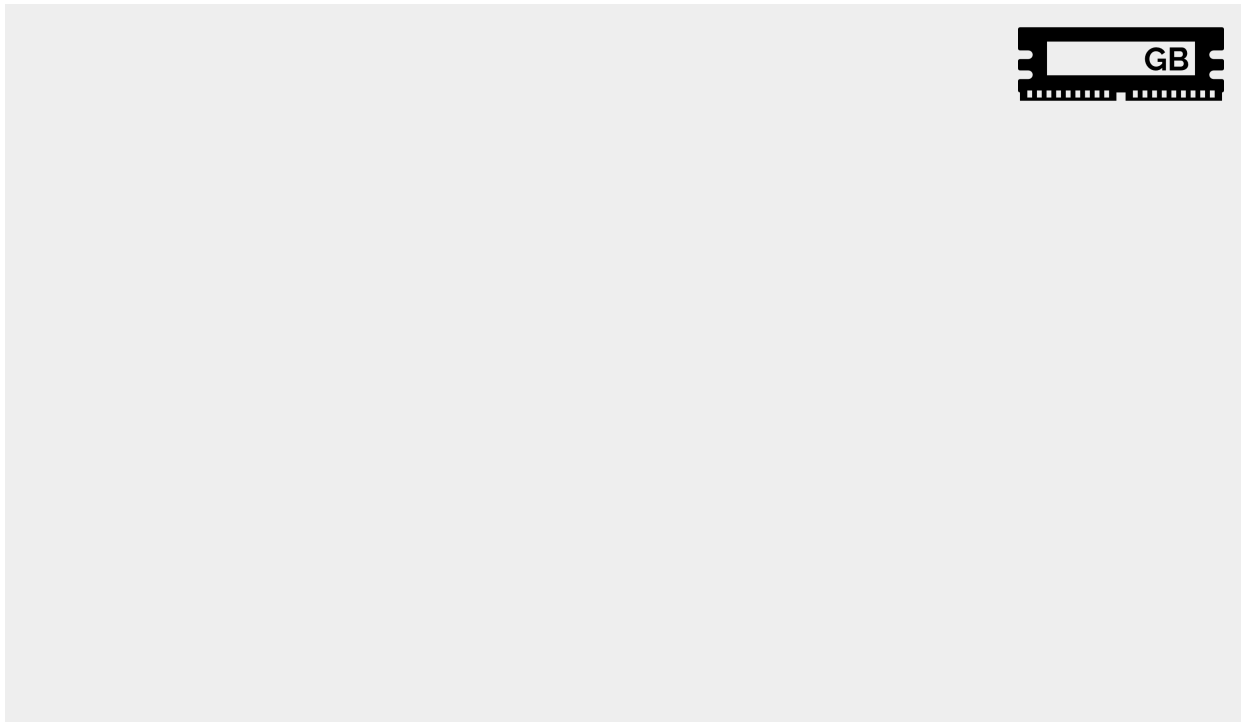
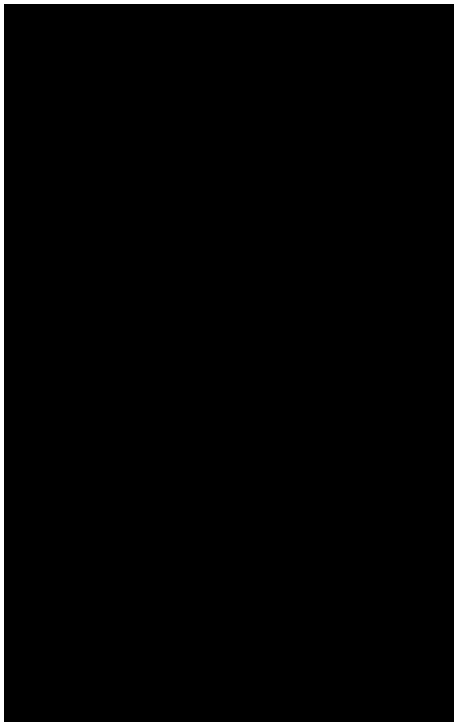




Variables in Python

What is variable in reality?



```
> a = 42
```



```
> a = 42
```

variable is a **reference** to an
object living in memory

a

0x001

42



```
> a = 42  
> id(a)
```

a

0x001

42



```
> a = 42  
> id(a)  
>> 0x001
```

a

0x001

42



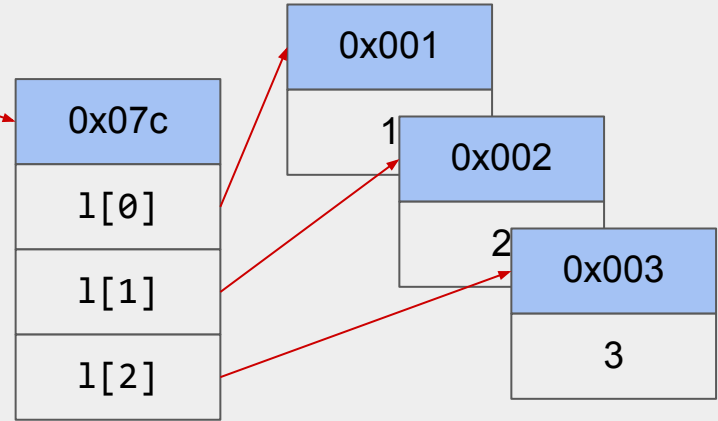
Mutable types


```
> l = [1, 2, 3]
```



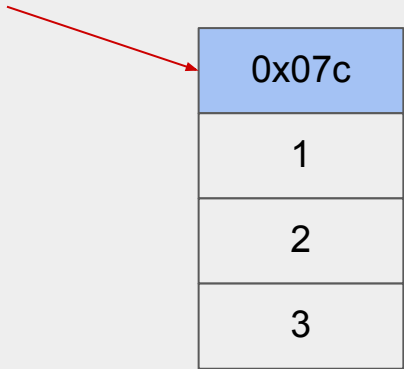
```
> l = [1, 2, 3]
```

l



```
> l = [1, 2, 3]
```

l

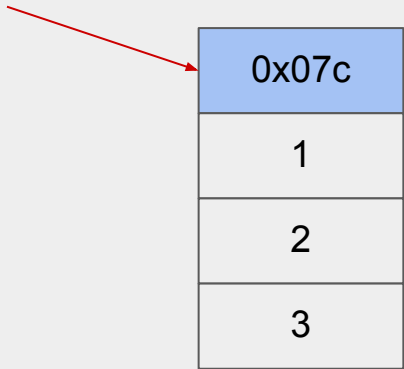


0x07c
1
2



```
> l = [1, 2, 3]  
> g = l
```

l

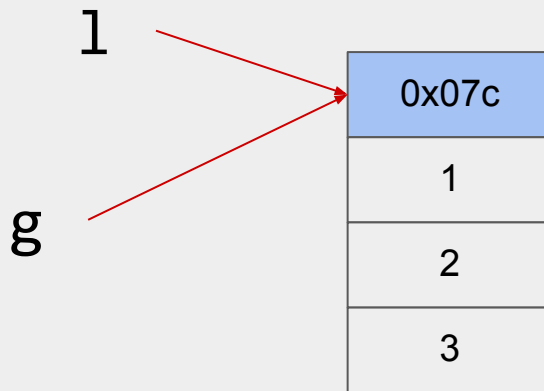


0x07c
1
2
3

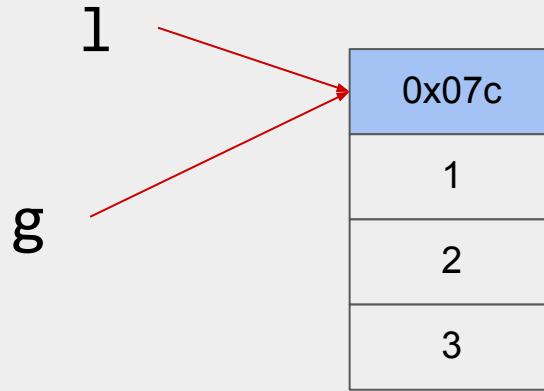


```
> l = [1, 2, 3]
> g = l
```

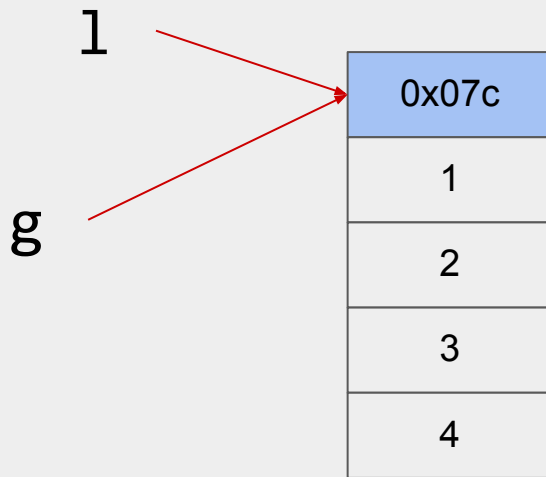
we are not creating new list,
instead we create **new**
reference to the same object



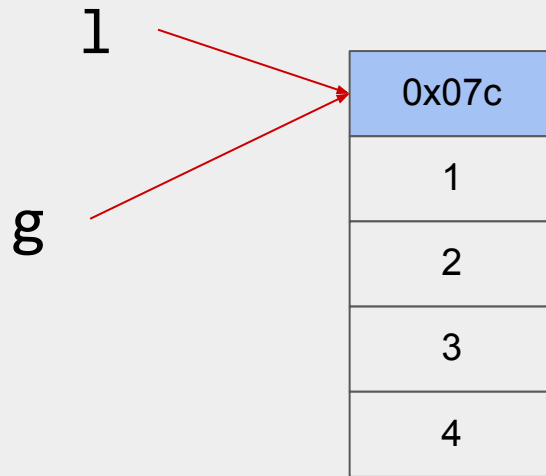
```
> l = [1, 2, 3]  
> g = l  
> g.append(4)
```



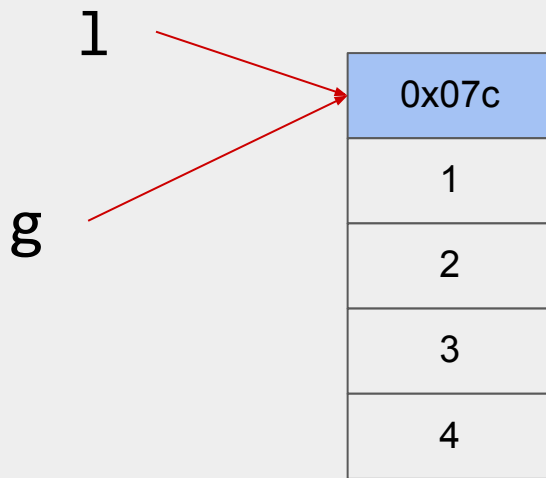
```
> l = [1, 2, 3]  
> g = l  
> g.append(4)
```



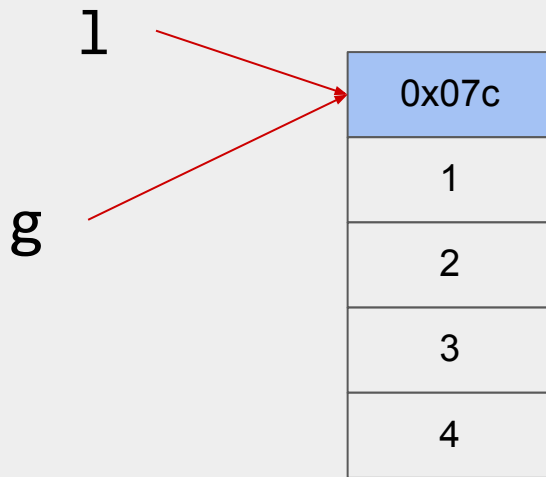
```
> l = [1, 2, 3]  
> g = l  
> g.append(4)  
> l
```



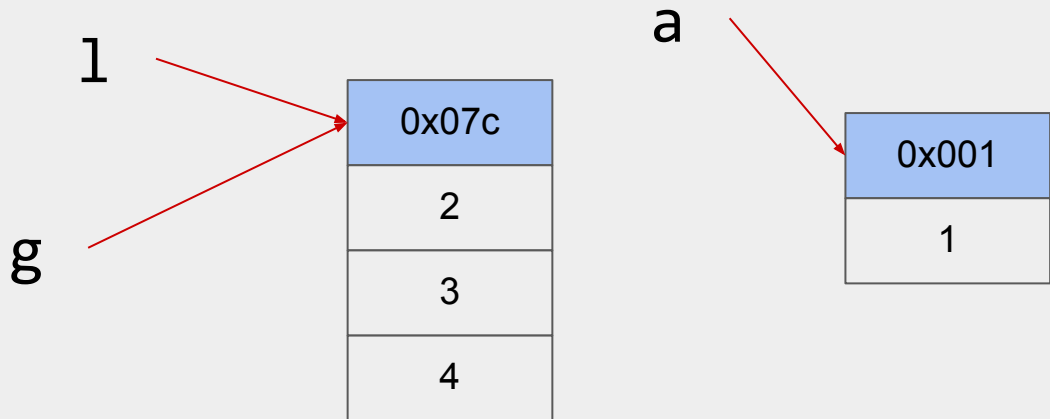

```
> l = [1, 2, 3]
> g = l
> g.append(4)
> l
>> [1, 2, 3, 4]
```



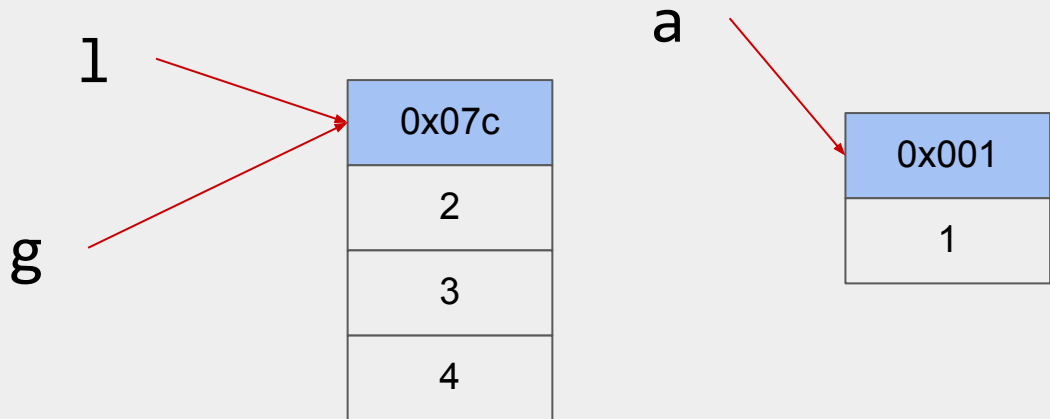
```
> l = [1, 2, 3]
> g = l
> g.append(4)
> l
>> [1, 2, 3, 4]
> a = l.pop(0)
```



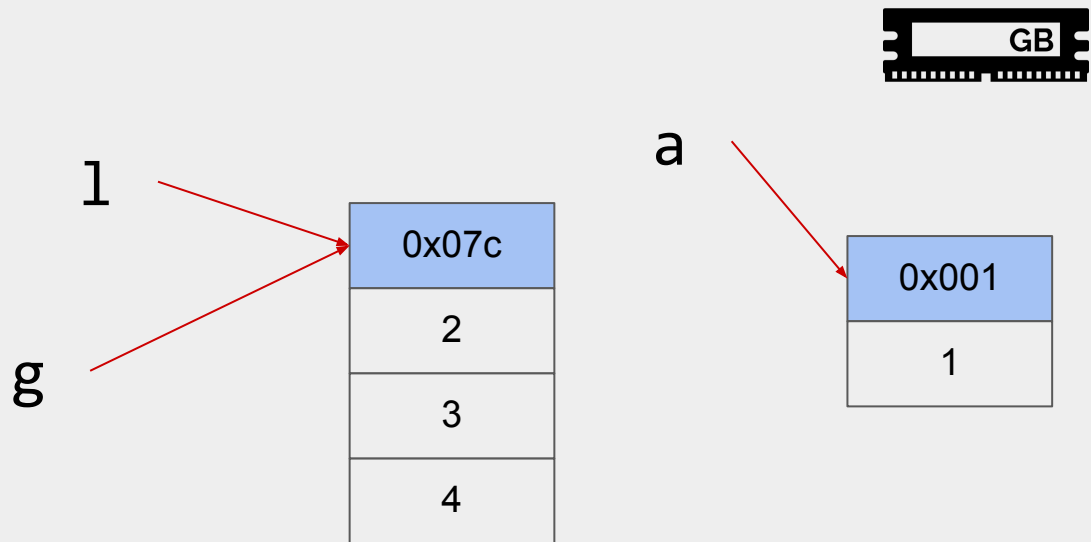
```
> l = [1, 2, 3]
> g = l
> g.append(4)
> l
>> [1, 2, 3, 4]
> a = l.pop(0)
```



```
> l = [1, 2, 3]
> g = l
> g.append(4)
> l
>> [1, 2, 3, 4]
> a = l.pop(0)
> g
```



```
> l = [1, 2, 3]
> g = l
> g.append(4)
> l
>> [1, 2, 3, 4]
> a = l.pop(0)
> g
>> [2, 3, 4]
```



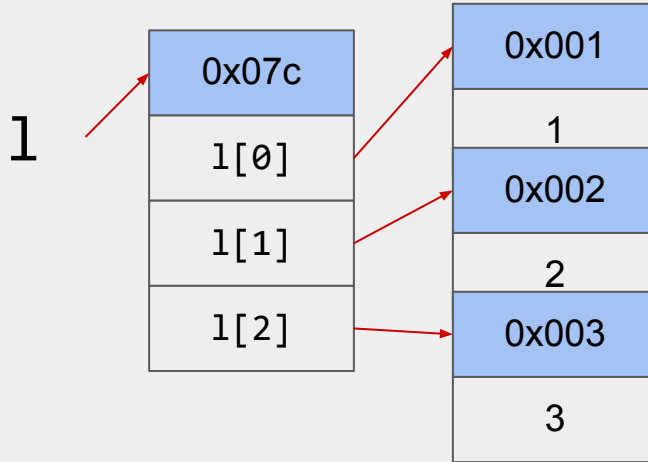
Quiz

```
l = [1, 2, 3]
g = [l[0], l[1], l[2]]
g.append(4)
print(len(l))
```

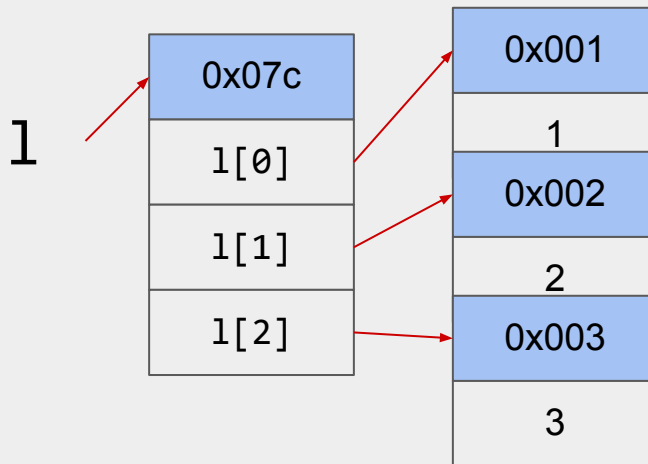
```
> l = [1, 2, 3]
```



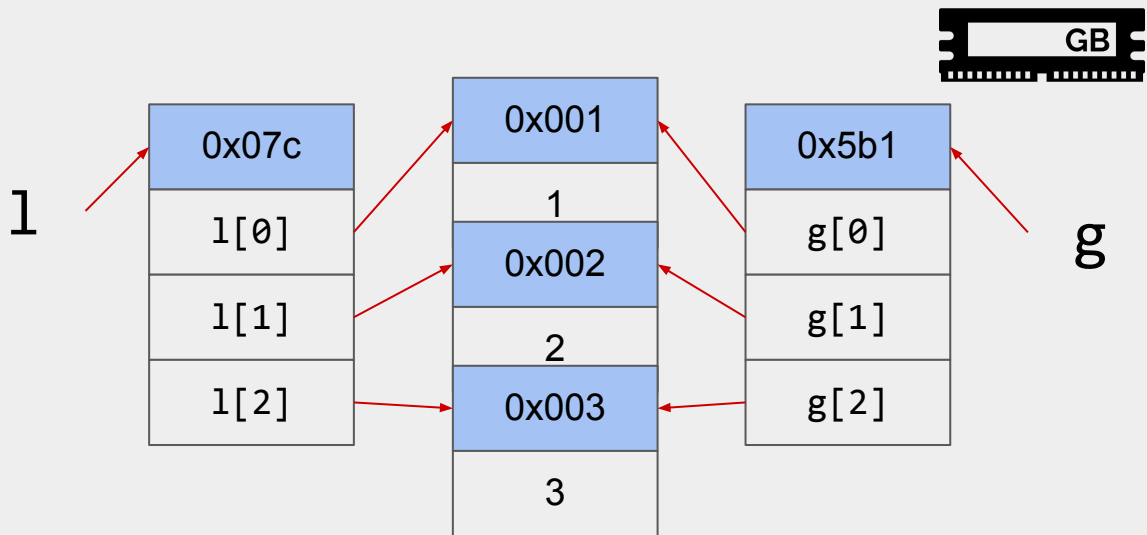
```
> l = [1, 2, 3]
```



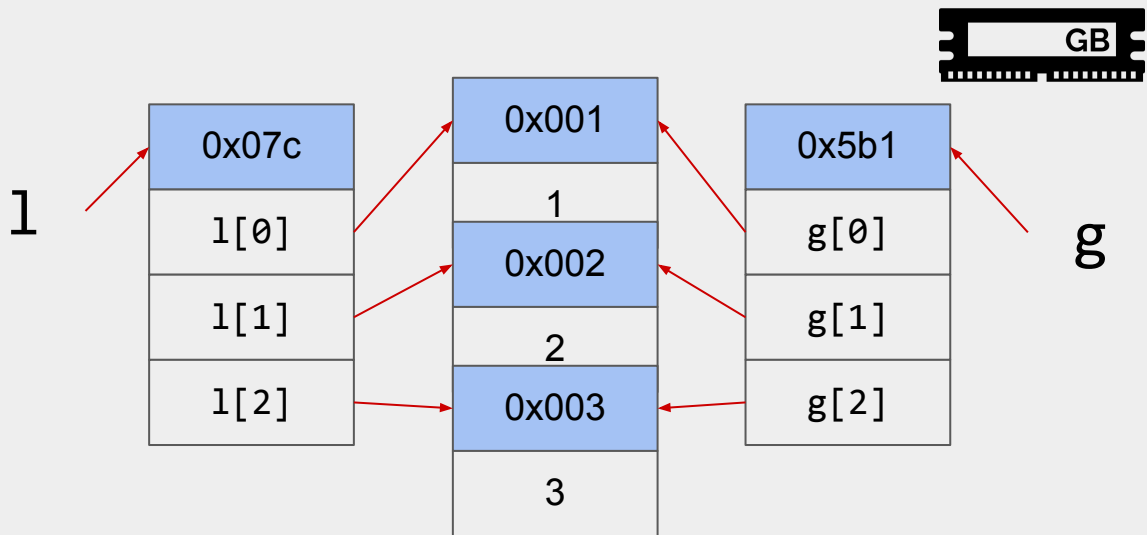

```
> l = [1, 2, 3]
> g = [l[0],
l[1], l[2]]
```



