Module 3

This is a single, concatenated file, suitable for printing or saving as a PDF for offline viewing. Please note that some animations or images may not work.

Module 3 Study Guide and Deliverables

Theme:	Building Python Projects
Readings:	• Chapter 2 (sections 2.2.8–2.2.14), Chapter 4, Chapter 6 (sections 6.1–6.5), Chapter 14 (sections 14.1–14.3) • Module Lecture Notes
Topics:	Strings, Collections, Control Flow, Iterations, Files, Lists
Assignments:	• Assignment 3 due on Tuesday, July 23 at 6:00 pm ET • Final Project Topic due on Wednesday, July 24 at 6:00 pm ET
Assessments:	Quiz 3: • Available Friday, July 19 at 6:00 am ET • Due on Tuesday, July 23 at 6:00 pm ET
Live Classroom:	• Wednesday, July 17 from 7:30-9:00 pm ET • Thursday, July 18 from 7:30-9:00 pm ET • Facilitator Session: To be determined

Learning Objectives

After successfully completing this module, the learner is expected to do the following:

- Manipulate text with strings and string functions.
- Use collections in programing.
- Use control flow.

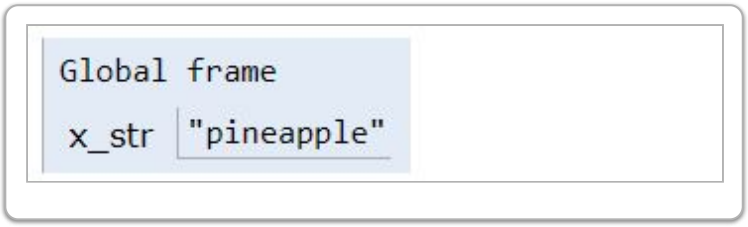
- Perform iterations.
- Open, access, read, and store files.
- Apply list in different applications.

Strings and Text Manipulation

Strings - Python String Overview

- A Python string is an object, not just an array of character data.
- A string is ordered and immutable.
- There are many built-in methods in Python to manipulate strings.

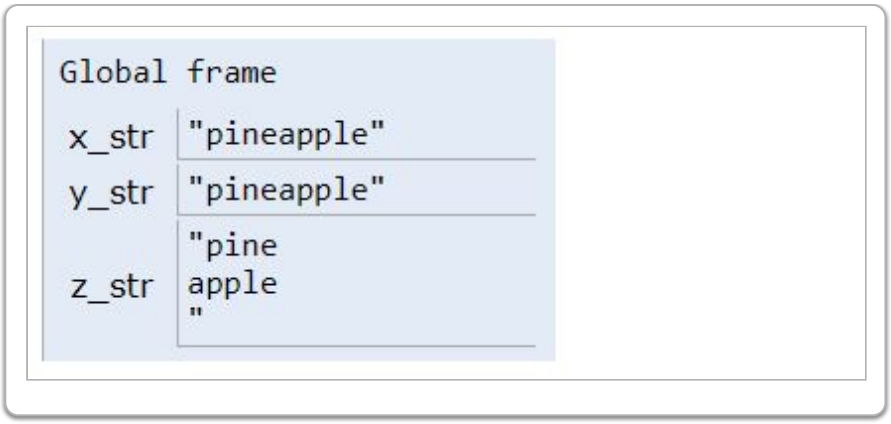
```
x_str = 'pineapple'
```



0	1	2	3	4	5	6	7	8
p	i	n	e	a	p	p	l	e

Defining Strings

```
x_str = 'pineapple' # single quote
y_str = "pineapple" # double quote
# triple quotes allow multi-line strings
z_str = """ pine
apple
"""
```



Test Yourself Exercises

Test Yourself 3.1.01

Write programs to show three ways to define the following (old English proverb) string *x_str*:

"after
meat
comes
mustard"

Suggested program:

First way:

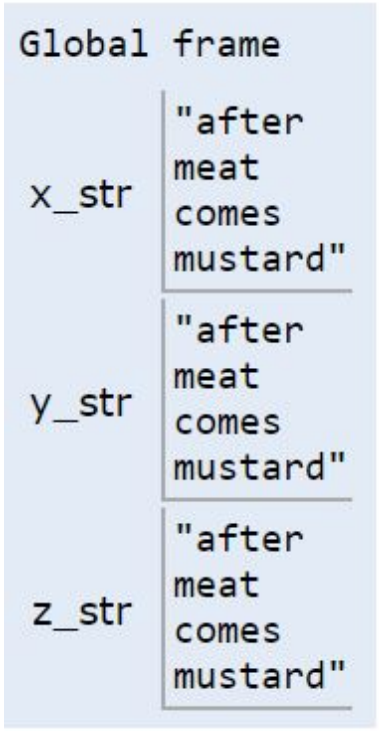
```
x_str =""" after
meat
comes
mustard """
```

Second way:

```
y_str = "after" + "\n" + "meat" + "\n" + \
           "comes" + "\n" + "mustard"
```

Third way:

```
z_str =""" after
meat """ + """
comes
mustard """
```



Test Yourself 3.1.02

How may newline characters are there in *x_str*?

Suggested answer: three newline characters.

String Encoding

Every character is mapped to an integer.

- Past: ASCII code for each charcater
- Now: UTF variable length encoding
    a. international alphabets
    b. memory efficiency

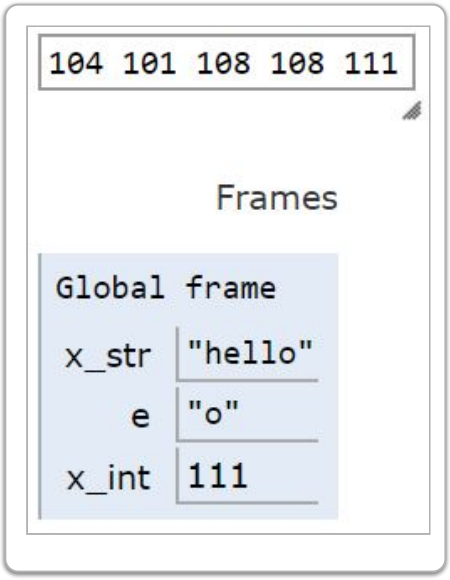
ord() and *chr()* are used for forward and reverse mapping.

ord() Function

ord(): maps character to its integer "value".

```
# print integer values
# for each character
x_str = 'hello'
```

```
for e in x_str:
    x_int = ord(e)
    print(x_int, end = " ")
```



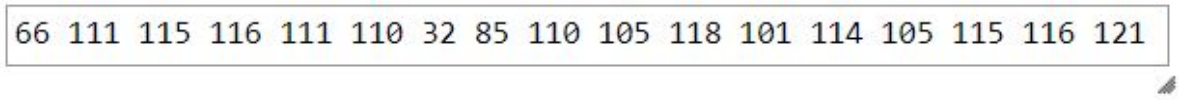
Test Yourself 3.1.03

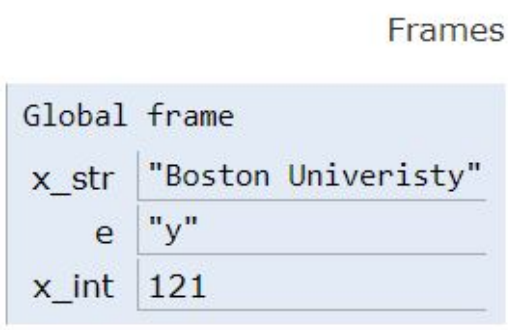
Write a program: use *ord()* to print integer values for each character in string *x_str*:

x_str = "Boston University"

Suggested program:

```
x_str = "Boston Univeristy"
for e in x_str:
    x_int = ord(e)
    print(x_int , end = " ")
```

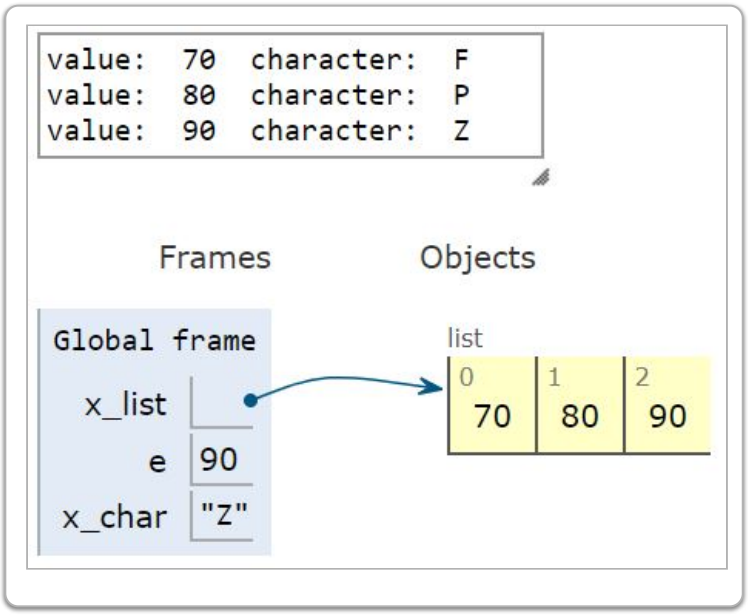




chr() Function

chr(): maps integer value to corresponding character.

```
x_list = [70, 80, 90]
for e in x_list:
    x_char = chr(e)
    print('value: ', e, 'character: ', x_char)
```



Test Yourself 3.1.04

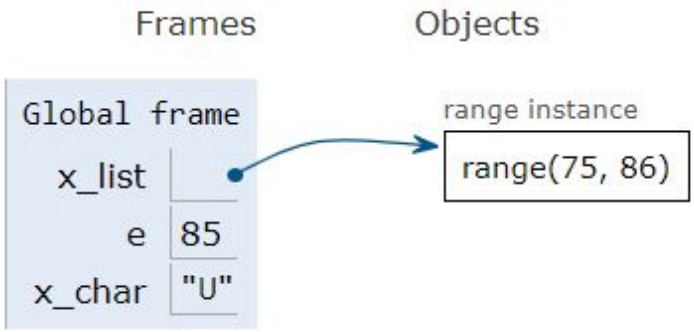
Write a program: use *chr()* to print characters for integers from 75 to 85.

x_str = "Boston University"

Suggested program:

```
x_list = range(75, 86) # note 86, not 85
for e in x_list:
    x_char = chr(e)
    print('value: ', e, 'character: ', x_char)
```

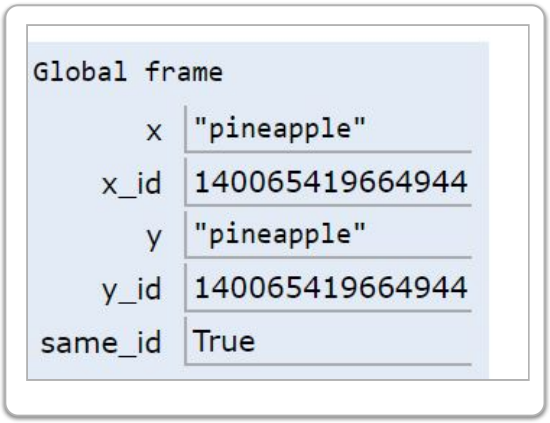
```
value:  75  character:  K
value:  76  character:  L
value:  77  character:  M
value:  78  character:  N
value:  79  character:  O
value:  80  character:  P
value:  81  character:  Q
value:  82  character:  R
value:  83  character:  S
value:  84  character:  T
value:  85  character:  U
```



String Immutability

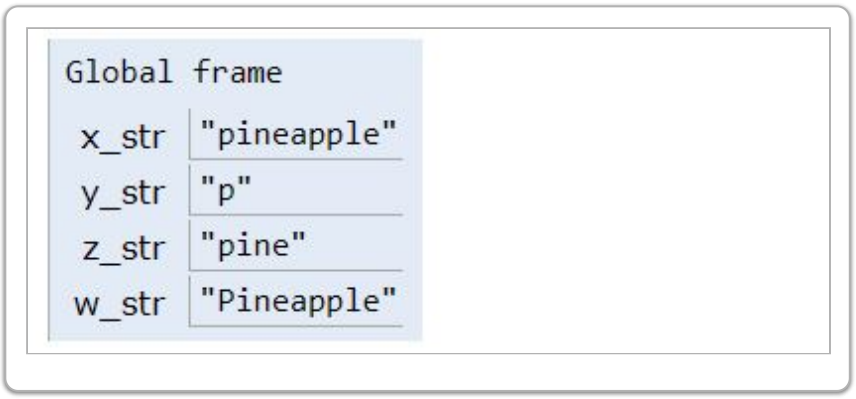
Python strings are immutable.

```
x = "pineapple"
x_id = id(x)
y = 'pine' + 'apple'
y_id = id(y)
same_id = (x_id == y_id)
```



Examples of String Methods

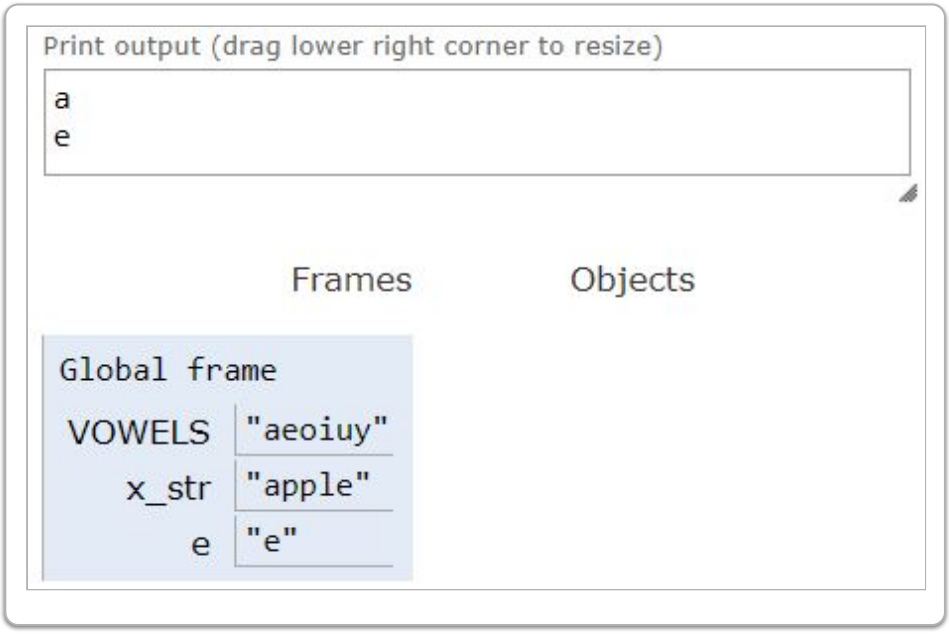
```
x_str = 'pineapple'
y_str = x_str[6]      # indexing
z_str = x_str[0 : 4]  # slicing
w_str = x_str.title() # capitalize first
```



0	1	2	3	4	5	6	7	8
p	i	n	e	a	p	p	l	e

Membership & Iteration

```
# print vowels in a string
VOWELS = 'aeoiuy'
x_str = 'apple'

for e in x_str:
    if e in VOWELS:
        print(e)
```



Test Yourself 3.1.05

Write a program: print all consonants in string *x_str*:

"after

meat

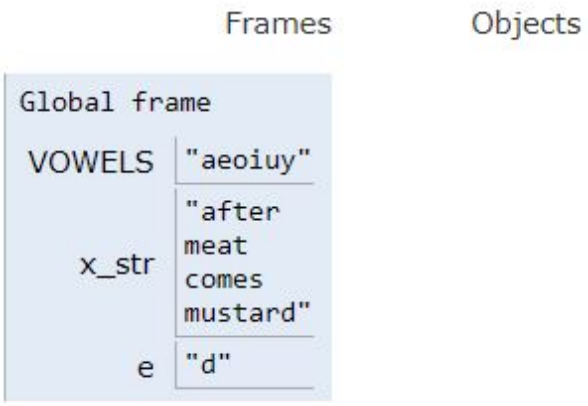
comes

mustard"

Suggested program:

```
# print consonants in a string
VOWELS = 'aeoiuy'
x_str = """ after
meat
comes
mustard """

for e in x_str:
    if e not in VOWELS and e!= '\n':
    print(e, end = " ")
```

```
f t r m t c m s m s t r d
```

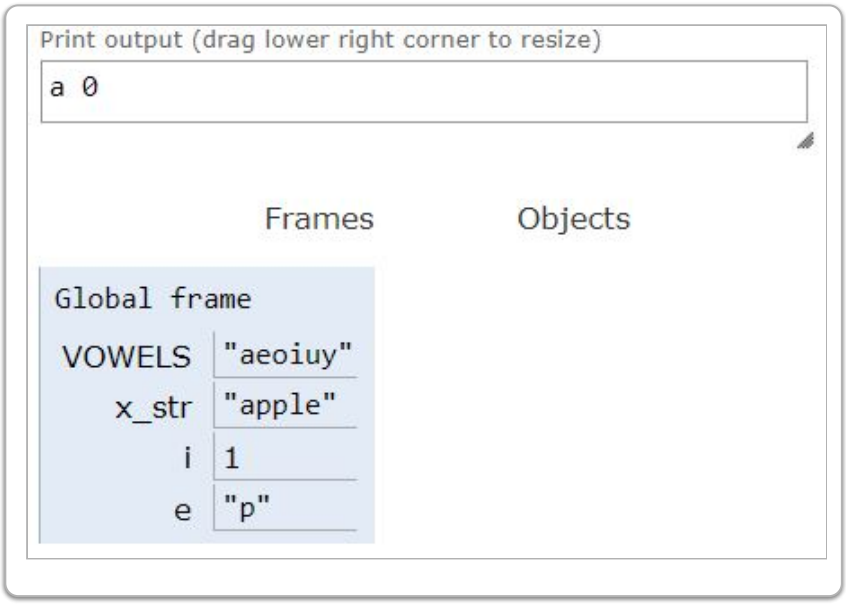


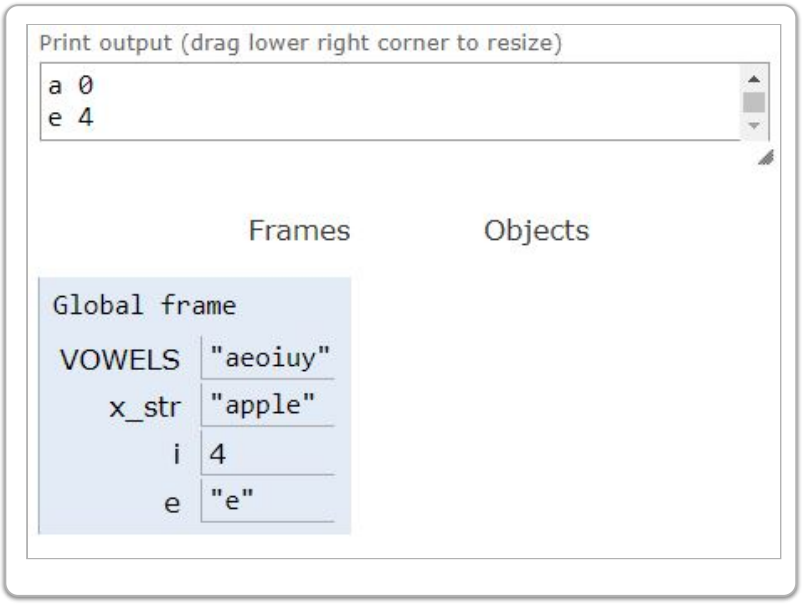
Iteration: *enumerate()*

- Using *enumerate()*, can get both index and element.
- Can use *enumerate()* in strings, lists, tuples.

```
# print vowels and positions from string
VOWELS = 'aeoiuy'
x_str = 'apple'

for i,e in enumerate(x_str):
```

```
e = x_str[i]
if e in VOWELS:
    print(e,i)
```





Test Yourself 3.1.06

Write a program: print all consonants and positions in string *x_str*:

"after
meat
comes
mustard"

Suggested program:

```
VOWELS = 'aeoiuy'
x_str = """ after
meat
comes
mustard """

for i, e in enumerate (x_str):
    if e not in VOWELS and e!='\n':
    print(i,e)
```

```
1 f
2 t
4 r
6 m
9 t
11 c
13 m
15 s
17 m
19 s
20 t
22 r
23 d
```

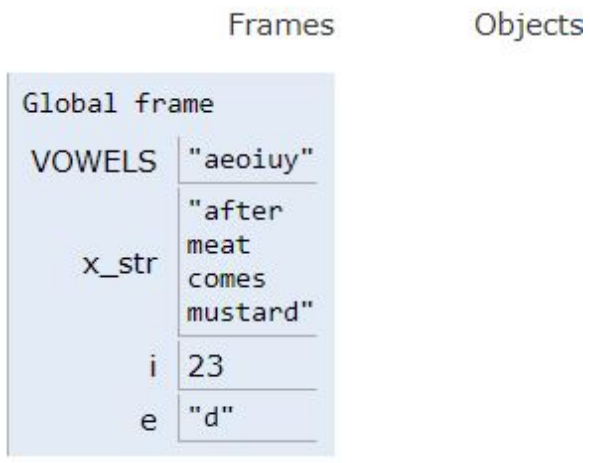
Test Yourself 3.1.07

Write a program: print vowels and positions in string x str without using *enumerate()*:

Suggested program:

```
VOWELS = 'aeoiuy'
x_str = """ after
meat
comes
mustard """

for i in range(len(x_str)):
    e = x_str[i]
    if e not in VOWELS and e!='\n':
        print(i,e)
```

```
1 f
2 t
4 r
6 m
9 t
11 c
13 m
15 s
17 m
19 s
20 t
22 r
23 d
```



Strings Functions

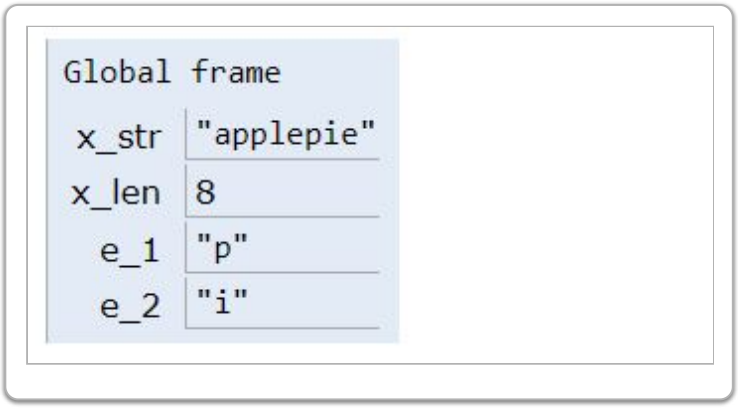
String Indexing

Indexing allows you to access individual characters in a string directly by using a numeric value. There are positive and negative indices.

- Positive index: String indexing is zero-based: the first character in the string has index 0, the next is 1, and so on.
- Negative index: As an alternative, Python uses **negative** numbers to index into a string: -1 means the last character, -2 means the next to last, and so on.
- Positive = negative + length

```
x_str = 'applepie'

# len(): returns the length of a string
x_len = len(x_str)
```

```
e_1   = x_str[1]
e_2   = x_str[-2]
```



0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
-8	-7	-6	-5	-4	-3	-2	-1

Test Yourself 3.1.08

Write a program: use positive and negative indices to extract "7" from *x_str*:

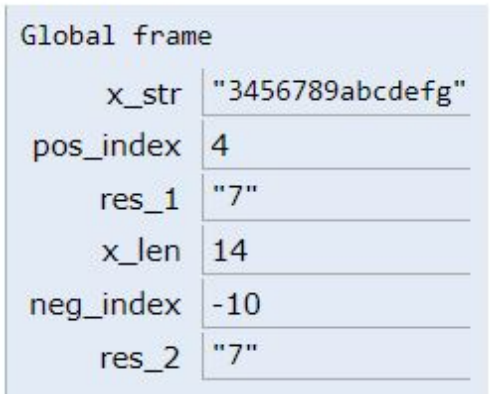
x_str = "3456789abcdefg"

Suggested program: number 7 is at index 4.

```
x_str = "3456789abcdefg"
pos_index = 4
res_1 = x_str[pos_index]

x_len = len(x_str)
neg_index = pos_index - x_len
res_2 = x_str[neg_index]

print('positive index:', pos_index)
print('negative index: ', neg_index)
```

```
positive index: 4
negative index:  -10
```

Frames



Test Yourself 3.1.09

Write a program: print positive and negative indices for even digits in *x_str*:

Suggested program:

```
x_str = "3456789abcdefg"
x_len = len(x_str)
for pos_index in range(x_len):
    e = x_str[pos_index]
    if e in "02468":
    neg_index = pos_index - x_len
    print(e, 'positive index:', pos_index)
    print(e, 'negative index: ', neg_index)
```

```
4 positive index: 1
4 negative index:  -13
6 positive index: 3
6 negative index:  -11
8 positive index: 5
8 negative index:  -9
```

Frames

Global frame

x_str	"3456789abcdefg"
x_len	14
pos_index	13
e	"g"
neg_index	-9

String Slicing

Slicing allows you to extract substrings from a string, by identifying a range of index numbers [start : end + 1 : step]:

- The first index number is where the slice starts (inclusive).
- The second index number is where the slice ends (exclusive).
- Step number: the number of characters to skip between two index numbers of a slice.
  - If the step number is 1, will take every character between two index numbers of a slice.
  - If the step number is 2, skip every other character between two index numbers.
  - If the step number is omitted, Python will default with 1.

We can use both positive and negative indices. Negative step is for reversals.

Use the string "applepie" as an example.

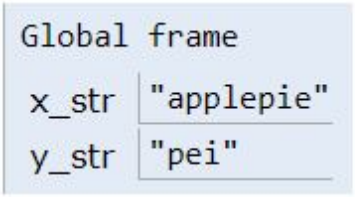
```
x_str = 'applepie'
```

Global frame

x_str	"applepie"

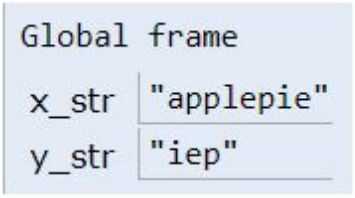
0	1	2	3	4	5	6	7
a	**p**	**p**	**l**	**e**	**p**	**i**	**e**
-8	-7	-6	-5	-4	-3	-2	-1

```
x_str = 'applepie'

y_str = x_str[ 2 :  7 : 2]
y_str = x_str[-6 : -1 : 2]
y_str = x_str[ 2 : -1 : 2]
y_str = x_str[-6 :  7 : 2]
```



0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
-8	-7	-6	-5	-4	-3	-2	-1

```
x_str = 'applepie'

y_str = x_str[ 6 :  1 : -2]
y_str = x_str[-2 : -7 : -2]
y_str = x_str[ 6 : -7 : -2]
y_str = x_str[-2 :  1 : -2]
```

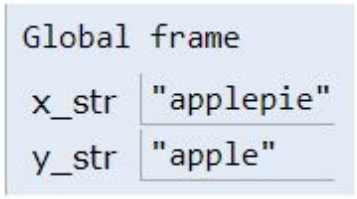


0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
-8	-7	-6	-5	-4	-3	-2	-1

Slicing with Default

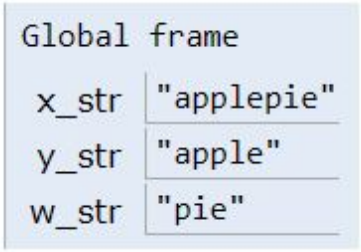
- when the first index number is omitted, Python will default with 0, the beginning of the string.
- When the the index number after the colon is omitted, Python will default to the last index, the end of the string.
- If the step number is omitted, Python will default with 1.

```
x_str = 'applepie'

y_str = x_str[0 : 5 : 1]
y_str = x_str[  : 5 : 1]  # assume defaults
y_str = x_str[  : 5]
```



0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
-8	-7	-6	-5	-4	-3	-2	-1

```
x_str = 'applepie'

y_str = x_str[  : 5]
w_str = x_str[5 :  ]
```



0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
-8	-7	-6	-5	-4	-3	-2	-1

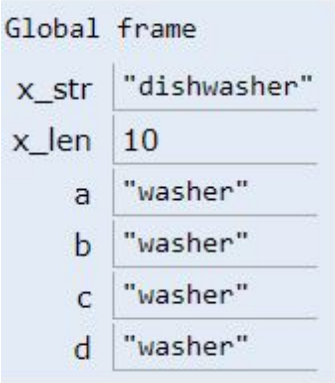
Test Yourself 3.1.10

Write a program: show four different ways to extract "wash" from *x_str*:

x_str = "dishwasher"

Suggested program:

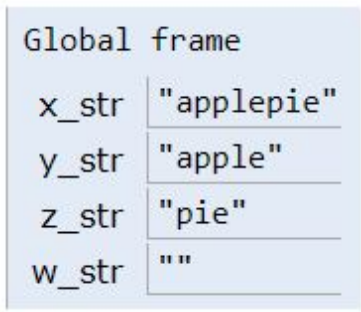
```
x_str = "dishwasher"
x_len = len(x_str)
a = x_str[4  : 10]
b = x_str[4  :   ]
c = x_str[-6 : 10]
d = x_str[-6 : ]
```



"Out-of-Bound" Slicing

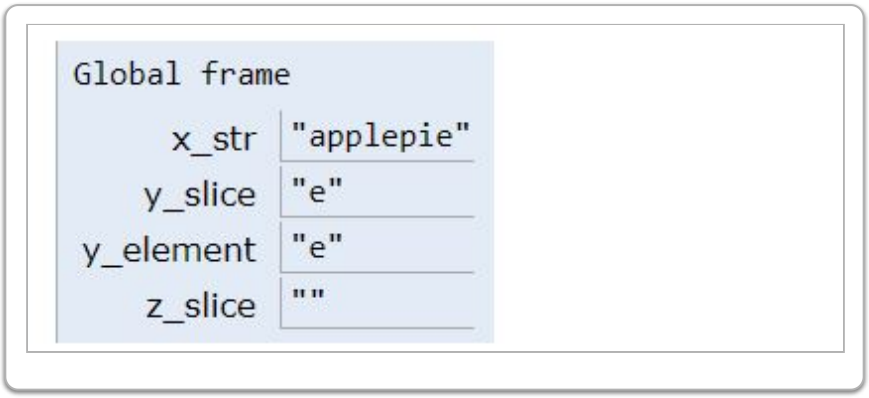
Python string slicing handles out of range indexes gracefully—get the "largest" substring and do not produce error.

```
x_str = 'applepie'
y_str = x_str[-100 : 5]
z_str = x_str[5 : 500 ]
w_str = x_str[400 : 500]
```



Slicing vs. Indexing

```
x_str     = 'applepie'
y_slice   = x_str[4:5]
y_element = x_str[4]
z_slice   = x_str[100:101]
z_element = x_str[100]     # error
```



0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
-8	-7	-6	-5	-4	-3	-2	-1

Test Yourself 3.1.11

Write a program: what is the result of the following slices from *x_str*:

"two plus two is four"

a. *x_str*[10]

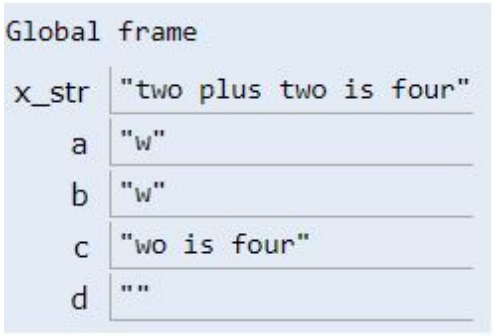
b. *x_str*[10 : 11]

c. *x_str*[10 : 2000]

d. *x_str*[2000 : 2001]

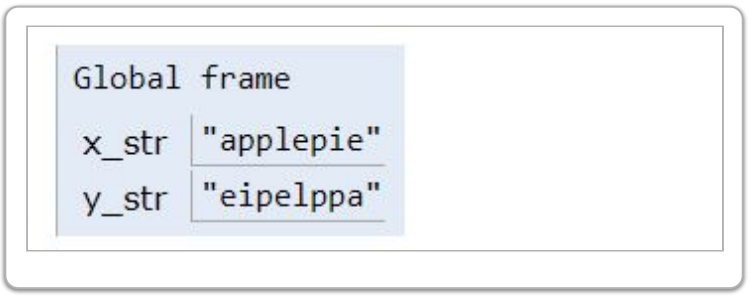
Suggested program:

```
x_str = "two plus two is four"

a = x_str[10]
b = x_str[10 : 11]
c = x_str[10 : 2000]
d = x_str[2000 : 2001]
```



String Reversal

```
x_str = 'applepie'
y_str = x_str[ : : -1]
```



Apply string reversal to check if a string is a palindrome.

```
x_str = 'never odd or even'
y_str = x_str.replace (' ', '')
if y_str == y_str[ : : -1]:
    print(x_str, ' is a palindrome')
else :
    print(x_str, ' is not a palindrome')
```

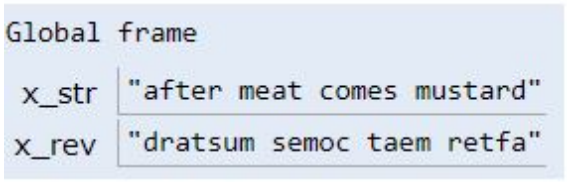
Test Yourself 3.1.12

Write a program: reverse the string *x_str*:

"after meat comes mustard"

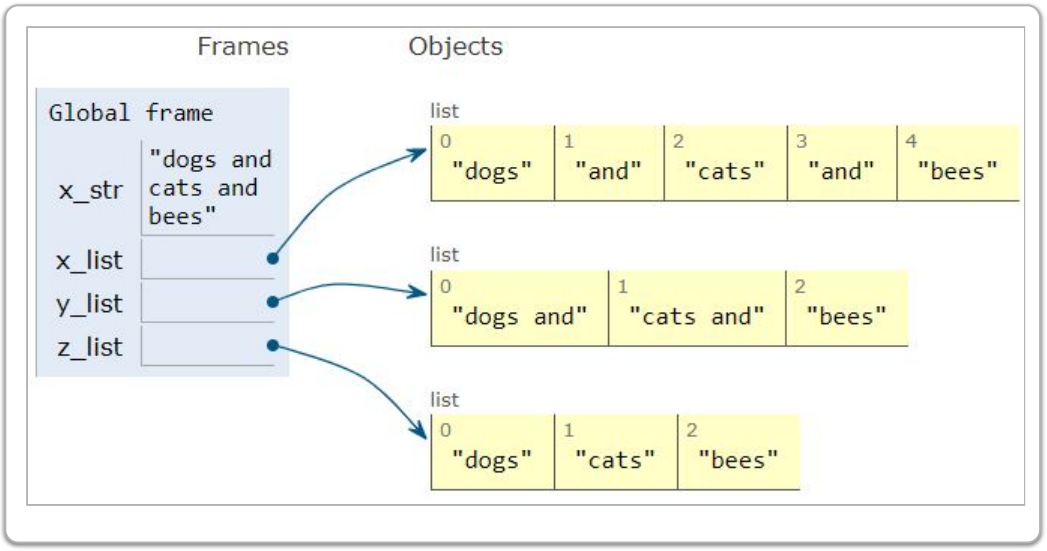
Suggested program:

```
x_str = "after meat comes mustard"
x_rev = x_str [ : : -1 ]
```



String *split()* Function

```
# split string using a separator
x_str = """ dogs and
cats and
bees """

x_list = x_str.split()
y_list = x_str.split(sep = '\n')
z_list = x_str.split(' and \n')
```



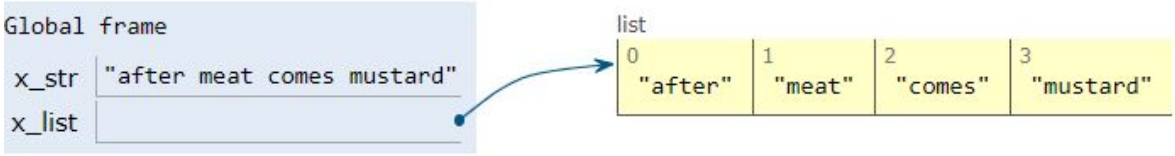
Test Yourself 3.1.13

Write a program: convert a string of words *x_str* into a list of words *x_list*:

"after meat comes mustard"

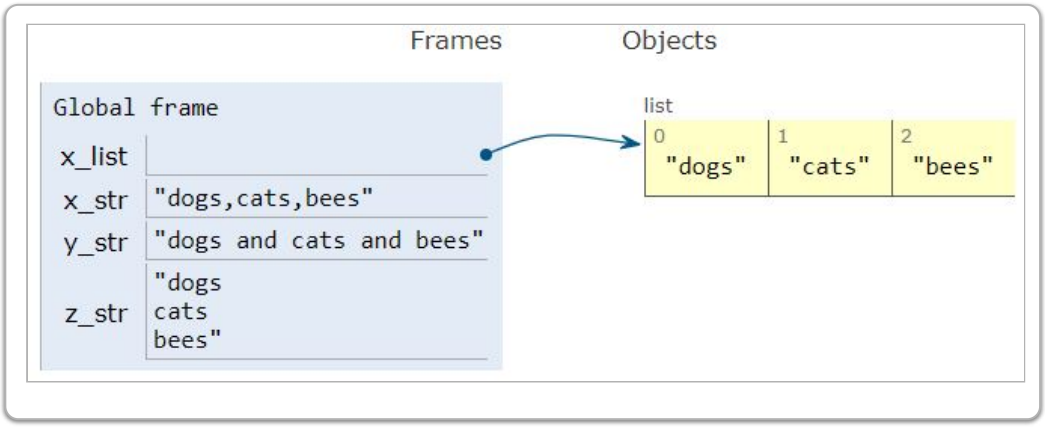
Suggested program:

```
x_str  = "after meat comes mustard"
x_list = x_str.split()
```



String *join()* Function

```
# join strings in list with separator
x_list = ['dogs', 'cats', 'bees']
x_str = ','.join(x_list)
y_str = ' and '.join(x_list)
z_str = '\n'.join(x_list)
```



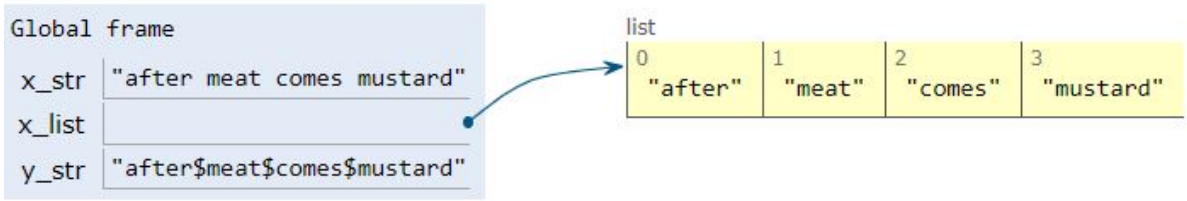
Test Yourself 3.1.14

Write a program: using *split()* and *join()* replace spaces with '$' in the string *x_list*:

"after meat comes mustard"

Suggested program:

```
x_str  = "after meat comes mustard"
x_list = x_str.split()
y_str  = "$".join(x_list)
```

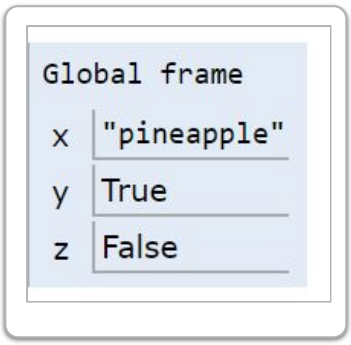


Strings Methods

- There are many string methods (around 50) available.
- This makes Python very useful to use for text processing.

