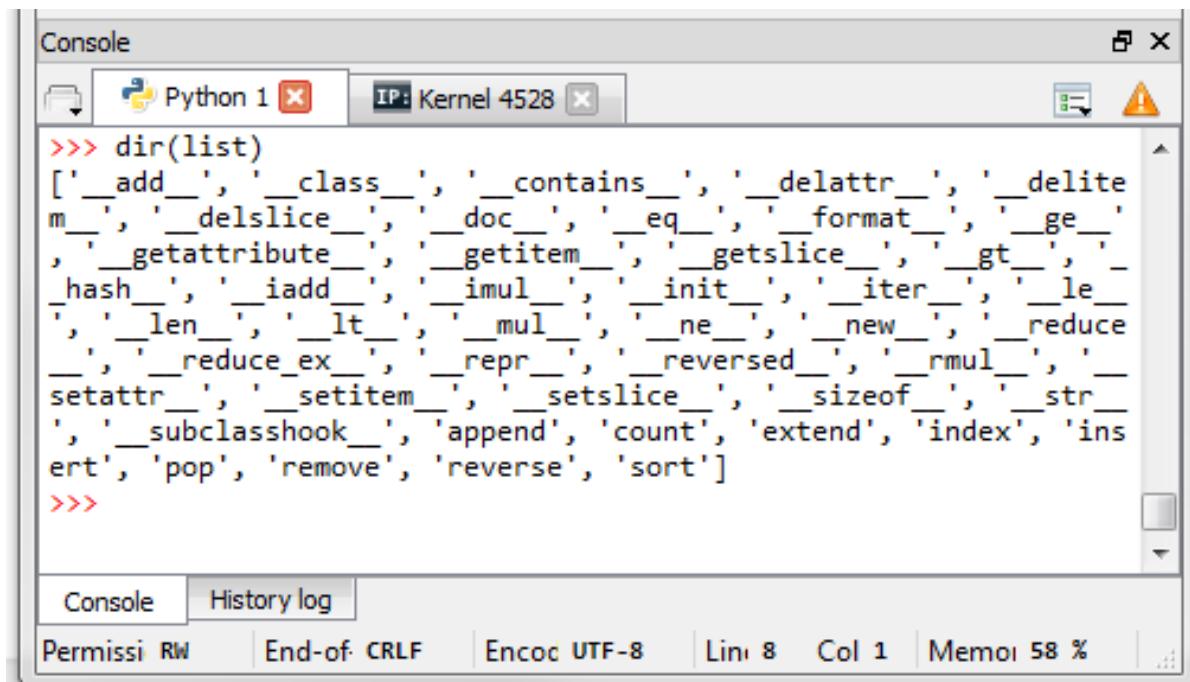


Lists: Methods

list Type Methods

>>> **dir(list)**



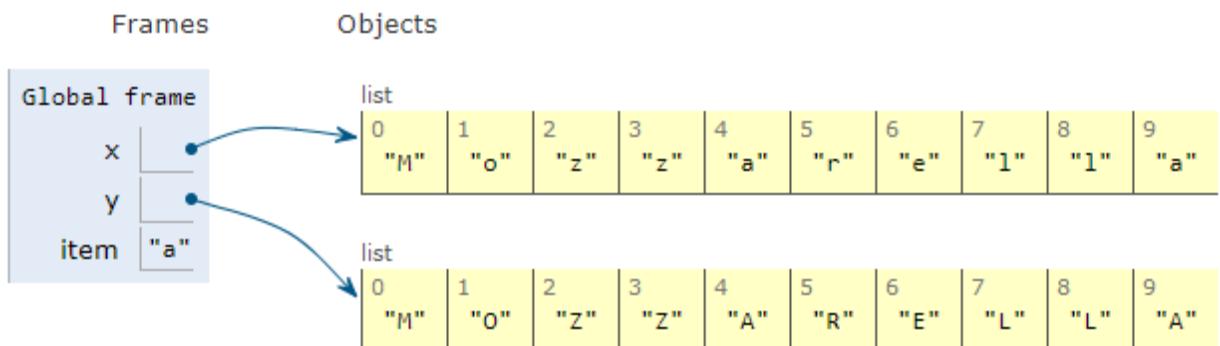
```
Console
Python 1 IP: Kernel 4528
>>> dir(list)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__delslice__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattribute__', '__getitem__', '__getslice__', '__gt__',
 '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__',
 '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__',
 '__setitem__', '__setslice__', '__sizeof__', '__str__',
 '__subclasshook__', 'append', 'count', 'extend', 'index', 'insert',
 'pop', 'remove', 'reverse', 'sort']
>>>
```

Console History log
Permissi RW End-of CRLF Encod UTF-8 Lin: 8 Col: 1 Memoi 58 %

```
>>> x = ['M','o','z','z','a','r','e','l','l','a']
>>> x.sort()
>>> print(x)
['M', 'a', 'a', 'e', 'l', 'l', 'o', 'r', 'z', 'z']
```

List Iteration

```
>>> # capitalize all letters in x  
>>> x = list("Mozzarella")  
>>> y = [] # y = list()  
>>> for item in x:  
>>>     y.append(item.upper())
```



