PYTHON

OVERVIEW

A Simple Example

```
x = 5
y = 10.5
z = 1 + 2j
w = "Boston University"
u = [x, y, z, w]
```

- no explicit declarations
- dynamic typing
- heterogeneous items

Easy to Learn

- C++

```
#include <iostream.h>
void main()
{
 cout<<"Welcome to Boston University"
     <<endl;
}
```

- Python

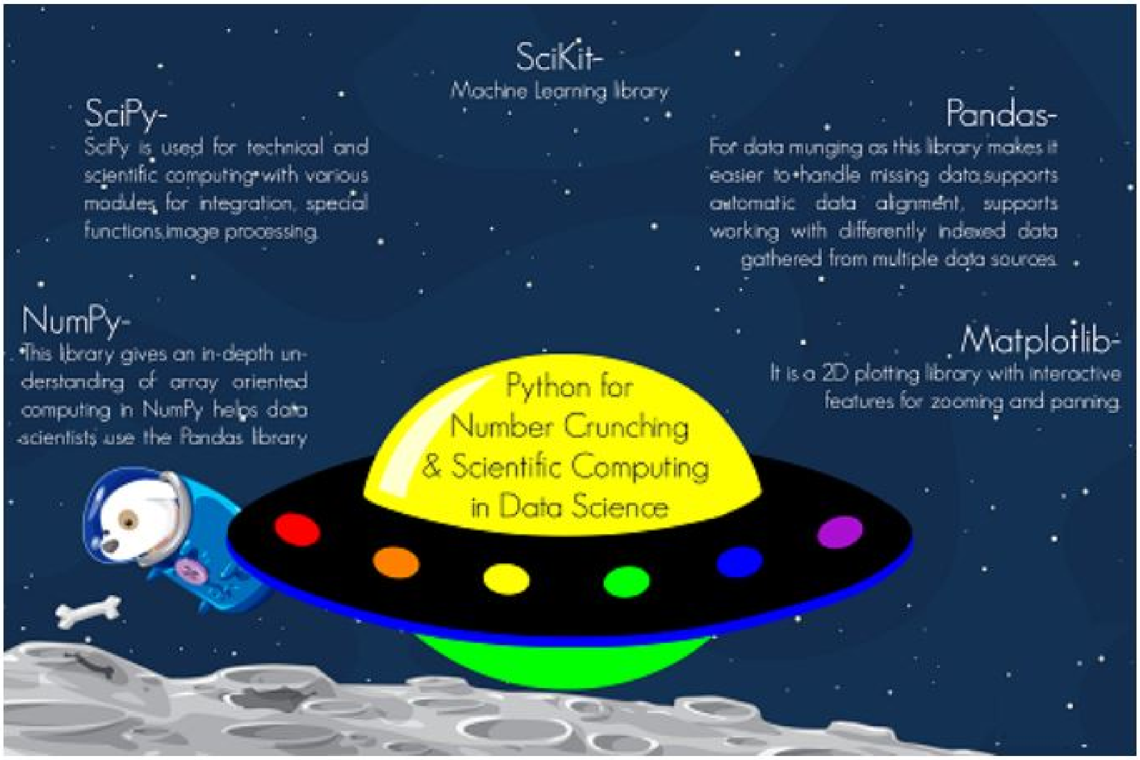
```
print("Welcome to Boston University")
```

- Python is intepreted
- no pre-defined type (”dynamic” typing)

Features of Python

- free
- object oriented
- powerful and flexible containers
- many built-in methods
- many libraries, toolkits and environments
- but: need to include many modules
- interpreted (not compiled)

Libraries



SciKit-
Machine Learning library

SciPy-
SciPy is used for technical and scientific computing with various modules for integration, special functions,image processing.

Pandas-
For data munging as this library makes it easier to handle missing data,supports automatic data alignment, supports working with differently indexed data gathered from multiple data sources.

NumPy-
This library gives an in-depth understanding of array oriented computing in NumPy helps data scientists use the Pandas library

Matplotlib-
It is a 2D plotting library with interactive features for zooming and panning.

- extend Python significantly

figure reprinted from www.kdnuggets.com with explicit permission of the editor

Libraries

- Numpy (numerical Python)
- Pandas (panel data)
- Scikit (machine learning)
- Scipy (scientific computing)
- Matplotlib (graphics)

Python IDE

- Spyder
- IDLE
- PyStudio
- PyCharm

Data Types

```
x_int     = 5
x_float   = 5.0
x_char    = "A"
x_bool    = True
x_complex = 1+2j
```

- primitive (”atoms”):

1. integer
2. float
3. char
4. boolean
5. complex

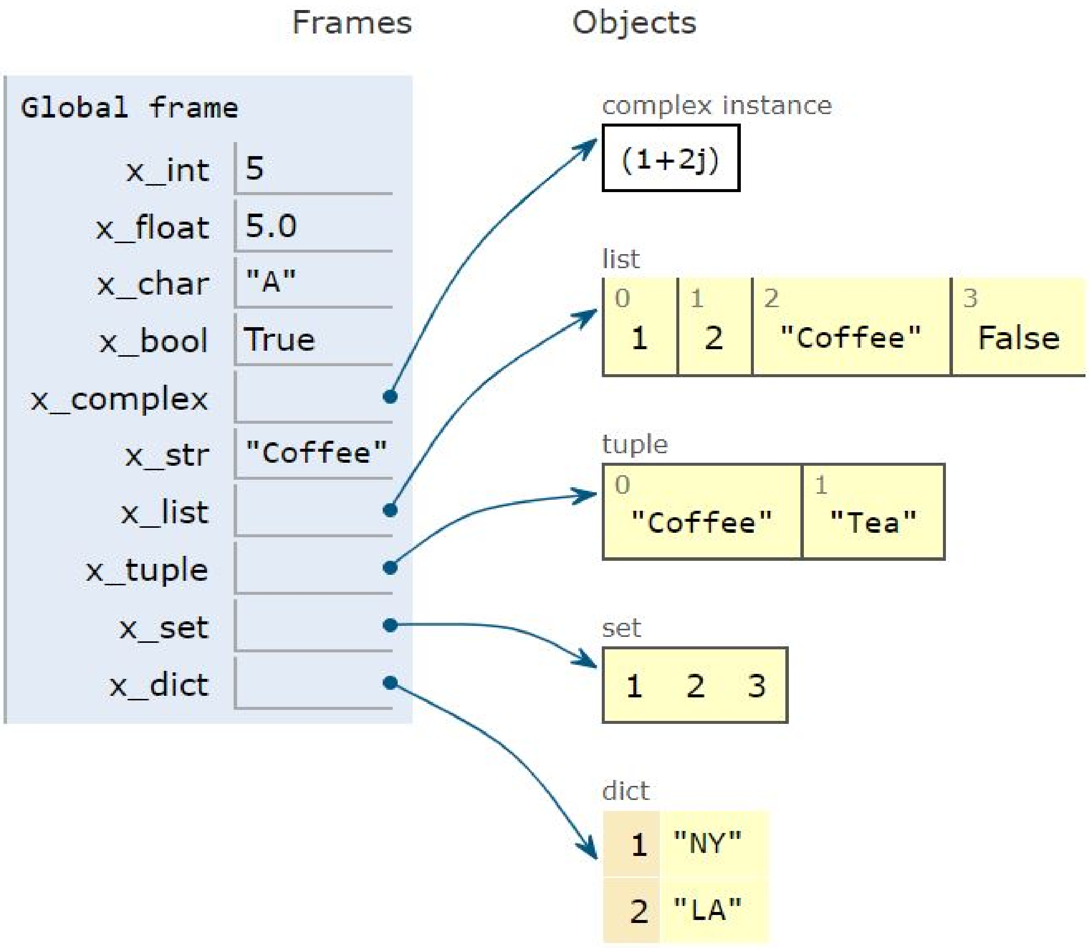
Data Types

```
x_str   = "Coffee"
x_list  = [1, 2, "Coffee", False]
x_tuple = ("Coffee", "Tea")
x_set   = { 1, 2, 3 }
x_dict  = {1: "NY", 2: "LA"}
```

- collections (”molecules”):

1. strings
2. lists
3. tuples
4. sets
5. dictionaries

Data Types Illustration



Global frame

x_int	5
x_float	5.0
x_char	"A"
x_bool	True
x_complex	●
x_str	"Coffee"
x_list	●
x_tuple	●
x_set	●
x_dict	●

complex instance

(1+2j)

list

0	1	2	3
1	2	"Coffee"	False

tuple

0	1
"Coffee"	"Tea"

set

1 2 3

dict

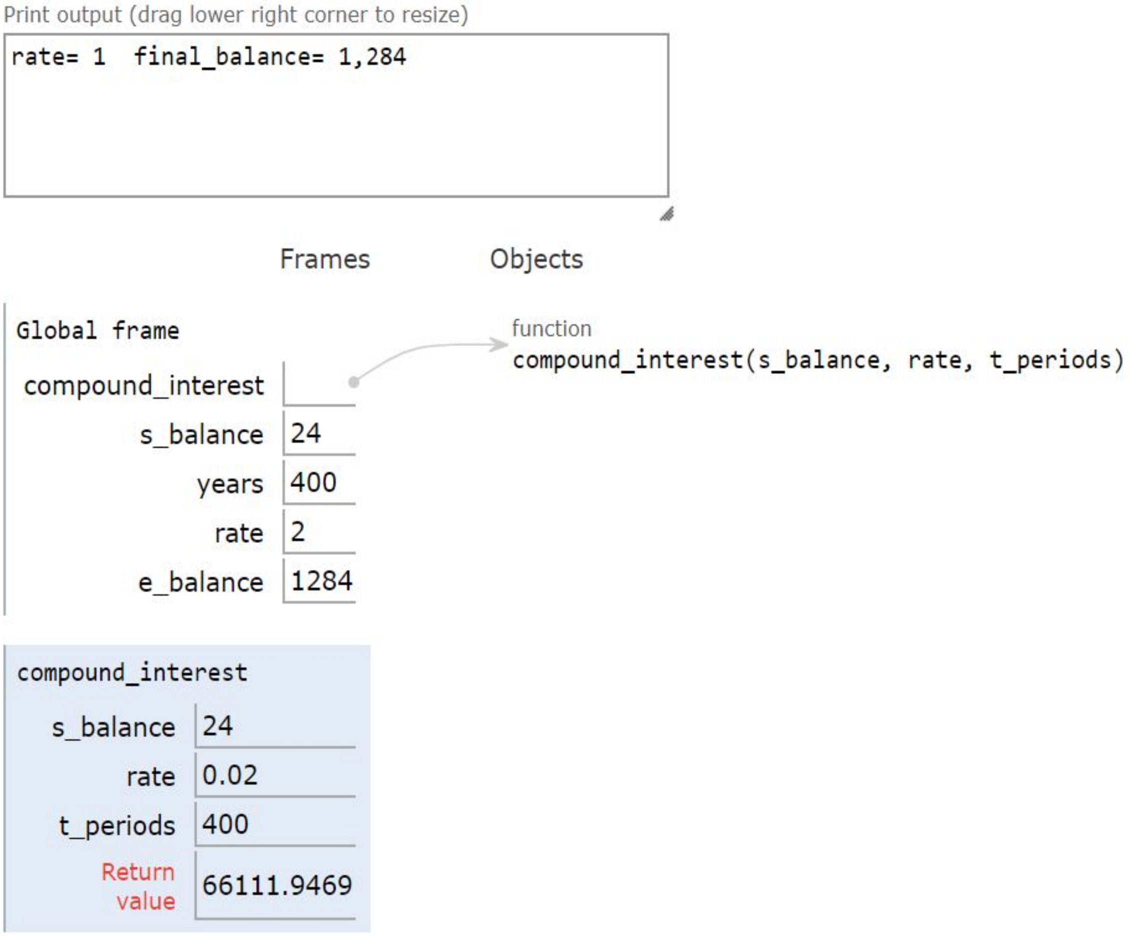
1	"NY"
2	"LA"

Example: Interest

```
def compound_interest(s_balance,rate,t_periods):
    return s_balance * (1 + rate)**t_periods

s_balance = 24
years = 400 # comment: sold in 1626
for rate in range(1,10):
    e_balance=int(compound_interest(s_balance,
                            rate/100, years))
    print("rate="’, rate,
             "final_balance=", f"{e_balance:,d}")
```

```
rate= 1  final_balance= 1,284
rate= 2  final_balance= 66,111
rate= 3  final_balance= 3,274,169
rate= 4  final_balance= 156,151,787
rate= 5  final_balance= 7,176,800,429
rate= 6  final_balance= 318,095,369,845
rate= 7  final_balance= 13,605,744,645,294
rate= 8  final_balance= 561,970,394,044,952
rate= 9  final_balance= 22,429,026,185,144,604
```

”Interest” Example Illustration



```
rate= 1  final_balance= 1,284
```

Global frame

compound_interest

s_balance 24

years 400

rate 2

e_balance 1284

compound_interest(s_balance, rate, t_periods)

compound_interest

s_balance 24

rate 0.02

t_periods 400

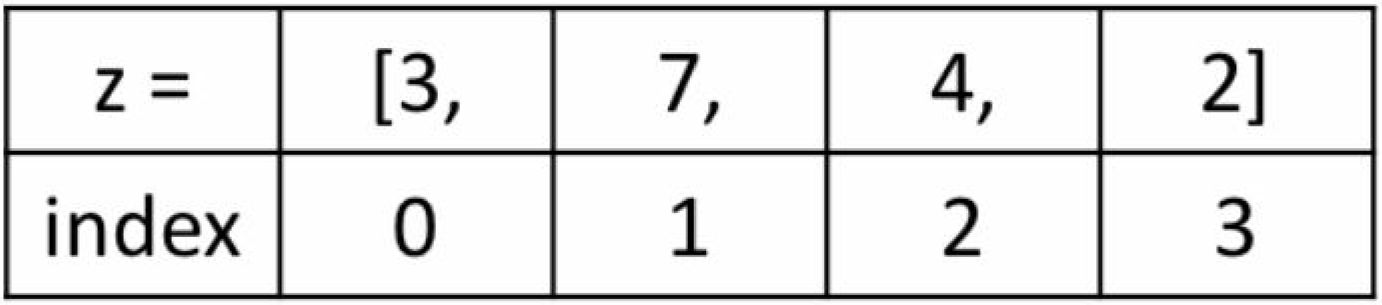
Return value 66111.9469

”Interest” Example Comments

- dynamic typing
- indentation
- iterations (for loop)
- functions (using def)
- arithmetic operations (+, **)
- (flexible) print
- concise code