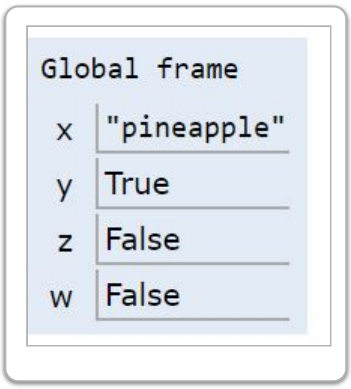
```
x = 'pineapple'
y = x.startswith("pi")
```

```
z = x.endswith("LE")
```



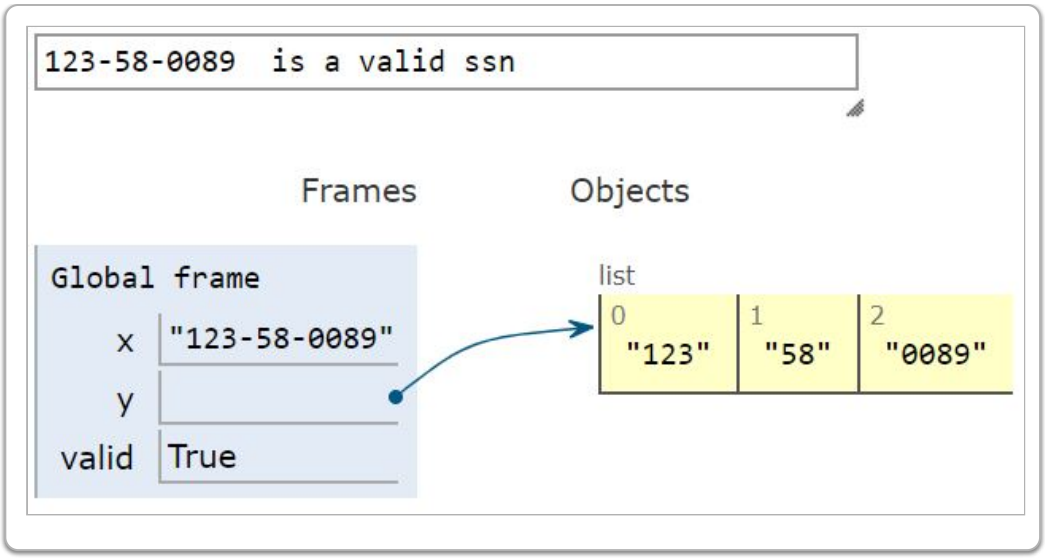
There are many methods to check formats.

```
x = 'pineapple'
y = x.islower()
z = x.isupper()
w = x.isdigit()
```



The isdigit() method in the following program verifies format for social security numbers.

```
x = "123-58-0089";
y = x.split("-")
valid = False
if len(y)==3:
   if (y[0].isdigit() is True) and \
      (y[1].isdigit() is True) and \
      (y[2].isdigit() is True):
          valid = True
if valid is True:
    print(x, ' is a valid ssn')
else:
    print(x, ' is not valid ssn')
```



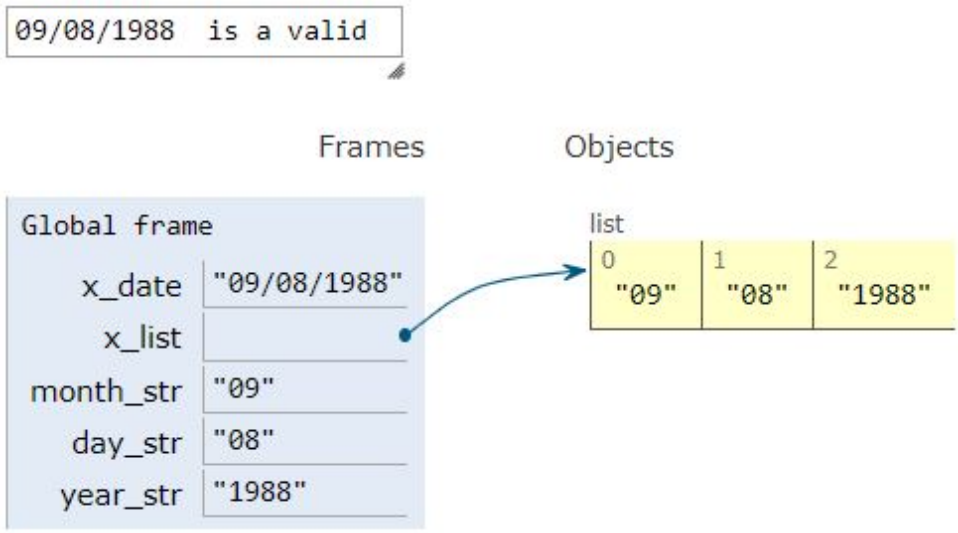
Test Yourself 3.1.15

Write a program: verify that only numeric values are entered for a date.

x_date = "09/08/1988"

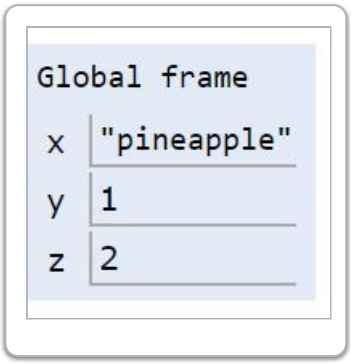
Suggested program:

```
# assume three numbers are entered
x_date = "09/08/1988"
x_list = x_date.split("/")

month_str = x_list[0]
day_str   = x_list[1]
year_str  = x_list[2]

if month_str.isdigit() is True and \
   day_str.isdigit() is True and \
   year_str.isdigit() is True:
    print(x_date, ' is a valid')
else:
    print(x_date , ' is invalid')
```



The count() method will do easy frequency counting.

```
x = 'pineapple'
y = x.count("apple")
z = x.count("e")
```



Test Yourself 3.1.16

Consider string *x_str*: "after meat comes mustard"

a. count the number of times character "*m*" appears.

b. compute position of the first character "*m*".

c. compute position of the second character "*m*".

d. replace "*mustard*" with "*dessert*".

Suggested program:

```
x_str = "after meat comes mustard"
x_count = x_str.count("m")

first_m = x_str.index("m")

if first_m >= 0:
    y_str = x_str[first_m + 1 : ]
    second_m = y_str.index("m")
```

```
            if second_m >=0:
                second_m = (first_m + 1) + second_m

    y_str = x_str.replace("mustard", "dessert")
```

Global frame

x_str	"after meat comes mustard"
x_count	3
first_m	6
y_str	"after meat comes dessert"
second_m	13

Collections

Python Data Types

Python types are the building blocks in a language, similar to noun, verb in English.

Python has two groups of types:

1. primitive types ("atoms")
2. collections ("molecules")

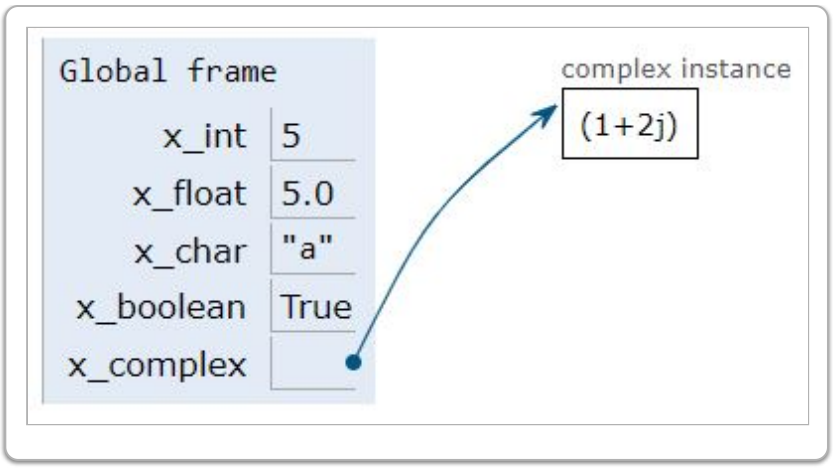
There are additional special types:

1. *None* type
2. *range* type

Primitive Types

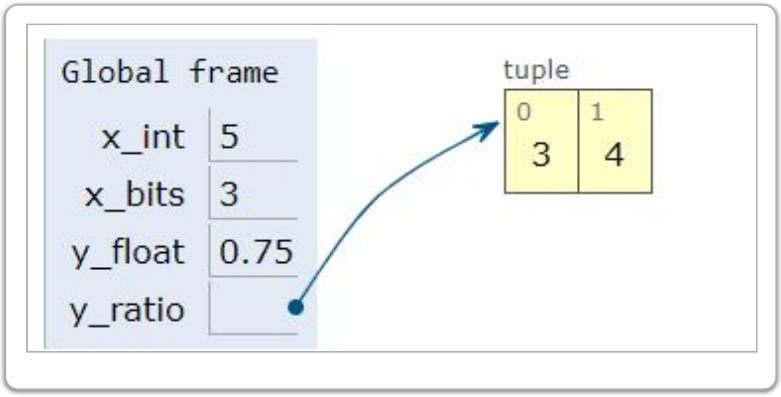
```
x_int     = 5
x_float   = 5.0
x_char    = 'a'
x_boolean = True
x_complex = 1 + 2j
```

Primitive data types also called 'atomic' data types – they are indivisible objects.



Primitive Type Method Examples

```
x_int   = 5
x_bits  = x_int.bit_length()

y_float = 0.75
y_ratio = y_float.as_integer_ratio()
```

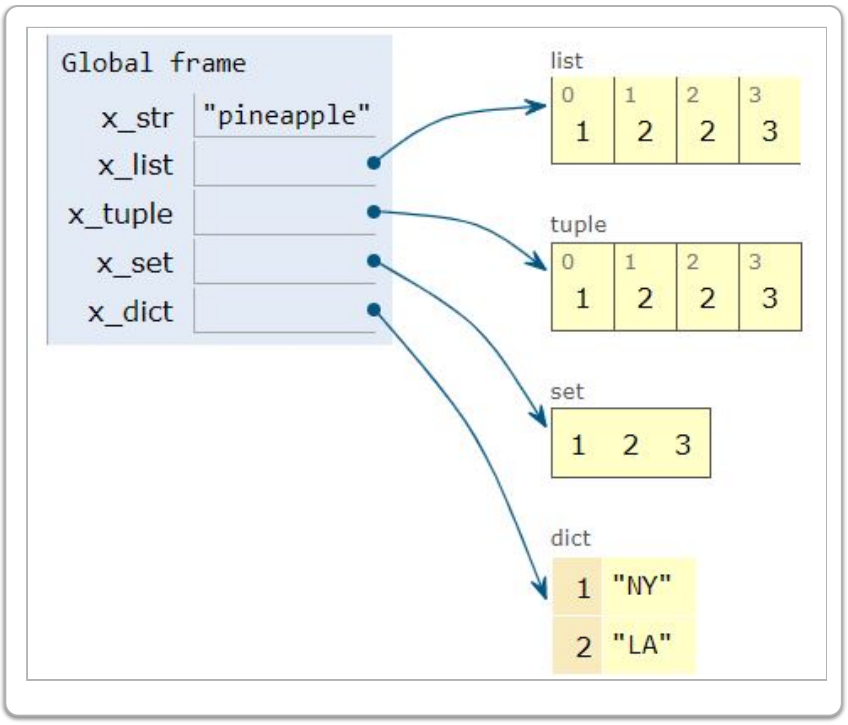


- 'atoms' are not just values
- objects with methods

Collection Types

```
x_str    = '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'}
```

Collection data types 'molecules' – they are complex objects.



- membership: *in/not in*
- iteration: *for*
- some ordered and/or mutable

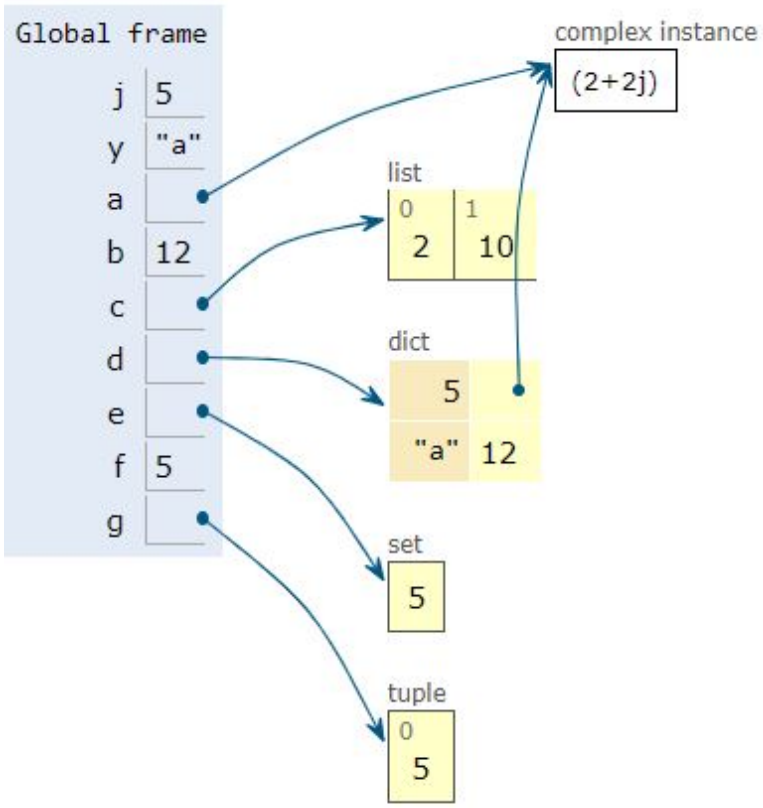
Test Yourself 3.2.01

For each line, indicate object type (primitive or collection).

```
j = 5
y = 'a'
a = 2 + 2j
b = 2 + 2*j
c = [2, 2*j]
d = {j : a, y: b}
e = {j}
f = (j)
g = (j, )
```

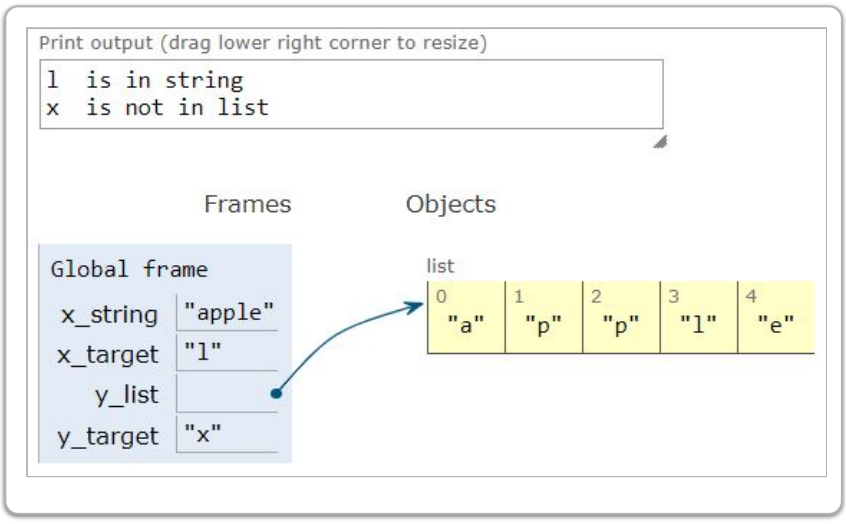
Suggested answer:

```
j = 5             # primitive (integer)
y = 'a'           # primitive (character)
a = 2 + 2j        # primitive (complex number)
b = 2 + 2*j       # primitive
c = [2, 2*j]      # collection (list)
d = {j : a, y: b} # collection (dictionary)
e = {j}           # collection (set)
f = (j)           # primitive (integer)
g = (j, )         # collection (tuple)
```



Membership: *in/not in*

```
x_string = 'apple'
x_target = 'l'
if x_target in x_string:
    print(x_target , ' is in string')

y_list = ['a','p','p','l','e']
y_target = 'x'
if y_target not in y_list:
    print(y_target, ' is not in list')
```



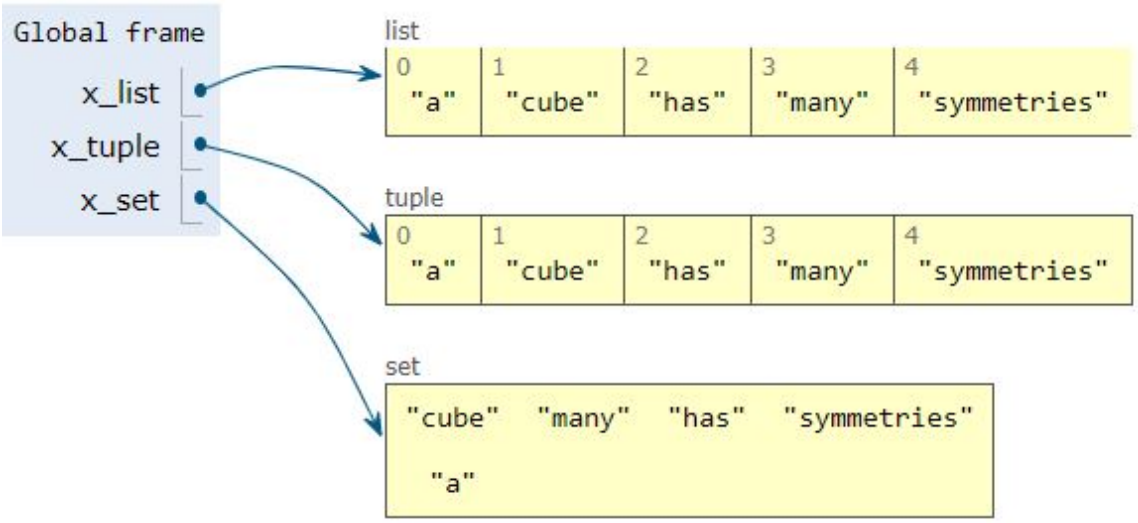
Test Yourself 3.2.02

Construct three different collections containing words from *x_str*:

a cube has many symmetries

Suggested answer:

```
x_list  = ["a","cube","has","many","symmetries"]
x_tuple = ("a","cube","has","many","symmetries")
x_set   = {"a","cube","has","many","symmetries"}
```



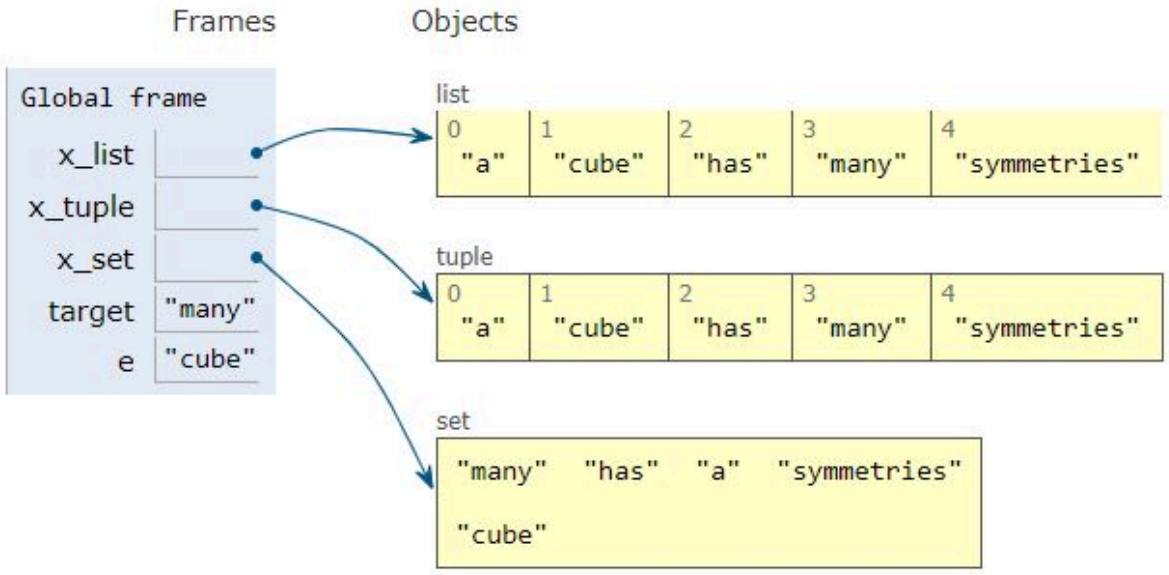
Test Yourself 3.2.03

Based on the last exercise, for each collection use iteration to check if it contains the word "*many*".

Suggested program:

```
x_list  = ["a","cube","has","many","symmetries"]
x_tuple = ("a","cube","has","many","symmetries")
x_set   = {"a","cube","has","many","symmetries"}

target = "many"

for e in x_list:
    if e == target:
        print(target, ' is in list')

for e in x_tuple :
    if e == target:
        print(target, ' is in tuple')

for e in x_set:
    if e == target:
        print(target, ' is in set')
```

```
many  is in list
many  is in tuple
many  is in set
```



Test Yourself 3.2.04

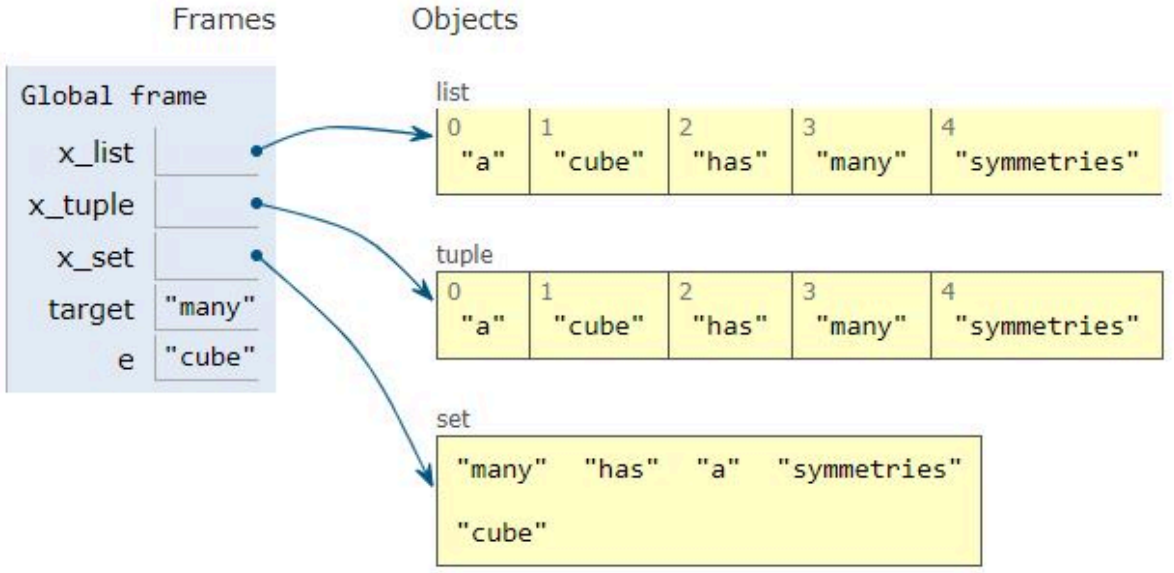
For each collection use in, not in check if it contains the word "*many*".

a cube has many symmetries

Suggested program:

```
x_list  = ["a","cube","has","many","symmetries"]
x_tuple = ("a","cube","has","many","symmetries")
x_set   = {"a","cube","has","many","symmetries"}

target = "many"

if target in x_list:
    print(target, ' is in list')
else:
    print(target, ' is not in list')

if target in x_tuple:
    print(target, ' is in tuple')
else:
    print(target, ' is not in tuple')

if target in x_set:
    print(target, ' is in set')
else:
    print(target, ' is not in set')
```

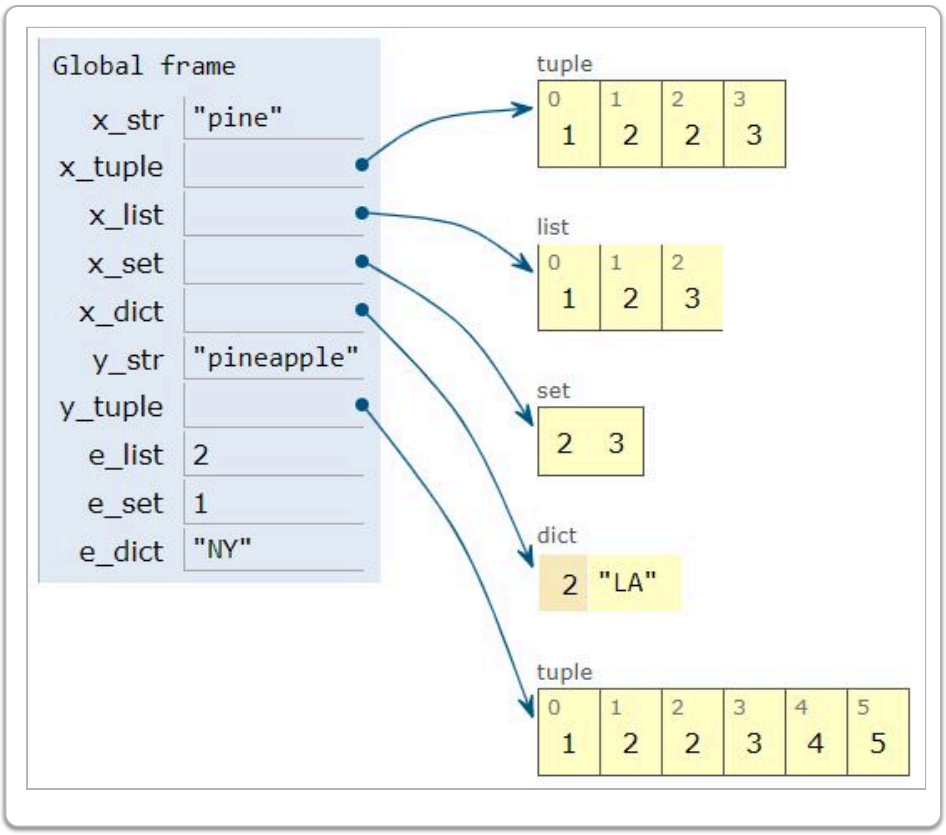
```
many  is in list
many  is in tuple
many  is in set
```



Collection Methods

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

y_string = x_string + 'apple'
y_tuple  = x_tuple  + (4, 5)
e_list   = x_list.pop(1)
e_set    = x_set.pop()
e_dict   = x_dict.pop(1)
```

- '+' is overloaded.
- Polymorphic methods: a same function that can be applied to different types. For example, .pop() method can be applied to a list, a set, or a dictionary data type.

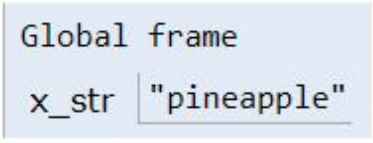


Python String, Tuple, and List

A Python Strings

- A Python string is an object, not just an array of character data.
- A string is ordered and immutable.
- There are many built-in methods in Python to manipulate strings.

```
x_str = 'pineapple'
```

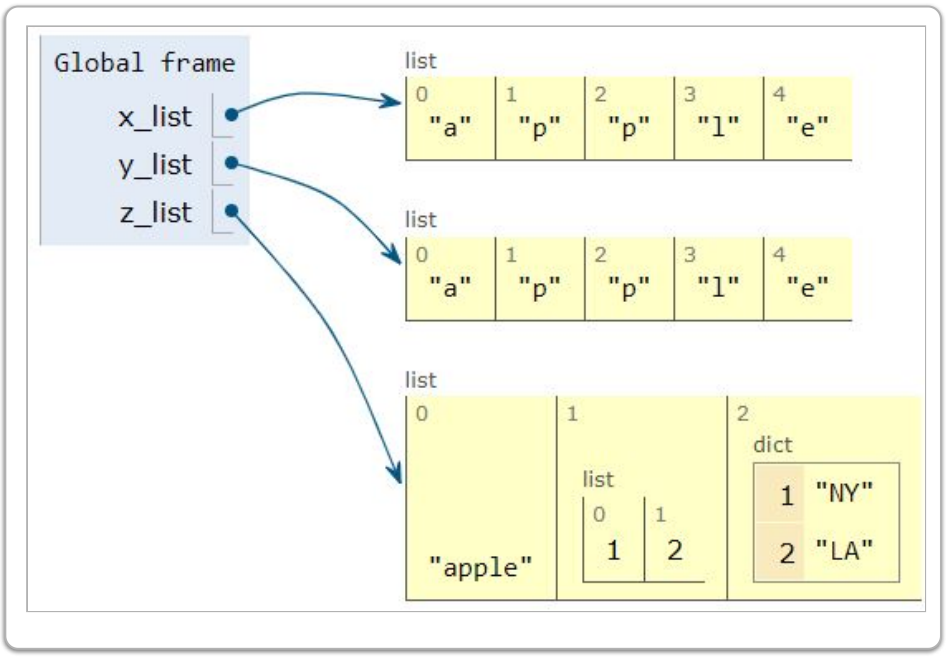


0	1	2	3	4	5	6	7	8
p	i	n	e	a	p	p	l	e

A Python List

- A Python list is an ordered collection.
- A list can contain any objects.

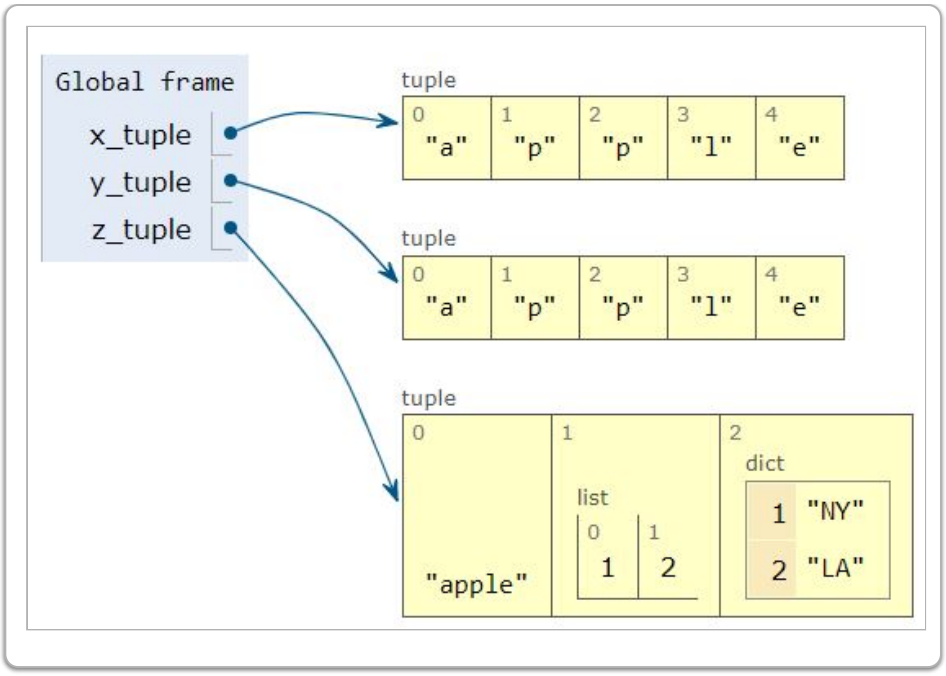
```
x_list = ['a','p','p','l','e']
y_list = list('apple')
z_list = ['apple', [1,2] , {1: 'NY', 2: 'LA'}]
```



A Python Tuple

- A Python tuple is an ordered collection (like list).
- A list can contain any objects.

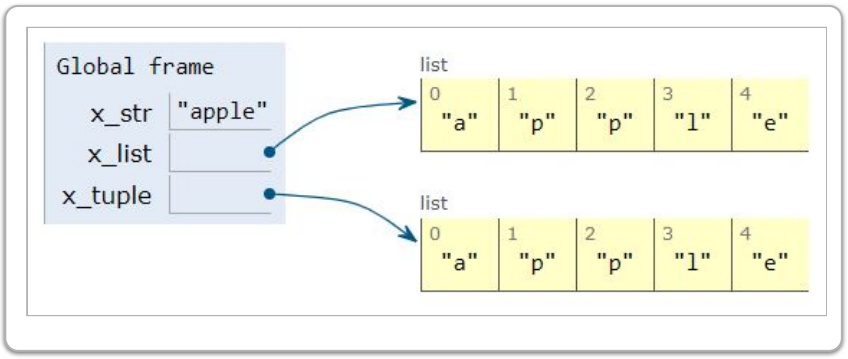
```
x_tuple = ('a','p','p','l','e')
y_tuple = tuple('apple')
z_tuple = ('apple', [1 ,2], {1: 'NY', 2: 'LA'})
```



Strings, Lists, Tuples

- Ordered collections
- Support indexing & slicing

```
x_string = 'apple'
x_list = ['a','p','p','l','e']
x_tuple = ('a','p','p','l','e')
```



0	1	2	3	4
a	p	p	l	e

Test Yourself 3.2.05

Show two different ways to construct *x_list* and *x_tuple* from *x_str*:

x_str = "apple"

Suggested program:

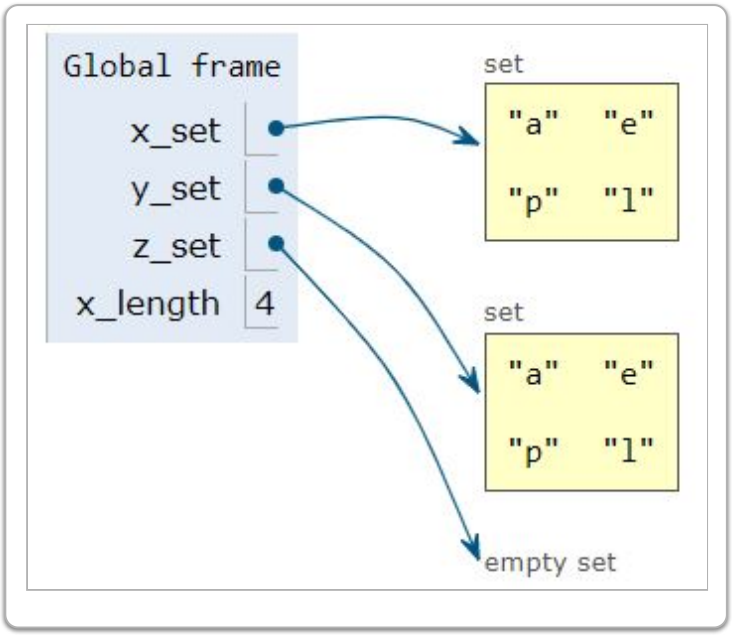
```
x_str = "apple"
x_list_1  = list(x_str)
x_tuple_1 = tuple(x_str)

x_list_2 = list()
for e in x_str:
    x_list_2 = x_list_2 + [e]

x_tuple_2 = tuple()
for e in x_str :
    x_tuple_2 = x_tuple_2 + (e, )
```

Python Set and Dictionary

A Python Set

- A Python set is un-ordered, unique elements.
- There are restrictions on elements.