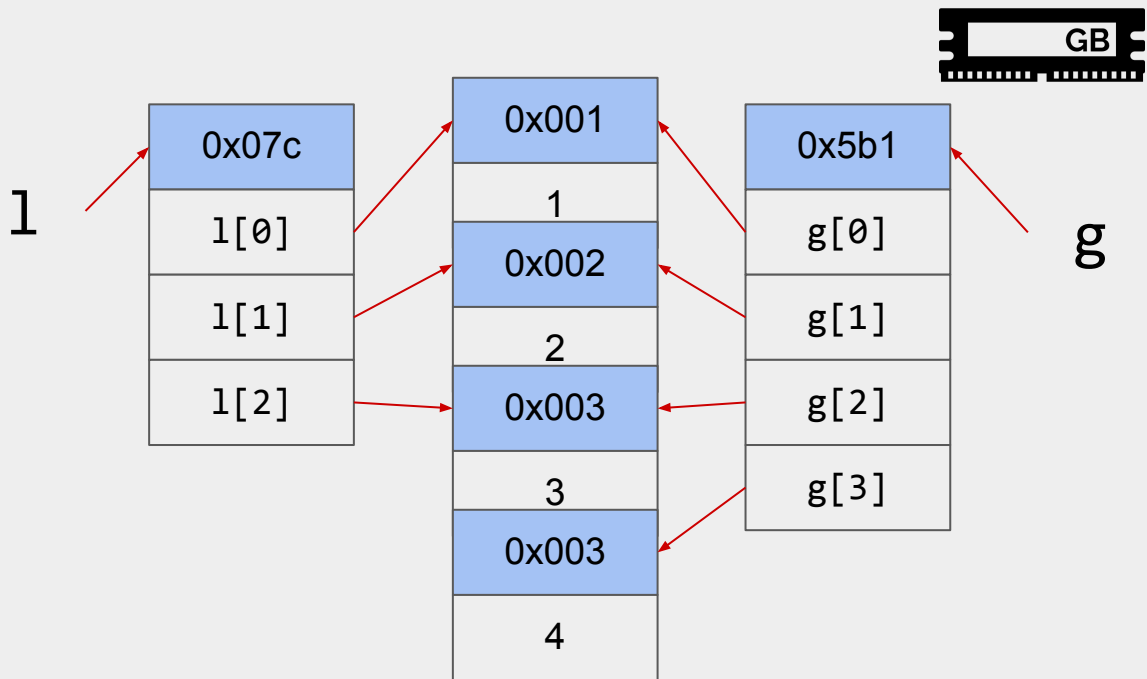
```
> l = [1, 2, 3]
> g = [l[0],
l[1], l[2]]
```



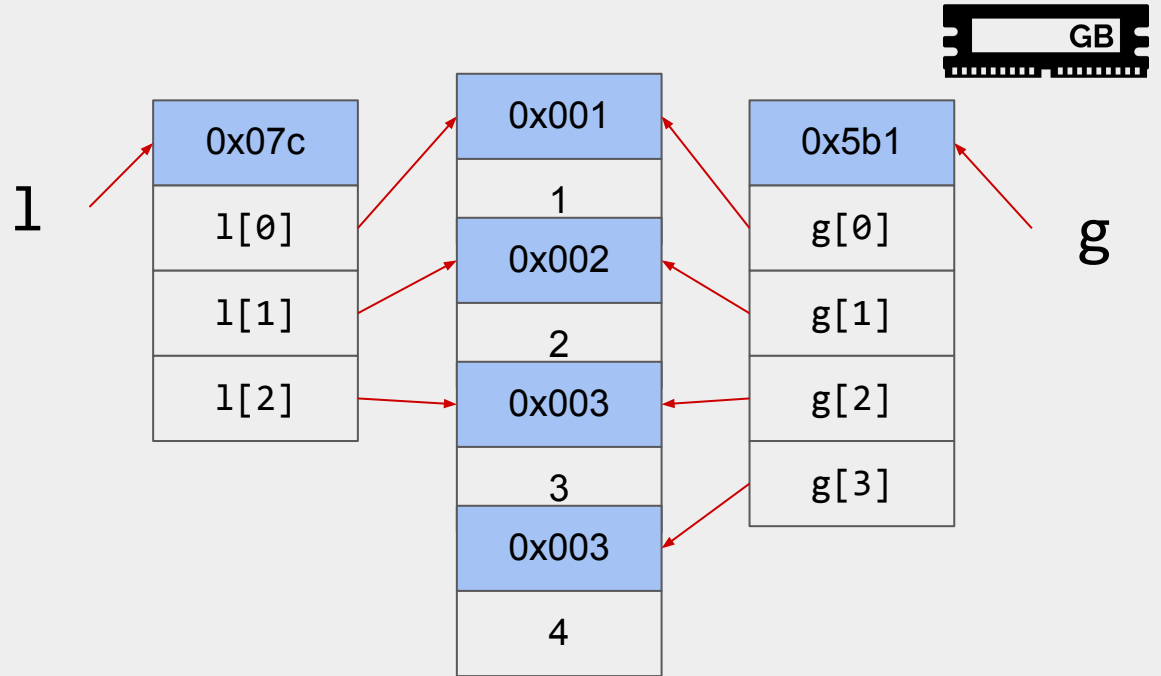
```
> l = [1, 2, 3]
> g = [l[0],
l[1], l[2]]
> g.append(4)
```



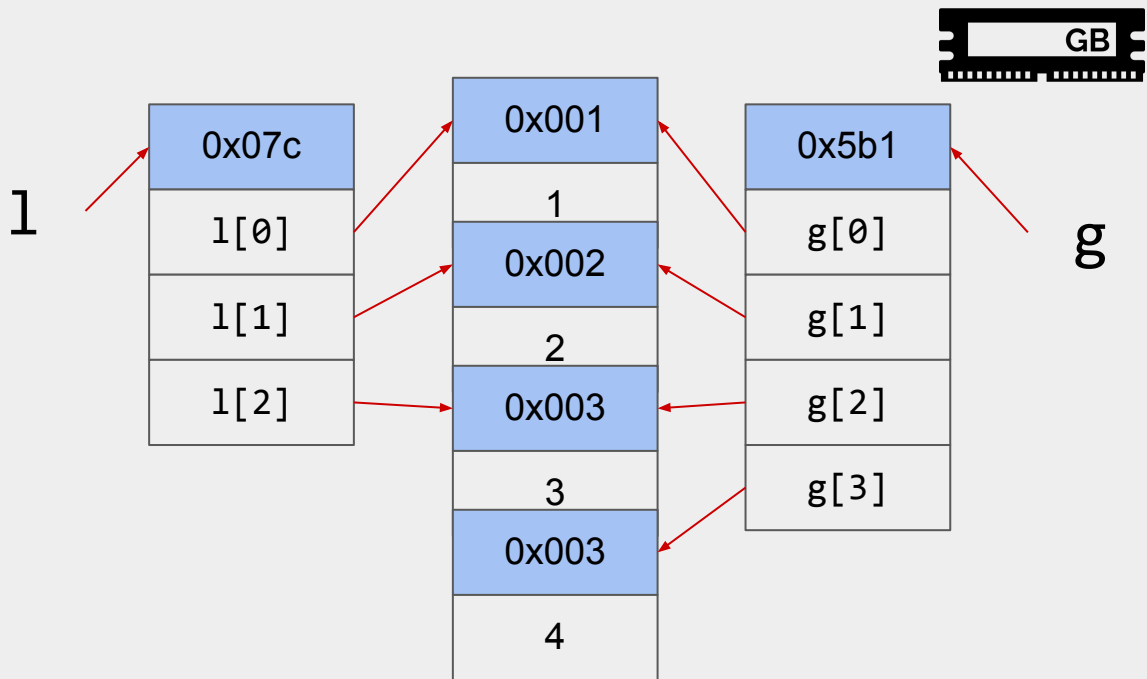
```
> l = [1, 2, 3]
> g = [l[0],
l[1], l[2]]
> g.append(4)
```



```
> l = [1, 2, 3]
> g = [l[0],
l[1], l[2]]
> g.append(4)
> len(l)
```



```
> l = [1, 2, 3]
> g = [l[0],
l[1], l[2]]
> g.append(4)
> len(l)
>> 3
```



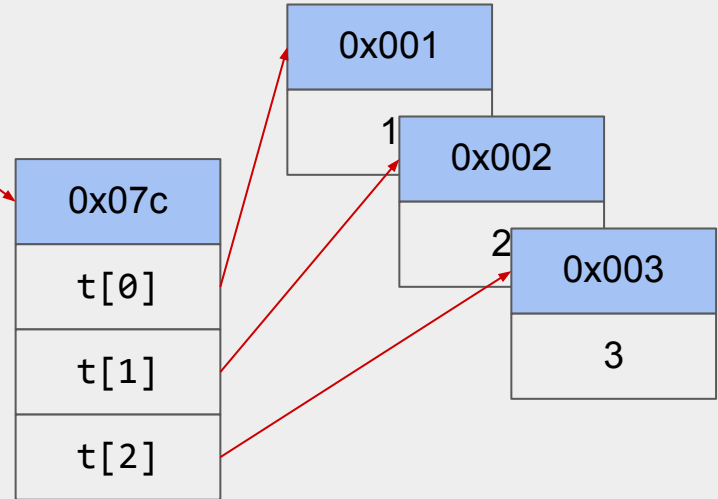
Immutable types

```
> t = (1, 2, 3)
```



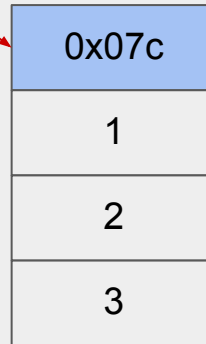

```
> t = (1, 2, 3)
```

t

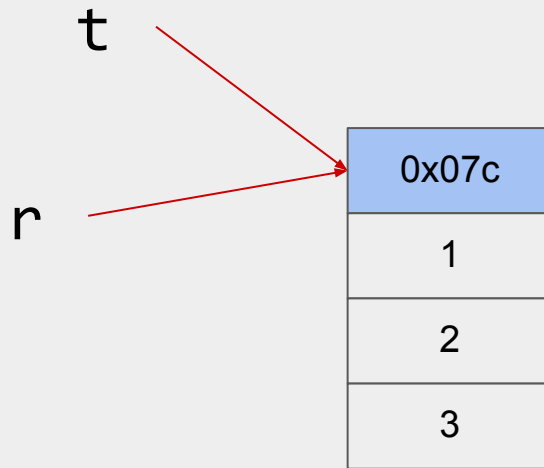


```
> t = (1, 2, 3)
```

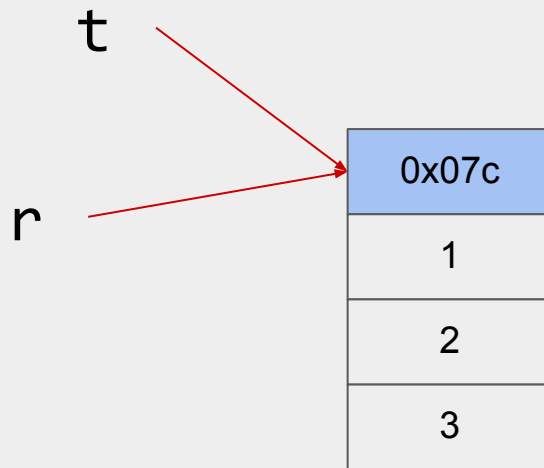
t



```
> t = (1, 2, 3)
> r = t
```



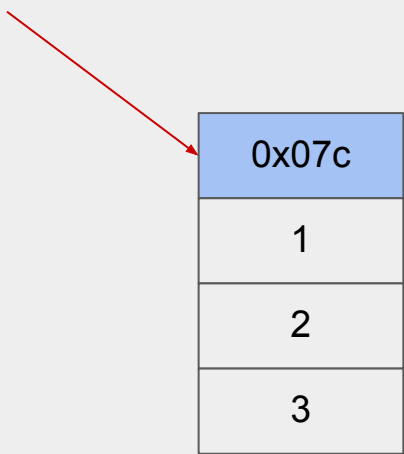
```
> t = (1, 2, 3)
> r = t
> r = r + (4,)
```



```
> t = (1, 2, 3)
> r = t
> r = r + (4,)
```

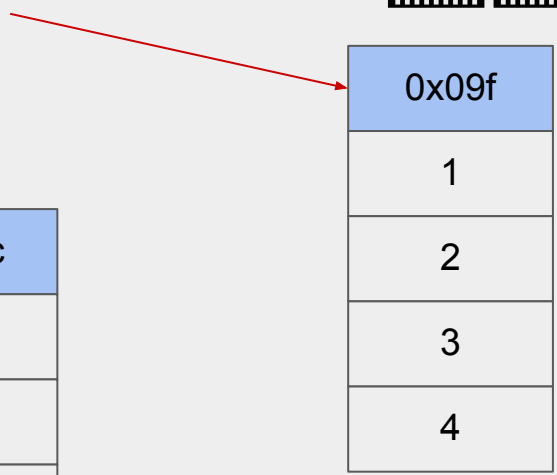
tuple is immutable, so new
object is created

t



0x07c
1
2
3

r



0x09f
1
2
4

```
> t = (1, 2, 3)
> r = t
> r = r + (4,)
```

it looks like there are two
integers 1 in memory, but
remember, in reality...

t

A red arrow points from the variable 't' to the top of a vertical list structure. The list has a blue header box containing the address '0x07c' and three white data boxes containing the values 1, 2, and 3.

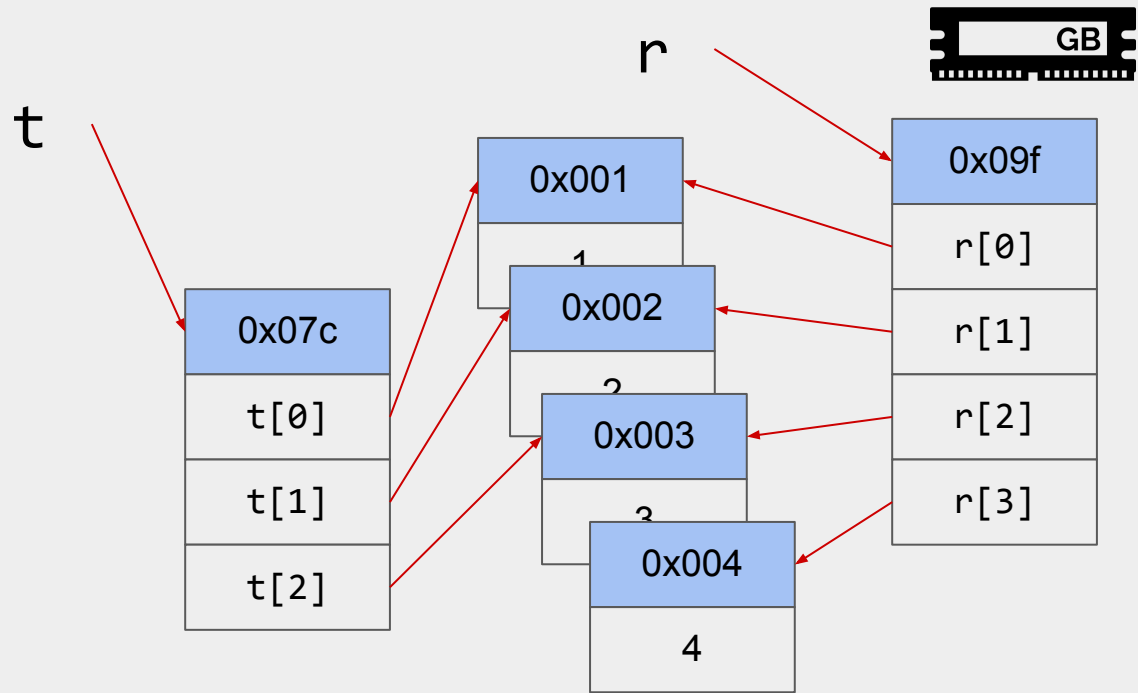
0x07c
1
2
3

r

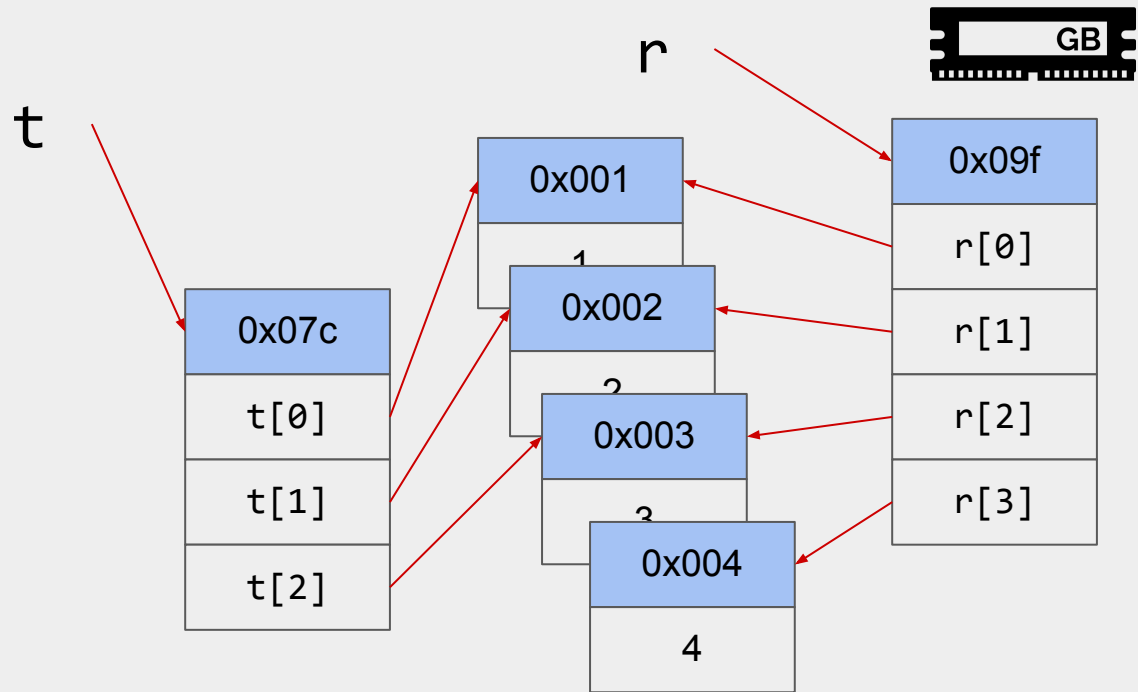
A red arrow points from the variable 'r' to the top of a vertical list structure. The list has a blue header box containing the address '0x09f' and four white data boxes containing the values 1, 2, 3, and 4. Above this list is a small icon of a memory card labeled 'GB'.

