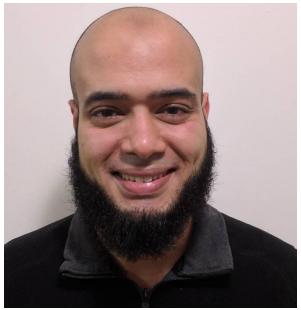
Python Programming

Immutable Objects

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

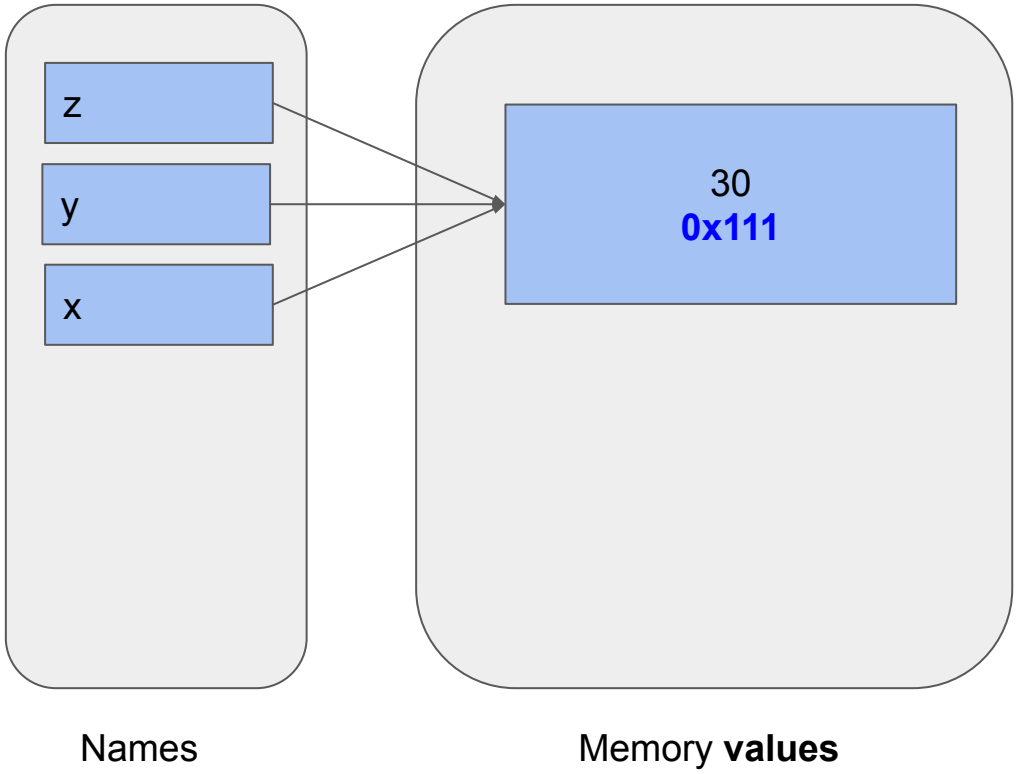


Immutable objects

- Once created in the memory: it **can’t** be changed!
- Many builtin are immutable:
  - int, float, bool, complex, tuple, frozenset, unicode
- Coming from C++?
  - int is not primitive, it is an immutable object
  - No pass by value or by reference. It is all about assigning: mutable or immutable

Memory for immutable objects

```
3   x = 30
4   y = x
5   z = 30
6   print(id(x))    # 0x111
7   print(id(y))    # 0x111
8   print(id(z))    # 0x111
9   print(id(30))   # 0x111
10
```

