Python CS-521

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

May 2, 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.

SEARCHING

Searching

- some collections are indexed
- they can be sorted for efficient searching
- can use a bisection search algorithm
- search is done in $O(\log n)$ steps

Search in Non-Indexed Collections

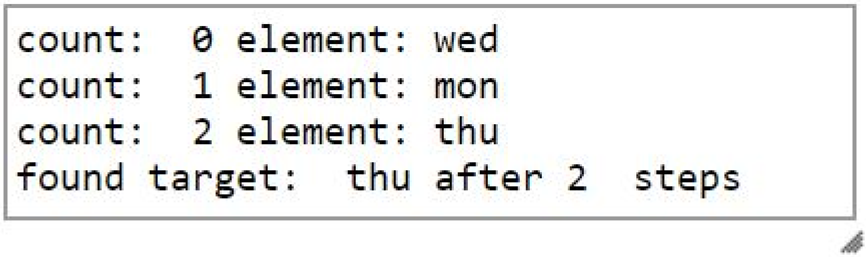
- these collections cannot be sorted
- sequential search
- need to examine $n/2$ items
- sequential vs. bisection:

#items n	**sequential:** $n/2$	**bisection:** $\log n$
100	50	8
1,000	500	10
1,000,000	500,000	20
1,000,000,000	500,000,000	30
1,000,000,000,000	500,000,000,000	40

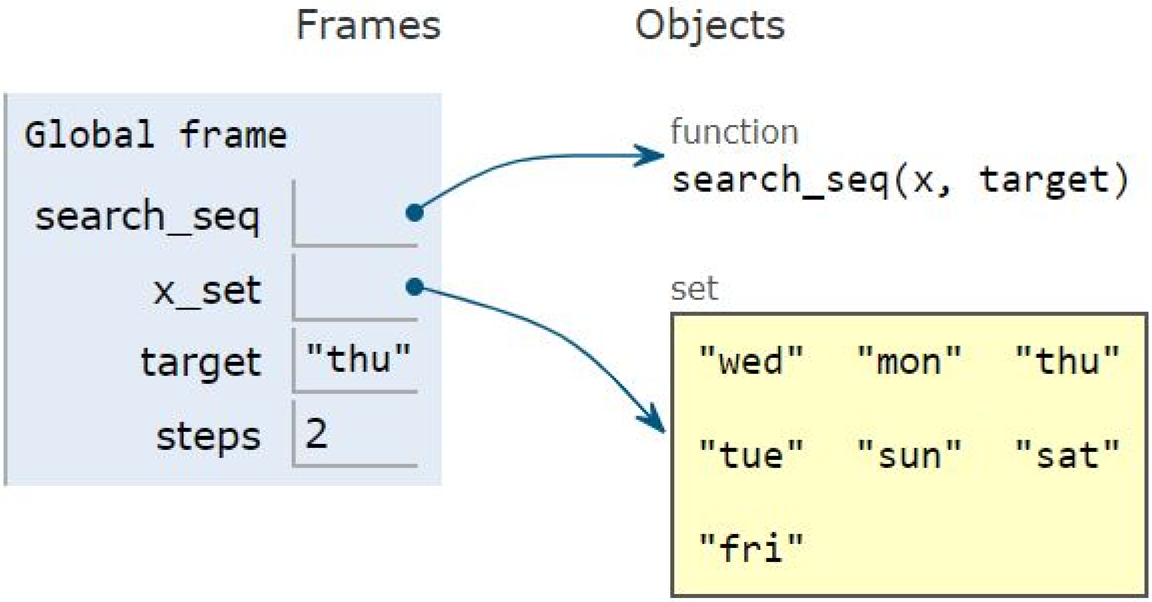
Sequential Search

```
def search_seq(x,target):
    count = 0
    for e in x:
        print(’count: ’, count, ’element:’, e)
        if e == target:
            return count
        count = count + 1
    return -1


x_set = {’mon’, ’tue’, ’wed’, ’thu’,
                ’fri’, ’sat’, ’sun’}
target=’thu’
steps = search_seq(x_set, target)
if steps >=0:
    print(’found target: ’, target,
                   ’after’,steps,’ steps’)
else:
    print(’did not find ’, target)
```