0x09f
1
2
3
4

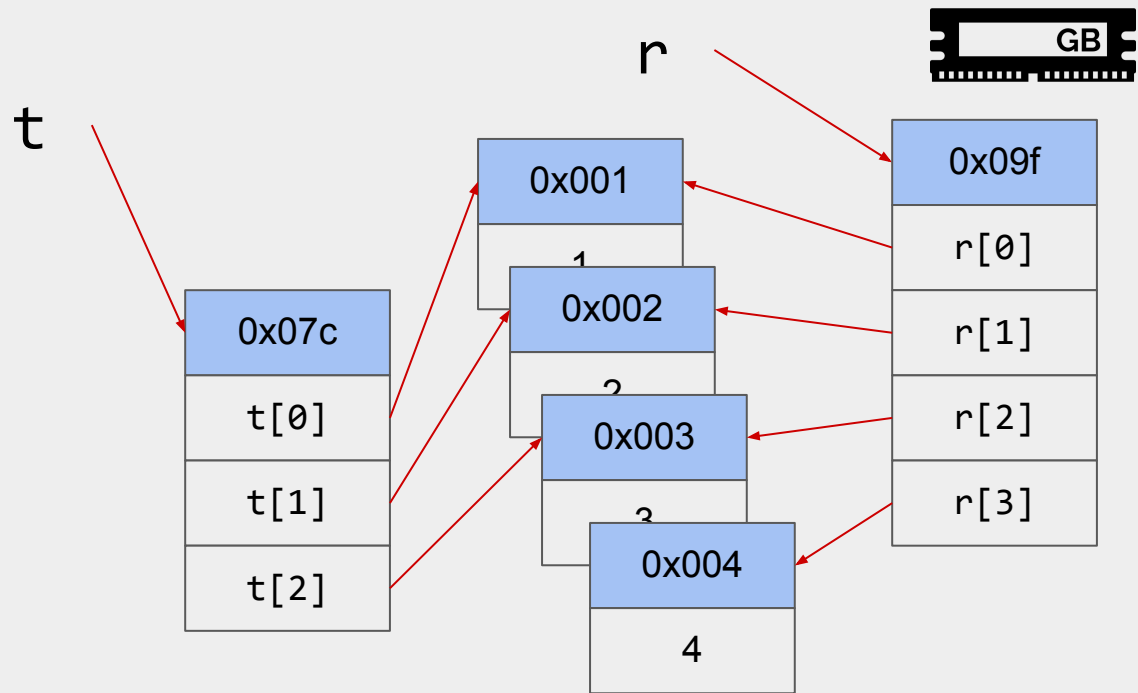
```
> t = (1, 2, 3)
> r = t
> r = r + (4,)
```



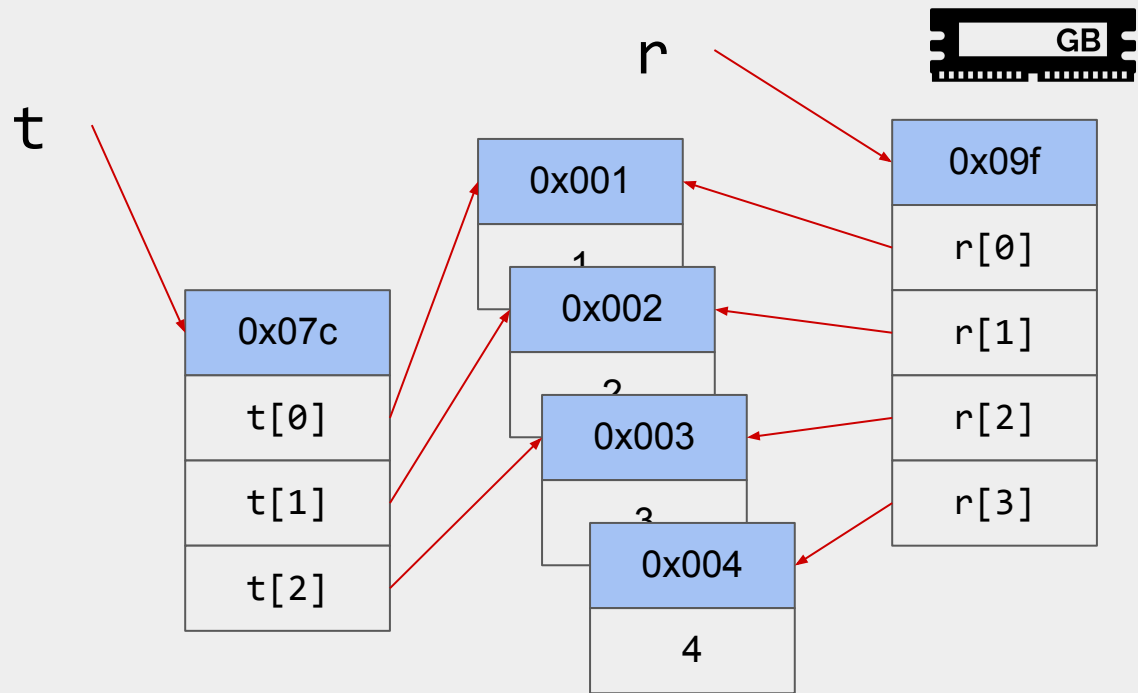
```
> t = (1, 2, 3)
> r = t
> r = r + (4,)
> t
```



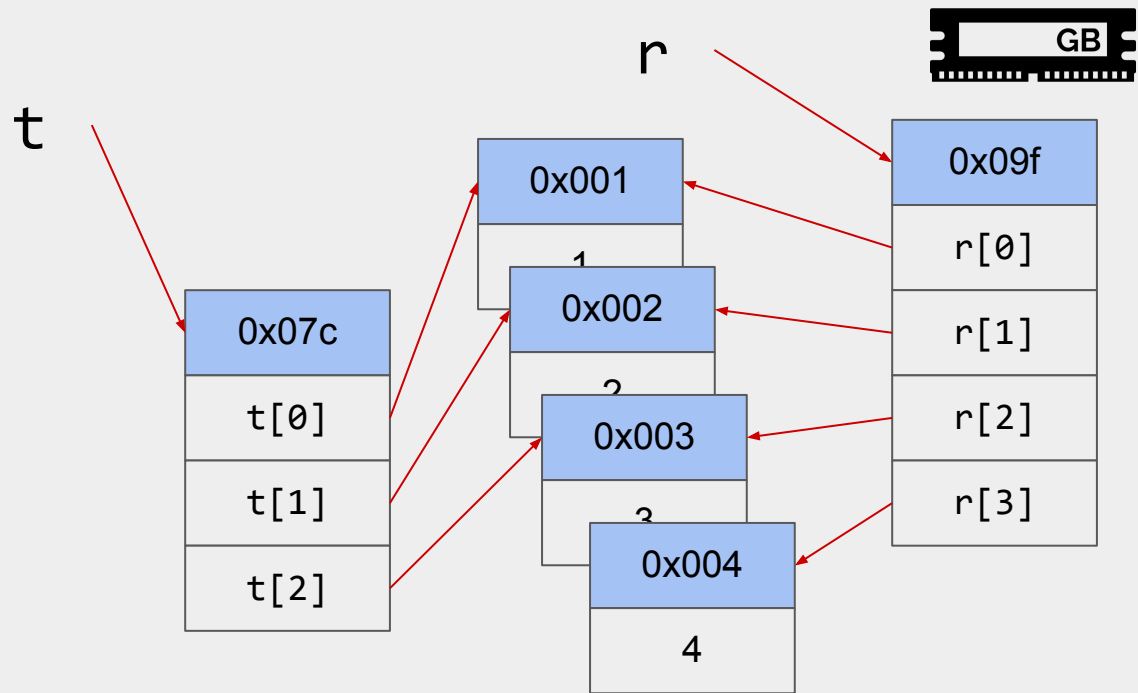

```
> t = (1, 2, 3)
> r = t
> r = r + (4,)
> t
>> (1, 2, 3)
```



```
> t = (1, 2, 3)
> r = t
> r = r + (4,)
> t
>> (1, 2, 3)
> id(t[0]) == id(r[0])
```



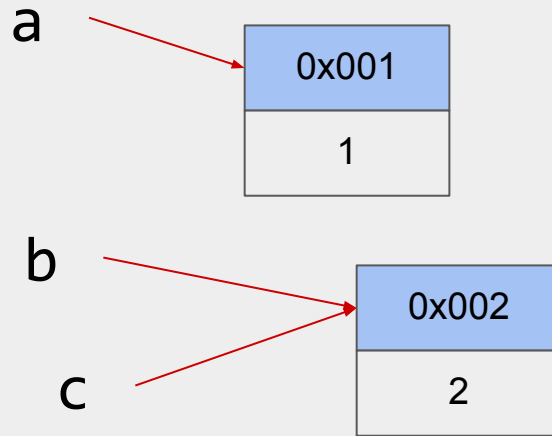
```
> t = (1, 2, 3)
> r = t
> r = r + (4,)
> t
>> (1, 2, 3)
> id(t[0]) == id(r[0])
>> True
```



Quiz

```
a = 1  
b = 2  
c = b ** b - b * a  
print(id(c) == id(b))
```

```
> a = 1  
> b = 2  
> c = b**b-b*a
```



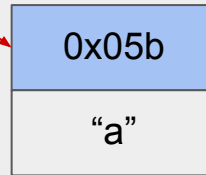
Checking immutability

```
> x = "a"
```



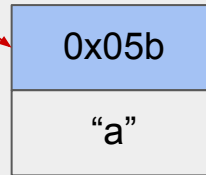
```
> x = "a"
```

x



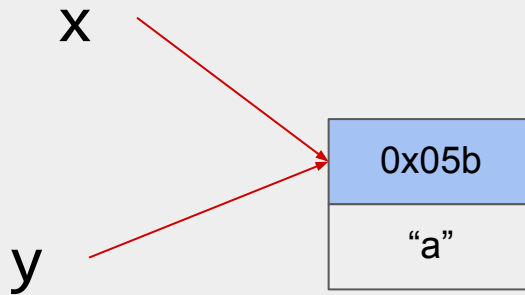

```
> x = "a"  
> y = x
```

X

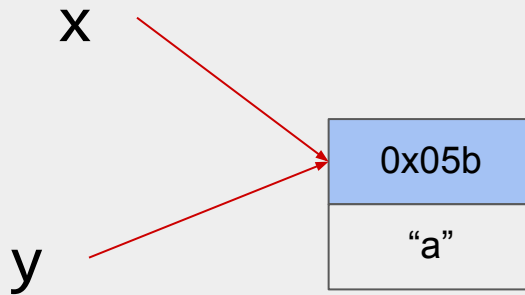


```
> x = "a"
```

```
> y = x
```

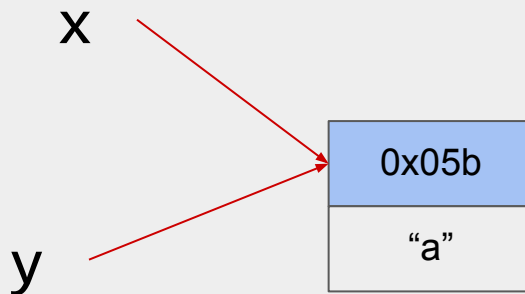


```
> x = "a"  
> y = x  
> y += "b"
```

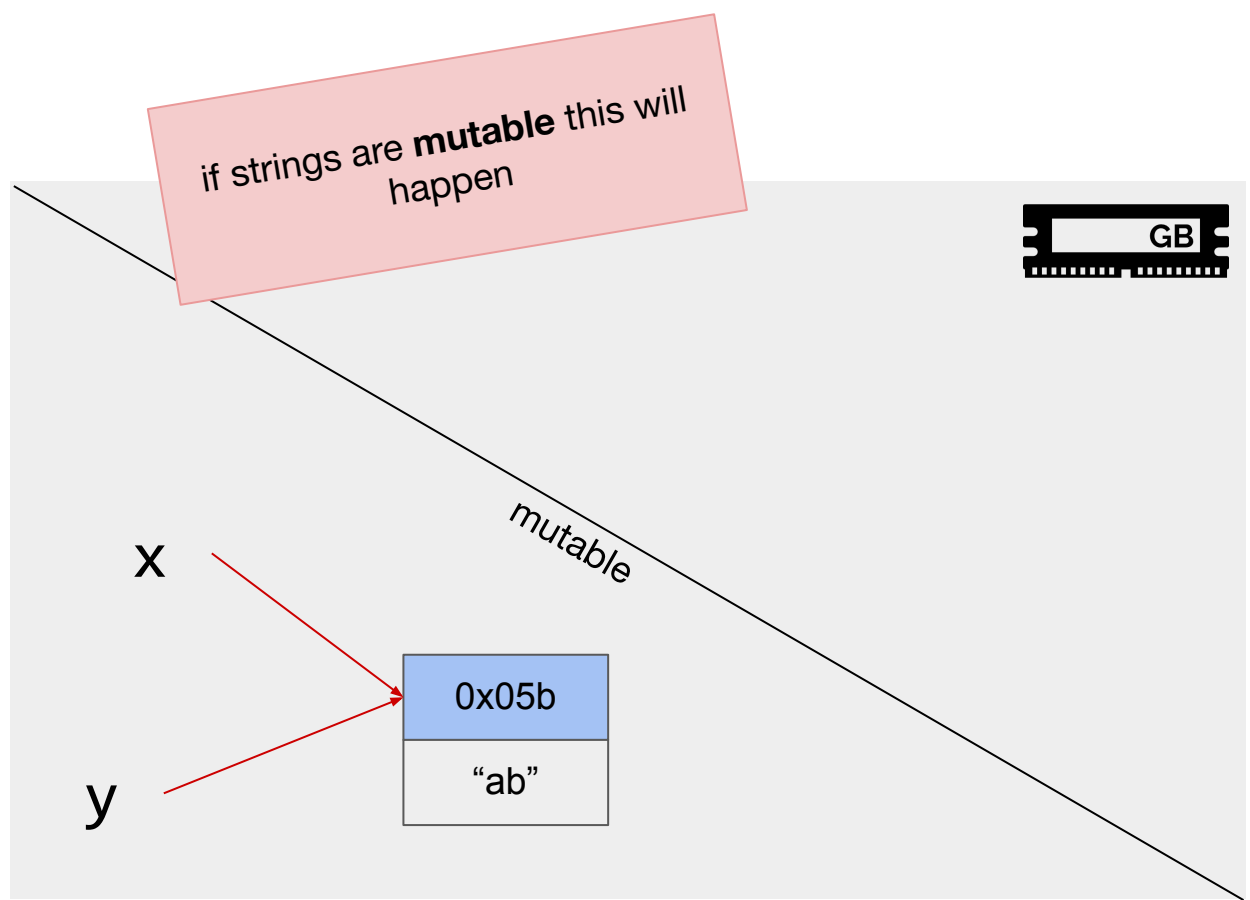


```
> x = "a"  
> y = x  
> y = y + "b"
```

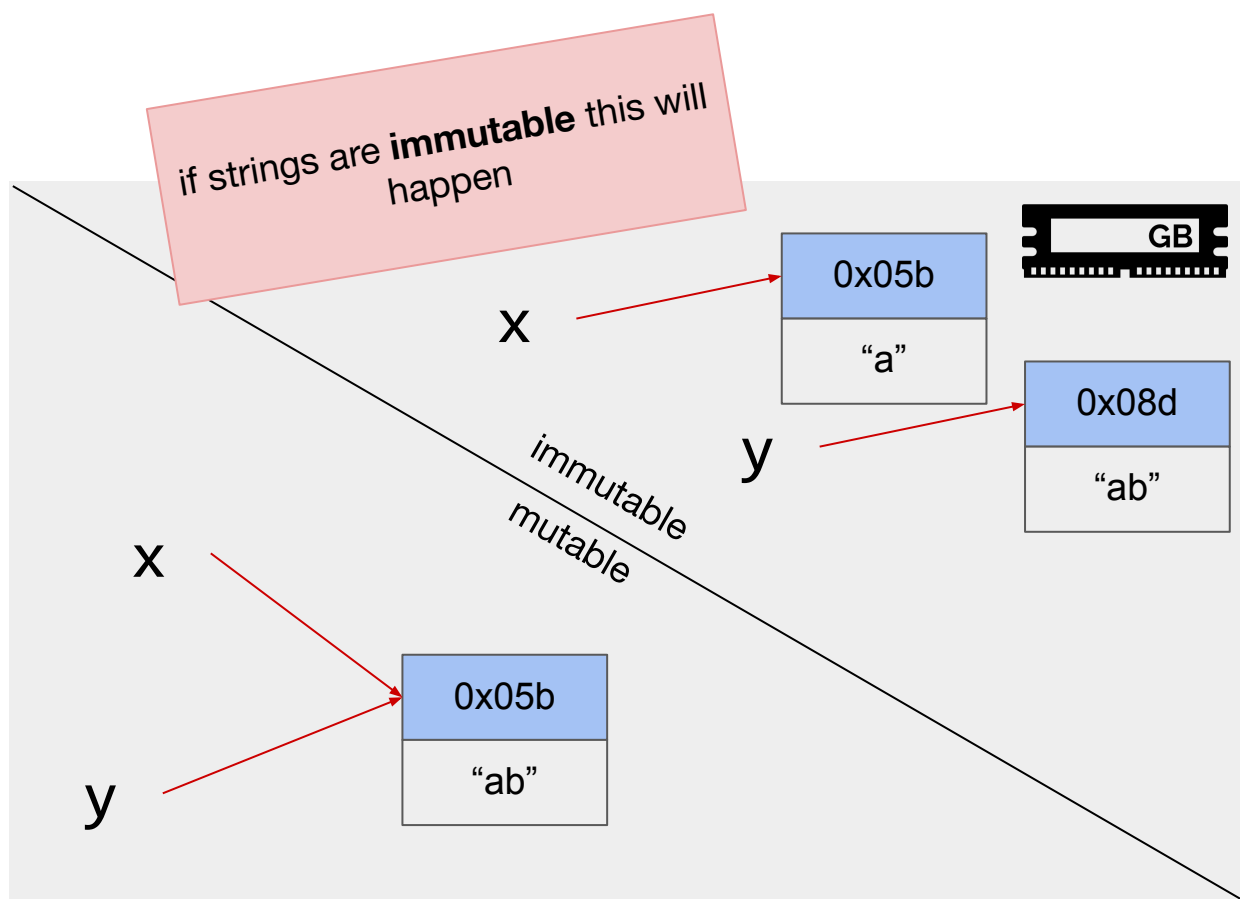
what will happen now?



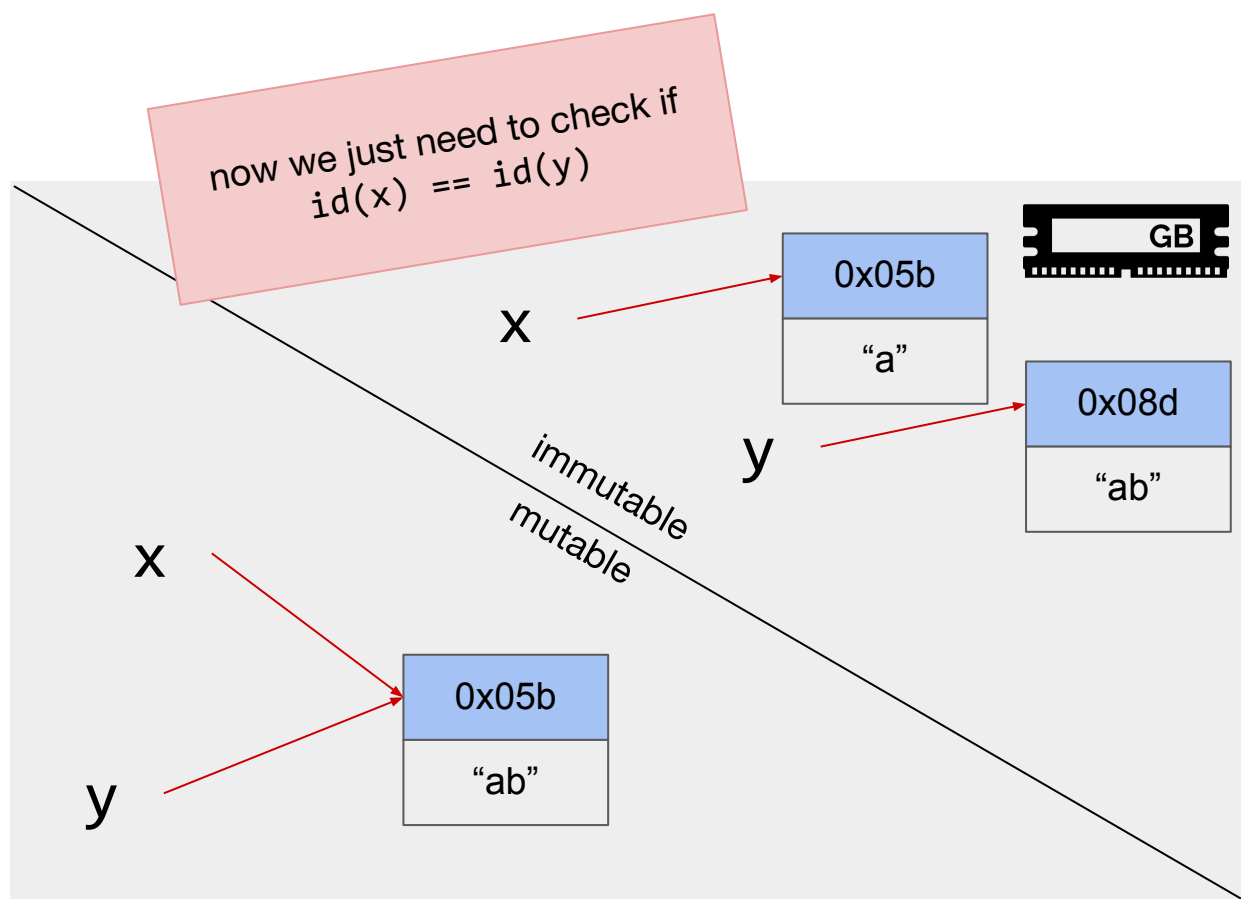
```
> x = "a"  
> y = x  
> y = y + "b"
```



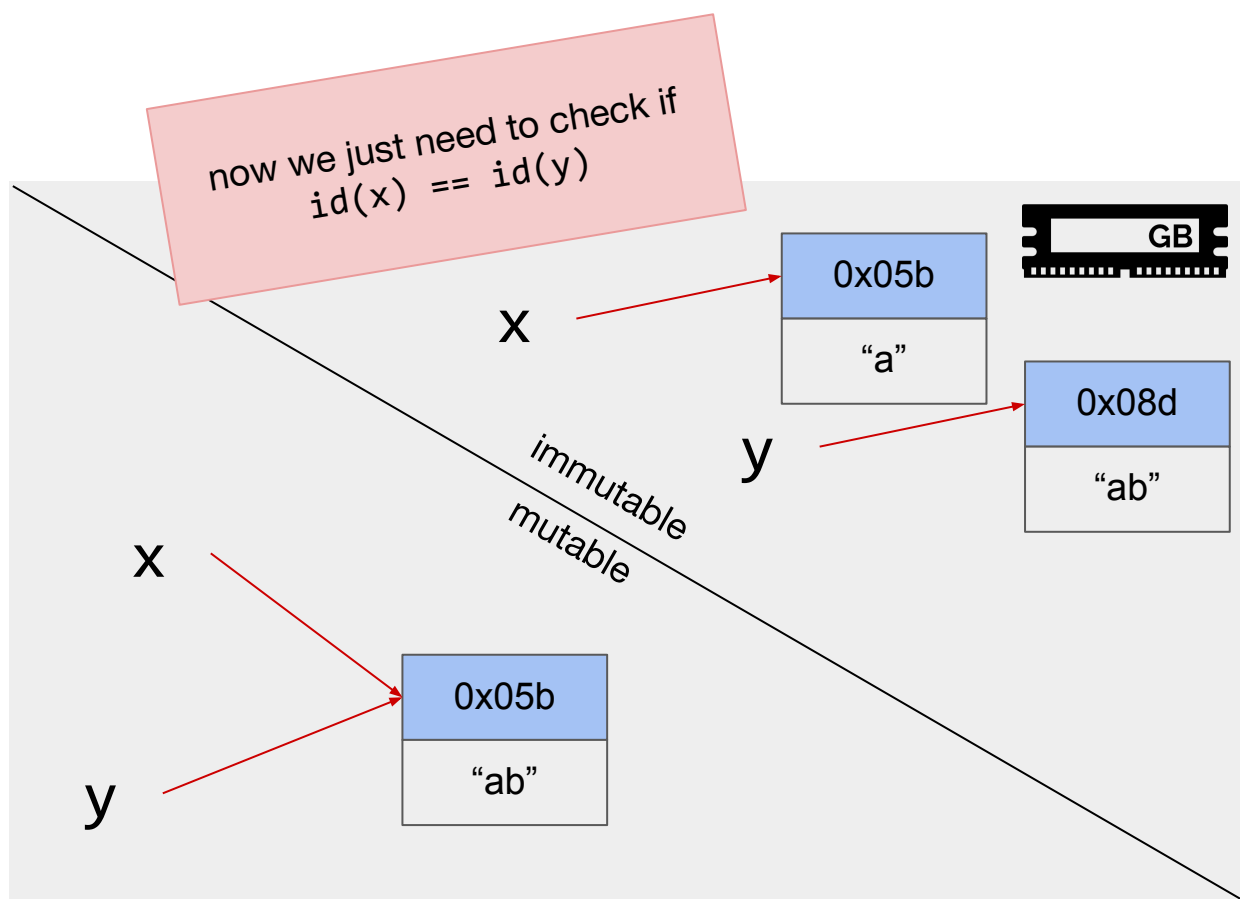
```
> x = "a"  
> y = x  
> y = y + "b"
```



