Python CS-521

Eugene Pinsky
Department of Computer Science
Metropolitan College, Boston University
Boston, MA 02215
email: epinsky@bu.edu

May 3, 2020

Abstract

This course will present an effective approach to help you learn Python. With extensive use of graphical illustrations, we will build understanding of Python and its capabilities by learning through many simple examples and analogies. The class will involve active student participation, discussions, and programming exercises. This approach will help you build a strong foundation in Python that you will be able to effectively apply in real-job situations and future courses.

MUTABILITY

Mutability

- everything in Python is an object
- mutable or immutable
- mutable can be changed
- immutable cannot

Mutability Examples

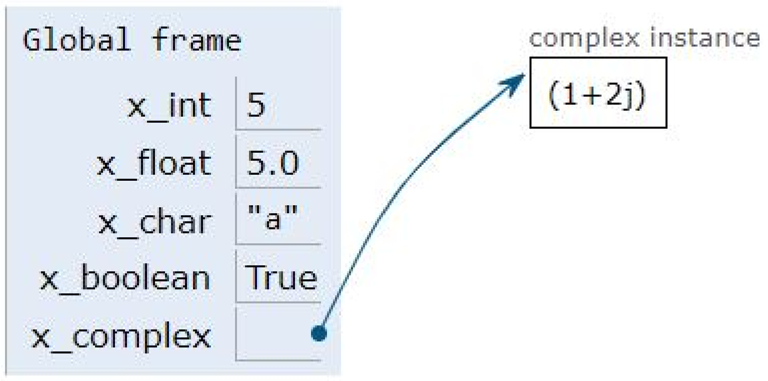
- mutable:
  lists, sets, dict
- immutable:
  primitive types, strings
  and tuples
- custom classes are
  typically mutable

id() **Function**

- every object has ”id”
- like an ”address”
- type at runtime
- type does not change
- state can change for mutable objects

Primitive Types

```
x_int     = 5
x_float   = 5.0
x_char    = 'a'
x_boolean = True
x_complex = 1 + 2j
```



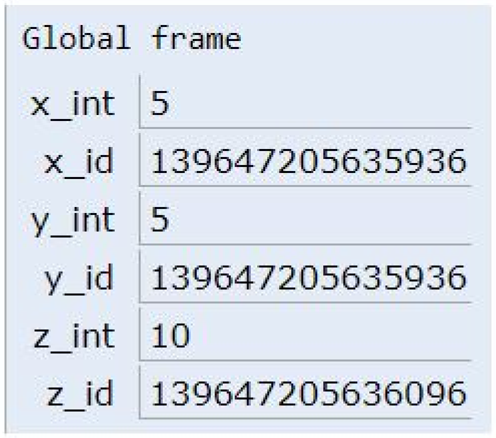
- 'atoms' - indivisible objects

Primitive Types

```
x_int = 5
x_id = id(x_int)

y_int = 5                # same type and value
y_id = id(y_int)         # same id

z_int = 10               # same type, different value
z_id = id(z_int)         # different id
```

