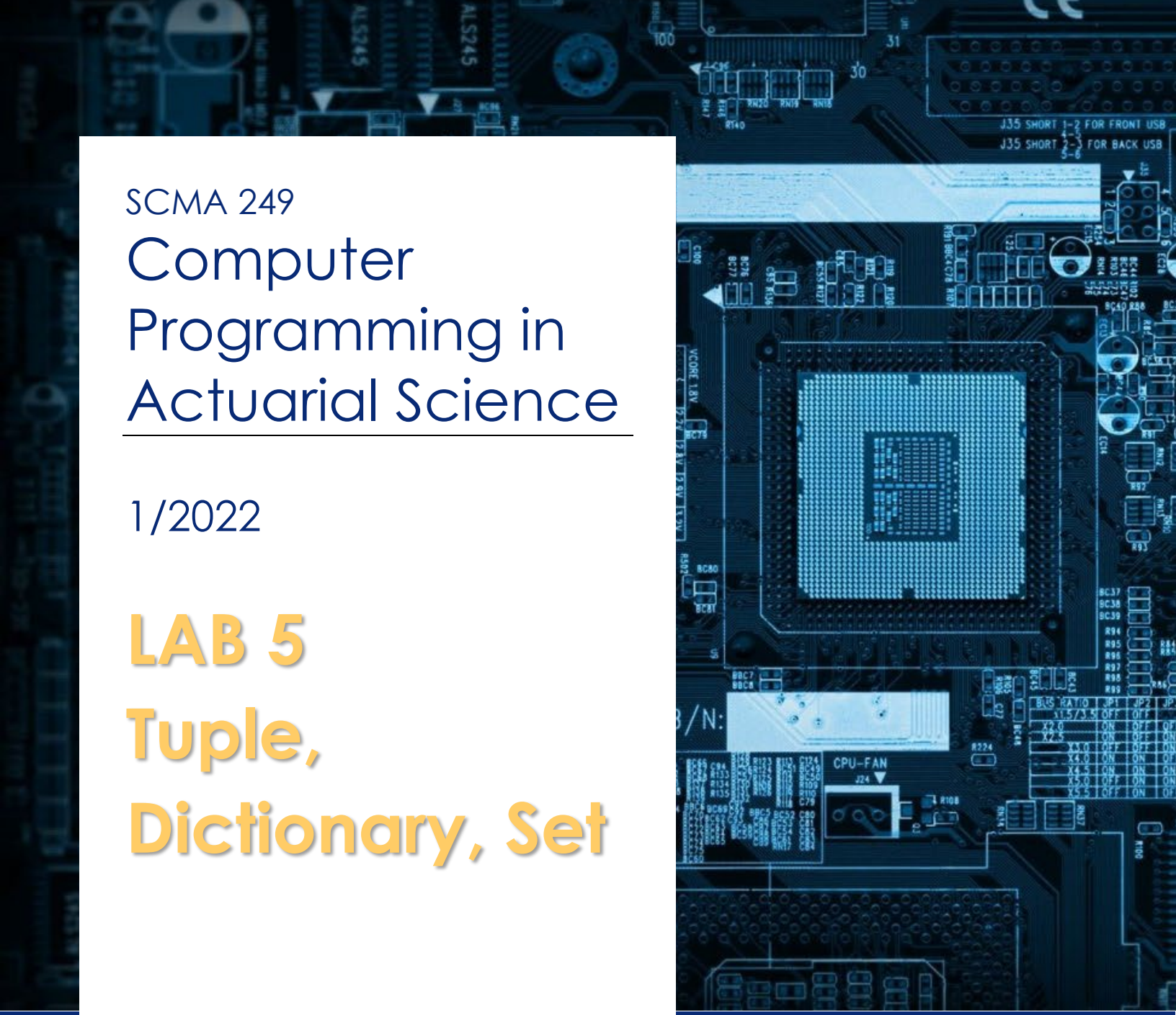

SCMA 249

Computer Programming in Actuarial Science

1/2022

LAB 5
Tuple, Dictionary, Set

LAB 5
Tuple, Dictionary, Set

Strings are a data type in Python for dealing with text. Python has a number of powerful features for manipulating strings.

5.1 Tuple ()

Another collection of Python objects is **tuple**. Tuple is similar to list in terms of indexing, nested objects and repetition.

