COMP2152 LAB MANUAL

OPEN SOURCE DEVELOPMENT

This booklet will help the reader understand the concepts, principles, and implementation of the Python programming language. By the end of the booklet, the reader will be able to code comfortably in Python.

Prepared by Ben Blanc

TABLE OF CONTENTS

CHAPTER 9

WORKING WITH DICTIONARIES

A dictionary is an unordered list key-value list. In other programming languages, they are referred to as associative arrays.

CREATING DICTIONARIES

EMPTY DICTIONARY DECLARATION

Takes the following syntax:

VARIABLE_*NAME* = {}

Below are examples of empty dictionaries

```
fruits = {}

colors = {}

cars = {}
```

DICTIONARY WITH VALUES DECLARATION

Takes the following syntax:

VARIABLE_*NAME* = { 'name1' : value1, 'name2' : value2, 'nameN' : valueN
}

```
fruits = {
    'red' : 'apple',
    'blue' : 'berries',
    'orange' : 'tangerine'
}

colors = {
    'fav' : 'red',
    'least' : 'blue',
    False : 0
}

cars = {
    'truck' : 'F-150',
    'compact' : 'civic',
    12 : 'twelve'
}
```

CHAPTER 9 AT A GLANCE

In this chapter you will learn how to code dictionaries, which are a series of data stored in one variable as a key-value pair. You will learn the syntax of creating, accessing and modifying dictionary.

To output the dictionary size, use the len() method.

```
print(len(fruits))
print(len(colors))
print(len(cars))
```

ACCESSING DICTIONARY ELEMENTS

The syntax for accessing an dictionary element is:

Dictionary_Name[key_name]

```
print(fruits['red'])
print(colors[False])
print(cars[12])
```

If you select an index that is not yet declared, you will not get a syntax error but when you run the program, you will get a KeyError

```
print(cars[True])
```

Results in

```
KeyError: True
```

UPDATING DICTIONARY ELEMENTS

The syntax for updating an dictionary element is very similar to accessing an dictionary element, but with the presence of assigning a value to the specified index

Dictionary_Name[key_name] = NEW_VALUE

```
fruits['red'] = 'strawberry'

colors['fav'] = 'orange'

cars['compact'] = 'accent'
```

REMOVING DICTIONARY ELEMENTS

To remove an dictionary element, use the del keyword and pass the dictionary variable and the index number

del Dictionary_Name[key_name]

```
del fruits['orange']
```

DELETING A DICTIONARY

To remove a dictionary, use the del keyword and pass the dictionary variable

del Dictionary_Name

```
del fruits
```

TYPECASTING TO A DICTIONARY

You can typecast a two-column multi-dimensional collection of items by using the dict() method

Dictionary_Name = dict(collection_of_items)

```
table = [
    [1, "one"],
    [2, "two"],
]


table_dict = dict(table)
print(table_dict)
```

DISPLAYING ALL DICTIONARY ELEMENTS

To display all of your list elements, you can use the print() method

```
print(fruits)

print(cars)

print(colors)
```

Or you can use a for loop.

As mentioned in Chapter 4, the syntax for the foreach loop is:

for variable_identifier in collection_of_data :

    statement(s)

The variable_indentifier is a variable that will represent each individual value in your collection_of_data. The collection_of_data is your dictionary.

```
for key in fruits :
    print("Key is "+ key)

for key in fruits.keys() :
    print("Key is "+ key)

for value in fruits.values() :
    print("Value is "+ value)

for key, value in fruits.items() :
    print("Key is {:s}, Value is {:s}".format( key, value))
```

In the for loop, when passing a dictionary, by default ONLY the dictionary keys are passed. You can specify passing the dictionary keys, values, or items (both keys and values) to the for loop.