Python Lists

```
# Define a list
z = [3, 7, 4, 2]
```



z =	[3,	7,	4,	2]
index	0	1	2	3

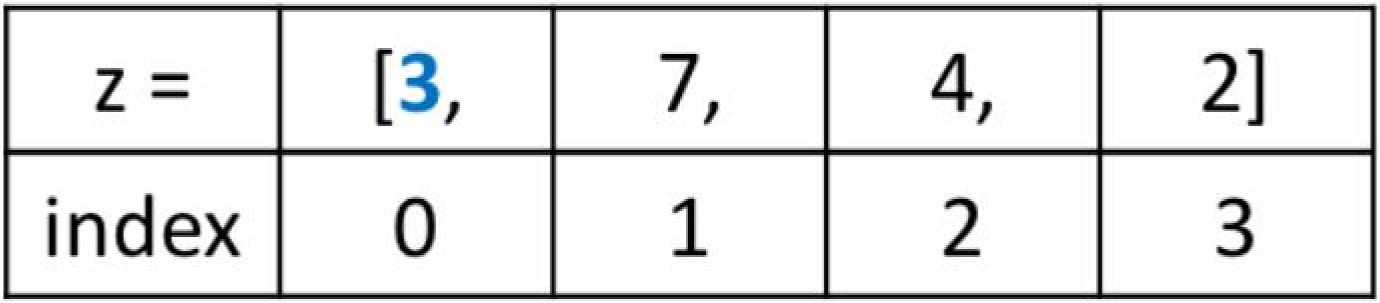
- can contain different data types

```
z = [3, True, "Boston", (1,2)]
```

figure reprinted from www.kdnuggets.com with explicit permission of the editor

Accesing Lists

```
z = [3, 7, 4, 2]
print(z[0])
```

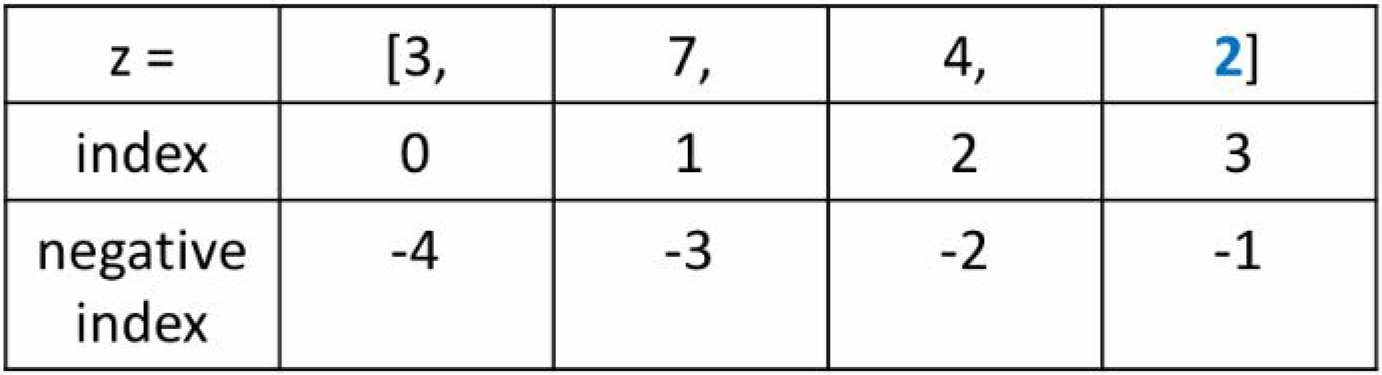


z =	[3,	7,	4,	2]
index	0	1	2	3

figure reprinted from www.kdnuggets.com with explicit permission of the editor

Negative Indexing

```
z = [3, 7, 4, 2]
print(z[-1])
```

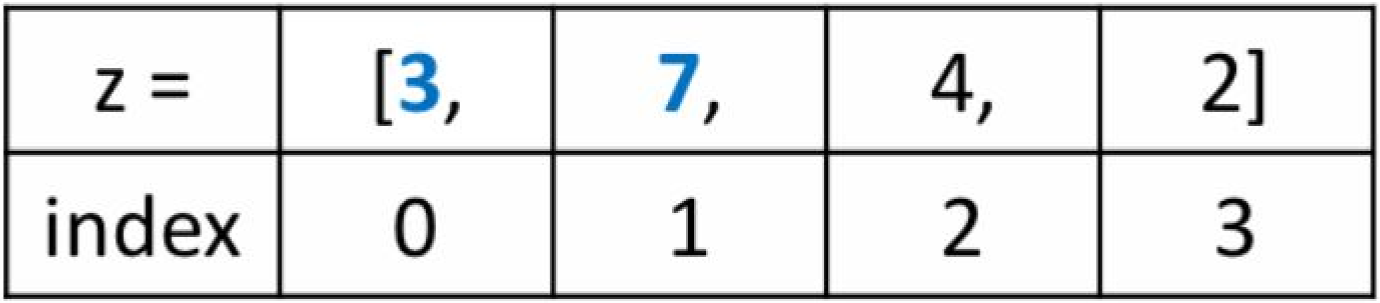


z =	[3,	7,	4,	2]
index	0	1	2	3
negative index	-4	-3	-2	-1

figure reprinted from www.kdnuggets.com with explicit permission of the editor

List Slicing

```
z = [3, 7, 4, 2]
print(z[0 : 2])
```



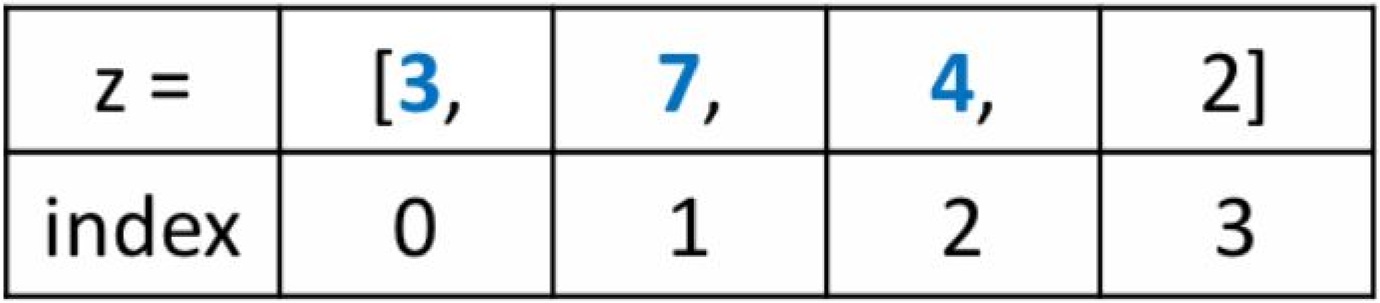
z =	[3,	7,	4,	2]
index	0	1	2	3

- everything up to but not including index 2

figure reprinted from www.kdnuggets.com with explicit permission of the editor

List Slicing (cont'd)

```
z = [3, 7, 4, 2]
print(z[ : 3])
```



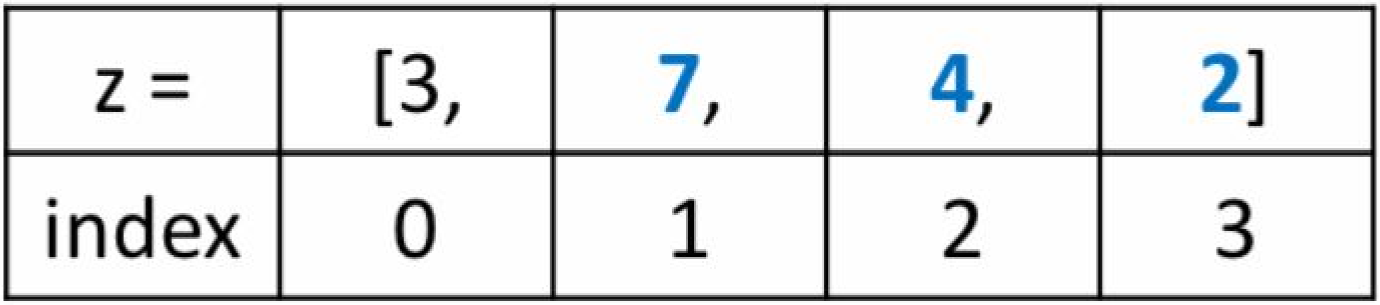
z =	[3,	7,	4,	2]
index	0	1	2	3

- everything up to but not including index 3

figure reprinted from www.kdnuggets.com with explicit permission of the editor

List Slicing (cont'd)

```
z = [3, 7, 4, 2]
print(z[ 1 : ])
```



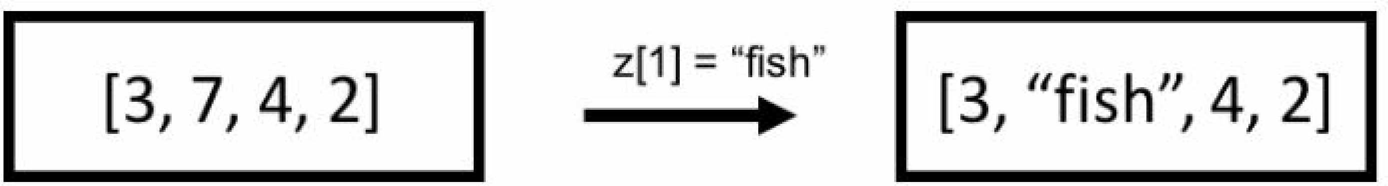
z =	[3,	7,	4,	2]
index	0	1	2	3

- from index 1 till the end

figure reprinted from www.kdnuggets.com with explicit permission of the editor

List Updating

```
z     = [3, 7, 4, 2]
z[1]  = "fish"
print(z)
```

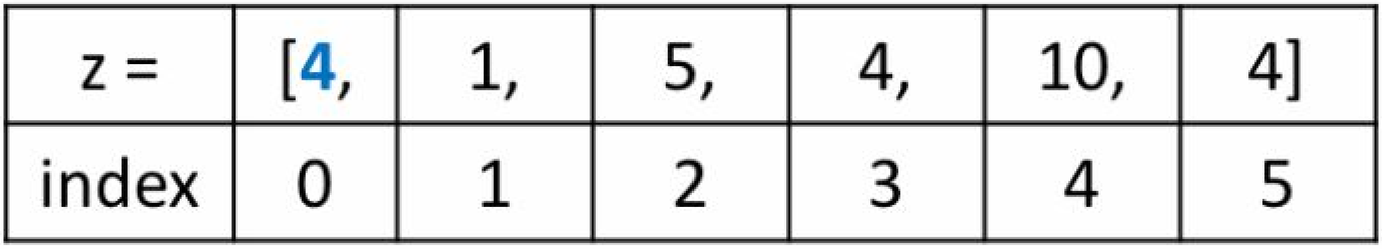


- from index 1 till the end

figure reprinted from www.kdnuggets.com with explicit permission of the editor

List *index*() Method

```
z     = [4, 1, 5, 4, 10, 4]
print(z.index(4))
```



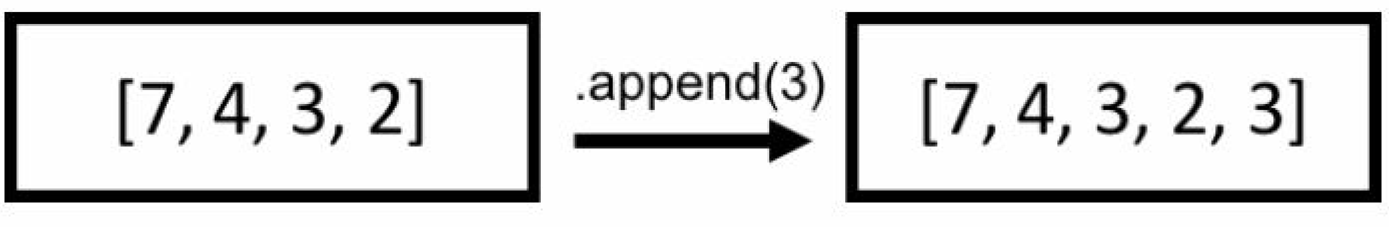
z =	[4,	1,	5,	4,	10,	4]
index	0	1	2	3	4	5

- first index with value 4

figure reprinted from www.kdnuggets.com with explicit permission of the editor

List *append*() Method

```
z = [7, 4, 3, 2]
z.append(3)
print(z.count(5))
```

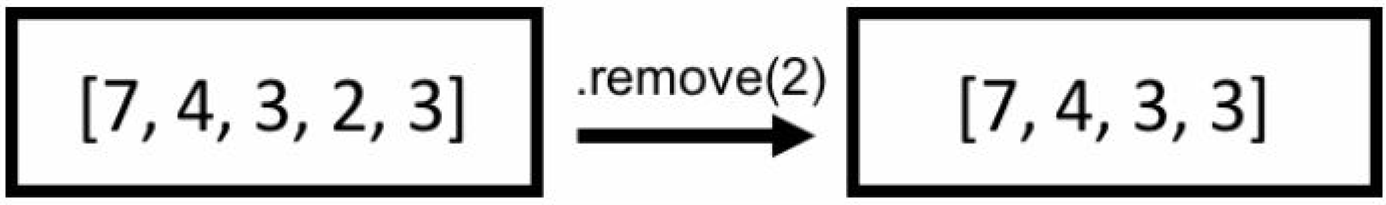


- add at the end of the list
- done ”in-place”

figure reprinted from www.kdnuggets.com with explicit permission of the editor

List *remove*() Method

```
z = [7, 4, 3, 2, 3]
z.remove(2)
print(z.count(5))
```

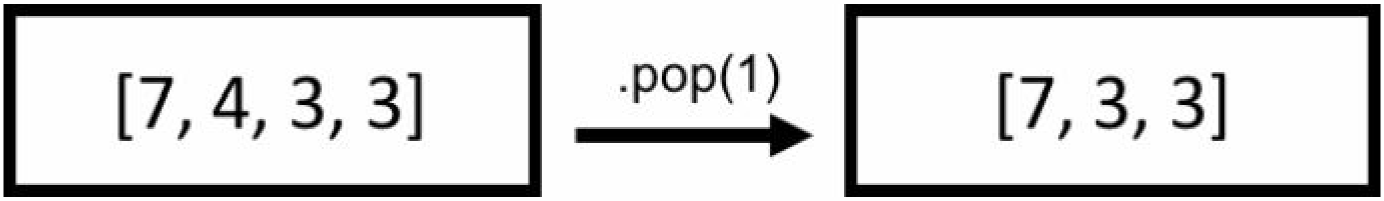


- removes first occurrence of value
- done ”in-place”

figure reprinted from www.kdnuggets.com with explicit permission of the editor

List *pop*() Method

```
z = [7, 4, 3, 3]
z.pop(1)
print(z.count(5))
```

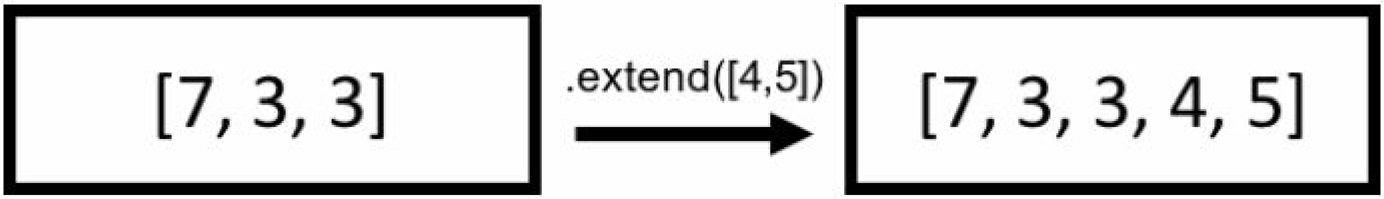


- removes at specified index
- default is -1 (end of list)

figure reprinted from www.kdnuggets.com with explicit permission of the editor

List *extend*() Method

```
z = [7, 3, 3]
z.extend([4, 5])
```

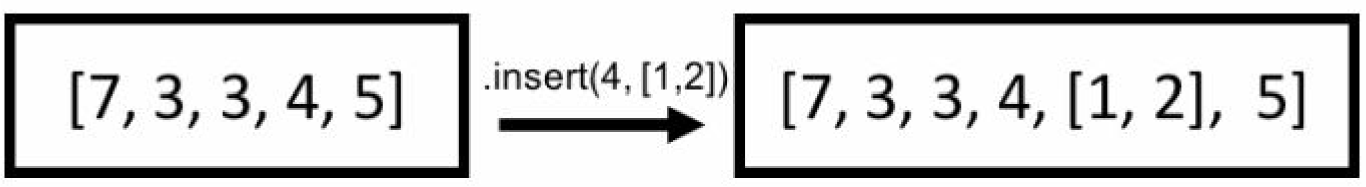


- add another list at the end
- done in place

figure reprinted from www.kdnuggets.com with explicit permission of the editor

List *insert*() Method

```
z = [7, 3, 3, 4, 5]
z.insert(4, [1, 2])
```



[7, 3, 3, 4, 5]