Question: What is a difference between a tuple and a list?
Answer: The difference between these two things is that **a tuple is immutable** whereas **a list is mutable**. Particularly, the elements in the tuple cannot be changed once it is assigned, in contrast, the elements in the list can be changed.

5.1.1 Creating Tuple

To create a tuple, a sequence of Python objects separating by commas is placed inside the parentheses (). Note that the parentheses are optional.

Example 5.1

	Input	Output
5.1.1	`tuple1 = ()` `print(tuple1)`	()
5.1.2	`tuple2 = (2,4,9)` `print(tuple2)`	(2, 4, 9)
5.1.3	`tuple3 = ('SCMA',[2,4,9])` `print(tuple3)`	('SCMA',[2, 4, 9])
5.1.4	`tuple4 = 'SCMA',[2,4,9]` `print(tuple4)`	('SCMA',[2, 4, 9])

5.1.5	`a, b = tuple4`
	`print(a)`
	`print(b)`
		SCMA
		[2,4,9]

Having one element within parentheses is not enough. We will need a trailing comma to indicate that it is, in fact, a tuple.

Example 5.2

```
mytuple = ('Computer Programming')
print(type(mytuple)) ⟶ str (string)
mytuple = ('Computer Programming',)
print(type(mytuple)) ⟶ tuple
```

5.1.2 Accessing Tuple

Accessing tuple element, i.e., indexing and slicing, method is similar to accessing list element discussed in LAB 4.

Example 5.3 the same as string

	Input	Output
5.3.1	`tuple5 = ('S','C','M','A',' ','2','4','9')` `print(tuple5)`	
5.3.2	`tuple5[0]`	
5.3.3	`tuple5[5:8]`	
5.3.4	`tuple5[-1]`	
5.3.5	`tuple5[-3:]`	

Example 5.4 List of tuple

	Input	Output
5.4.1	`l1 = [('A','S'),(2,0,1,9),('SC',)]` `print(l1)`	0 1 2 [('A','S'),(2,0,1,9),('SC',)]
5.4.2	`l1[1]`	1.0 1.1 1.2 1.3 (2,0,1,9)
5.4.3	`l1[1][3]`	9
5.4.4	`l1[1][3] = 8` ↳ assigning value	Error

Example 5.5 Tuple of list

	Input	Output
5.5.1	`t1 = (['A','S'],[2,0,1,9],['SC'])` `print(t1)`	(['A','S'],[2,0,1,9],['SC'])
5.5.2	`t1[1]`	(2,0,1,9)
5.5.3	`t1[1][3]`	9
5.5.4	`t1[1][3] = 8` `print(t1)`	(['A','S'],[2,0,1,8],['SC'])

We can use + operator to combine two tuples. This is also called **concatenation**. We can also **repeat** the elements in a tuple for a given number of times using the * operator. Both + and * operations result in a new tuple.

Example 5.6

```
#Concatenaion
print(('P','y')+('t','h','o','n'))
#Repeat
print(('P','y')+('t','h','o','n')*3)
print((('P','y')+('t','h','o','n'))*3)
```

5.1.3 Deleting Tuple

We cannot delete or remove the element from a tuple. However, deleting a tuple entirely is possible using the keyword `del`.

Example 5.7

```
del t1
print(t1)
```

5.1.4 Tuple Methods and Operations

Methods that add items or remove items are not available with tuple. **Only the following two methods are available.**

Example 5.8

Create a tuple `t1 = ('s','u','n','t','a','r','e','e')`.

	Input	Output
5.8.1	`#count(x)-Return the number of items x` `print(t1.count('e'))`	2
5.8.2	`#index(x)-Return the index of the first items that is equal to x` `print(t1.index('n'))`	2
5.8.3	`#Membership test-test if an item exist in a tuple or not, using the word "in"` `print('e' in t1)`	True
5.8.4	`print('i' in t1)`	False

5.8.5	`#Iterating through a tuple`
	`for name in ('Atom','Beam','Pink'):`
	`  print('Hello',name,'!')`
		Hello Atom !
		Hello Beam !
		Hello Pink !

5.1.5 Advantages of Tuple over List

Since tuples are quite similar to lists, both of them are used in similar situations as well. However, there are certain advantages of implementing a tuple over a list. Below listed are some of the main advantages:

- We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
- Since tuples are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
- Tuples that contain immutable elements can be used as a key for a dictionary. With lists, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

5.2 Dictionary {}

A Python dictionary is an unordered collection of items. It is more general version of a list, therefore it is changeable and indexed.