DICTIONARY METHODS

Below is a table of some of the Dictionary class methods

Name	Description	Arguments	Return	Example

keys()	Get a dictionary view object of keys of the dictionary. You can typecast this view object as a list using the list() method	None	Returns a dictionary view object of keys of the dictionary.	`print(colors.keys())`
				`print(cars.keys())`
				`print(fruits.keys())`
				`print(list(colors.keys()))`
				`print(list(cars.keys()))`
				`print(list(fruits.keys()))`

fromkeys()	Creates a dictionary new dictionary from a list/tuple	1)List/Tuple
		2)default value for key (optional)
			New dictionary	`d1 = dict.fromkeys([2,4])`
				`d2 = dict.fromkeys((7,9))`
				`d3 = dict.fromkeys([2,4]`
				`, "hello")`
				`d4 = dict.fromkeys((7,9)`
				`, "goodbye")`

values()	Get a dictionary view object of key values of the dictionary. You can typecast this view object as a list using the list() method		Returns a dictionary view object of key values of the dictionary.	`print(colors.values())`
				`print(cars.values())`
				`print(fruits.values())`
				`print(list(colors.values()))`
				`print(list(cars.values()))`
				`print(list(fruits.values()))`

Name	Description	Arguments	Return	Example

items()	Get a dictionary view object of key-value pairs of the dictionary. You can typecast this view object as a list using the list() method.		Returns a dictionary view object containing tuple pairs of key-value pairs of key the dictionary.	`print(colors.items())`
				`print(cars.items())`
				`print(fruits.items())`
				`print(list(colors.items()))`
				`print(list(cars.items()))`
				`print(list(fruits.items()))`

get()	Retrieves a value from the dictionary of the specified key	1)object key 2)default value if key not found	Returns object value of specified key	`print(cars.get('van'))`
				`print(cars.get('suv'))`
				`print(colors.get('family'`
				`, 'uh-oh!'))`
				`print(colors.get('fav'`
				`, 'I have none'))`
				`print(fruits.get('green'`
				`, 'no way!'))`

update()	Appends a dictionary to an existing dictionary	Dictionary to append	None. Updates existing dictionary	`colors.update(fruits)` `print(colors)`

setdefault()	Retrieves a value from the dictionary of the specified key. If key not found, key will be set with default value.	1)object key 2)default value if key not found	Returns object value of specified key	`print(cars.setdefault('van'))`
				`print(cars.setdefault('suv'))`
				`print(cars)`
				`print(colors.setdefault('family'`
				`, 'uh-oh!'))`
				`print(colors.setdefault('fav'`
				`, 'I have none'))`
				`print(colors)`
				`print(fruits.setdefault('green'`
				`, 'no way!'))`
				`print(fruits)`

clear()	Removes the dictionary object	None	None	`fruits.clear()`
				`colors.clear()`
				`cars.clear()`

SEARCHING A DICTIONARY

Use the IN operator to determine if a key or value exists in a dictionary

```
if "mykey" in colors.keys():
    print("mykey found in keys!")

if "mykey" in colors.values():
    print("mykey found in values!")
```

TWO-DIMENSIONAL DICTIONARIES

A two-dimensional dictionary can be thought of like a table. Below is a table of 3 rows and 4 columns

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Notice that the first index is always 0.

DECLARATION

Takes the either of the following syntax:

VAR_*NAME* =
{
row1 : {
row1_column1_key: row1_column1_value,
row1_column2_key : row1_column2_value
rowN_columnN_key : rowN_columnN_value

},
row2 : {
row2_column1_key: row2_column1_value,
row2_column2_key : row2_column2_value
rowN_columnN_key : rowN_columnN_value

},
}

Note that the last row of data does not have a comma, however, including one will NOT result in an error.

```
rates = {
    'mortgage' : {
                'home' : 5.5,
                'auto': 3.5,
    },

    'loan': {
        'secured': 7.5,
        'unsecured': 8.5,
    },

}
```

The **len()** method returns the number or rows of the dictionary.

```
print(len(rates))
```

MULTI-DIMENSIONAL DICTIONARIES

You can create many dimensions of a dictionary as long as you can keep track on them.

