Q :

Understanding Object Dependencies

Which of the following is the basic building block for any of Python's scientific computing, data science, and general programming libraries we use today?

A :

Python object.

press

Correct! Everything in Python starts as an object.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Ref :

Creating functions

In this exercise, we will review functions, as they are key building blocks of object-oriented programs.

For this, we will create a simple function average\_numbers() which averages a list of numbers. Remember that lists are a basic data type in Python that we can build using the [] bracket notation.

Here is an example of a function that returns the square of an integer:

def square\_function(x):

x\_squared = x\*\*2

return x\_squared

Q :

Create a function average\_numbers(), which takes a list num\_list as input and then returns avg as output.

Inside the function, create a variable, avg, that takes the average of all the numbers in the list.

Call the average\_numbers function on the list [1, 2, 3, 4, 5, 6] and assign the output to the variable my\_avg.

Print out my\_avg.

# Create function that returns the average of an integer list

def average\_numbers(num\_list):

avg = sum(num\_list)/float(len(num\_list)) # divide by length of list

return avg

# Take the average of a list: my\_avg

my\_avg = average\_numbers([1, 2, 3, 4, 5, 6])

# Print out my\_avg

print(my\_avg)

<script.py> output:

3.5

Good job! Now that you have created a function, try playing around by adding your own list.

Ref :

Creating a complex data type

In this exercise, we'll take a closer look at the flexibility of the list data type, by creating a list of lists.

In Python, lists usually look like our list example below, and can be made up of either simple strings, integers, or a combination of both.

list = [1,2]

In creating a list of lists, we're building up to the concept of a NumPy array.

Create a variable called matrix, and assign it the value of a list.

Within the matrix list, include two additional lists: [1,2,3,4] and [5,6,7,8].

Print the matrix list.

# Create a list that contains two lists: matrix

matrix = [[1,2,3,4] , [5,6,7,8]]

# Print the matrix list

print(matrix)

<script.py> output:

[[1, 2, 3, 4], [5, 6, 7, 8]]

Good job! What happens when you nest two lists in a list? How can you access those elements? Try creating a tripply-nested list and finding out.

Ref :

**What are NumPy Arrays most similar to?**

What other Python data structure can one-dimensional NumPy arrays be thought of as similar, but not exactly identical to?

A : Lists.

Press

Correct! One-dimensional NumPy arrays look a lot like lists, and building them is very similar.

Ref :

# Create a function that returns a NumPy array

In this exercise, we'll continue working with the numpy package and our previous structures.

We'll create a NumPy array of the float (numerical) data type so that we can work with a multi-dimensional data objects, much like columns and rows in a spreadsheet.

Q :

* Import numpy as np.
* Declare variable my\_matrix and set it to [[1,2,3,4], [5,6,7,8]].
* Declare a function called return\_array(), which takes a list matrix as input, and returns an array object as output. In the body, declare a variable array set it to np.array(matrix, dtype = float).
* Call return\_array() on the my\_matrix list, and print out the output.

# Import numpy as np

import numpy as np

# List input: my\_matrix

my\_matrix = [[1,2,3,4], [5,6,7,8]]

# Function that converts lists to arrays: return\_array

def return\_array(matrix):

array = np.array(matrix, dtype = float)

return array

# Call return\_array on my\_matrix, and print the output

print(return\_array(my\_matrix))

<script.py> output:

[[1. 2. 3. 4.]

[5. 6. 7. 8.]]

Good job! Now that you have created a NumPy array, take a look at the NumPy docs (by googling) and investigate other data types arrays can take (as arguments).

Ref :

# Creating a class

We're going to be working on building a class, which is a way to organize functions and variables in Python. To start with, let's look at the simplest possible way to create a class.

Q :

* Declare a class called DataShell.
* Our class will not do much: simply include the passstatement in the body of the DataShell class.

# Create a class: DataShell

class DataShell:

pass

Good job! Try creating different classes with different names. See what kind of naming convention Python will accept.

Q :

# Difference between a class and an object

Fill in the blanks: \_\_\_\_ are instances of \_\_\_\_ and can have both variables and functions.

A :

Objects, classes.

Press

# Understanding what we're building

What high-level Python object will our DataShell be most like once we're done finishing building it?

##### Answer the question

**50 XP**

##### Possible Answers

* 

A Pandas dataframe.

press1

* 

A Python method.

press2

* 

A scikit-learn model.

press3

* 

A NumPy Array.

press

That's right! We're trying to build out a Pandas object.

Ref :

# Object: Instance of a Class

As we learned earlier, a class is like a blueprint: we can make many copies of our class.

When we do this, we say that we are instantiating our class. These instances are called objects.

Here is an example of class instantiation:

object\_name = ClassName()

Q :

* Create an empty class called DataShell. Only include the pass statement inside of the class definition.
* Instantiate the DataShell class and assign the newly created object to the my\_data\_shell variable.
* Print my\_data\_shell and explore its contents.

# Create empty class: DataShell

class DataShell:

# Pass statement

pass

# Instantiate DataShell: my\_data\_shell

my\_data\_shell = DataShell()

# Print my\_data\_shell

print(my\_data\_shell)

<script.py> output:

<\_\_main\_\_.DataShell object at 0x7f413b4d30f0>

Good job! Now you have created an instance of the DataShell class. Try creating additional instances with different names!