```
x_set = {'a','p','p','l','e'}
y_set = set('apple')
z_set = set()
```

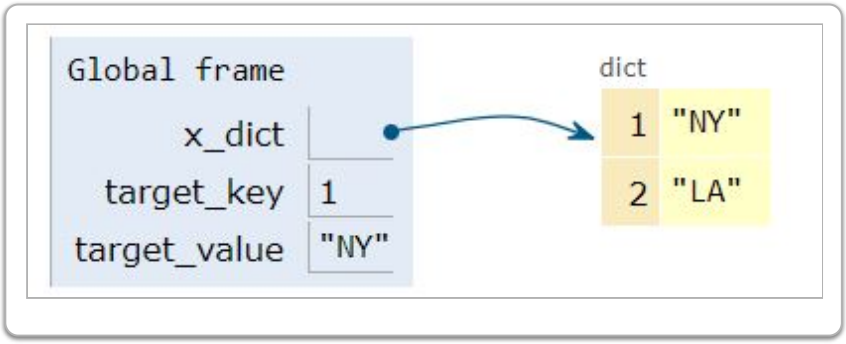


A Python Dictionary

- A Python dictionary is a collection of (key, value) pairs.

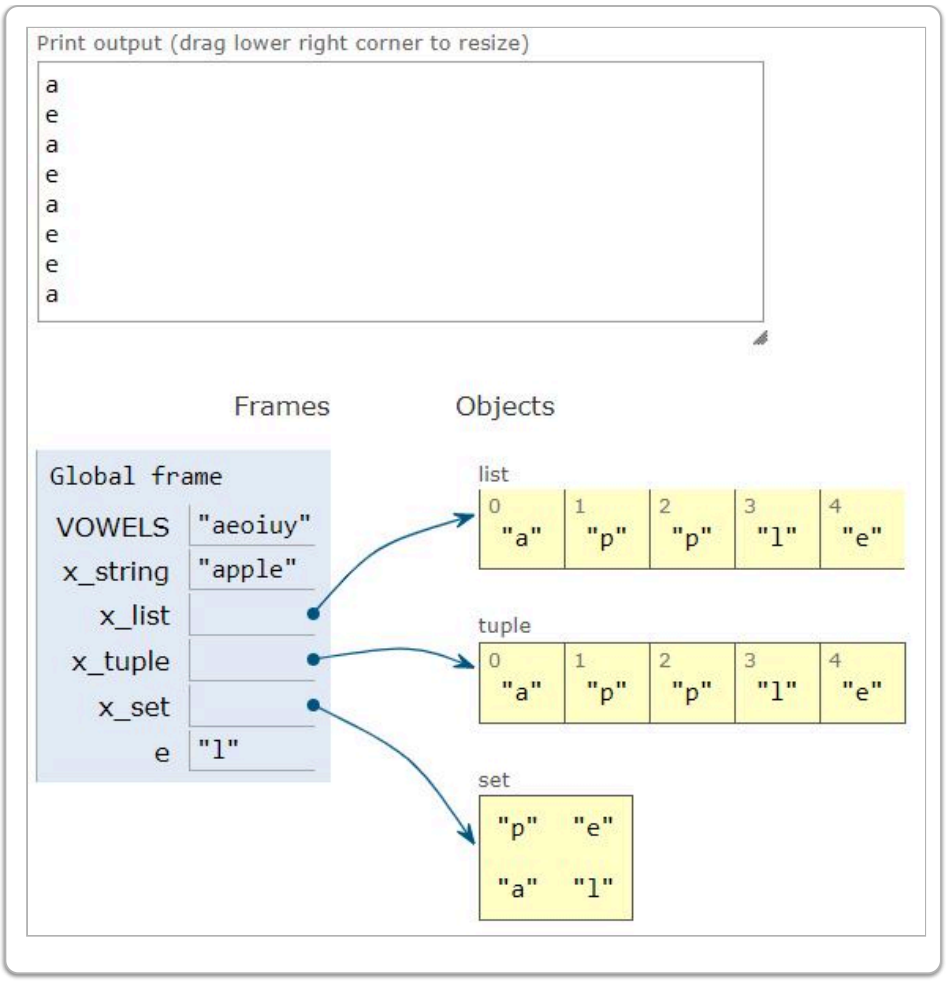
- Such pairs are called items.
- There are built-in functions for keys, values and items.

```
x_dict = {1: 'NY', 2: 'LA'}
target_key = 1
target_value = x_dict[target_key]
```



Iteration in Collections

```
VOWELS = 'aeoiuy'
x_string = 'apple'
x_list   = ['a','p','p','l','e']
x_tuple  = ('a','p','p','l','e')
x_set    = {'a','p','p','l','e'}

for e in x_string:
    if e in VOWELS:
        print(e, end='')

for e in x_list:
    if e in VOWELS:
        print(e, end='')

for e in x_tuple:
    if e in VOWELS:
        print(e, end='')

for e in x_set:
    if e in VOWELS:
        print(e, end='')
```



Test Yourself 3.2.06

Write iterations to print consonants from the following collections:

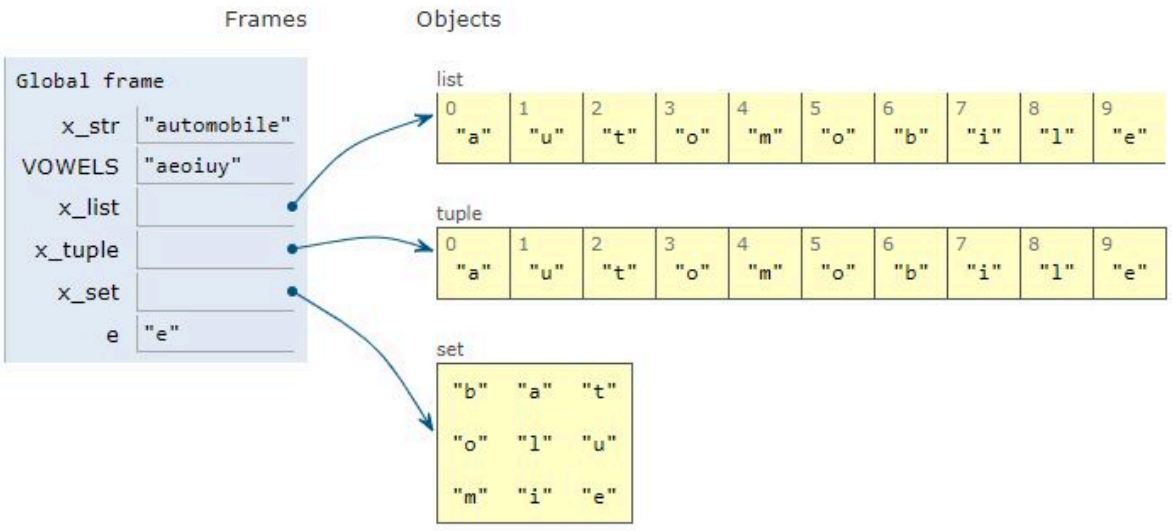
```
x_str   = "automobile"
x_list  = list(x_str)
x_tuple = tuple(x_str)
x_set   = set(x_str)
```

Suggested program:

```
x_string = 'automobile'
VOWELS = 'aeoiuy'

x_list  = list(x_str)
x_tuple = tuple(x_str)
x_set   = set(x_str)

print(" from list: ")
for e in x_list:
    if e not in VOWELS:
        print(e, end='')

print("\n from tuple: ")
for e in x_tuple:
    if not e in VOWELS:
        print(e, end='')
```

```
    print("\n from set: ")
    for e in x_set:
        if not e in VOWELS:
            print(e, end='')
```

```
 from list:
tmbl
 from tuple:
tmbl
 from set:
btlm
```



Test Yourself 3.2.07

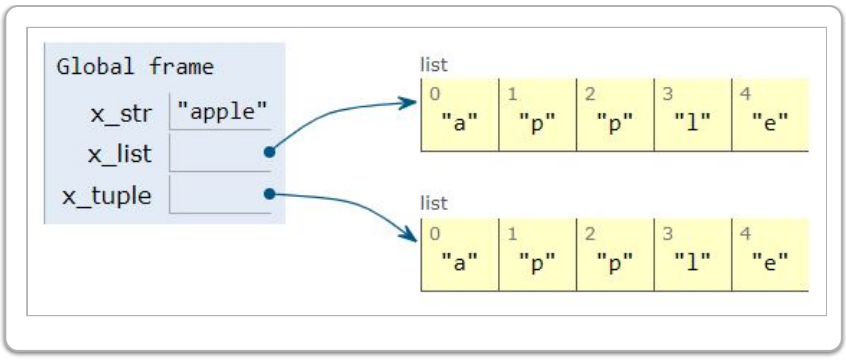
Based on the last exercise, why does *x_set* contain one less element than the other three collections?

Suggested answer: we have two characters "o", only one will be in the set.

Ordered Collections

- Enumerate: strings, lists and tuples are ordered collections.
- Support indexing & slicing.

```
x_string = 'apple'
x_list = ['a','p','p','l','e']
x_tuple = ('a','p','p','l','e')
```

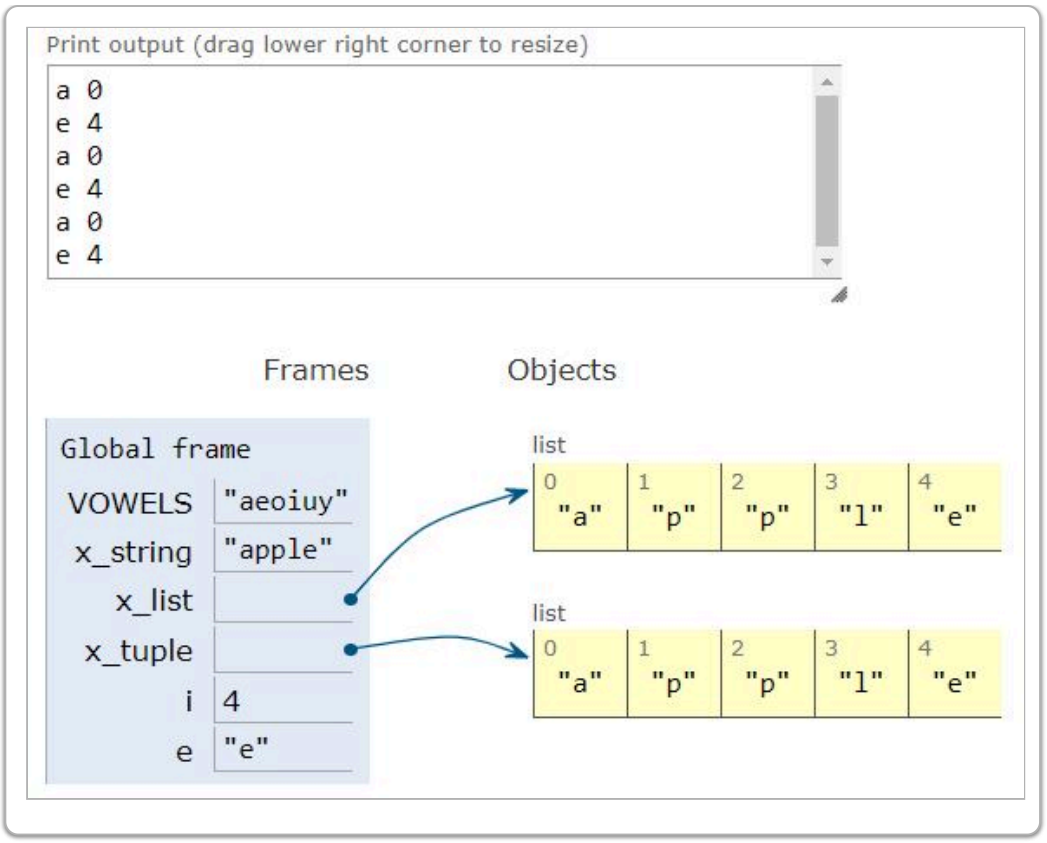


0	1	2	3	4
a	p	p	l	e

enumerate() in Collections

- Can use *enumerate()* in ordered collections only: strings, lists, tuples.

```
# print vowels and positions from collections
VOWELS = 'aeoiuy'
x_string = 'apple'
x_list   = ['a','p','p','l','e']
x_tuple  = ('a','p','p','l','e')

for i,e in enumerate(x_string):
    e = x_string[i]
    if e in VOWELS:
        print(e, i)

for i,e in enumerate(x_list):
    e = x_list[i]
    if e in VOWELS:
        print(e, i)

for i,e in enumerate(x_tuple):
    e = x_tuple[i]
    if e in VOWELS:
        print(e, i)
```



Test Yourself 3.2.08

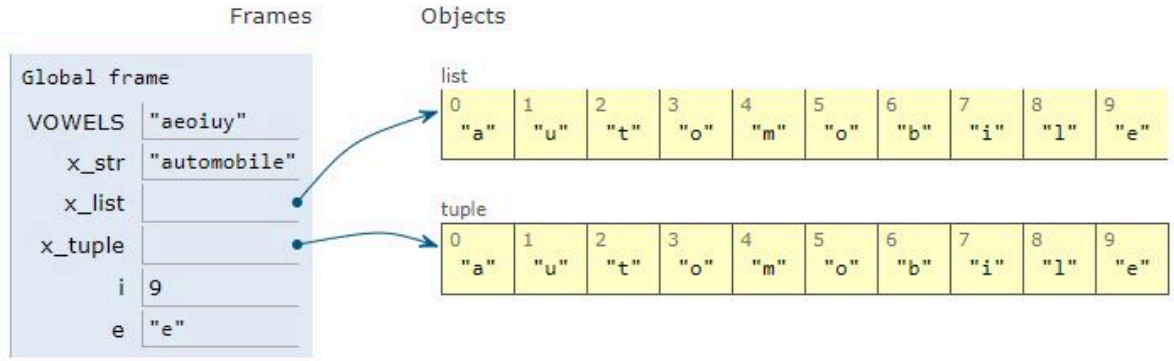
Use *enumerate()* iteration to print consonants and their positions from the following collections:

```
x_str   = "automobile"
x_list  = list(x_str)
x_tuple = tuple(x_str)
```

Suggested program:

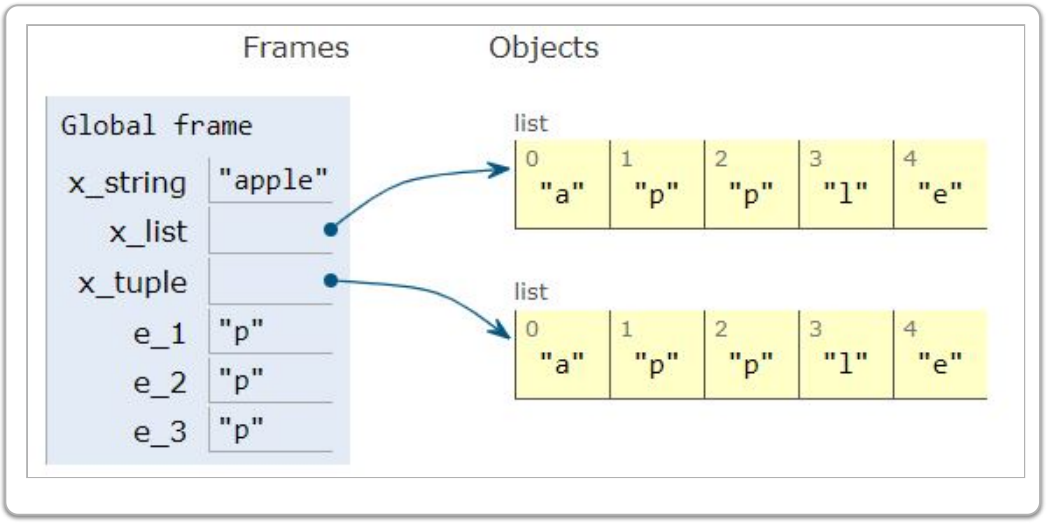
```
# print non - vowels and positions
VOWELS = 'aeoiuy'

x_str   = 'automobile'
x_list  = list(x_str)
x_tuple = tuple(x_str)

print(" from string: ")
for i,e in enumerate(x_str):
    if e not in VOWELS:
        print(e,i, end=" ")

print("\n from list: ")
for i,e in enumerate(x_list):
    if e not in VOWELS:
        print(e,i, end=" ")

print("\n from tuple: ")
for i,e in enumerate(x_tuple):
    if e not in VOWELS:
        print(e,i, end=" ")
```

```
 from string:
t 2 m 4 b 6 l 8
 from list:
t 2 m 4 b 6 l 8
 from tuple:
t 2 m 4 b 6 l 8
```



Indexing in Collections

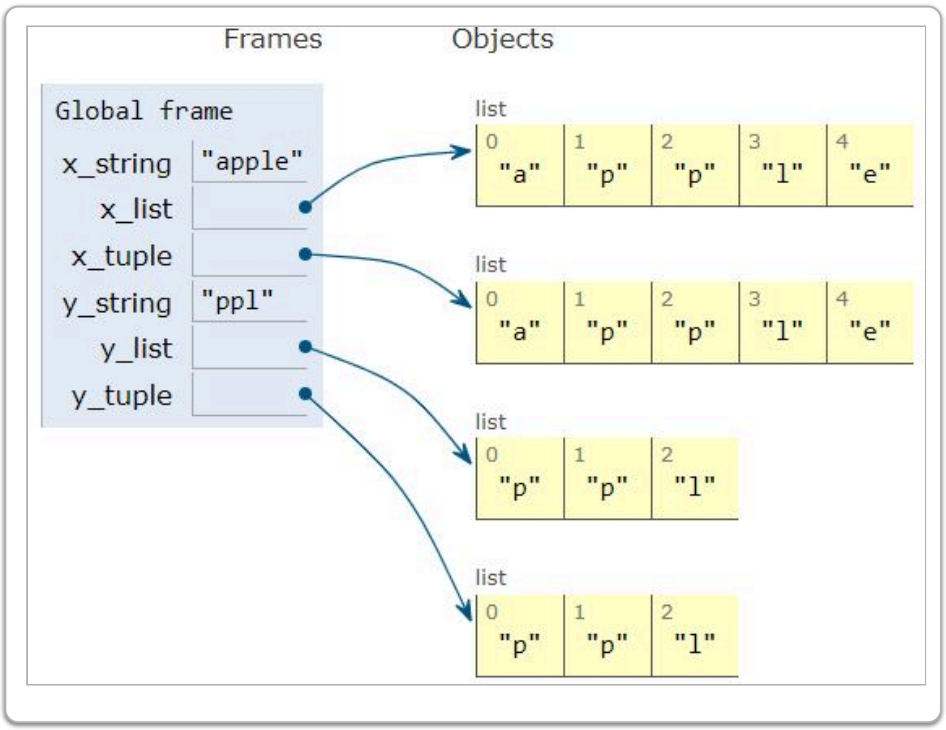
```
x_string = 'apple'
x_list   = ['a','p','p','l','e']
x_tuple  = ('a','p','p','l','e')
e_1      = x_string[1]
e_2      = x_list[1]
e_3      = x_tuple[1]
```



0	1	2	3	4
a	p	p	l	e

Slicing in Collections

```
x_string = 'apple'
x_list   = ['a','p','p','l','e']
x_tuple  = ('a','p','p','l','e')
y_string = x_string[1 : 4]
y_list   = x_list[1 : 4]
y_tuple  = x_tuple[1 : 4]
```



Mutability

Mutable collections:

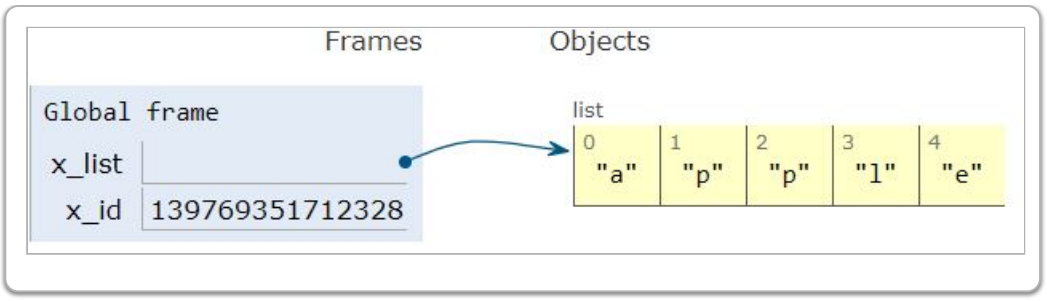
- lists
- set
- dictionary

Immutable collections:

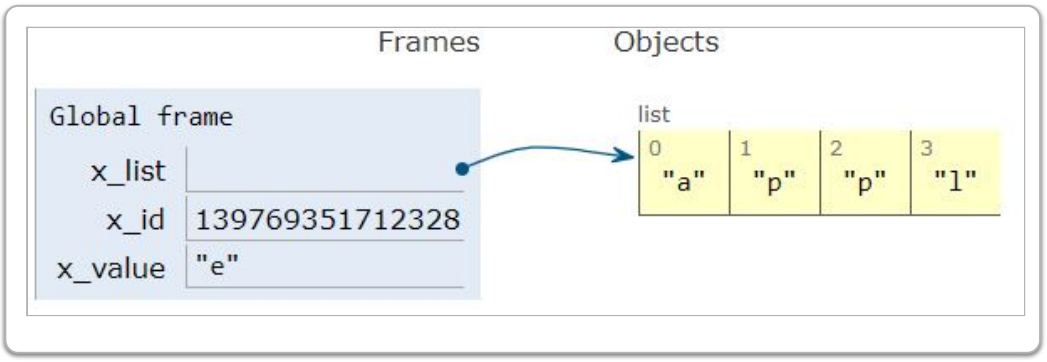
- strings
- tuples

List Mutability

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

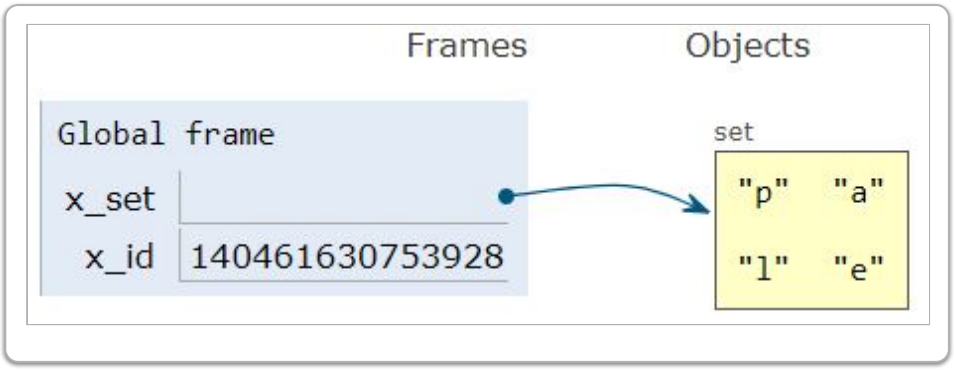



```
x_value = x_list.pop(-1)  # remove last
x_id    = id(x_list)
```

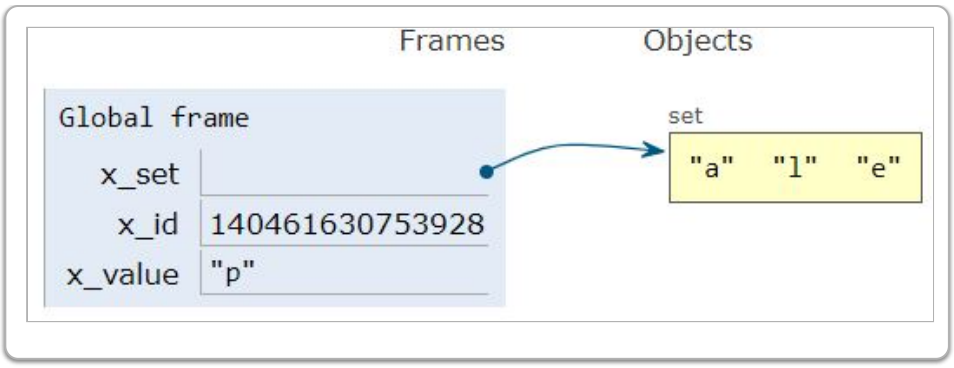


Set Mutability

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

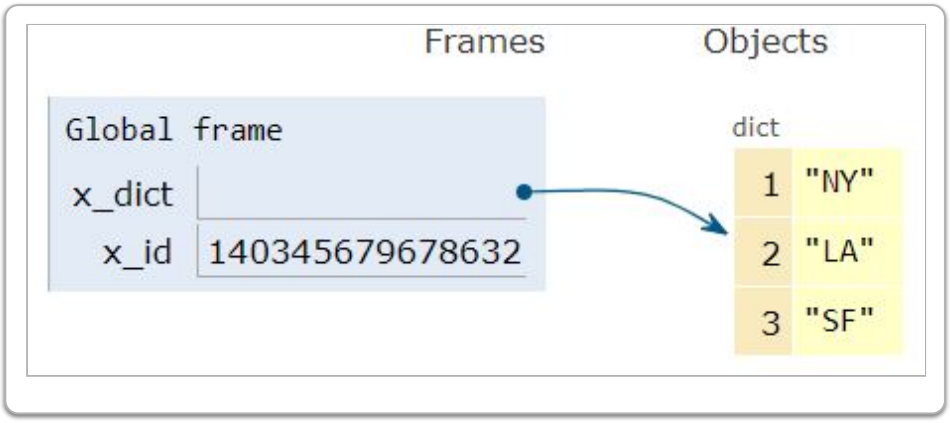



```
x_value = x_set.pop()  # remove random
x_id    = id(x_set)
```

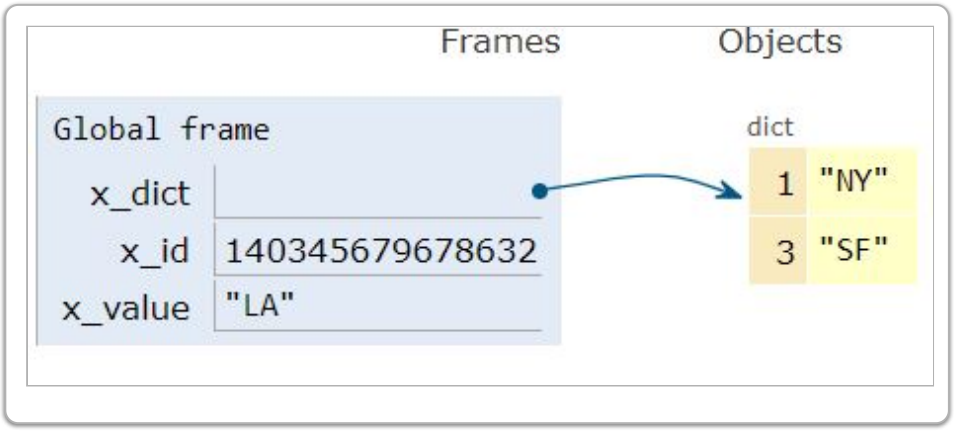


Dictionary Mutability

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




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



Summary of Collections

Collection	Ordered	Mutable

string	yes	no
list	yes	yes
tuple	yes	no
set	no	yes
dictionary	no	yes

There are some variations:

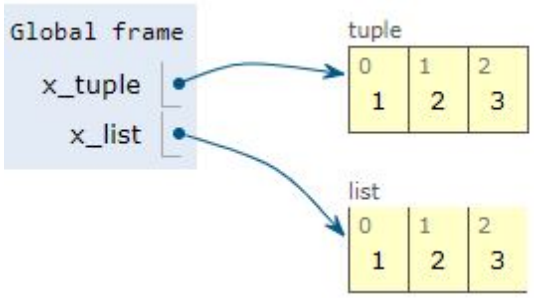
1. "frozen" set (immutable)
2. ordered dictionary

Test Yourself 3.2.09

Is it possible to convert an immutable collection to a mutable one with same elements?

Suggested program:

```
# it is possible: tuple → list
x_tuple = (1, 2, 3)
x_list  = list(x_tuple)
```

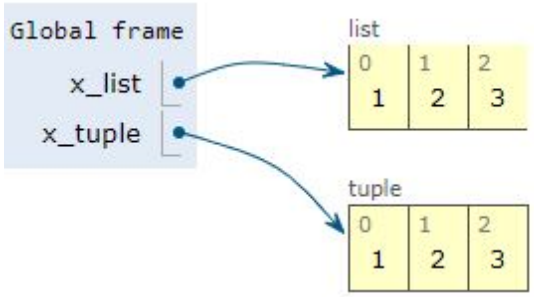


Test Yourself 3.2.10

Is it possible to convert a mutable collection to an immutable one with same elements?

Suggested program:

```
# it is possible: list → tuple
x_list = [1, 2, 3]
x_tuple  = tuple(x_list)
```



Common Methods - *clear()*, *copy()*, *count()*, and *index()*

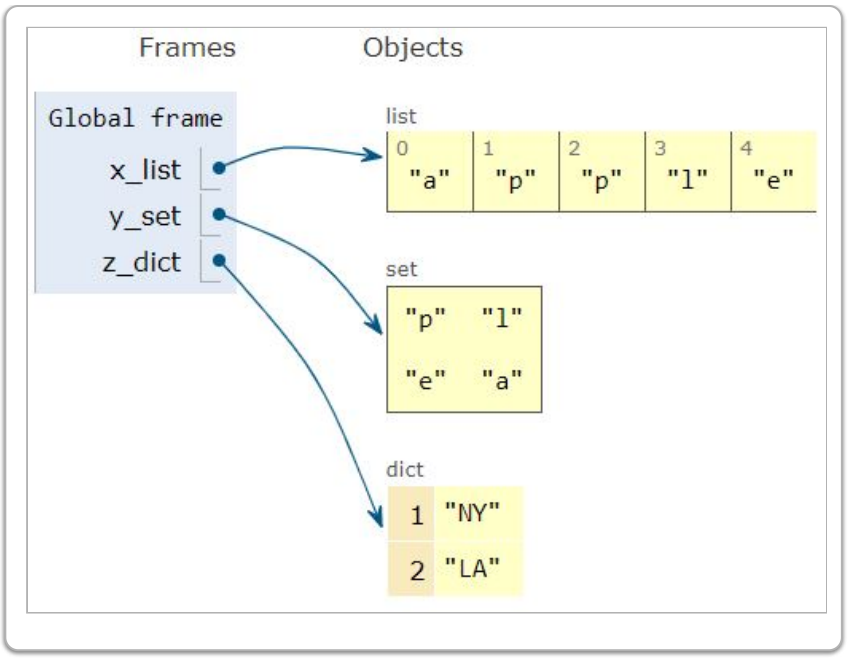
Method	str	list	tuple	set	dict
clear	no	yes	no	yes	yes
copy	no	yes	no	yes	yes
count	yes	yes	yes	no	no
index	yes	yes	yes	no	no
pop	no	yes	no	yes	yes
remove	no	yes	no	yes	no
update	no	no	no	yes	yes

1. Many polymorphic methods
2. Ordered collections: string, list, tuple
3. Mutable collections: list, set, dictionary

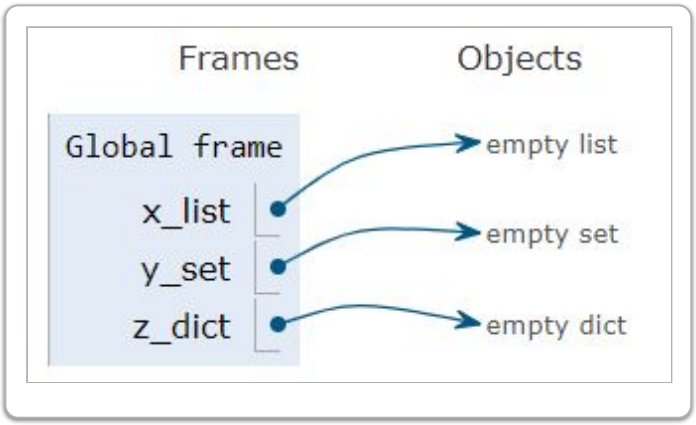
Collections: *clear()*

- The *clear()* method applies to mutable collections.
- The *clear()* method is done in place.