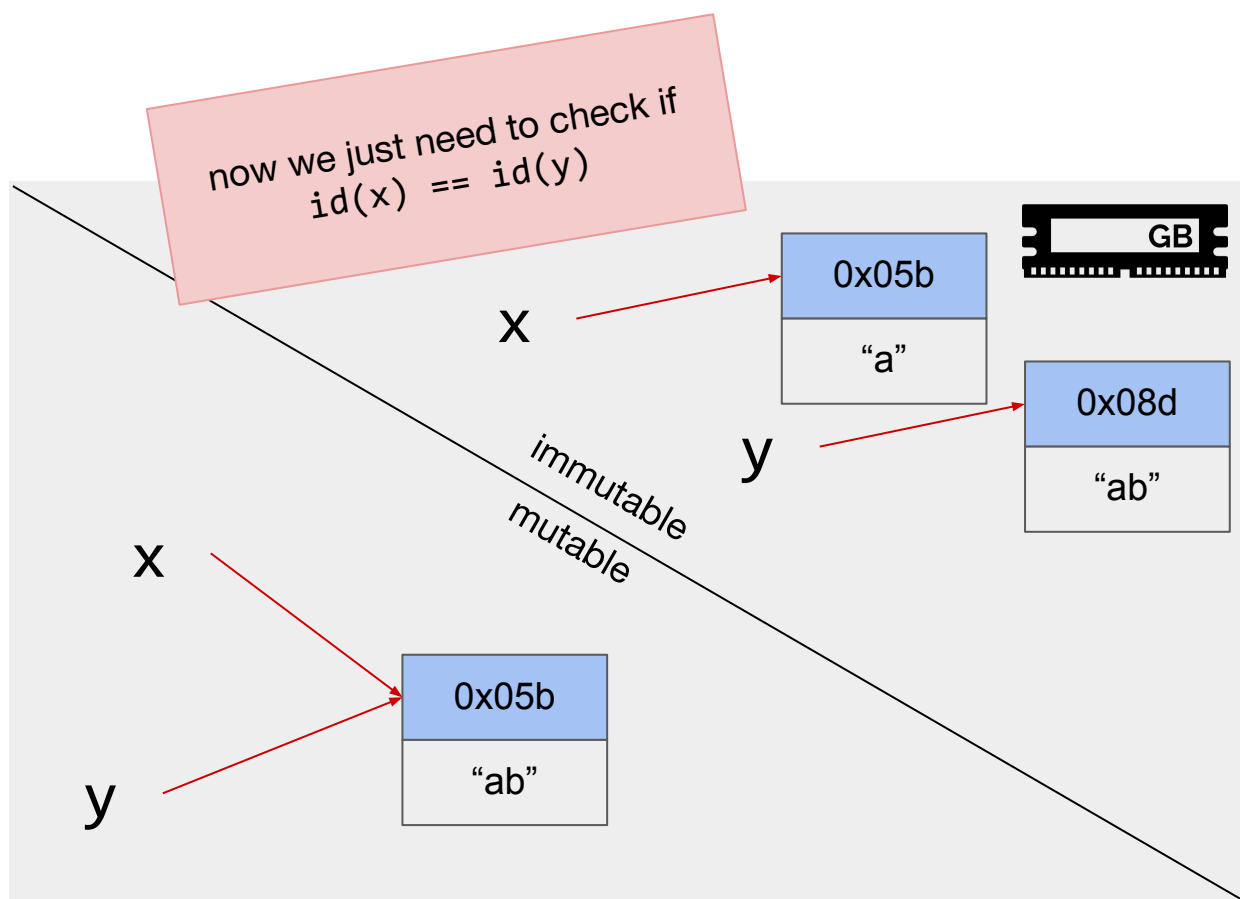
```
> x = "a"  
> y = x  
> y = y + "b"
```



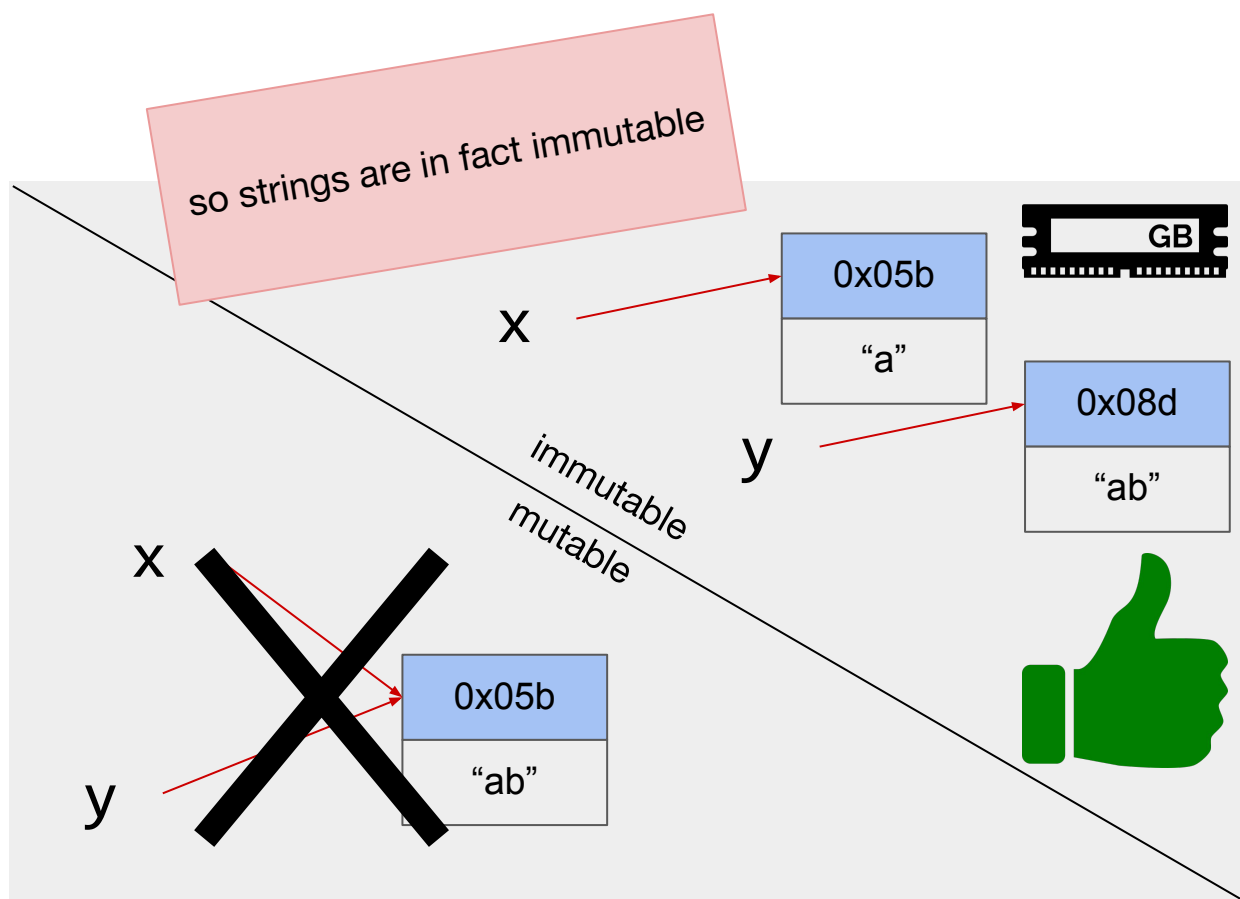
```
> x = "a"  
> y = x  
> y = y + "b"  
> id(x) == id(y)
```



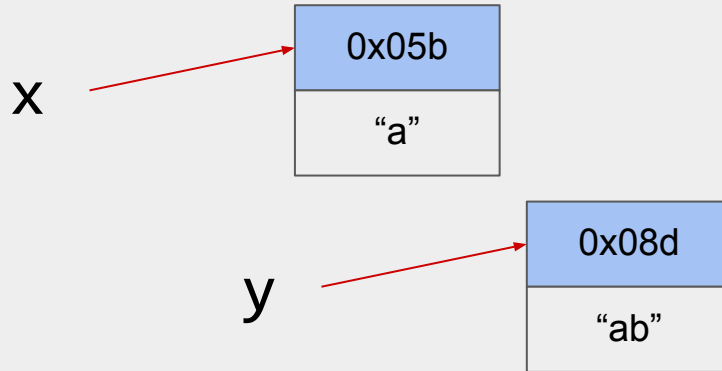

```
> x = "a"  
> y = x  
> y = y + "b"  
> id(x) == id(y)  
>> False
```



```
> x = "a"  
> y = x  
> y = y + "b"  
> id(x) == id(y)  
>> False
```



```
> x = "a"  
> y = x  
> y = y + "b"  
> id(x) == id(y)  
>> False
```



Quiz

```
s = {1, 2, 3}
r = s
s.add(0)
print(len(s) == len(r))

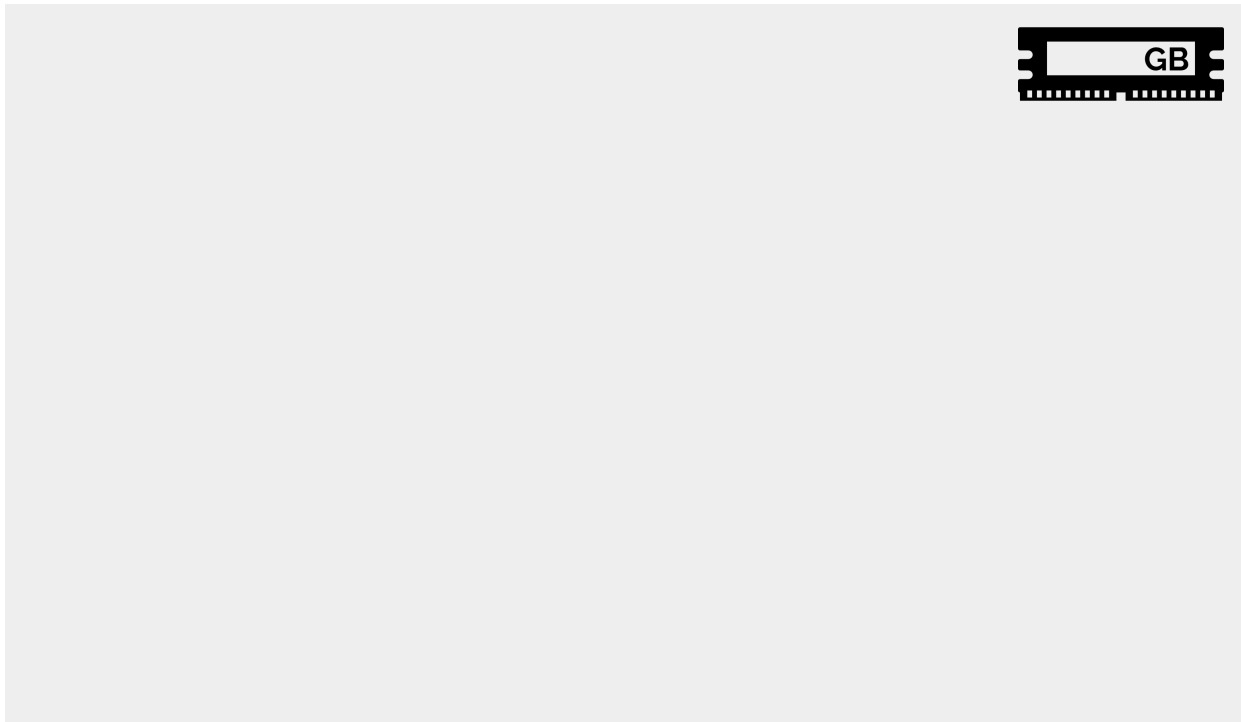
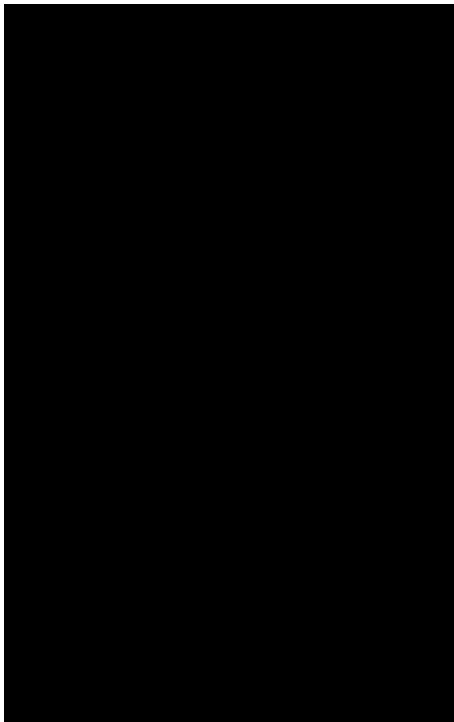
>> True
```

Are sets in Python immutable or mutable?

Python data types

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

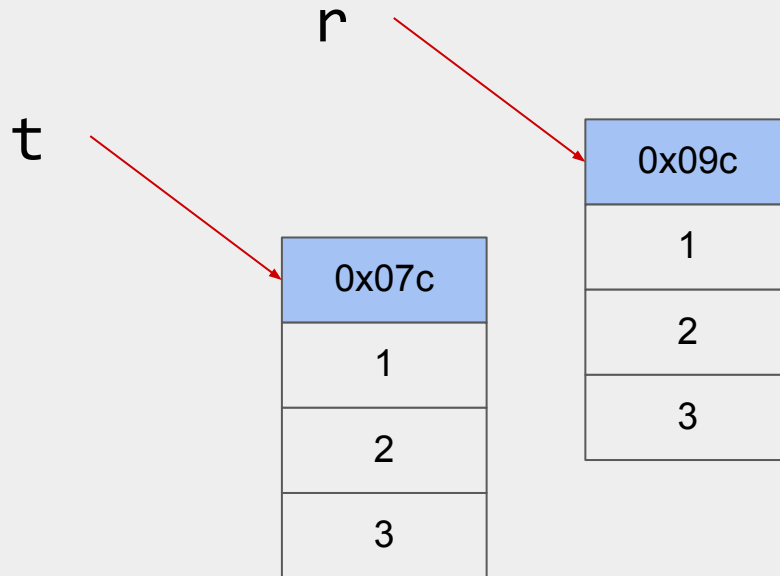
“is” versus “==”



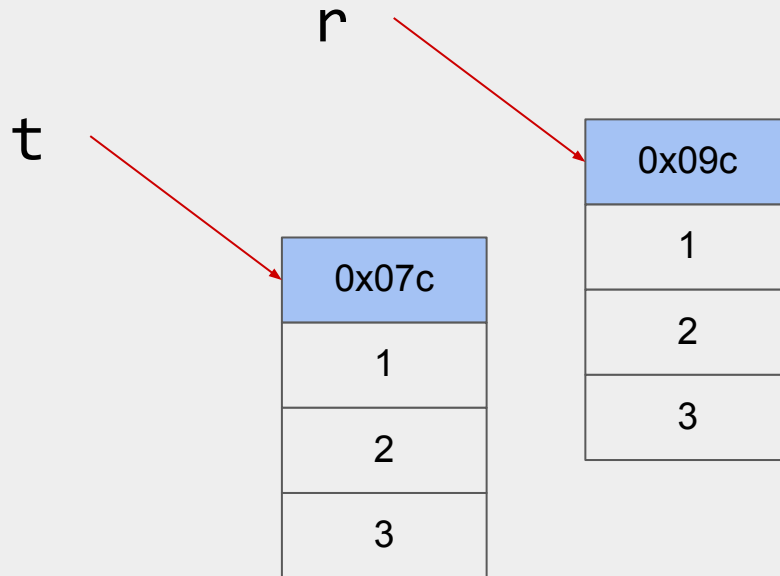

```
> t = (1, 2, 3)  
> r = (1, 2, 3)
```



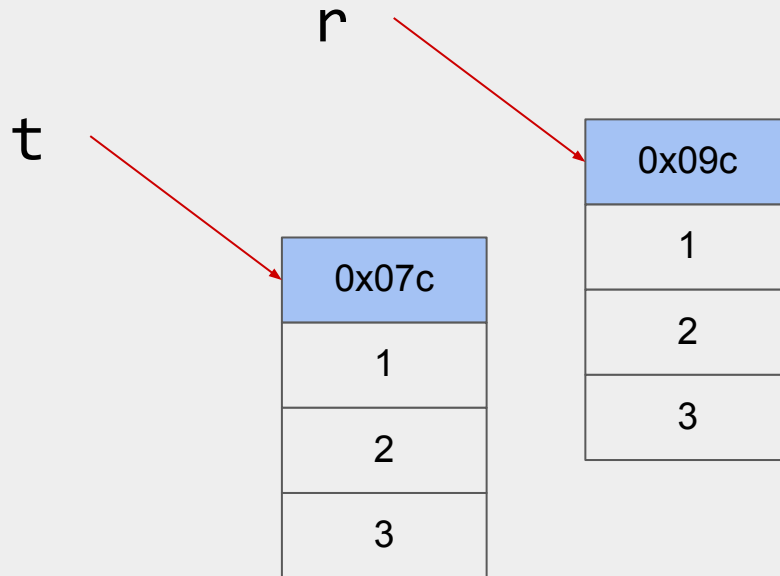
```
> t = (1, 2, 3)
> r = (1, 2, 3)
```



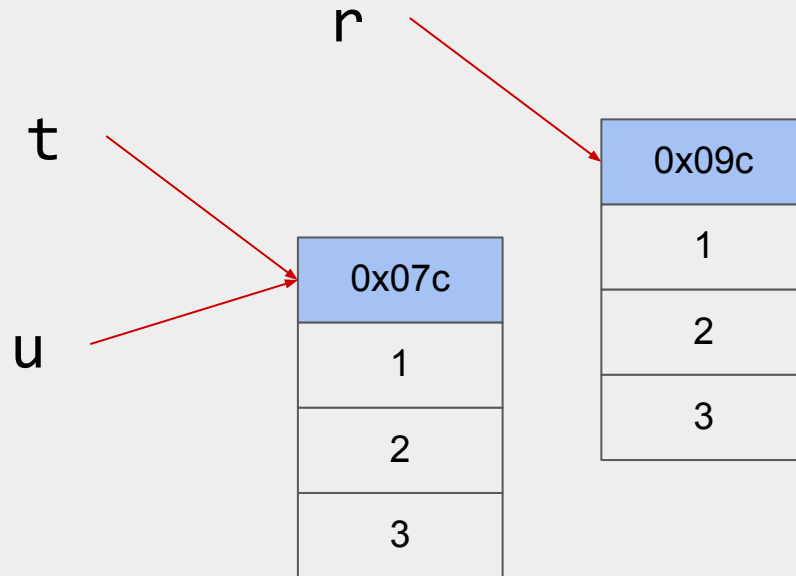
```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
```



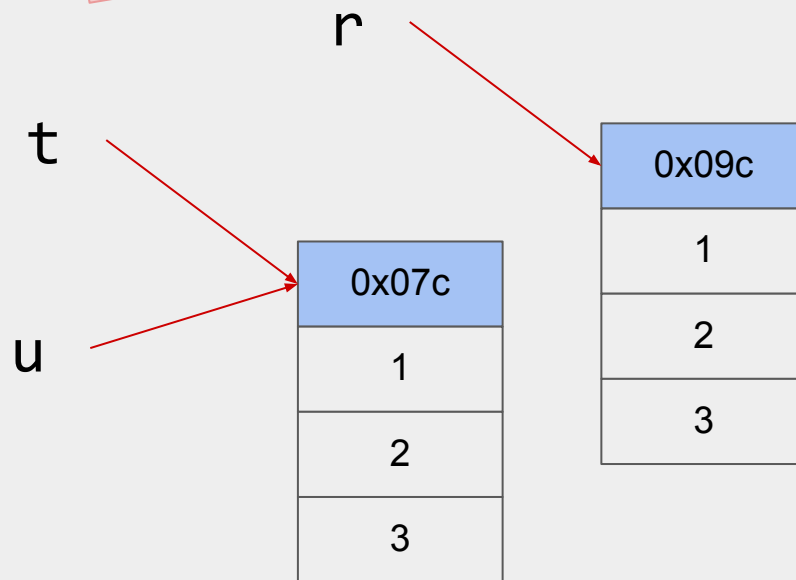
```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
```



```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
```



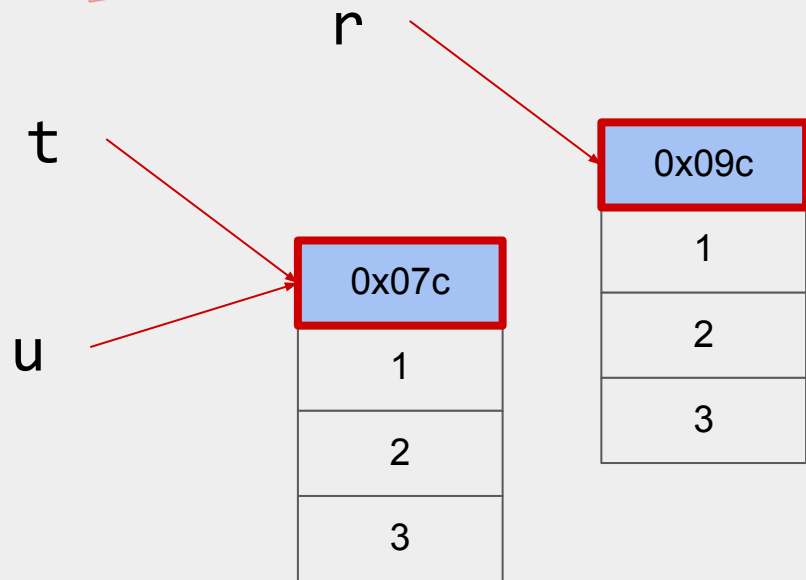
```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
> t is r
```