5.2.1 Creating Dictionary

Creating a dictionary is as simple as placing items inside **curly braces {}** separated by comma. Each item has a key and the corresponding value expressed as a pair separated by a colon :, i.e., **key:value**. Values can be of any data type and can repeat while keys, act like index, must be of immutable type and must be unique. Keys are often strings, however they can be integers, floats, and many other things.

Example 5.9

Here is a dictionary of the days in the months of the year:

```
days = {'JAN':31, 'FEB':28, 'MAR':31, 'APR':30,
        'MAY':31, 'JUN':30, 'JUL':31, 'AUG':31,
        'SEP':30, 'OCT':31, 'NOV':30, 'DEC':31}
print(days)
```

Note that the order of items in a dictionary will not necessarily be the order in which put them into the dictionary. Internally, Python rearranges things in a dictionary in order to optimize performance.

Example 5.10

```
mydict = {}
mydict['pi'] = 3.14
mydict['22/7'] = 3.14
mydict[2.99] = [2,9,9]
mydict[(2,4,9)] = 249
print(mydict)
```

5.2.2 Working with Dictionary

Accessing

To access the element in dictionary, we use keys.

Example 5.11

	Input	Output
5.11.1	`days['JAN']`	31
5.11.2	`mydict['pi']`	3.14
5.11.3	`mydict[(2,4,9)]`	249
5.11.4	`mydict[22/7]`	Keyerror : 3.142857142857143
5.11.5	`mydict.get(22/7)`	

Changing or Adding Element

Since dictionary is similar to list, hence dictionary is mutable. We can add new items (Example 5.10) or change the value of existing items using assignment operator.

Example 5.12

	Input	Output

5.12.1	`#Add the element`
	`mydict['ADD'] = 'add the element to dict'`
	`print(mydict)`	

5.12.2	`#Change the element`
	`mydict['pi'] = 22/7`
	`print(mydict)`	

Deleting Element

Similar to delete the element in tuple, we use **`del`** to delete the element in dictionary.

Example 5.13

	Input	Output

5.13.1	`#Delete the element we added in Example 5.12.1 from mydict`
	`del mydict['ADD']`
	`print(mydict)`	

5.13.2	`#Delete a dictionary mydict`
	`del mydict`
	`print(mydict)`	

Example 5.14

Create a dictionary to collect the personal information by using a student ID as a key.

```
n = int(input('Enter number of student:'))
slist = {}
for i in range(0,n):
    print('Please enter the information of student ', i+1)
    sid = str(input('Enter student ID:'))
    sname = input('Enter name:')
    slast = input('Enter lastname:')
    snick = input('Enter nickname:')
    stel = str(input('Enter phone number:'))
    smail = input('Enter e-mail address:')
    slist[sid] = [sname, slast, snick, stel, smail]
    print('Thank you!\n')

print(slist)
```

Dictionary Methods

Methods that are available with dictionary are tabulated below.

Python Dictionary Methods

Method	Description
clear()	Remove all items form the dictionary.
copy()	Return a shallow copy of the dictionary.
fromkeys(`seq` [, `v`])	Return a new dictionary with keys from `seq` and value equal to `v` (defaults to `None`).
get(`key` [, `d`])	Return the value of `key`. If `key` doesnot exit, return `d` (defaults to `None`).
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.
pop(`key` [, `d`])	Remove the item with `key` and return its value or `d` if `key` is not found. If `d` is not provided and `key` is not found, raises `KeyError`.
popitem()	Remove and return an arbitary item (key, value). Raises `KeyError` if the dictionary is empty.
setdefault(`key` [, `d`])	If `key` is in the dictionary, return its value. If not, insert `key` with a value of `d` and return `d` (defaults to `None`).
update([`other`])	Update the dictionary with the key/value pairs from `other`, overwriting existing keys.
values()	Return a new view of the dictionary's values

Source: Programiz

5.3 Set {}

Set in Python is one type of Python data types which works like mathematical set. The elements in a set is enclosed with a curly braces `{}`. Note that the elements in a set must be unique. Python's built-in set type has the following characteristics:

- Sets are unordered.
- Set elements are unique. Duplicate elements are not allowed.
- A set itself may be modified, but the elements contained in the set must be of an immutable type.