Sequential Search (cont'd)



```
count:  0 element: wed
count:  1 element: mon
count:  2 element: thu
found target:  thu after 2  steps
```



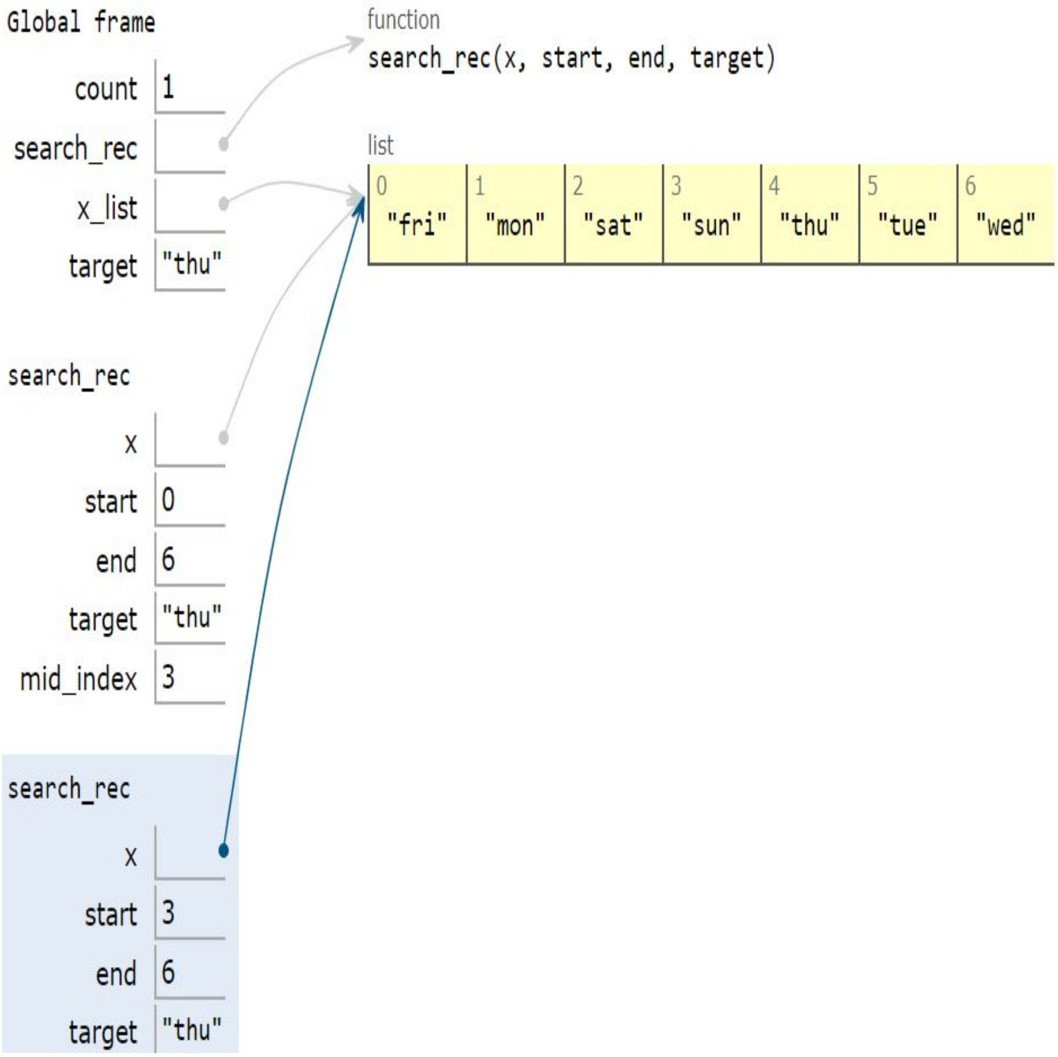
Recursive Bisection

```python
# assume x is sorted in increasing order
def search_rec(x, start, end, target):
    global count
    count   = count + 1
    if count > 10:
        return None
    if start == end:
        if x[start] == target:
            return start
        else:
            return None
    else:
        mid_index = (start+end)//2
        if x[mid_index] == target:
            return mid_index
        elif x[mid_index] < target:
            return search_rec(x, mid_index,
                              end, target)
        else:
            return search_rec(x, start,
                              mid_index,target)
```

