EUROPYTHON PRAGUE & REMOTE 17-23 July 2023

Zero-Copy Zen

Boost Performance with Memory View

EUROPYTHON PRAGUE & REMOTE
17-23 July 2023

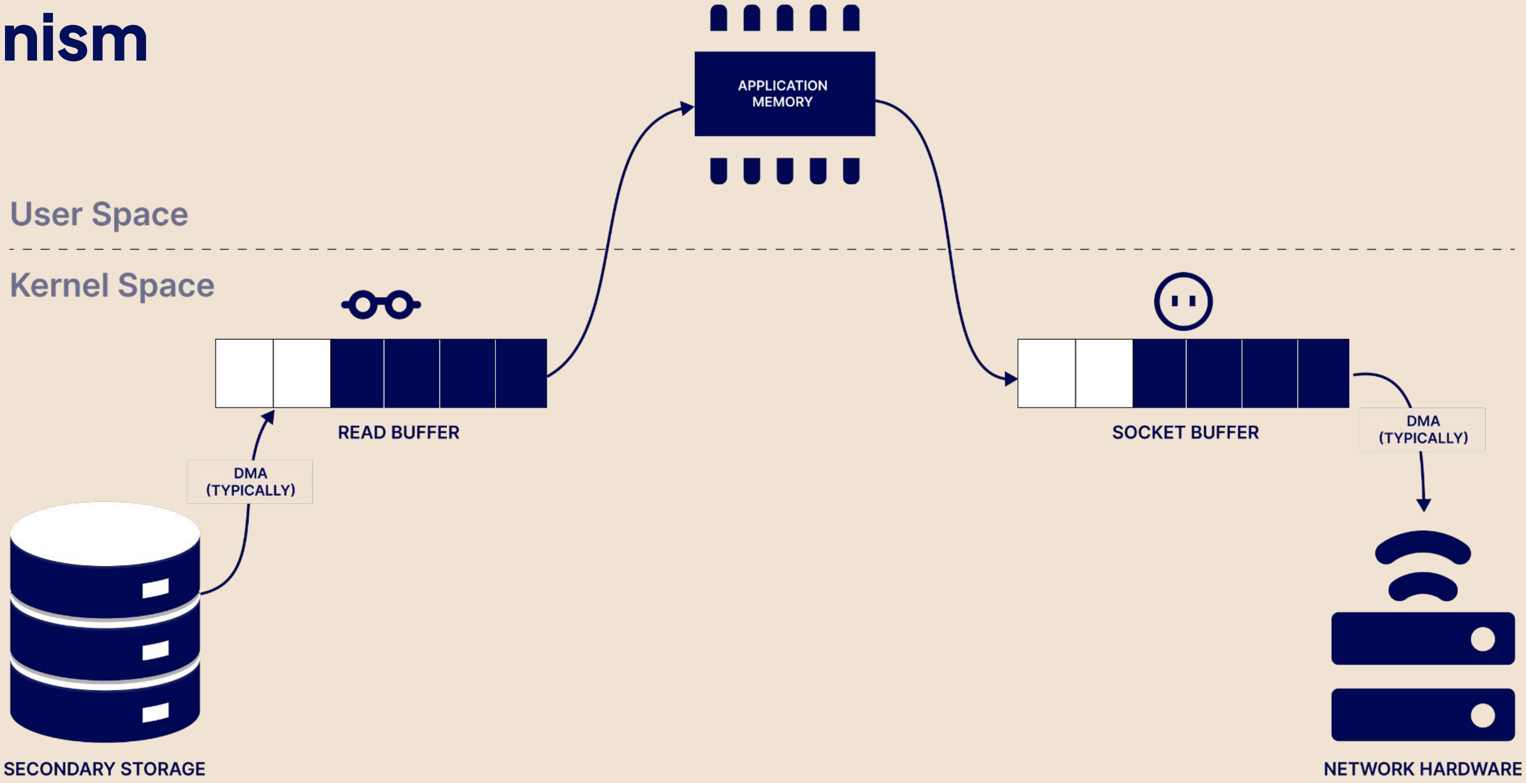
Kesia Mary Joies
Product Engineer
UST | Strollby