List Iteration Example: Remove Vowels

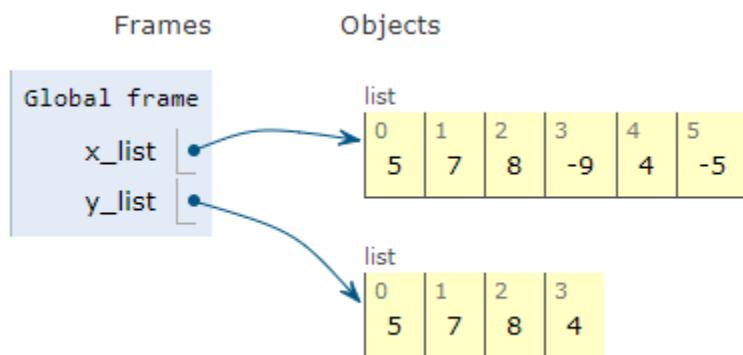
```
>>> x = list("Mozzarella")
>>> y = []
>>> # same as y = list()
>>> VOWELS = list("aeiou")
>>> for item in x:
>>>     if item not in VOWELS:
>>>         y.append(item)
```



General Form of List Comprehension

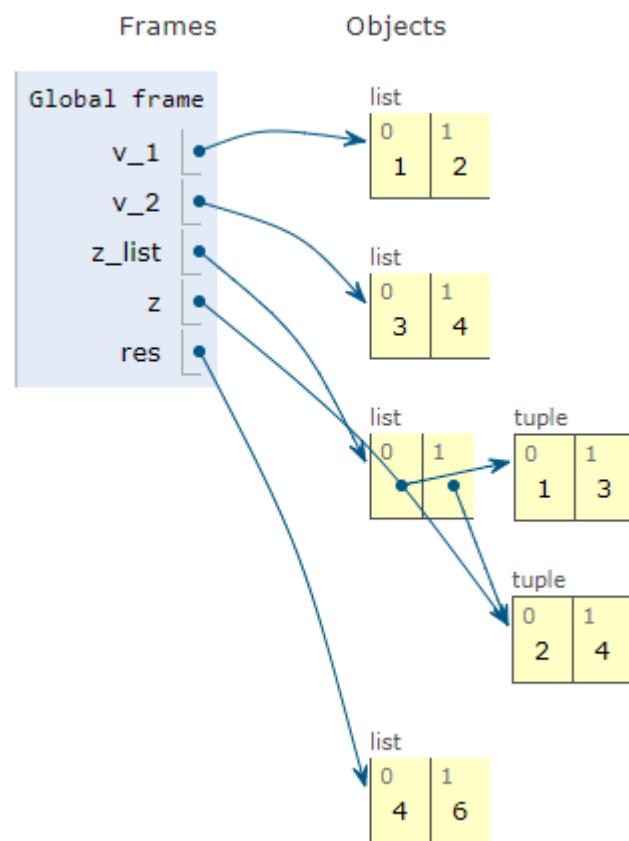
[expression for item in iterable
if condition]

```
>>> x_list = [5, 7, 8, -9, 4, -5]
>>> y_list = [x + 1 for x in x_list if x > 0]
>>> y_list
>>> [5, 7, 8, 4]
```



List Comprehension: Vector Addition

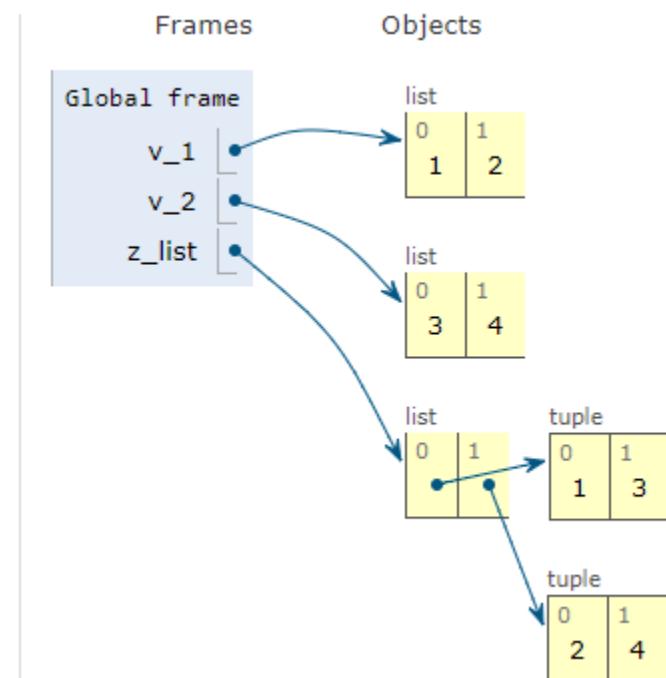
```
>>> v_1 = [1 ,2]; v_2 = [3, 4]
>>> z_list = zip(v_1, v_2)
```



```
>>> res = [z[0] + z[1] for z in z_list]  
>>> res  
[4, 6]
```