.insert(4, [1,2])

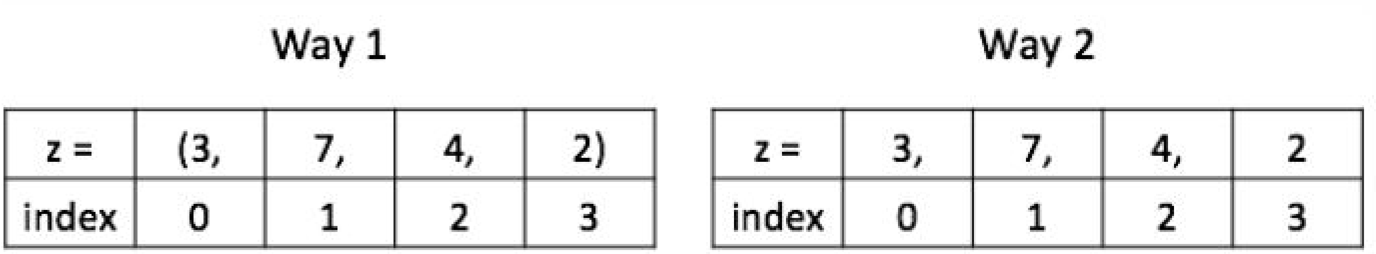
[7, 3, 3, 4, [1, 2], 5]

- insert an item *before* index
- done ”in place”

figure reprinted from www.kdnuggets.com with explicit permission of the editor

Python Tuples

```
# Define a tuple
z = (3, 7, 4, 2)        # way 1
z =  3, 7, 4, 2         # way 2
```



Way 1

z =	(3,	7,	4,	2)
index	0	1	2	3

Way 2

z =	3,	7,	4,	2
index	0	1	2	3

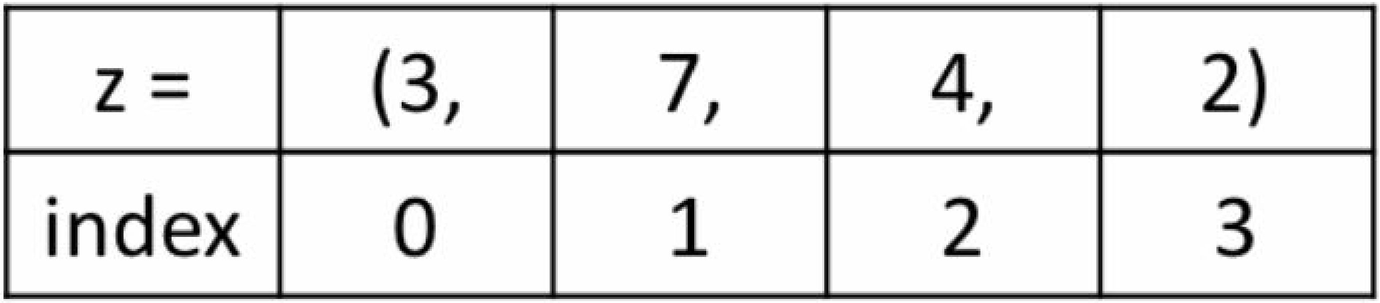
- both ways are equivalent
- can contain different data types

```
z = (3, True, "Boston", [1,2], (5, ))
```

figure reprinted from www.kdnuggets.com with explicit permission of the editor

Accesing Tuples

```
z = (3, 7, 4, 2)
print(z[0])
```

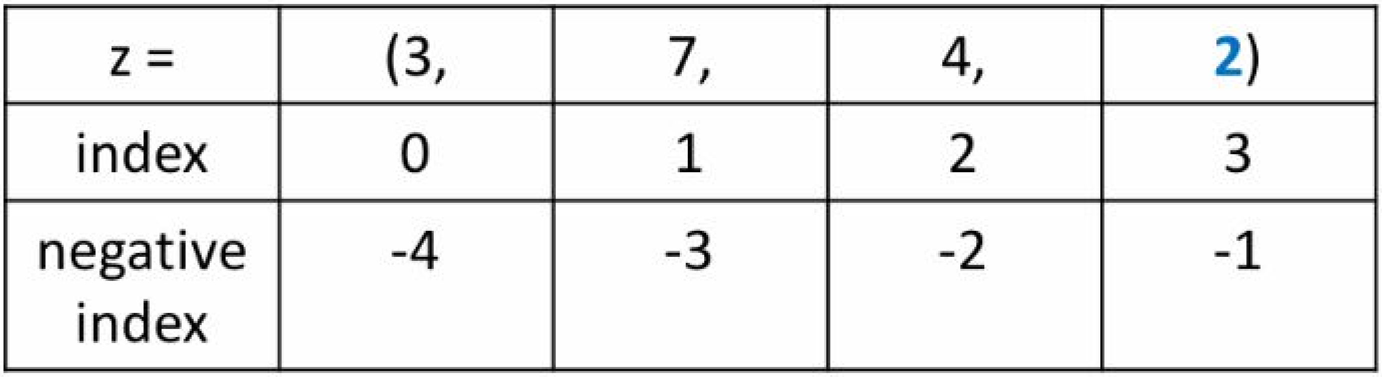


z =	(3,	7,	4,	2)
index	0	1	2	3

figure reprinted from www.kdnuggets.com with explicit permission of the editor

Accesing Tuples with Negative Indexing

```
z = (3, 7, 4, 2)
print(z[-1])
```

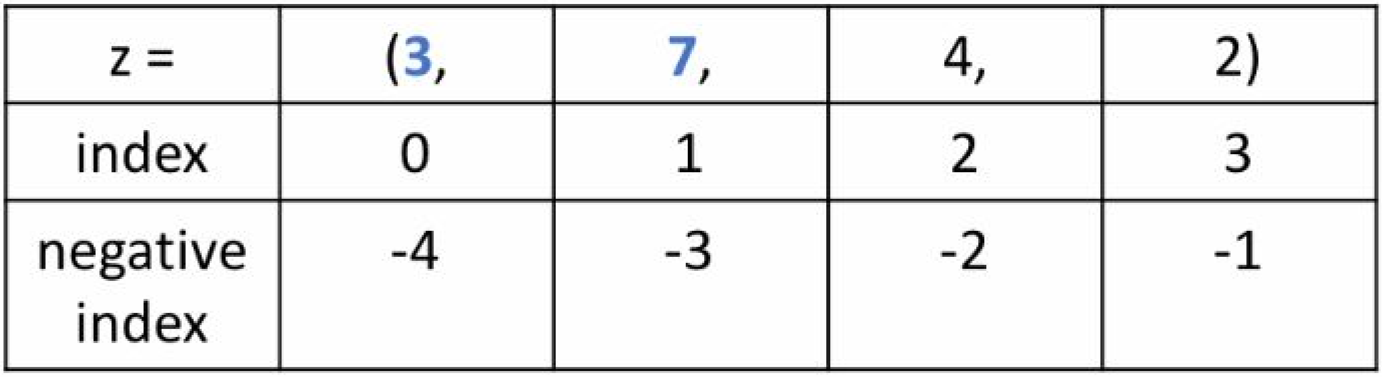


z =	(3,	7,	4,	2)
index	0	1	2	3
negative index	-4	-3	-2	-1

figure reprinted from www.kdnuggets.com with explicit permission of the editor

Tuple Slicing

```
z = (3, 7, 4, 2)
print(z[0 : 2])
```



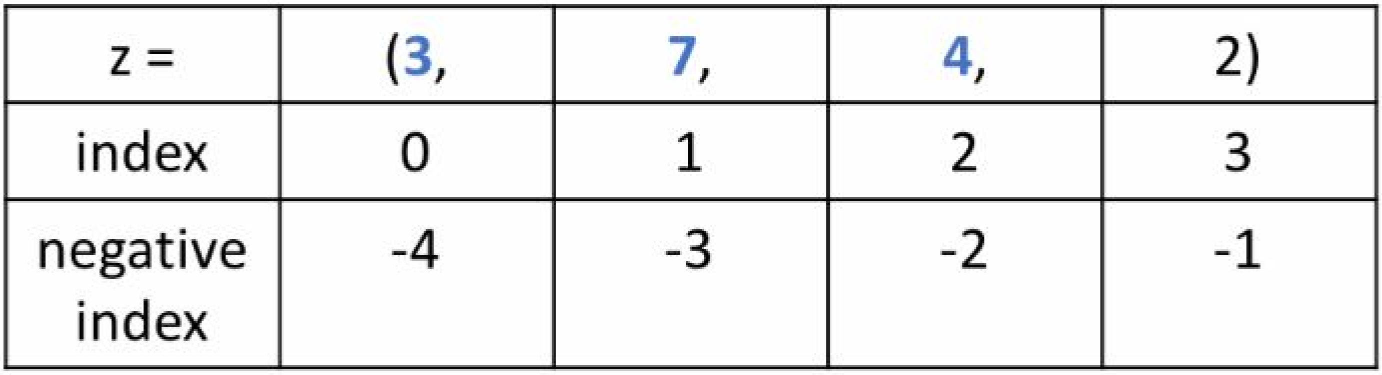
z =	(3,	7,	4,	2)
index	0	1	2	3
negative index	-4	-3	-2	-1

- everything up to but not including index 2

figure reprinted from www.kdnuggets.com with explicit permission of the editor

Tuple Slicing (cont'd)

```
z = (3, 7, 4, 2)
print(z[ : 3])
```



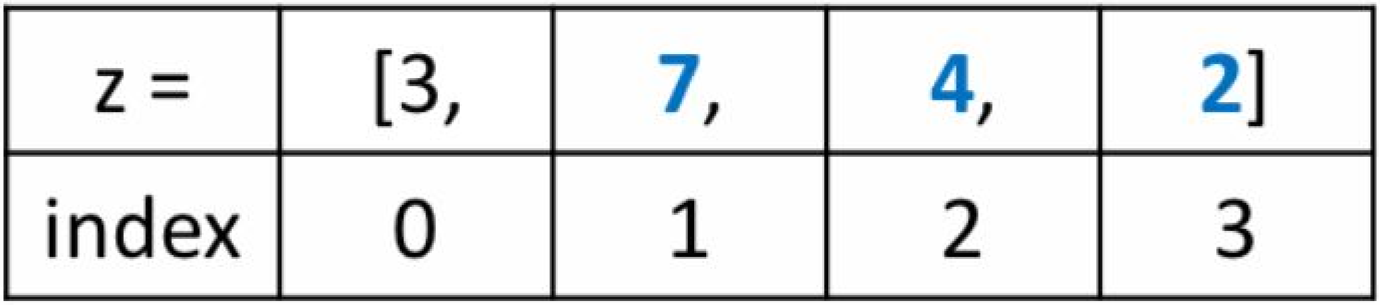
z =	(3,	7,	4,	2)
index	0	1	2	3
negative index	-4	-3	-2	-1

- everything up to but not including index 3

figure reprinted from www.kdnuggets.com with explicit permission of the editor

List Slicing (cont'd)

```
z = [3, 7, 4, 2]
print(z[ -4 : -1])
```



z =	[3,	7,	4,	2]
index	0	1	2	3

- can use negative values for slicing

figure reprinted from www.kdnuggets.com with explicit permission of the editor

Object Mutability

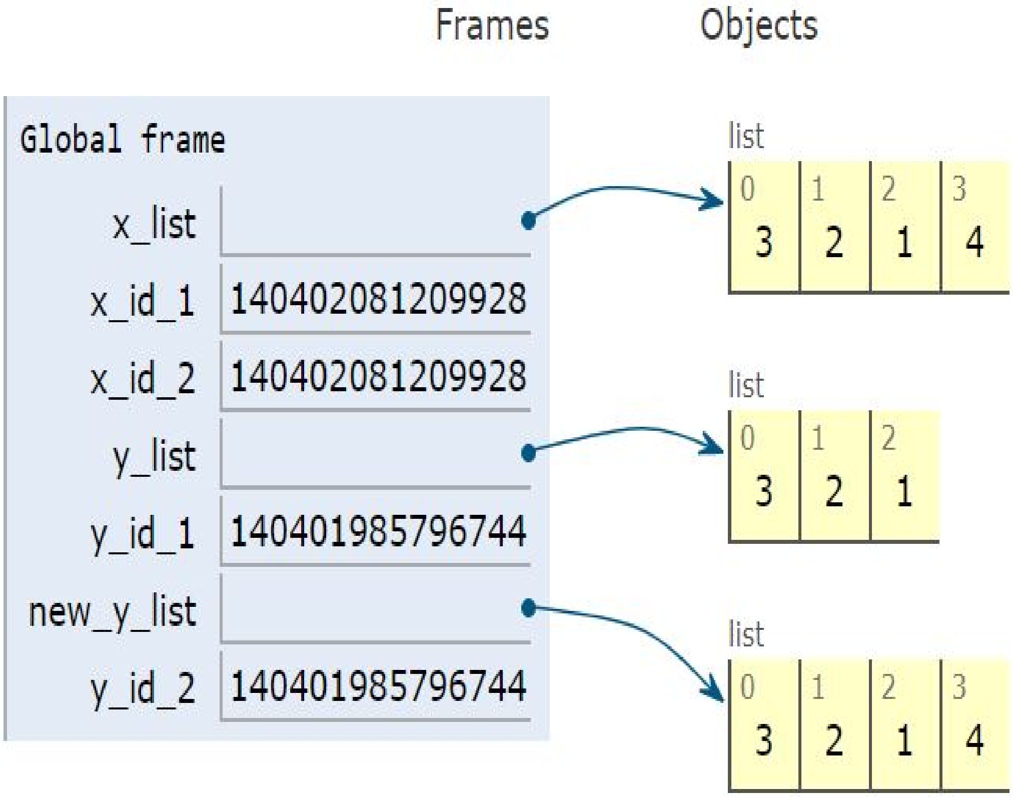
- each object is assigned an address - *id*()
- mutable object - object can be changed ”in-place”
- retains the same ”address”
- immutable: all primitive types, strings and tuples
- mutable: lists, sets and dictionaries

Example: Mutability

- appending to a list

```
# object stays in place
x_list = [3,2,1]
x_id_1 = id(x_list)
x_list.append(4)
x_id_2 = id(x_list)


# new object created at new address
y_list = [3,2,1]
y_id_1 = id(y_list)
new_y_list = y_list + [4]
y_id_2 = id(y_list)
```

Illustration of Mutability (append)



Global frame

x_list

x_id_1 140402081209928

x_id_2 140402081209928

y_list

y_id_1 140401985796744

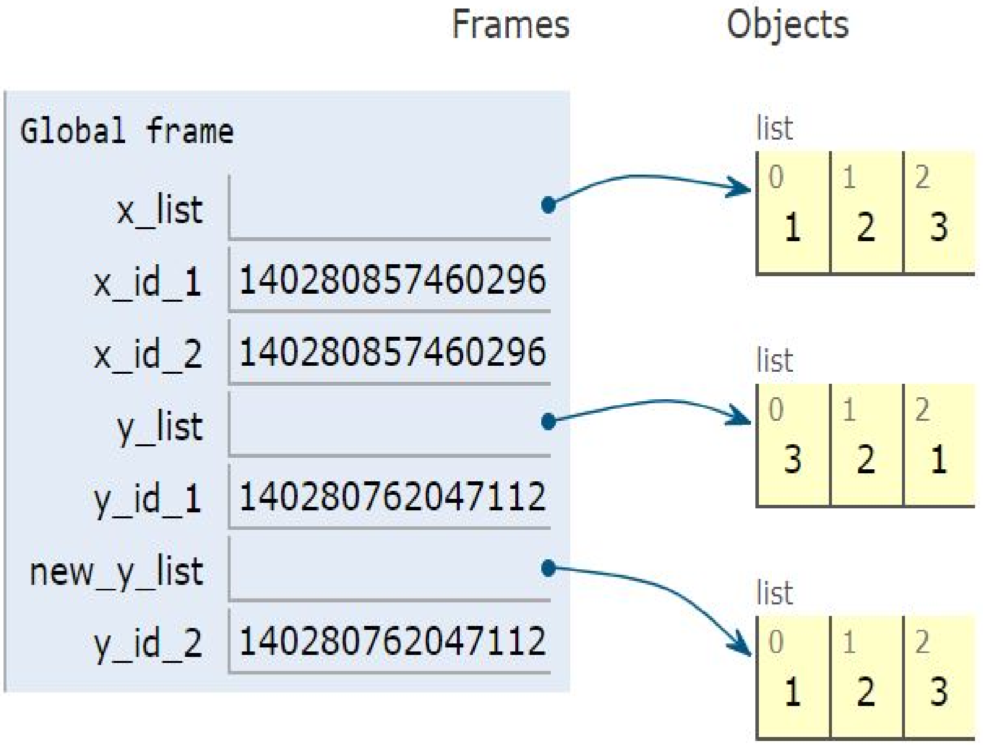
new_y_list

y_id_2 140401985796744

Sort() vs. *Sorted*()