```
x_list = ['a','p','p','l','e']
y_set  = {'a','p','p','l','e'}
z_dict = {1: 'NY', 2: 'LA'}
```

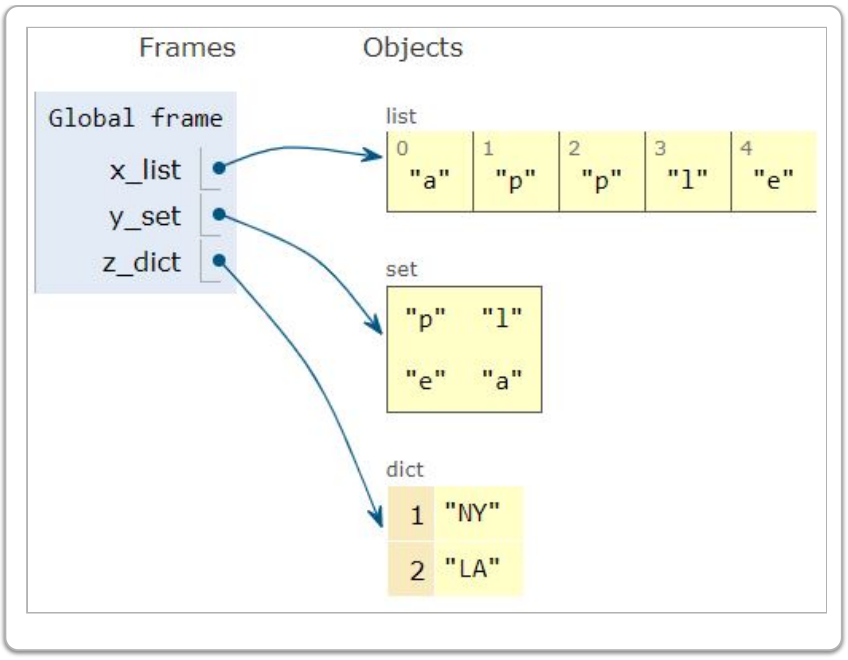



```
x_list.clear()
y_set.clear()
z_dict.clear()
```

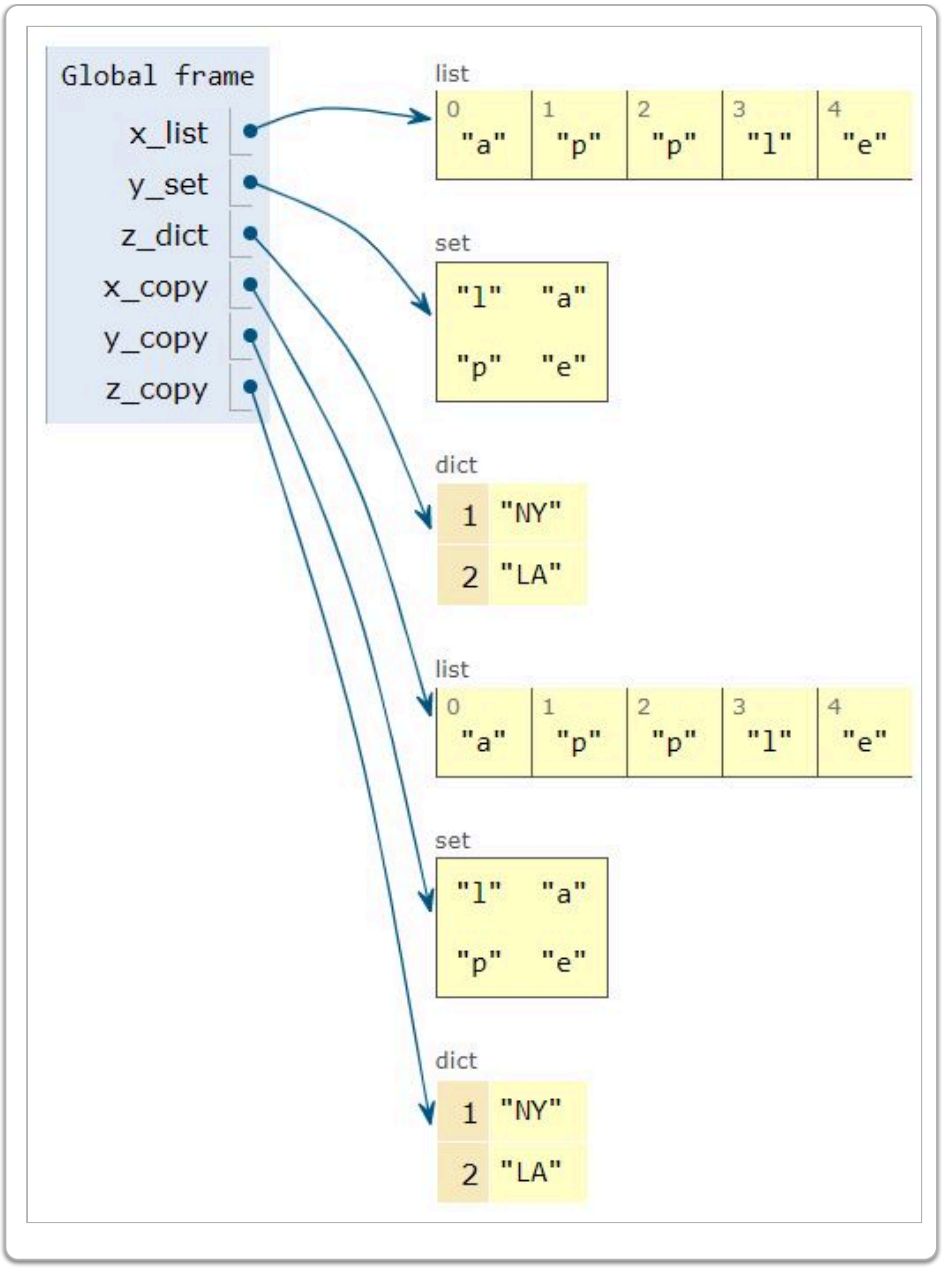


Collections: *copy()*

```
x_list = ['a','p','p','l','e']
y_set  = {'a','p','p','l','e'}
z_dict = {1: 'NY', 2: 'LA'}
```



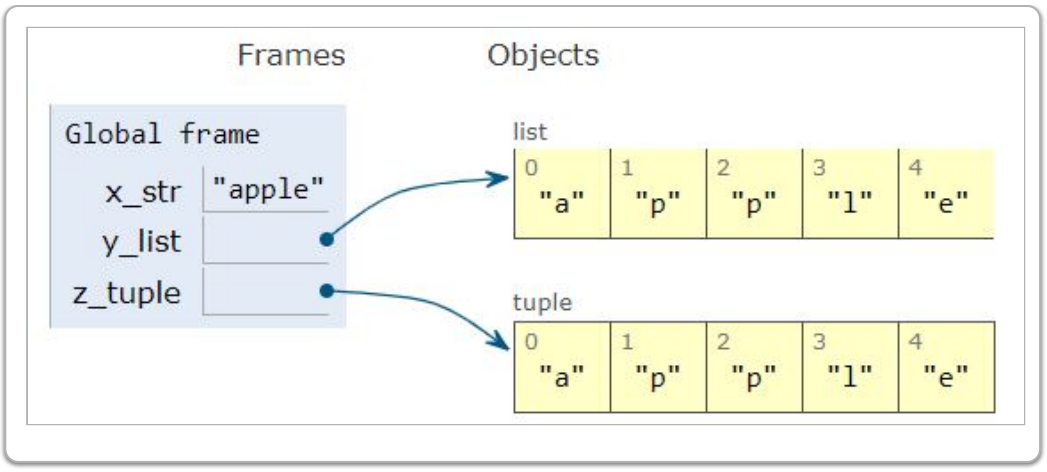

```
x_copy = x_list.copy()
y_copy = y_set.copy()
z_copy = z_dict.copy()
```



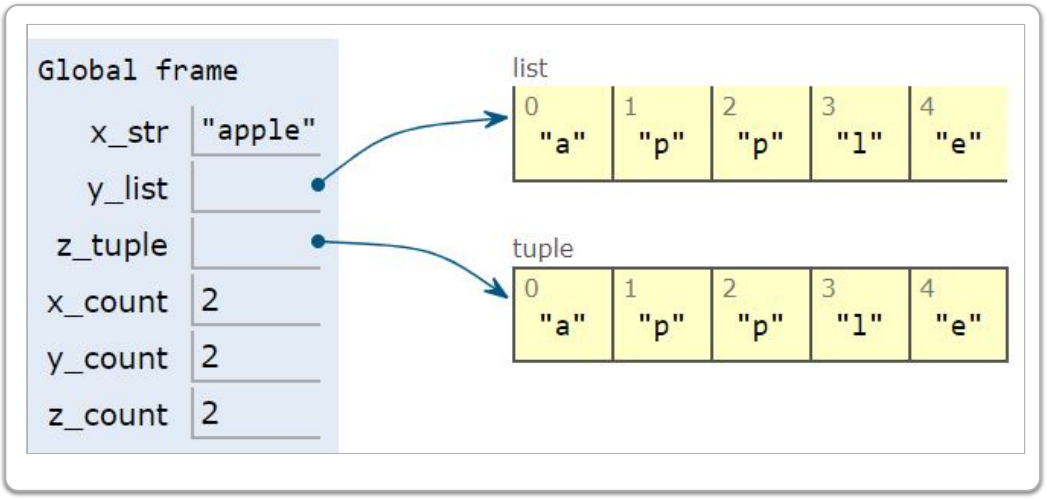
Collections: *count()*

The *count()* method counts number of occurencies.

```
x_str   = 'apple'
y_list  = ['a','p','p','l','e']
z_tuple = ('a','p','p','l','e')
```




```
x_count = x_str.count('p')
y_count = y_list.count('p')
z_count = z_tuple.count('p')
```



Test Yourself 3.2.11

Count the number of occurrences of "y" in

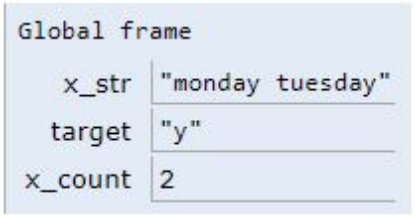
x_str="monday tuesday"

Suggested program:

```
x_str = "monday tuesday"
target = "y"
x_count = x_str.count(target)
print(target, ' occurs ', x_count, 'times')
```

```
y  occurs  2  times
```

Frames



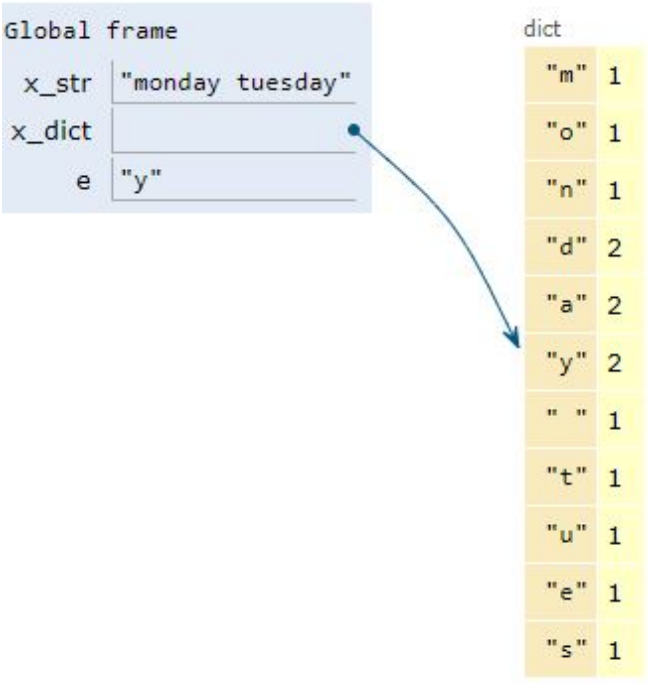
Test Yourself 3.2.12

For each letter construct a dictionary of frequency counts:

x_str="monday tuesday"

Suggested program:

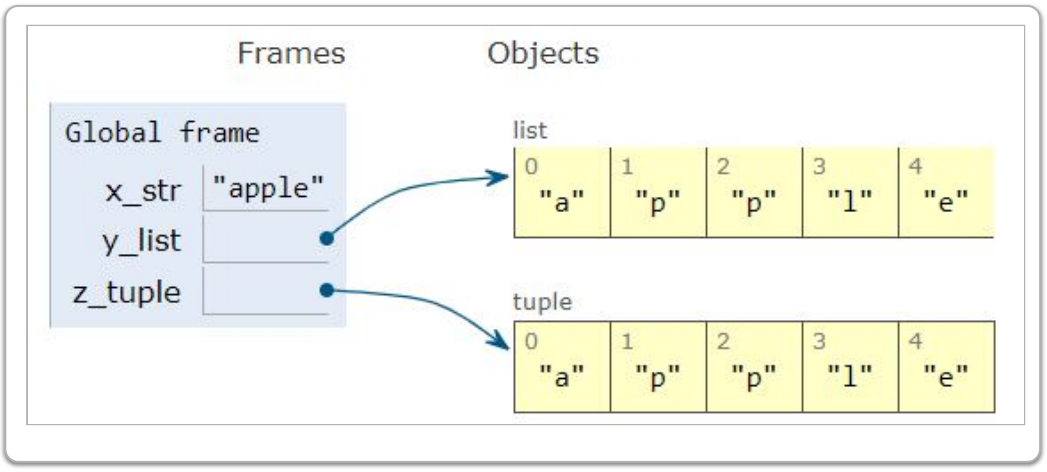
```
x_str = "monday tuesday"
x_dict = dict()

for e in x_str:
    if e not in x_dict.keys():
        x_dict[e] = 1
    else:
        x_dict[e] = x_dict[e] + 1
```



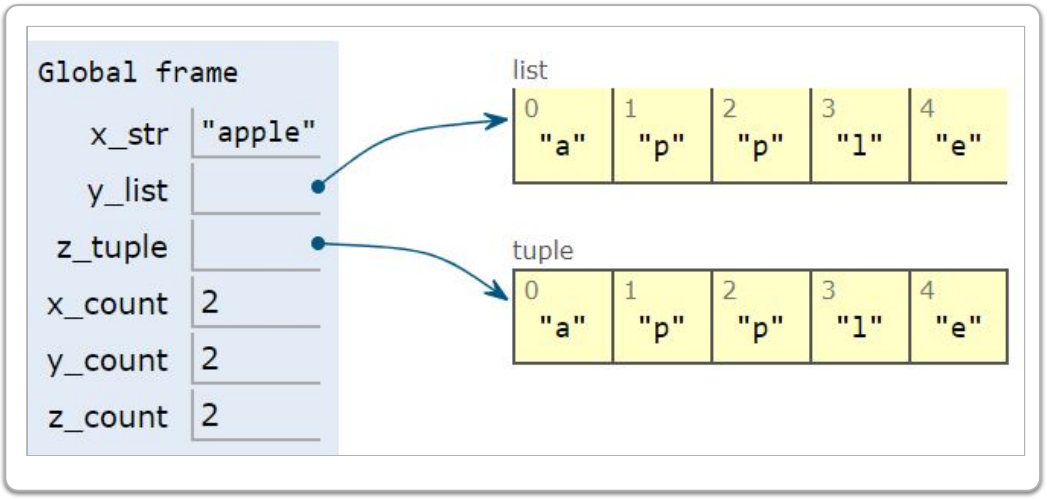
Collections: *index()*

- The *index()* method applies to ordered collections.
- The *index()* method indexes of first ocuurency.
- Note: value must exist.

```
x_str   = 'apple'
y_list  = ['a','p','p','l','e']
z_tuple = ('a','p','p','l','e')
```




```
x_count = x_str.index('e')
y_count = y_list.index('p')
z_count = z_tuple.index('l')
```



Test Yourself 3.2.13

Count how many times the letter "a" appears in

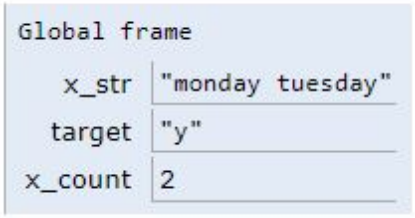
x_str="monday tuesday"

Suggested program:

```
x_str = "monday tuesday"
x_count = x_str.count("a")
```

```
y  occurs  2  times
```

Frames



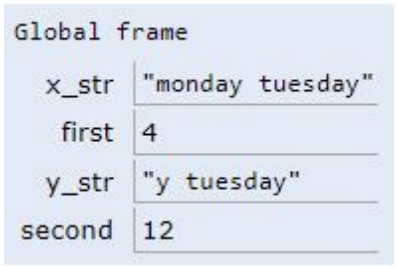
Test Yourself 3.2.14

Compute the position of second "a" in the same string

x_str="monday tuesday"

Suggested program:

```
x_str = "monday tuesday"
first = x_str.index("a")
if first >= 0:
    y_str = x_str[first + 1 :]
    second = y_str.index("a")
    if second >=0:
        second = (first + 1) + second
```



Common Methods - *pop()*, *remove()*, and *update()*

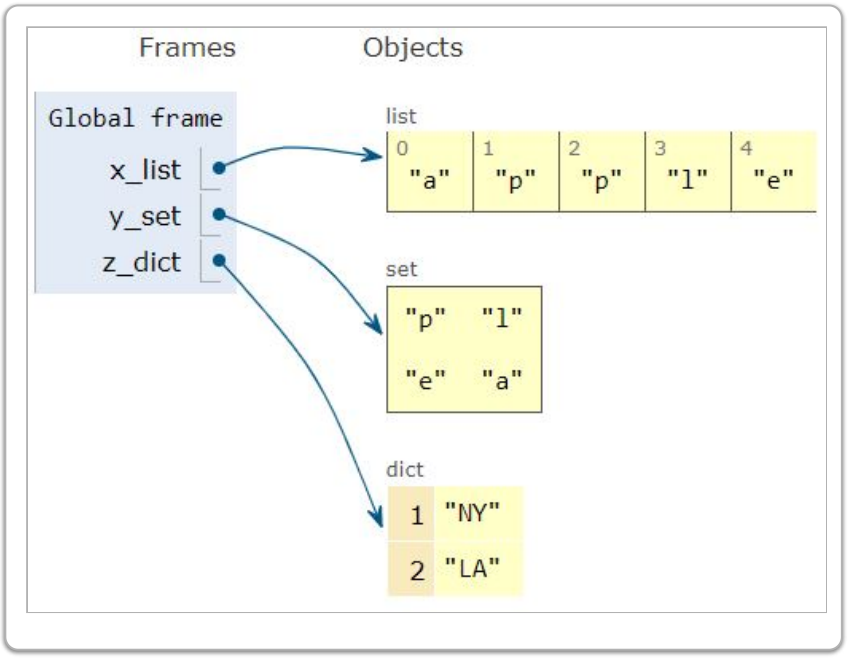
Method	str	list	tuple	set	dict

clear	no	yes	no	yes	yes
copy	no	yes	no	yes	yes
count	yes	yes	yes	no	no
index	yes	yes	yes	no	no
pop	no	yes	no	yes	yes
remove	no	yes	no	yes	no
update	no	no	no	yes	yes

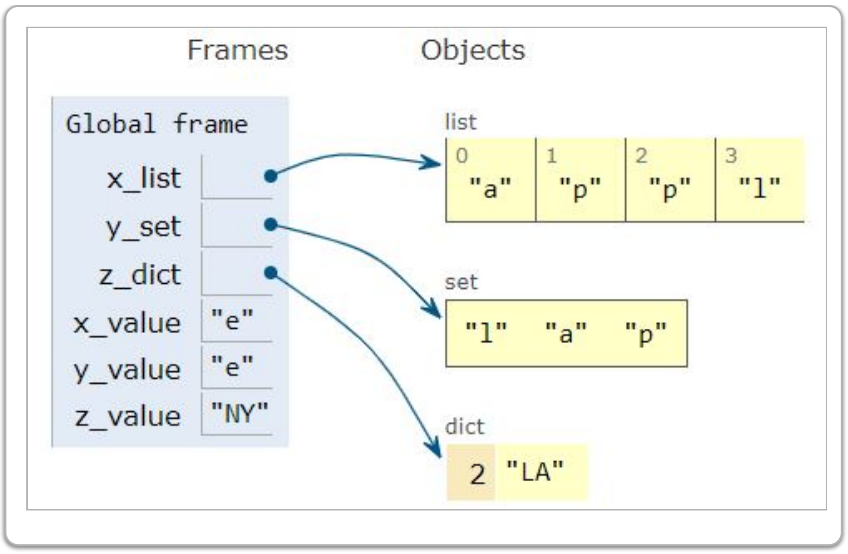
Collections: *pop()*

- The *pop()* method applies to mutable collections.
- The *pop()* method is done in place.

```
x_list = ['a','p','p','l','e']
y_set  = {'a','p','p','l','e'}
z_dict = {1: 'NY', 2: 'LA'}
```



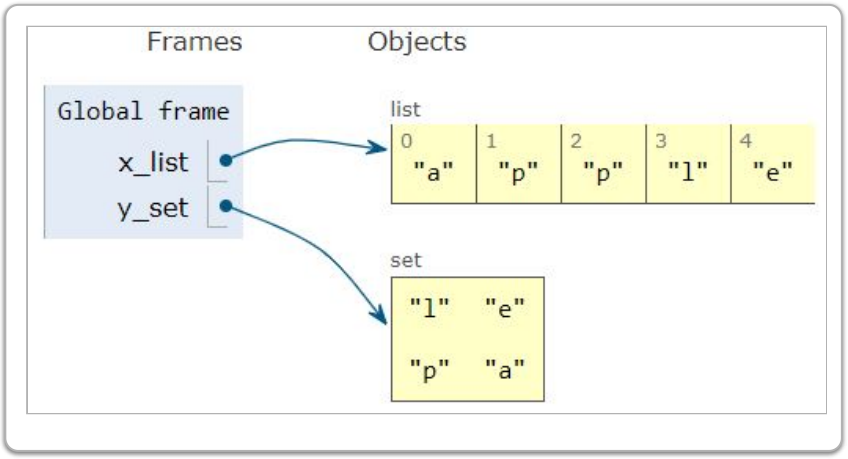

```
x_value = x_list.pop(-1) # last element
y_value = y_set.pop()    # random element
z_value = z_dict.pop(1)  # value for key =1
```



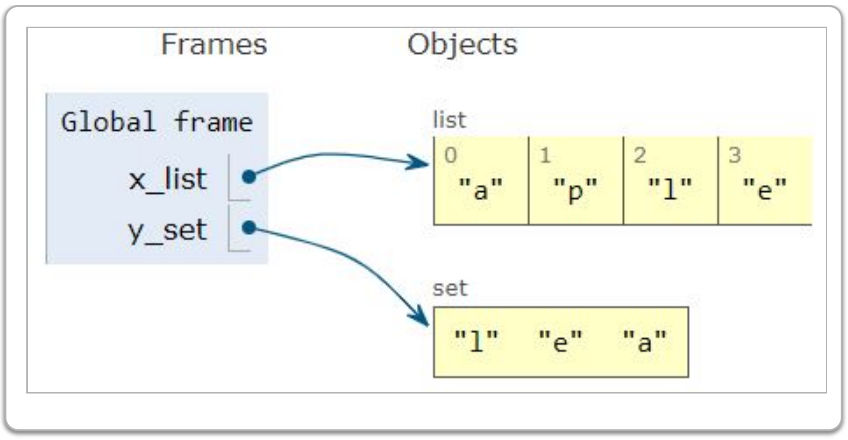
Collections: *remove()*

- The *remove()* method removes by value.
- The *remove()* method applies to mutable collections.
- The *remove()* method is done in place.

```
x_list = ['a','p','p','l','e']
y_set  = {'a','p','p','l','e'}
```




```
x_list.remove('p')
y_set.remove('p')
```



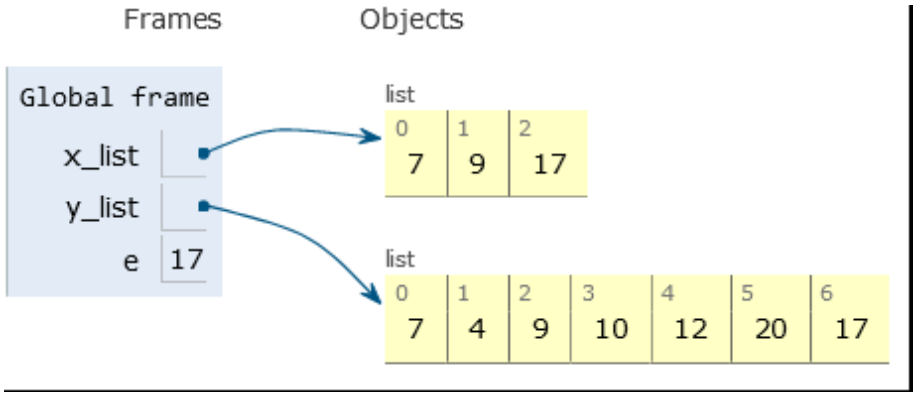
Test Yourself 3.2.15

Remove even numbers from

x_list=[7,4,9,10,12,20,17]

Suggested program:

```
x_list = [7,4,9,10,12,20,17]
x_set  = set(x_list)
for e in x_set:
    if e % 2 == 0:
        x_list.remove(e)
```



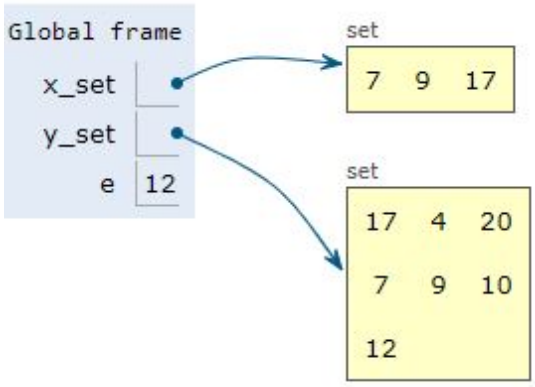
Test Yourself 3.2.16

Remove even numbers from

x_set={7,4,9,10,12,20,17}

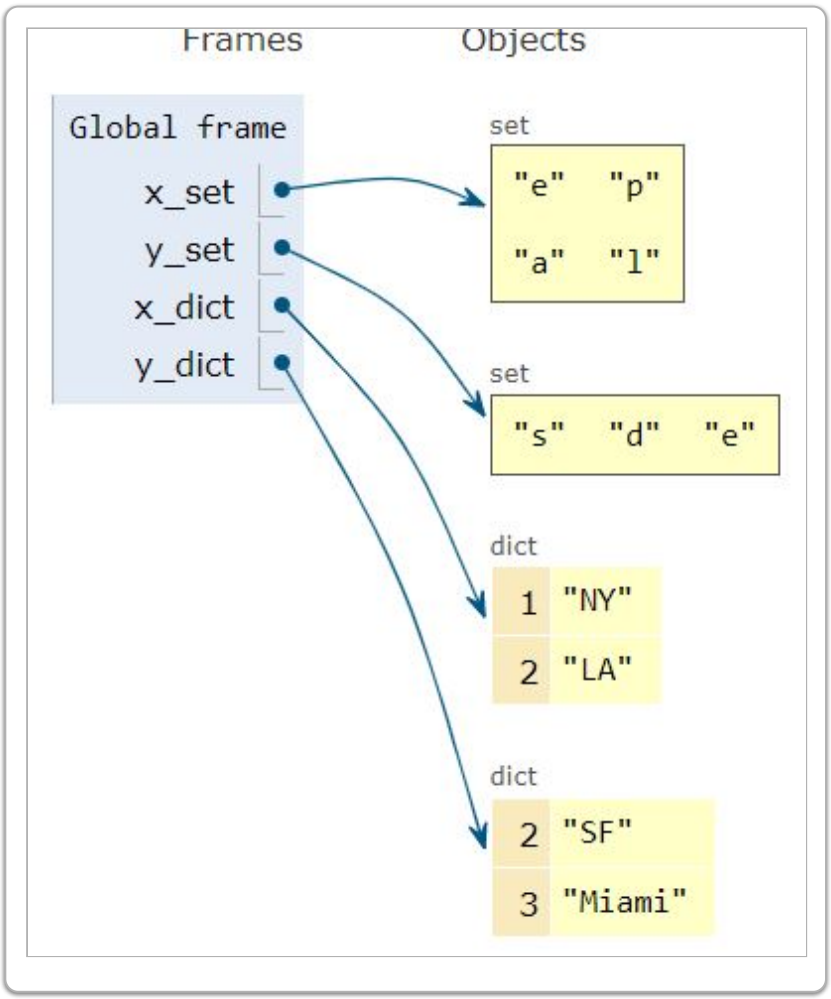
Suggested program:

```
x_set = {7,4,9,10,12,20,17}
y_set  = x_set.copy()
for e in y_set:
    if e % 2 == 0:
        x_set.remove(e)
```

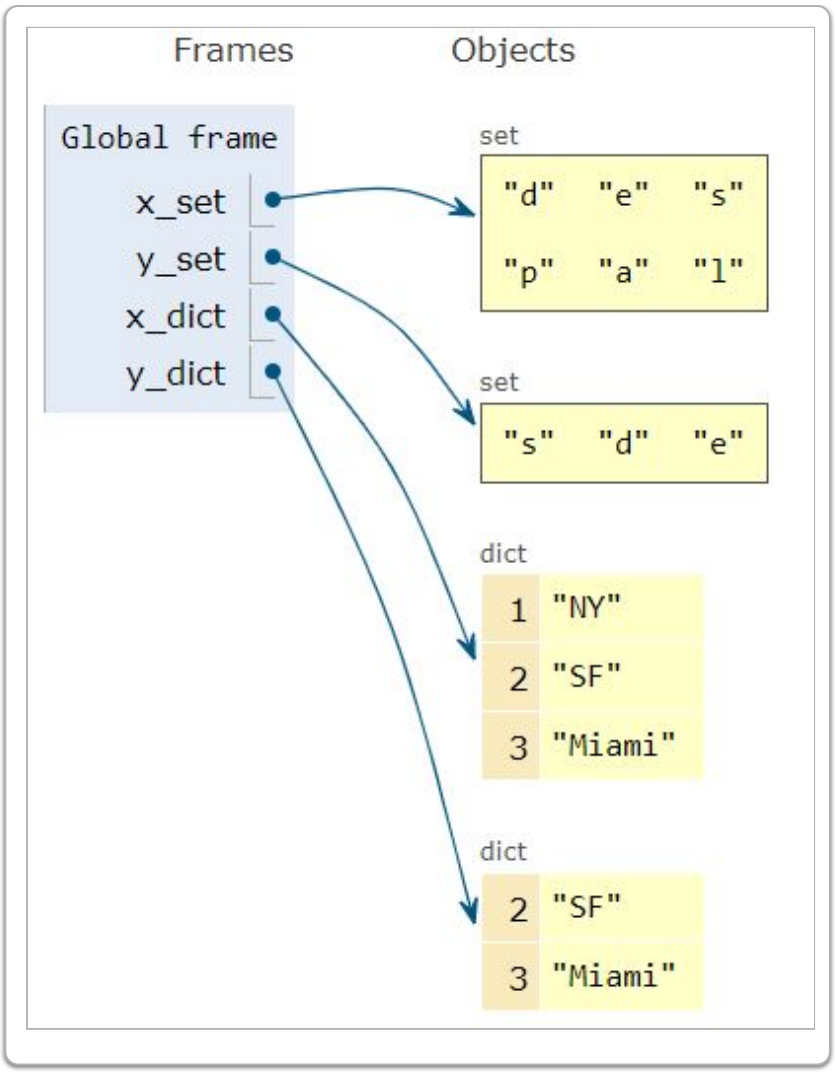


Collections: *update()*

```
x_set  = {'a','p','p','l','e'}
y_set  = {'s','e','e','d']
x_dict = {1: 'NY', 2: 'LA'}
y_dict = {2: 'SF', 3: 'Miami'}
```




```
x_set.update(y_set)
x_dict.update(y_dict)
```



Test Yourself 3.2.17

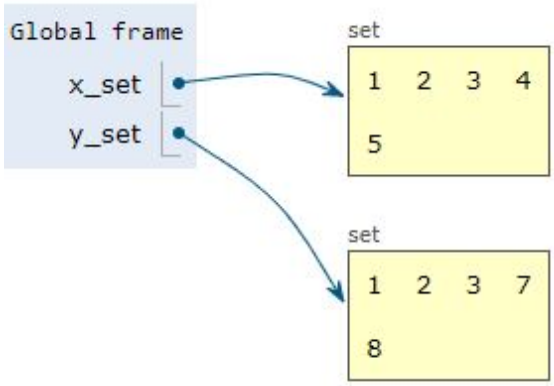
Use *update()* to transform *x_set* into *y_set*

x_set = {1, 2, 3, 4, 5}

y_set = {1, 2, 3, 7, 8}

Suggested program:

```
x_set = {1, 2, 3, 4, 5}
y_set = x_set.copy()
y_set.remove(4)
y_set.remove(5)
y_set.update({7 ,8})
```



Collections: Summary

Collections Summary

- string, list, tuple, set, dictionary
- iterable objects
- support membership methods
- ordered: strings, lists, tuples
- indexing & slicing (ordered)
- mutable: lists, sets, dictionary
- many common methods: clear(), copy(), count(), index(), pop(), remove(), update()

Control Flow

Control Flow Constructs

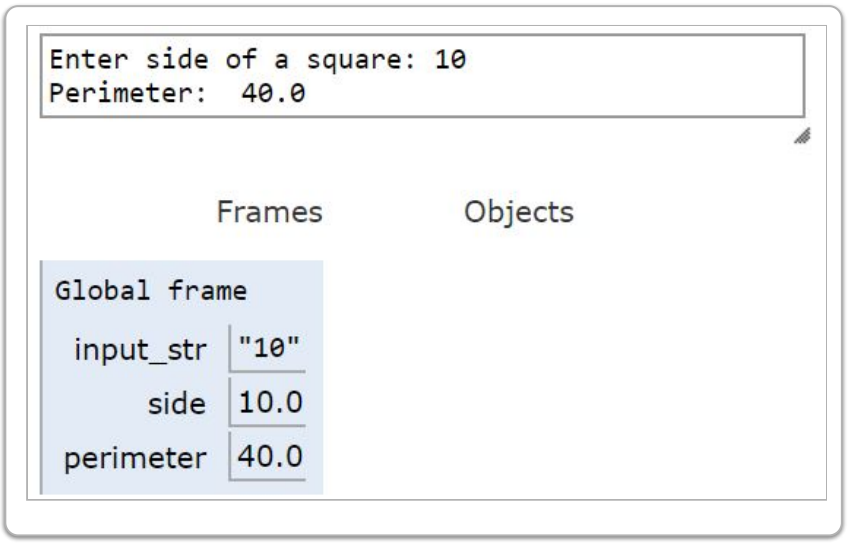
Q: Python programs execute line by line. How to change this sequence?

A: Can use control flow.

1. *if else* – conditional
2. *for* – 1-st loop construct
3. *while* – 2-nd loop construct

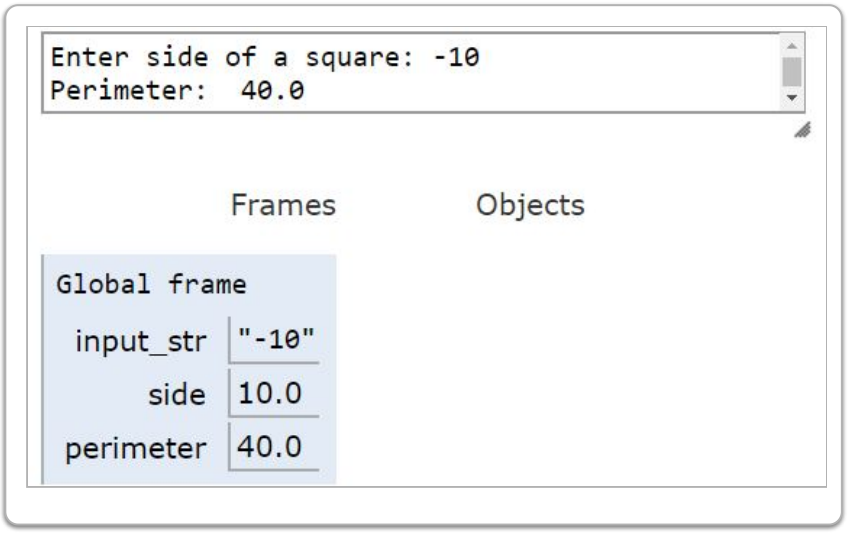
if Statement

```
input_str = input('Enter side of a square: ')
side      = float(input_str)
perimeter = 4 * side
print('Perimeter: ', perimeter)
```



The above program calculates a square's perimeter according to the user-inputted side length, but there is no checking for type or sign. The program should check the validity of user input by using an *if* statement. In the example below, a conditional statement is used.

```
input_str = input('Enter side of a square: ')
side      = float(input_str)
if side < 0:
    side      = abs(side)
perimeter = 4 * side
print('Perimeter: ', perimeter)
```



if Statement Syntax

```
if logical condition:
    statement 1
    statement 2
--------
execution continues
```

As we can see from the perimeter calculation example, logical condition evaluates to *True* or *False*.