Recursive Bisection (cont’d)

```
x_list = [’mon’, ’tue’, ’wed’, ’thu’,
                 ’fri’, ’sat’, ’sun’]
x_list.sort()
target = ’thu’
count = 0
index = search_rec(x_list, 0,
                    len(x_list)-1, target)
if index is not None:
    print(’index=’, index,’steps:’, count)
else:
    print(’target not found, steps:’, count)
```

Recursive Search Step



Global frame

count	1
search_rec	
x_list	
target	"thu"

function
search_rec(x, start, end, target)

list

0	1	2	3	4	5	6
"fri"	"mon"	"sat"	"sun"	"thu"	"tue"	"wed"

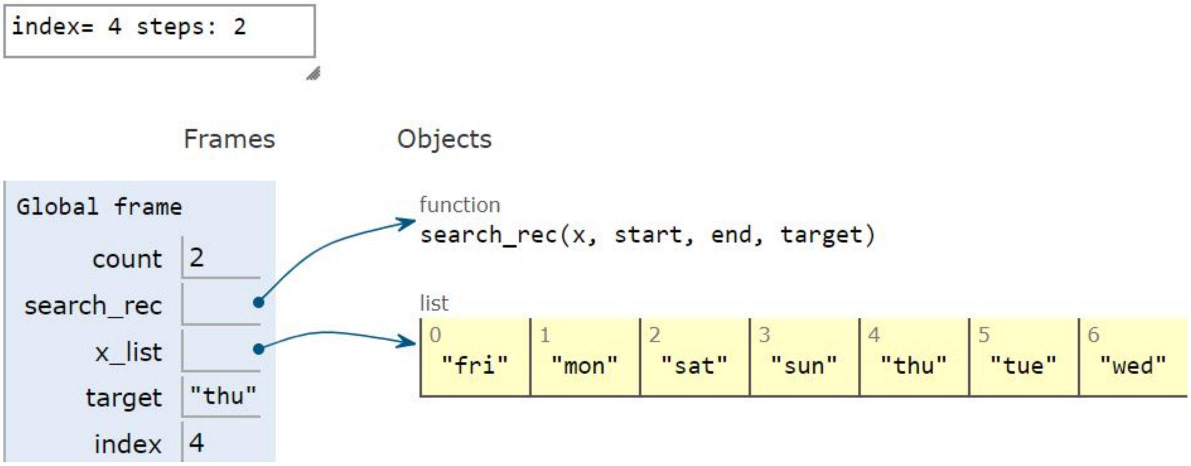
search_rec

x	
start	0
end	6
target	"thu"
mid_index	3

search_rec

x	
start	3
end	6
target	"thu"

Recursive Final Step



```
index= 4 steps: 2
```

Global frame

count	2
search_rec	
x_list	
target	"thu"
index	4

function
search_rec(x, start, end, target)

list

0	1	2	3	4	5	6
"fri"	"mon"	"sat"	"sun"	"thu"	"tue"	"wed"

