Unit 2: Python Non-Primitive Data Structures

Unit Objectives

In this unit, you will be introduced to Python data structures, their uses, methods, and the basic concepts about how these data structures behave. In addition, you will learn the similarities and differences between them. Upon successful completion of this unit, you should be able to understand the following:

- ☐ Objects and Data Structures
- ☐ Lists
- ☐ List Manipulation Techniques
- ☐ Strings as Lists
- ☐ Tuples
- ☐ Sets
- ☐ Literals

Lesson 1: Objects and Data Structures

Lesson Objectives

In this lesson, you will learn about objects and classes. You will also learn a[illegible]ariables mutability, how to identify built-in objects and how to combine different data types. Up[illegible]mpletion of this lesson, you should be able to understand the following:

- ☐ Combine Different Data Types
- ☐ Python Built-In Objects, Strings, Numbers and Booleans
- ☐ Variables Mutability
- ☐ Structured Built-In Objects

Combine Different Data Types

Each Python data type has properties. These properties are different for each data type. You can combine data types that are of the same type. For example, you can combine a float variable with another float variable. You cannot combine a string with a float.

In the following example, we will combine two float variables.

```
var_S = 3.2
var_F = 4.5
print(type(var_S))
print(type(var_F))
print(var_S + var_F)
```

The following is a line-by-line explanation of the above code:

Code Lines	Description
`var_S = 3.2`	Assigns `var_S` the value of `3.2`.
`var_F = 4.5`	Assigns `var_F` the value of `4.5`.
`print(type(var_S))`	Displays the data type of variable `var_S` which is `float`.
`print(type(var_F))`	Displays the data type of variable `var_F` which is `float`.
`print(var_S + var_F)`	Adds the value of `var_S` and the value of `var_F` and displays the addition result.

The following figure illustrates the above code and its output:

```
var_S = 3.2
var_F = 4.5
print(type(var_S))
print(type(var_F))
print(var_S + var_F)

<class 'float'>
<class 'float'>
7.7
```

Figure 1-1: Combining two float variables

To combine different data types, you need to convert one so that both of them are of the same type. You can convert a string variable to be a float. For example, `var_S` is a string variable, you can use `float(var_S)` to convert it to float variable. Then, you can combine it to `var_F`.

```
var_S = 3.2
var_F = 4.5
print(type(var_S))
print(type(var_F))
print(float(var_S) + var_F)
```

The following is a line-by-line explanation of the above code:

Code Lines	Description
`var_S = "3.2"`	Assigns `var_S` the value of "`3.2`".
`var_F = 4.5`	Assigns `var_F` the value of `4.5`.
`print(type(var_S))`	Displays the data type of variable `var_S` which is
`print(type(var_F))`	Displays the data type of variable `var_F` which is `float`.
`print(float(var_S) + var_F)`	Converts the value of the string variable `va` `.2"` to float 3.2. It adds the result to the value of `var_F` and displa `e` result of the addition.

The following figure illustrates the above code and its output:

```
var_S = "3.2"
var_F = 4.5

print(type(var_S))
print(type(var_F))

print(float(var_S) + var_F)
```

```
<class 'str'>
<class 'float'>
7.7
```

Figure 1-2: Converting data type to properly com ine differ a ty

You may use the same conversion notatio onver er data types. To convert a data type to an integer, use `int(var)`. To convert a data type string use `str(var)`.

The following example shows an incor o com bine a String and a Float variable.

```
var_S = "3.2"
var_F = 4.5
print(type(var_S))
print(type(var_F))
print(var_S + var_
```

The following is a line-b anation of the above code:

Code Lines	Description
`var_S = "3.2`	Assigns `var_S` the value of "`3.2`".
`var_F = 4`	Assigns `var_F` the value of `4.5`.
`print(ty (va S)`	Displays the data type of variable `var_S` which is `string`.
`p c(type(va`	Displays the data type of variable `var_F` which is `float`.
`int var_F)`	Adds the value of `var_S` and the value of `var_F` and displays the addition result.

This code will result in an error since we attempted to combine two different data types. The figure on the following page illustrates the above code and its output:

```
var_S = "3.2"
var_F = 4.5

print(type(var_S))
print(type(var_F))

print(var_S + var_F)
```

```
<class 'str'>
<class 'float'>
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [2], in <cell line: 7>()
      4 print(type(var_S))
      5 print(type(var_F))
----> 7 print(var_S + var_F)

TypeError: can only concatenate str (not "float") to str
```

Figure 1-3: Incorrect way to combine different data types

Learn the Skill

var_S is a String variable with a value of = "3.2", use float(var_S) to convert it to a float variable, adding to it the value 4.5, print the result and the type of th it variables.

Python Built-In Objects: Strings, Numbers, and Boolean

Data in Python is represented by logic ers called ***objects***. In a Python program, all data is stored as objects or as relationships b ween ects. Data variables such as integers, strings, and floats are treated as objects in Pyth

A Python ***Class*** is a blueprint for maki object. It is considered the outline that describes an object. During the execution ogram, stance of a class is created as an object following the specifications of such cl . Eve a type represented by a different class. The class defines the different methods to be sed with is data type and what are their specific purpose.