Aby M Joseph
Product Engineer
UST | Strollby

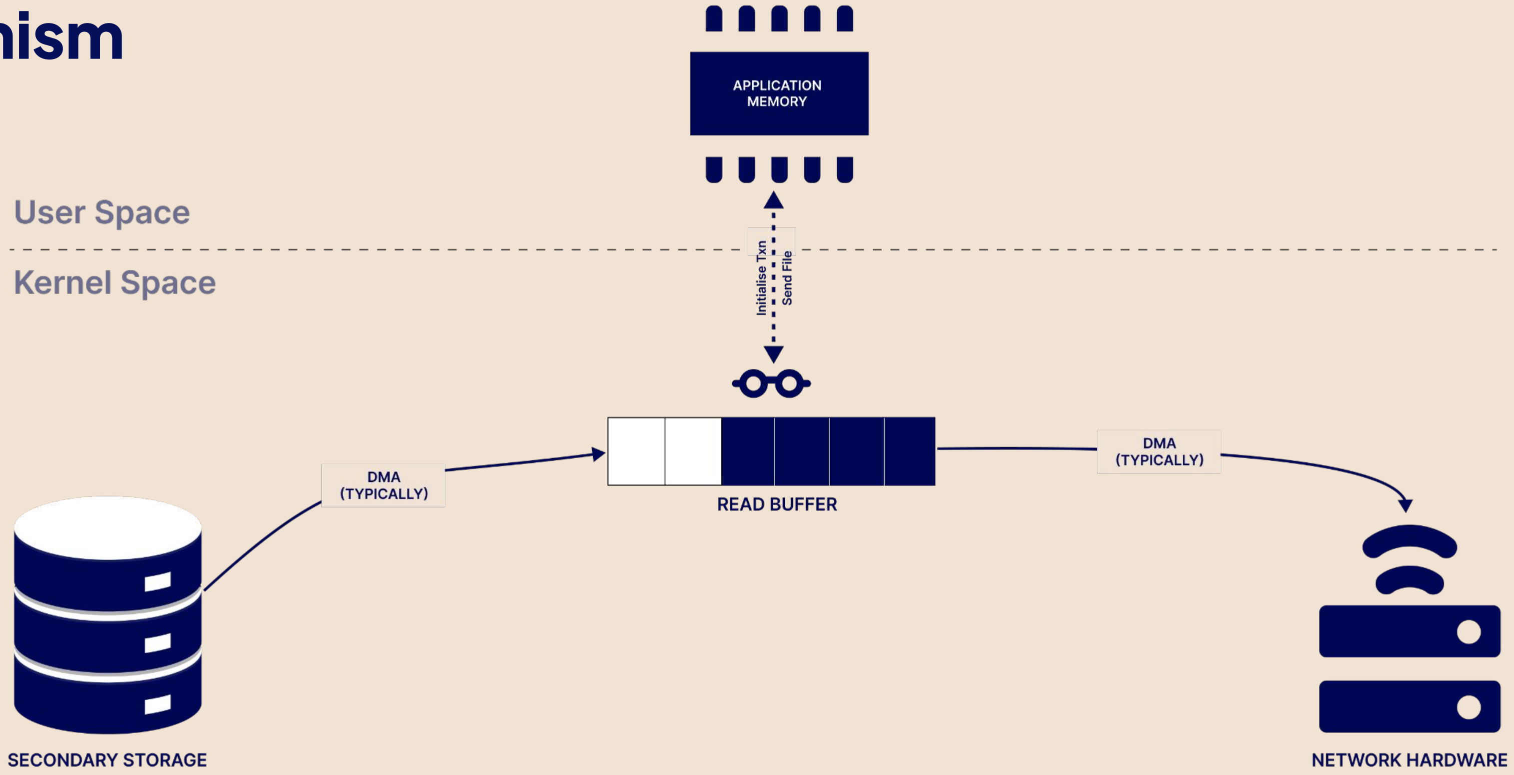
Zero-copy

A method to copy data from the disk/network to the memory without passing through the CPU