List Comprehension: Vector Dot Product

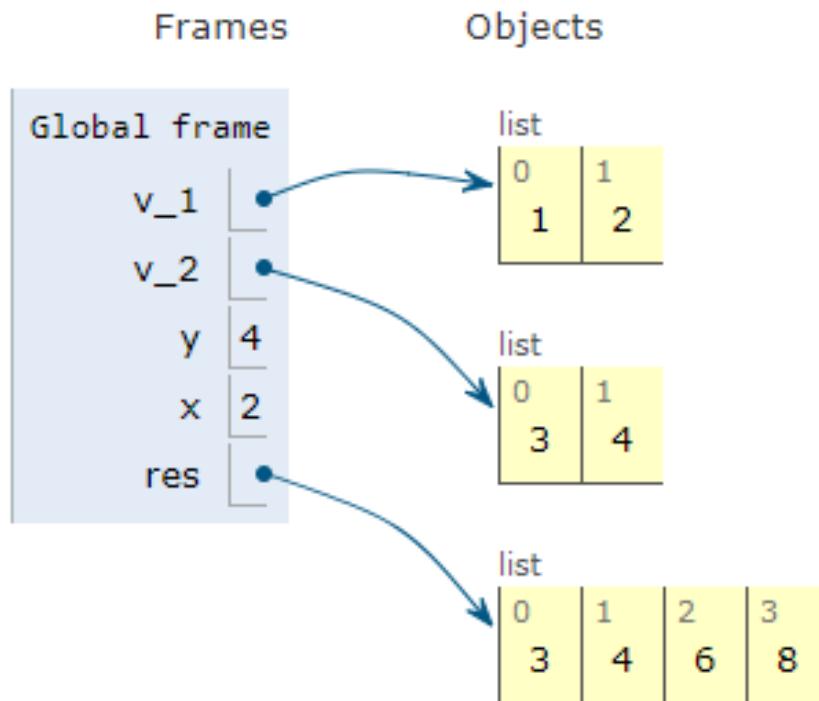
```
>>> v_1 = [1 ,2]  
>>> v_2 = [3, 4]  
>>> z_list = zip(v_1, v_2)
```



```
>>> res = sum(z[0] * z[1] for z in z_list)  
>>> res  
11
```

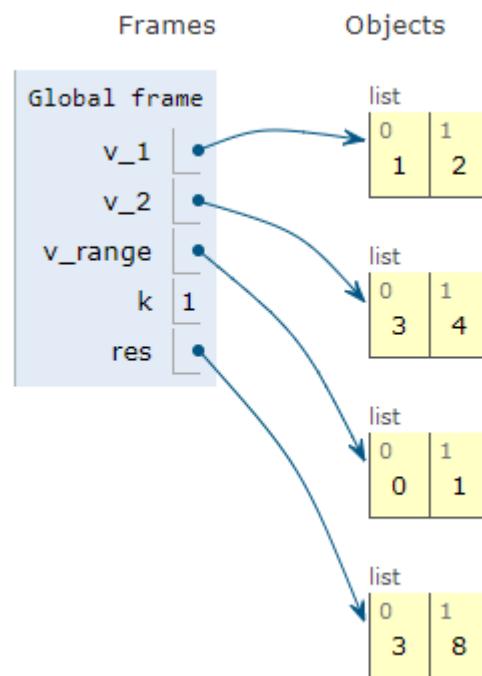
List Comprehension: Vector Cross Product

```
>>> v_1 = [1 ,2]
>>> v_2 = [3, 4]
>>> res = [x*y for x in v_1 for y in v_2]
>>> res
[3, 4, 6, 8]
```



List Comprehension: Hadamard Vector Product

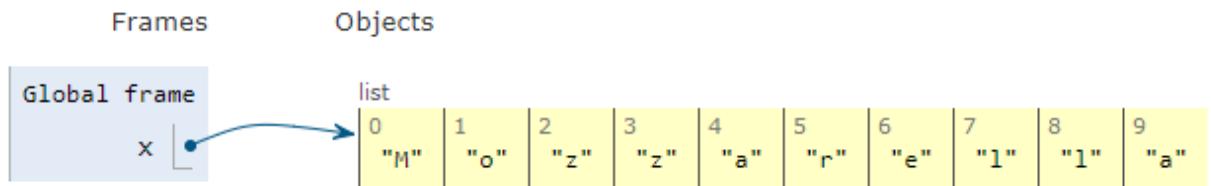
```
>>> # pointwise multiplication  
>>> v_1 = [1 ,2];  
>>> v_2 = [3, 4]  
>>> v_range = range(len(v_1))  
>>> res = [ v[k]+w[k]  for k in v_range ]  
>>> res  
[3, 8]
```



List Methods: *append()*

	0	1	2	3	4	5	6	7	8	9	
['M',	'o',	'z',	'z',	'a',	'r',	'e',	'l',	'l',	'a']
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1		

```
>>> x = ['M','o','z','z','a','r','e','l','l','a']
```

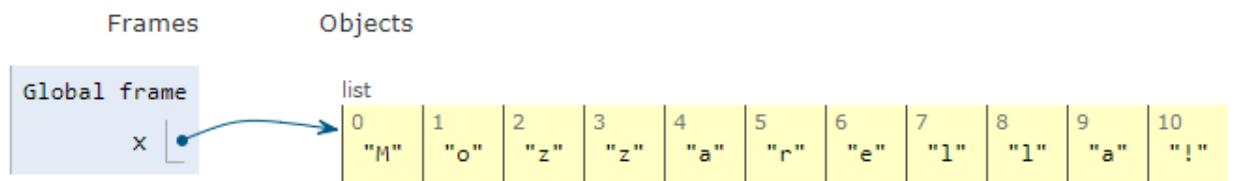


```
>>> # append '!' to the end of list x
```

```
>>> x.append('!')
```

```
>>> x
```

