple elements from a dictionary by using the for loop.

```
# Retrieving values by using for loop
my_dict = {'name': 'Phil', 'country': 'America'}
for i in my_dict:
   print ("Key: " + i + " and Value: " + my_dict[i]);
# Using the variable 'i' directly will return only the key in the dictionary and not the value.
# The key is used just like an index
```

```
#Output:
Key: name and Value: Phil
Key: country and Value: America
```

You cannot directly concatenate an integer with a string when using the for loop. However, we can use the format() method in the following manner.

```
my_dict = {'name': 'Phil', 'age': 50, 'country': 'America'}
for i in my_dict:
print ("Key: {0} and Value: {1}".format(i, my_dict[i]));

#Output:
Key: name and Value: Phil
Key: age and Value: 50
Key: country and Value: America
```

4.5 Built-In Functions of Dictionary

Python offers some functions in a dictionary to perform tasks.

- **all():**

The python library function 'all()' checks whether all entries or items in the iterable is true. If the iterable contains completely true element, the any() function returns 'true' whereas it returns a 'false' if there is any one false element in the iterable object, that is iterable contains a 0 or false. If there is no element in the object, then the all() returns "true".

Syntax: all(iterable).

```
my_dict = {10 : 'rose', 20 : 'lily', 30 : 'tulip', 40 : 'lotus', 50 : 'marigold'}
print(all(my_dict))
#Output :
True
# All keys are True
my_dict = {0 : 'rose', 20 : 'lily', 30 : 'tulip', 40 : 'lotus', 50 : 'marigold'}
print(all(my_dict))
#Output :
False
# 0 is False
```

- **any():** The python library function 'any()' checks whether minimum one entry/item in the iterable is true. If the iterable contains atleast one true element, the any() function returns 'true' whereas it returns a 'false' if there is no true element in the iterable object. That is the iterable contain only either 0 or false.

Syntax: any(iterable).

```
my_dict = {10 : 'rose', 20 : 'lily', 30 : 'tulip', 40 : 'lotus'}
print(any(my_dict))
#Output :
True
# All keys are True
my_dict = {}
print(any(my_dict))
Output :
False
# Empty dictionary
```

- **len():** It returns the number of items (length) in a dictionary.

```
my_dict = {10 : 'rose', 20 : 'lily', 30 : 'tulip', 40 : 'lotus'}
print(len(my_dict))
#Output :
4
```

- **cmp():** Compares the items of two dictionaries. However, this function is not available in Python 3.
- **sorted():** Python has the library function to easily sort any sequence, 'sorted()'. Sorted() function gives sorted output and it does not alter the original sequence.

The syntax is:Sorted(iterable, key, reverse)

The parameter 'iterable' is mandatory. Iterable disclose the object to be sorted. While key and reverse parameters are optional. If the reverse parameter is given value true, then reverse sorting takes place.

```
# Sorting the key as a list
my_dict = {10 : 'rose', 20 : 'lily', 30 : 'tulip', 40 : 'lotus'}
```

```
print(sorted(my_dict))
#Output:
[10, 20, 30, 40]
# Using the reverse parameter
my_dict = {10 : 'rose', 20 : 'lily', 30 : 'tulip', 40 : 'lotus'}
print(sorted(my_dict, reverse = True))
#Output:
[40, 30, 20, 10]
# Reverse key order
# Using the key function
my_dict = {'10' : 'rose', '2000' : 'lily', '30' : 'tulip', '400' : 'lotus'}
print(sorted(my_dict, key = len))
#Output:
['10', '30', '400', '2000']
# Thelen function sorts the keys based on their length
```

Activity 3

Create a dictionary for student marks of 10 students, where the student's name is the "key," and the marks are the "value". Add a new item pair for "Priya," who got 75 marks in her test. Then, sort the items in descending order, based on the student names.

SELF-ASSESSMENT QUESTIONS – 3

17. Associative array are also known as ____________.
18. Each element/ value in a dictionary is assigned a unique _______.
19. We use the _______ operator to add or update values in a dictionary.
20. We use the all() function to know the number of entries in a dictionary. [True/**False**]
21. The key is used like an index to iterate in the for loop. [**True**/False]
22. Which of the following creates a dictionary?
    a) d = {rose : 35, 'nick':34}
    b) d = {“rose”:40, “nick”:45}
    c) d = {40, ”rose”, 45, ”nick”}
    d) All of the mentioned