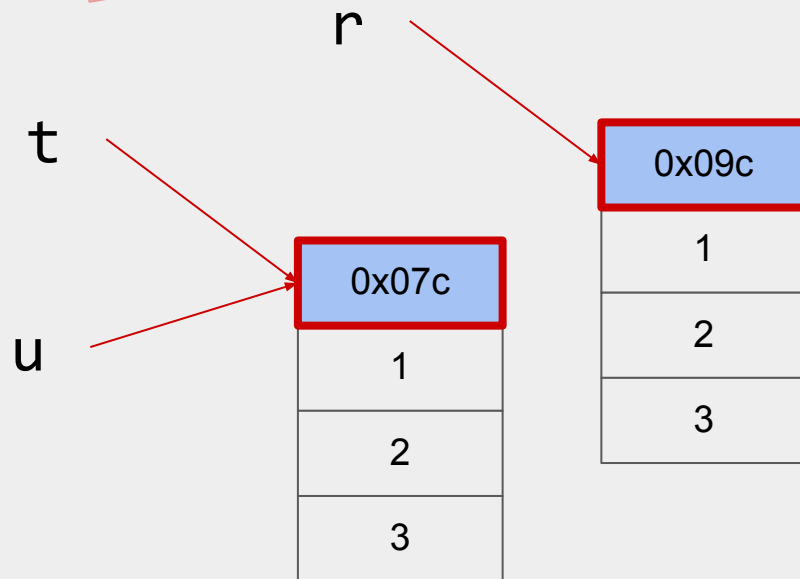
“is” operator check if the memory location of two variables is the same



```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
> t is r
```



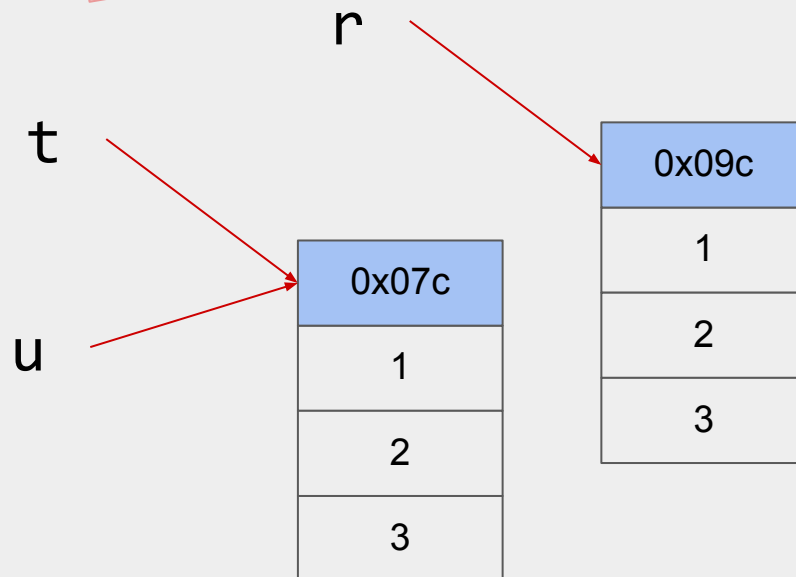
```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
> t is r
>> False
```



“is” operator check if the memory location of two variables is the same



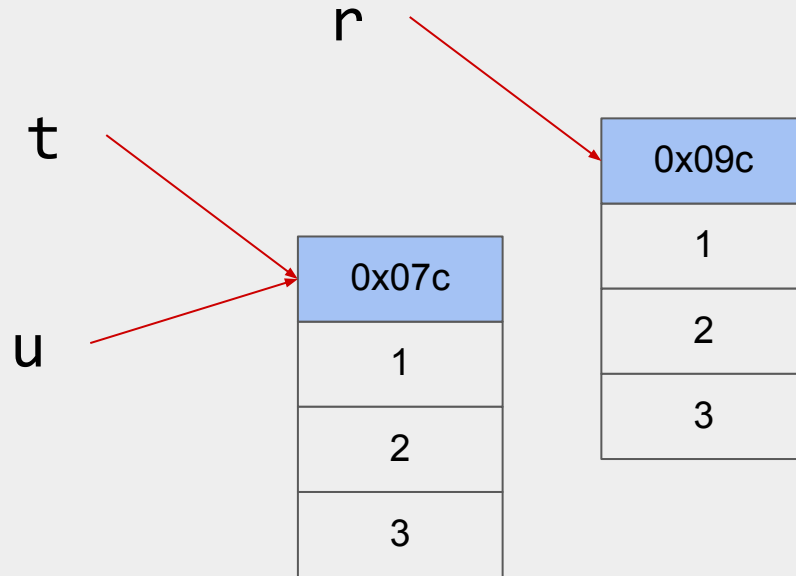

```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
> t is r
>> False
```



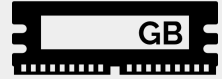
“is” operator check if the memory location of two variables is the same



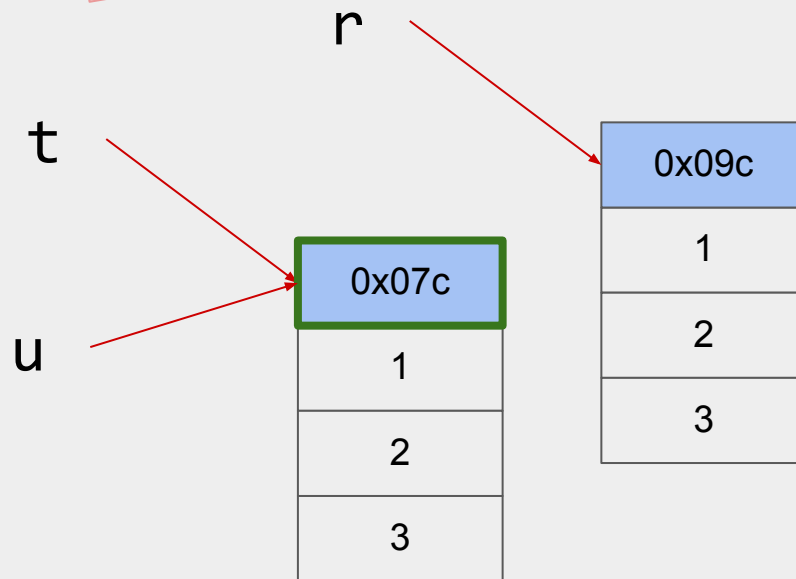
```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
> t is r
>> False
> t is u
```



“is” operator check if the memory location of two variables is the same



```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
> t is r
>> False
> t is u
```

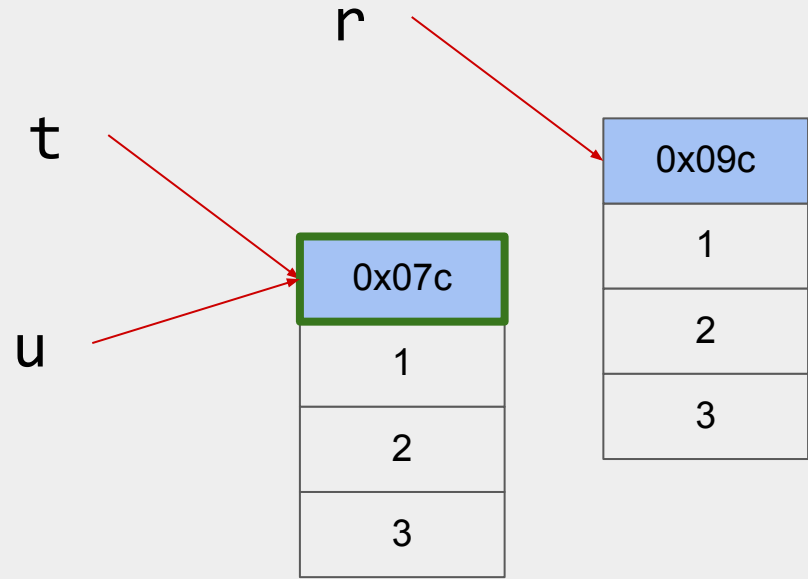


“is” operator check if the memory location of two variables is the same

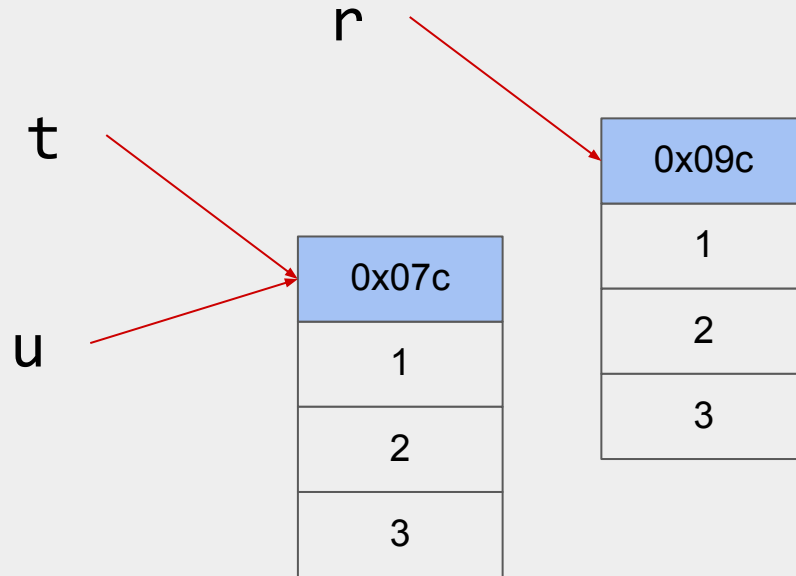


```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
> t is r
>> False
> t is u
>> True
```

“is” operator check if the memory location of two variables is the same



```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
> t is r
>> False
> t is u
>> True
```

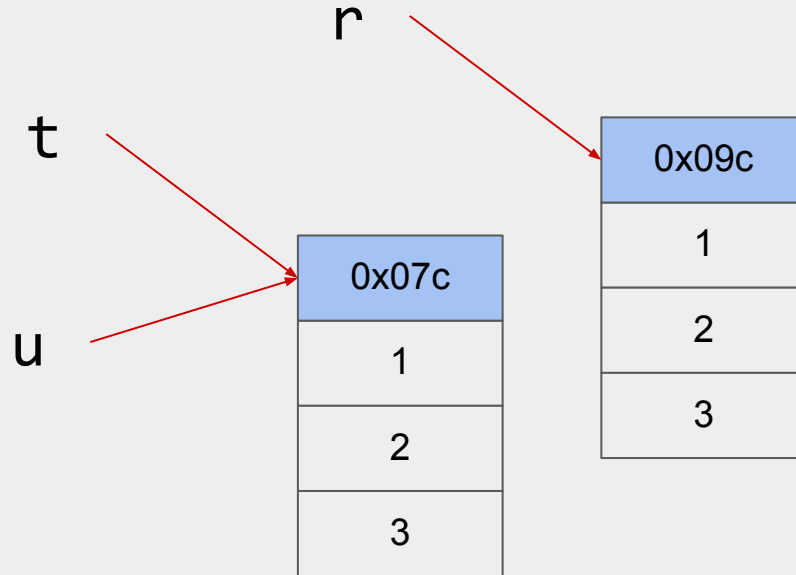


“is” operator check if the memory location of two variables is the same



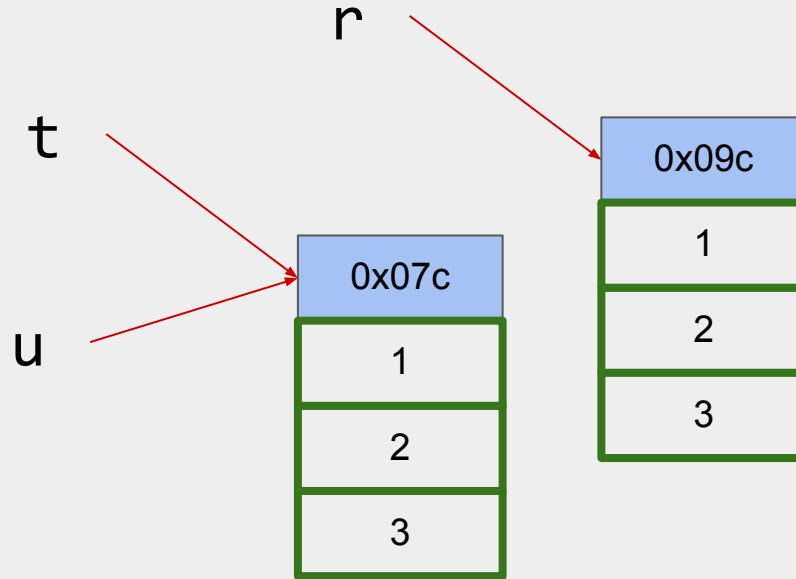
```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
> t is r
>> False
> t is u
>> True
> t == r
```

“==” operator check if the two variables refer to the same values



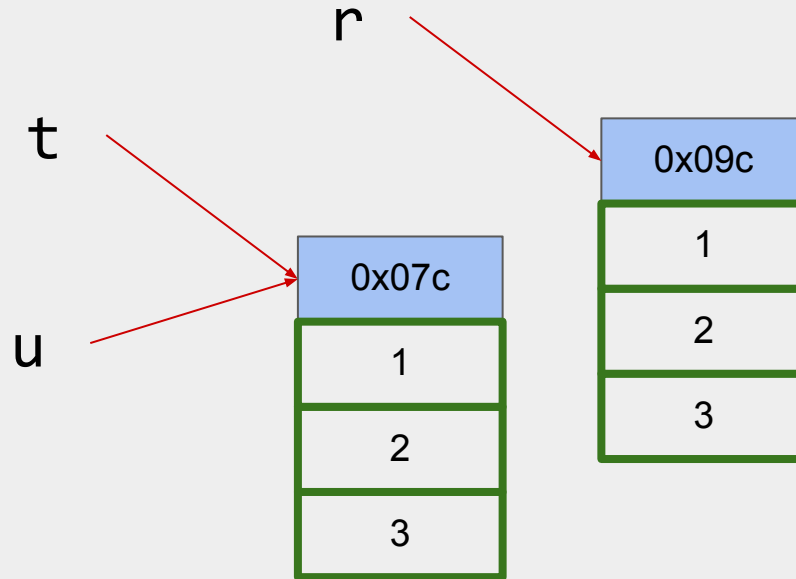
```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
> t is r
>> False
> t is u
>> True
> t == r
```

“==” operator check if the two variables refer to the same values

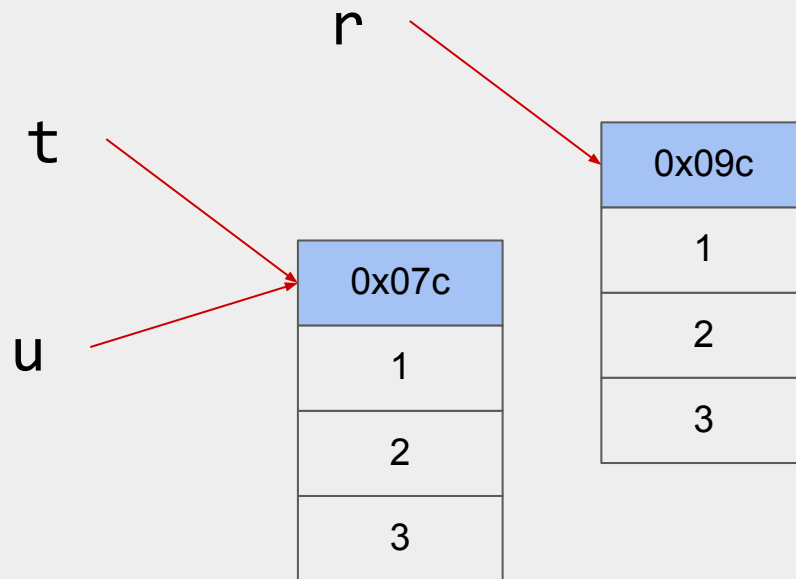


```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
> t is r
>> False
> t is u
>> True
> t == r
>> True
```

“==” operator check if the two variables refer to the same values




```
> t = (1, 2, 3)
> r = (1, 2, 3)
> u = t
> t is r
>> False
> t is u
>> True
> t == r
>> True
```



Quiz

Which of the following sentences are true:

For two Python variables a and b:

- a. if `a == b` evaluates to True, `a is b` will also be True
- b. if `a == b` evaluates to True, `a is b` may or may not be True
- c. if `a is b` evaluates to True, `a == b` will be True
- d. if `a is b` evaluates to True, `a == b` may or may not be True

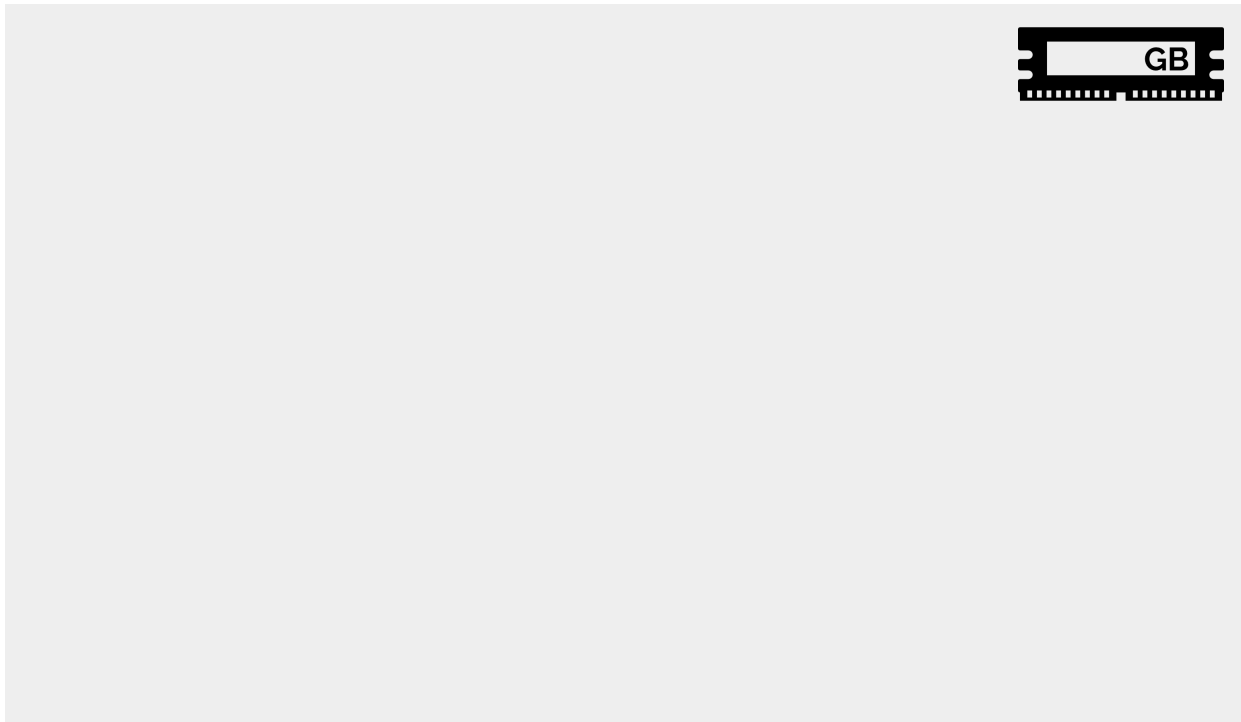
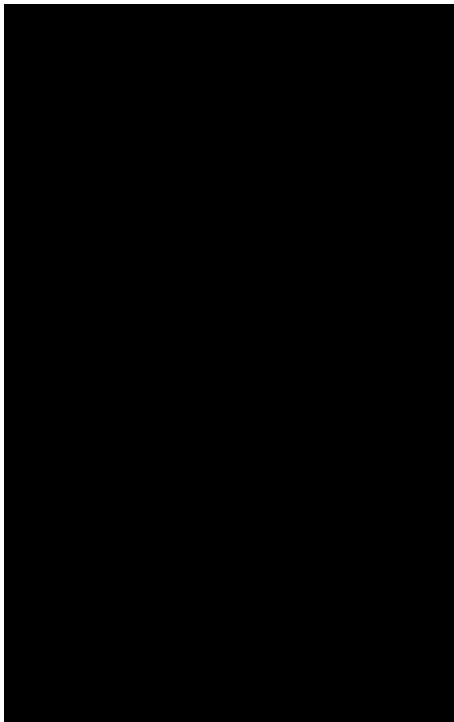
Quiz

Which of the following sentences are true:

For two Python variables a and b:

- a. if `a == b` evaluates to True, `a is b` will also be True
- b. if `a == b` evaluates to True, `a is b` may or may not be True
- c. if `a is b` evaluates to True, `a == b` will be True
- d. if `a is b` evaluates to True, `a == b` may or may not be True

pass by reference



```
> def fun(l):  
..  l.append(0)  
..  return l
```



```
> def fun(1):  
..  l.append(0)  
..  return l
```


fun



0x7f8
fun.__code__
fun.__name__
...

```
> def fun(1):  
..  l.append(0)  
..  return l  
> h = [1, 2, 3]
```

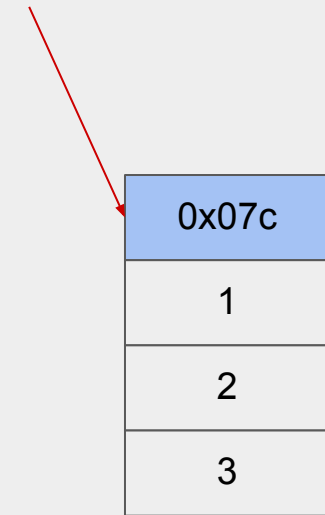
fun



0x7f8
fun.__code__
fun.__name__
...