```
input_str = input('Enter side of a square: ')
side      = float(input_str)
if side < 0:
    side      = abs(side)
    perimeter = 4 * side
print('Perimeter: ', perimeter)
```

Test Yourself 3.3.01

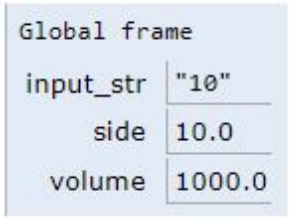
Write a program to ask for a side of a cube and compute its volume.

Suggested program:

```
input_str = input('Enter side of a cube: ')
side      = float(input_str)
if side < 0:
    side  = abs(side)
volume = side**3
print('Cube Volume: ', volume)
```

```
Enter side of a cube: 10
Cube Volume:  1000.0
```

Frames Objects



Test Yourself 3.3.02

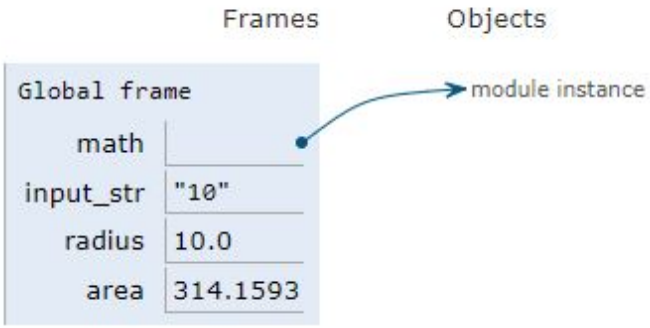
Write a program to ask for a radius of a circle and compute its area.

Suggested program:

```
import math
input_str = input('Enter radius of a circle: ')
radius    = float(input_str)
if radius < 0:
    radius = abs(radius)
```

```
area = math.pi * radiu **2
print('Circle Area: ', area)
```

```
Enter radius of a circle: 10
Circle Area:  314.1592653589793
```

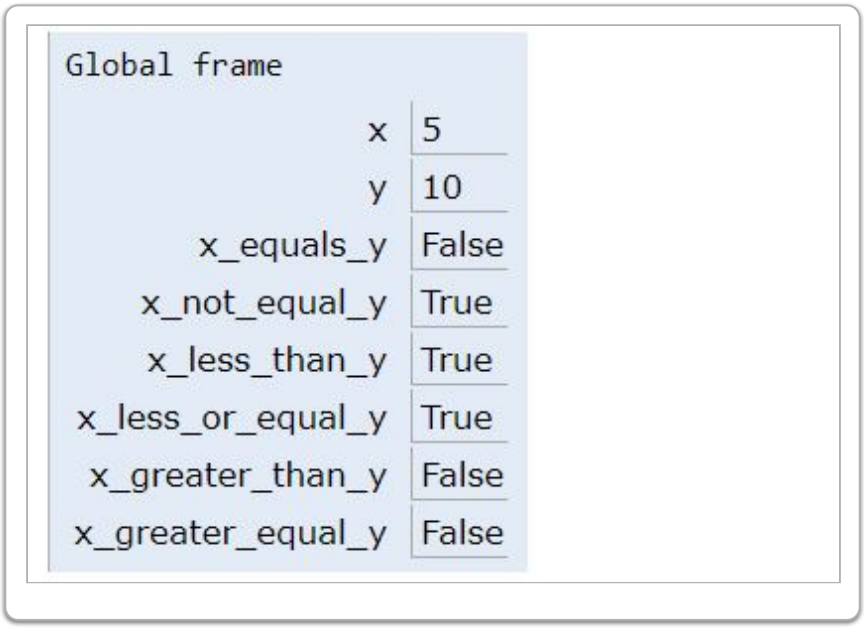


Logical Conditions

Logical Operators

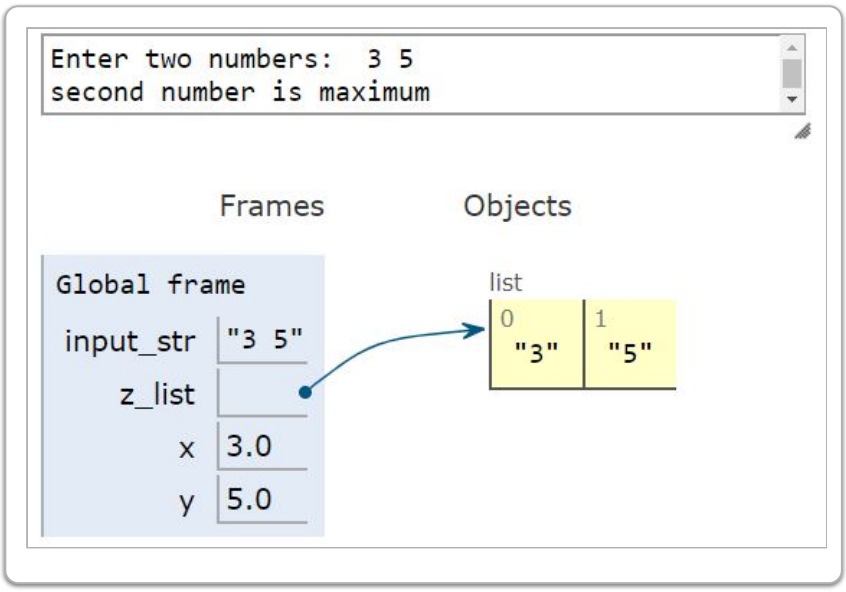
==, !=, <, <=, >, >=

```
x, y              = 5, 100
x_equals_y        = (x == y)
x_not_equal_y     = (x != y)
x_less_than_y     = (x < y)
x_less_or_equal_y = (x <= y)
x_greater_than_y  = (x > y)
x_greater_equal_y = (x >= y)
```



if...else Statement

```
input_str = input('Enter two numbers: ')
z_list = input_str.split()
x = float(z_list[0]); y = float(z_list[1])
if x >= y:
    print('first number is maximum')
else:
    print('second number is maximum')
```



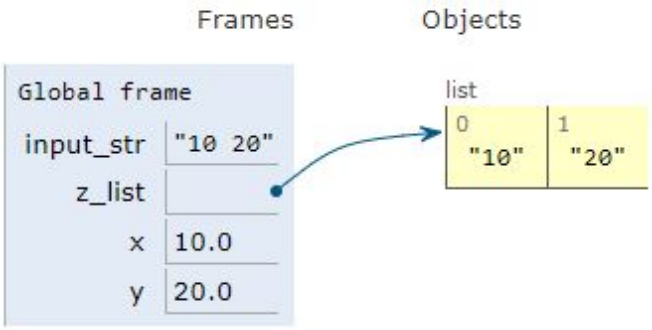
Test Yourself 3.3.03

Write a program to ask for two numbers and compute their maximum.

Suggested program:

```
input_str = input("Enter two numbers: ")
z_list = input_str.split()
x = float(z_list[0])
y = float(z_list[1])
if x >= y:
     print("max is: ", x)
else:
     print("max is: ", y)
```

```
Enter two numbers:  10 20
max is :  20.0
```



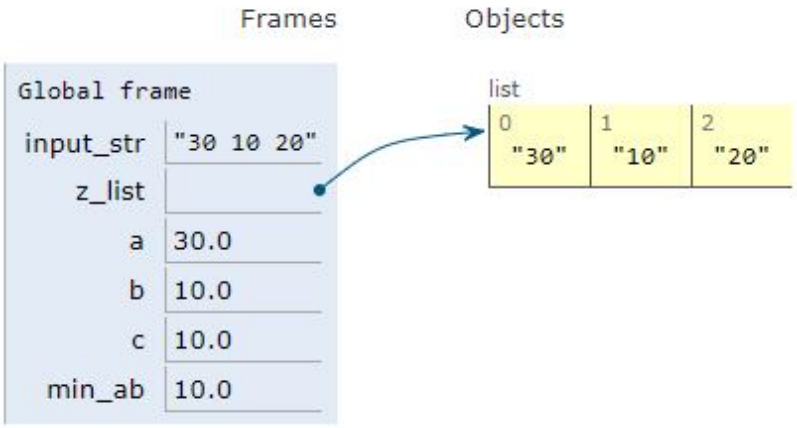
Test Yourself 3.3.04

Write a program to ask for three numbers and compute their minimum.

Suggested program:

```
# without using elif construct
input_str = input("Enter three numbers: ")
z_list = input_str.split()
a = float(z_list[0])
b = float(z_list[1])
c = float(z_list[2])

if a <= b:
     min_ab = a
else:
     min_ab = b
if min_ab <= c:
     print("min of three numbers is ", min_ab)
else:
     print("min of three numbers is ", c)
```

```
Enter three numbers:30 10 20
min of three numbers is  10.0
```



if...else with Logical Operators

```
a, b, c = 2, 6, 4
if a < b and b < c
    print(a,b,c, ' is increasing sequence')
else:
    print(a,b,c, ' is not increasing sequence')
```

```
2 6 4  is not increasing sequence
```

Test Yourself 3.3.05

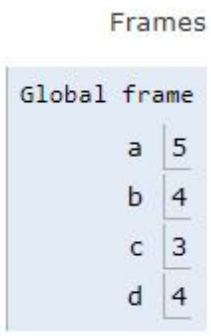
Write a program to check if numbers *a, b, c, d* form a decreasing sequence.

Suggested program:

```
a = 5; b = 4; c = 3; d = 4
if (a>=b) and (b>=c) and (c>=d):
    print(a,b,c,d, " is a decreasing seq")
```

```
    else:
        print(a,b,c,d, " is not a decreasing seq")
```

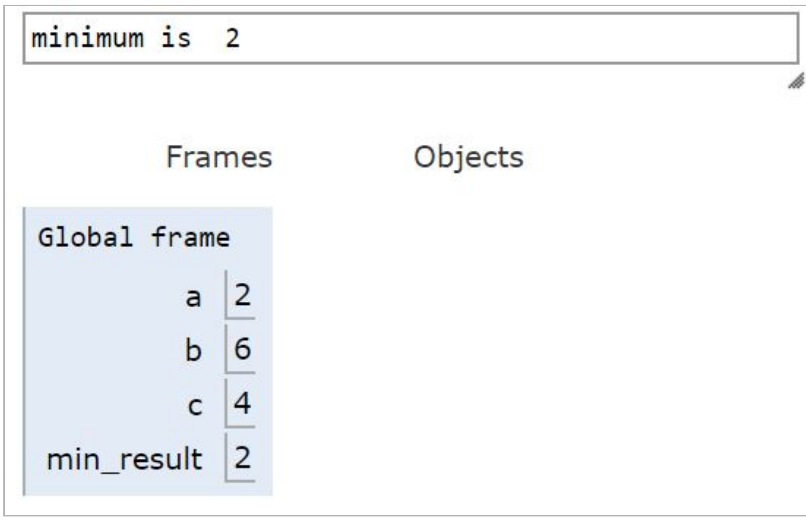
```
5 4 3 4  is not a decreasing seq
```



Nested *if ... elif ... else*

```
a, b, c = 2, 6, 4
if (a <= b) and (a <= c)
    min_result = a
elif (b <= a) and (b <= c)
    min_result = b
else:
    min_result = c
print('minimum is ', str(min_result))
```

```
minimum is  2
```



Python Iteration Constructs

There are two iteration constructs:

a. *for* loop

```
fibonacci = [1,1,2,3,5,8,13,21,34,55]
x_sum = 0
for e in fibonacci:
    x_sum = x_sum + e
```

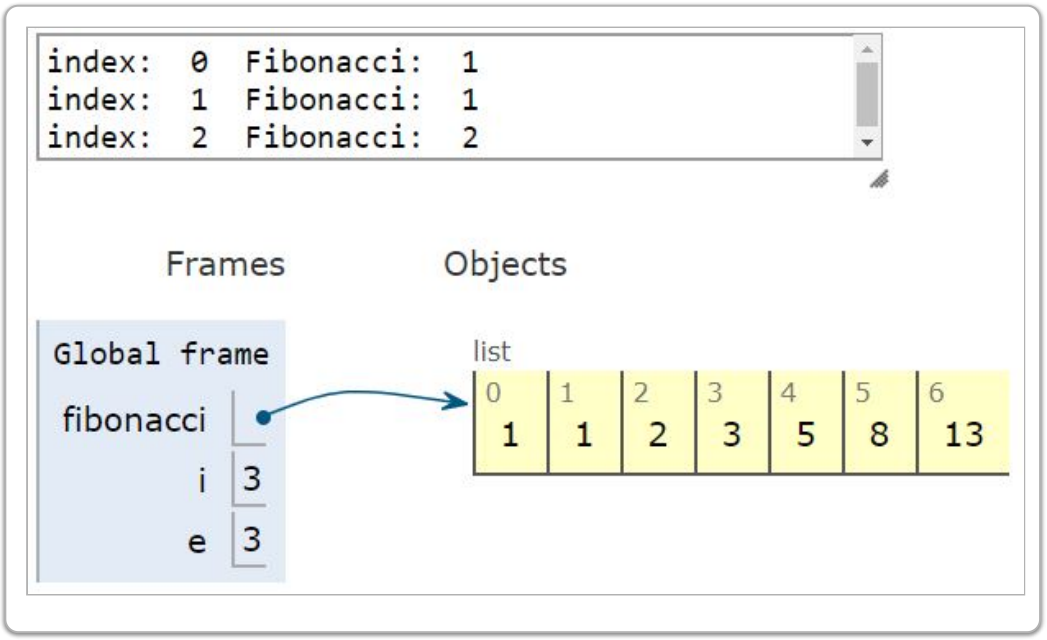
b. *while* loop

```
fibonacci = [1,1,2,3,5,8,13,21,34,55]
y_sum = 0
counter = len(fibonacci)
while(counter > 0):
    e       = fibonacci.pop()
    y_sum   = y_sum + e
    counter = counter - 1
```

for Statement

for statement has the same syntax across collections.

```
fibonacci = [1,1,2,3,5,8,13]
for i, e in enumerate (fibonacci):
    print(index, str(i),
              'Fibonacci: ', str(e))
```



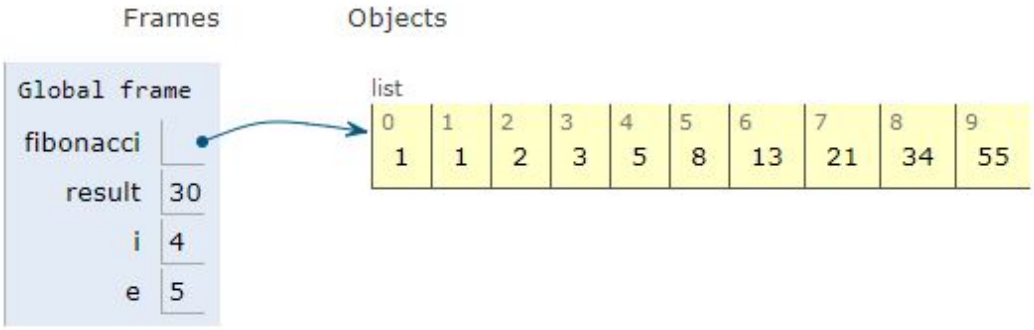
Test Yourself 3.3.06

Use *for* loop to compute the product of first five Fibonacci numbers.

Suggested program:

```
fibonacci = [1,1,2,3,5,8,13,21,34,55]
result = 1
for i in range(5):
    e = fibonacci[i]
    result = result * e
print("product of 5 fib. nambers: ", result)
```

```
product of 5 fib. nambers:  30
```

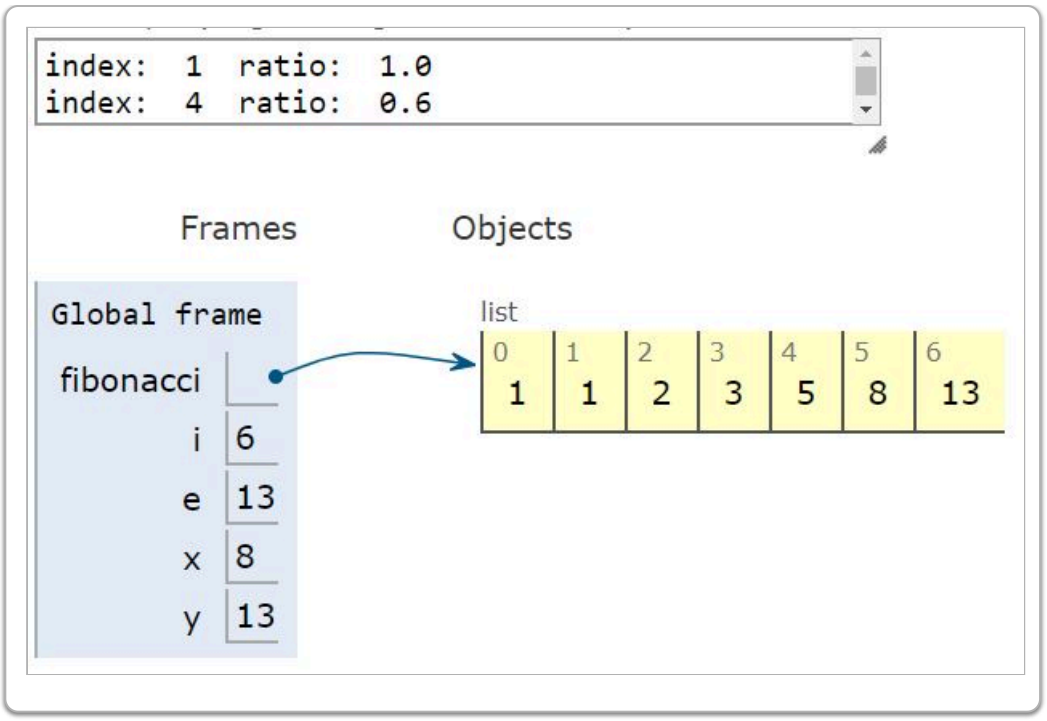


continue Statement

```
for <iterator> in <iterable> :
    statement 1
------
statement
if <test> :
    continue
```

The following example uses the *continue* statement to calculate the ratio of consecutive odd fibonacci numbers.

```
fibonacci = [1,1,2,3,5,8,13]
for i, e in enumerate (fibonacci):
    if i > 0:
        x = fibonacci[i-1]; y = fibonacci[i]
        if(x%2 == 0) or (y%2 == 0):
            continue
        print('index: ', i, 'ratio: ',float(x)/y)
```

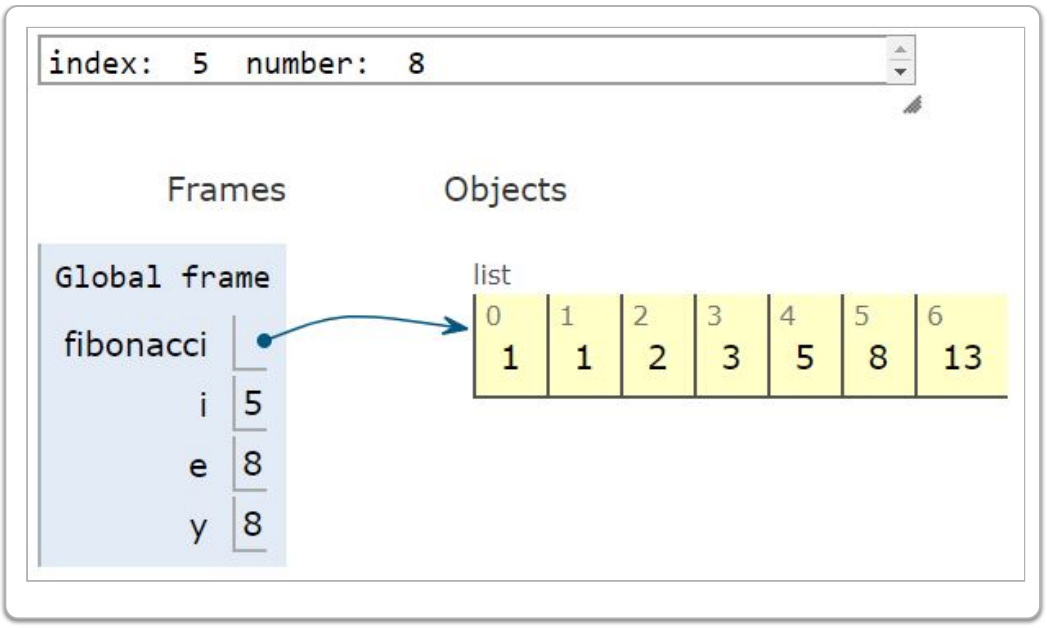


break Statement

```
for <iterator> in <iterable> :
    statement 1
------
statement
if <test>:
    break
statement n
```

The following example uses the *break* statement to find the first number bigger than 5.

```
fibonacci = [1,1,2,3,5,8,13]
for i, e in enumerate (fibonacci):
    if i > 0:
        y = fibonacci[i]
        if y > 5:
                    break
print('index: ', i, 'number: ', y)
```

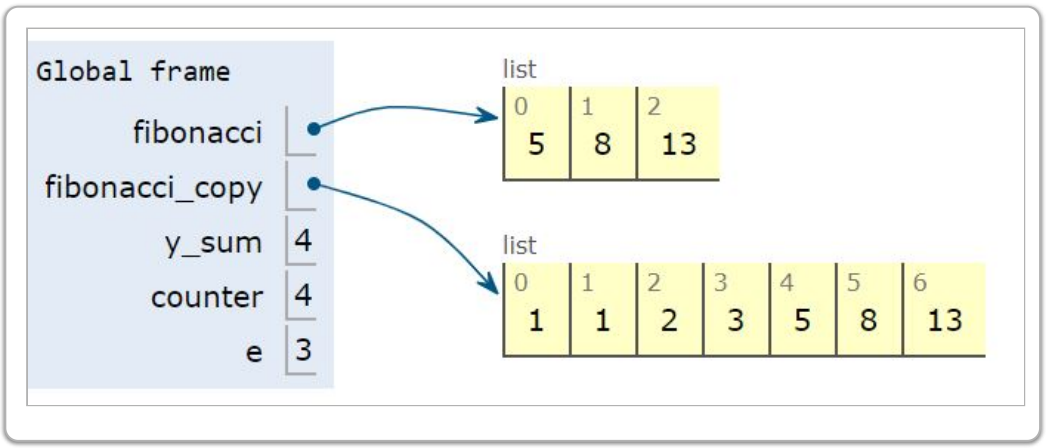


while Statement

```
while condition is True:
    statement 1
    -----
    statement n
execution continues
```

The following example uses the *while* statement to calculate the sum of fibonacci numbers.

```
fibonacci      = [1,1,2,3,5,8,13]
fibonacci_copy = [1,1,2,3,5,8,13]
y_sum = 0
counter = len(fibonacci)
while (counter > 0):
    e       = fibonacci.pop(0)
    y_sum   = y_sum + e
    counter = counter - 1
```



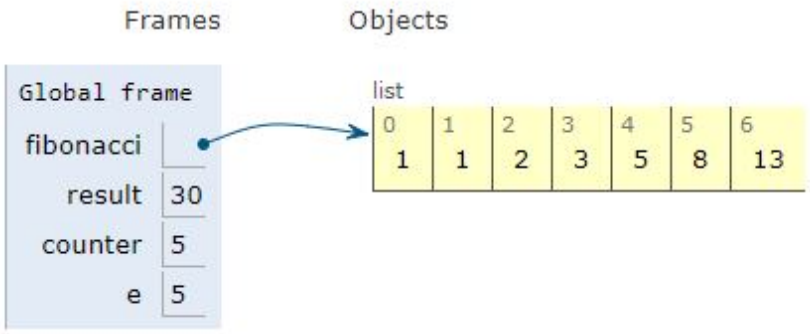
Test Yourself 3.3.07

Use *while* loop to compute the product of first five Fibonacci numbers.

Suggested program:

```
fibonacci = [1,1,2,3,5,8,13]
result    = 1
counter   = 0
while(counter < 5):
    e       = fibonacci[counter]
    result  = result * e
    counter = counter + 1
print("product of 5 fib. nambers: ", result)
```

```
product of 5 fibonacci:  30
```



pass Statement

The *pass* statement used when

- statement required by Python
- placeholder for future code

```
fibonacci = [1,1,2,3,5,8,13]
for i, e in enumerate (fibonacci):
    if i == 3:
        pass # implement later
             # cannot leave blank
    print('index: ', i, 'number: ', e)
```

In the program, pass is not ignored, but comment(s) are ignored.

Iterations

Loops: Introduction

```
for e in [1, 1, 2, 3, 5, 8, 13]:
    print (e, end =',')
```

```
1 1 2 3 5 8 13 21 34
```

There are two iteration constructs in Python:

a. *for* loop: repeat specified number of times.
b. *while* loop: repeat until some condition is met.

for vs. *while*

In the program below,

- *for* – print all numbers.
- *while* – stops after firrst 5 are printed.

```
fibonacci = [1,1,2,3,5,8,13]
for e in fibonacci:
    print (e, end =',')

print('\n just first 5 fibonacci numbers: ')
count = 0
while count < 5:
    print('\n', fibonacci[count], end =' ')
    count = count + 1
```

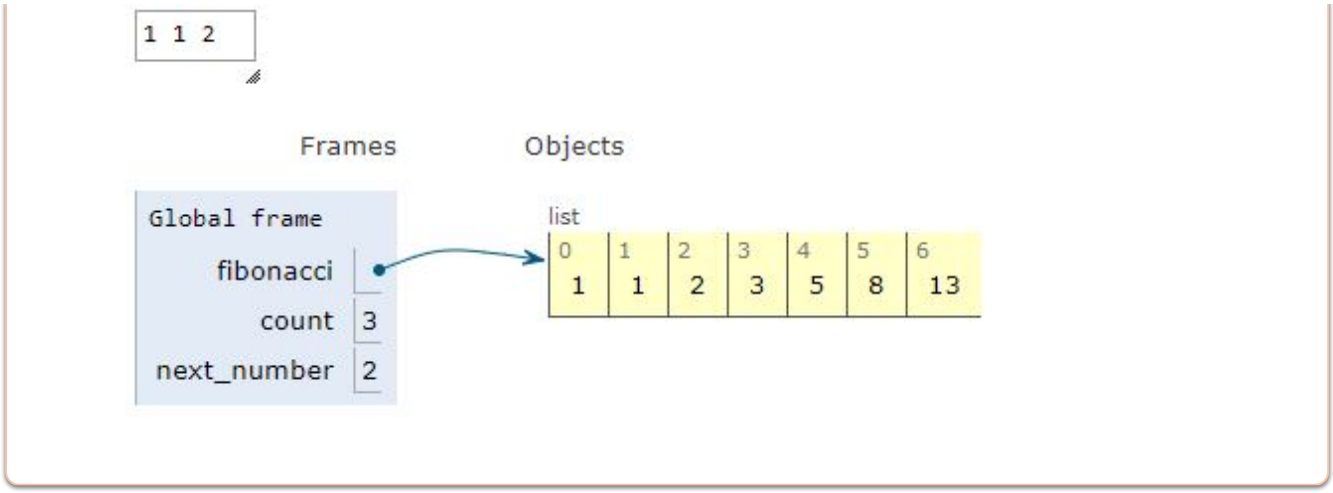
```
1  1  2  3  5  8  13
 just first 5 fibonacci numbers:
1 1 2 3 5
```

Test Yourself 3.4.01

Print the first three Fibonacci numbers.

Suggested program:

```
fibonacci = [1,1,2,3,5,8,13]
counter  = 0
while(counter < 3):
    next_number = fibonacci[count]
    print(next_number, end=' ')
    count = count + 1
```



Python vs. C/C++

C/C++:

- explicit increment and checks of the index.

```
int main() {
int i;
for (i = 1; i <=10; i ++) {
    printf (" %d ", i); }
return 0; }
```

```
1 2 3 4 5 6 7 8 9 10
```

Python:

- value is immediate

```
for e in range (1, 11, 1):
    print (e, end =',')
```

- same syntax in *iterable* objects

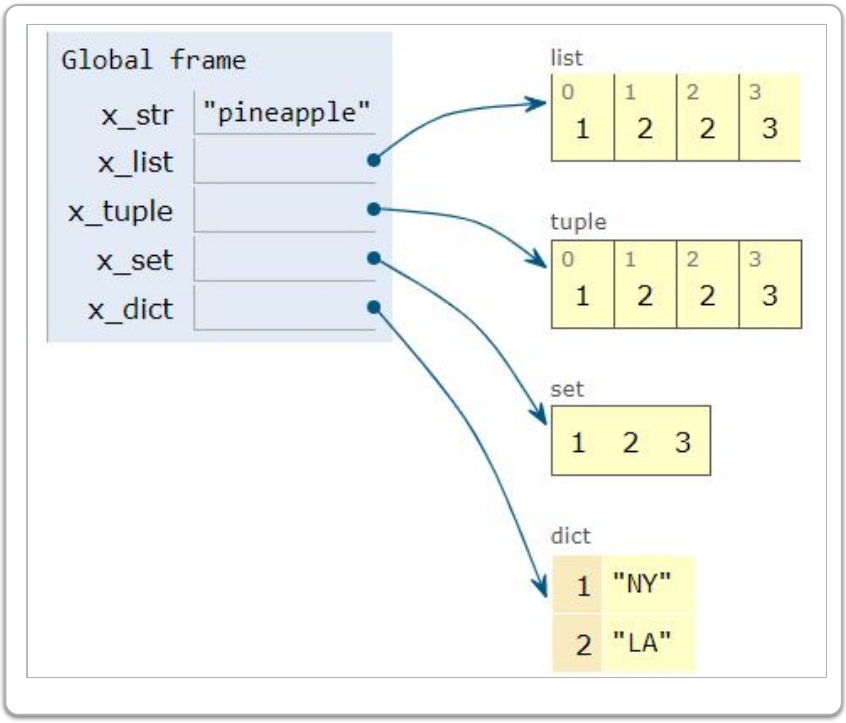
Iterable Objects and *Iterable* Collections

Iterable Objects

- *iterable*: object returns its members one by one.
- *iterator*: object representing a stream of data.
- Python *for* statement:
    a. creates an *iterator*
    b. asks for *next* item
    c. stops when *StopIteration* exception is generated

Iterable Collections

```
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'}
```



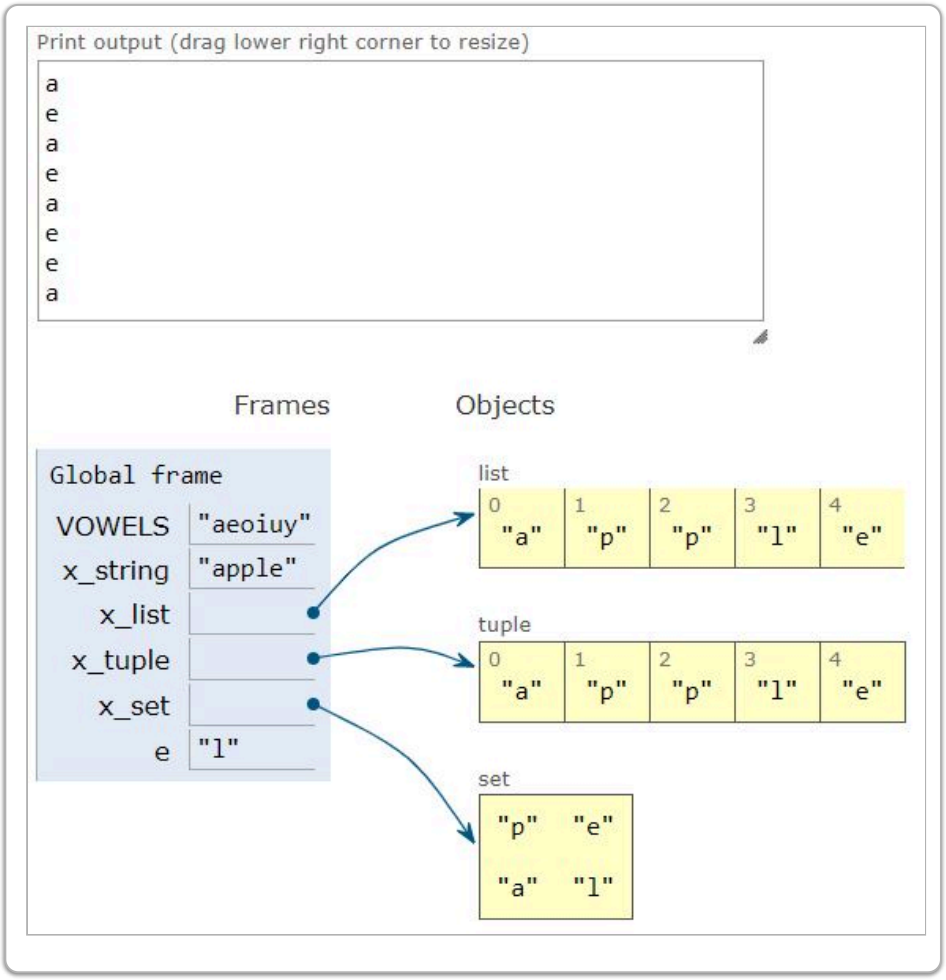
Iterable collections can loop using *for* construct.

```
for element in <collection>:
    statement_1
    -----------
    statement_n
```

for Loop in Collections

```
VOWELS = 'aeoiuy'
x_string = 'apple'
x_list   = ['a','p','p','l','e']
x_tuple  = ('a','p','p','l','e')
x_set    = {'a','p','p','l','e'}

for e in x_string:
    if e in VOWELS:
        print(e, end='')

for e in x_list:
    if e in VOWELS:
        print(e, end='')

for e in x_tuple:
    if e in VOWELS:
        print(e, end='')

for e in x_set:
```

```
if e in VOWELS:
    print(e, end='')
```



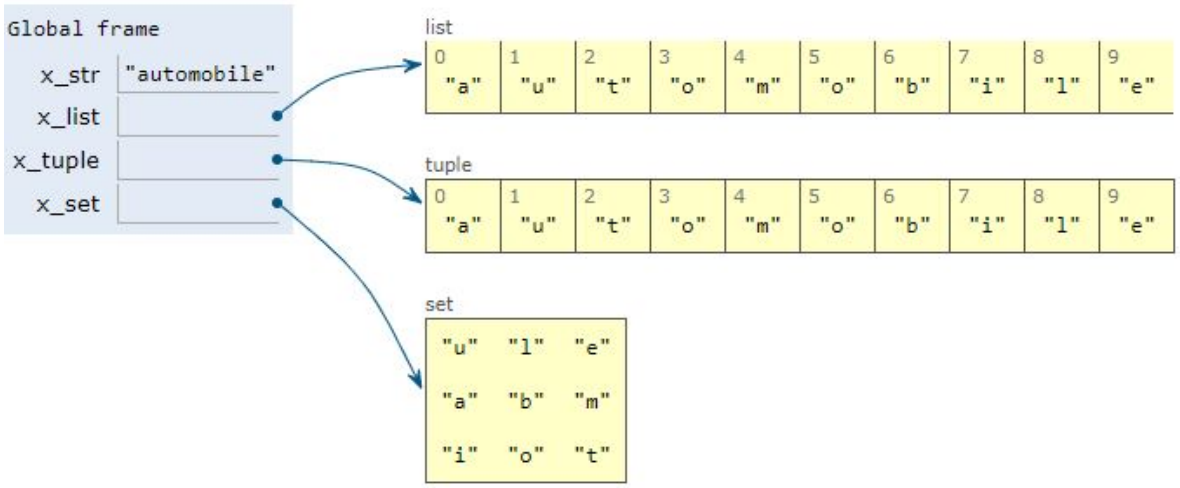
Test Yourself 3.4.02

Construct string, list, tuple, and set collections with characters from *x_str*:

x_str = "automobile"

Suggested program:

```
x_str   = "automobile"
x_list  = list(x_str)
x_tuple = tuple(x_str)
x_set   = set(x_str)
```



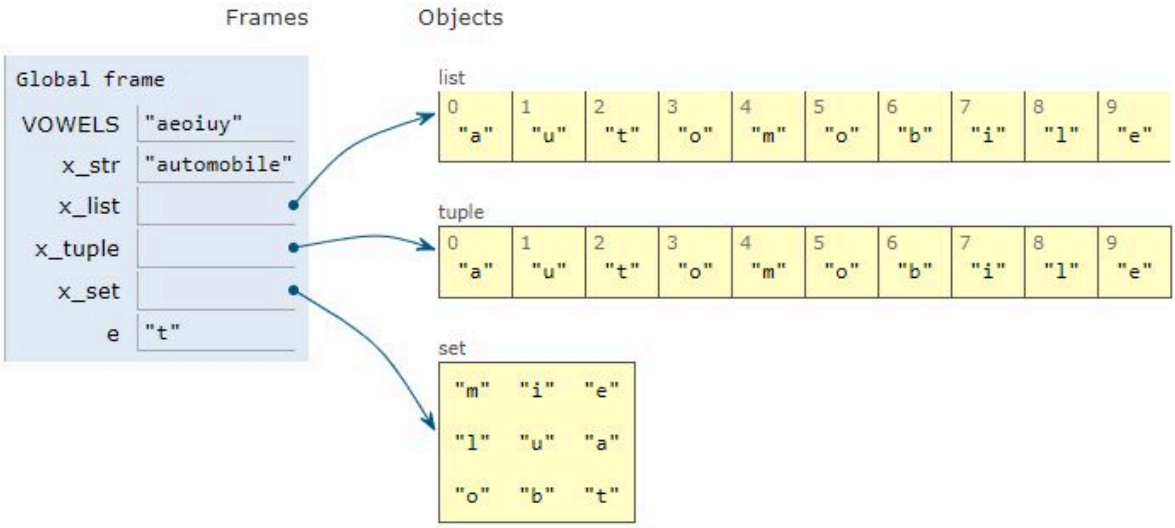
Test Yourself 3.4.03

Based on the last exercise, use *for* loop iteration to print consonants in each collection.

Suggested program:

```
VOWELS = 'aeoiuy'
x_str   = 'automobile'

x_list  = list(x_str)
x_tuple = tuple(x_str)
x_set   = set(x_str)
print(" consonants in string: ", end = " ")
for e in x_str:
    if e not in VOWELS:
        print(e, end='')
print("\n consonants in list: ", end = " ")
for e in x_list:
    if e not in VOWELS:
        print(e, end='')
print("\n consonants in tuple: ", end = " ")
for e in x_tuple:
    if e not in VOWELS:
        print(e, end='')
print("\n consonants in set: ", end = " ")
for e in x_set:
    if e not in VOWELS:
        print(e, end='')
```

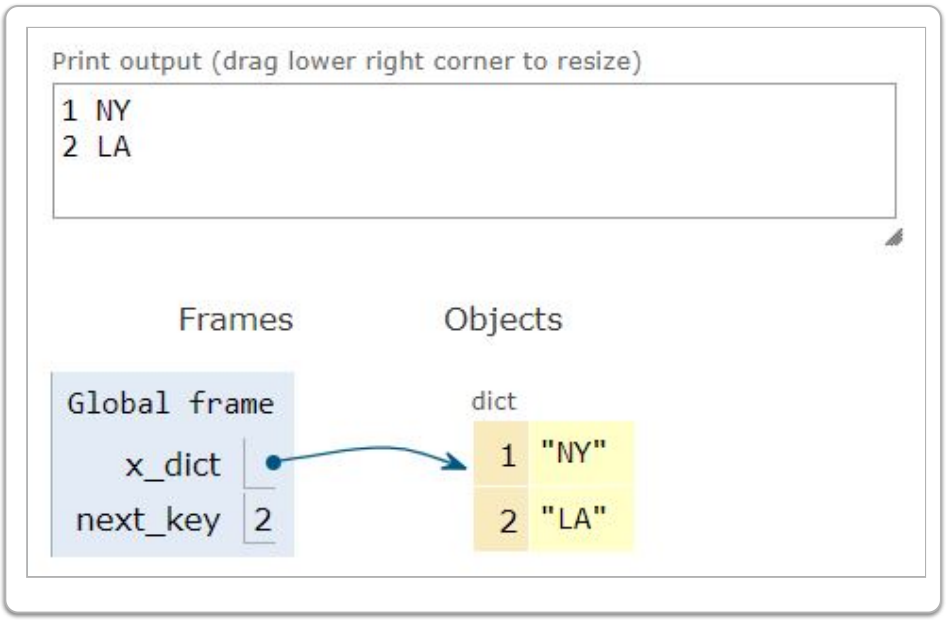
```
consonants in string:  tmbl
consonants in list:  tmbl
consonants in tuple:  tmbl
consonants in set:  mlbt
```



for Loop in Dictionaries

Iteration in dictionaries is different from other collections—iterate by keys.

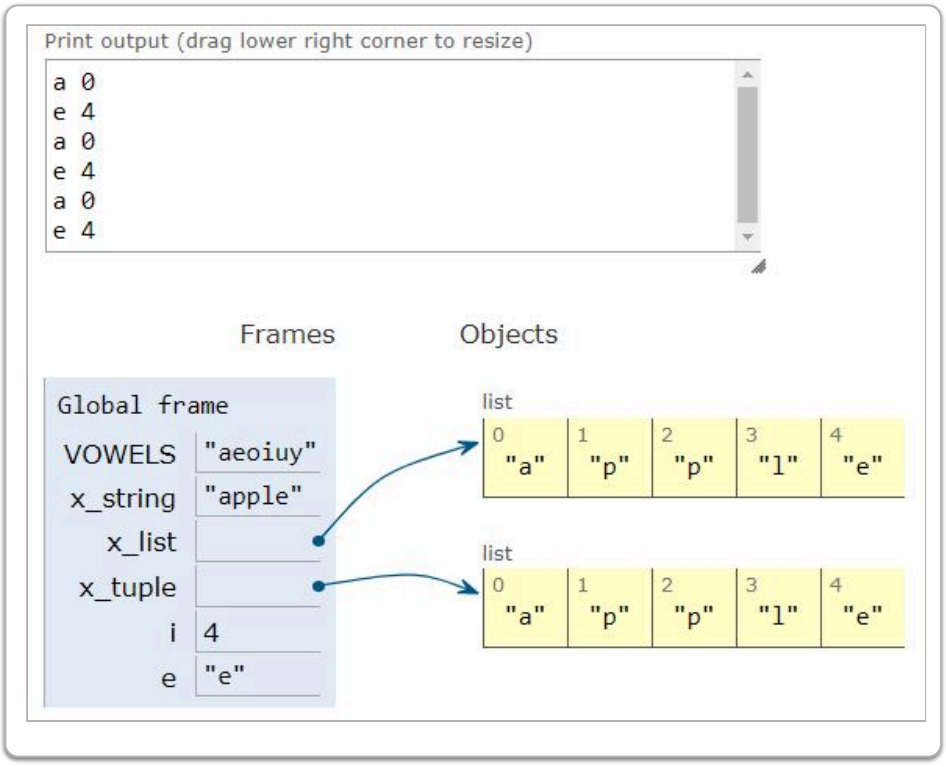
```
# print dictionary keys and values
x_dict = {1: 'NY', 2: 'LA'}

for next_key in x_dict.keys():
    print(next_key, x_dict[next_key])
```



enumerate() in Iterations

- Can use *enumerate()* in ordered collections only: strings, lists, tuples.