Traditional Mechanism



Zero-copy Mechanism



Let’s look into an example

```python
import socket
import hashlib

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_port = 8082
server_addr = ("0.0.0.0", server_port)
print(f"Start server on port {server_port}")
sock.bind(server_addr)
sock.listen(1)

while True:
    print("Waiting for connection")
    connection, client_addr = sock.accept()
    size = 0
    try:
        i = 0
        while True:
            data = connection.recv(65536)
            i += 1
            if data:
                size += len(data)
            else:
                print("Done receiving data")
                break
        print(f"Total size: {size}")
    finally:
        connection.close()
```

Receiving server

```python
import socket
import time

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_port = 8082
server_address = ('127.0.0.1', server_port)
sock.connect(server_address)

start = time.time()
try:
    with open(r'/tmp/large_file', 'rb') as f:
        message = f.read()
        sock.sendall(message)
finally:
    sock.close()

end = time.time()
print('Total time: ', end-start)
```

Traditional Copy Client

```python
import socket
import time

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_port = 8082
server_address = ('127.0.0.1', server_port)
sock.connect(server_address)

start = time.time()
try:
    with open(r'/tmp/large_file', 'rb') as f:
        message = f.read()
        sock.sendall(message)
finally:
    sock.close()

end = time.time()
print('Total time: ', end-start)
```

```
"""
# Start server
$ python zerocopy_server.py
Start server on port 8082
Waiting for connection
Done receiving data
Total size: 2147483648
Waiting for connection
$ truncate -s 2G /tmp/large_file
$ python traditional_client.py
Total time:  2.4229979515075684
"""
```

A test run using a test file with 2GB size

```python
import os
import socket
import time

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_port = 8082
server_address = ("127.0.0.1", server_port)
sock.connect(server_address)
start = time.time()

try:
    with open(r"/tmp/large_file", "rb") as f:
        ret = 0
        offset = 0

    while True:
                ret = os.sendfile(sock.fileno(), f.fileno(), offset, 65536)
                offset += ret
                if ret == 0:
                    break
finally:
    sock.close()

end = time.time()
print("Total time: ", end - start)
```

Zero –Copy Client

```python
import os
import socket
import time

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_port = 8082
server_address = ("127.0.0.1", server_port)
sock.connect(server_address)
start = time.time()

try:
    with open(r"/tmp/large_file", "rb") as f:
        ret = 0
        offset = 0

    while True:
            ret = os.sendfile(sock.fileno(), f.fileno(), offset, 65536)
            offset += ret
            if ret == 0:
                break
finally:
    sock.close()

end = time.time()
print("Total time: ", end - start)
```

```
"""
# Server output
Waiting for connection
Done receiving data
Total size: 2147483648
Waiting for connection
...
# Zero copy client
$ python zerocopy_client.py
Total time:  1.161919116973877
"""
```

In Python, we can use Zero-copy by using memoryview

Python’s built-in types to manipulate binary data

Python’s built-in types to manipulate binary data

Bytes

Python’s built-in types to manipulate binary data

Byte Array

Bytes

Python's built-in types to manipulate binary data

Byte Array

Bytes

Memory View

Bytes

Immutable data type

Bytes

Return a byte object

Immutable data type

Bytes

Return a byte object

Immutable data type

Must be iterable of integers between 0 <= x < 256

Bytes

Syntax:

bytes([source[, encoding[, errors]]])

bytes.py

```python
data = bytes(4)
print(data)
# b'\x00\x00\x00\x00'

print(type(data))
# <class 'bytes'>
```

```
count = [1, 5, 4, 8, 2]
count_new = bytes(count)
print(count_new)
# b'\x01\x05\x04\x08\x02'
print(type(count_new))
# <class 'bytes'>
```

```python
count = [1, 5, 4, 8, 2]
count_new = bytes(count)
print(count_new)
# b'\x01\x05\x04\x08\x02'
print(type(count_new))
# <class 'bytes'>

print(*count_new)
```

```python
count = [1, 5, 4, 8, 2]
count_new = bytes(count)
print(count_new)
# b'\x01\x05\x04\x08\x02'
print(type(count_new))
# <class 'bytes'>

print(*count_new)
# 1 5 4 8 2
```

```python
count = [1, 5, 4, 8, 2]
count_new = bytes(count)
print(count_new)
# b'\x01\x05\x04\x08\x02'
print(type(count_new))
# <class 'bytes'>

print(*count_new)
# 1 5 4 8 2

count_new[0] = 100
```

```python
count = [1, 5, 4, 8, 2]
count_new = bytes(count)
print(count_new)
# b'\x01\x05\x04\x08\x02'
print(type(count_new))
# <class 'bytes'>

print(*count_new)
# 1 5 4 8 2

count_new[0] = 100
# TypeError: 'bytes' object does not support item assignment
```