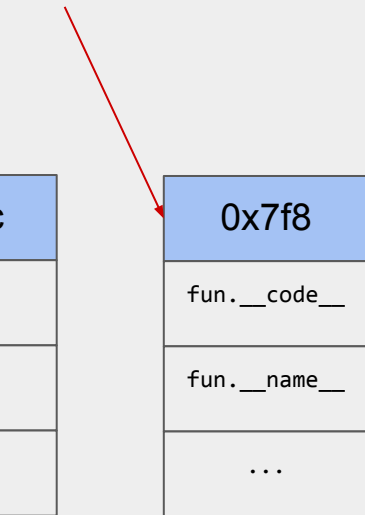

```
> def fun(1):  
..  l.append(0)  
..  return l  
> h = [1, 2, 3]
```

h



0x07c
1
2
3

fun

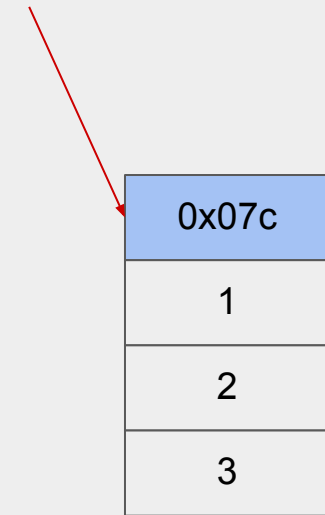


0x7f8
fun.__code__
fun.__name__
...



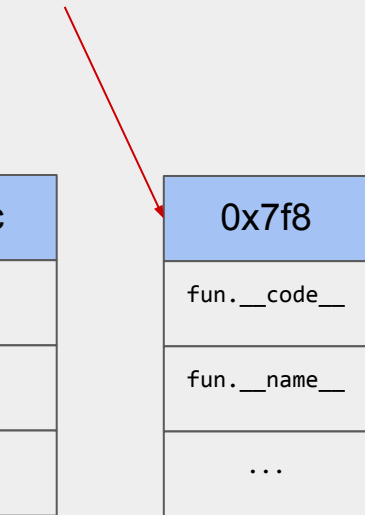
```
> def fun(1):  
.. 1.append(0)  
.. return 1  
> h = [1, 2, 3]  
> g = fun(h)
```

h



0x07c
1
2
3

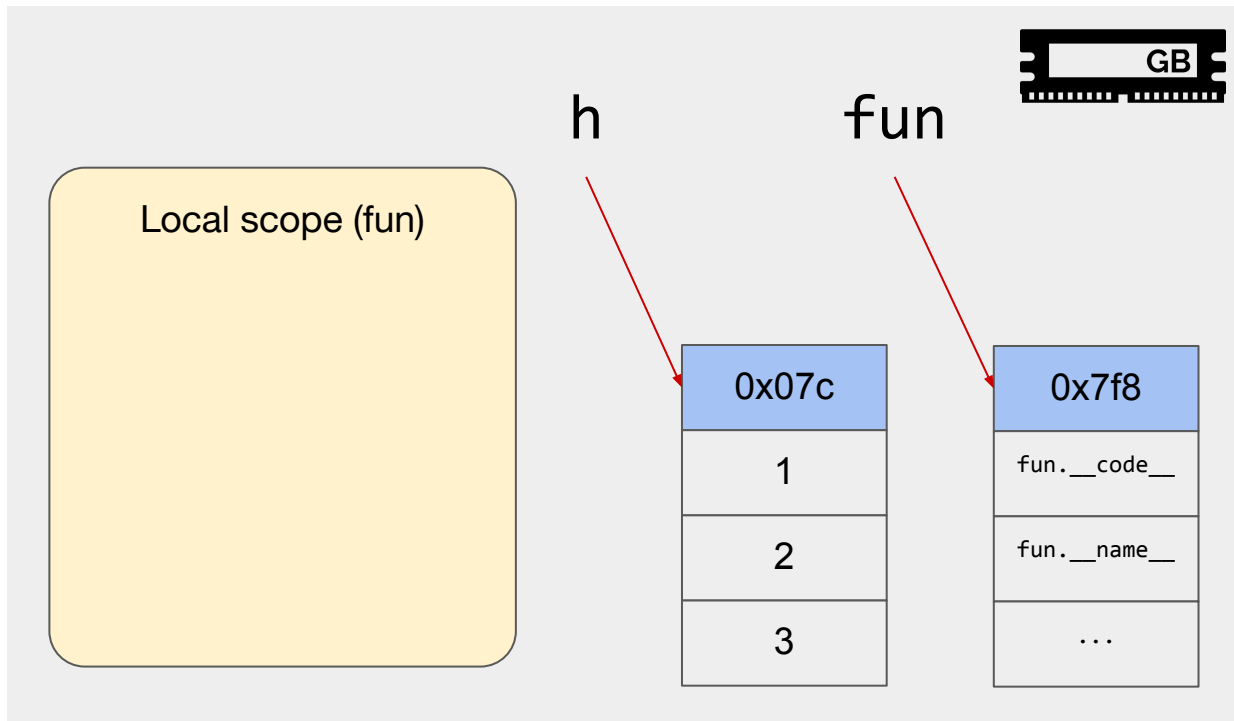
fun



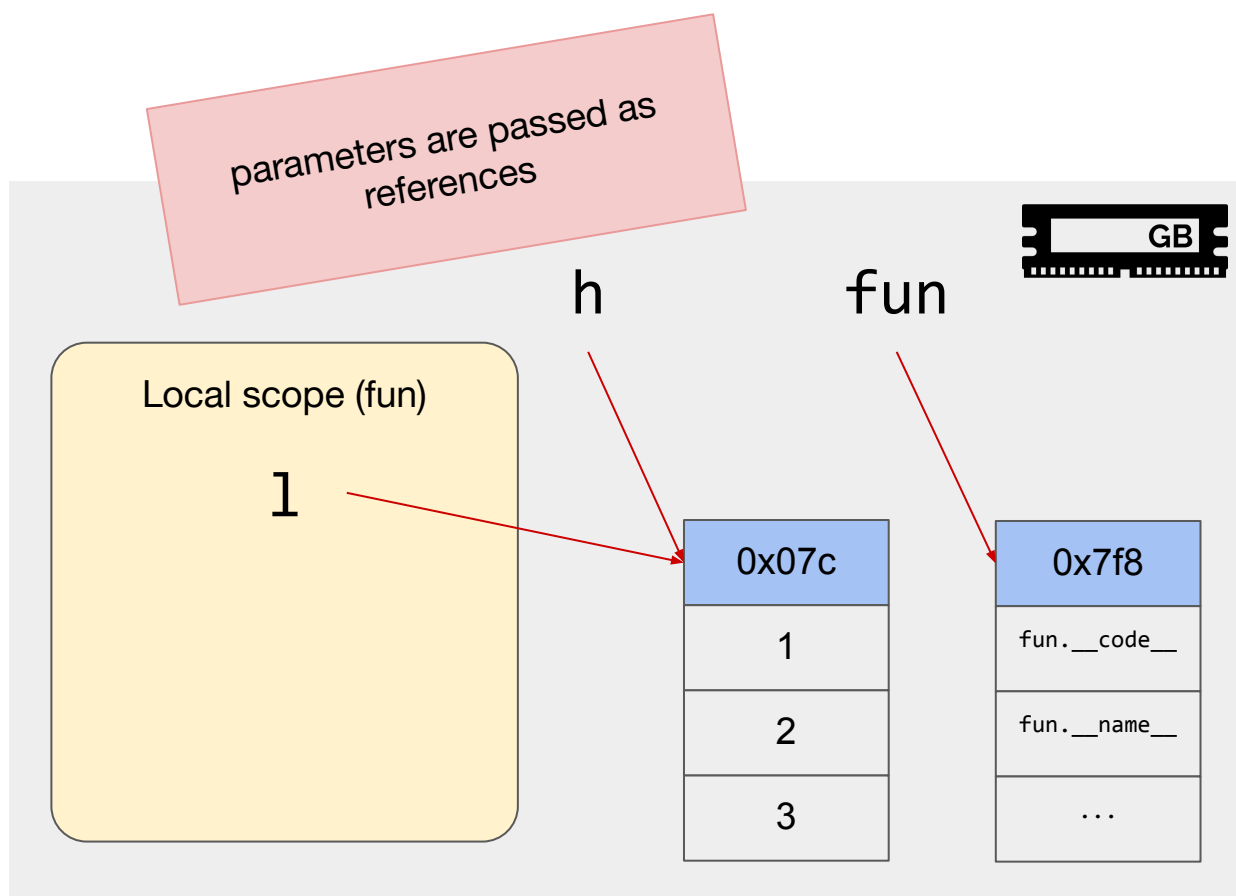
0x7f8
fun.__code__
fun.__name__
...



```
> def fun(1):  
.. 1.append(0)  
.. return 1  
> h = [1, 2, 3]  
> g = fun(h)
```

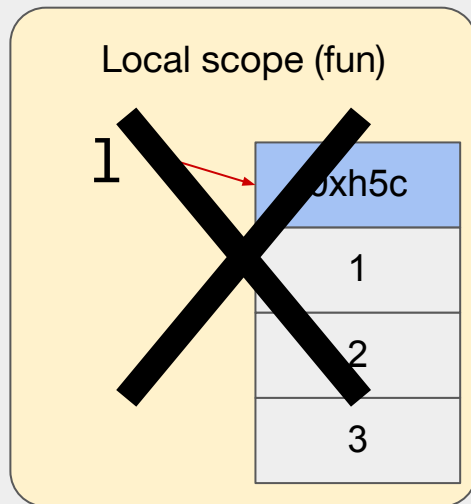


```
> def fun(1):  
.. 1.append(0)  
.. return 1  
> h = [1, 2, 3]  
> g = fun(h)
```

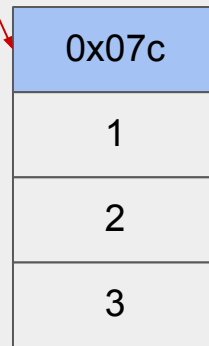


```
> def fun(1):  
.. 1.append(0)  
..  return 1  
> h = [1, 2, 3]  
> g = fun(h)
```

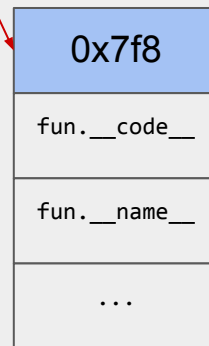
... and not like values



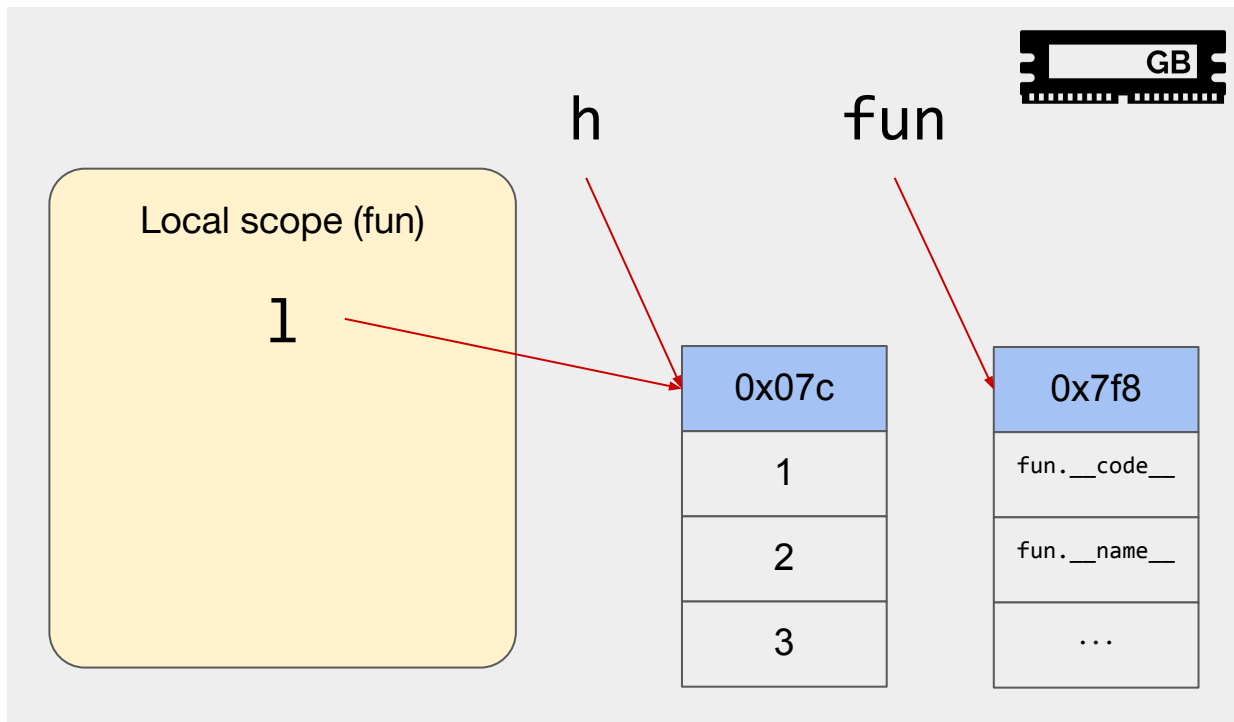
h



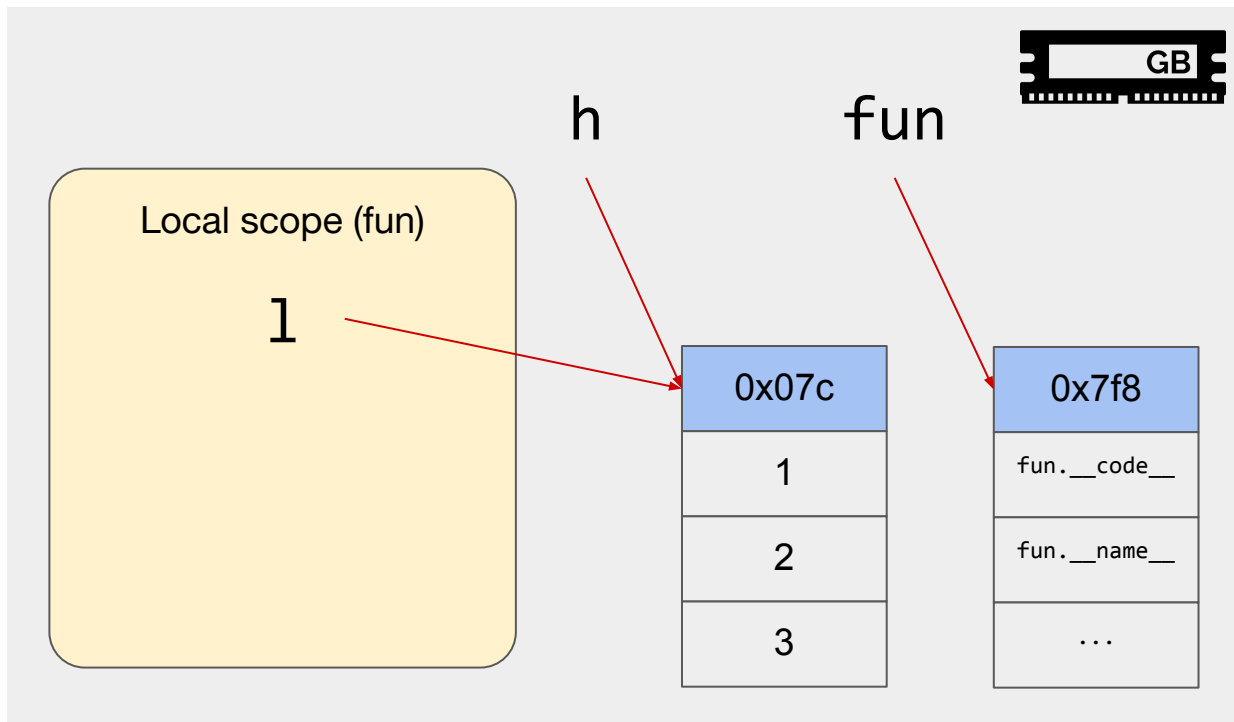
fun



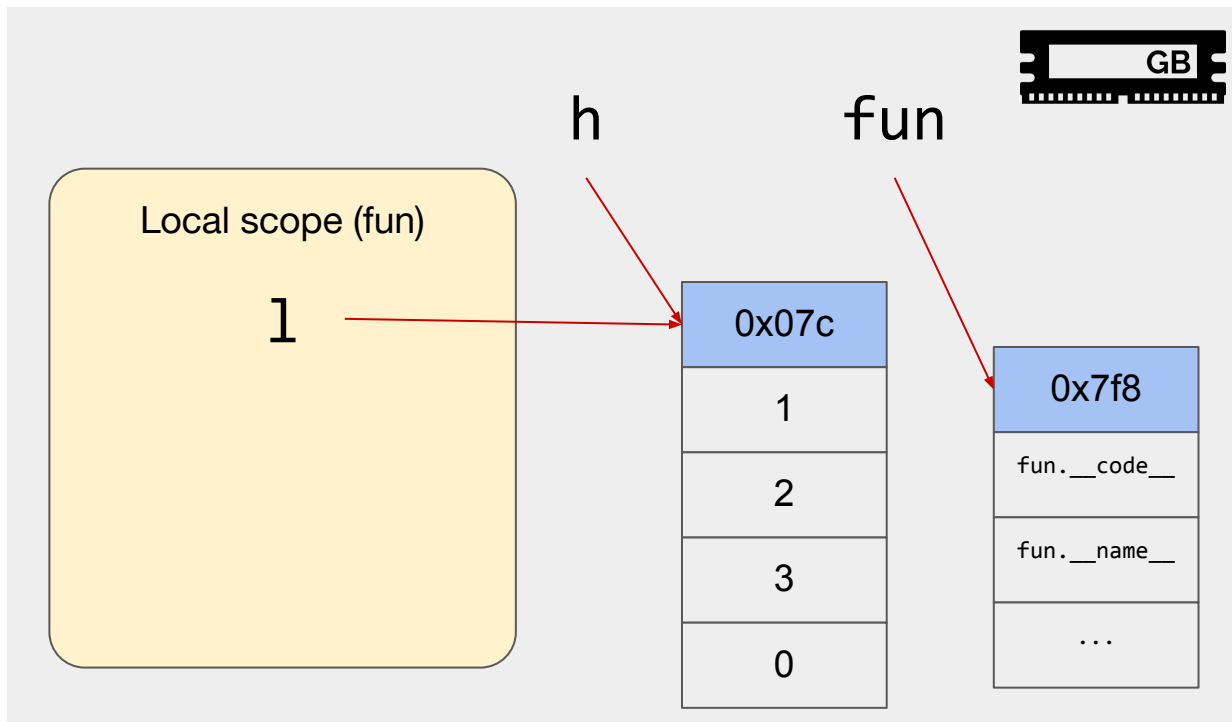
```
> def fun(1):  
.. 1.append(0)  
.. return 1  
> h = [1, 2, 3]  
> g = fun(h)
```



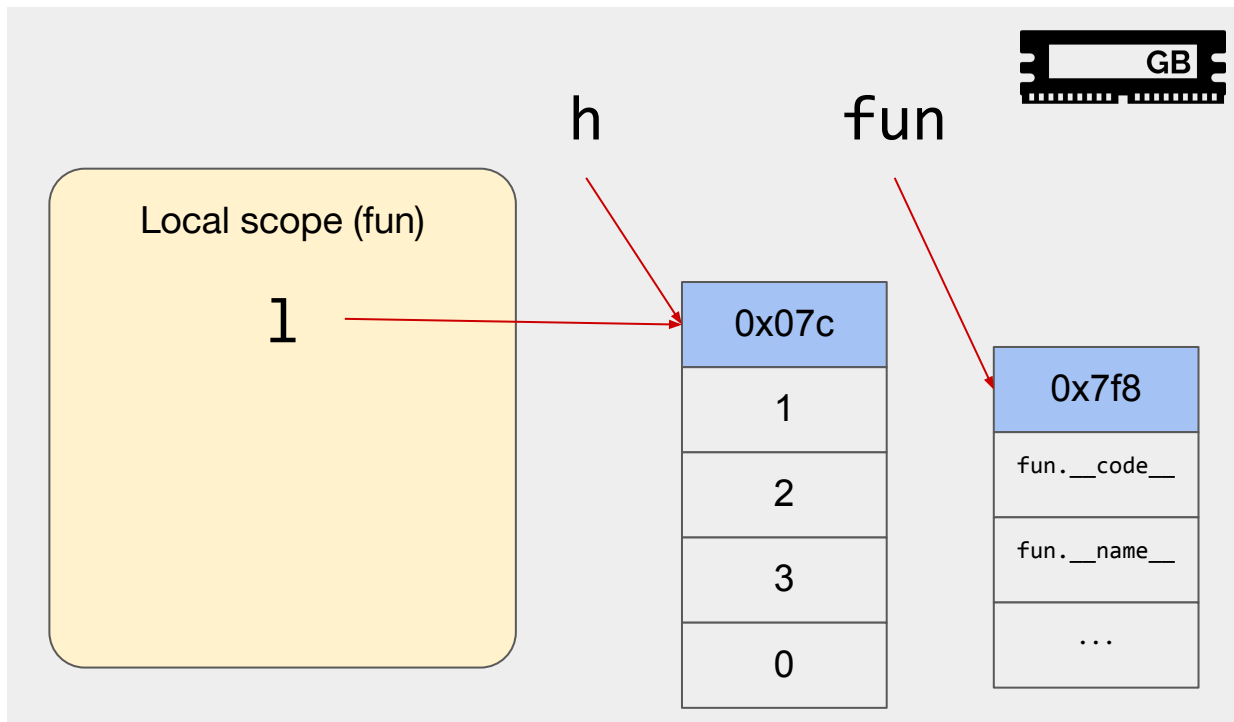
```
> def fun(1):  
.. 1.append(0)  
.. return 1  
> h = [1, 2, 3]  
> g = fun(h)
```