Let us explore the basi uilt-i s like Numbers, Strings, and Booleans. Previously, we have used the function `type()` t eck the d a type of an object. In this lesson, we will use the function `e()` to check if an object is o tain type. For example:

```
x = 1
print( pe(x
isinst ce (x, nt)
```

Th llowing is a by-line explanation of the above code:

de es	Description
x = 1	Assigns x a value of 1.

`print(type(x))`	Displays the data type of the variable x which is int.
`isinstance (x, int)`	Uses `isinstance()` to check if the data type of the variable is int.

The following figure illustrates the above code and its output:

```
x = 1
print(type(x))
isinstance(x, int)
```

```
<class 'int'>
True
```

Figure 1-4: Identifying object class

You can create your own defined classes and instantiate objects from them as well.

Learn the Skill

Use the function instance to check the type of the object x against the appropriate data type, where x = 1.

Variables Mutability

Mutability can describe the behavior of variables when we assign them to other variables. When a variable is immutable, once a value is linked to this named variable, that value can't be changed. In the following example, we create var_1 and assign the value x. Then, we assign var_1 to var_2.

```
var_1=x
var_2=var_1
```

The behavior of var_2 once assigned depends on its data type if it is mutable or immutable. The following diagrams will illustrate the difference between mutable and immutable data type behaviors.

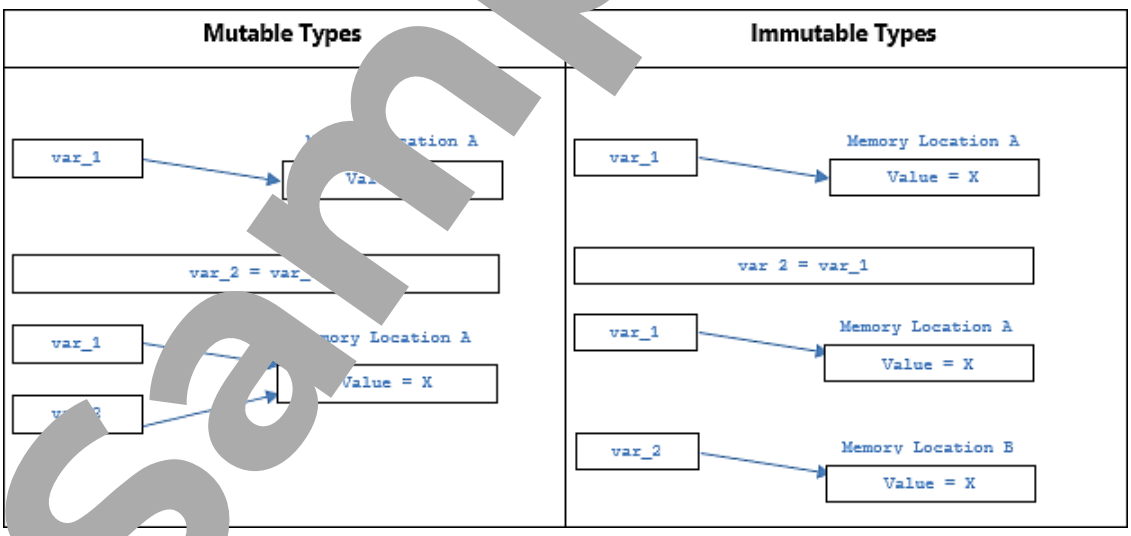

Figure 1-5: Behavior of mutable vs. immutable types

The illustration on the left side explains the behavior of mutable data types when we assign `var_1` to `var_2`. In mutable data types, both variables point to the same memory location. A change in `var_2` value will change `var_1` value as well. For example, changing the value of `var_2` to the following code to `y` will change the value of `var_1` to `y` as well.

```
var_1=x
var_2=var_1
var_2=y
```

The illustration on the right side explains the behavior of immutable data types when we assign the original variable `var_1` to an immutable variable `var_2`. `var_2` points to a copy of the memory location (Memory Location B). A change in `var_2` will not change `var_1` value.

For example, changing the value of `var_2` in the following code to y will not change the value of `var_1` to y. The value of `var_1` will remain x.

```
var_1=x
var_2=var_1
var_2=y
```

Built-in Python types include Integers, Strings, Booleans, Dictionaries, Lists, Tuples, Sets, and Files. They are either Mutable or Immutable types:

Mutable Built-In Types

1. List
2. Sets
3. Dictionaries

Immutable Built-In Types

1. Strings
2. Numbers
3. Booleans
4. Tuples

Learn the Skill

1. When a variable is immutable and a value linked to this named variable, that value can be changed.
    a. True
    b. False
2. List mutable built-in types in Python.
3. List immutable built-in types in Python.

Structured Built-In Objects

Python data structures are repositories that can store and arrange data. Lists, Tuples and Se re the basic Python data structures.

Lists store groups of items in an ordered manner. The order of the items in the will not If you add an item to the list, this item will be appended at the end of the list.