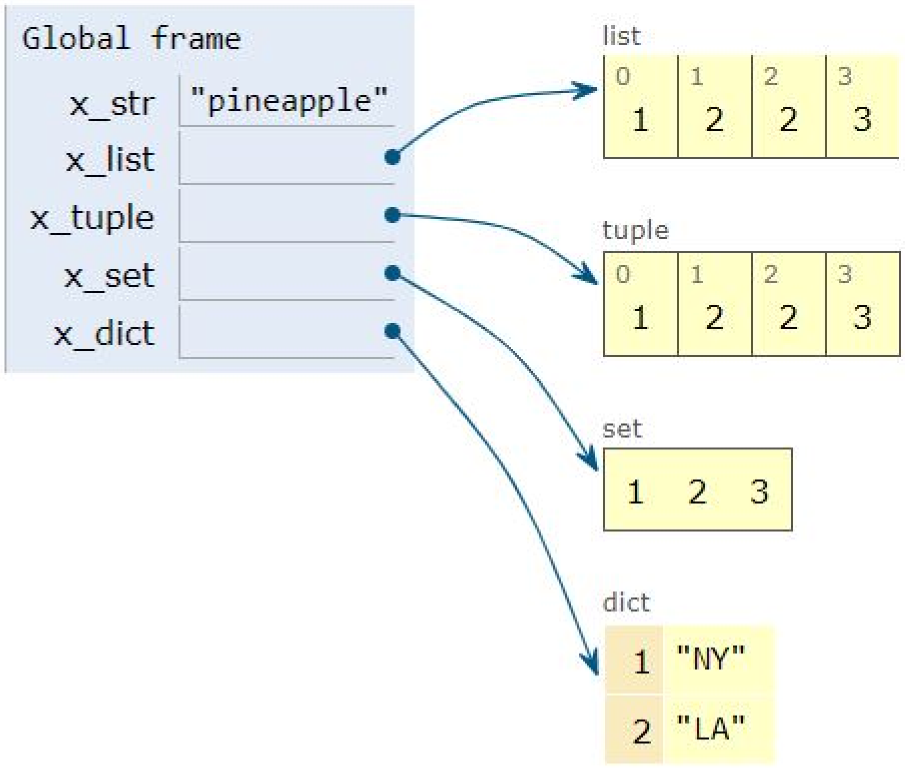


```
Global frame
x_int  5
x_id   139647205635936
y_int  5
y_id   139647205635936
z_int  10
z_id   139647205636096
```

- all are immutable

Collection Types

```
x_string = 'pineapple'
x_list   = [1,  2, 2, 3]
x_tuple  = (1, 2, 2, 3)
x_set    = {1, 2, 2, 3} # note duplicates
x_dict   = {1: 'NY', 2: 'LA'}
```



- 'molecules' - complex objects

Mutability

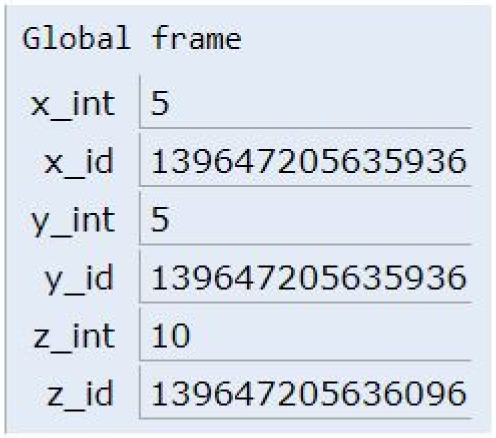
- mutable collections:
  1. list
  2. set
  3. dictionary
- immutable collections:
  1. strings
  2. tuples

Immutability in Strings

```
x_str = 'Apple'
x_id = id(x_str)

y_str = 'Apple'
y_id = id(y_str)

z_str = 'apple'
z_id = id(z_str)
```



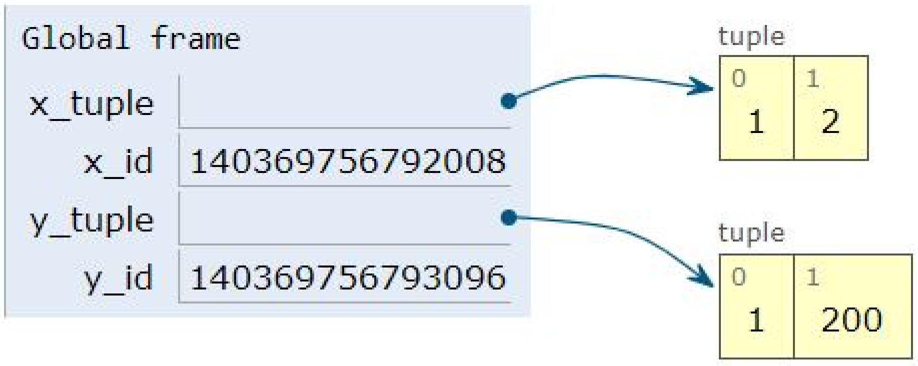
Global frame

x_int	5
x_id	139647205635936
y_int	5
y_id	139647205635936
z_int	10
z_id	139647205636096