Bytearray

Bytearray

Similar to bytes

Bytearray

Supports mutability

Similar to bytes

bytearray.py

```python
data = bytearray(4)
print(data)
# bytearray(b'\x00\x00\x00\x00')

print(type(data))
# <class 'bytearray'>
```

bytearray.py

```
count = [10, 20, 15, 80, 53]
count_new = bytearray(count)
print(type(count_new))
# <class 'bytearray'>

print(*count_new)
# 10 20 15 80 53


count_new[0] = 100
print(*count_new)
# 100 20 15 80 53
```

Memory View

Zero-Copy View

Memory View

Slicing and Indexing

Zero-Copy View

Memory View

Slicing and Indexing

Zero-Copy View

Efficient Data Manipulation

Memory View

Syntax:

memoryview(object)

```
>>> view = memoryview(b'hello world')
>>> view
<memory at 0x0000025D2D26B1C8>
```

```
>>> view = memoryview(b'hello world')
>>> view
<memory at 0x0000025D2D26B1C8>

>>> view[0]
104
```

```
>>> view = memoryview(b'hello world')
>>> view
<memory at 0x0000025D2D26B1C8>

>>> view[0]
104

>>> chr(view[0])
'h'
```

memory_view.py

```python
byte_array = bytearray('EURO', 'utf-8')
mv = memoryview(byte_array)
print(*mv)
# 69 85 82 79
```

Converts to byte array using str.encode()

memory_view.py

```
byte_array = bytearray('EURO', 'utf-8')
mv = memoryview(byte_array)
print(*mv)
# 69 85 82 79

print(type(mv))
# <class 'memoryview'>
```

Memory View

Uses buffer protocol

So, what is
buffer protocol ?

Buffer Protocol

It is a protocol that provides a way to access the internal data of an object.

Whenever we perform some action on an object, Python needs to create a copy of the object.

list_slice.py

```python
my_list = [1, 2, 3, 4, 5]
print(f'{my_list=}')
# my_list=[1, 2, 3, 4, 5]

sliced_list = my_list[1:3]
print(f'{sliced_list=}')
# sliced_list=[2, 3]
```

Buffer Protocol

But the problem is, even if we *cannot access this protocol* with the standard code base, this is accessible to us at the C-API level.

Buffer Protocol

But the problem is, even if we *cannot access this protocol* with the standard code base, this is accessible to us at the C-API level.

So, in Python, if we want to expose the same protocol in Python, we *need to use memoryview*.

Buffer Protocol

Improves execution speed

Buffer Protocol

Use less memory

Improves execution speed

Buffer Protocol

Use less memory

Improves execution speed

Works on large data

PEP 688 – Making the buffer protocol accessible in Python

- A better way to interact with the buffer protocol directly.
- Now supports additional types with the buffer protocol.
- Simplified Workflow
- Performance Boost

Diving into Memory View

memory_view.py

```python
s1 = b"Hello World"
s2 = bytearray(b'Hello World')

s1View = memoryview(s1)
s2View = memoryview(s2)

print("Bytes is readonly?:", s1View.readonly)
# Bytes is readonly?: True
print("Byte array is readonly?:", s2View.readonly)
# Byte array is readonly?: False
```

memory_view.py

```python
# Modifying object through writeable view

s2 = bytearray(b'Hello World!')
s2View = memoryview(s2)

print("Before:", s2)
# Before: bytearray(b'Hello World!')
```

memory_view.py

```python
# Modifying object through writeable view

s2 = bytearray(b'Hello World!')
s2View = memoryview(s2)

print("Before:", s2)
# Before: bytearray(b'Hello World!')

s2View[6:12] = b'Python'
print("After:", s2)
# After: bytearray(b'Hello Python')
```

```python
import array
a = array.array('l', [-11111111, 22222222, -33333333, 44444444])
m = memoryview(a)
print(m[0])
# -11111111
print(m[-1])
# 44444444
print(m[::2].tolist())
# [-11111111, -33333333]
```