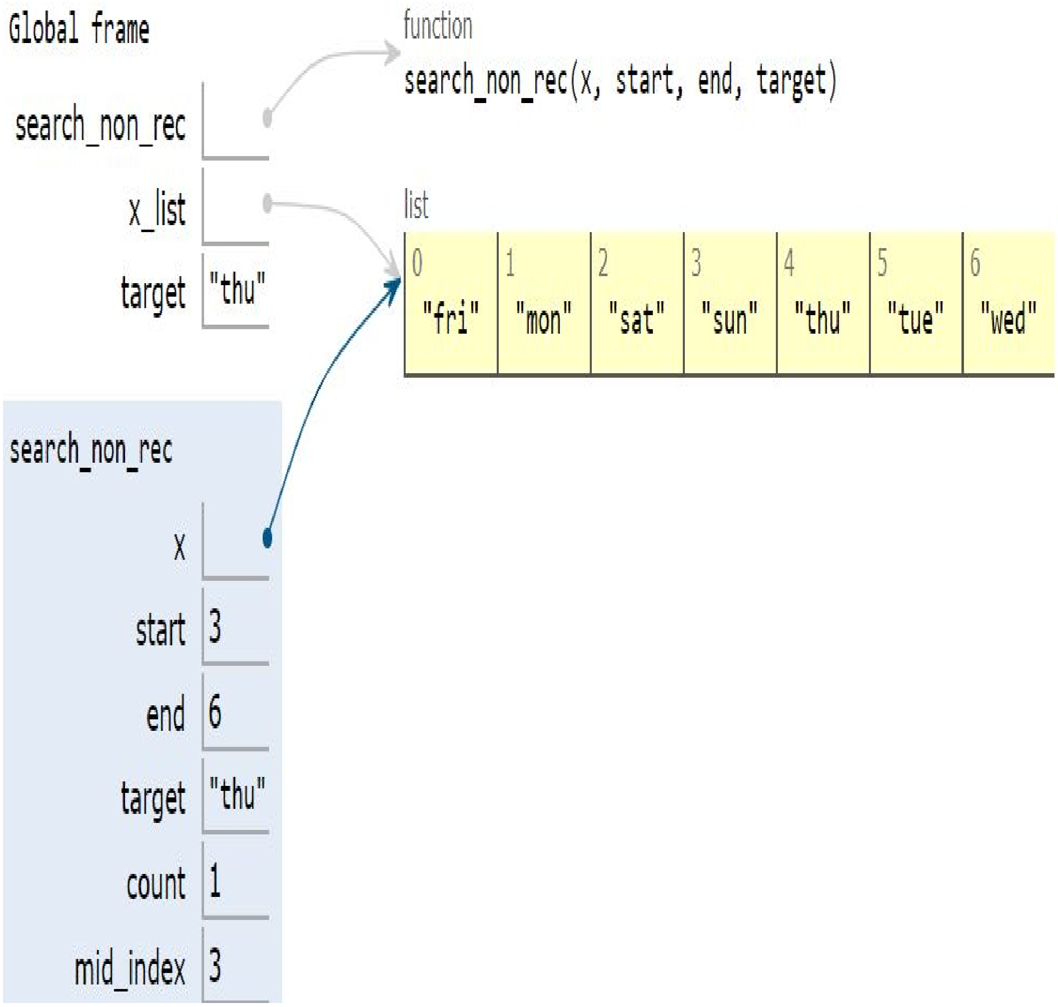
Iterative Bisection

```python
# assume x is sorted in increasing order
def search_non_rec(x, start, end, target):
    count = 0
    while count < len(x):
        count = count + 1
        if start == end:
            if x[start] == target:
                return start, count
            else:
                return None, None
        else:
            mid_index = (start+end)//2
            if x[mid_index] == target:
                return mid_index, count
            elif x[mid_index] < target:
                start = mid_index
            else:
                end = mid_index
```

Iterative Bisection (cont’d)

```
x_list = [’mon’, ’tue’, ’wed’, ’thu’,
                 ’fri’, ’sat’, ’sun’]
x_list.sort()
target = ’thu’

index, count  = search_non_rec(x_list, 0,
                       len(x_list)-1, target)
if index is not None:
    print(’index=’, index,’steps:’, count)
else:
    print(’target not found’)
```

Iterative Bisection Step



Global frame

search_non_rec

x_list

target "thu"

function

search_non_rec(x, start, end, target)

list

0	1	2	3	4	5	6
"fri"	"mon"	"sat"	"sun"	"thu"	"tue"	"wed"