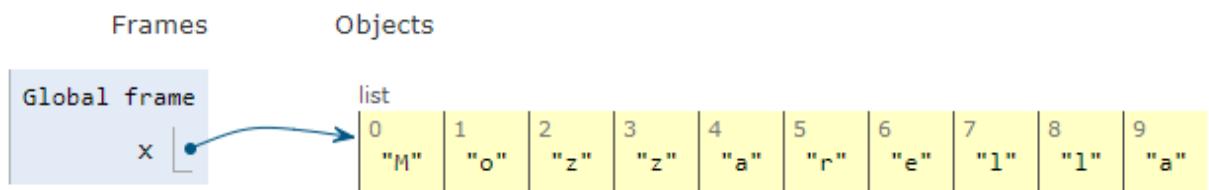
```
>>> ['M','o','z','z','a','r','e','l','l','a','!']
```



List Methods: *count()*

	0	1	2	3	4	5	6	7	8	9	
['M'	'o'	'z'	'z'	'a'	'r'	'e'	'l'	'l'	'a']
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

```
>>> x = ['M','o','z','z','a','r','e','l','l',a']
```

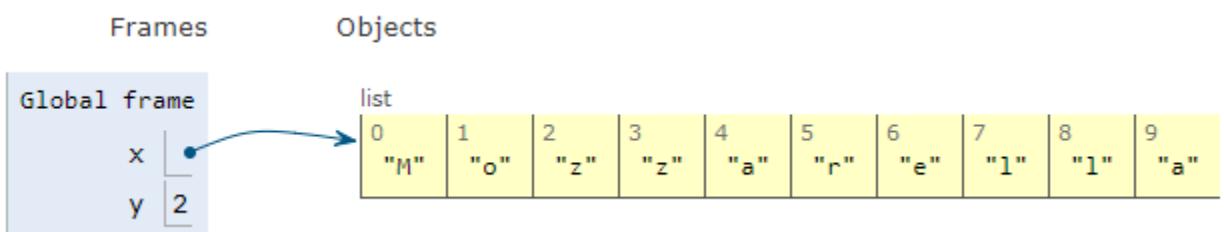


```
>>> # number of occurrences of 'a' in x
```

```
>>> y = x.count('a')
```

```
>>> y
```

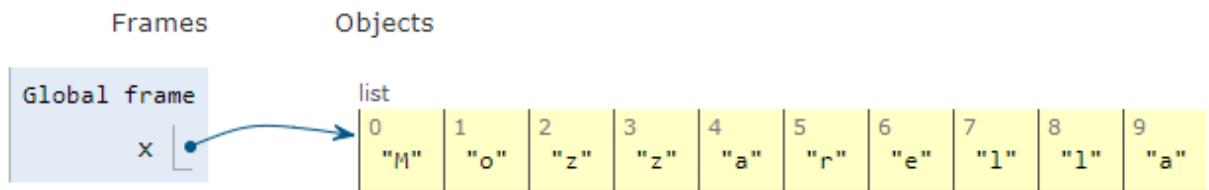
```
>>> 2
```



List Methods: *extend()*

[0	1	2	3	4	5	6	7	8	9]
'M',	'o',	'z',	'z',	'a',	'r',	'e',	'l',	'l',	'a'		
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1		

```
>>> x = ['M','o','z','z','a','r','e','l','l',a']
```

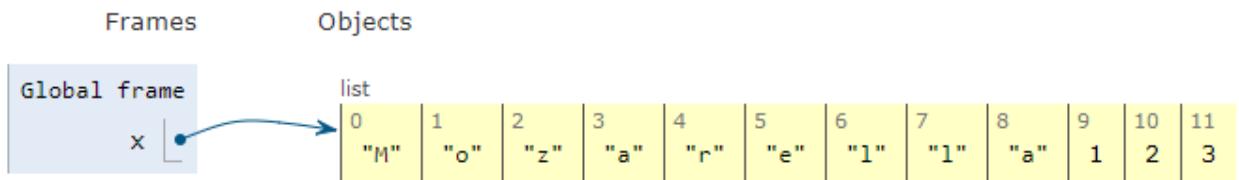


```
>>> append all items in list [1, 2, 3] to x
```

```
>>> x.extend([1, 2, 3])
```

```
>>> x
```

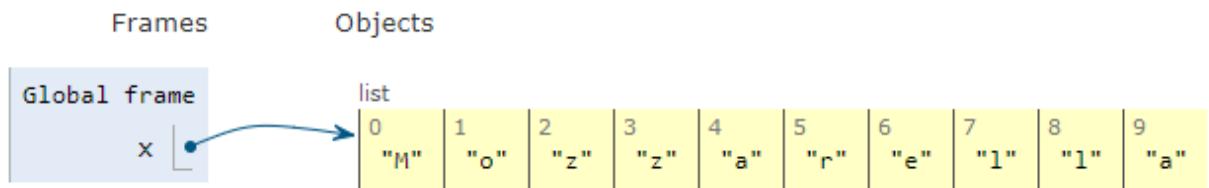
```
>>> ['M','o','z','z','a','r','e','l','l',a',1,2,3]
```