The following code creates a list and display its items.

```
list_A = [3, 4, 5]
print(list_A)
print(type(list_A))
```

The following is a line-by-line explanation of the above code:

Code Lines	Description
`list_A = [3, 4, 5]`	Assigns the values `[3, 4,` to `list_A`.
`print(list_A)`	Displays the content of the lis
`print(type(list_A))`	Displays the data type of `list_A` which is list.

The following figure illustrates the above code and its output:

```
list_A = [3, 4, 5]
print(list_A)
print(type(list_A))
```

```
[3, 4, 5]
<class 'list'>
```

Figure 1-6: Creating a list

Lists allow duplication as items in the t can b epeated.

```
list_B = [3, 4, 4, 5]
print(list_B)
print(type(list_B))
```

The following is a line-by-lin explanat of the above code:

Code Lines	scription
`list_B = [3, 4, 5]`	Assigns the items `[3, 4, 4, 5]` to `list_B`. This list has number 4 repeated twice.
`print(list`	Displays the content of the list.
`print(typ lis B`	Displays the data type of `list_B` which is list.

The following figure illustrates the above code and its output:

```
list_B = [3, 4, 4, 5]
print(list_B)
print(type(list_B))
```

```
[3, 4, 4, 5]
<class 'list'>
```

Figure 1-7: Lists allow duplicate items

A ***Tuple*** store is a group of items stored in an ordered manner. However, Tuples are im able, while lists are mutable. Which means we cannot modify Tuples after creatin Tuples allows duplication as well.

The following code creates a tuple and display its items:

```
tuple_A = (3, 4, 5)
print(tuple_A)
print(type(tuple_A))
```

The following is a line-by-line explanation of the above code:

Code Lines	Description
`tuple_A = (3, 4, 5`	Assigns the items `(3, 4, 5)` to `tuple_A`.
`print(tuple_A)`	Displays the conte tuple.
`print(type(tuple_A))`	Displays the data pe `tu e_A` which is tuple.

The following figure illustrates the above nd it t:

```
tuple_A = (3, 4, 5)
print(tuple_A)
print(type(tuple_A))
```

```
(3, 4, 5)
<class 'tuple'>
```

Figure 1-8: Creating a tuple

Unlike Lists and Tuples, a ***S*** group items that are not ordered, and these items are unique as they cannot be re eated. T llowing code creates a set and display its items.

```
set_A = {"3 ",  "}
print(set_A)
print(type(se
```

The following is by-lin xplanation of the above code:

Code Lines	Description
`set = {"3 "4 ", "  }`	Assigns the items `{"3 ", "4 ", "5 "}` to `set_A`.
`int(  _`	Displays the content of the set.
`(type t_A))`	Displays the data type of `set_A` which is set.

The following figure illustrates the above code and its output:

```
set_A = { "3 ", "4 ", "5 "}
print(set_A)
print(type(set_A))
```

```
{'5 ', '4 ', '3 '}
<class 'set'>
```

Figure 1-9: Creating a set

Learn the Skill

1. Create a list in Python and print its type, use the values 3,4,5 for its items
2. Create a tuple in Python and print its type, use the values 3,4,5 for its it
3. Create a set in Python and print its type, use the values 3,4,5 s.

Lesson Summary

In this lesson, you learned about objects and classes. You also learned about variables mutability, how to identify built-in objects and how to combine different data types. You should now understand the following:

- ☑ Combine Different Data Types
- ☑ Python Built-In Objects, Strings, Numbers and Booleans
- ☑ Variables Mutability
- ☑ Structured Built-In Objects

Practice Exercise

Input the following code, then observe the output.

```
myVar = 321
print(isinstance(myVar, int))
myVar = str(myVar)
print(isinstance(myVar, int))

days = ["Sunday", "Monday", "Tuesday"]
print(type(days))
other_days = days
other_days.append("Wednesday")
print(days)
```

Practice Questions

1. What is the expected output of this code?

```
var_int = 99
var_sum = 100
print("Sum of " + str(var_int) + " +  =  + tr(var_sum))
```

   a. Sum of 99 + 1 = 100
   b. Sum of 100 + 1 = 99
   c. Sum of 99 + 100 = 1

2. Fill in the blank with the correct w d.

   A Python _____ is a bl t for r king a new object. It is considered the outline that describes an object.

3. The output of the follow True.

```
f = 4.4
isinstance (x,
```

   a. True
   b. False

4. Select all th n imm utable built-in types (Select all that apply)

   a. rin
   b. Nu ers
   c. Sets
   e. B eans
   f. ples

5. Given this code:

```
li = [3, 4, 5]
```

Which code will display the type of variable `li`?

a. `len(li)`
b. `print(li)`
c. `print(type(li))`
d. `type(li)`

6. Which line of code defines a tuple variable that contains the following items: `"apple " / "orange " / "banana "`?

a. `fruit = ("apple ", "orange ", "banana ")`
b. `. fruit = ["apple ", "orange ", "banana "]`
c. `fruit = {"apple ", "orange ", "banana`