Immutability in Tuples

```
# cannot replace individual elements
x_tuple = (1, 2)
x_id = id(x_tuple)

# x_tuple[1] = 200    # ILLEGAL !!!!
y_tuple = (1, 200)
y_id = id(y_tuple)
```

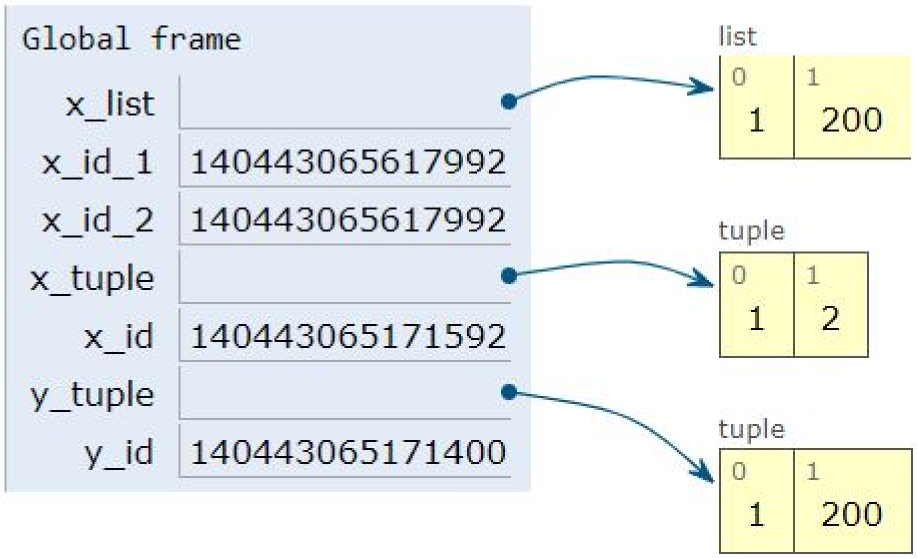


- need a new object

List vs. Tuple

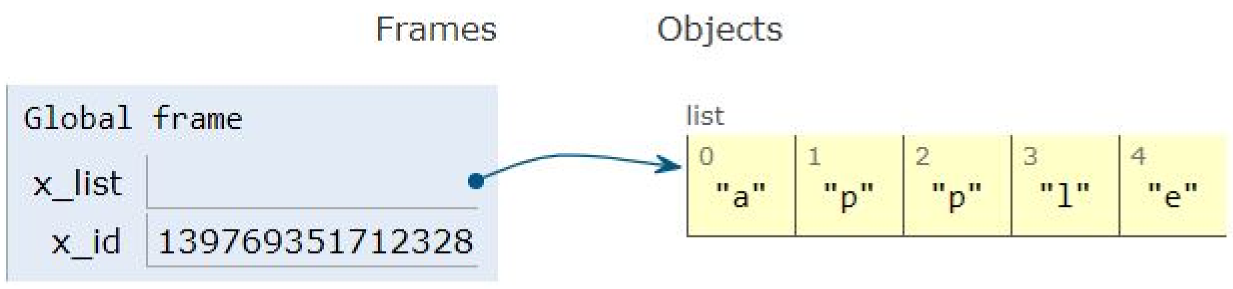
```
x_list = [1, 2]
x_id_1 = id(x_list)
x_list[1] = 200        # modify in-place
x_id_2 = id(x_list)

x_tuple = (1, 2)
x_id = id(x_tuple)
# x_tuple[1] = 200    # illegal, need new tuple
y_tuple = (1, 200)
y_id = id(y_tuple)
```

