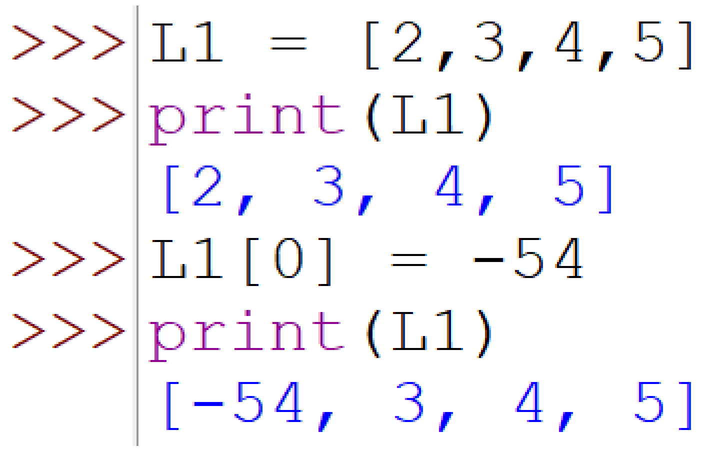
Mutable vs immutable types

Week 5

Mutable data types

- To avoid mistakes that are difficult to find when dealing with functions, it is also important to know about mutable amd immutable data types.
- **Mutable types** are those whose values can be changed after they are created.
- Common mutable datatypes: **list**, **dictionary**, **set**.
- We will consider them later in this course, but you better know about this now.

List

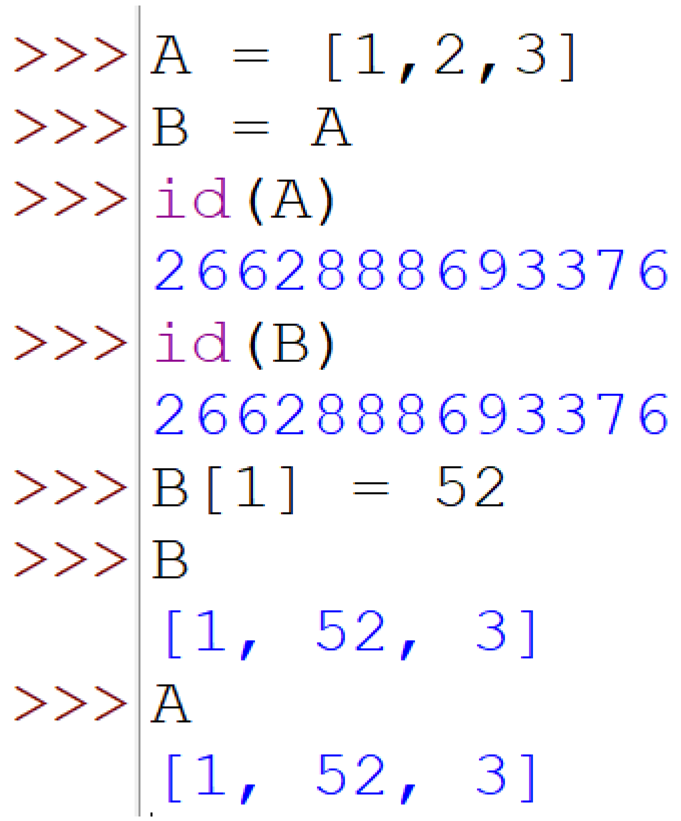


```
>>> L1 = [2,3,4,5]
>>> print(L1)
    [2, 3, 4, 5]
>>> L1[0] = -54
>>> print(L1)
    [-54, 3, 4, 5]
```

Warning

- In Python, when you set B = A and A is a mutable datatype (like a list or dictionary), both A and B refer to the same object in memory.
- This means that any changes made to the object through B will also be reflected when accessing it through A, and vice versa.