search_non_rec

x

start 3

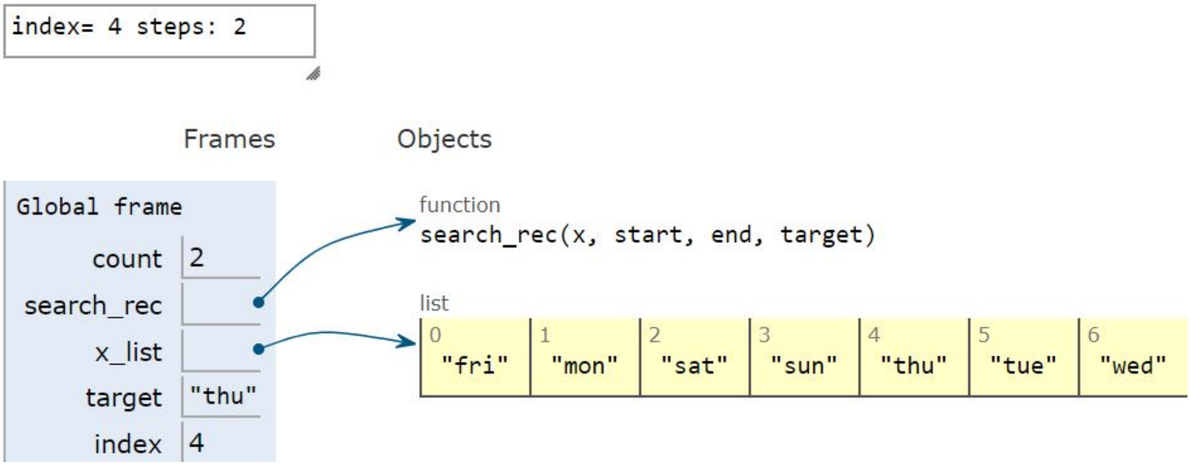
end 6

target "thu"

count 1

mid_index 3

Iterative Bisection Final Step



```
index= 4 steps: 2
```

Global frame

count	2
search_rec	
x_list	
target	"thu"
index	4

function
search_rec(x, start, end, target)

list

0	1	2	3	4	5	6
"fri"	"mon"	"sat"	"sun"	"thu"	"tue"	"wed"

Bisection Root Computation

- assume $f(x)$ is continuous
- want root r: $f(r) = 0$
- assume $a < b$, $f(a)f(b) < 0$
- root $r \in [a, b]$