List Methods: *index()*

	0	1	2	3	4	5	6	7	8	9
['M'	'o'	'z'	'z'	'a'	'r'	'e'	'l'	'l'	'a'
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> x = ['M','o','z','z','a','r','e','T','l','a']
```

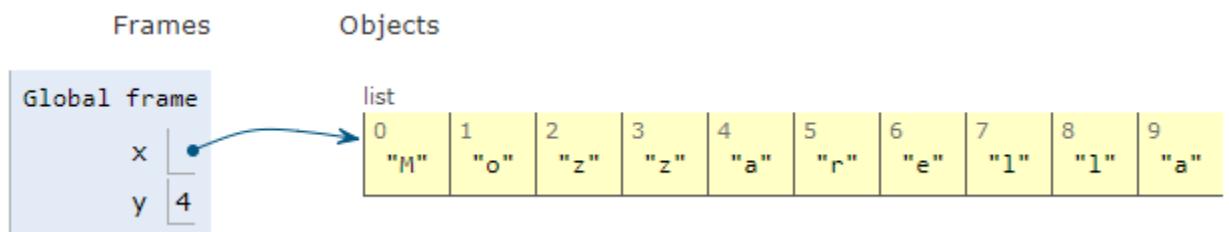


```
>>> # first occurrence of 'a' in list x
```

```
>>> y = x.index('a')
```

>>> y

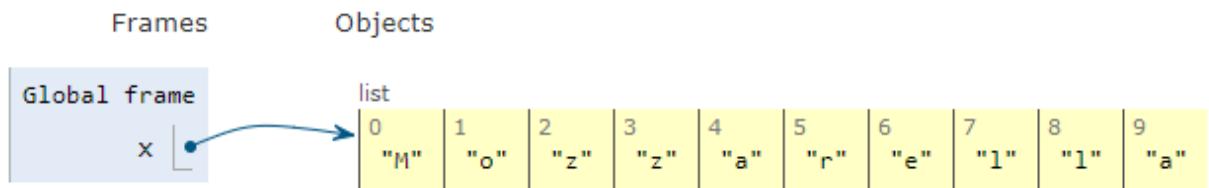
>>> 4



List Methods: *insert()*

	0	1	2	3	4	5	6	7	8	9
['M'	'o'	'z'	'z'	'a'	'r'	'e'	'l'	'l'	'a'
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> x = ['M','o','z','z','a','r','e','T','T','a']
```

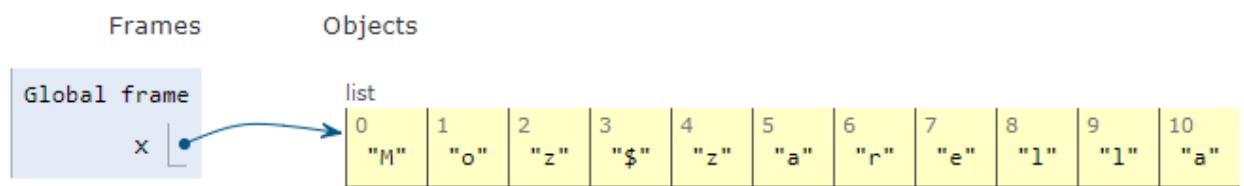


```
>>> # insert '$' at index 3 in list x
```

```
>>> x.insert(3, '$')
```

>>> X

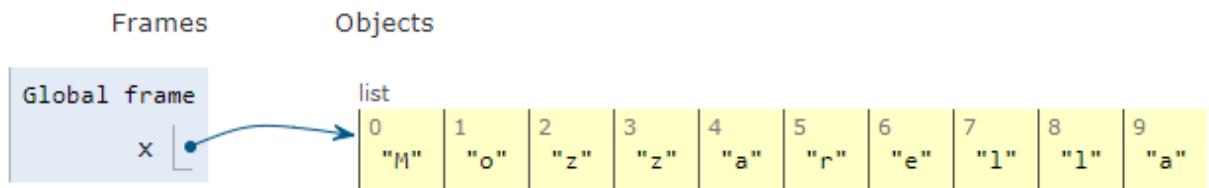
```
>>> ['M','o','z','$','z','a','r','e','l','l','a']
```



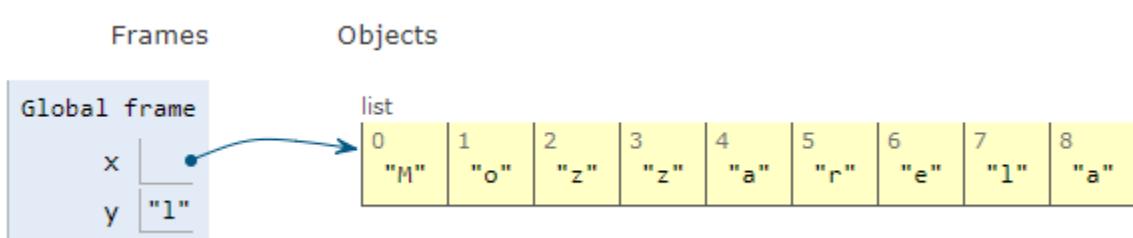
List Methods: $pop(i=-1)$

	0	1	2	3	4	5	6	7	8	9
['M',	'o',	'z',	'z',	'a',	'r',	'e',	'l',	'l',	'a'
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> x = ['M','o','z','z','a','r','e','T','l','a']
```