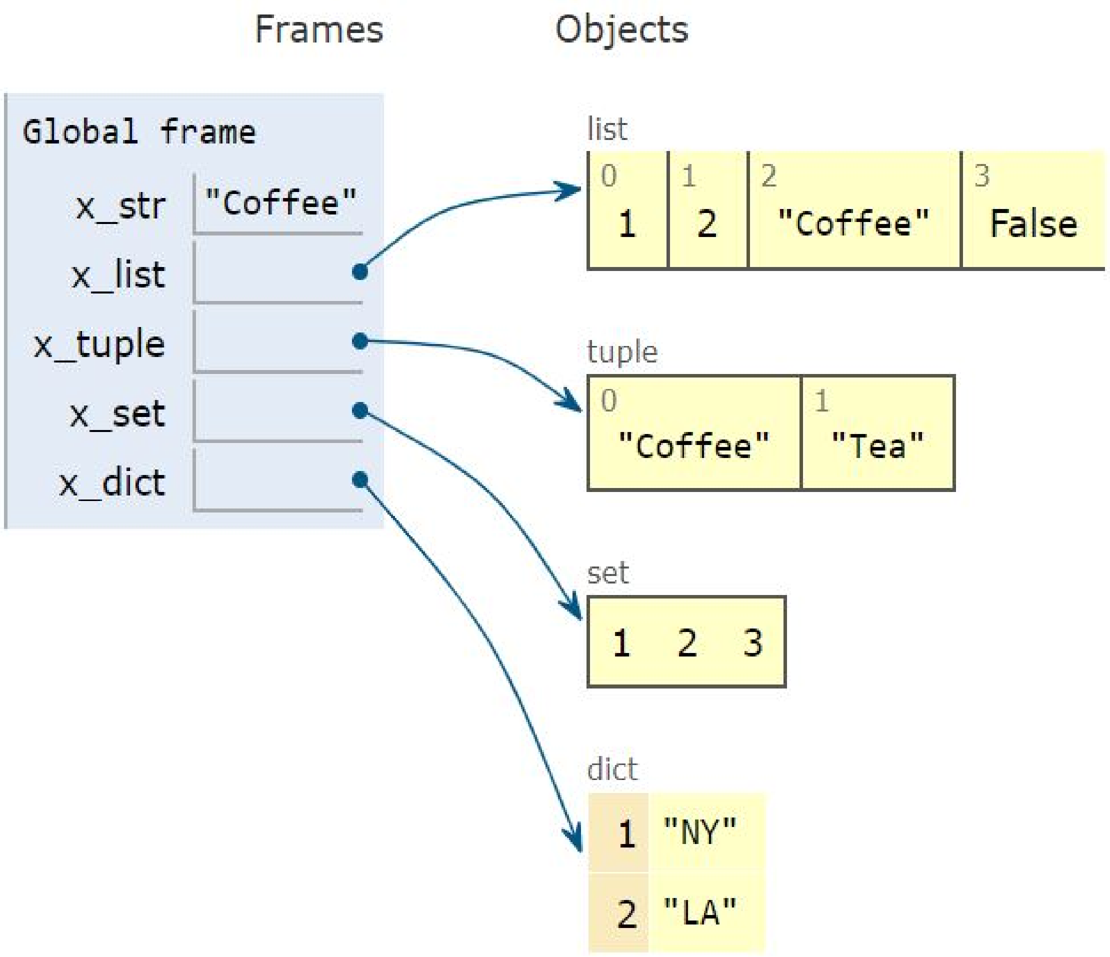
```
x_list = [3,2,1]
x_id_1 = id(x_list)
x_list.sort()
x_id_2 = id(x_list)

y_list = [3,2,1]
y_id_1 = id(y_list)
new_y_list = sorted(y_list)
y_id_2 = id(y_list)
```

Illustration of Mutability (sorting)



x_list	
x_id_1	140280857460296
x_id_2	140280857460296
y_list	
y_id_1	140280762047112
new_y_list	
y_id_2	140280762047112

String



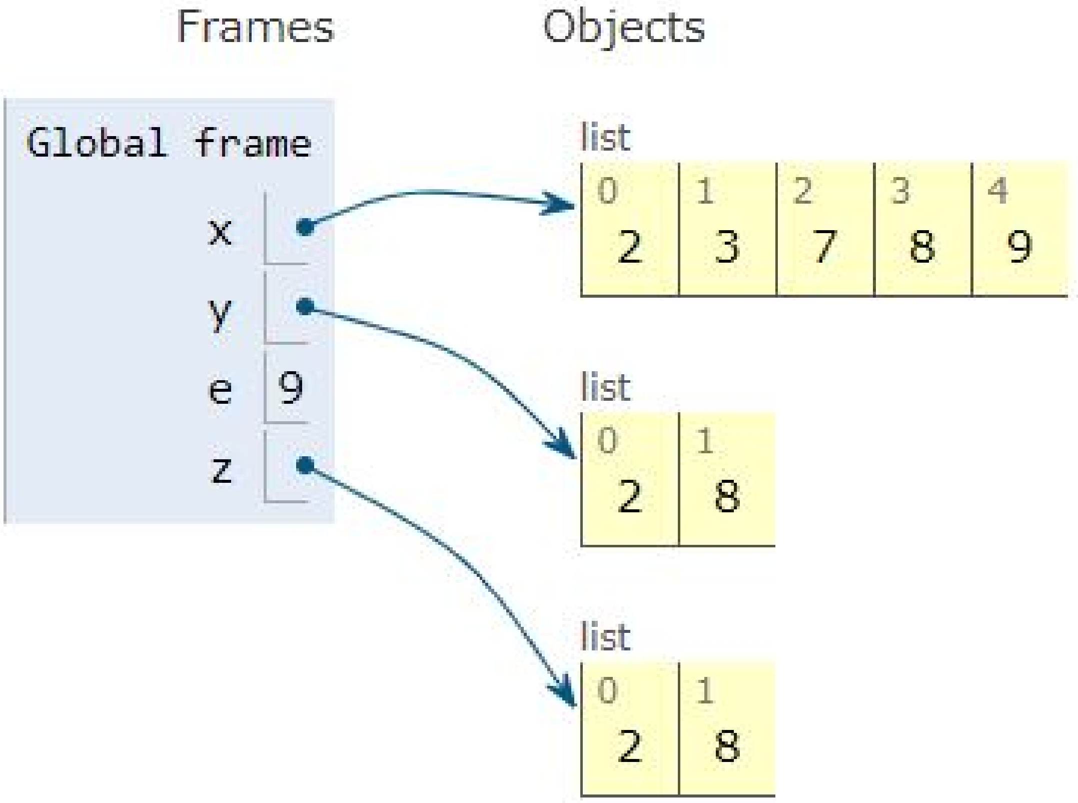
- immmutable, ordered
- items are characters

List Comprehension

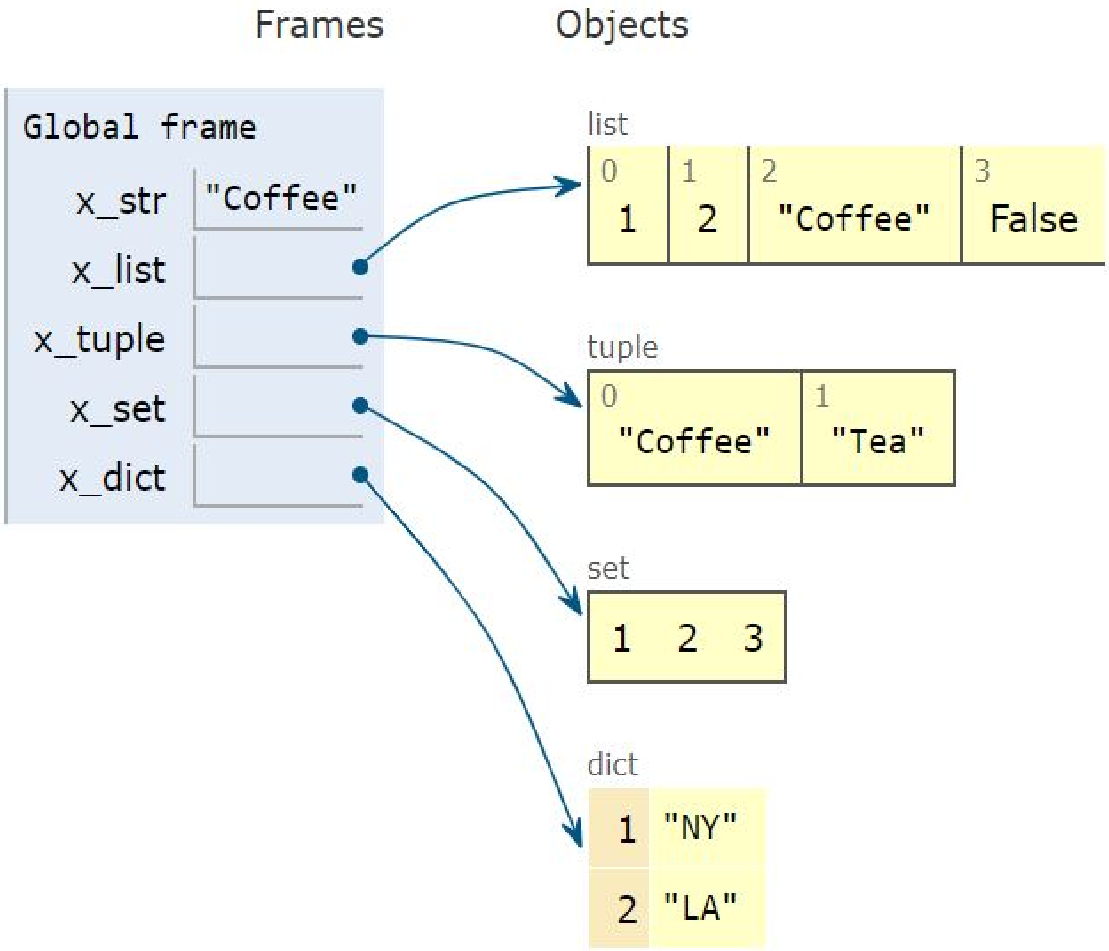
```
x = [2,3,7,8,9]

y = []
for e in x:
    if e % 2 == 0:
        y.append(e)

z = [e for e in x if e % 2 == 0]
```

- construct one list from another some condition(s)
- can use a simple iteration
- ”better”: list comprehension

List Comprehension Illustration

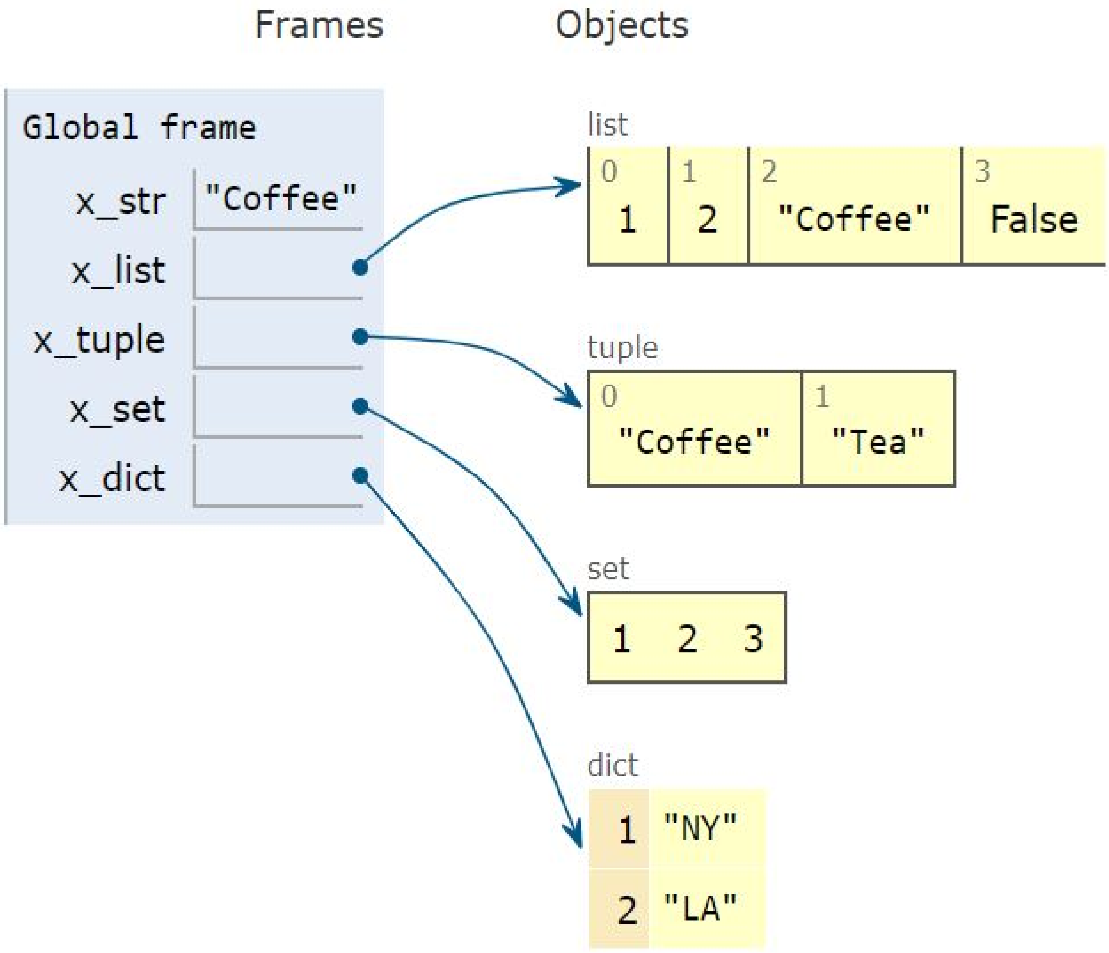


Tuple



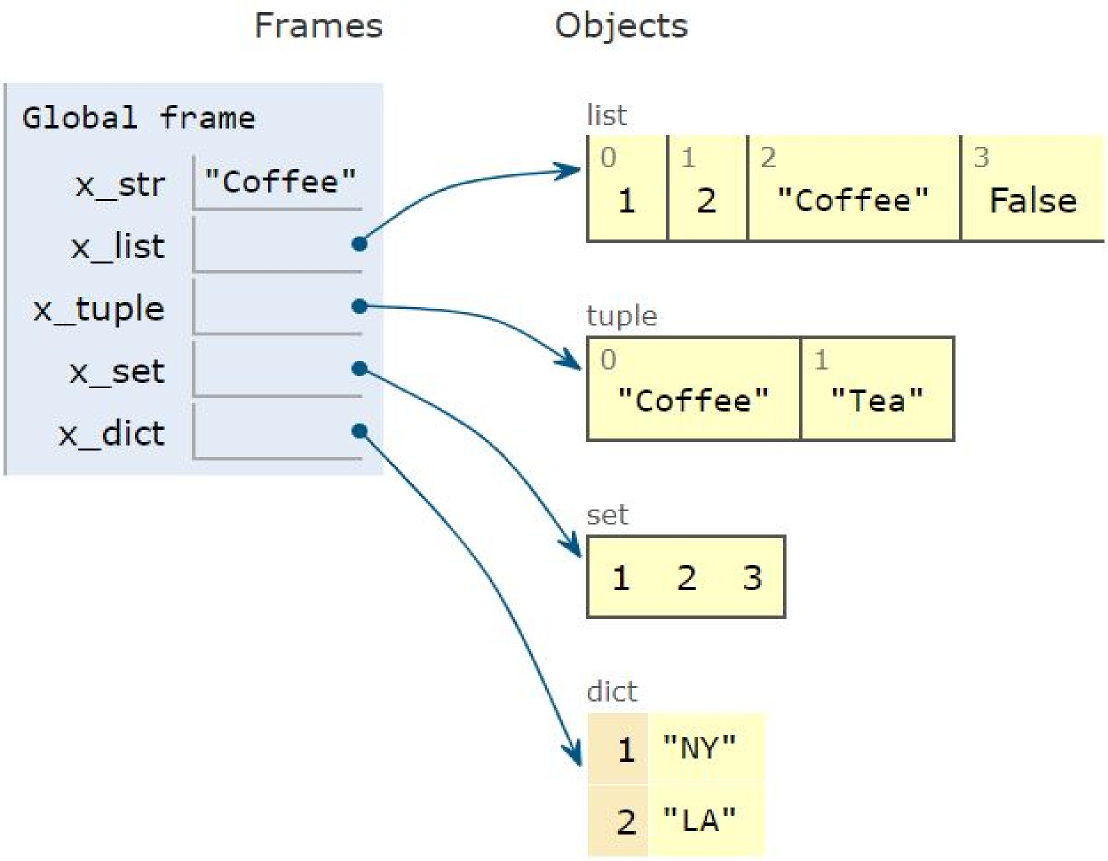
- immutable, ordered
- items of any type