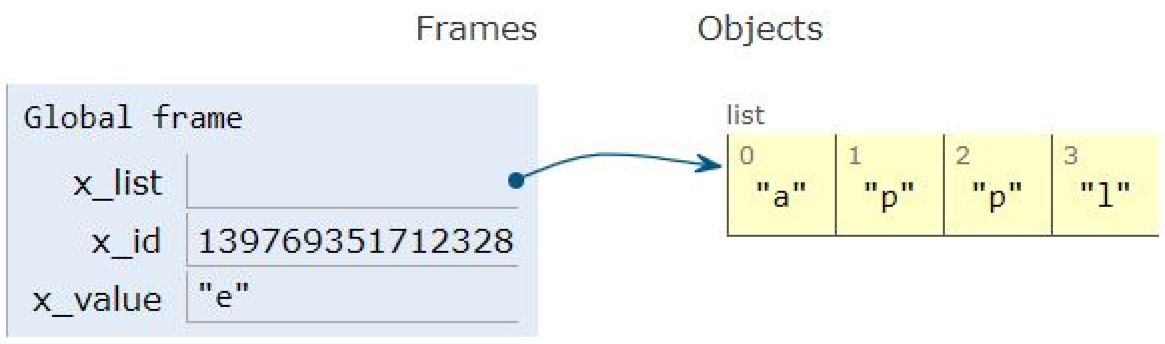


List Mutability

```
x_list    = ['a', 'p', 'p', 'l', 'e']
x_id      = id(x_list)
```

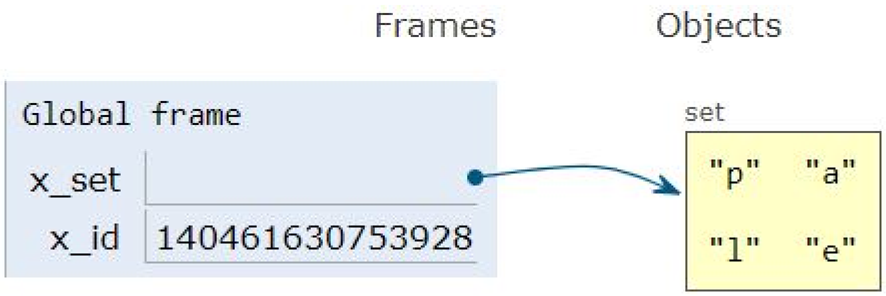


```
x_value   = x_list.pop(-1)  # remove last
x_id      = id(x_list)
```

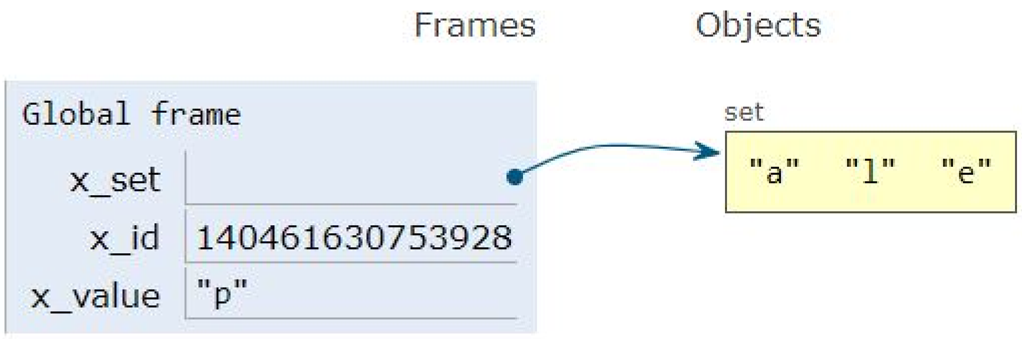


Set Mutability

```
x_set     = {'a', 'p', 'p', 'l', 'e'}
x_id      = id(x_set)
```