```
>>> # return the value at index i and  
>>> # remove this value from the list  
>>> y = x.pop(-3)  
>>> x  
>>> ['M' 'o' 'z' 'z' 'a' 'r' 'e' 'l' 'a']
```



List Element Removal

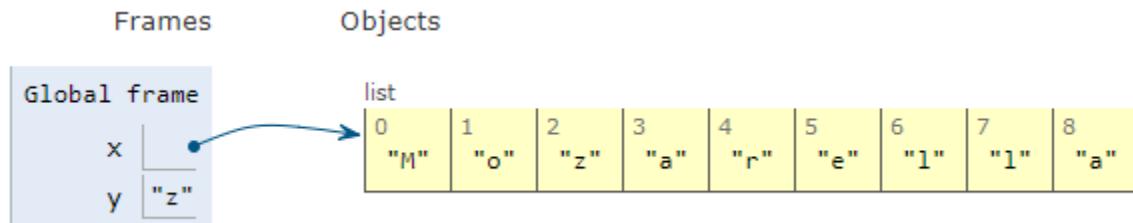
	0	1	2	3	4	5	6	7	8	9	
['M',	'O',	'Z',	'Z',	'a',	'r',	'e',	'l',	'l',	'a']
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1		

```
>>> x = ['M','o','z','z','a','r','e','l','l',a']
```

```
>>> y = x.pop(2)
```

```
>>> y
```

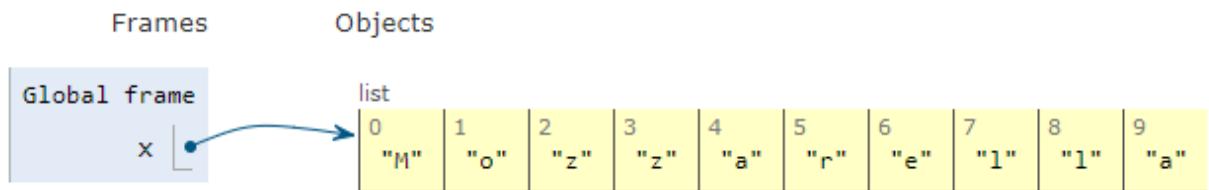
```
>>> z
```



List Methods: *remove()*

	0	1	2	3	4	5	6	7	8	9	
['M'	'o'	'z'	'z'	'a'	'r'	'e'	'l'	'l'	'a']
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

```
>>> x = ['M','o','z','z','a','r','e','l','l',a']
```

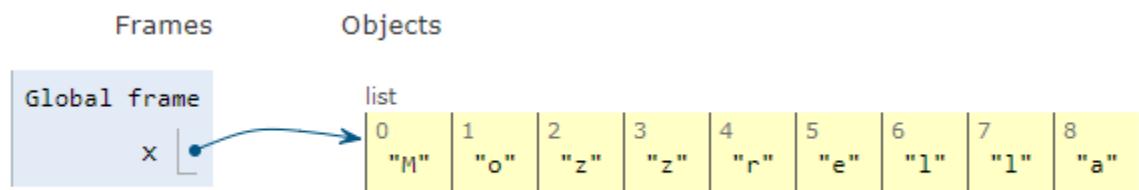


```
>>> # remove first 'a' from list x
```

```
>>> x.remove('a')
```

```
>>> x
```

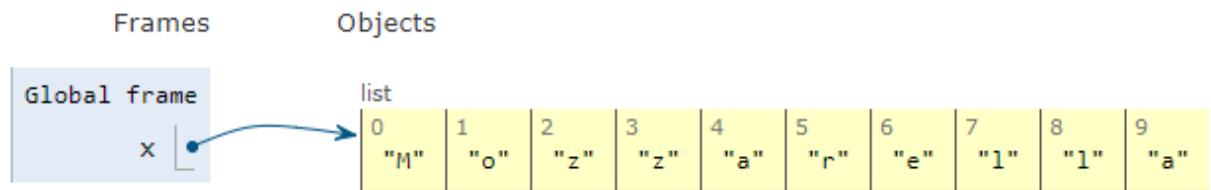
```
>>> ['M','o','z','z','r','e','l','l','a']
```



List Methods: *reverse()*

	0	1	2	3	4	5	6	7	8	9	
['M'	'o'	'z'	'z'	'a'	'r'	'e'	'l'	'l'	'a']
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

```
>>> x = ['M','o','z','z','a','r','e','l','l',a']
```

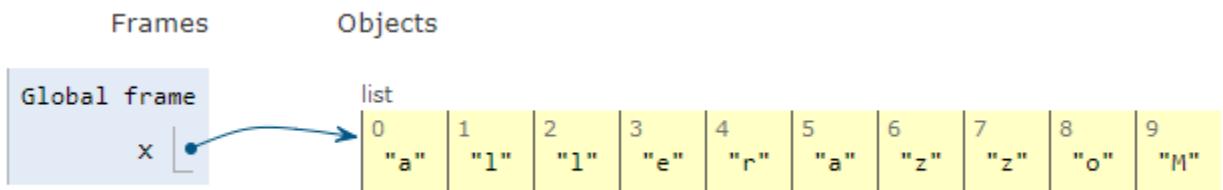