Set



- mutable, unordered
- items are immutable

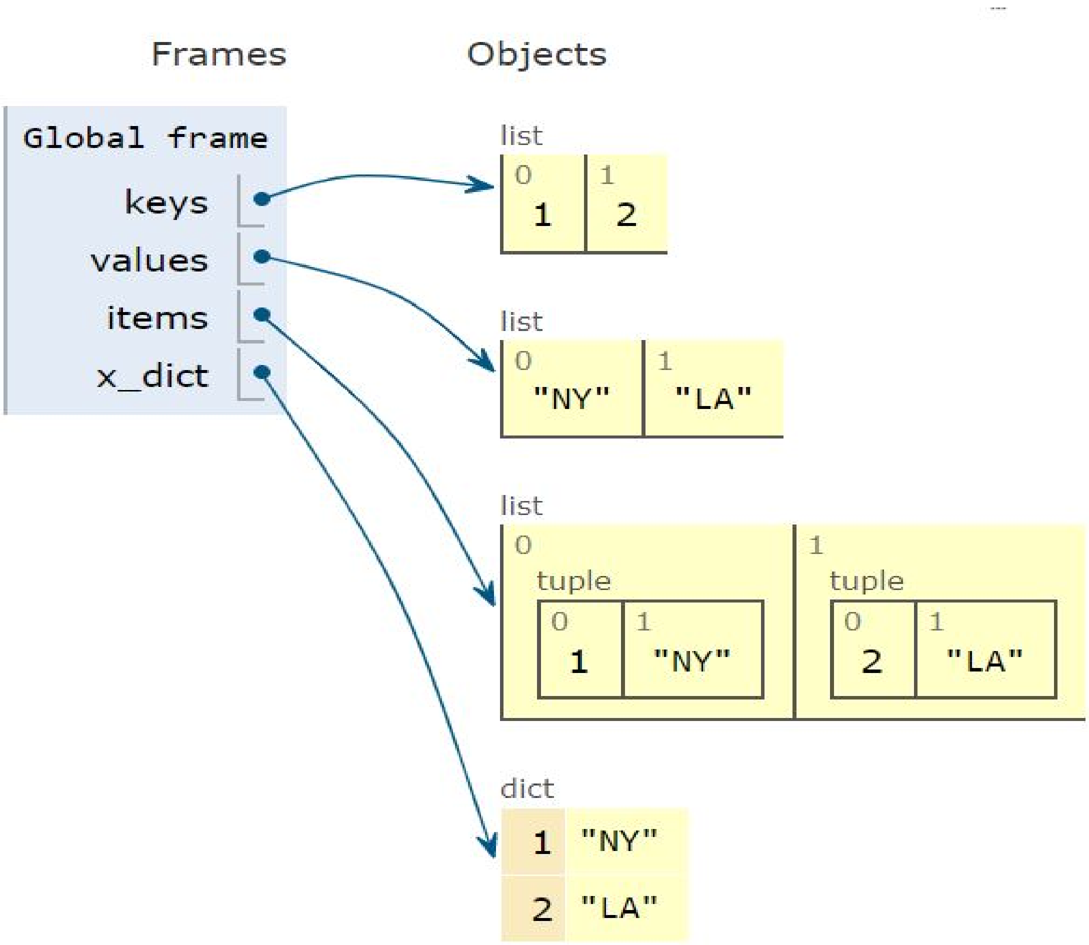
Dictionary



- mutable, key-value pairs
- keys immutable

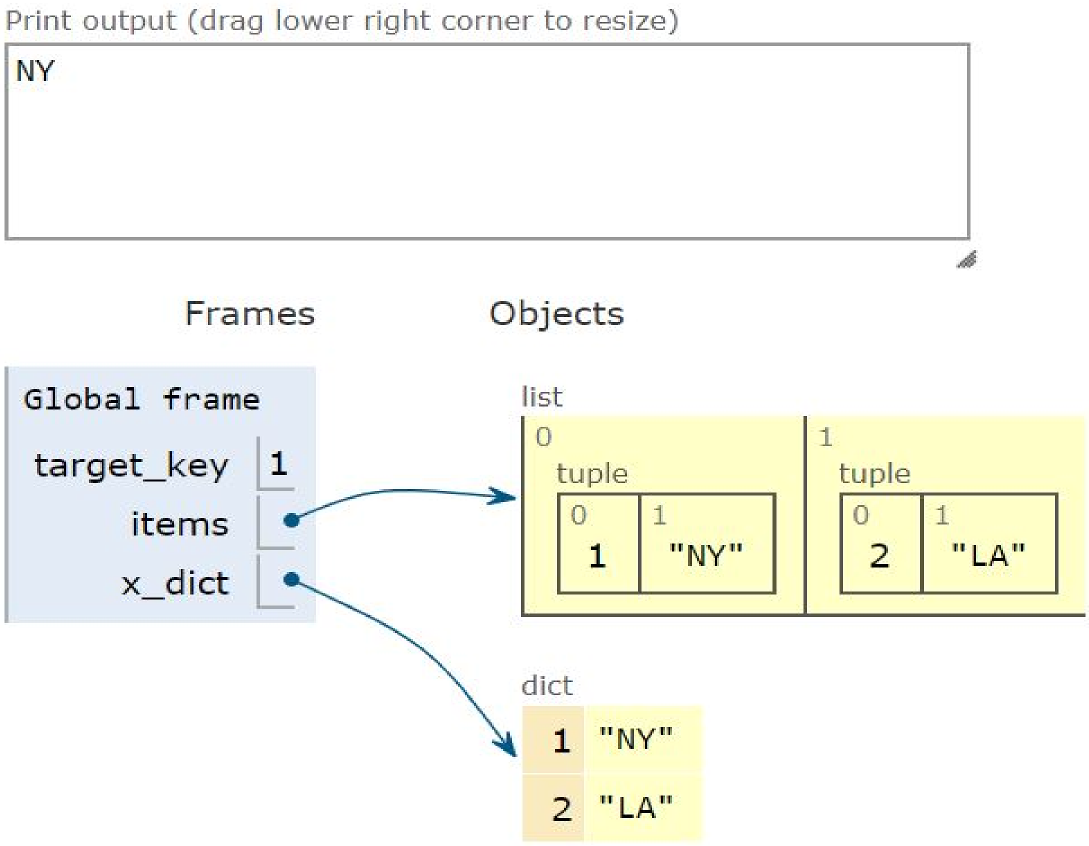
Creating a Dictionary

```
keys = [1,2]
values = ["NY", "LA"]
items = list(zip(keys, values))
x_dict = dict(items)
```



Accessing a Dictionary

```
target_key = 1
items = list(zip([1,2], ["NY", "LA"]))
x_dict = dict(items)
if target_key in x_dict.keys():
    print(x_dict[target_key])
```

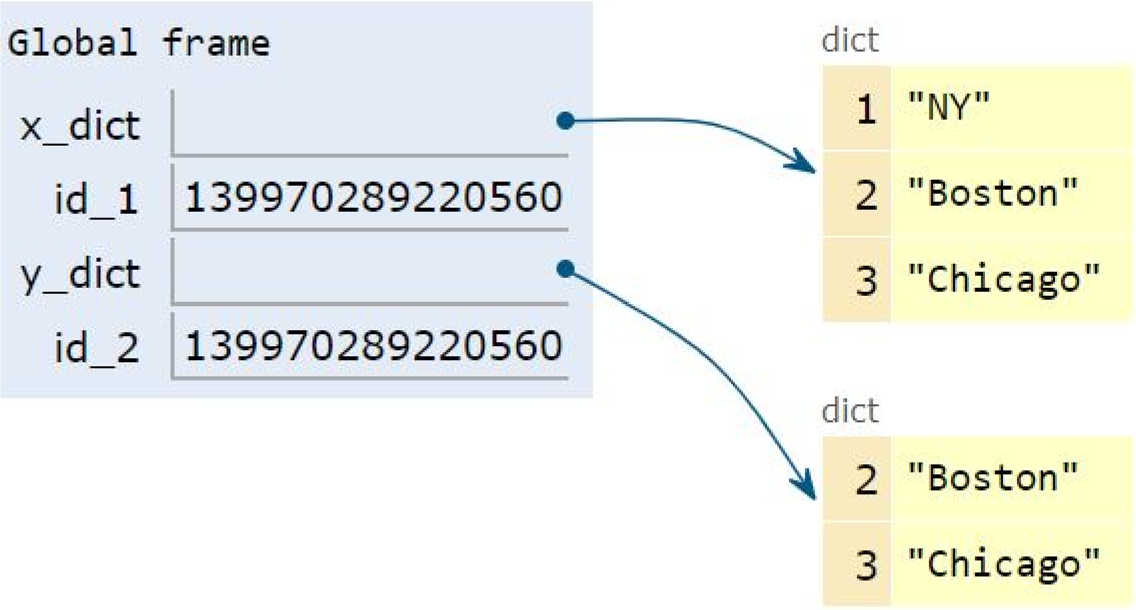


Print output (drag lower right corner to resize)

```
NY
```

Updating a Dictionary

```
x_dict = dict(zip([1,2],["NY","LA"]))
id_1   = id(x_dict)
y_dict = dict(zip([2,3],
              ["Boston","Chicago"]))
x_dict.update(y_dict)
id_2 = id(x_dict)
```



Global frame

x_dict	
id_1	139970289220560
y_dict	
id_2	139970289220560

dict

1	"NY"
2	"Boston"
3	"Chicago"

dict

2	"Boston"
3	"Chicago"

- mutable - ”in-place” update

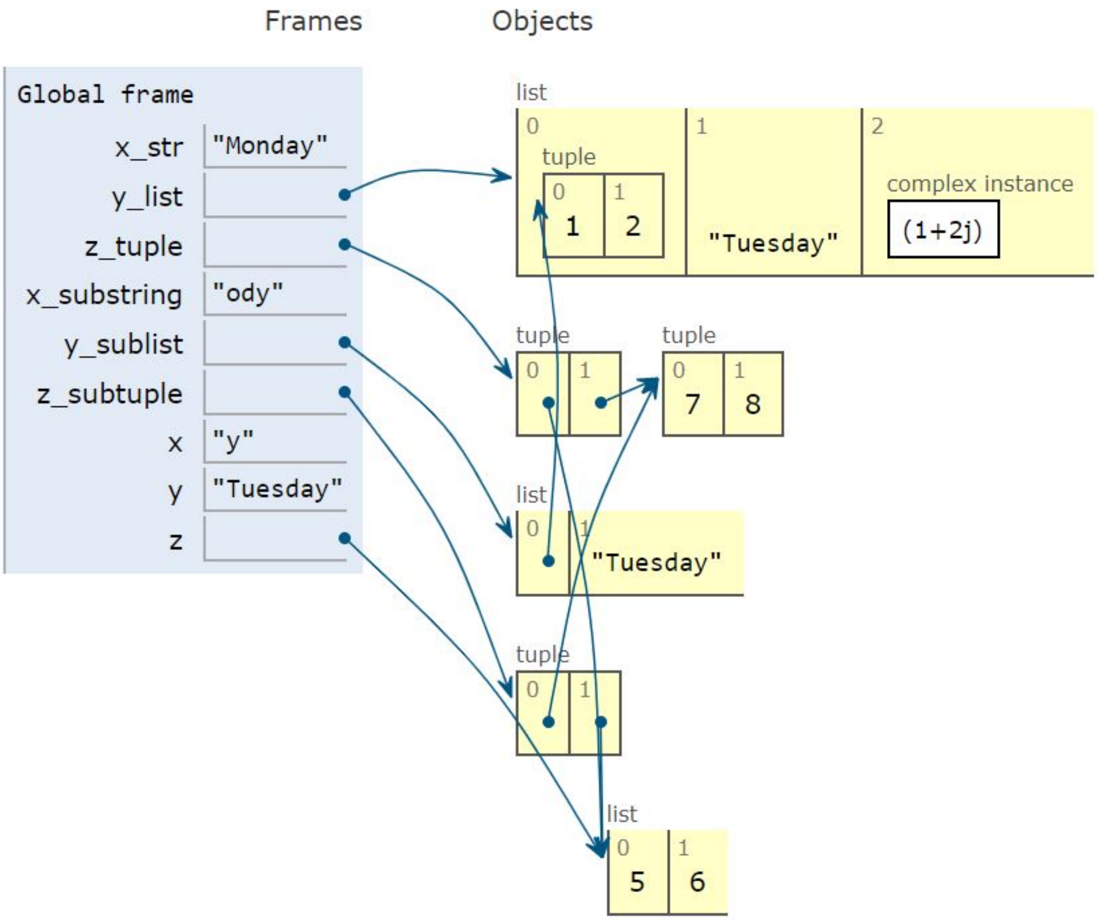
Indexing & Slicing

```
x_str = 'Monday'
y_list = [(1,2), 'Tuesday', 1+2j]
z_tuple = ([5,6], (7, 8) )
x = x_str[4]
y = y_list[1]
z = z_tuple[0]
x_substring = x_str[1: : 2]
y_sublist = y_list[: 2]
z_subtuple = z_tuple[: : -1]
```

- applies to lists, tuples and strings
- access element by index
- use slicing for sub-containers
- slice specification:

