The memoryview () function in Python

memoryview () in Python returns the memory view of the argument.

Before understanding what a memory view is, we need to talk a little bit about Python's protocol buffers.

What is Python's buffer protocol?

The buffer protocol provides a way to access the internal data of an object. This internal data can be a memory array or buffer.

The buffer protocol allows one object to expose data (buffer) inside it and another object can access those buffers without the need for intermediate replication.

This protocol is only accessible to data at C-API level and does not use the normal code base. Therefore, to display data in the same protocol as a regular Python code base, we need a memory view.

What is memory view?

Memory view is a safe way to display buffer protocols in Python. It allows you to access the buffer inside an object by creating a memory view object.

Why are buffer protocol and memory view so important in Python?

Whenever we perform actions on an object (executing a function on an object, cutting an array .), Python creates a copy of the object. This will consume memory, slowing down processing if we work with large amounts of data.

memoryview () is an important function of Python

By using the buffer protocol, we can allow an access object to use / modify data without having to copy an extra copy. This will help the program use less memory and speed up the code processing.

The memoryview function syntax ()

To display the protocol buffer using **memoryview ()** , we use the following syntax:

```
memoryview()
```

Memoryview () function parameter

The **memoryview ()** function has only one parameter:

1. Object: The object containing the data you want to use **memoryview ()** to access. This object must support protocol buffers (bytes, bytearray).

The value returned from the memoryview () function

The **memoryview ()** function returns the memory view of the object.

Example 1: How does memoryview () function work in Python?

```
#random bytearray random_byte_array = bytearray('TIPSMAKE', 'utf-8') mv = memory
```

When running the program, the result returned is:

```
81 b'QU' [81, 85, 65, 78, 84, 82, 73, 77, 65, 78, 71]
```

Here, we create a **mv** memory view from the **random_byte_array** byte array .

Then we access the index at position 0 of mv, the value of this index is Q. This index is printed according to the ASCII value of 81.

Next, we access the 0 and 1 position indexes of **mv** , QU, and convert them into bytes. Finally, we access all of the **mv** 's indexes and convert them into an ASCII list.

Example 2: Editing data with memoryview ()

```
# random bytearray random_byte_array = bytearray('QTIPSMAKE', 'utf-8') print('be
```

When running the program, the result is:

```
before fixing: bytearray(b'QTIPSMAKE') after: bytearray(b'TIPSMAKE')
```

Here, we update the number 1 position index of **mv** from V to U, 85 being U.'s ASCII code.

Because the memory view **mv** object references the same buffer / memory, updating the index in **mv** also updates **random_byte_array** .

tips and guides. Thank you for reading and for following us regularly.