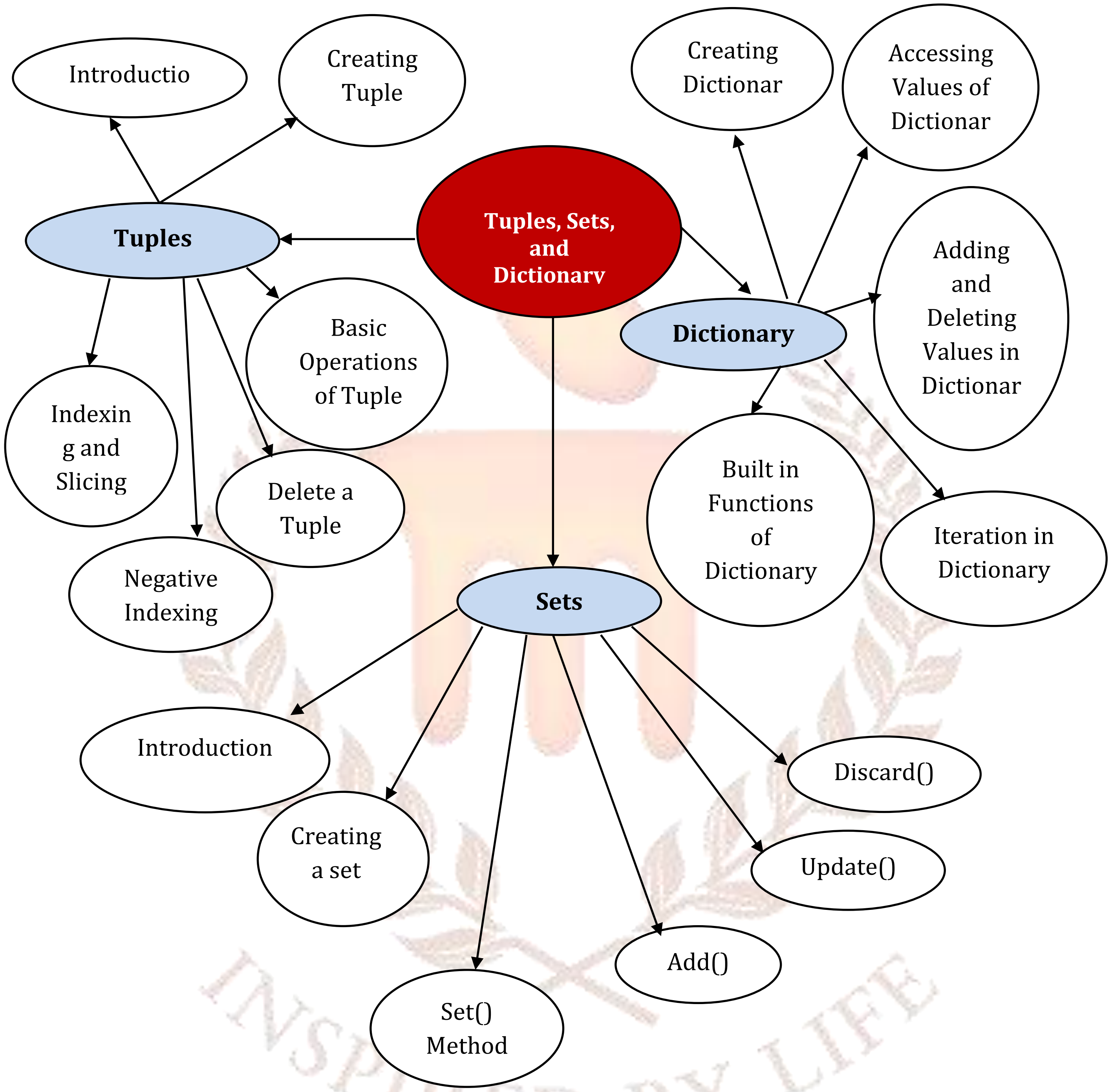

Fig 1: Conceptual Map

5. SUMMARY

- Data structures are the basic constructs used by programmers to organize and access data quickly. Data structures are often referred to as sequences. Also, they are a collection of data objects or elements using which programs are executed.
- Data structures may be ordered (each element has a unique index or key using which they are retrieved), or maybe unordered (the elements have no proper order and give different outputs on retrieval).
- A tuple is an ordered data structure that has immutable elements enclosed in round brackets. Each element in it is indexed. Programmers use index values to retrieve individual or sequences of elements.
- Individual elements are retrieved through indexing, where the element index is mentioned in '[]' brackets. Positive indexing starts from '0' and moves from left to right in the sequence. Negative indexing gives the '-1' value to the last element and decreases from right to left.
- Segments of a sequence are retrieved through slicing by using the slicing operator (':’).
- We can perform basic mathematical functions on a tuple, like addition, multiplication, etc.
- A set is an unordered data structure whose elements cannot be indexed. The elements are immutable, non-duplicated, and are enclosed in curly brackets.
- Python has built-in set methods like update(), discard(), set(), add(). These methods do not return a value.
- The set () method is used to create a new set or convert another iterable into a set.
- Add() method, add single elements or iterables to an existing set.
- We add multiple elements or iterables by using the update() method.
- Discard() is used to remove elements or iterable from a set. Unlike remove() it does not raise an error if an item does not exist in a set.
- A dictionary is an ordered data structure whose elements are defined as a (key: value) pair. The key must be of an immutable data type, whereas the value can be of any data type. The keys make item retrieval faster from dictionaries.
- The key-value pairs must be separated by (:), and a comma should separate each item. We can access items from a dictionary by mentioning the key in '[]' brackets.