  $\left[\ \text{start} : \ \text{end} + 1 : \ \text{step}\ \right]$

Indexing & Slicing Illustration

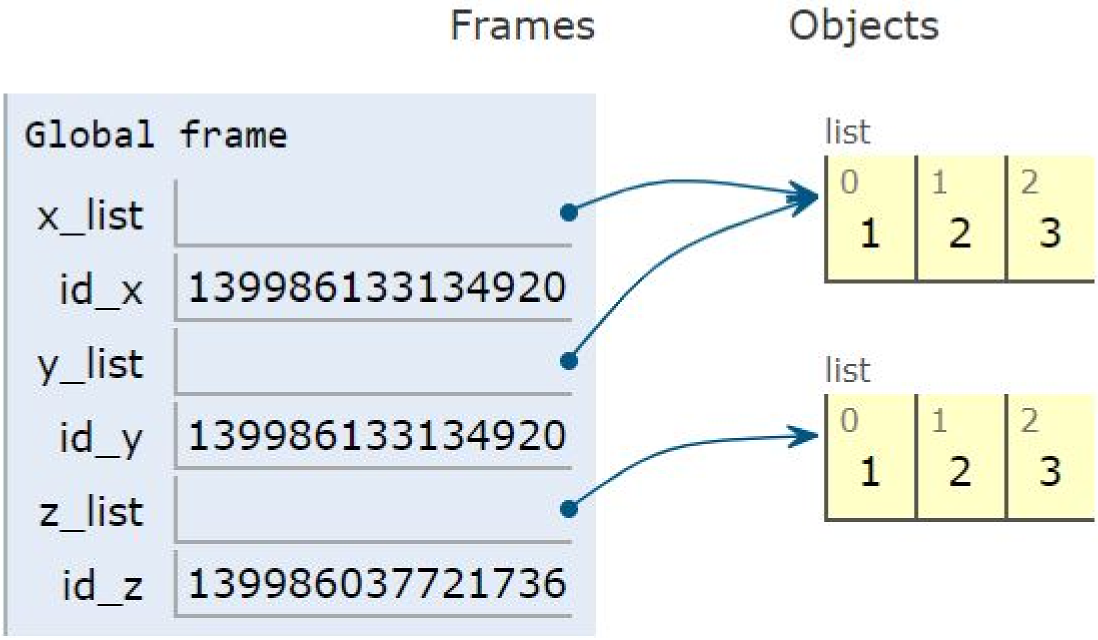


Copying With Slicing

```
x_list = [1, 2, 3]
id_x   = id(x_list)

y_list = x_list
id_y   = id(y_list)

z_list = x_list[ : ]
id_z   = id(z_list)
```

- for lists with immutable elements, can use [:] to create a copy
- does not work with mutable elements
- use *copy* module

List Copy Illustration



Frames

Objects

Global frame

x_list

id_x 139986133134920

y_list

id_y 139986133134920

z_list

id_z 139986037721736

list

0	1	2
1	2	3

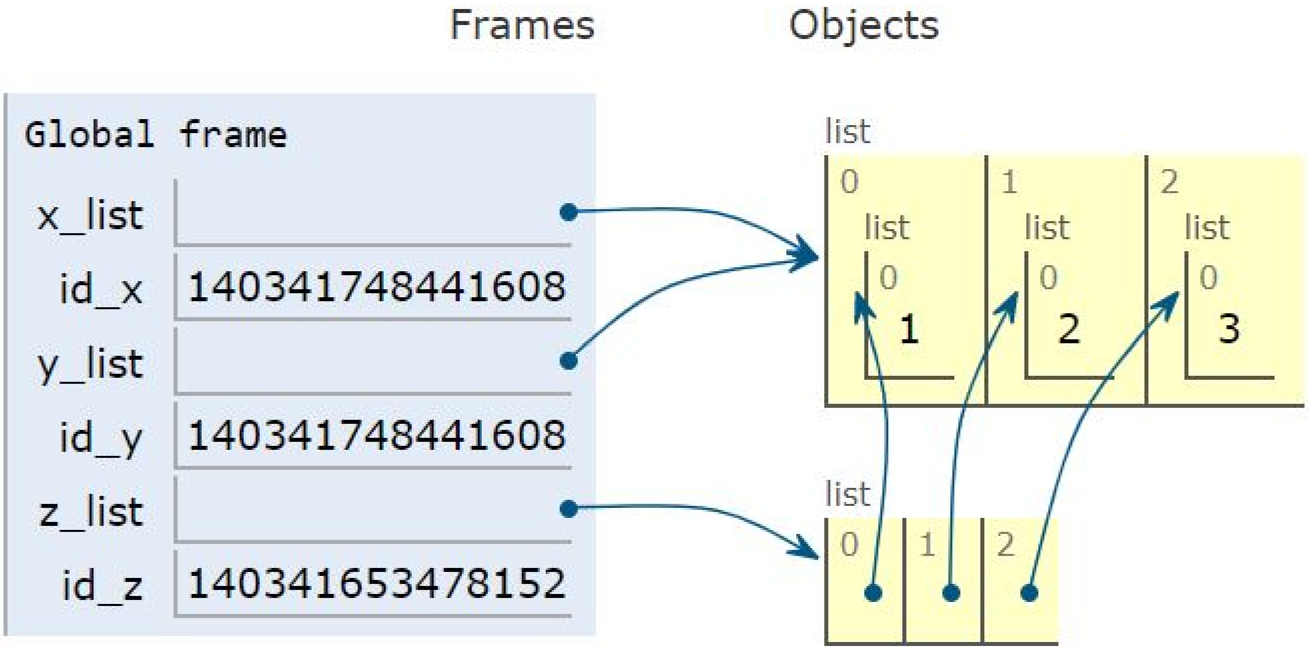
list

0	1	2
1	2	3

- ”[:]” makes a copy

Copying With Slicing

```
x_list = [[1], [2], [3]]
id_x   = id(x_list)
y_list = x_list
id_y   = id(y_list)
z_list = x_list[ : ]
id_z   = id(z_list)
```



Global frame

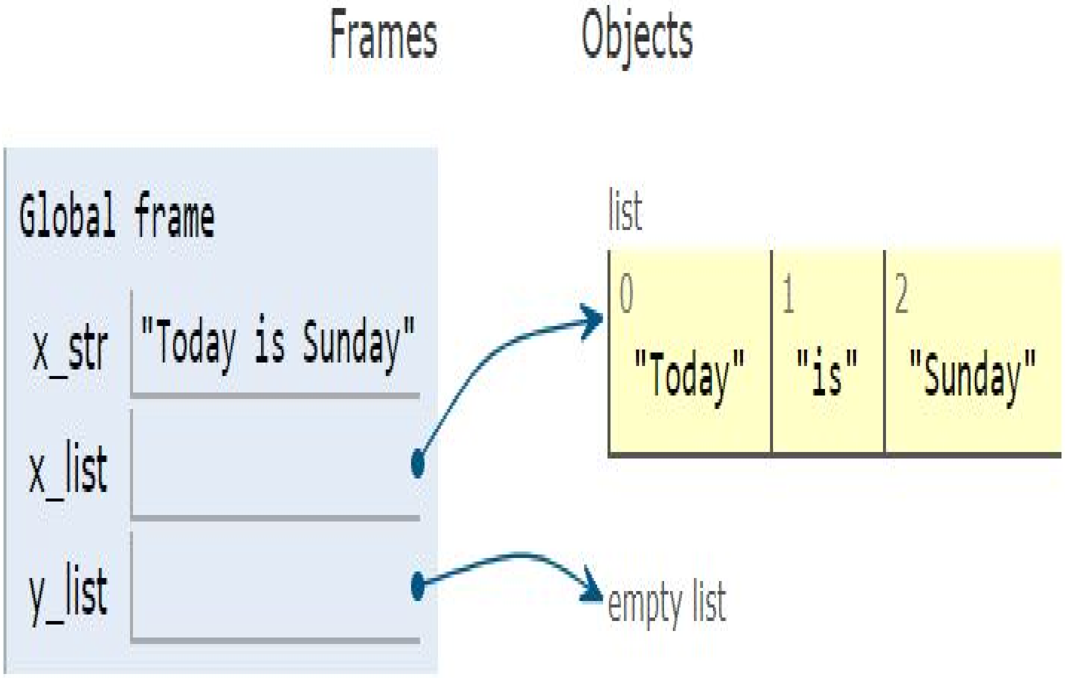
x_list	
id_x	140341748441608
y_list	
id_y	140341748441608
z_list	
id_z	140341653478152

Iteration Example (1)

- reverse all words in string ”Today is Sunday”

```
x_str = "Today is Sunday"

x_list = x_str.split()
y_list = []

for word in x_list:
    reverse_word = word[ : : -1]
    y_list.append(reverse_word)

y_str = " ".join(y_list)
```

Iteration Example (2)



x_str "Today is Sunday"

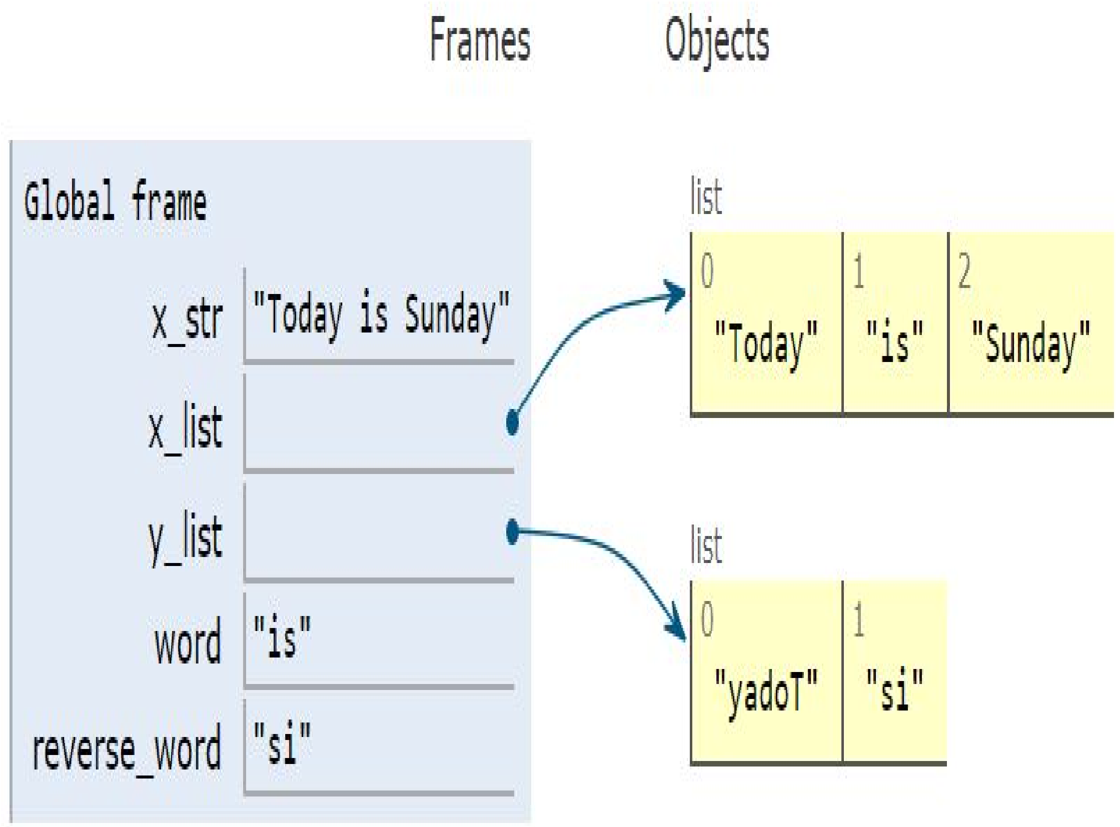
x_list

y_list

0	1	2
"Today"	"is"	"Sunday"

- use *split*() to create a list of words

Iteration Example (3)



Global frame

x_str	"Today is Sunday"
x_list	
y_list	
word	"is"
reverse_word	"si"

list

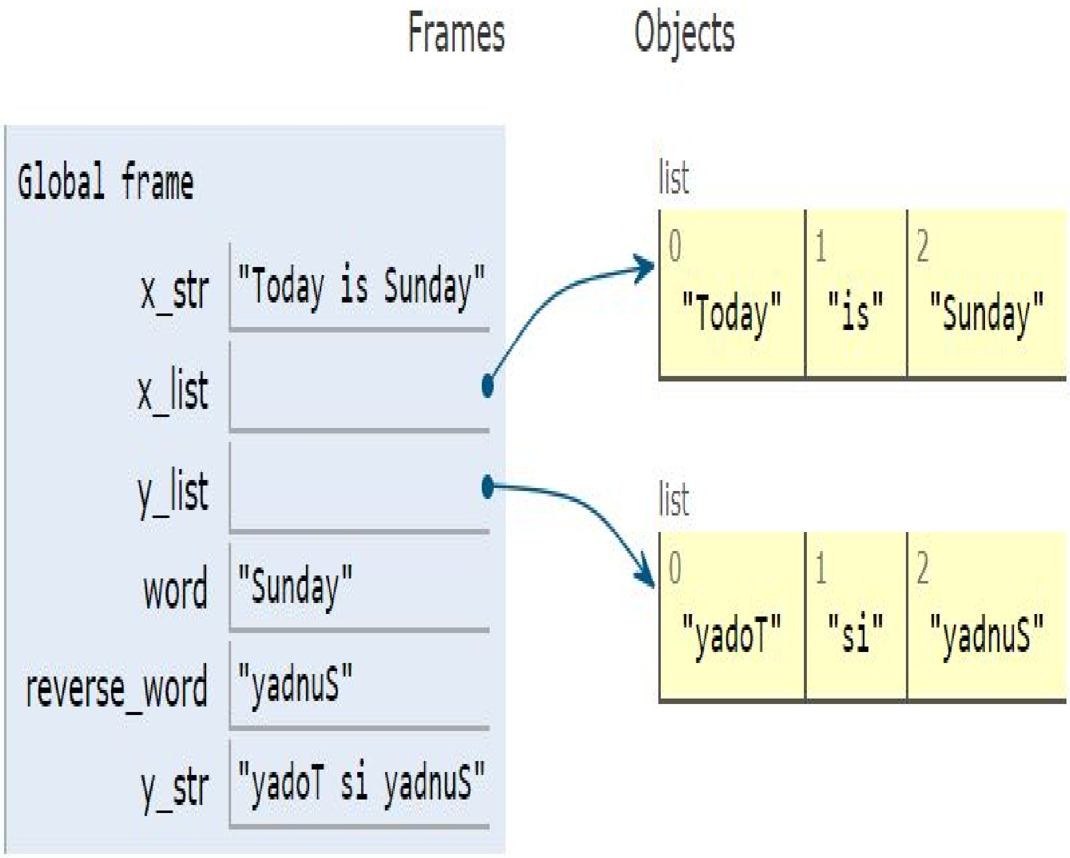
0	1	2
"Today"	"is"	"Sunday"

list

0	1
"yadoT"	"si"

- use ”[::-1]” to reverse string

Iteration Example (4)



Global frame

x_str	"Today is Sunday"
x_list	
y_list	
word	"Sunday"
reverse_word	"yadnuS"
y_str	"yadoT si yadnuS"

list

0	1	2
"Today"	"is"	"Sunday"

list

0	1	2
"yadoT"	"si"	"yadnuS"

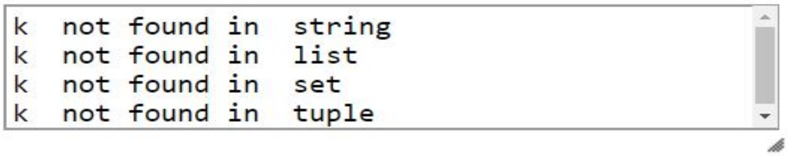
- use *join*() to construct a string from a list