Comparison

Without Memory View

```python
import time

print("Type\t\t\tIterations\t\tTime Taken")
print("--------------------------------------------")

for n in (100000, 200000, 300000, 400000, 500000):
    data = b'x'*n
    start = time.time()
    b = data
    while b:
        b = b[1:]
    end = time.time()
    tm = end - start
    print(f'bytes \t\t\t{n}\t\t\t {tm:0.2f}')
```

```python
import time

print("Type\t\t\tIterations\t\tTime Taken")
print("-------------------------------------------")

for n in (100000, 200000, 300000, 400000, 500000):
    data = b'x'*n
    start = time.time()
    b = data
    while b:
        b = b[1:]
    end = time.time()
    tm = end - start
    print(f'bytes \t\t\t{n}\t\t\t {tm:0.2f}')
```

Without Memory View

```python
import time

print("Type\t\t\tIterations\t\tTime Taken")
print("-------------------------------------------")

for n in (100000, 200000, 300000, 400000, 500000):
    data = b'x'*n
    start = time.time()
    b = memoryview(data)
    while b:
        b = b[1:]
    end = time.time()
    tm = end - start
    print(f'memoryview  \t\t{n}\t\t\t {tm:0.2f}')
```

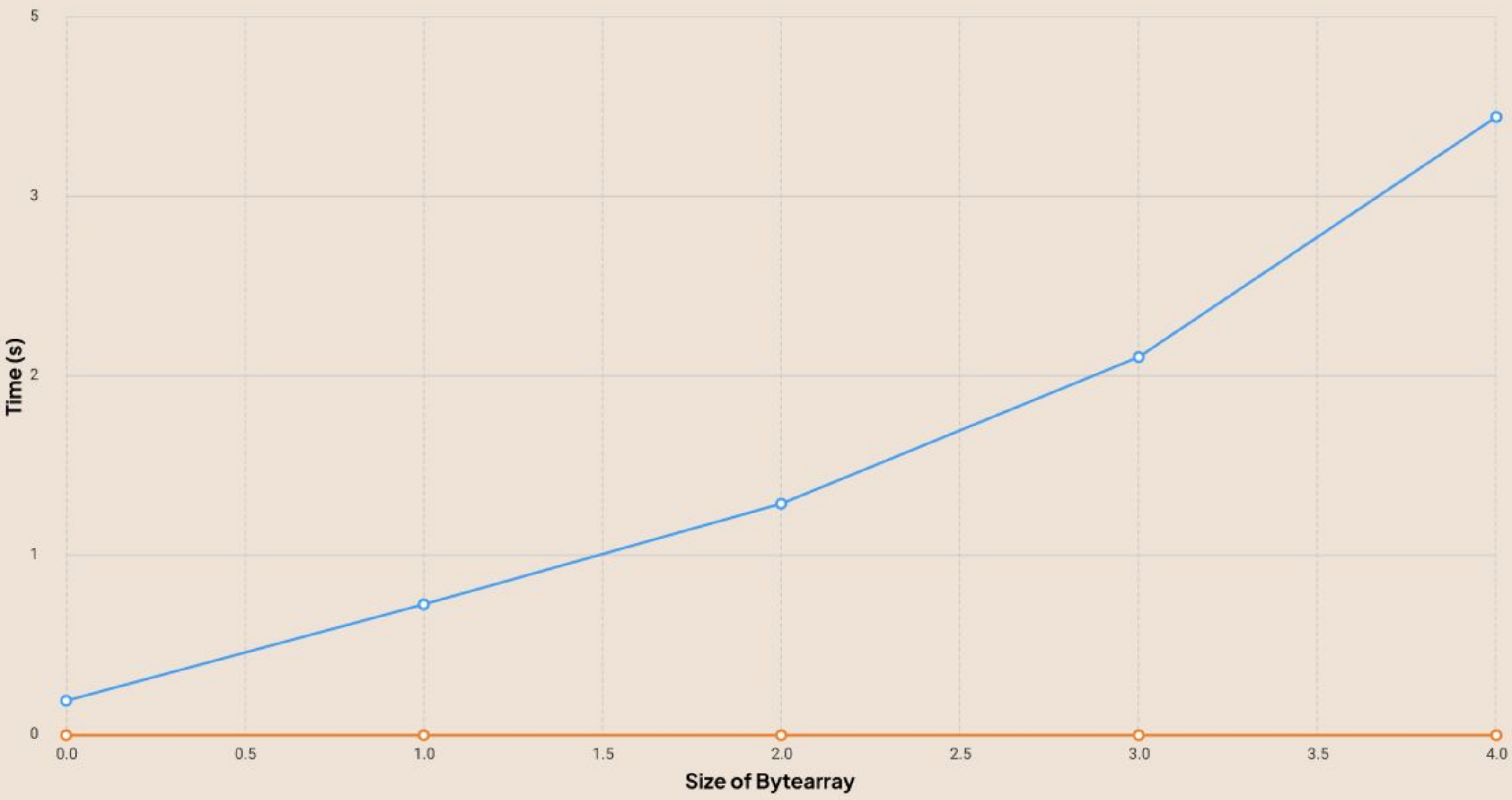
With Memory View

Without Memory View

```
"""
Type              Iterations       Time Taken
--------------------------------------------
bytes             100000            0.24
bytes             200000            0.91
bytes             300000            1.62
bytes             400000            2.64
bytes             500000            4.33
"""
```

With Memory View

```
"""
Type                Iterations       Time Taken
--------------------------------------------
memoryview            100000            0.02
memoryview            200000            0.03
memoryview            300000            0.05
memoryview            400000            0.07
memoryview            500000            0.08
"""
```


Time (s)
5
3
2
1
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Size of Bytearray

Applications

Example – Numeric and Scientific Computing:

```python
import numpy as np