- The "=" operator can add item pairs. It enables the addition of new item pairs or the updating of existing key values. Using the 'del' keyword, it deletes items in a dictionary.
- Using for loop Dictionary items can be iterated.
- Built-in functions like all(), any(), sorted() can be used to perform tasks on dictionaries.

6. GLOSSARY

Object: Any data that has a defined value and behaviour. Also called element or item.

Argument: A value passed to a function or a method when calling them.

Iterable: It is an object that can be iterated over. In sequences, it returns its members one after the other.

Function: A series of statements that return a value to the caller.

Hashable: An object is said to be hashable if its value never changes during its lifetime. Hashable objects are used as a dictionary key and a set member.

Immutable: An object that has a fixed value.

Key function: It is a callable that delivers a value used for sorting or ordering.

Object: Any data that has a defined value and behavior. Also called element or item.

Argument: A value passed to a function or a method when calling them.

Iterable: It is an object that can be iterated over. In sequences, it returns its members one after the other.

Function - A series of statements that return a value to the caller.

Hashable - An object is said to be hashable if its value never changes during its lifetime. Hashable objects are used as a dictionary key and a set member.

Immutable - An object that has a fixed value.

7. CASE STUDY

Robots: The Game

A version of the classic console game "Robot" was created, in which the programmer used a subclass of dictionary called a box. The robot is a turn-based game in which the player tries to stay alive by avoiding relentless robots. The game is set up so that the robots move one square towards the player in every move. The player dies if the robot catches them. If the robots collide with each other, they die and leave behind "junk," which also kills them.

The game's primary strategy is that the player has to position themselves so that the robots colloid with each other or the junk piles every time they move towards the player. The player also can teleport three times in the game.

The programmer used a subclass of the dictionary known as "box" to create the game grid and move the game elements according to the player's actions. The box is a subclass that overrides a few base functionalities to ensure everything stored in the dictionary can be accessed as an attribute or key value. It acts as a replacement for dictionaries and can include multiple dictionaries, lists, etc. Each dictionary or list added to the box object becomes a part of the sub-class and is retrieved using unique keys assigned to it. The programmer used the box as a collection of the grid dimensions and object coordinates that can be retrieved whenever the box is called.

Source- *www.openbookproject.net*

Discussion Questions:

1. How is the box sub-class better than a normal dictionary?
2. Can you guess when the programmer used the box in the program code is?
3. Try to create a box sub-class that can create the game grid?

8. TERMINAL QUESTIONS

SHORT ANSWER QUESTIONS

Q1. Which is faster for data retrieval, list, or dictionary?

Q2. Retrieve the first four numbers from the given tuple by using negative indexing:

tup = (3, 56, 23, 4, 55, 7, 1, 8, 532)

Q3. How are lists and tuples different in Python?

Q4. Determine the output of the code given below.

```
student = {'Student': 'Dave', 'mark': 100}
print(student['Student'])
print(student['DOB'])
```

Q5. Correct the code given below so that no KeyError will be thrown.

```
student = {'Student': 'Dave', 'mark': 100}
print(student['Student'])
print(student['DOB'])
```

LONG ANSWER QUESTIONS

Q1. Write a program to find the length, minimum and maximum value, and iterate over the set.

Q2. Write a program to calculate the sum of elements in each tuple present in a list of tuples. The given tuples are:

(1,5)

(4,3)

(2,4)

(9,3)

Q3. Write a program to convert the following two dictionaries into a list of lists:

{1 : 'blue', 2 : 'green', 3 : 'brown', 4 : 'pink', 5 : 'black'}

{'10' : 'Austin', '22' : 'Natasha', '34' : 'Alfred', '54' : 'Jamie'}

Q4. Write a program to find the area of each squares having sides 1, 2 ,3 ,4. Use dictionary for storing the results.

Q5. Write a program to find the number of unique alphabets in string "programming" using set.

8.1 Answers

SELF ASSESSMENT QUESTIONS

1. Tuple
2. Commas
3. Square brackets
4. Multiplication operator
5. False
6. False
7. False
8. B. A tuple is immutable. Hence, we cannot change the values of the elements.
9. Discard()
10. {} bracktes
11. update()
12. True
13. False
14. True
15. True. A set can only contain hashable/immutable data types. A list is a mutable data type and hence shows an error.
16. C. A set does not allow duplicates
17. Dictionary
18. Key
19. “=” operator
20. False
21. True
22. B. Dictionaries are created by using the key: value format.