```
# print vowels and positions from collections
VOWELS = 'aeoiuy'
x_string = 'apple'
x_list   = ['a','p','p','l','e']
x_tuple  = ('a','p','p','l','e')

for i,e in enumerate(x_string):
    e = x_string[i]
    if e in VOWELS:
        print(e,i)

for i,e in enumerate(x_list):
    e = x_list[i]
    if e in VOWELS:
        print(e,i)

for i,e in enumerate(x_tuple):
    e = x_tuple[i]
    if e in VOWELS:
        print(e,i)
```



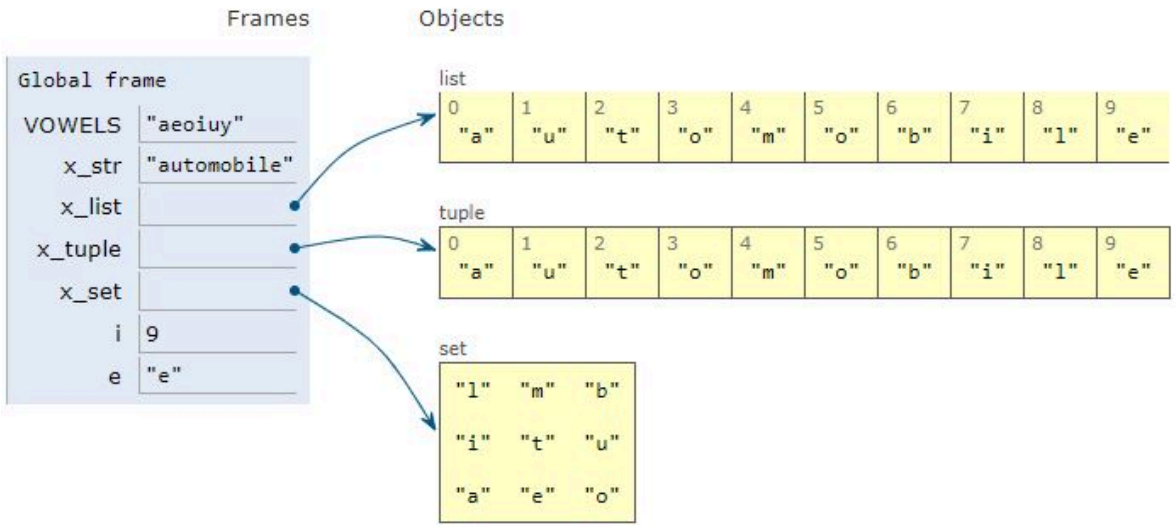
Test Yourself 3.4.04

Based on the last exercise (constructing string, list, tuple, and set collections with characters from *x_str*: x_str = "automobile"), for each collection, use *enumerate*() to print consonants and their positions.

Suggested program:

```
# cannot do it for sets
VOWELS = 'aeoiuy'
x_str   = 'automobile'
x_list  = list(x_str)
x_tuple = tuple(x_str)
x_set   = set(x_str)

print(" (pos, cons.) in string: ", end = " ")
for i, e in enumerate(x_str):
    if e not in VOWELS:
        print((i,e), end=" ")

print(" (pos, cons.) in list: ", end = " ")
for i, e in enumerate(x_list):
    if e not in VOWELS:
        print((i,e), end=" ")

print(" (pos, cons.) in tuple: ", end = " ")
for i, e in enumerate(x_tuple):
    if e not in VOWELS:
        print((i,e), end=" ")
```

```
(pos,cons.) in string:  (2, 't') (4, 'm') (6, 'b') (8, 'l')
(pos, cons) in list:  (2, 't') (4, 'm') (6, 'b') (8, 'l')
(pos, cons.) in tuple:  (2, 't') (4, 'm') (6, 'b') (8, 'l')
```

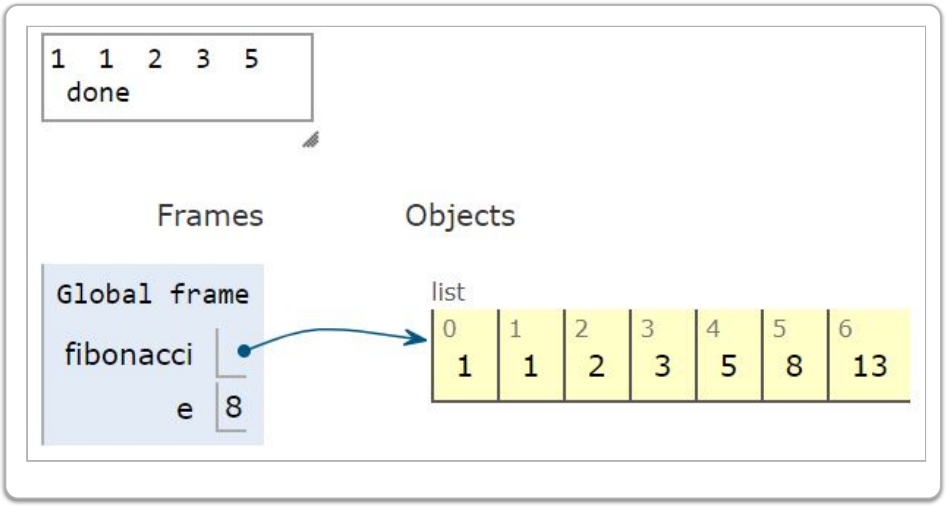


break Statement

break statement terminates the current loop.

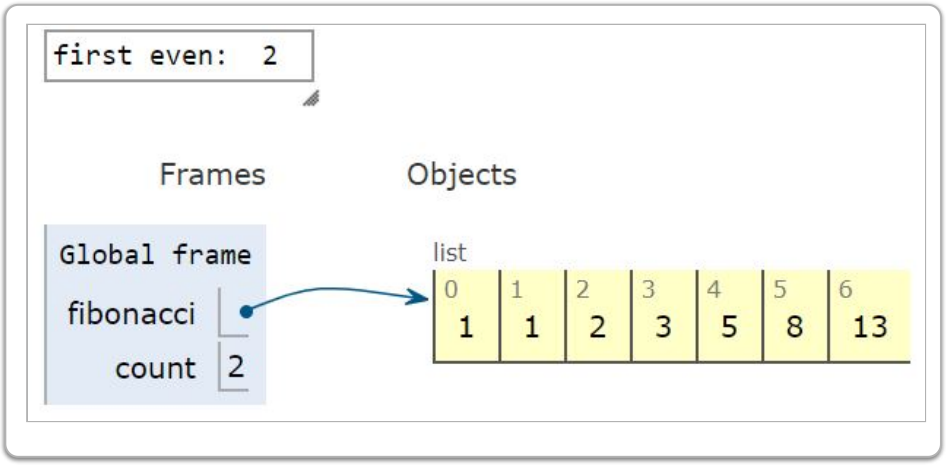
```
# stop as soon as numbers get > 5
fibonacci = [1, 1, 2, 3, 5, 8, 13]
for e in fibonacci:
    if e > 5:
        break
```

```
        print(e, end =' ')
print('\n done')
```



break statement can be used with *while* loop.

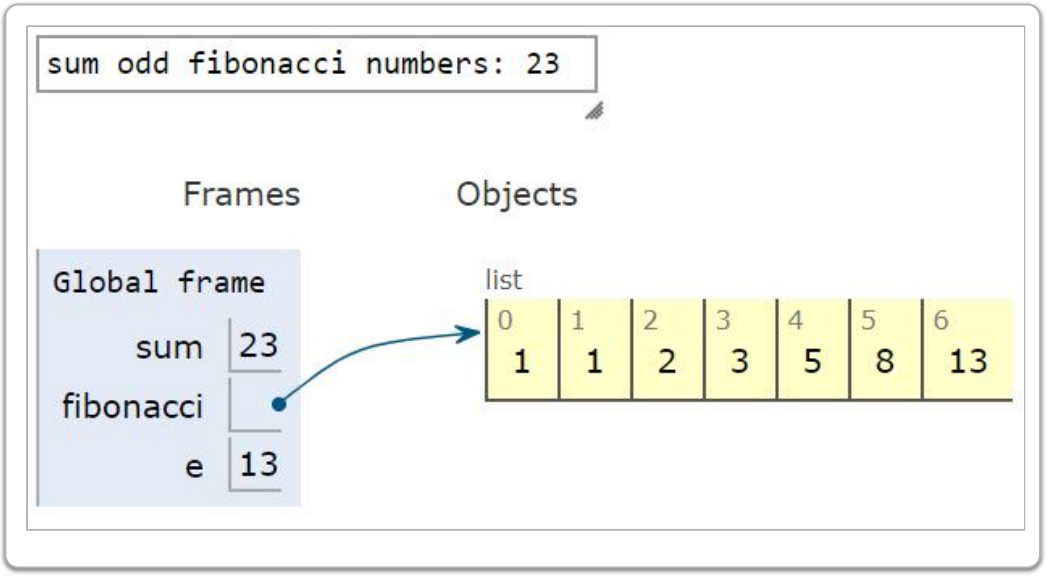
```
# find the first even fibonacci number
fibonacci = [1, 1, 2, 3, 5, 8, 13]
count = 0
while count < 5:
    if fibonacci[count]%2 == 0:
        print('first even: ', fibonacci[count])
        break
    count = count + 1
```



continue Statement

continue statement: go back to the loop.

```
sum = 0
fibonacci  = [1, 1, 2, 3, 5, 8, 13]
for e in fibonacci :
    if e % 2 == 0:
        continue
    sum = sum + e
print('sum odd fibonacci numbers: ', sum)
```



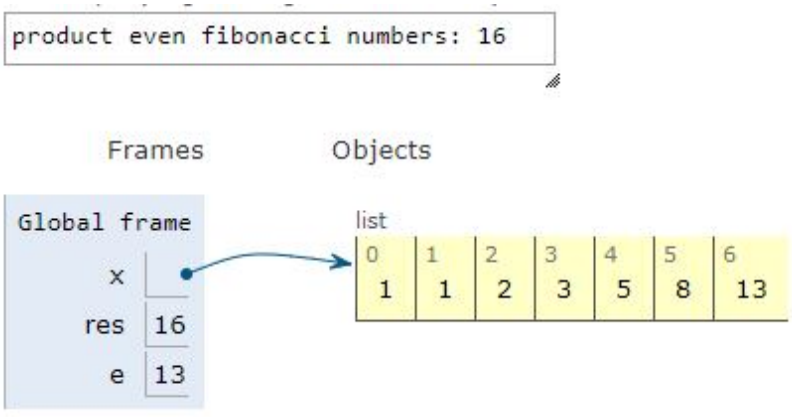
Test Yourself 3.4.05

Use *for* loop to compute the product of even fibonacci numbers from the list:

x = [1, 1, 2, 3, 5, 8, 13]

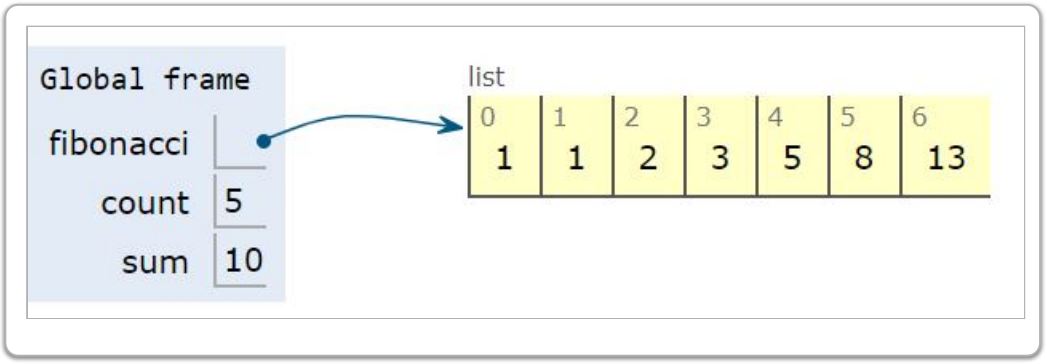
Suggested program:

```
x  = [1, 1, 2, 3, 5, 8, 13]
res = 1
for e in x:
    if e % 2 == 1:
        continue
    res = res * e
print("product even fibonacci numbers:", res)
```



continue statement can be used with *while* loop.

```
fibonacci  = [1, 1, 2, 3, 5, 8, 13]
count = -1; sum = 0
while count < 5:
    count = count + 1
    if fibonacci[count]%2 == 0:
        continue
    sum = sum + fibonacci [count]
print('sum odd fibonacci numbers: ', sum)
```



Test Yourself 3.4.06

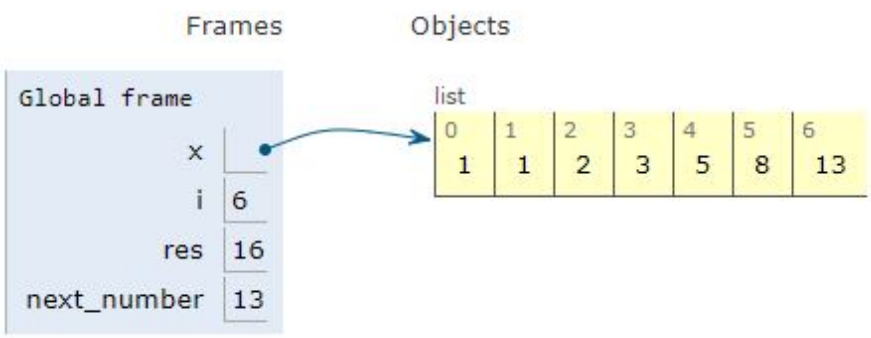
Use *while* loop to compute the product of even fibonacci numbers from the list:

x = [1, 1, 2, 3, 5, 8, 13]

Suggested program:

```
x     = [1, 1, 2, 3, 5, 8, 13]
count = -1
res   = 1
while i < len(x)-1:
    i = i + 1
    next_number = x[i]
    if next_number % 2 == 1:
        continue
    res = res * next_number
print('product even fibonacci numbers:', res)
```

```
product even fibonacci numbers: 16
```



pass Statement

- placeholder for future code

```
fibonacci = [1, 1, 2, 3, 5, 8, 13]
for e in fibonacci:
    pass # code to be added later
print('execution continues')
```

- the following is an error:

```
fibonacci = [1, 1, 2, 3, 5, 8, 13]
for e in fibonacci:

print('execution continues')
```

Iterations: Summary

Iterations Summary

- collections are iterable
- *for* loops iterate over a collections
- indexed collections support *enumerate*()
- *while* loops repeat when certain conditions are True
- *continue* & *break* control loop execution

File and File Manipulation

Files Overview

In this unit, we will do the following:

- Learn methods to open a file for reading or writing.
- Contrast file access modes.
- Differentiate ways to read data.
- Learn Python mechanisms to store and restore objects with files.

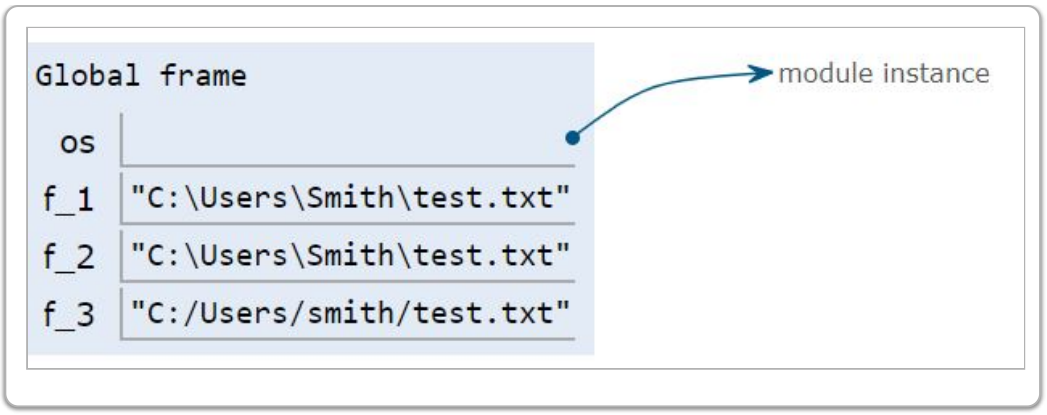
File Operations

- file: location to store data
- file operations have 3 steps:
    1. open a file (to read/write)
    2. read and/or write
    3. close the file
- *open* file command
- returns file handle object

```
file_name = r'C:\Users\Smith\test.txt'
text_file = open(filename, 'r')  # mode is for reading
data = text_file.read()
text_file.close()
```

File Name Specification

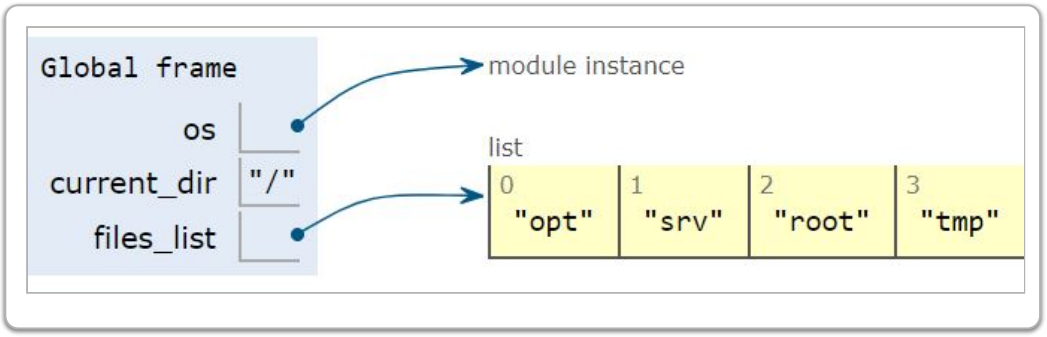
```
import os

# real string
f_1 = r'C:\Users\Smith\test. txt'

# use escape charcaters for Window file name
f_2 = 'C:\\Users\\Smith\\test.txt'

# os independent name
f_3 = os.path.join('C:','Users','smith',
                        'test.txt')
```



List Current Directory

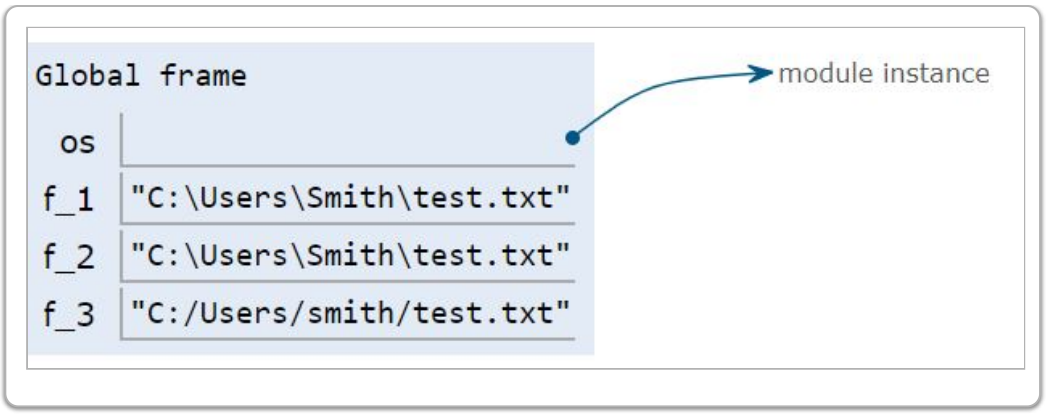
- can create/delete directories
- can create, modify, delete files

```
import os
current_dir = os.getcwd()
files_list = os.listdir(current_dir)
```



File Access Modes

```
import os

file_name = os.path.join('C:','Users','smith',
                         'test.txt')
text_file = open(file_name, 'r')
```



Mode	Meaning
a	Open for writing. Creates the file if it does not exist. The stream is positioned at the end of the file Subsequent writes to the file will always end up at the then current end of file.
a+	Open for reading and writing. Creates the file if it does not exist. The stream is positioned at the end of the file. Subsequent writes to the file will always end up at the then current end of file.
r	Open for reading (default). An error occurs if it does not exist. The stream is positioned at the beginning of the file.
r+	Open for reading and writing. An error occurs if it does not exist. The stream is positioned at the beginning of the file.
w	Open for writing. Creates the file if it does not exist. Otherwise, it is truncated to zero length. The stream is positioned at the beginning of the file.
w+	Open for reading and writing. Creates the file if it does not exist. Otherwise, it is truncated to zero length. The stream is positioned at the beginning of the file.
x	Open for exclusive creation. If the file already exists, the operation fails.

Different Ways to Read Data

Reading Whole Text File Into a String

```
import os

dir_name  = os.getcwd()
file_name = os.path.join(dir_name, 'test.txt')
```

```
text_file = open(file_name, 'r')
result = text_file.read()
text_file.close()

print(result)
```

```
Monday
Tuesday
Wednesday
```

The program above shows an example to read a whole text file into one string.

Read Characters From a File

You can also read next *n* characters from a file.

```
import os

dir_name  = os.getcwd()
file_name = os.path.join(dir_name, 'test.txt')

text_file = open(file_name, 'r')
result = text_file.read(10)
text_file.close()

print(result)
```

```
Monday
Tue
```

Read Characters From a Line

Another method is to read next *n* characters till newline from a file.

```
import os

dir_name  = os.getcwd()
file_name = os.path.join(dir_name, 'test.txt')

text_file = open(file_name, 'r')
result = text_file.readline(10)
text_file.close()

print(result)
```

```
Monday
```

Read Next Line

The following example shows another method that reads one line at a time from a file.

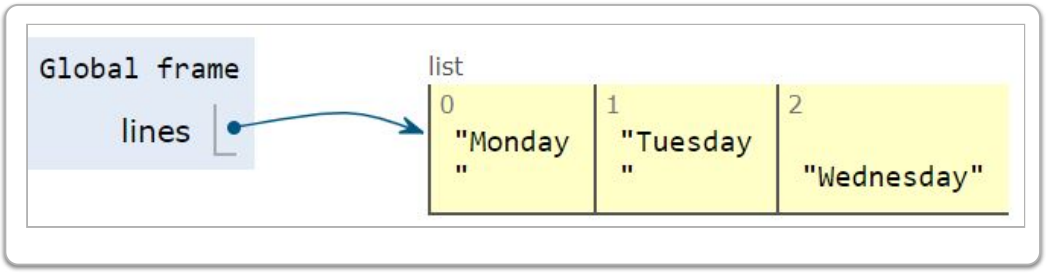
```
import os

dir_name  = os.getcwd()
file_name = os.path.join(dir_name, 'test.txt')

text_file = open(file_name, 'r')
print('next line: ', text_file. readline())
print('next line: ', text_file. readline())
text_file.close()

print(result)
```

```
next line:   Monday

next line:   Tuesday
```

Read Lines Into List

The following example shows another method that reads a file into a list of lines.

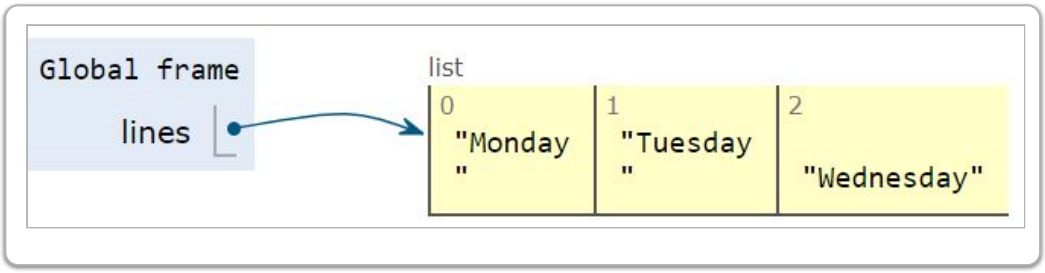
```
import os

dir_name  = os.getcwd()
file_name = os.path.join(dir_name, 'test.txt')

text_file = open(file_name, 'r')
lines     = text_file.readlines ()
text_file.close()

print(result)
```



Using Content Manager

Using content manager, the file is automatically closed.

```
import os

dir_name  = os.getcwd()
file_name = os.path.join(dir_name, 'test.txt')
with open(file_name, 'r') as text_file:
    lines = text_file.readlines()

print(lines)
```

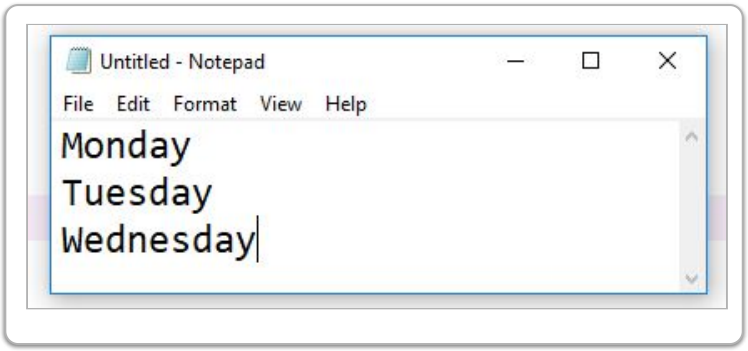


Store and Restore Objects with Files

Writing Strings to File

Strings can be written to a file with individual lines.

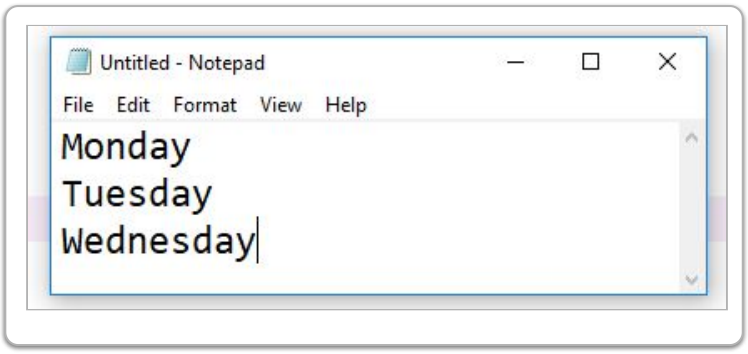
```
import os

dir_name  = os.getcwd()
file_name = os.path.join(dir_name, 'test.txt')
text_file = open(file_name, 'w')
text_file.write('Monday\n')
text_file.write('Tuesday\n')
text_file.write('Wednesday')
text_file.close()
```



Writing List of Strings to File

List of strings can be written to a file with individual lines.

```
import os

dir_name  = os.getcwd()
file_name = os.path.join(dir_name, 'test.txt')

lines = ['Monday\n', 'Tuesday\n', 'Wednesday']
text_file = open(file_name, 'w')
text_file.write("".join(lines))
text_file.close()
```



Save/Restore Objects

```
import os
import pickle

dir_name  = os.getcwd()
file_name = os.path.join(dir_name, 'objects.txt')
x_dict = {1: 'NY', 2: 'LA'}

dump_file = open(file_name, 'wb')
pickle.dump(x_dict, dump_file)
dump_file.close()
----------
----------
dump_file = open(file_name, 'rb')
y_dict    = pickle.load(dump_file)
dump_file.close()
```

The above program shows examples of saving/restoring objects:

- use module pickle
- not identical objects

Files: Test Yourself Exercises

The following are some exercise questions for you to practice. Please read each question, think carefully, click "Show hint" to review and relearn what you have learned, figure out your own answer or write your own program first, and then click "Show Answer" to compare yours to the suggested answer or the possible solution.

Test Yourself 3.5.01

Count number of vowels in a text file.

Suggested program:

```
import os
VOWELS    = "aeoiuy"
dir_name  = os.getcwd()
file_name = os.path.join(dir_name, "test.txt")
text_file = open(file_name, "r")
result    = text_file.read()
count     = 0

for e in result:
    if e in VOWELS:
        count = count + 1

text_file.close()
print("number of vowels: ", count)
```

Test Yourself 3.5.02

Display (complete) text file in reverse order.

Suggested program:

```
import os
dir_name  = os.getcwd()
file_name = os.path.join(dir_name, "test.txt")

text_file = open(file_name, "r")
result    = text_file.read()
print(result[:: -1])
text_file.close()
```

Test Yourself 3.5.03

Write lines from file A to file B in reverse.

Suggested program:

```
import os
dir_name  = os.getcwd()
file_A    = os.path.join(dir_name, "test.txt")
file_B    = os.path.join(dir_name, "test_r.txt")

input_file  = open(file_A, "r")
output_file = open(file_B, "w")
result      = input_file.read().splitlines()
```

```
result_rev  = result[ : : -1]
new_text    = "\n".join(result_rev)
output_file.write(new_text)

input_file.close()
output_file.close()
```

Test Yourself 3.5.04

What are some of the file related modules in Python.

Suggested program:

1. os

2. pickle (save/restore objects)

3. shutil: high-level file operations (e.g. *copyfile*())

Test Yourself 3.5.05

What is the use of *with*?

Suggested program: ensures that a resource is "cleaned" after execution.

Test Yourself 3.5.06

Print length of each line in a file.

Suggested program:

```
dir_name = os.getcwd()
file_A   = os.path.join(dir_name, "test.txt")

input_file = open(file_A, "r")
result     = input_file.read().splitlines()
for i, e in enumerate(result):
    print("line: ", i, " length: ", len(e))
input_file.close()
```

Files: Summary

Files Summary

- flexible capability for files

- multiple file modes
- can process characters or lines
- save/restore objects as files
- additional modules to handle specialized files

Lists

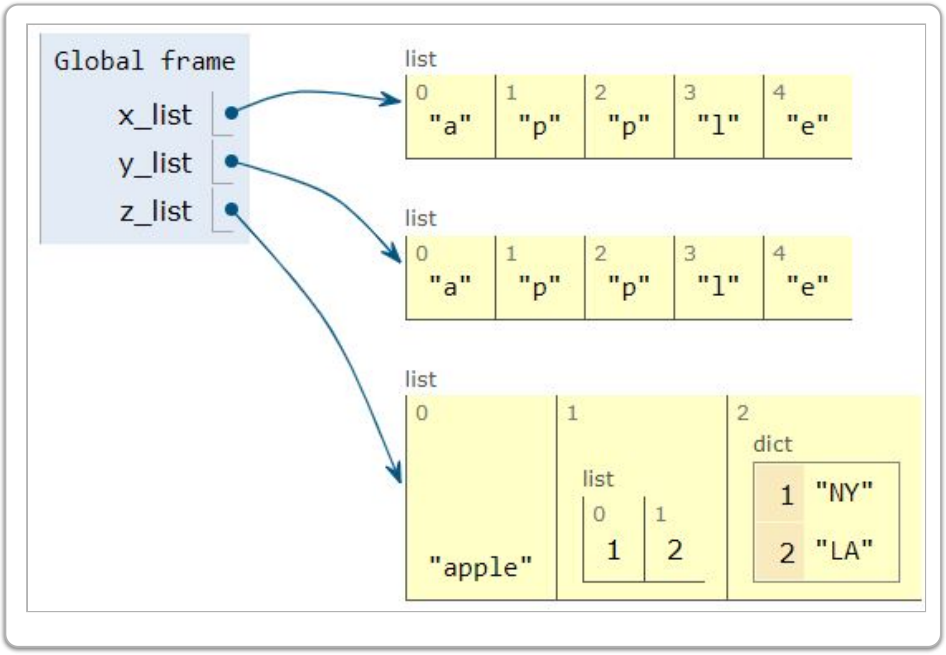
Python Lists Overview

A Python List

Let's review what we have learned about Python list in the module earlier.

- A Python list is an ordered collection.
- A list can contain any objects.

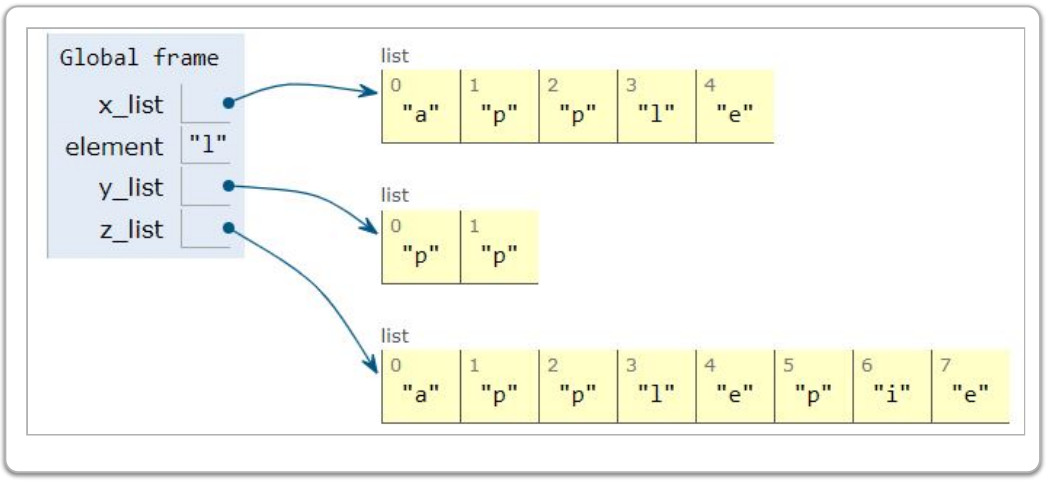
```
x_list = ['a','p','p','l','e']
y_list = list('apple')
z_list = ['apple', [1,2] , {1: 'NY', 2: 'LA'}]
```



Examples Methods

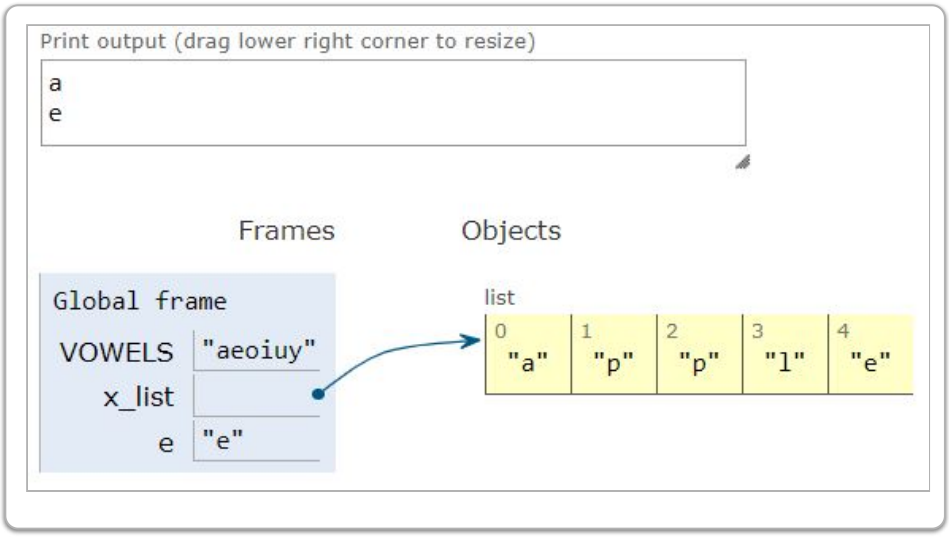
```
x_list  = ['a','p','p','l','e']
element = x_list[3]        # indexing
```

```
y_list  = x_list[1 : 3]    # slicing

z_list  = ['a','p','p','l','e']
z_list.extend(['p','i','e'])
```



Membership & Iteration

```
VOWELS = 'aeoiuy'
x_list  = ['a','p','p','l','e']

for e in x_list:
    if e in VOWELS:
        print(e)
```



Test Yourself 3.6.01

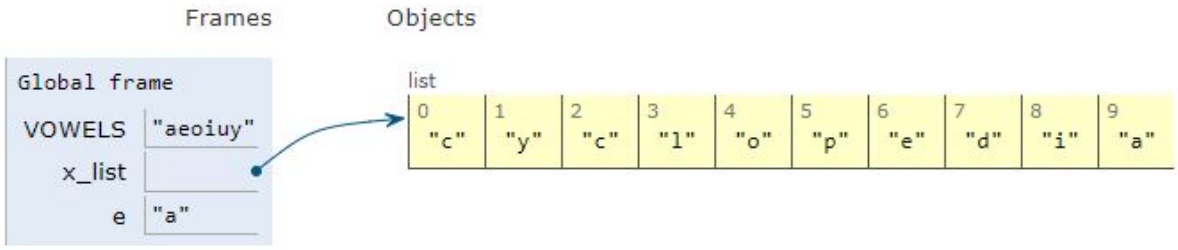
Print consonants in *x_list*:

x_list=list("cyclopedia")

Suggested program:

```
VOWELS = 'aeoiuy'
x_list  = list("cyclopedia")

for e in x_list:
    if e not in VOWELS:
        print(e, end = " ")
```

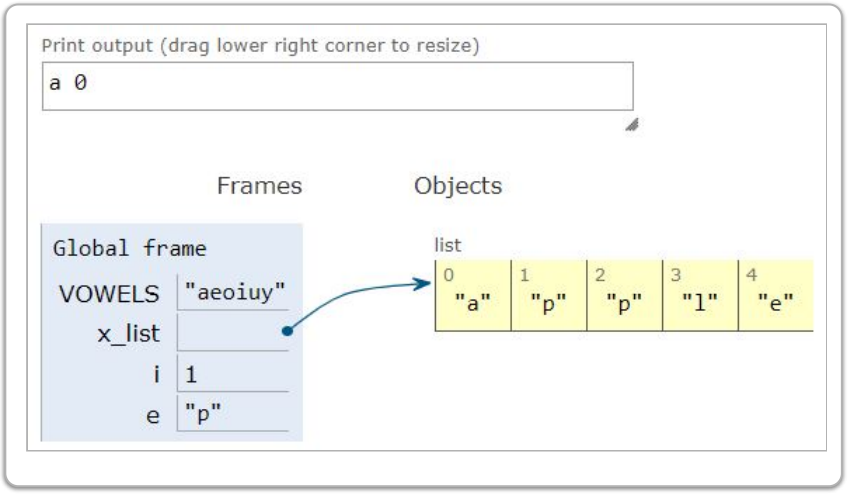
```
c c l p d
```

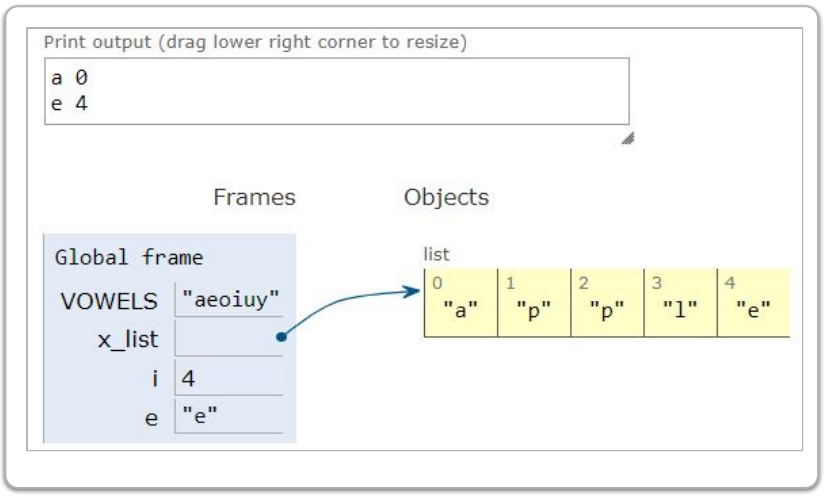


Iteration: *enumerate()*

- Can use *enumerate()* in strings, lists, and tuples.
- Using *enumerate()*, can get both index and element.