# Create a large NumPy array
size = 1000000
large_array = np.arange(size)

# Create a memory view of the array
mem_view = memoryview(large_array)

# Double the values in the array using memory view
for i in range(len(mem_view)):
    mem_view[i] *= 2

# Print the first 10 elements of the modified array
print(large_array[:10])
# [ 0  2  4  6  8 10 12 14 16 18]
```

Example – Video Streaming

```python
def timecode_to_index(video_id, timecode):
    ...
    # Returns the byte offset in the video data

def request_chunk(video_id, byte_offset, size):
    ...
    # Returns size bytes of video_id's data from the
offset
video_id = ...
timecode = '01:09:14:28'
byte_offset = timecode_to_index(video_id, timecode)
size = 20 * 1024 * 1024
video_data = request_chunk(video_id, byte_offset, size)
```

Example – Video Streaming

```
import timeit

def run_test():
    chunk = video_data[byte_offset:byte_offset + size]
    # Call socket.send(chunk), but ignoring for
benchmark
result = timeit.timeit(
    stmt='run_test()',
    globals=globals(),
    number=100) / 100

print(f'{result:0.9f} seconds')
>>>
0.004925669 seconds
```

Example – Video Streaming

```
video_view = memoryview(video_data)

def run_test():
    chunk = video_view[byte_offset:byte_offset + size]
    # Call socket.send(chunk), but ignoring for
benchmark
result = timeit.timeit(
    stmt='run_test()',
    globals=globals(),
    number=100) / 100

print(f'{result:0.9f} seconds')
>>>
0.000000250 seconds
```

Benefits

Reduced Memory Footprint

Benefits

Better Performance

Reduced Memory Footprint

Benefits

Better Performance

Reduced Memory Footprint

Improved Scalability

References

1. Idea of zero copy [with example]
2. Effective Python › Item 74: Consider memoryview and bytearray for Zero-Copy Interactions with bytes
3. Python memoryview()
4. Python Memoryview Example for Beginners
5. memoryview in Python
6. Python memoryview() Function (Buffer Protocol And Memory View)
7. PEP 688 – Making the buffer protocol accessible in Python

EUROPYTHON 17-23 July 2023
PRAGUE & REMOTE

Thank you

@kesiajoies

@abymjoseph