(a) compute $m = (a + b)/2$

(b) if $|f(m)| < \epsilon$ then m is r

(c) else take $[a, m]$ and $[m, b]$

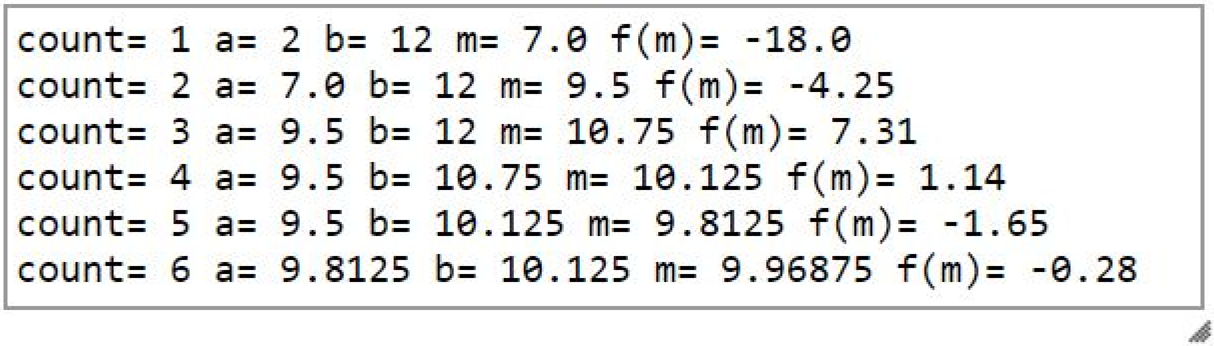
(d) look for r in $[a, m]$ or $[m, b]$

(e) recursively or iteratively

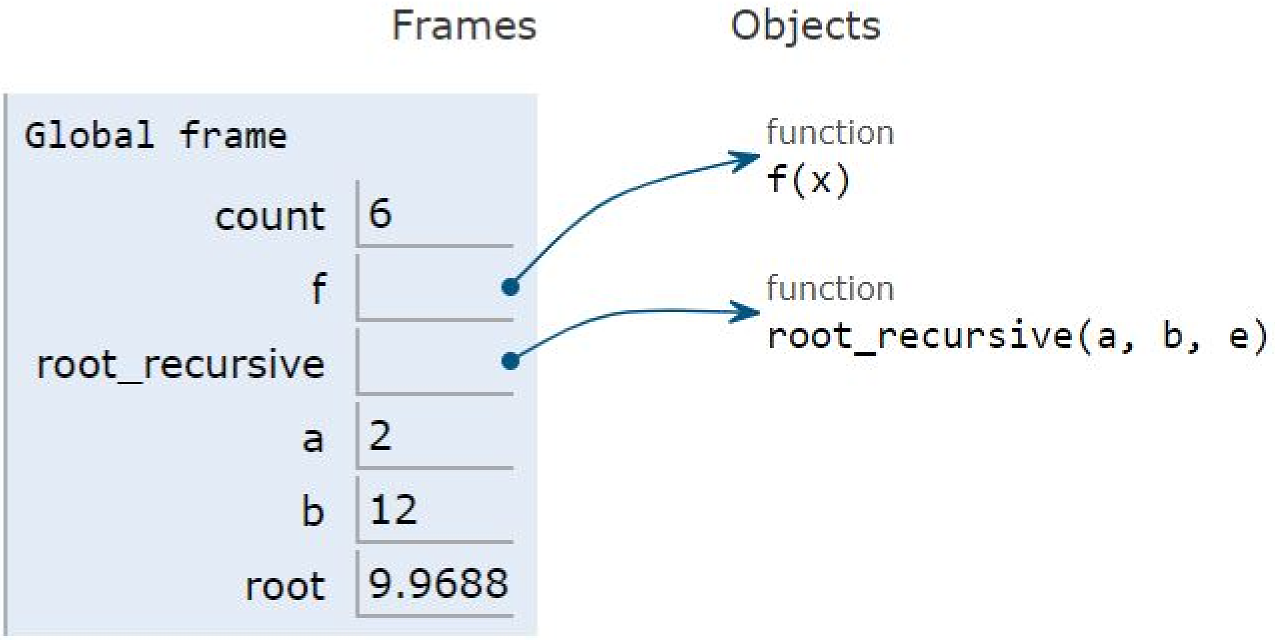
Recursive Algorithm

```
def f(x):
    return x**2 - 11*x + 10

def root_recursive(a,b, e):
    global count
    count   = count + 1
    m       = (a + b)/2.0
    m_value = f(m)
    print(’count=’, count, ’a=’, a, ’b=’, b,
           ’m=’, m, ’f(m)=’, round(m_value,2))
    if abs(m_value) < e:
        return m
    else:
        a_value = f(a)
        if a_value * m_value < 0:
            return root_recursive(a, m, e)
        else:
            return root_recursive(m, b, e)

a = 2; b = 12; count = 100
root = root_recursive(2, 12, e=0.5)
```

Recursive Root Computation



```
count= 1 a= 2 b= 12 m= 7.0 f(m)= -18.0
count= 2 a= 7.0 b= 12 m= 9.5 f(m)= -4.25
count= 3 a= 9.5 b= 12 m= 10.75 f(m)= 7.31
count= 4 a= 9.5 b= 10.75 m= 10.125 f(m)= 1.14
count= 5 a= 9.5 b= 10.125 m= 9.8125 f(m)= -1.65
count= 6 a= 9.8125 b= 10.125 m= 9.96875 f(m)= -0.28
```



Frames Objects

Global frame

count 6

f

root_recursive

a 2

b 12

root 9.9688

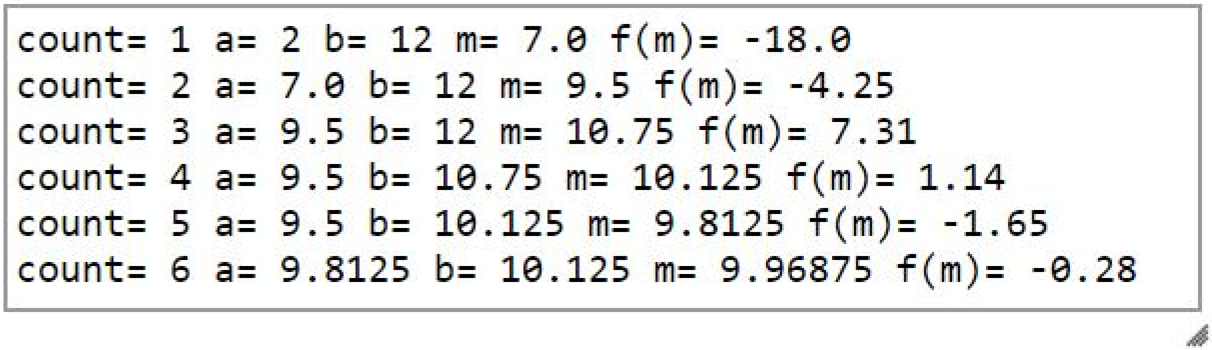
function
f(x)

function
root_recursive(a, b, e)