5.3.1 Creating Set

There are 2 options for creating a set which are enclose all element with curly braces `{}` as mentioned above or use the `set` function. See the example below. **Recall that curly braces are also used to denote dictionaries, and `{}` is the empty dictionary. To get the empty set, use the set function with no arguments, i.e., `set()`.**

Example 5.15

	Input	Output
5.15.1	`s1 = {1,2,3}` `print(s1)`	{1,2,3}
5.15.2	`s2 = {'t','h','r','e','e'}` `print(s2)`	{'h','t','r','e'}
5.15.3	`s3 = set({1,2,'t','h'})` `print(s3)`	{1,2,'h','t'}
5.15.4	`s4 = set([1,4,4,4,5,1])` `print(s4)`	{1,4,5}
5.15.5	`set('This is a test.')`	-

Notice that Python will store the data in a set in whatever order it wants to, not necessarily the order you specify. It's the data in the set that matters, not the order of the data. This means that indexing has no meaning for sets.

5.3.2 Working with Set

Since set in Python is similar to a mathematical set, therefore we can use the following operators.

Example 5.16

Define two sets as `X = {3,4,5}` and `y = {1,2,3,4,5}`.

Operator	Symbol	Method	Description
Union	`x | y`	`x.union(y)`	Returns a set which is the union of sets `x` and `y`.
Intersection	`x & y`	`x.intersection(y)`	Returns a set which is the intersection of sets `x` and `y`.
Difference	`x - y`	`x.difference(y)`	Returns the set difference of `x` and `y` (the elements included in `x`, but not included in `y`).
Symmetric Difference	`x ^ y`		Returns the symmetric difference of sets `x` and `y` (the elements belonging to either `x` or `y`, but not to both sets simultaneously). Mathematically, $(x \cup y) - (x \cap y)$.
Subset	`x <= y`	`x.issubset(y)`	Returns true if `x` is a subset of `y`.
Check Element	`t in y`		Check if **`t`** is an element of `y`.

SUMMARY of list, tuple, dict, and set

	list	tuple	dict	Set
Description	• Ordered • Mutable	• Ordered • Immutable	• Unordered • Mutable • Pair element, i.e., **`key:value`** • Unique **`key`**	• Unordered • Mutable • Unique element
Creating	Square brackets **`[]`** `list()`	Parentheses **`()`** `tuple()`	Curly brackets **`{}`** `dict()`	Curly brackets **`{}`** `set()`
Usage	Use lists if you have collection of data that doesn't need random access.	Use tuples when your data cannot change. A tuple is used in combination with a dictionary.	• When you need a logical association between `key:value` pair • When you need fast lookup for your data, based on a custom key	• Membership testing and the elimination of duplicate entries • When you need uniqueness for the elements
Data type of element	All types	All types	• **`Keys`** can be **`int`**, **`float`**, **`str`**, **`tuple`**, **`bool`** • **`Values`** can be any type	**`int`**, **`float`**, **`str`**, **`tuple`**, **`bool`**

Accessing and Indexing	Indexed by using integers
	`l[i]`
	`l[a:b:step]`
		Indexed by using integers
		`t[i]`
		`t[a:b:step]`
			• Indexed by using keys
			`d[key]`
			• `Cannot be sliced`
				• Can be indexed
				• Cannot be sliced

Iteration	`for e in l` Left-to-right index order	`for e in t` Left-to-right index order	• `for e in d` • `for e in d.keys()` • `for e in d.values()` • `for k,v in d.items()` Return object with random index	`for e in s` Return object with random index

References

- Kittipon P, Kittipob P, Somchai P, Sukree S. Python 101. V1.0.2., Chulalongkorn University Printing House; 2018.
- Andrew J. Python: The Ultimate Beginners Guide!., CreateSpace Independent Publishing Platform; 2016.
- Brian H. A Practical Introduction to Python Programming., Crative Commons Attribution-Noncommercial-Share Alike 3.0; 2015.
- www.geeksforgeeks.org/ (accessed on September 16, 2019)
- www.programiz.com/ (accessed on September 16, 2019)
- www.realpython.com/ (accessed on September 16, 2019)
- www.slideshare.net/rampalliraj/learn-python-for-beginners-part2 (accessed on September 16, 2019)