Takes the either of the following syntax:

VAR_*NAME* =
{
row1 : {
row1_column1_key: {
row1_column1_keys_ row1_column1_key : value,
row1_column1_keys_ row1_column2_key : value,
},

row1_column2_key: {
row1_column2_keys_ row1_column1_key : value,
row1_column2_keys_ row1_column2_key : value,
},

row2 : {
row2_column1_key: {
row2_column1_keys_ row2_column1_key : value,
row2_column1_keys_ row2_column2_key : value,
},

row2_column2_key: {
row2_column2_keys_ row2_column1_key : value,
row2_column2_keys_ row2_column2_key : value,
},

},
}

DECLARATION

```
ToWatch = {
    'educational' : {
        'youth' : {
            'Youth Show 1' : 'Showtime for this tvshow',
            'Youth Show 2': 'Play time for this item',
        },
        'kids' : {
            'barney' : '4pm weekdays',
            'magic school bus' : '8am every day'
        }

    },
    'comedy': {
        'teenage': {
            'Teenage Show 1': 'Friday at 6pm',
            'Double Trouble': 'Saturday and Sundays',
        },
        '17A': {
            'scary movie': '8pm Friday night',
            'documentary': '11am saturday morning'
        }

    },

}
```

ACCESSING MULTI-DIMENSIONAL ELEMENTS

Whether accessing two-dimensional or multi-dimensional dictionary elements, the syntax is:

Dictionary_Name[first_dimension][second_dimension][Nth_dimension]

Take the **rates** dictionary from above

```
rates = {
    'mortgage' : {
                'home' : 5.5,
                'auto': 3.5,
    },

    'loan': {
        'secured': 7.5,
        'unsecured': 8.5,
    },

}
```

You can access the value 5.5 with the following code:

```
print( rates['mortgage']['home'])
```

You can access the value 8.5 with the following code:

```
print( rates['loan']['unsecured'])
```

Take the **ToWatch** dictionary above

```
ToWatch = {
    'educational' : {
        'youth' : {
            'Youth Show 1' : 'Showtime for this tvshow',
            'Youth Show 2': 'Play time for this item',
        },
        'kids' : {
            'barney' : '4pm weekdays',
            'magic school bus' : '8am every day'
        }

    },
    'comedy': {
        'teenage': {
            'Teenage Show 1': 'Friday at 6pm',
            'Double Trouble': 'Saturday and Sundays',
        },
        '17A': {
            'scary movie': '8pm Friday night',
            'documentary': '11am saturday morning'
        }

    },

}
```

You can access the value 4pm weekdays with the following code

```
print(ToWatch['educational']['kids']['barney'])
```

You can access the value 24 with the following code

```
print(ToWatch['comedy']['teenage']['Double Trouble'])
```

If you select an index that is not declared, you will not get a syntax error but when you run the program, you will get a runtime KeyError

```
print( rates['loan']['not set'])
```

```
KeyError: 'not set'
```

Updating Dictionary Elements

The syntax for updating an dictionary element is very similar to accessing an dictionary element, but with the presence of assigning a value to the specified index

Dictionary_Name[first_dimension][second_dimension][Nth_dimension] = NEW_VALUE

```
rates['mortgage']['auto'] = 6.2

rates['loan']['secured'] = 9.7

ToWatch['educational']['youth']['Youth Show 1'] = "No Entry"

ToWatch['comedy']['17A']['documentary'] = "A good one"
```

Displaying All Dictionary Elements

To display all of your dictionary elements, you use the for loop, but be sure to code the loop within the loop

```
for i in rates :
    for j in rates[i] :
        print(i, ":",j, "=", rates[i][j])
```

The more dimensions, the more inner loops

```
for i in ToWatch :
    for j in ToWatch[i] :
        for z in ToWatch[i][j]:
            print(i, ":",j, ":", z, "=", ToWatch[i][j][z])
```