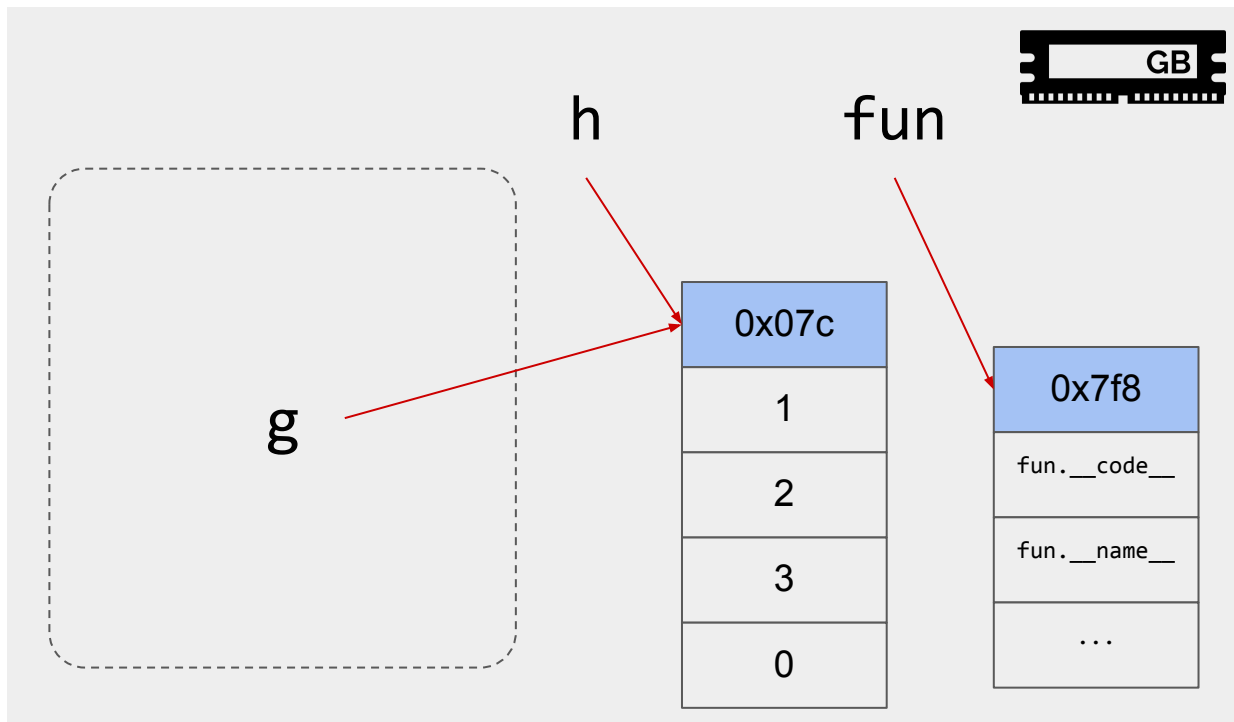
```
> def fun(1):  
.. 1.append(0)  
.. return 1  
> h = [1, 2, 3]  
> g = fun(h)
```



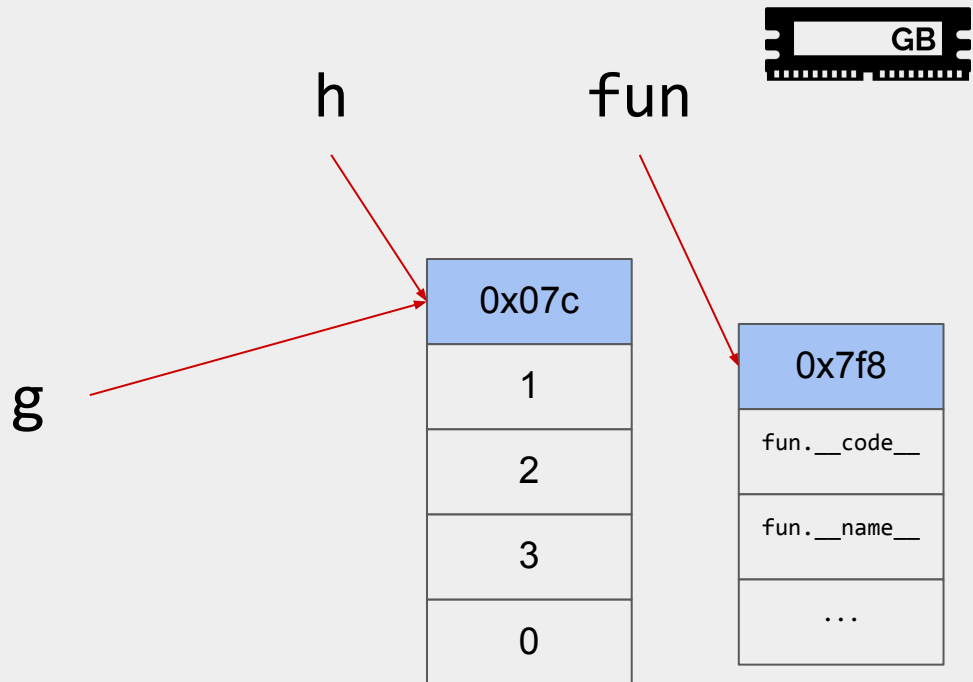

```
> def fun(l):  
..  l.append(0)  
..  return l  
> h = [1, 2, 3]  
> g = fun(h)
```



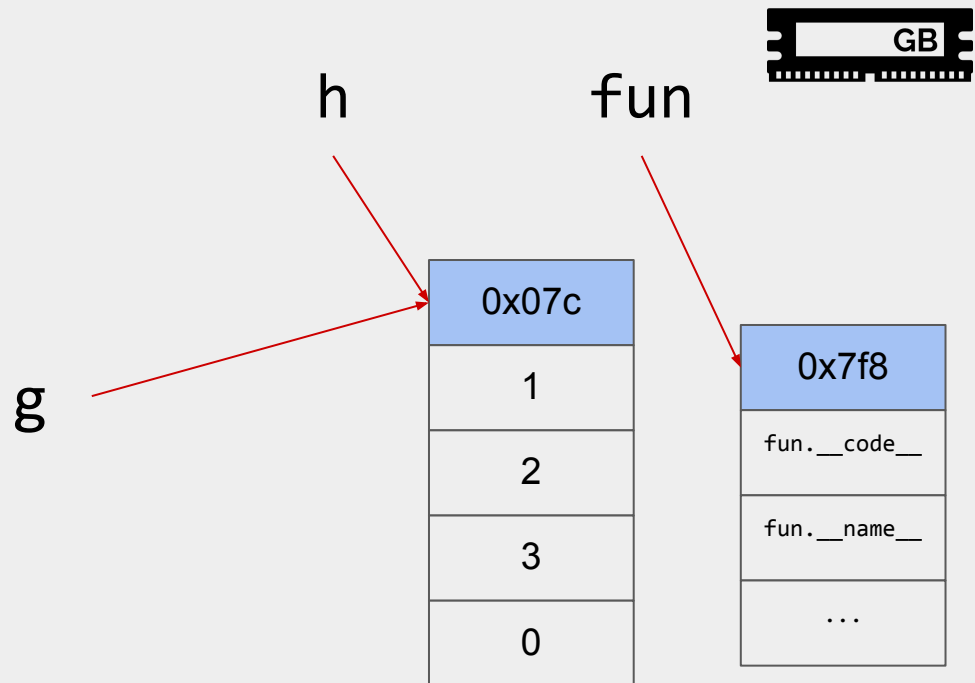
```
> def fun(1):  
.. 1.append(0)  
.. return 1  
> h = [1, 2, 3]  
> g = fun(h)
```



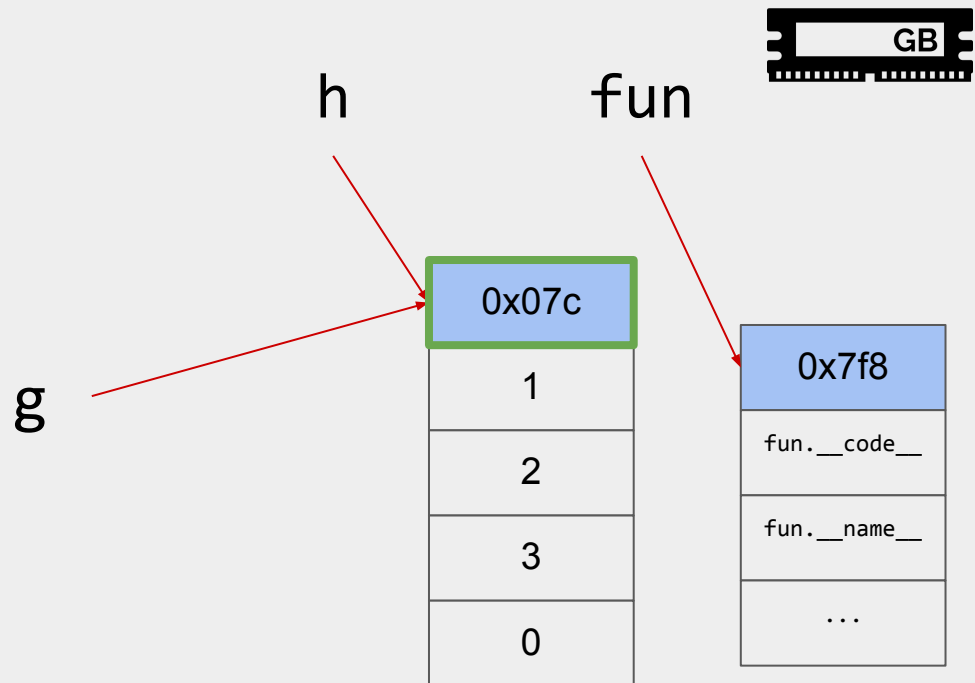
```
> def fun(l):  
.. l.append(0)  
.. return l  
> h = [1, 2, 3]  
> g = fun(h)
```



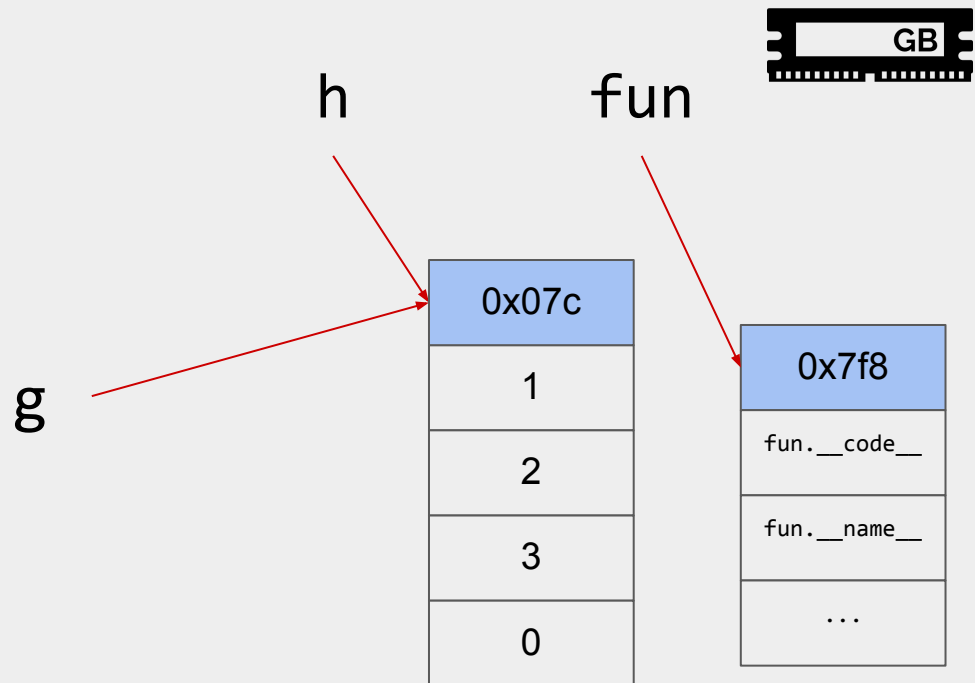
```
> def fun(l):  
..  l.append(0)  
..  return l  
> h = [1, 2, 3]  
> g = fun(h)  
> g is h
```



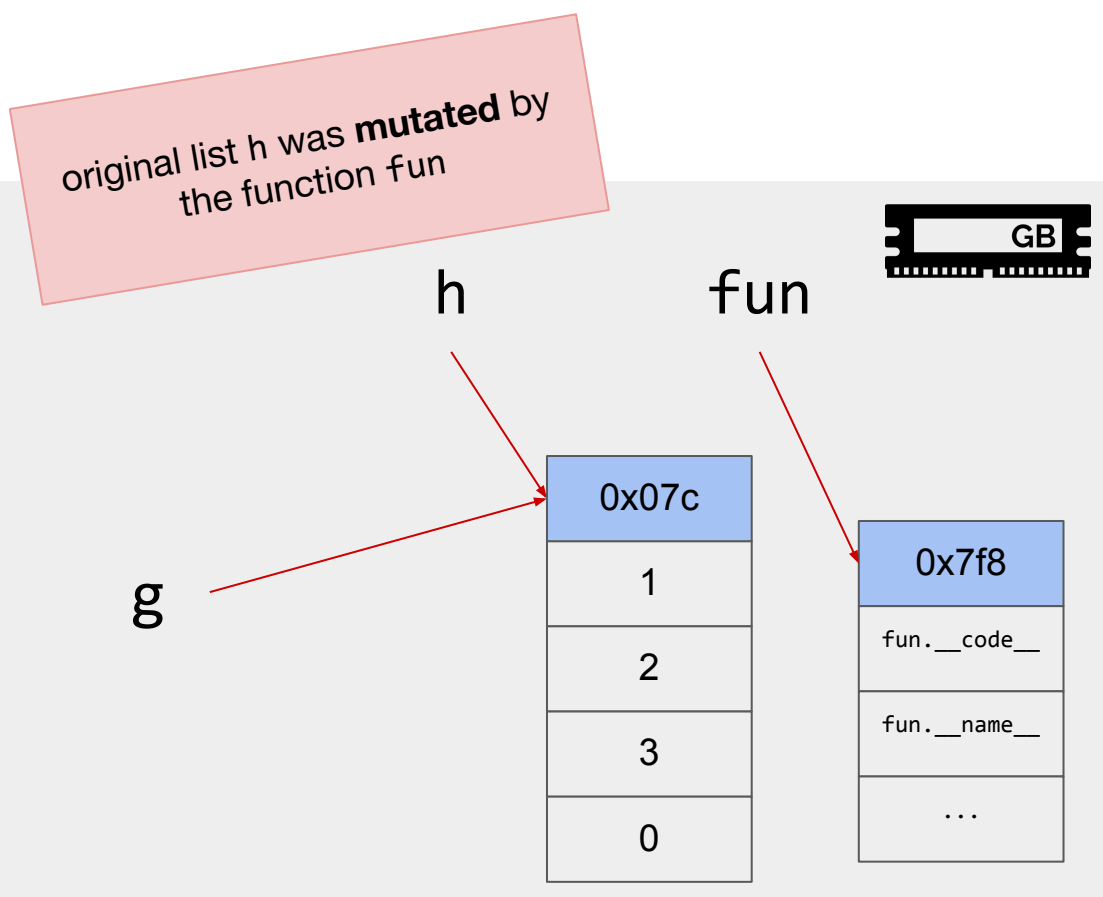
```
> def fun(l):  
.. l.append(0)  
.. return l  
> h = [1, 2, 3]  
> g = fun(h)  
> g is h  
>> True
```



```
> def fun(1):  
.. 1.append(0)  
.. return 1  
> h = [1, 2, 3]  
> g = fun(h)  
> g is h  
>> True  
> h
```



```
> def fun(l):  
.. l.append(0)  
.. return l  
> h = [1, 2, 3]  
> g = fun(h)  
> g is h  
>> True  
> h  
>> [1, 2, 3, 0]
```



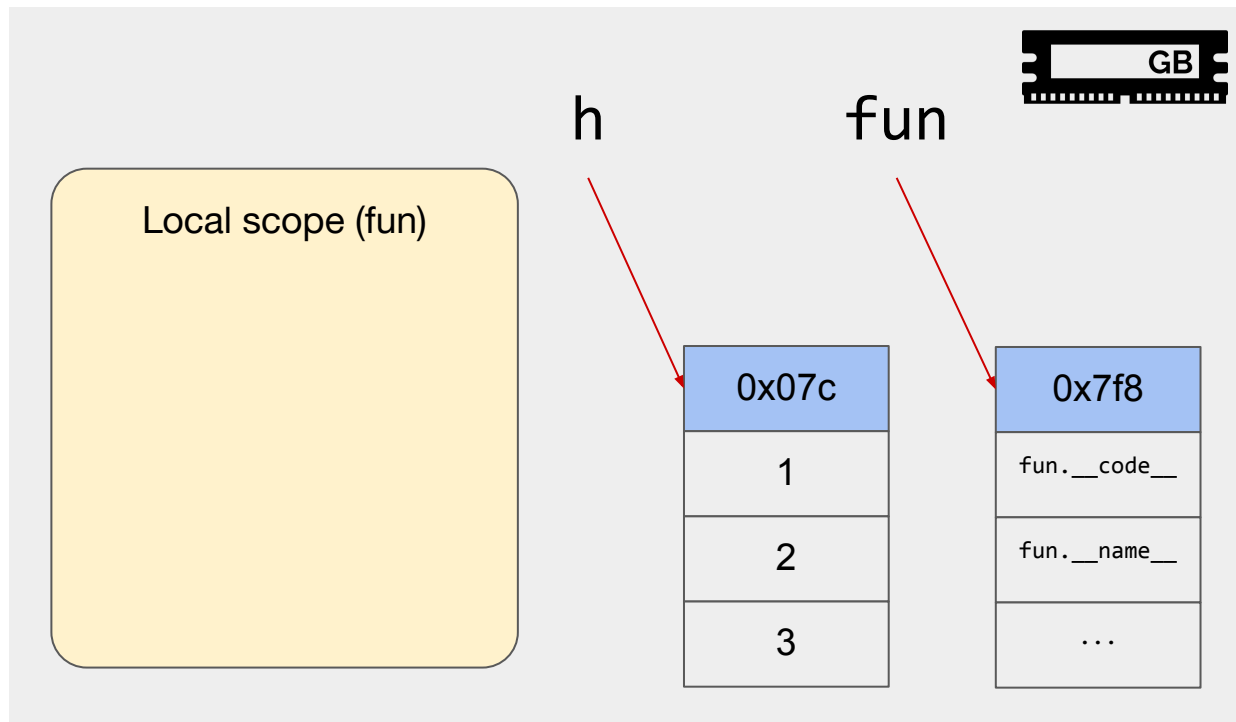
Quiz

```
def fun(t):  
    t = t + (0, )  
    return t
```

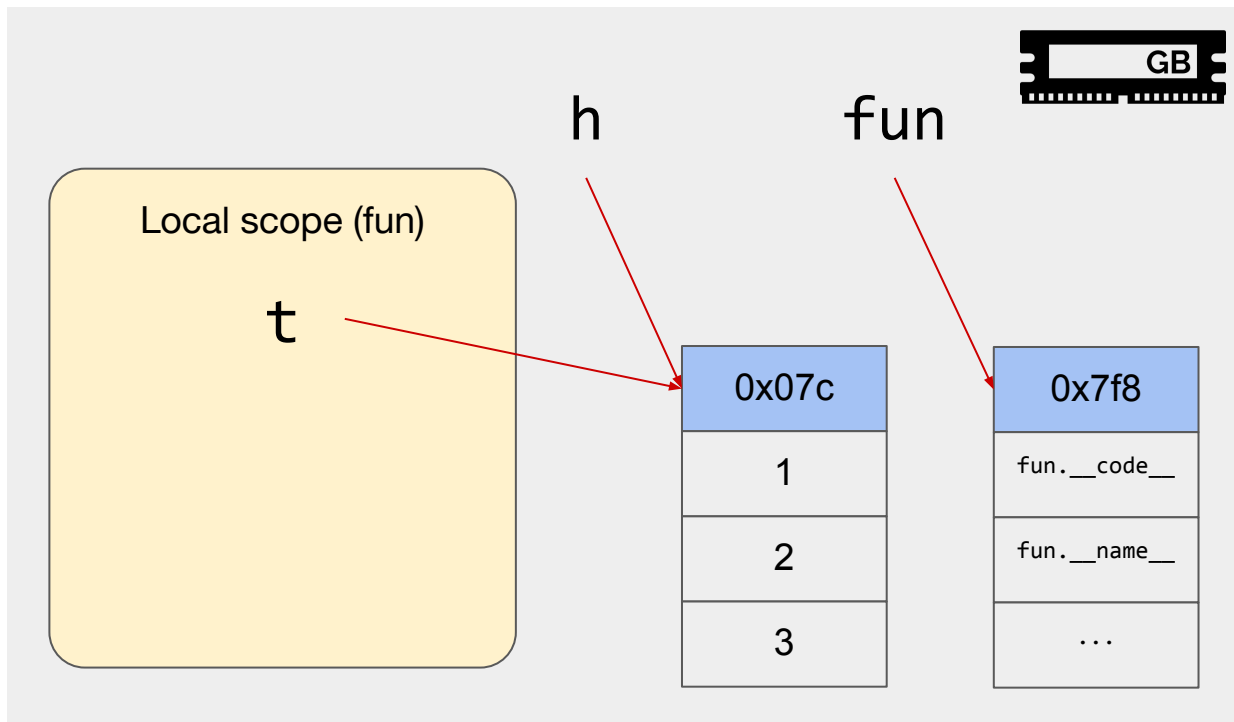
```
h = (1, 2, 3)  
g = fun(h)  
print(h, h is g)
```