```
>>> # reverse list x in-place
```

```
>>> x.reverse()
```

```
>>> x
```

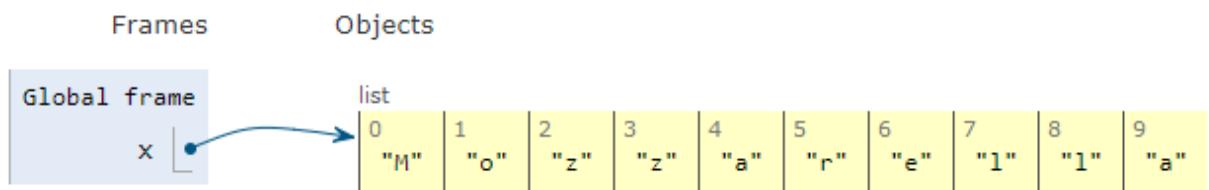
```
>>> ['a','l','l','e','r','a','z','z','o','M']
```



List Methods: *sort()*

	0	1	2	3	4	5	6	7	8	9
['M',	'o',	'z',	'z',	'a',	'r',	'e',	'l',	'l',	'a'
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> x = ['M','o','z','z','a','r','e','T','l','a']
```

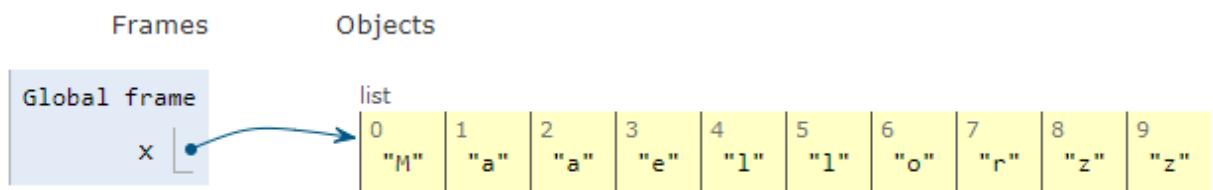


```
>>> # sort list x in-place
```

```
>>> x.sort()
```

>>> X

```
>>> ['M','a','a','e','l','l','o','r','r','z']
```



min(), max(), sum()



```
>>> x.sort()
```

```
>>> min_x = min(x); max_x = max(x)
```

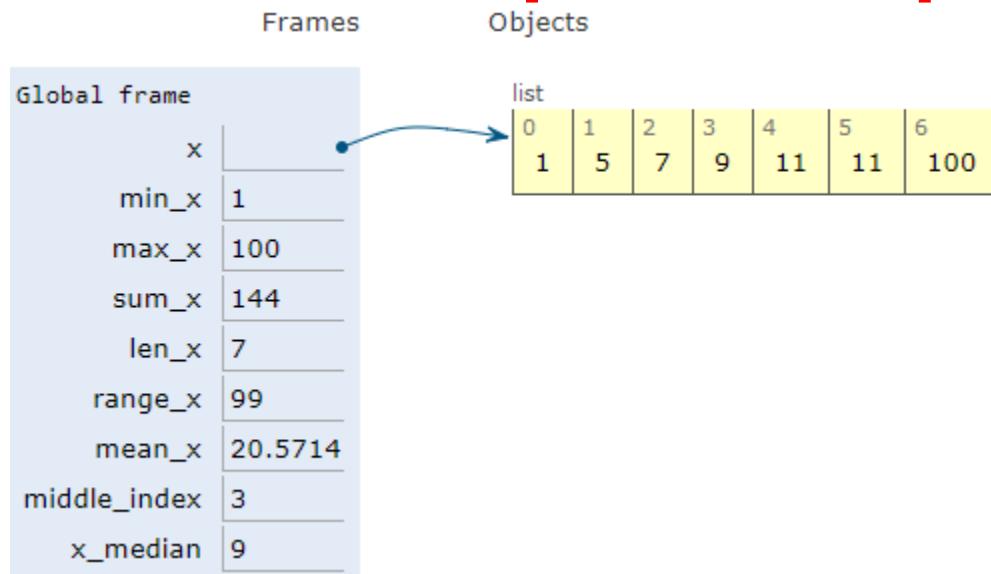
```
>>> sum_x = sum(x); len_x = len(x)
```

```
>>> range_x = max_x - min_x
```

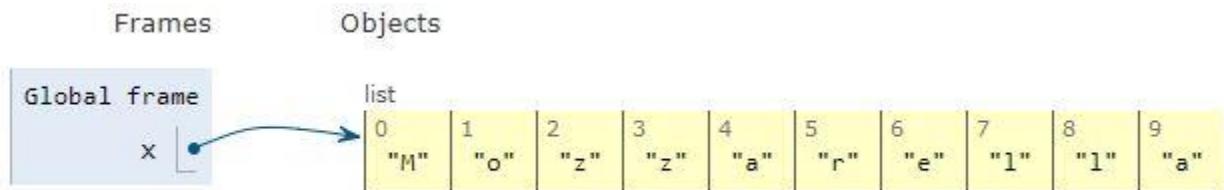
```
>>> mean_x = float(sum_x)/len_x
```

```
>>> middle_index = len_x // 2
```