Membership Example

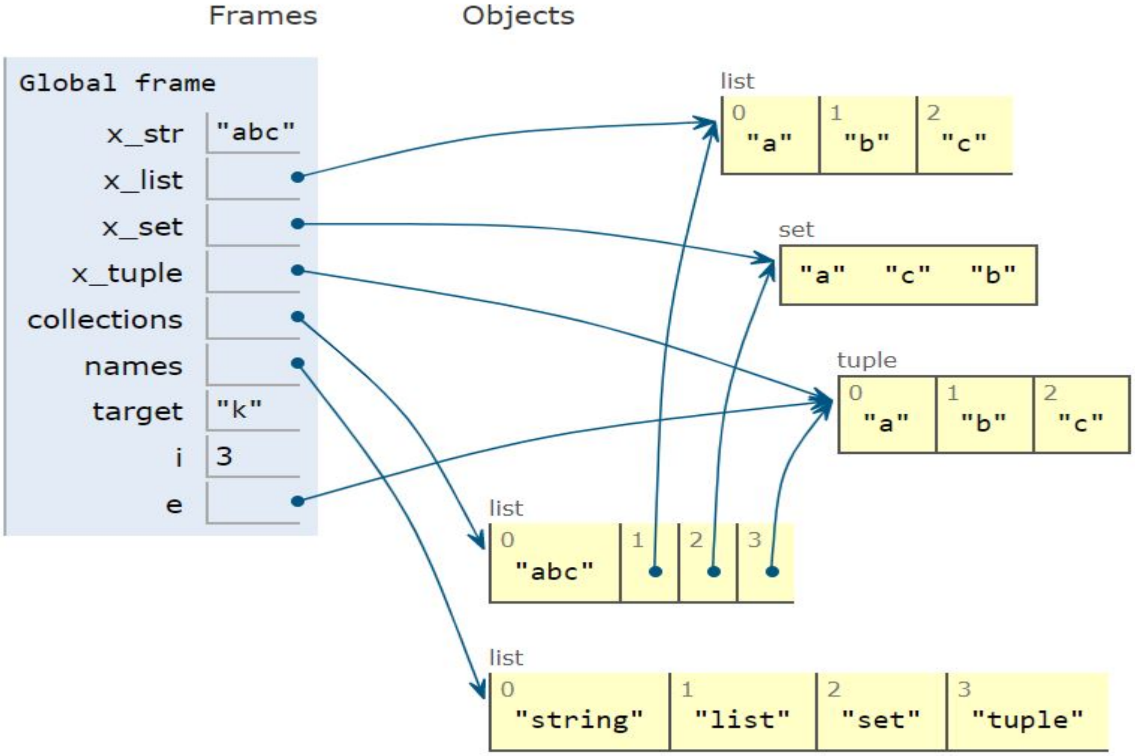
```
x_str   = "abc"
x_list  = ["a", "b", "c"]
x_set   = {"a", "b", "c"}
x_tuple = ("a", "b", "c")

collections = [x_str, x_list,
               x_set, x_tuple]
names = ["string","list","set","tuple"]

target = "k"
for i, e in enumerate(collections):
    if target not in e:
        print(target, " not found in ",
               names[i])
```

- in/not in for any container
- can get both index and next element via *enumerate*()

Membership Illustrated



```
k  not found in  string
k  not found in  list
k  not found in  set
k  not found in  tuple
```

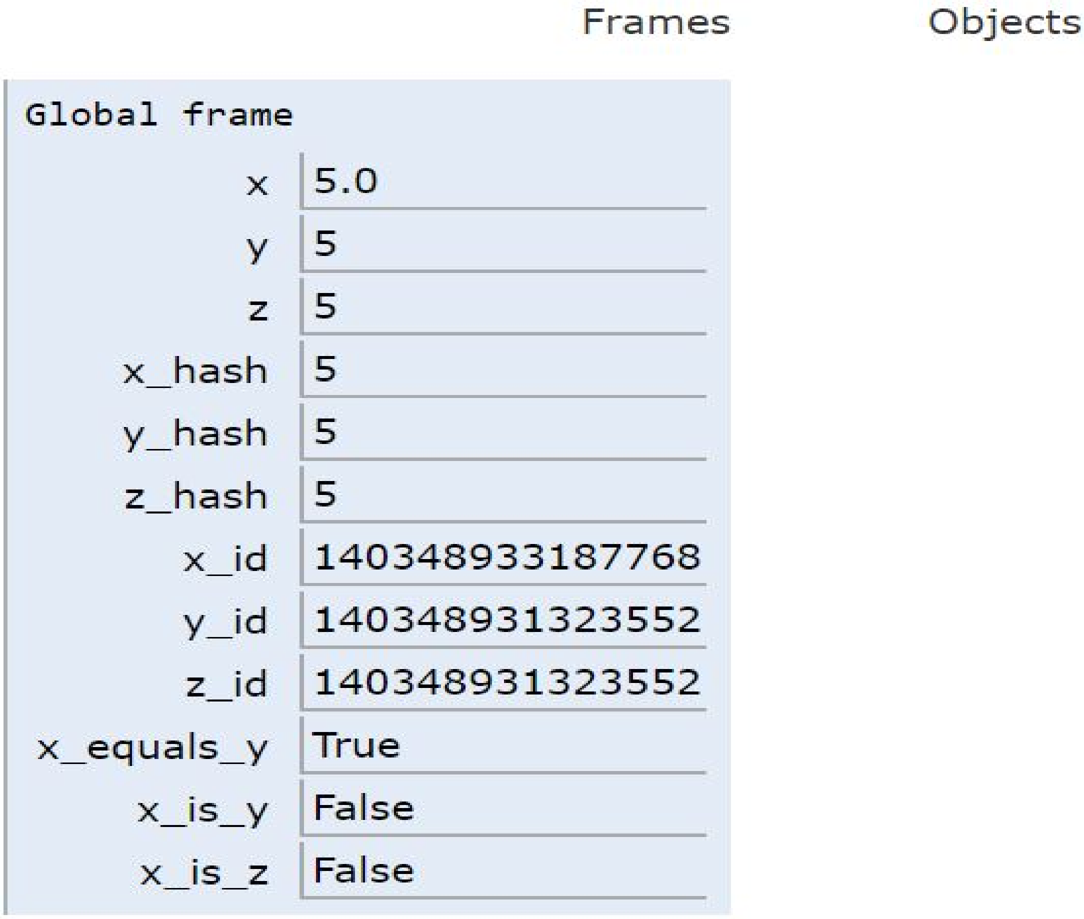


Object Equality

```
x = 5.0; y = 5; z = 5

x_hash = hash(x)
y_hash = hash(y)
z_hash = hash(z)

x_id = id(x); y_id = id(y); z_id = id(z)

x_equals_y = (x == y)    # true:
x_is_y     = (x is y)    # false:
x_is_z     = (x is z)    # false:
```

- ”==” compares (hash) values
- ”is” compares (id) addresses

Object Equality Illustration

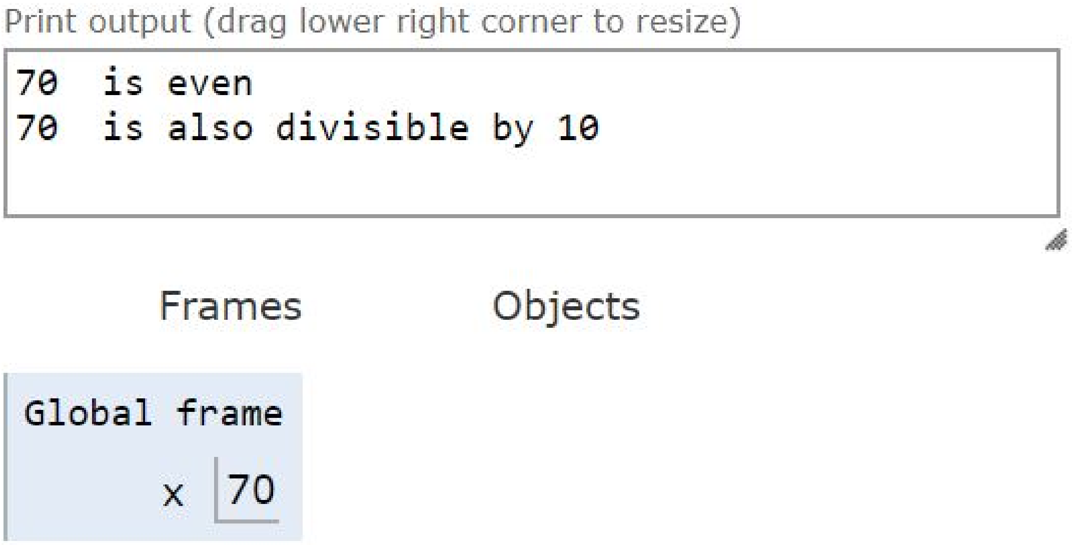


Global frame

x	5.0
y	5
z	5
x_hash	5
y_hash	5
z_hash	5
x_id	140348933187768
y_id	140348931323552
z_id	140348931323552
x_equals_y	True
x_is_y	False
x_is_z	False

Control Flow

```
x = 70
if x % 2 == 0:
    print(x, " is even")
    if x % 5 == 0:
        print(x, "also divisible by 10")
else:
    print(x, " is odd ")
```

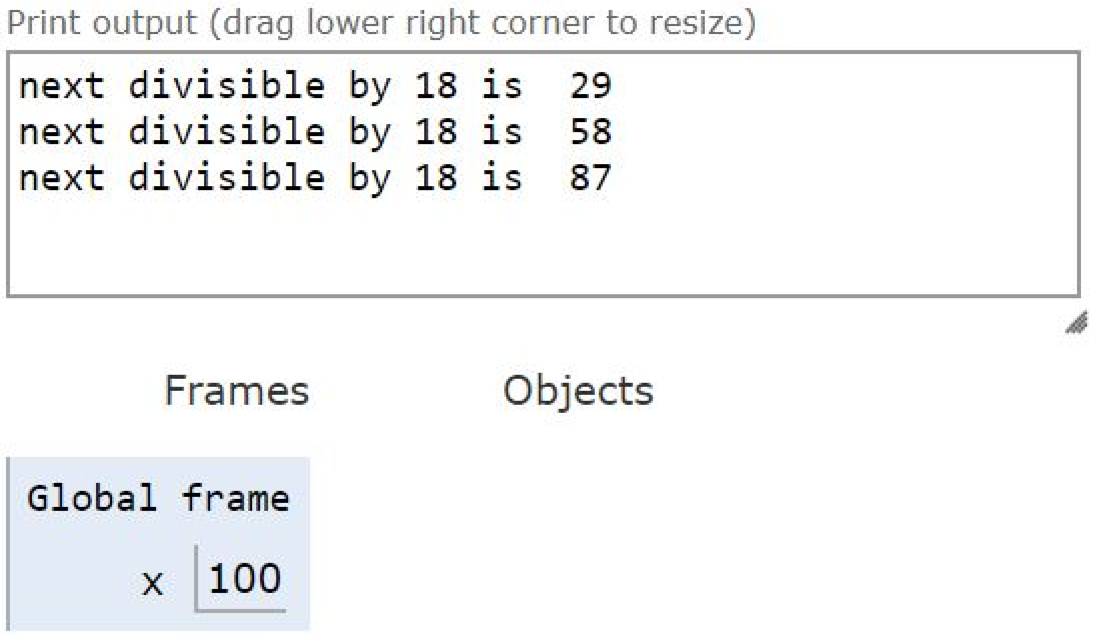


```
70  is even
70  is also divisible by 10
```

- use indentation

While Loops

```
x = 1
while x < 100:
    if x % 29 == 0:
        print("next div. by 18 is", x)
    x = x + 1
```



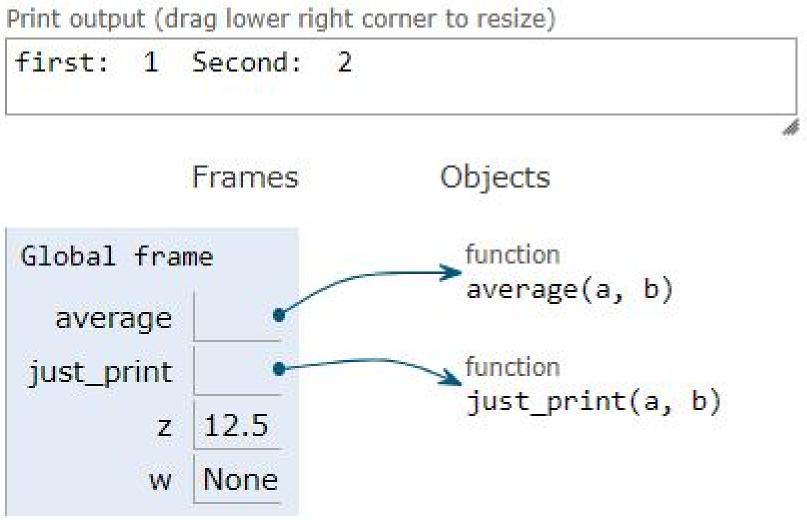
```
next divisible by 18 is  29
next divisible by 18 is  58
next divisible by 18 is  87
```

Functions

```
def average(a,b):
    """ docstring - average """
    return (a+b)/2.0

def just_print(a,b):
    print("first: ", a, " Second: ", b)

z = average(10,15)
w = just_print(1, 2)
```

- use *def* keyword
- always return a value
- returns *None* if no explicit return statement
- parameters passes and returns as tuples

Function Illustration



Print output (drag lower right corner to resize)

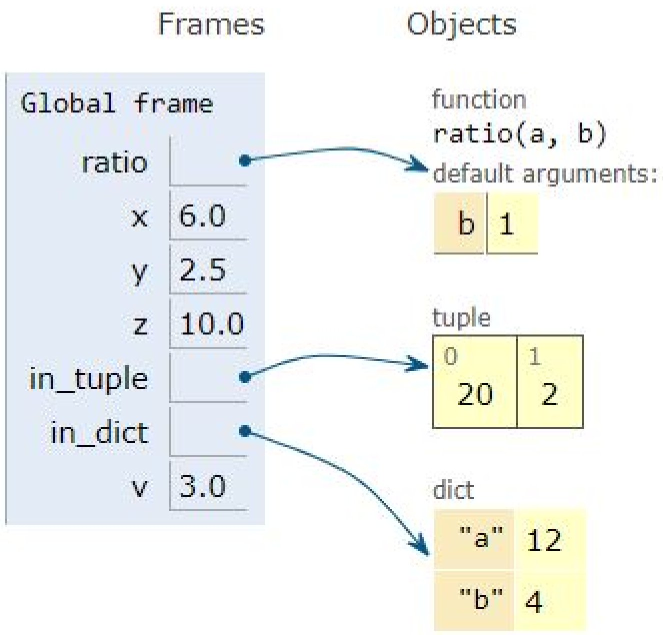
```
first:  1  Second:  2
```

Parameter Passing

```
def ratio(a, b = 1):
    return float(a)/b

x = ratio(6)              # can omit b

y = ratio(5, 2)           # positional

z = ratio(b = 2, a = 8)   # named keyword

in_tuple = (20, 2)        # single *
z = ratio(*in_tuple)

in_dict = {"a":12,"b":4}  # double **
v = ratio(**in_dict)
```