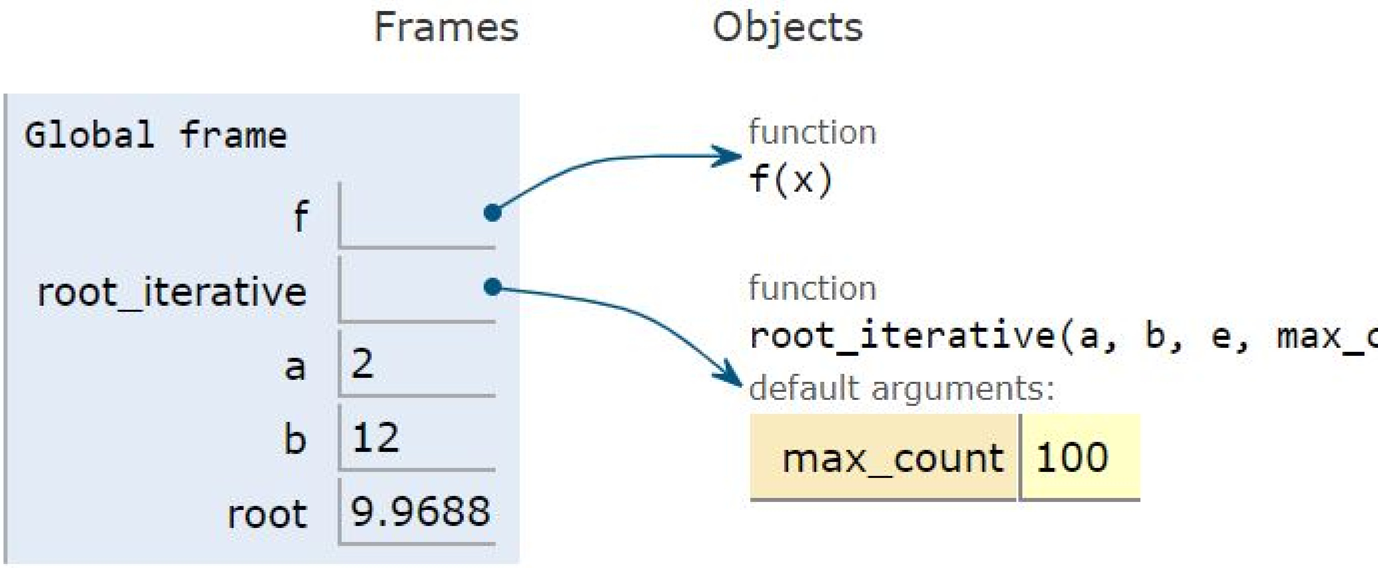
Iterative Method

```
def f(x):
    return x**2 - 11*x + 10

def root_iterative(a,b, e, max_count = 100):
    count   = 0;
    while count < max_count:
        count = count + 1
        m = (a + b)/2.0; m_value = f(m)
        print(’count=’, count, ’a=’, a, ’b=’, b,
           ’m=’, m, ’f(m)=’, round(m_value,2))
        if abs(m_value) < e:
            return m
        else:
            a_value = f(a)
            if a_value * m_value < 0:
                b = m
            else:
                a = m
    else:  # can use else with while in Python
        print(’no root after’, max_count, ’steps’

a = 2; b = 12;
root = root_iterative(2, 12, e=0.5)
```

Iterative Root Computation



```
count= 1 a= 2 b= 12 m= 7.0 f(m)= -18.0
count= 2 a= 7.0 b= 12 m= 9.5 f(m)= -4.25
count= 3 a= 9.5 b= 12 m= 10.75 f(m)= 7.31
count= 4 a= 9.5 b= 10.75 m= 10.125 f(m)= 1.14
count= 5 a= 9.5 b= 10.125 m= 9.8125 f(m)= -1.65
count= 6 a= 9.8125 b= 10.125 m= 9.96875 f(m)= -0.28
```


Frames
Objects
Global frame
f
root_iterative
a
2
b
12
root
9.9688
function
f(x)
function
root_iterative(a, b, e, max_c
default arguments:
max_count
100