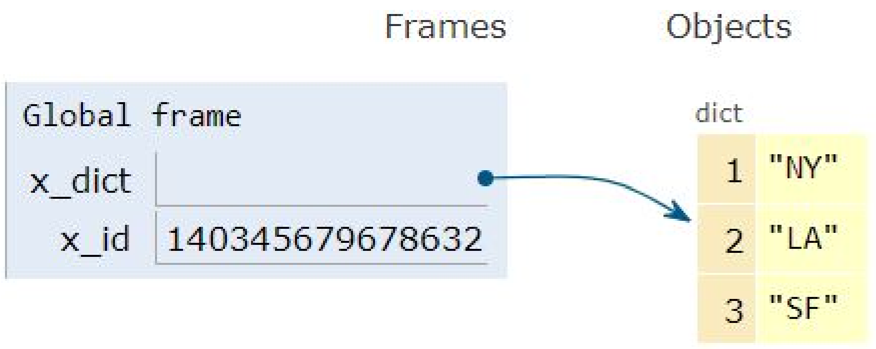


```
x_value   = x_set.pop()   # remove random
x_id      = id(x_set)
```



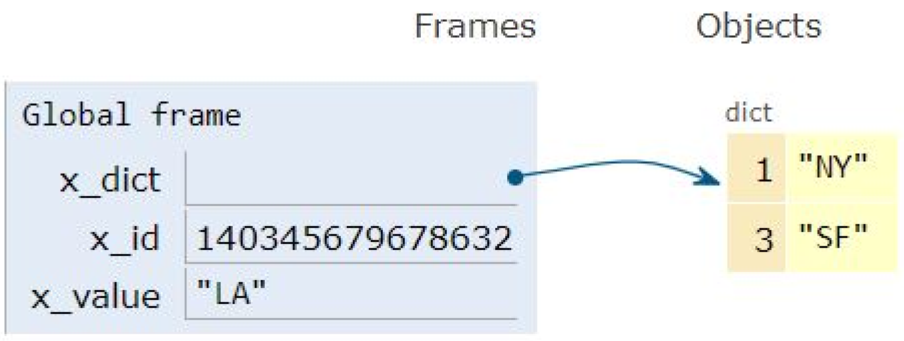
Dictionary Mutability

```
x_dict   = {1 : 'NY', 2: 'LA', 3: 'SF'}
x_id     = id(x_dict)
```



x_id: 140345679678632

```
x_value = x_dict.pop(2)   # remove key = 2
x_id     = id(x_dict)
```



x_id: 140345679678632
x_value: "LA"

Summary of Collections

Collection	Ordered	Mutable
string	**yes**	**no**
list	**yes**	**yes**
tuple	**yes**	**no**
set	**no**	**yes**
dictionary	**no**	**yes**

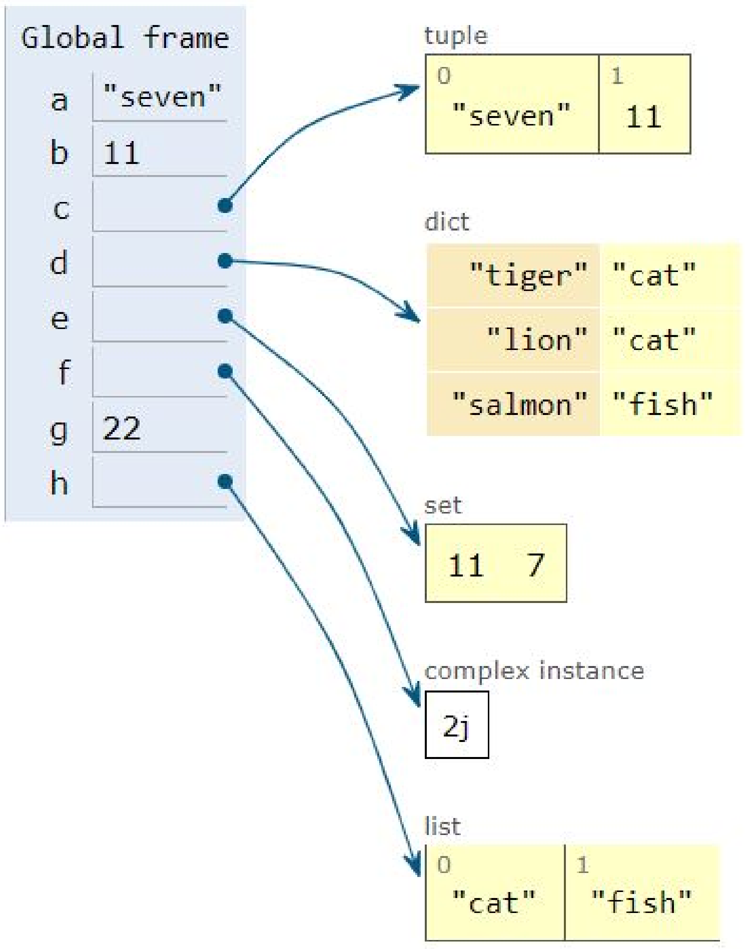
- some variations:
  1. ’frozen’ set (immutable)
  2. ordered dictionary