TERMINAL QUESTIONS

SHORT ANSWER QUESTIONS

Answer 1: A list takes more time to retrieve or find data as the program filters through every value in the list. On the other hand, a dictionary has a well-defined key: value item pairs that allow you to quickly retrieve data as long as you have the right key.

Answer 2: Answer code:

```
tup = (3, 56, 23, 4, 55, 7, 1, 8, 532)
print(tup[:-5])
```

The negative indexing starts from the end of the sequence and is placed in between the elements.

Answer 3: Difference between List and Tuple:

LIST	TUPLE
They are mutable	They are immutable
Errors are more likely to occur in lists	Less likely to occur
Iterations are time-consuming	Iterations are comparatively Faster
Syntax: l = [10, ‘Luck’, 20]	Syntax: t = (10, ‘Luck’, 20)

Answer 4: Output:

```
Dave
KeyError: 'DOB'
```

#The above code will give error because student dictionary do not have element 'DOB'.

Answer 5: Corrected code:

```
student = {‘Student’ : ‘Dave’, ‘Mark’ : 100}
# Get the element from dictionary if exist.
print(student.get(‘Student’))
print(student.get(‘Mark’))
```

LONG ANSWER QUESTIONS

Answer 1: Answer code:

```
#Create a set
s = set([5, 10, 3, 15, 2, 20])
#Find the length use len()
print(len(s))
```

```
#Find maximum value
print(max(s))
#Find minimum value
print(min(s))
# Iterate over a set
for n in s:
  print(n)
#Output:
6
20
2
2
3
5
10
15
20
>
```

Answer 2: Answer code:

```
def test(l):
    result = map(sum, l)
    return list(result)
# sum of elements in each tuple of list of tuples
nums = [(1,5), (4,3), (2,4), (9,3)]
print("The original list of tuples:")
print(nums)
print("\n Total of all the elements of each tuple filledin the said list of tuples:")
print(test(nums))
#Output:
The original list of tuples:
[(1, 5), (4, 3), (2, 4), (9, 3)]
```

Total of all the elements of each tuple filled in the said list of tuples:

[6, 7, 6, 12]

>

Answer 3: Answer code:

```
def test(d):
   res = list(map(list, d.items()))
   return res
# Converting first dictionary
color_d = {1 : 'blue', 2 : 'green', 3 : 'brown', 4 : 'pink', 5 : 'black'}
print("\nOriginal Dictionary:")
print(color_d)
print("Converted dictionary into a list of lists:")
print(test(color_d))
# Converting second dictionary
color_d = {'10' : 'Austin', '22' : 'Natasha', '34' : 'Alfred', '54' : 'Jamie'}
print("\nOriginal Dictionary:")
print(color_d)
print("Converted dictionary into a list of lists:")
print(test(color_d))
#Output:
Original Dictionary:
{1: 'blue', 2: 'green', 3: 'brown', 4: 'pink', 5: 'black'}
Converted dictionary into a list of lists:
[[1, 'blue'], [2, 'green'], [3, 'brown'], [4, 'pink'], [5, 'black']]
Original Dictionary:
{'10': 'Austin', '22': 'Natasha', '34': 'Alfred', '54': 'Jamie'}
Converted dictionary into a list of lists:
[['10', 'Austin'], ['22', 'Natasha'], ['34', 'Alfred'], ['54', 'Jamie']]
```

Answer 4: Answer code:

```
# Initializing dictionary
area = {}
```

```
# loop through numbers from 1 to 4
for x in range(1, 5):
area[x] = x*x # Find each area and store in the dictionary
print(area) # Print the area

#output
{1: 1, 2: 4, 3: 9, 4: 16}
```

Answer 5:

```
word='programming'
# Convert the word into set
unique_alphabets=set(word)
# Find the length of the set
length_of_set=len(unique_alphabets)
print('Unique alphabets:',length_of_set)
```

9. SUGGESTED BOOKS AND E-REFERENCES

BOOKS:

- Narasimha Karumanchi(2015), Data Structure and Algorithmic Thinking with Python. 1st edn. Career Monk
- Dr. BasantAgarwal, Benjamin Baka (2018) Hands-On Data Structures and Algorithms with Python. 2nd edn. Packt Publishing

E-REFERENCES:

- "Python 3" viewed on 28 March 2021, <https://www.programiz.com/python-programming/tutorial>
- Dictionaries in Python, viewed on 28 March 2021, <https://www.studytonight.com/python/dictionaries-in-python>
- Common Python Data Structures, viewed on 28 March 2021, <https://realpython.com/python-data-structures/#dictionaries-maps-and-hash-tables>