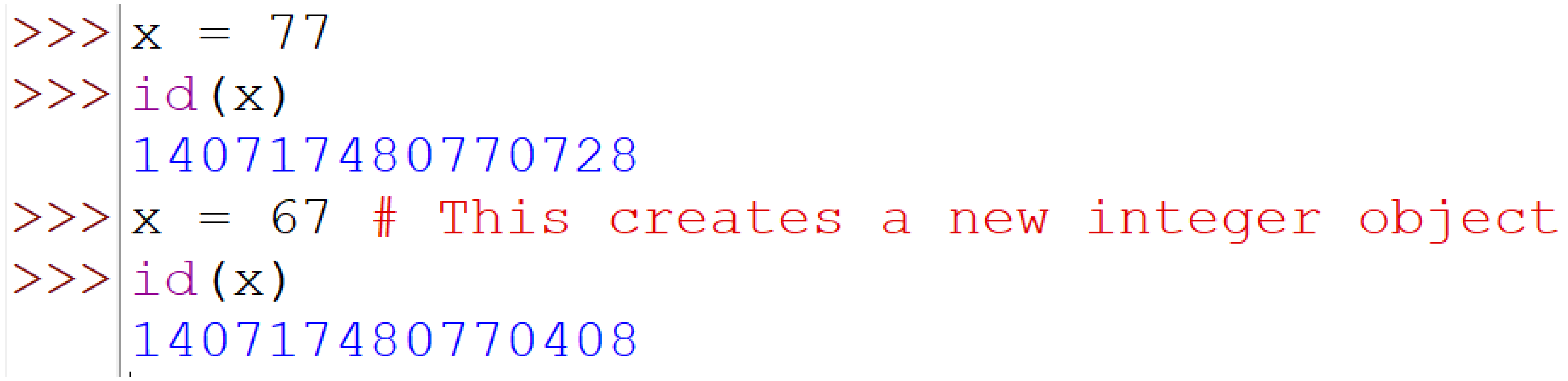
Example



```
>>> A = [1,2,3]
>>> B = A
>>> id(A)
    2662888693376
>>> id(B)
    2662888693376
>>> B[1] = 52
>>> B
    [1, 52, 3]
>>> A
    [1, 52, 3]
```

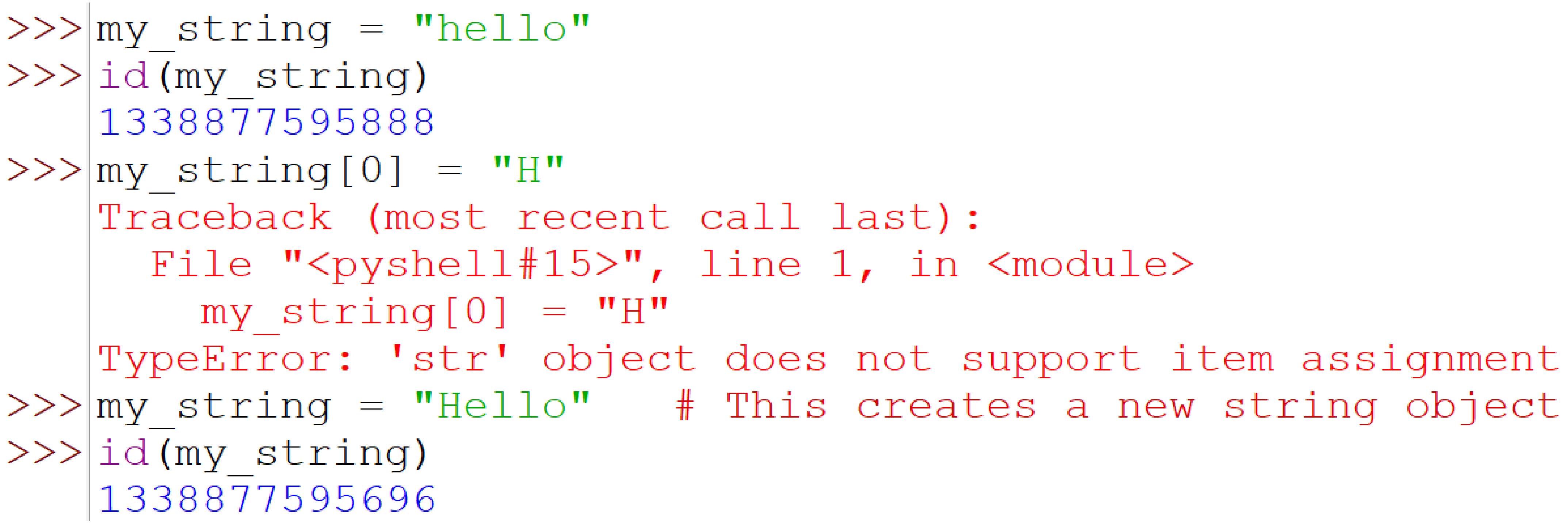
Immutable data types

- Immutable types are those whose values cannot be changed after they are created.
- Common immutable datatypes: **string**, **tuple**, **integer**, **float**, **Boolean**



```
>>> x = 77
>>> id(x)
    140717480770728
>>> x = 67 # This creates a new integer object
>>> id(x)
    140717480770408
```

Strings are immutable



```
>>> my_string = "hello"
>>> id(my_string)
    1338877595888
>>> my_string[0] = "H"
    Traceback (most recent call last):
      File "<pyshell#15>", line 1, in <module>
        my_string[0] = "H"
    TypeError: 'str' object does not support item assignment
>>> my_string = "Hello"    # This creates a new string object
>>> id(my_string)
    1338877595696
```

Warning

If you pass a mutable object as a parameter to a function and change its value inside the function, the object will change.

File: `list_parameter.py`