Exercise(s):

- which built-in types are mutable?
- is it possible to modify an immutable object?
- which built-in types are immutable?

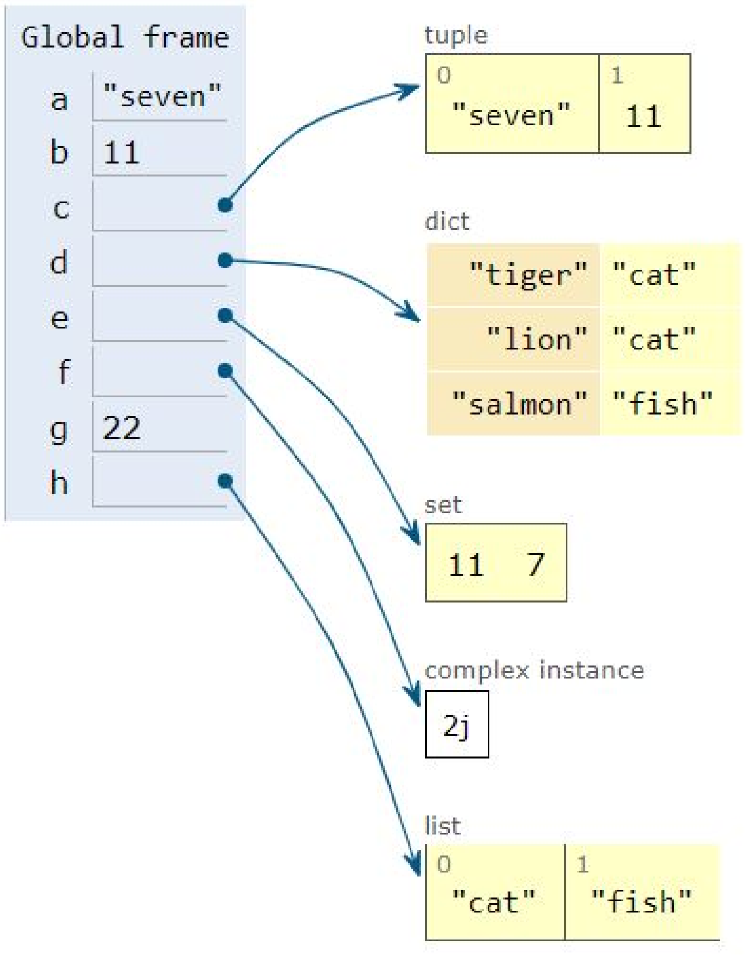
Exercise(s):

- which objects are mutable?



Exercise(s):

- which objects are non-primitive, hashable and immutable?


Global frame
a "seven"
b 11
c
d
e
f
g 22
h
tuple
0 "seven"
1 11
dict
"tiger" "cat"
"lion" "cat"
"salmon" "fish"
set
11 7
complex instance
2j
list
0 "cat"
1 "fish"