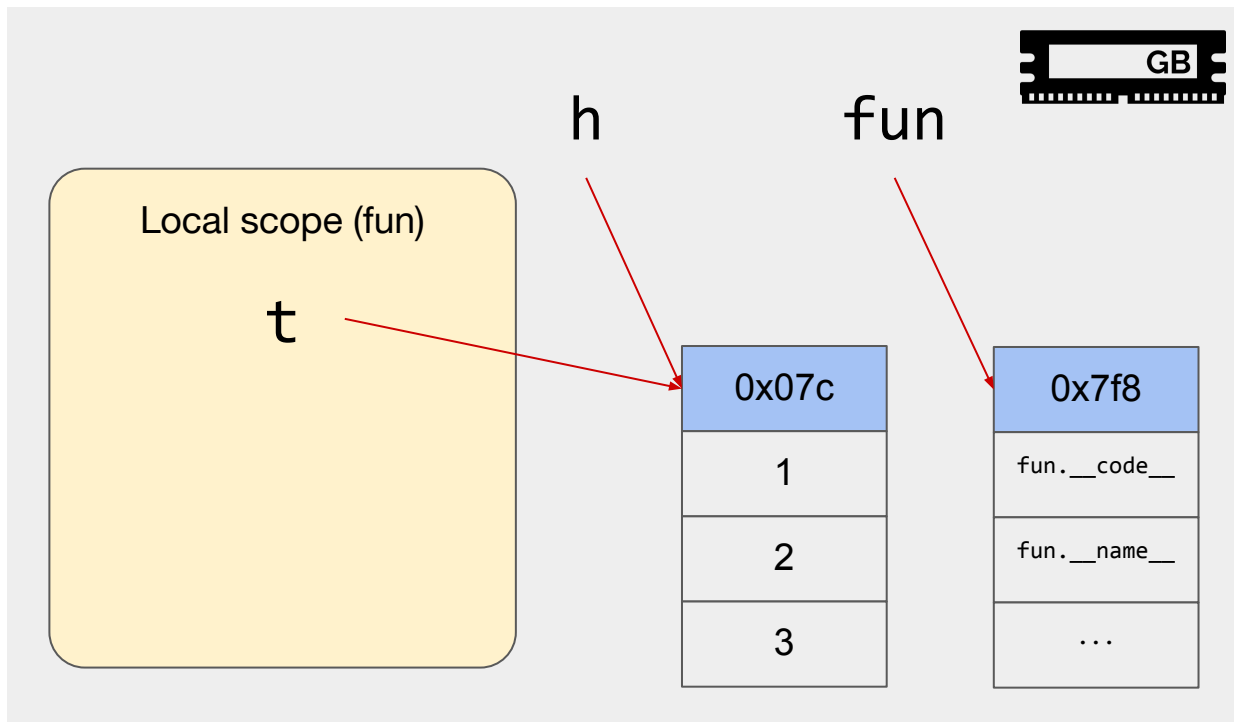
```
> def fun(t):  
.. t = t + (0,)  
.. return t  
> h = (1, 2, 3)  
> g = fun(h)
```



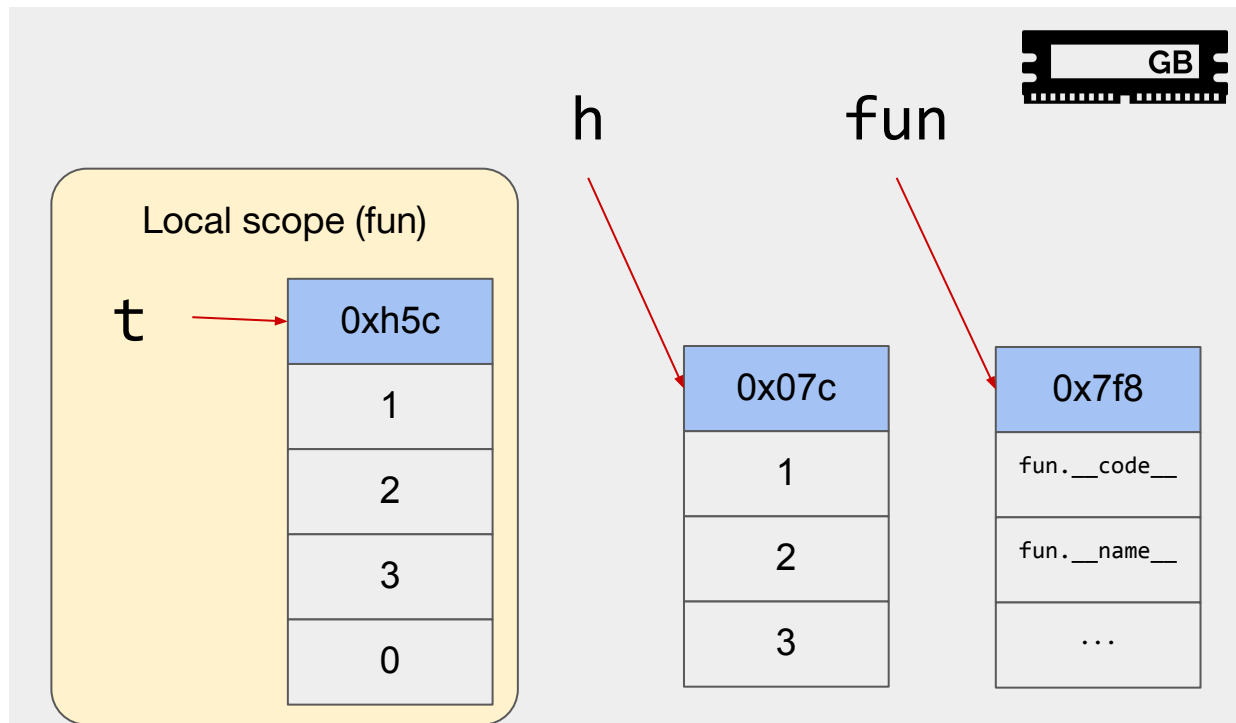
```
> def fun(t):  
.. t = t + (0,)   
.. return t  
> h = (1, 2, 3)  
> g = fun(h)
```



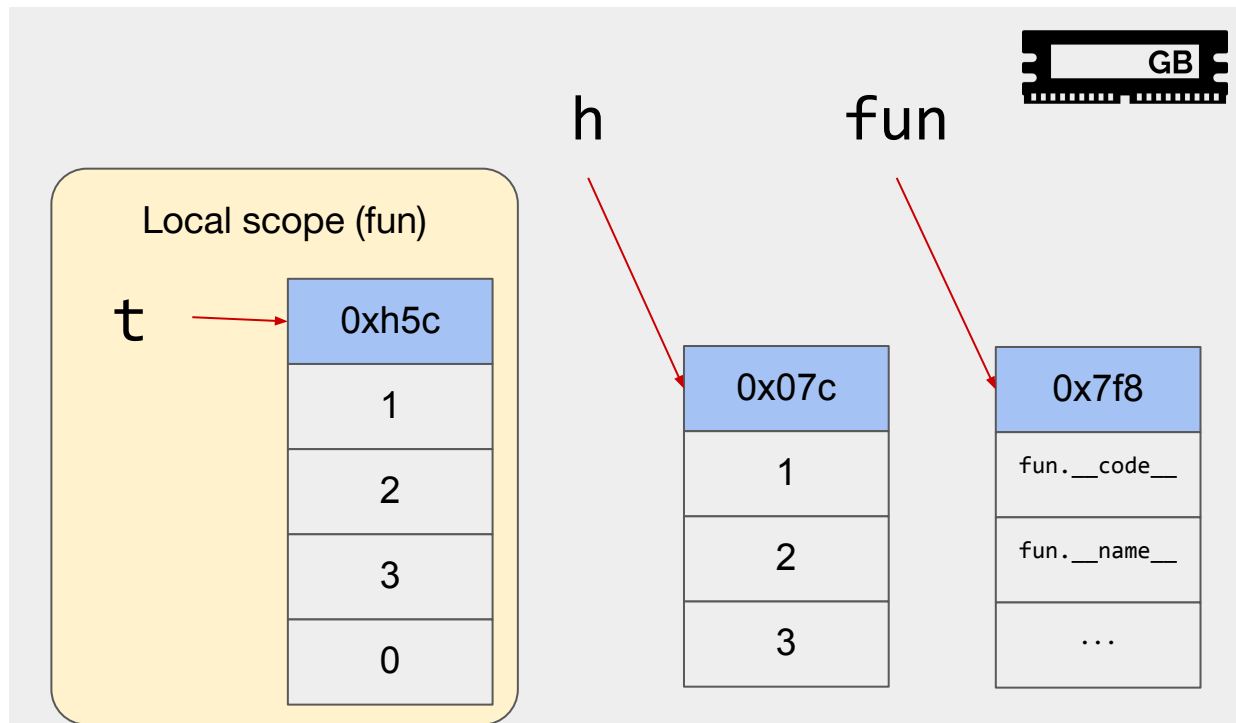
```
> def fun(t):  
.. t = t + (0,)   
.. return t  
> h = (1, 2, 3)  
> g = fun(h)
```



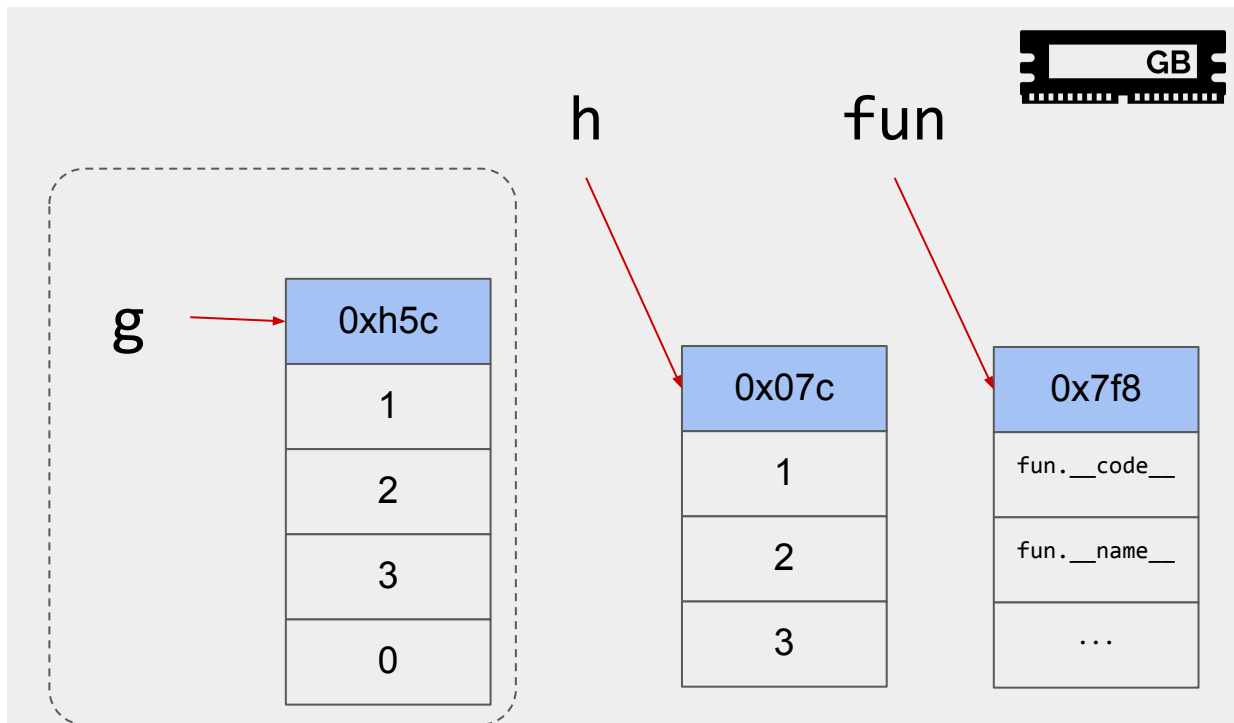
```
> def fun(t):  
.. t = t + (0,)   
.. return t  
> h = (1, 2, 3)  
> g = fun(h)
```



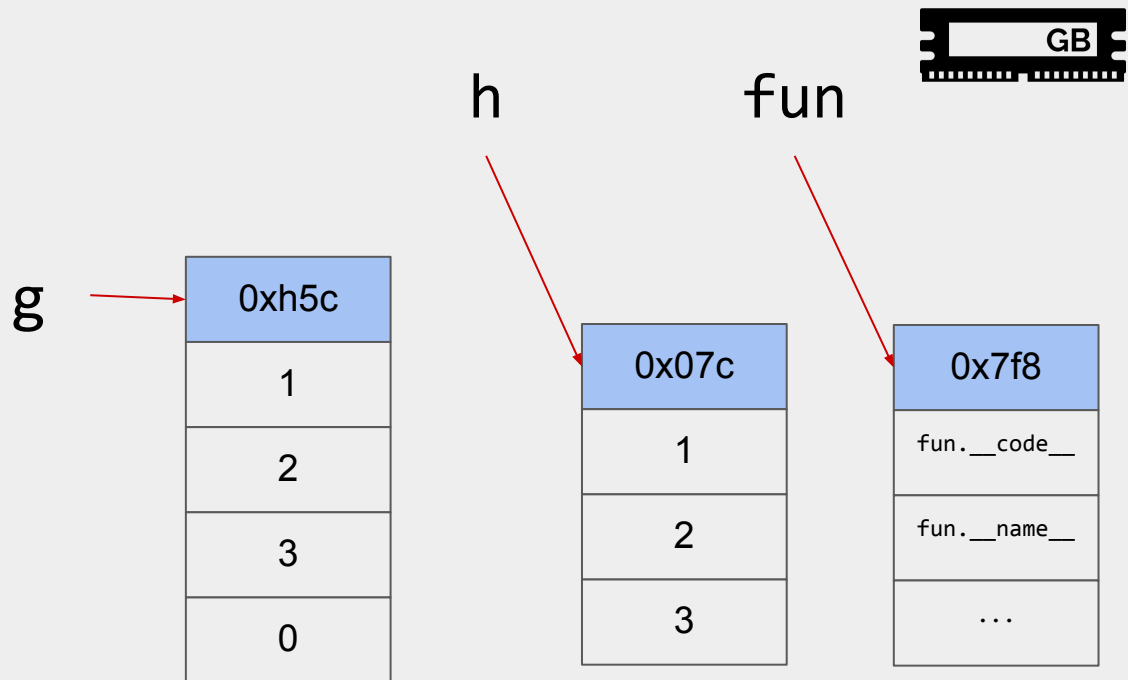
```
> def fun(t):  
.. t = t + (0,)  
.. return t  
> h = (1, 2, 3)  
> g = fun(h)
```



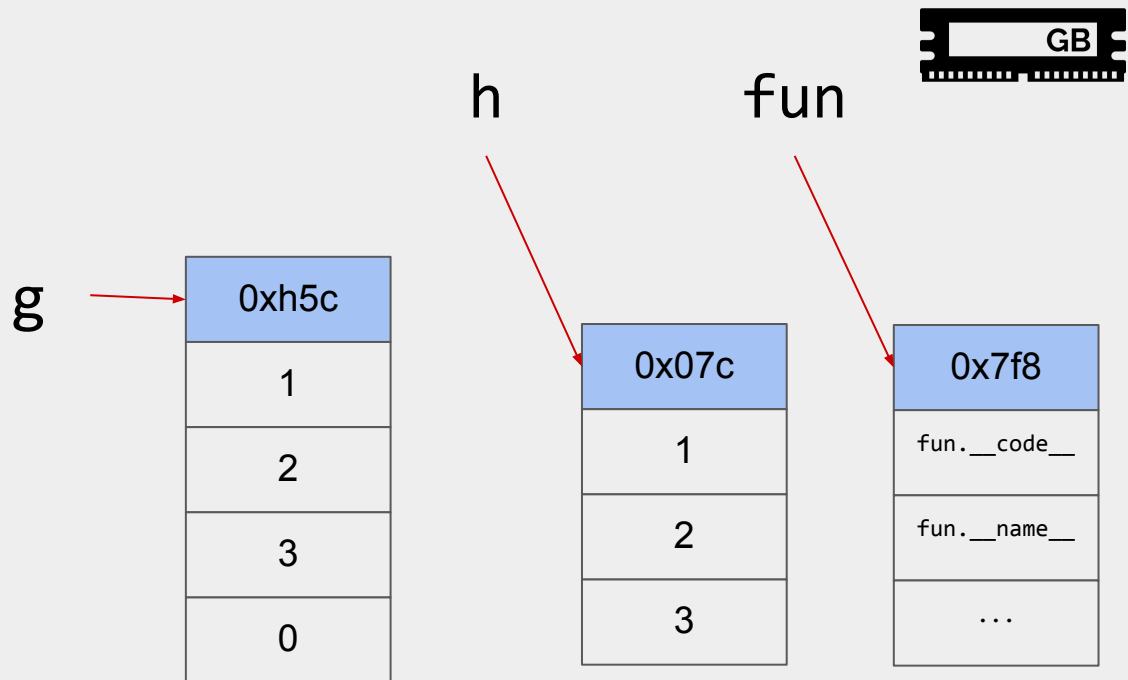
```
> def fun(t):  
.. t = t + (0,)  
.. return t  
> h = (1, 2, 3)  
> g = fun(h)
```



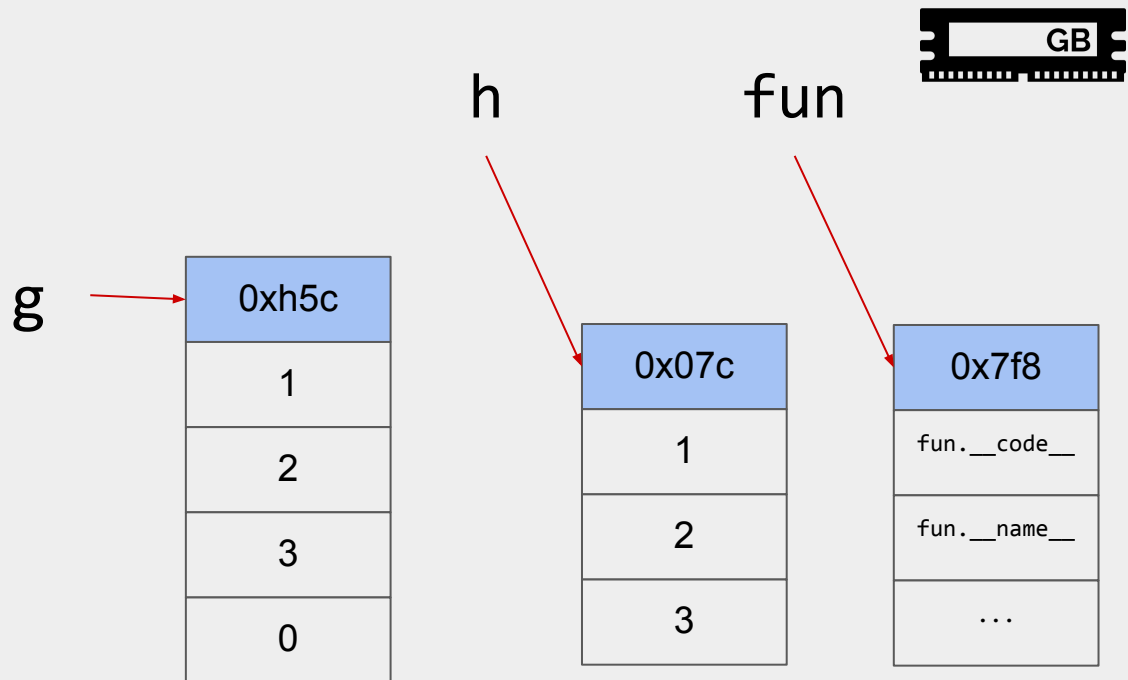
```
> def fun(t):  
.. t = t + (0,)  
.. return t  
> h = (1, 2, 3)  
> g = fun(h)
```



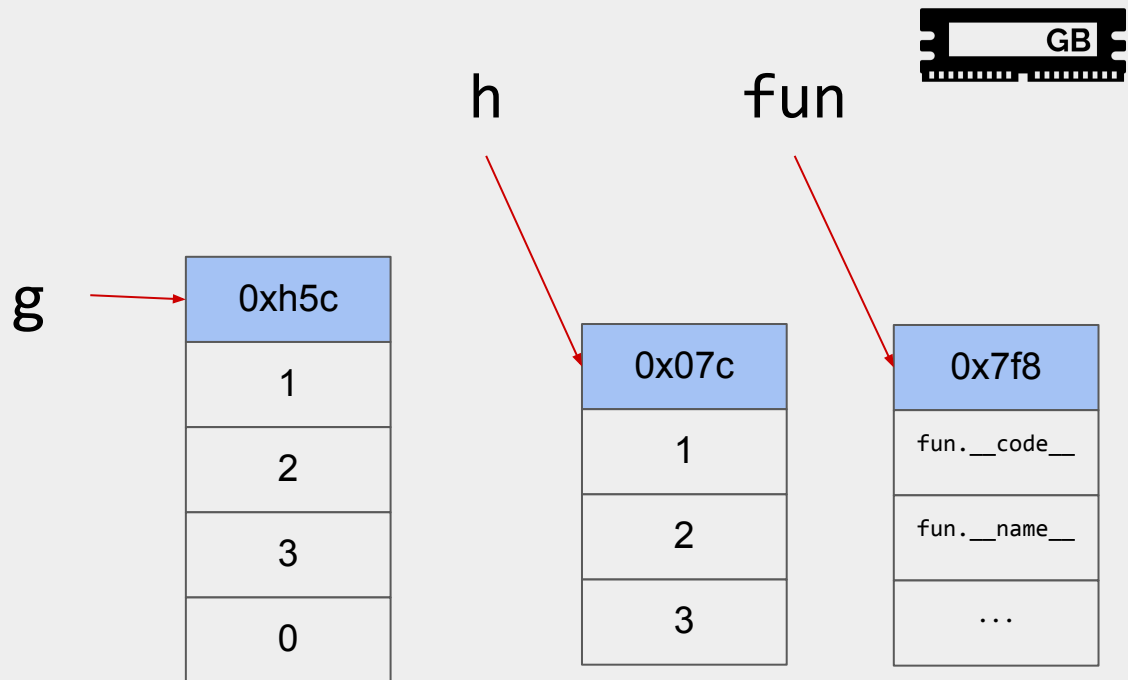
```
> def fun(t):  
.. t = t + (0,)  
.. return t  
> h = (1, 2, 3)  
> g = fun(h)  
> h
```




```
> def fun(t):  
.. t = t + (0,)   
.. return t  
> h = (1, 2, 3)  
> g = fun(h)  
> h  
>> (1, 2, 3)
```



```
> def fun(t):  
.. t = t + (0,)   
.. return t  
> h = (1, 2, 3)  
> g = fun(h)  
> h  
>> (1, 2, 3)  
> h is g
```



```
> def fun(t):  
.. t = t + (0,)  
.. return t  
> h = (1, 2, 3)  
> g = fun(h)  
> h  
>> (1, 2, 3)  
> h is g  
>> False
```