- many ways to pass parameters

Parameter Passing Illustration

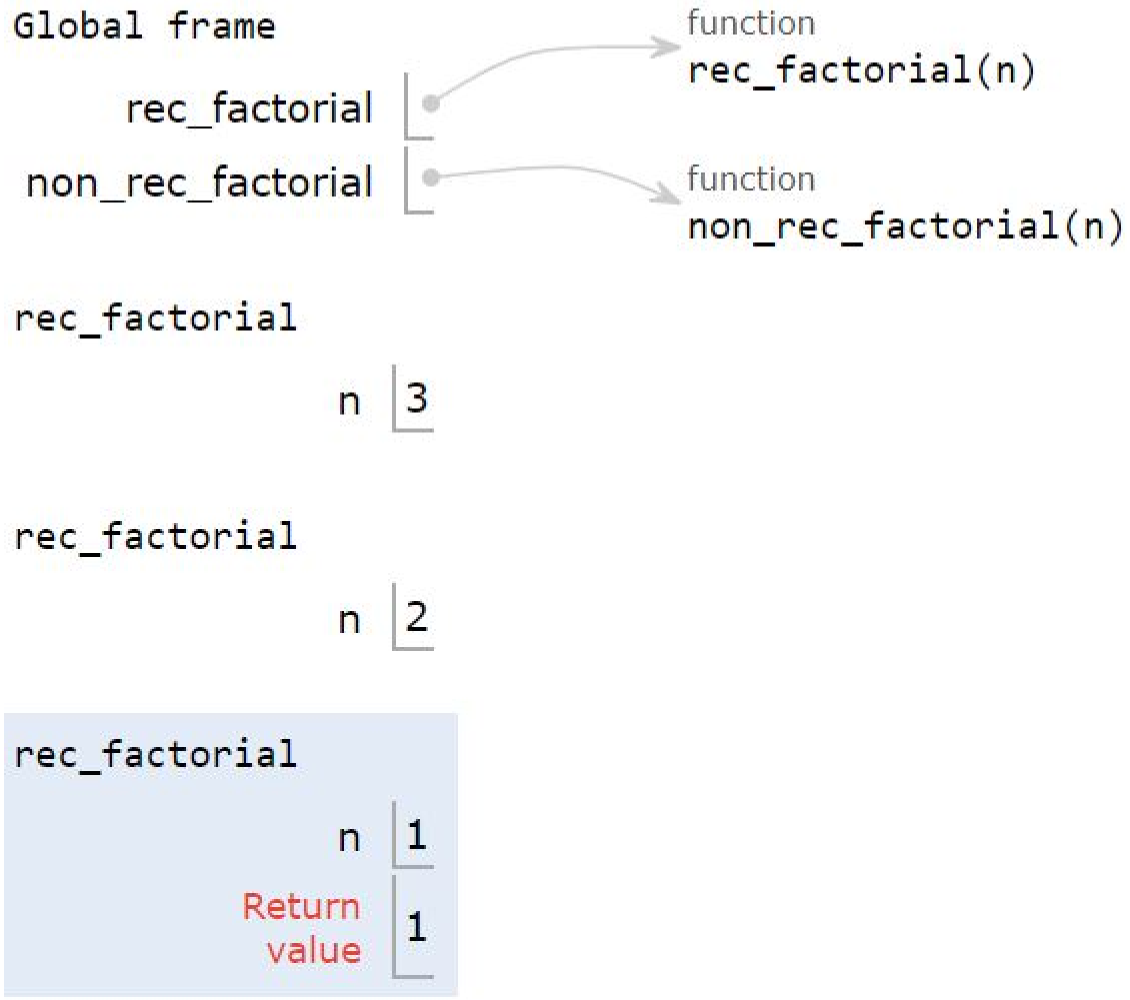


Recursive Functions

```
def rec_factorial(n):
    if  n <=0:
        return 1
    else:
        return n * rec_factorial(n-1)

def non_rec_factorial(n):
    res = 1
    for i in range(1, n+1):
        res = res * i
    return res

x = rec_factorial(3)
y = non_rec_factorial(3)
```

- recursive functions call themselves

Recursive Function Illustration



Global frame

rec_factorial → function rec_factorial(n)

non_rec_factorial → function non_rec_factorial(n)

rec_factorial

n 3

rec_factorial

n 2

rec_factorial

n 1

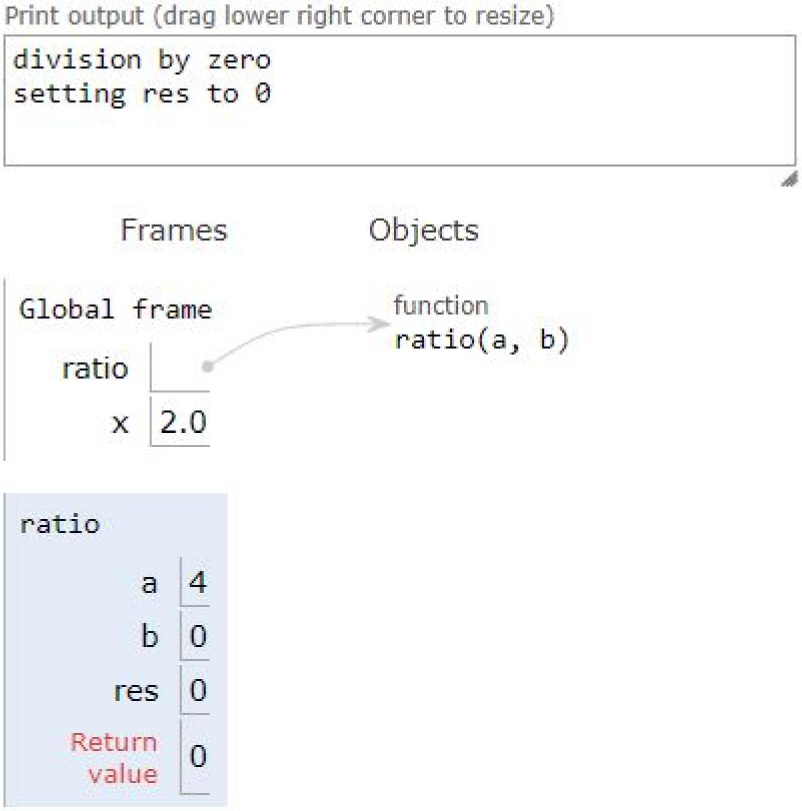
Return value 1

Try/Except

```
def ratio(a, b):
    try:
        res = a/b
    except Exception as e:
        print(e)
        print('setting res to 0')
        res = 0
    finally:
        return res

x = ratio(4, 2)
y = ratio(4, 0)
```

- exceptions are Python objects
- optional ”finally”
- can capture and process multiple exceptions

Try/Except Illustration



Print output (drag lower right corner to resize)

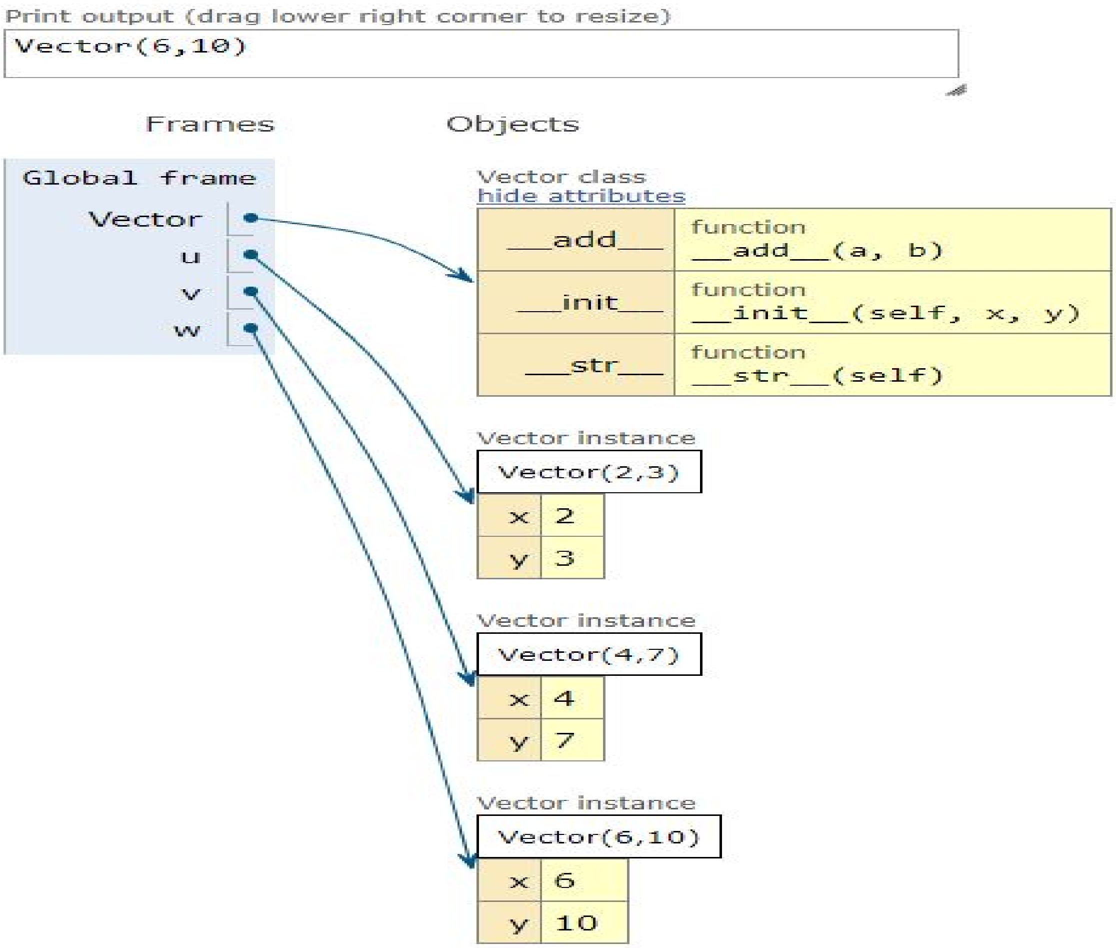
```
division by zero
setting res to 0
```

Python Class

```
class Vector():
    def __init__(self, x,y):
        self.x = x
        self.y = y
    def __str__(self):
        return "Vector(%s,%s)"\
                %(self.x,self.y)
    def __add__(a, b):
        return Vector(a.x+b.x,a.y+b.y)

u = Vector(2, 3)
v = Vector(4, 7)
w = u + v
print(w)
```

- no public/private members
- override ”+” via __add__()
- can extend by *inheritance*

Class Illustration



Print output (drag lower right corner to resize)

```
Vector(6,10)
```

Global frame

Vector	●
u	●
v	●
w	●

Vector class
hide attributes

__add__	function __add__(a, b)
__init__	function __init__(self, x, y)
__str__	function __str__(self)

Vector instance

Vector(2,3)

x	2
y	3

Vector instance

Vector(4,7)

x	4
y	7

Vector instance

Vector(6,10)

x	6
y	10

Multiple Inheritance

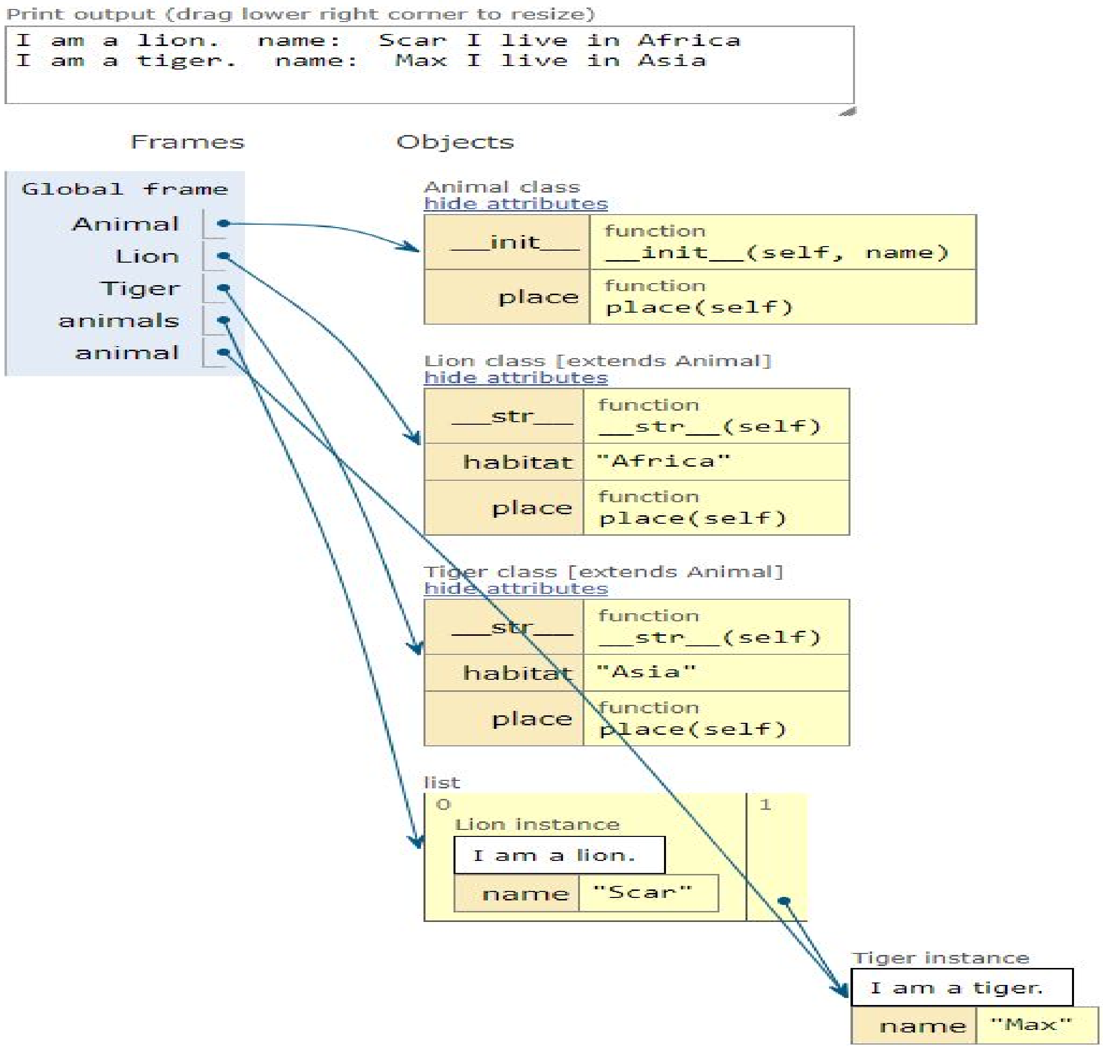
```
class Animal():
    def __init__(self, name):
          self.name = name
    def place(self):
        raise NotImplementedError("left \
                                   to subc

class Lion(Animal):
    habitat = "Africa"
    def __str__(self):
        return("I am a lion. ")
    def place(self):
        return "I live in Africa"
```

Multiple Inheritance (cont'd)

```
class Tiger(Animal):
    habitat = "Asia"
    def __str__(self):
        return("I am a tiger. ")
    def place(self):
        return "I live in Asia"

animals=[Lion("Scar"), Tiger("Max")]

for animal in animals:
    print(animal, "name: ",
          animal.name, animal.place())
```

- can simulate ”abstract” class

Inheritance Illustration



Print output (drag lower right corner to resize)

```
I  am  a  lion.   name:   Scar  I  live  in  Africa
I  am  a  tiger.   name:   Max  I  live  in  Asia
```

Optimization Example

```
import math
import scipy
def f(params):
    x = params
    return 3*(x**3) - math.exp(x)

init_guess = 0.5
optimization=scipy.optimize.minimize(f,
                       init_guess,
                       method='SLSQP')
print(optimization)
```

```
 fun: -1.3000579373912886
    jac: array([ -3.82065773e-05])
message: 'Optimization terminated successfully.'
   nfev: 16
    nit: 5
   njev: 5
 status: 0
success: True
      x: array([ 0.40895637])
```

Language Features

- objects
- namespaces and modules
- simple types and containers
- control flow

Concepts Check:

(a) interpreted language
(b) Numpy, Pandas, Matplotlib
(c) primitive data types
(d) Python collections
(e) lists, strings, tuples
(f) indexing and slicing
(g) sets and dictionaries
(h) comprehension constructs

Concepts Check:

(a) mutability

(b) membership constructs

(c) iteration with *for* and *while*

(d) control flow with *if ... else*

(e) functions

(f) parameter passing

(g) classes

(h) inheritance