```
# print vowels and positions from list
VOWELS = 'aeoiuy'
x_list = ['a','p','p','l','e']

for i,e in enumerate(x_list):
    e = x_list[i]
    if e in VOWELS:
        print(e,i)
```





Test Yourself 3.6.02

Print consonants and their positions in *x_list*:

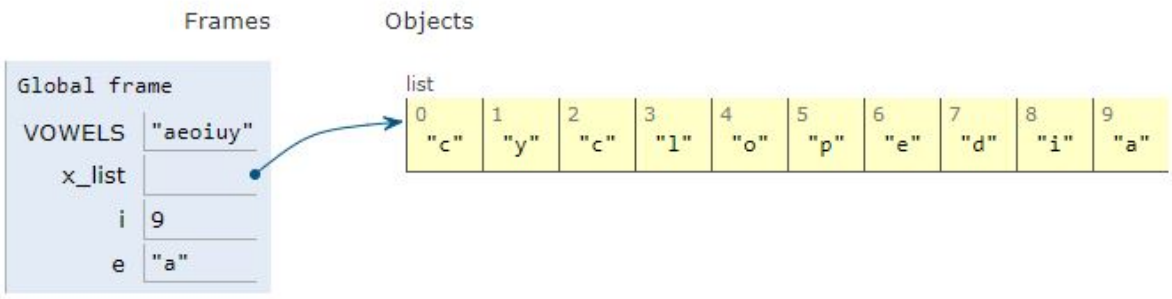
x_list=list("cyclopedia")

Then write code without using enumerate()

Suggested program:

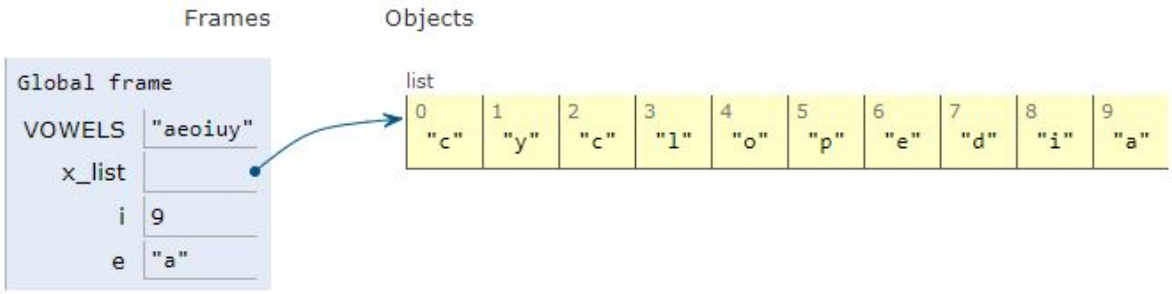
```
VOWELS = 'aeoiuy'
x_list  = list("cyclopedia")

for i,e in enumerate(x_list):
    if e not in VOWELS:
        print((e,i), end = " ")
```

```
('c', 0) ('c', 2) ('l', 3) ('p', 5) ('d', 7)
```




```
#code without using enumerate()
VOWELS = 'aeoiuy'
x_list  = list("cyclopedia")

for i in range(len(x_list)):
    e = x_list[i]
    if e not in VOWELS:
        print((e,i), end = " ")
```

```
('c', 0) ('c', 2) ('l', 3) ('p', 5) ('d', 7)
```



Lists Functions Part I

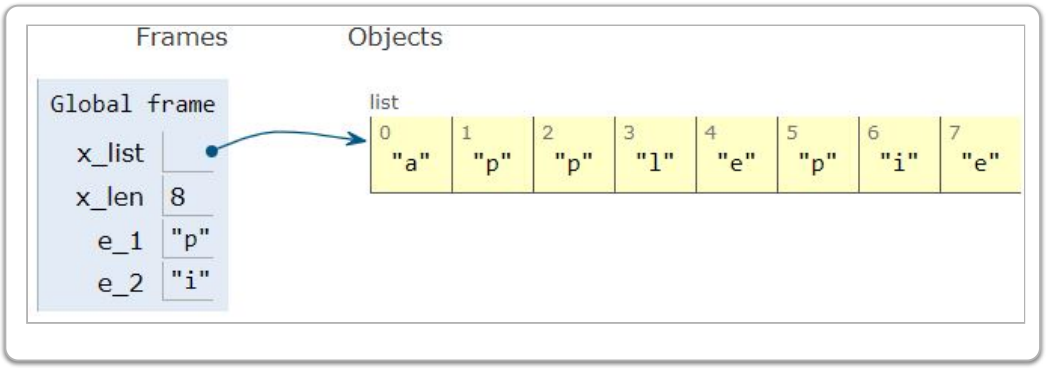
List Indexing

Lists work similarly to strings, the index() method finds the given element in a list and returns its position. If the same element is present more than once, the method returns the index of the first occurrence of the element. Again, there are positive and negative indices.

- Positive index: Python indexing starts at 0, not 1. The first element in the list has index 0, the next is 1, and so on.
- Negative index: Python also allows you to index from the end of the list using a negative number, where [-1] returns the last element, [-2] returns the next to last, and so on.

- Positive = negative + length

```
x_list = ['a','p','p','l','e','p','i','e']
x_len = len(x_list)
e_1   = x_list[1]
e_2   = x_list[-2]
```



0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
-8	-7	-6	-5	-4	-3	-2	-1

Test Yourself 3.6.03

Compute elements from *x_list*:

x_list=list("cyclopedia")

(a) a = x_list[0]

(b) b = x_list[1]

(c) c = x_list[-1]

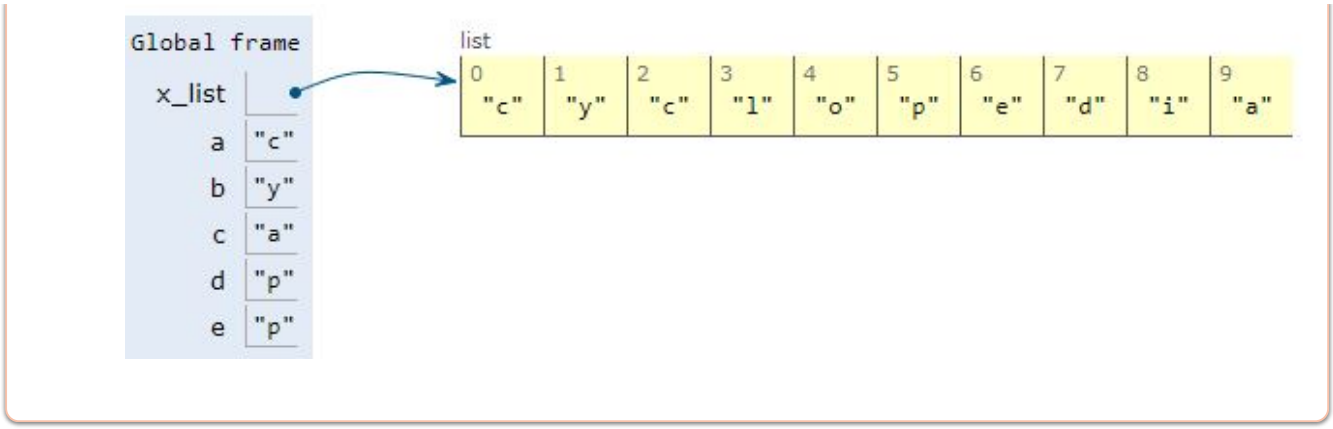
(d) d = x_list[5]

(e) e = x_list[-5]

(f) f = x_list[25]

Suggested program:

```
x_list = list("cyclopedia")
a = x_list[0]
b = x_list[1]
c = x_list[-1]
d = x_list[5]
e = x_list[-5]
f = x_list[25]     # erorr
```



List Slicing

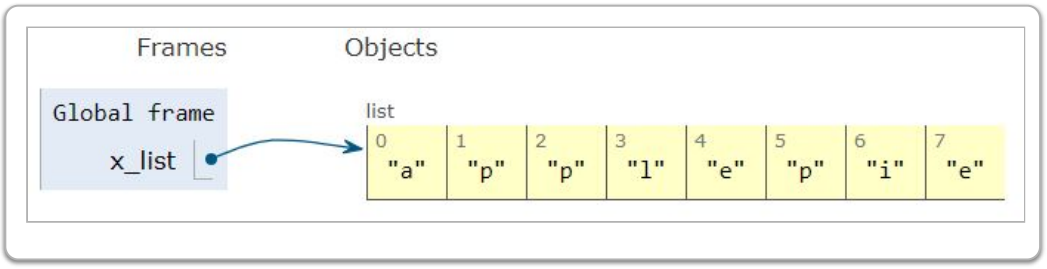
In order to access a range of elements in a Python list, you need to slice a list, by identifying a range of index numbers [start : end + 1 : step]:

- The first index number is where the slice starts (inclusive).
- The second index number is where the slice ends (exclusive).
- Step number: the number of characters to skip between two index numbers of a slice.
  - If the step number is 1, will take every element between two index numbers of a slice.
  - If the step number is 2, skip every other element between two index numbers.
  - If the step number is omitted, Python will default with 1.

We can use both positive and negative indices. Negative step is for reversals.

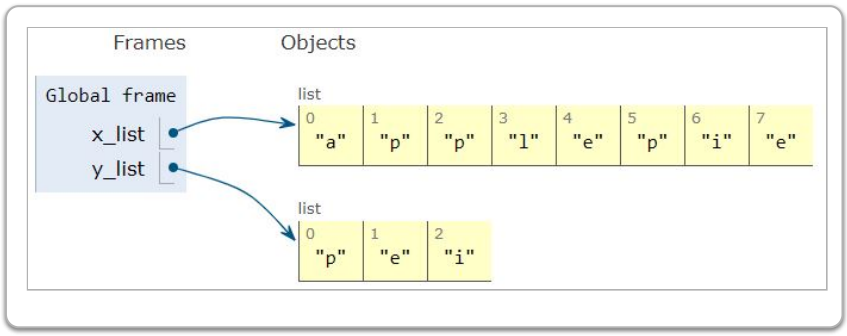
Use the list "applepie" as an example.

```
x_list = ['a','p','p','l','e','p','i','e']
```



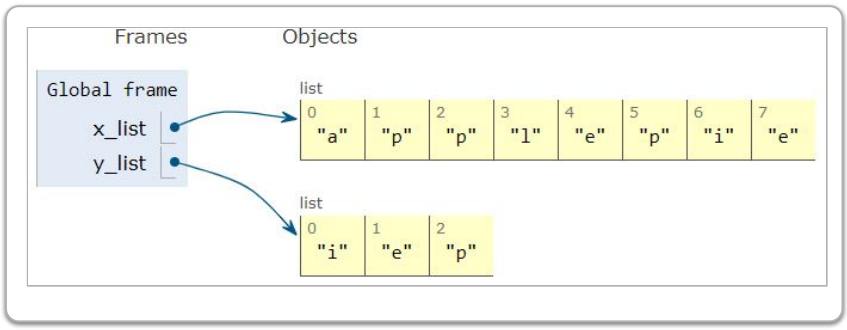
0	1	2	3	4	5	6	7
a	**p**	**p**	**l**	**e**	**p**	**i**	**e**
-8	-7	-6	-5	-4	-3	-2	-1

```
x_list = ['a','p','p','l','e','p','i','e']

y_list= x_list[ 2 :  7 : 2]
y_list= x_list[-6 : -1 : 2]
y_list= x_list[ 2 : -1 : 2]
y_list= x_list[-6 :  7 : 2]
```



0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
-8	-7	-6	-5	-4	-3	-2	-1

```
x_list = ['a','p','p','l','e','p','i','e']

y_list = x_list [ 6 :  1 : -2]   # in reverse
y_list = x_list [-2 : -7 : -2]
y_list = x_list [ 6 : -7 : -2]
y_list = x_list [-2 :  1 : -2]
```



0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
-8	-7	-6	-5	-4	-3	-2	-1

Test Yourself 3.6.04

Compute slices from *x_list*:

x_list=list("cyclopedia")

(a) a = x list[0 : 10 : 2]

(b) b = x_list[1 : 9 : 3]

(c) c = x_list[-10 : -2 : 3]

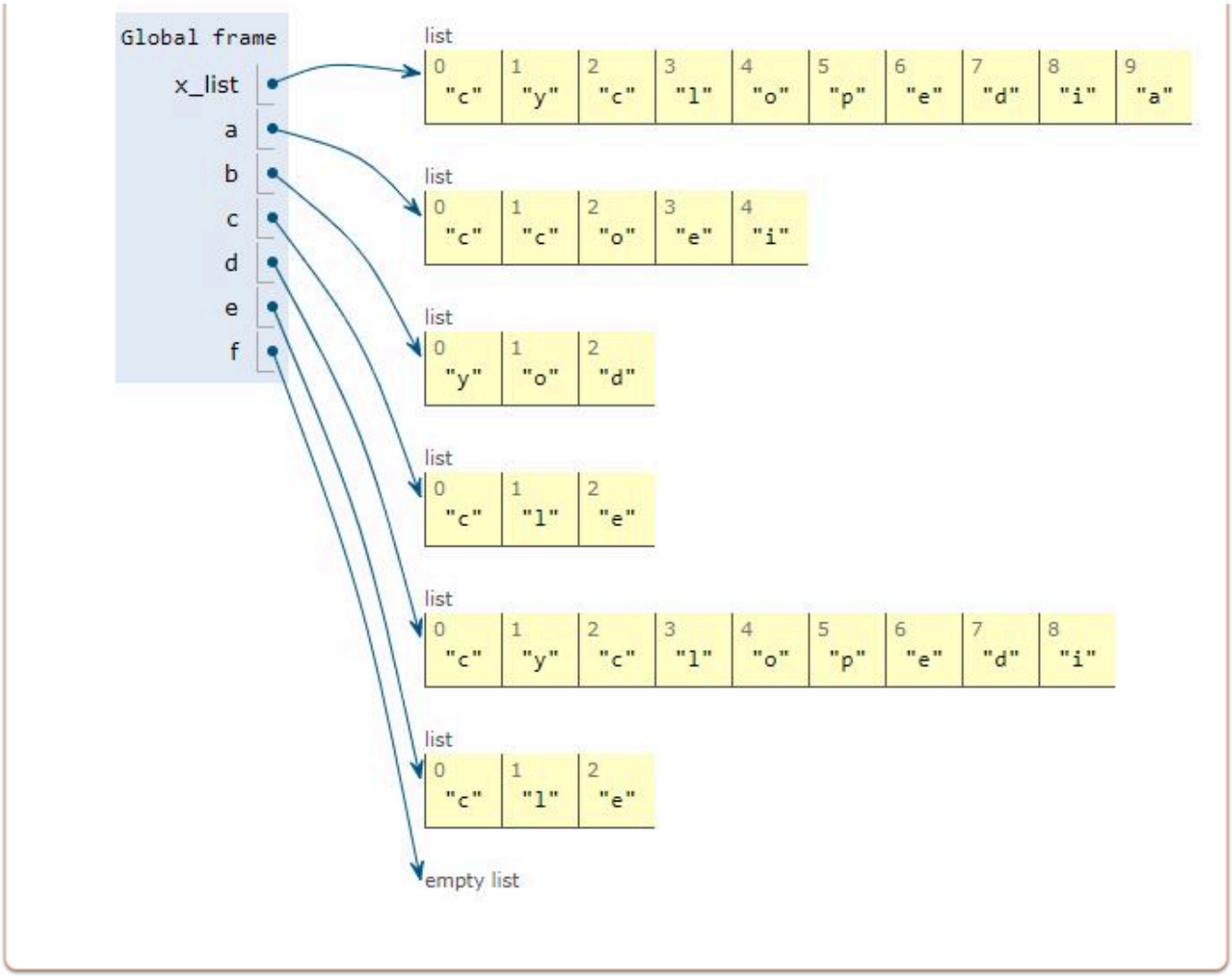
(d) d = x_list[0 : -1 : 1]

(e) e = x_list[0 : -2 : 3]

(f) f = x_list[30 : 5 : 5]

Suggested program:

```
x_list = list("cyclopedia")
a = x_list[0   : 10 : 2]
b = x_list[1   : 9  : 3]
c = x_list[-10 : -2 : 3]
d = x_list[0   : -1 : 1]
e = x_list[0   : -2 : 3]
f = x_list[30  : 5  : 5]
```

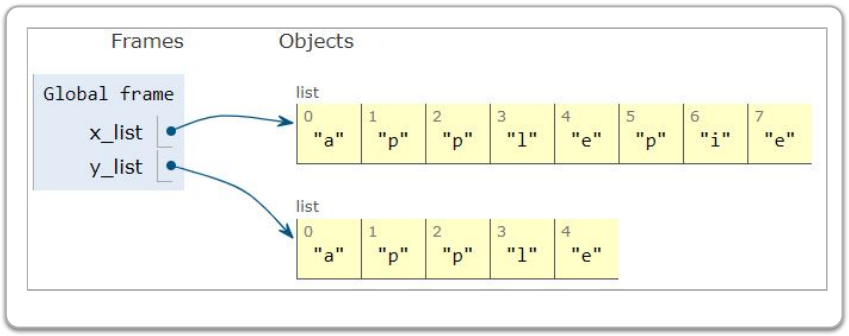


Slicing with Default

[start : end + 1 : step]

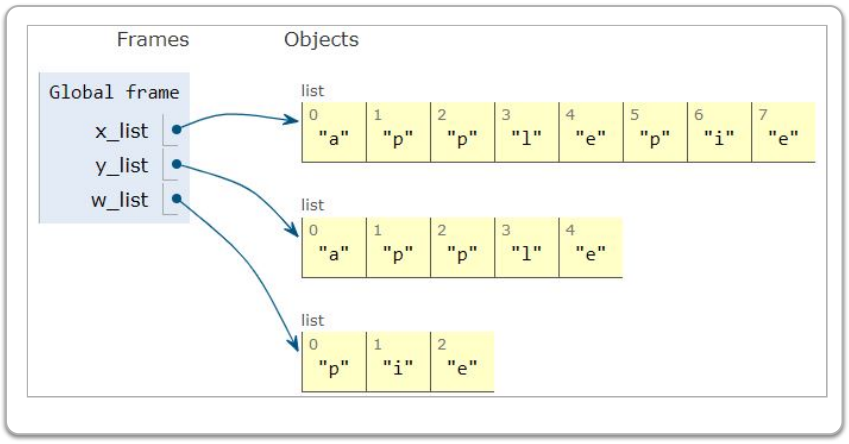
- when the first index number is omitted, Python will default with 0, the beginning of the list.
- When the the index number after the colon is omitted, Python will default to the last index, the end of the list.
- If the step number is omitted, Python will default with 1.

```
x_list = ['a','p','p','l','e','p','i','e']

y_list = x_list[0 : 5 : 1]
y_list = x_list[  : 5 : 1]  # assume defaults
y_list = x_list[  : 5]
```



0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
-8	-7	-6	-5	-4	-3	-2	-1

```
x_list = ['a','p','p','l','e','p','i','e']
y_list = x_list[  : 5]
w_list = x_list[5 :  ]
```

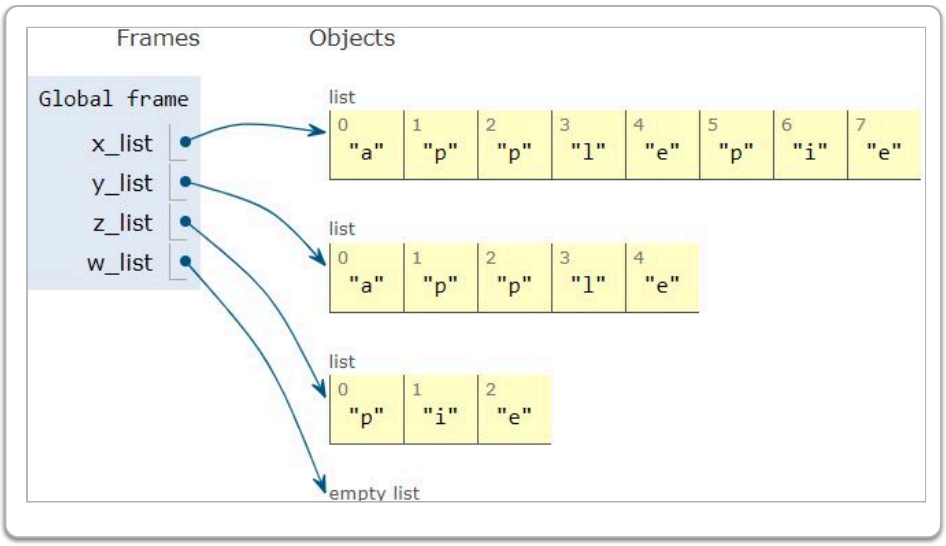


0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
-8	-7	-6	-5	-4	-3	-2	-1

"Out-of-Bound" Slicing

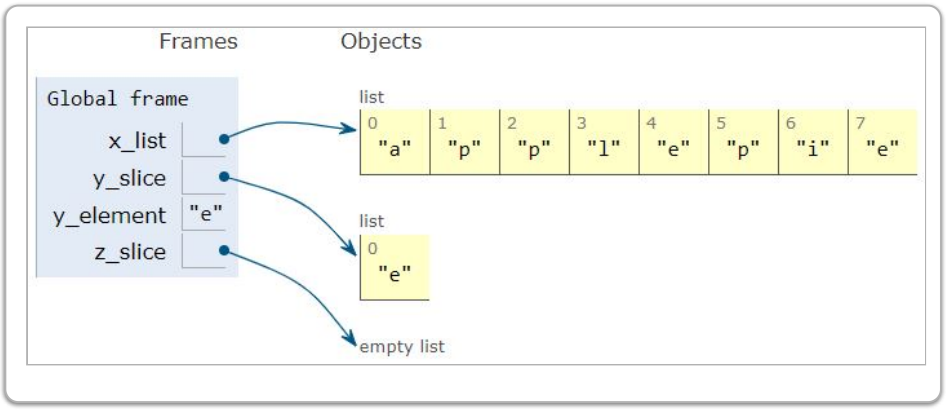
Similar to Python string slicing, Python list slicing handles out of range indexes gracefully—get the "largest" sub-list and do not produce error.

```
x_list = ['a','p','p','l','e','p','i','e']
y_list = x_list[-100 : 5]
z_list = x_list[5 : 500 ]
w_list = x_list[400 : 500]
```



Slicing vs. Indexing

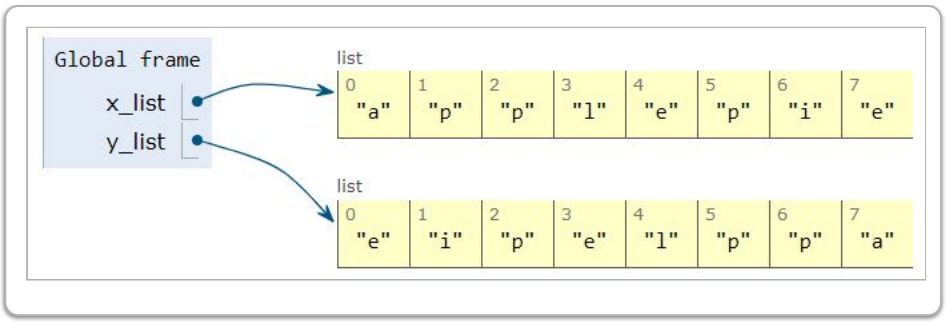
```
x_list = ['a','p','p','l','e','p','i','e']
y_slice   = x_list[4:5]
y_element = x_list[4]
z_slice   = x_list[100:101]
z_element = x_list[100]     # error
```



0	1	2	3	4	5	6	7
a	p	p	l	e	p	i	e
-8	-7	-6	-5	-4	-3	-2	-1

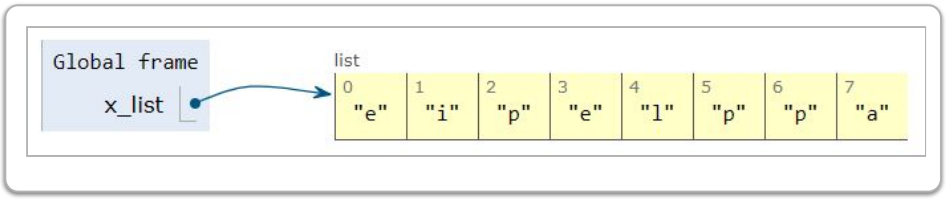
List Reversal

```
x_list = ['a','p','p','l','e','p','i','e']
y_list = x_list[ : : -1]     # new object
```



Compare with *reverse*():

```
x_list.reverse()    # not a new object
```



Test Yourself 3.6.05

Compute slices from *x_list*:

x_list=list("cyclopedia")

(a) a = x_list[10 : 0 : -1]

(b) b = x_list[10 : : -2]

(c) c = x_list[: : -2]

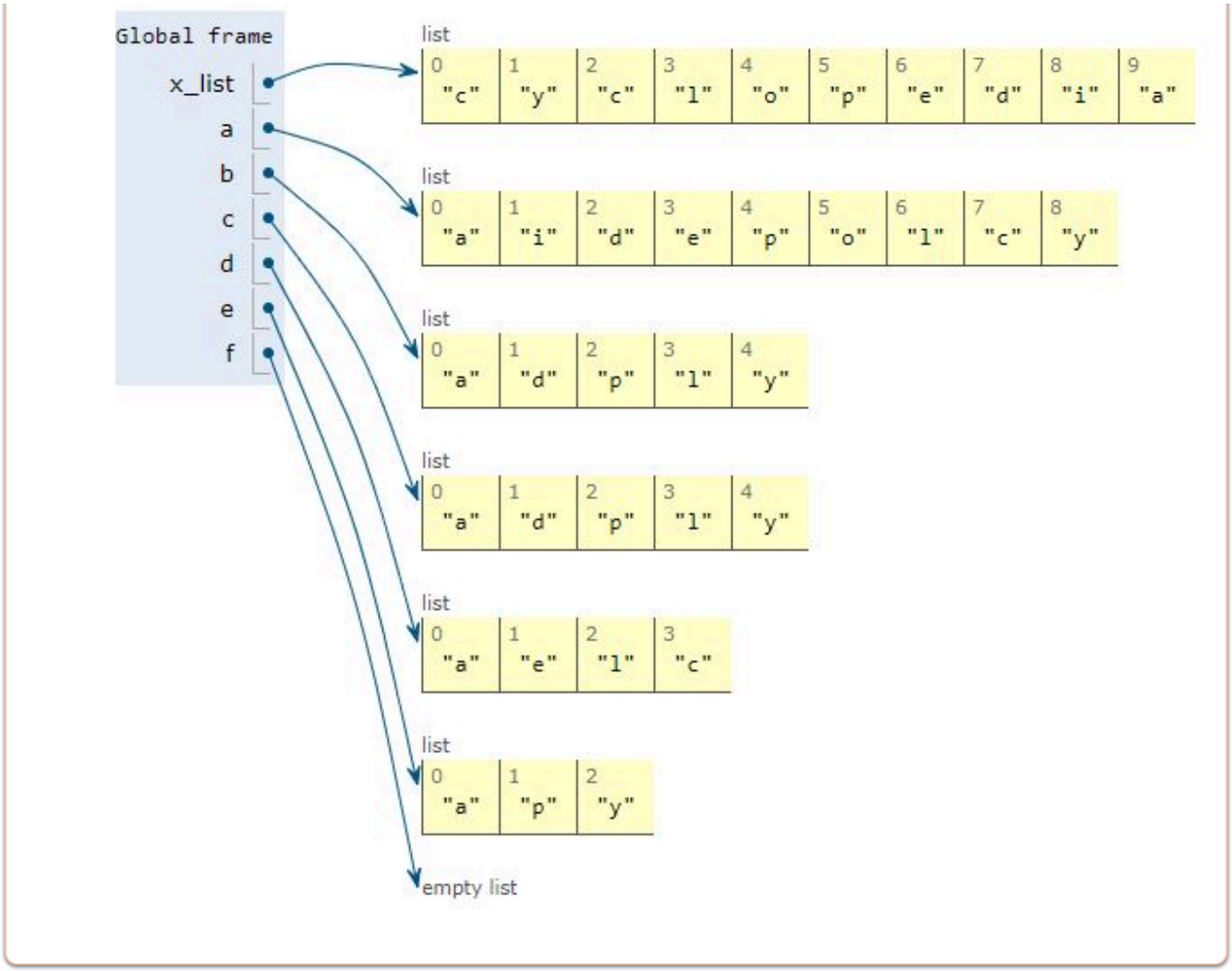
(d) d = x_list[: : -3]]

(e) e = x_list[: : -4]

(f) f = x list[0 : -1 : -1]

Suggested program:

```
x_list = list("cyclopedia")
a = x_list[10 : 0  : -1]
b = x_list[10 :    : -2]
c = x_list[   :    : -2]
d = x_list[   :    : -3]
e = x_list[   :    : -4]
f = x_list[0  : -1 : -1]
```

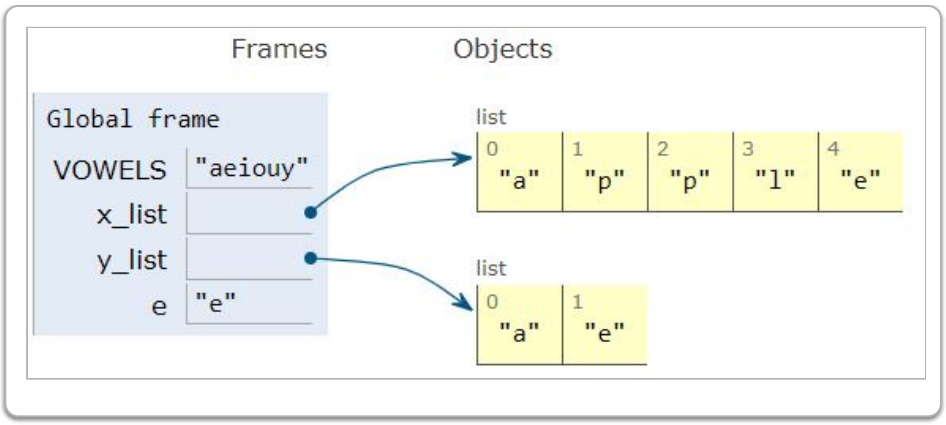


Lists Functions Part II

List *append()* Function

Use the *append()* method when you want to add **a single item** to the end of a list.