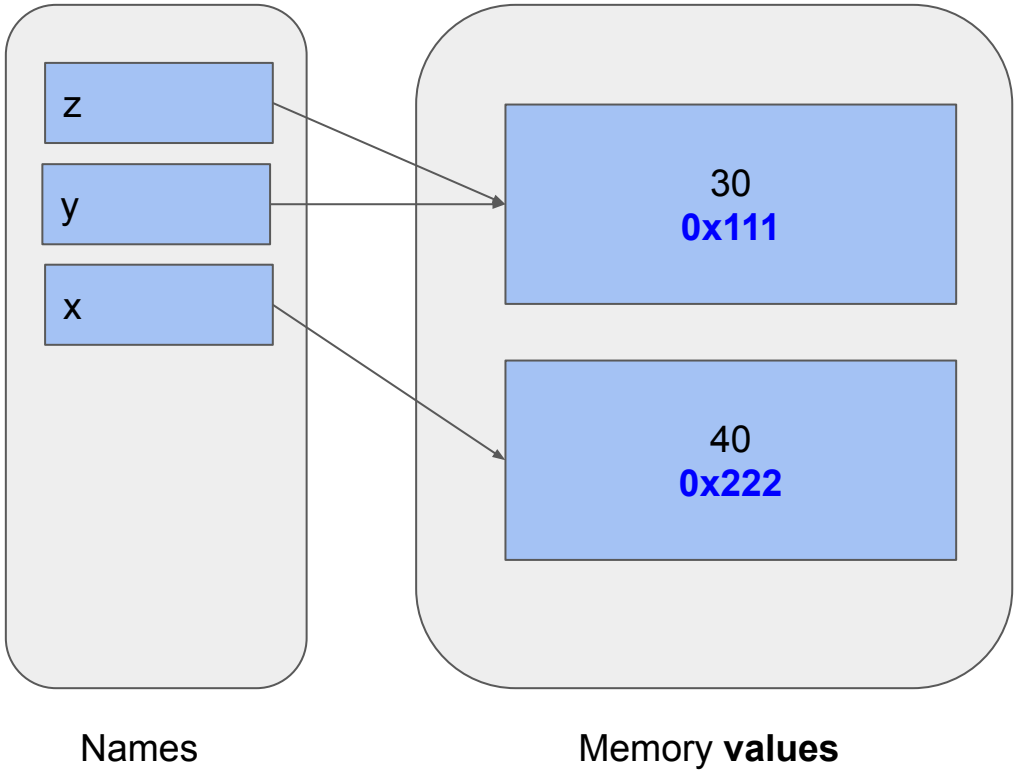


- Value 30 is created in memory
  - Int is immutable
  - NO one can change value at memory address 0x111
- Anything that needs value 30 **might** be bounded to 0x111
  - *No guarantee for line 8 & 9*
  - *CPython's peephole optimizer*

Memory for immutable objects

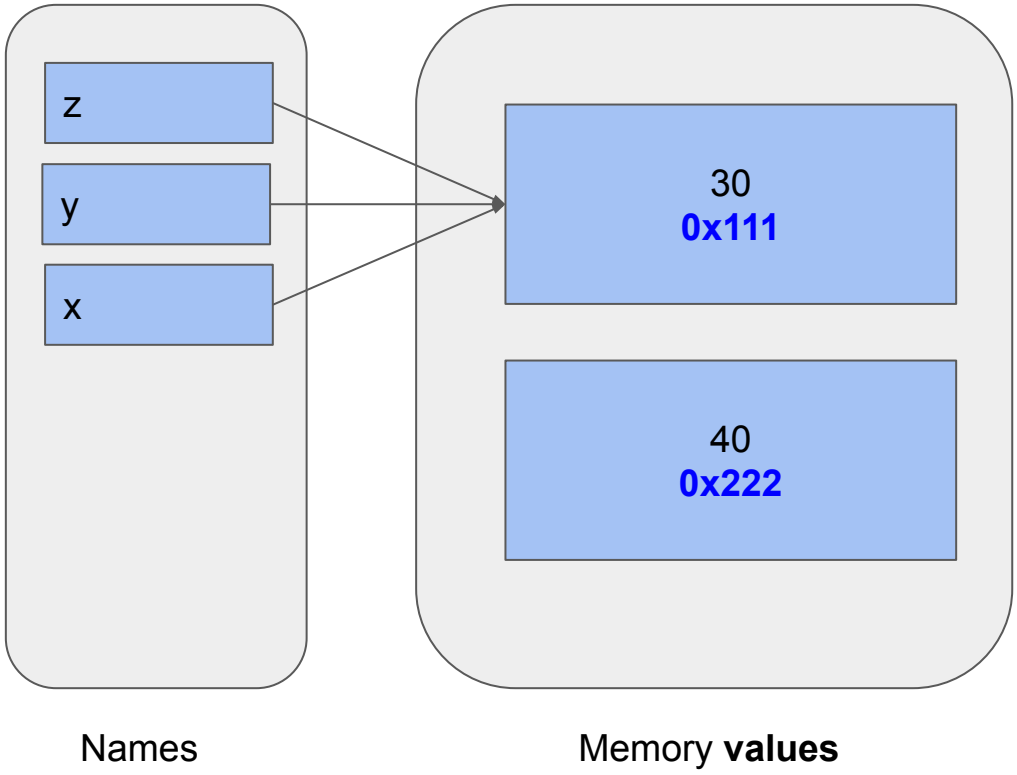
```
3   x = 30
4   y = x
5   z = 30
6   print(id(x))    # 0x111
7   print(id(y))    # 0x111
8   print(id(z))    # 0x111
9   print(id(30))   # 0x111
10
11  x += 10
12  print(id(x))    # 0x222 ***
13  print(id(y))    # 0x111
```

- 40 is a new values (don’t exist)
- Python creates the value in memory
- Now x points to it
- z and y: no change



Memory for immutable objects

```
3   x = 30
4   y = x
5   z = 30
6   print(id(x))    # 0x111
7   print(id(y))    # 0x111
8   print(id(z))    # 0x111
9   print(id(30))   # 0x111
10
11  x += 10
12  print(id(x))    # 0x222 ***
13  print(id(y))    # 0x111
14
15  x = 30
16  print(id(x))    # 0x111
```



- 40 now has zero references
  - It can be removed

Same concepts with other immutable objects

```
3   x = 'Hey'
4   y = x
5   z = 'Hey'
6
7   print(x is z)
8
9   print(id(x))          # 0x111
10  print(id(y))          # 0x111
11  print(id(z))          # 0x111
12  print(id('Hey'))      # 0x111
13
14  x += ' Most'
15  print(id(x))          # 0x222 ***
16  print(id(y))          # 0x111
17
18  x = 'Hey'
19  print(id(x))          # 0x111
20
21  #x[0] = 'R'           # TypeError
```

Tuple: mix of mutable and immutable

```
class Employee:
    def __init__(self, name):
        self.name = name

obj1 = Employee('Mostafa')  # mutable
obj2 = 'mostafa'            # immutable

my_tuple = (obj1, obj2)
#my_tuple[0] = Employee('belal')     # TypeError
y, z = my_tuple
print(y.name)    # Mostafa

# we can't replace tuple items
# but if an item is mutable, we can change its content

obj1.name = 'ziad'
y, z = my_tuple
print(y.name)    # ziad
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."