**FSDS MAY BATCH 2022(Python Basics 18)**

**Submitted by: Shubham Tiwari**

Q1. Create a zoo.py file first. Define the hours() function, which prints the string ‘Open 9-5 daily’. Then, use the interactive interpreter to import the zoo module and call its hours() function.

Ans: An example of how to create the **zoo.py** file with a **hours()** function:

# File: zoo.py

**def hours():**

**print("Open 9-5 daily")**

To import the **zoo** module and call its **hours()** function, we can follow these steps:

1. Open a Python interpreter in our terminal or IDE.
2. Type **import zoo** to import the **zoo** module. Make sure that the **zoo.py** file is in the same directory as our Python interpreter, or else specify the full path to the file.
3. Type **zoo.hours()** to call the **hours()** function from the **zoo** module.

The interactive interpreter will look like:

**import zoo**

**zoo.hours()**

**Open 9-5 daily**

Q2. In the interactive interpreter, import the zoo module as menagerie and call its hours() function.

Ans: To import the **zoo** module as **menagerie** and call its **hours()** function in the interactive interpreter, you can do the following:

1. Open a Python interpreter in our terminal or IDE.
2. Type **import zoo as menagerie** to import the **zoo** module with the alias **menagerie**. Make sure that the **zoo.py** file is in the same directory as our Python interpreter, or else specify the full path to the file.
3. Type **menagerie.hours()** to call the **hours()** function from the **menagerie** module.

The interaction will look like :

**import zoo as menagerie**

**menagerie.hours()**

**Open 9-5 daily**

In step 2, we use the as keyword to give the zoo module an alias menagerie. This is useful if we want to avoid naming conflicts with other modules or if we simply prefer a different name for the module.

Q3. Using the interpreter, explicitly import and call the hours() function from zoo.

Ans: To explicitly import and call the **hours()** function from **zoo** in the Python interpreter, we can do the following:

1. Open a Python interpreter in your terminal or IDE.
2. Type **from zoo import hours** to import only the **hours()** function from the **zoo** module. Make sure that the **zoo.py** file is in the same directory as your Python interpreter, or else specify the full path to the file.
3. Type **hours()** to call the **hours()** function directly.

The interaction will look like:

**from zoo import hours**

**hours()**

**Open 9-5 daily**

In step 2, we use the **from** keyword to import only the **hours()** function from **zoo**. This is useful if we only need a specific function or a few functions from a module, as it saves us from having to type the module name each time we call the function.

Q4. Import the hours() function as info and call it.

Ans: To import the **hours()** function from **zoo** and give it an alias **info** in the Python interpreter, we can do the following:

1. Open a Python interpreter in your terminal or IDE.
2. Type **from zoo import hours as info** to import the **hours()** function from the **zoo** module and give it an alias **info**. Make sure that the **zoo.py** file is in the same directory as your Python interpreter, or else specify the full path to the file.
3. Type **info()** to call the **hours()** function using the **info** alias.

The interaction will look like:

**from zoo import hours as info**

**info()**

**Open 9-5 daily**

Note that in step 2, we use the as keyword to give the hours() function an alias info. This is useful if we want to use a different name for the function or if we're importing multiple functions with similar names from different modules.

Q5. Create a plain dictionary with the key-value pairs ‘a’: 1, ‘b’: 2, and ‘c’: 3, and print it out.

Ans:

The following code will apply into it:

# create the dictionary

**my\_dict = {'a': 1, 'b': 2, 'c': 3}**

# print the dictionary

**print(my\_dict)**

**#Output:**

**{'a': 1, 'b': 2, 'c': 3}**

Here, we create the dictionary using curly braces {} and separating the key-value pairs with a colon :. We assign this dictionary to the variable my\_dict.

Then we use the print() function to display the contents of the my\_dict dictionary. The output shows that the dictionary has three key-value pairs: 'a': 1, 'b': 2, and 'c': 3.

Q6 .Make an OrderedDict called fancy from the same pairs listed in 5 and print it. Did it print in the same order as plain?

Ans:

from collections import OrderedDict

# create the OrderedDict

fancy = OrderedDict([('a', 1), ('b', 2), ('c', 3)])

# print the OrderedDict

print(fancy)

# Output:

OrderedDict([('a', 1), ('b', 2), ('c', 3)])

Here we import the **OrderedDict** class from the **collections** module. Then, we create the **fancy** **OrderedDict** using the same key-value pairs as before, but this time we use a list of tuples to specify the key-value pairs.

Finally, we print the **fancy** **OrderedDict**. The output shows that the **fancy** **OrderedDict** has the same key-value pairs as the original dictionary, and they are printed in the same order as before: 'a': 1, 'b': 2, and 'c': 3. The **OrderedDict** remembers the order in which the keys were inserted, whereas a plain dictionary does not guarantee the order of the keys.

Q7. Make a default dictionary called dict\_of\_lists and pass it the argument list. Make the list dict\_of\_lists[‘a’] and append the value ‘something for a’ to it in one assignment. Print dict\_of\_lists[‘a’].

Ans:

from collections import defaultdict

# create the defaultdict with default value as list

dict\_of\_lists = defaultdict(list)

# append a value to the list assigned to key 'a'

dict\_of\_lists['a'].append('something for a')

# print the value of dict\_of\_lists['a']

print(dict\_of\_lists['a'])

#Output

['something for a']

Here, we first import the **defaultdict** class from the **collections** module. We create the **dict\_of\_lists** defaultdict with a default value of an empty list.

Next, we append a value to the list assigned to the key 'a' using the list append() method.

Finally, we print the value of **dict\_of\_lists['a']**, which outputs the list **['something for a']**.

If we were to add another value to the same key 'a', it would be appended to the same list without overwriting the previous value:

**dict\_of\_lists['a'].append('another thing for a')**

**print(dict\_of\_lists['a'])**

#Output

['something for a', 'another thing for a']