```
# extract a sub - list of vowels
VOWELS = 'aeiouy'
x_list = ['a','p','p','l','e']

y_list = []
for e in x_list:
    if e in VOWELS:
            y_list.append(e)
```



Test Yourself 3.6.06

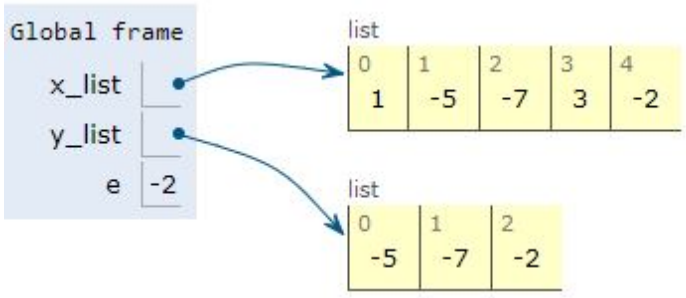
Use *append()* to construct *y_list* with negative elements from *x_list*:

x_list = [1,-5, -7, 3, -2]

y_list = [-5, -7, -2]

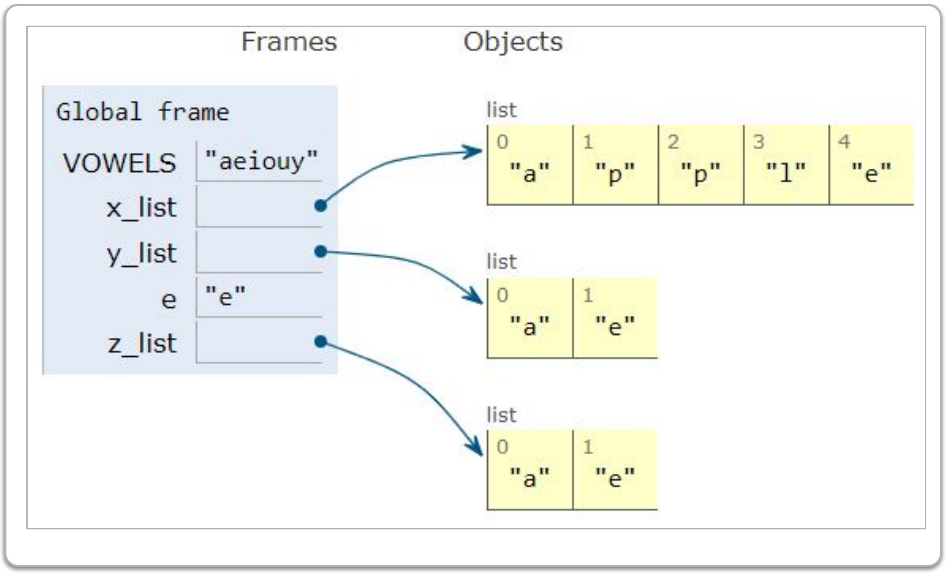
Suggested program:

```
x_list = [1,-5, -7, 3, -2]
y_list = []

for e in x_list:
    if e < 0:
         y_list.append(e)
```



List *comprehension*

```
VOWELS = 'aeiouy'
x_list = ['a','p','p','l','e']
y_list = []
for e in x_list:
    if e in VOWELS:
            y_list.append(e)
z_list = [e for e in x_list if e in VOWELS]
```



Test Yourself 3.6.07

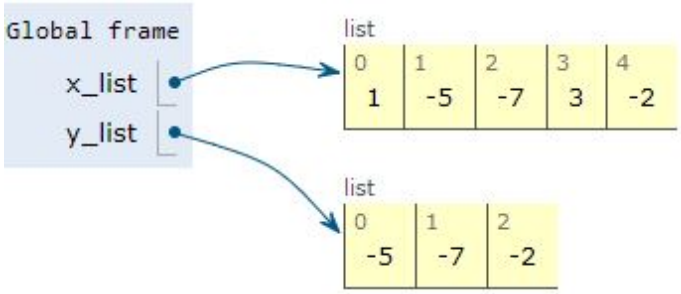
Use list comprehension to construct *y_list* with negative elements from *x_list*:

x_list = [1,-5, -7, 3, -2]

y_list = [-5, -7, -2]

Suggested program:

```
x_list = [1,-5, -7, 3, -2]
y_list = [e for e in x_list if e < 0]
```



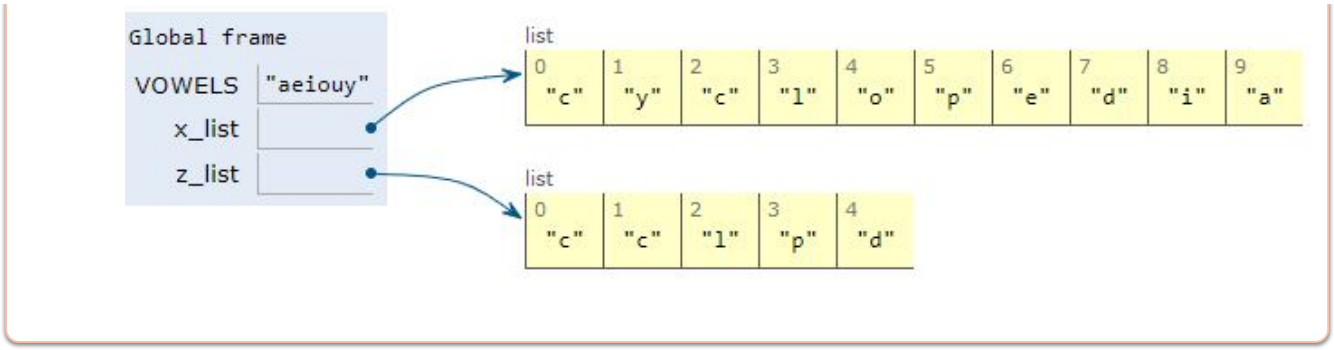
Test Yourself 3.6.08

Use list comprehension to construct a list of consonants in *x_list*:

x_list=list("cyclopedia")

Suggested program:

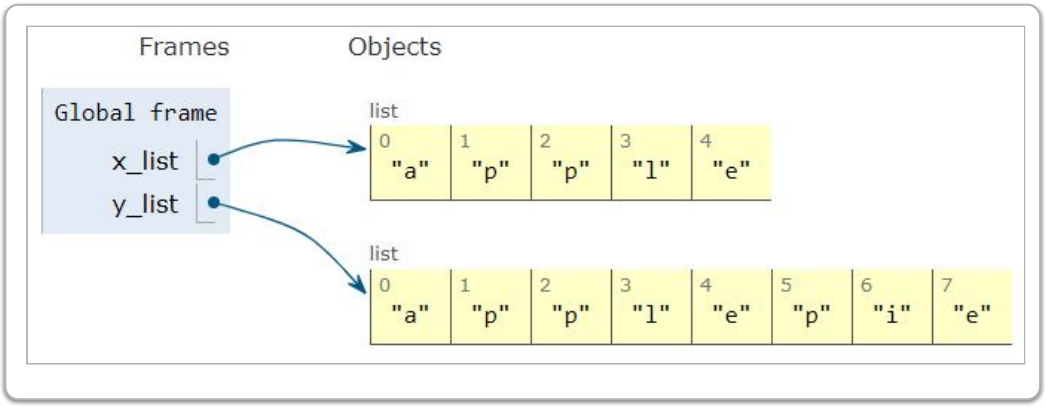
```
VOWELS = 'aeiouy'
x_list = list("cyclopedia")
z_list = [e for e in x_list if e not in VOWELS]
```



List *extend()* Function

Use the *extend()* method if you need to **append several items to the end of a list as individual items**.

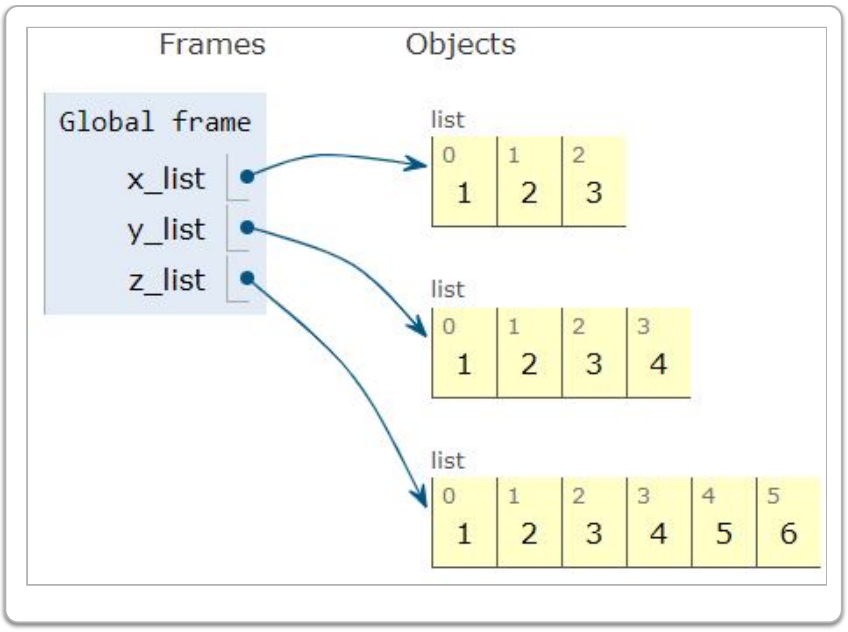
```
# extend a list by another list
x_list = ['a','p','p','l','e']
y_list = ['a','p','p','l','e']

y_list.extend(['p','i','e'])
```



append() vs. *extend()*

1. Lists can "grow" *in place* by using *append()* and *extend()* (the length of a list *can* vary).
2. *append()* adds a single element to the end of the list while *.extend()* can add multiple individual elements to the end of the list. If you want to add several individual items to a list using .append(), you need to use .append() several times.

```
x_list = [1, 2, 3]

y_list = [1, 2, 3]
y_list.append(4)          # add a single element

z_list = [1, 2, 3]
z_list.extend([4,5,6])    # add another list
```



Test Yourself 3.6.09

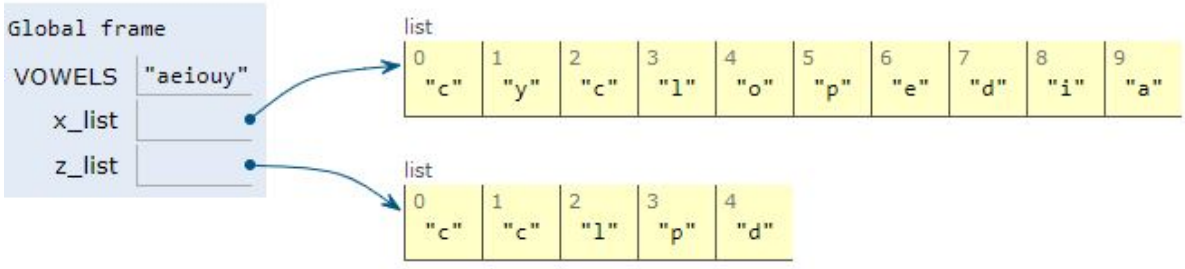
Show two ways to merge *x_list* and *y_list*:

x_list = [1, -5, -7, 3]

y_list = [3, 1, 5]

Suggested program:

```
x_list = [1, -5, -7, 3]
y_list = [3, 1, 5]

z_list = x_list.copy()
for e in y_list:
    z_list.append(e)

w_list = x_list.copy()
w_list.extend(y_list)
```



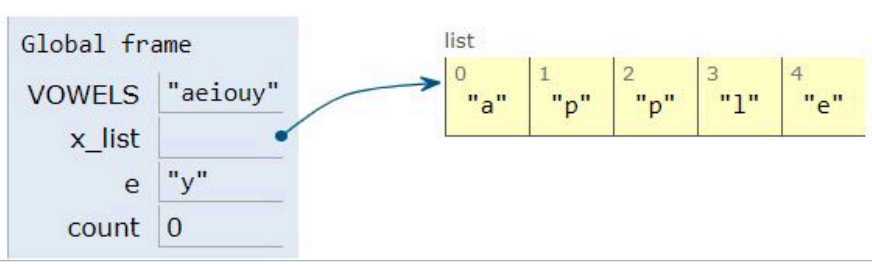
List *count()* Function

```
# count vowels
VOWELS = 'aeiouy'
x_list = ['a','p','p','l','e']
```

```
for e in VOWELS:
    count = 0
    if e in x_list:
        count = x_list.count(e)
        print('vowel: ', e, 'count: ', count)
```

Print output (drag lower right corner to resize)

```
vowel:  a  count:  1
vowel:  e  count:  1
vowel:  i  count:  0
vowel:  o  count:  0
vowel:  u  count:  0
vowel:  y  count:  0
```



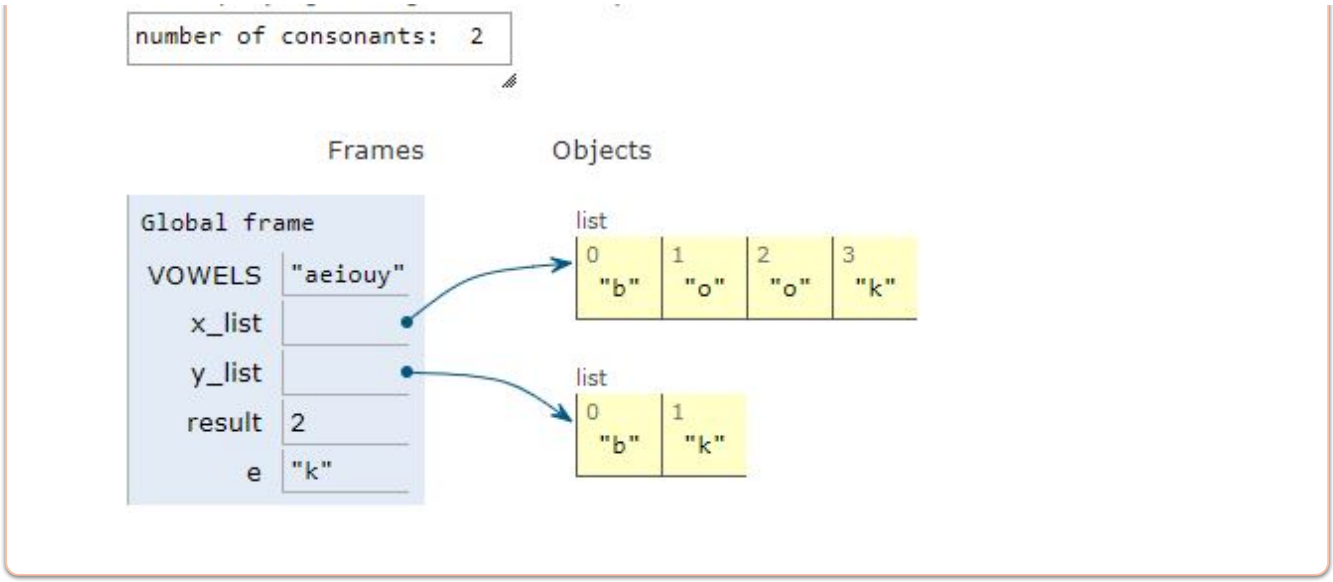
Test Yourself 3.6.10

Count consonants in *x_list*:

x_list = ["b","o","o","k"]

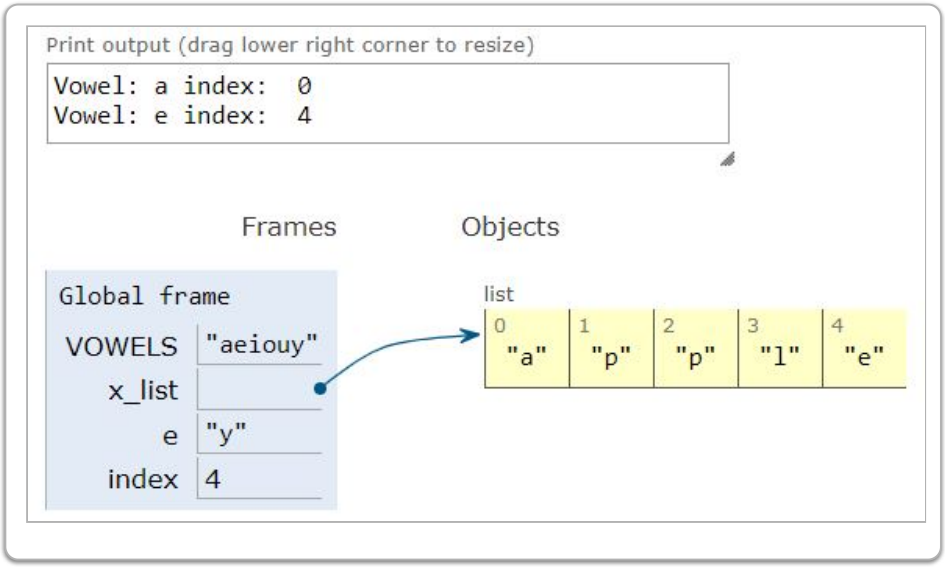
Suggested program:

```
VOWELS = 'aeiouy'
x_list = ["b","o","o","k"]
y_list = []
result = 0

for e in x_list:
    if e not in VOWELS and e not in y_list:
        y_list.append(e)
        result = result + x_list.count(e)

print("number of consonants: ", result)
```



List *index()* Function

```
# first occurence of vowels in a list
VOWELS = 'aeiouy'
x_list = ['a','p','p','l','e']

for e in VOWELS:
    if e in x_list:
       index = x_list.index(e)
       print('Vowel: ', e, 'index: ', index)
```



Test Yourself 3.6.11

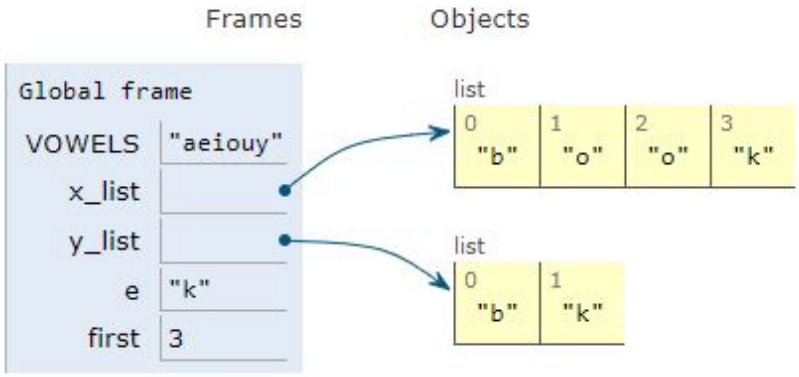
Print indices for first occurence of consonants in *x_list*:

x_list = ["b","o","o","k"]

Suggested program:

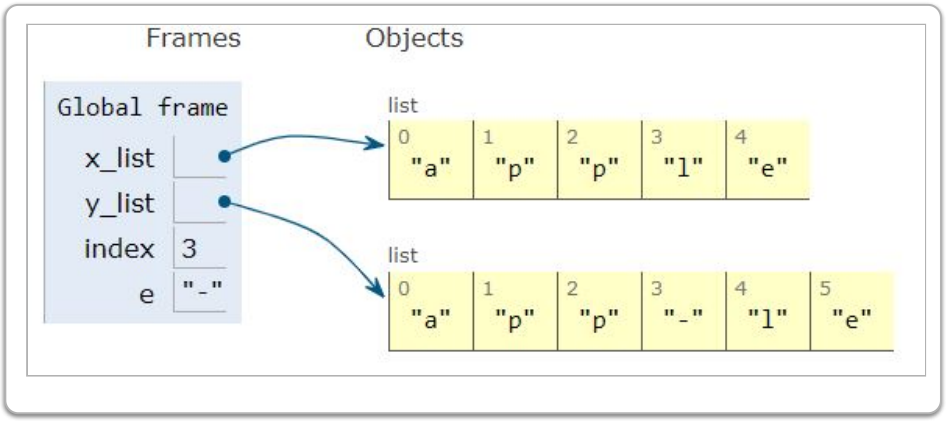
```
VOWELS = 'aeiouy'
x_list = ["b","o","o","k"]
y_list = []

# compute list of consonants w/o duplication
for e in x_list:
    if e not in VOWELS and e not in y_list:
        y_list.append(e)

for e in y_list:
    first = x_list.index(e)
    print(e, " first index: ", first)
```

```
b  first index:  0
k  first index:  3
```



List *insert()* Function

```
# insert element at index in a list
x_list = ['a','p','p','l','e']
y_list = ['a','p','p','l','e']

index = 3
e = '-'
y_list.insert(index, e)
```



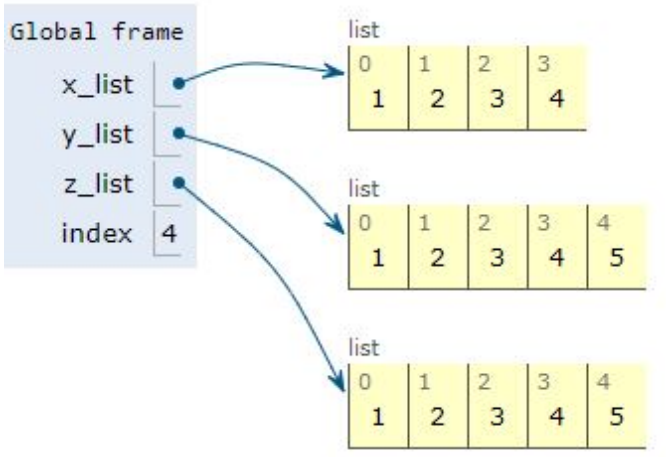
Test Yourself 3.6.12

Show two ways to add number 5 at the end of *x_list*:

x_list = [1, 2, 3, 4]

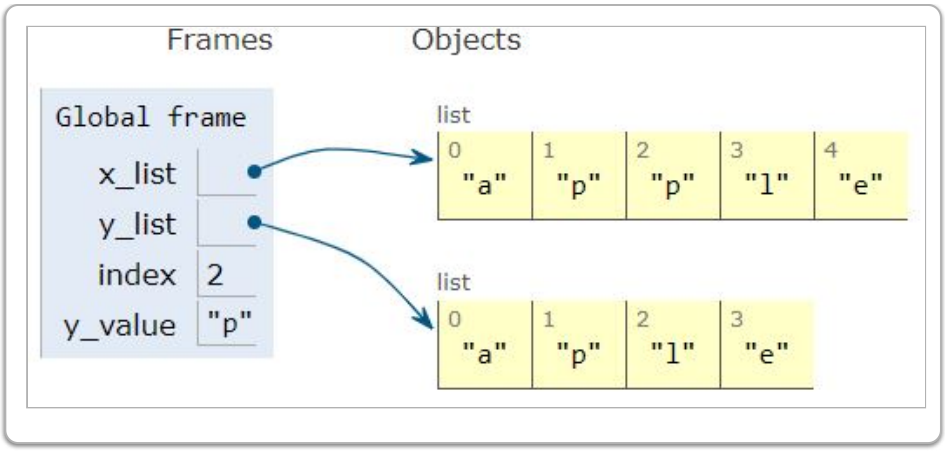
Suggested program:

```
x_list = [1, 2, 3, 4]

y_list = x_list.copy()
y_list.append(5)

z_list = x_list.copy()
index = len(z_list)
z_list.insert(index, 5)
```



List *pop(i=-1)* Function

```
# remove element at index 2
x_list = ['a','p','p','l','e']
y_list = ['a','p','p','l','e']

index = 2
y_value = y_list.pop(index)
```



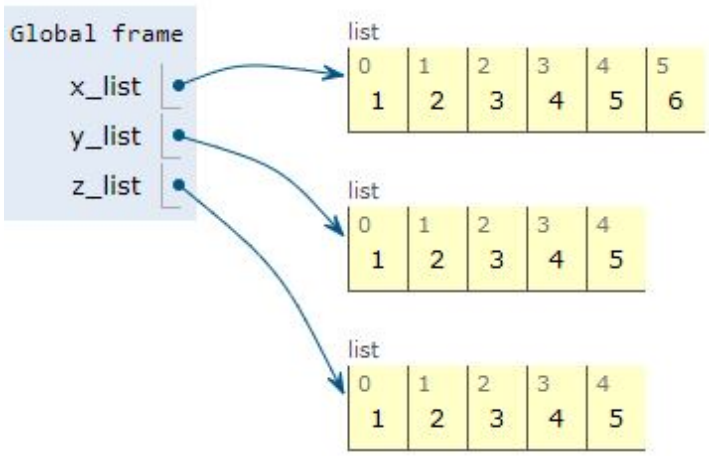
Test Yourself 3.6.13

Show two ways to remove last element from *x_list*:

x_list = [1, 2, 3, 4, 5, 6]

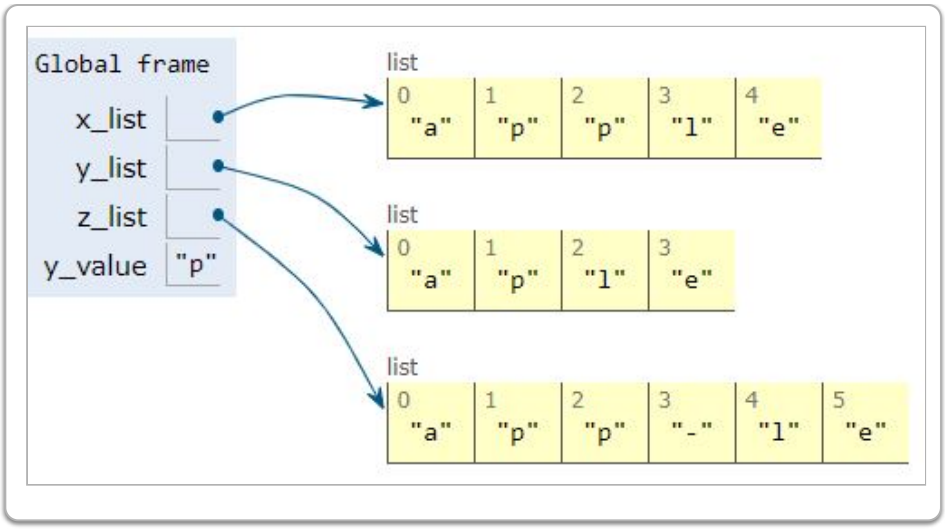
Suggested program:

```
x_list = [1 ,2 ,3 ,4 ,5 ,6]
y_list = x_list.copy()
y_list.pop(-1)
z_list = x_list.copy()
z_list = z_list[ : : -1]
```



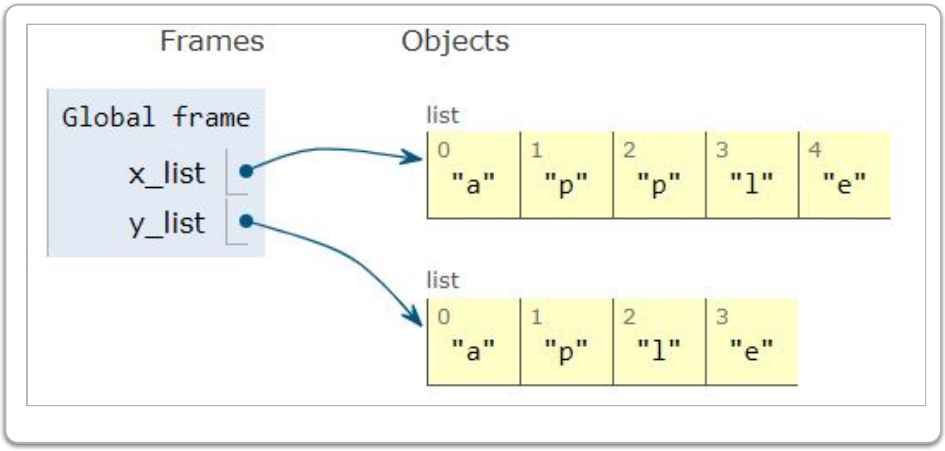
pop() vs. *insert()*

```
# pop: remove element at index
# insert: insert element at index
x_list = ['a','p','p','l','e']
y_list = ['a','p','p','l','e']
z_list = ['a','p','p','l','e']

y_value = y_list.pop(2)
z_list.insert(3, '-')
```



List *remove()* Function

```
# remove first occurence from a list
x_list = ['a','p','p','l','e']
y_list = ['a','p','p','l','e']

y_list.remove('p')
```



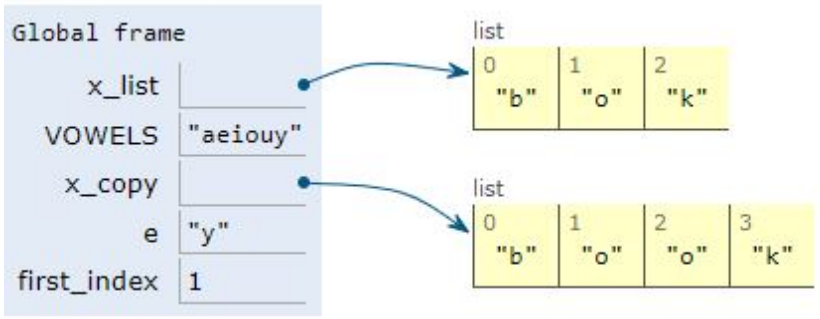
Test Yourself 3.6.14

Remove first occurence of vowels in *x_list*:

x_list = ["b","o","o","k"]

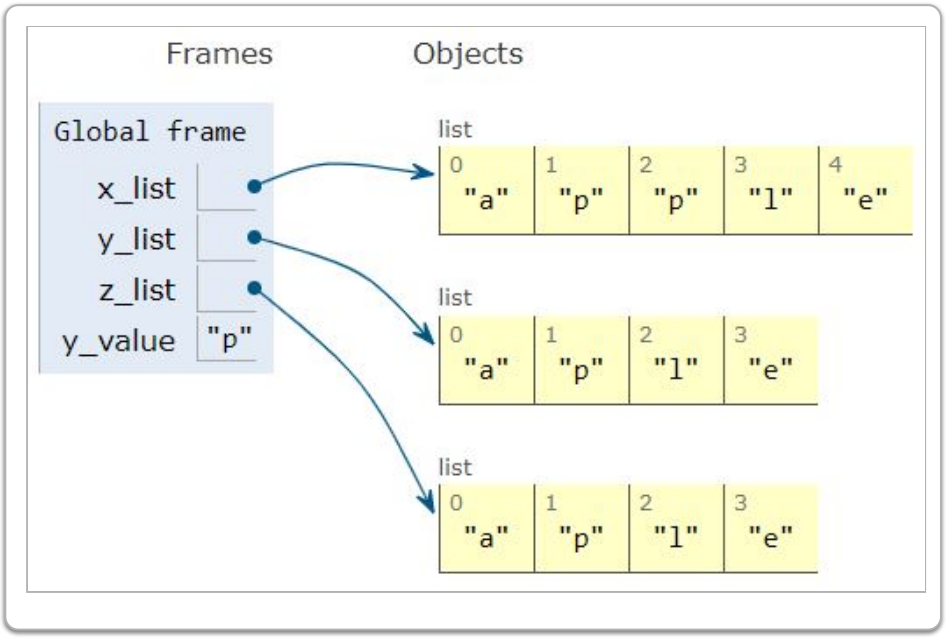
Suggested program:

```
x_list = ["b","o","o","k"]
VOWELS = 'aeiouy'
x_copy = x_list.copy()
for e in VOWELS:
    if e in x_list:
            first_index = x_list.index(e)
                x_list.pop(first_index)
```



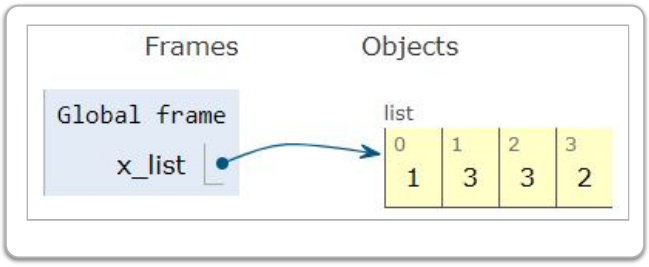
remove() vs. *pop()*

```
# pop: remove element at index
# remove: remove first occurence by value
x_list = ['a','p','p','l','e']
y_list = ['a','p','p','l','e']
z_list = ['a','p','p','l','e']

y_value = y_list.pop(2)
z_list.remove('p')
```

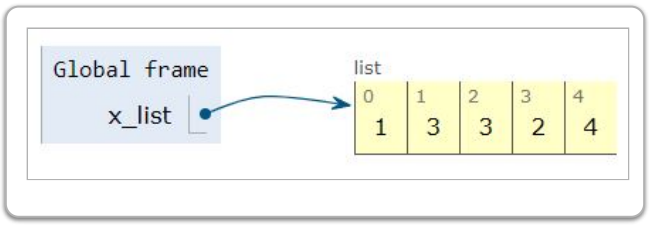


Common List Methods

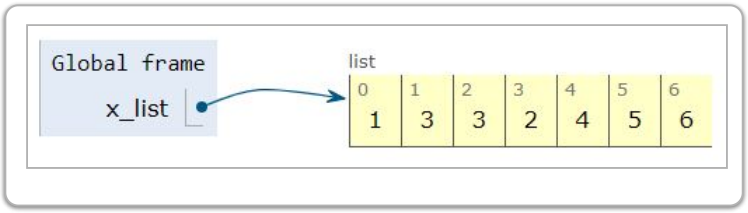
```
x_list = [1,3,3,2]
```



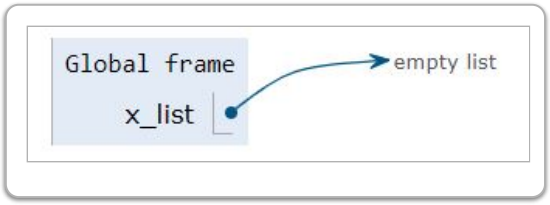

```
x_list = [1,3,3,2]

# adds a single element
x_list.append(4)
```



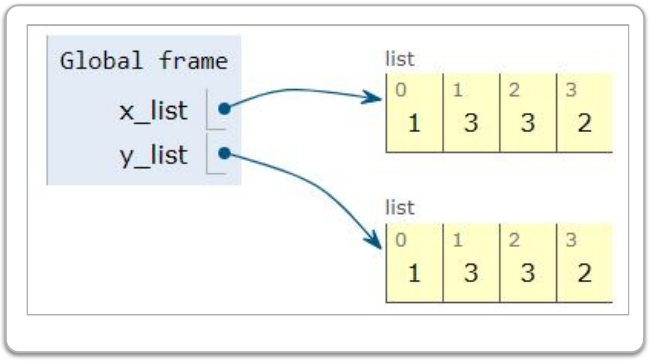

```
x_list = [1,3,3,2]

# adds a new list
x_list.extend([4,5,6])
```



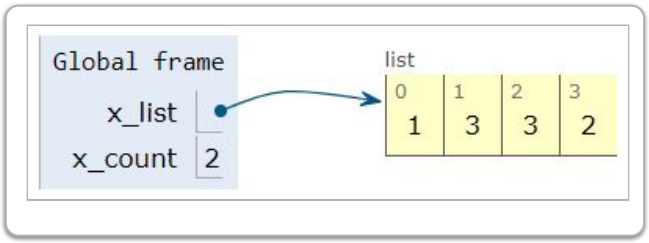

```
x_list = [1,3,3,2]

x_list.clear()
```



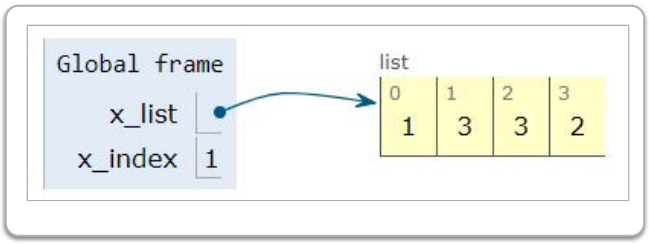

```
x_list = [1,3,3,2]

# shallow copy only!
y_list = x_list.copy()
```



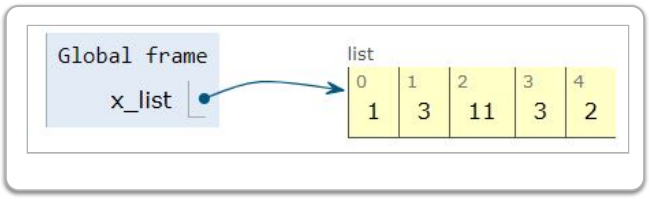
```
x_list = [1,3,3,2]

# count e = 3
x_count = x_list.count(3)
```



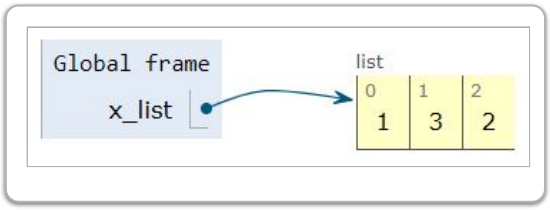

```
x_list = [1,3,3,2]

# index of first e=3
x_index = x_list.index(3)
```



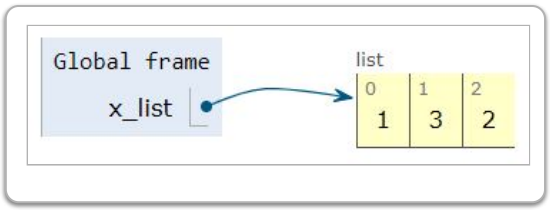

```
x_list = [1,3,3,2]

# insert 11 at index=2
x_index = x_list.index(3)
```



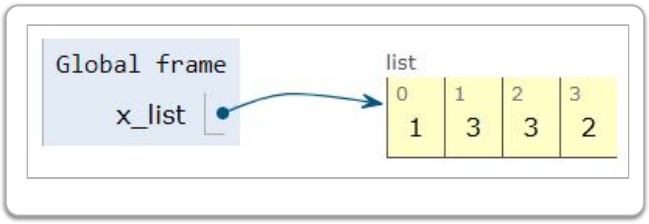

```
x_list = [1,3,3,2]

# remove at index 2
x_list.pop(2)
```



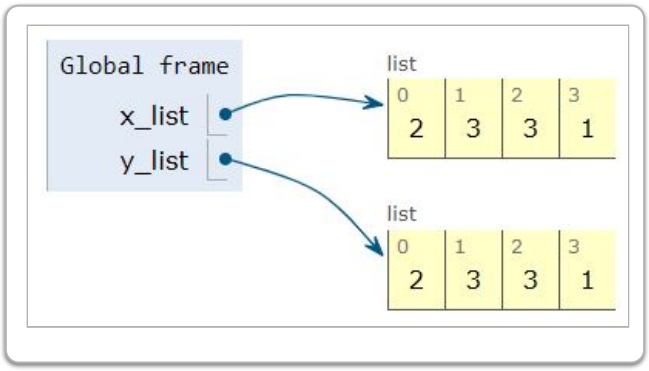

```
x_list = [1,3,3,2]

# remove first 3
x_list.remove(3)
```




```
x_list = [1,3,3,2]

x_list.sort(0)
```



```
x_list = [1,3,3,2]

y_list = x_list[::-1]

# reverse order in-place
x_list.reverse()
```



extend() vs. '+'

```
x_list = [1, 2, 3]
y_list = [1, 2, 3]
y_id_1 = id(y_list)
y_list.extend([4, 5])     # in-place
y_id_2 = id(y_list)

z_list = [1, 2, 3]
z_id_1 = id(z_list)
z_list = z_list + [4, 5]  # new object
z_id_2 = id(z_list)
```