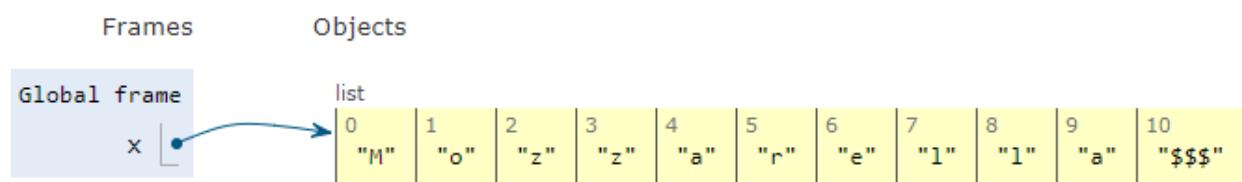
```
>>> x_median = x[middle_index]
```



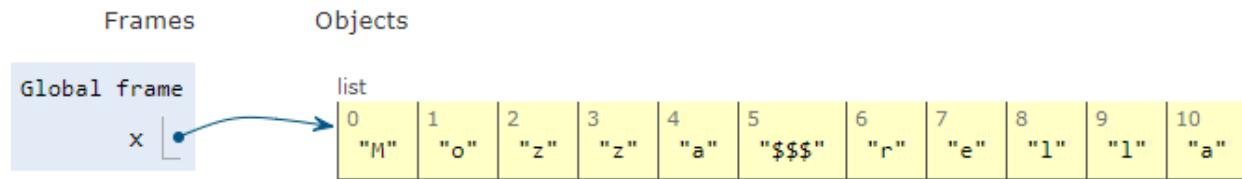
Common List Methods



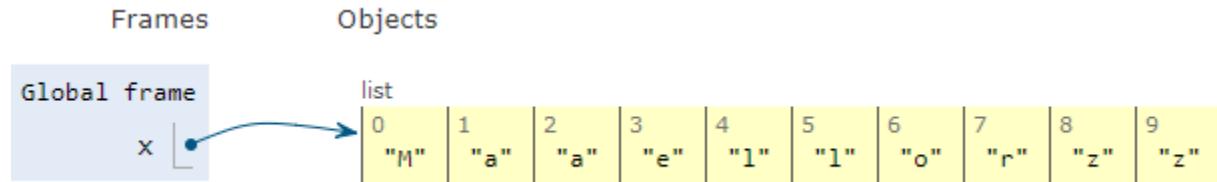
>>> x.append('\$')



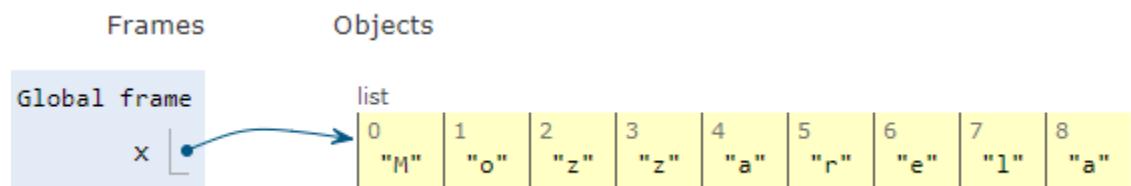
>>> x.insert(5, '\$')



>>> x.sort()



>>> x.remove('l')



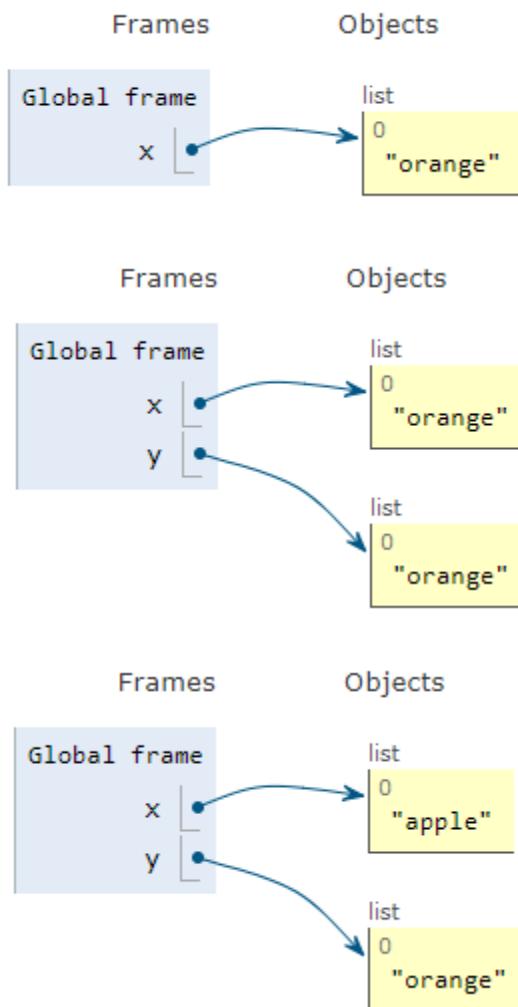
List Copy: Quiz

- what is the result of last print statement?

```
>>> x = ['orange']
>>> print(len(x))
>>> 1
>>> print(x[0])
>>> orange
>>> y = x
>>> x[0] = 'apple'
>>> print(x[0], y[0])
```

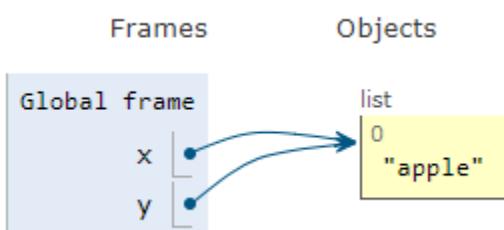
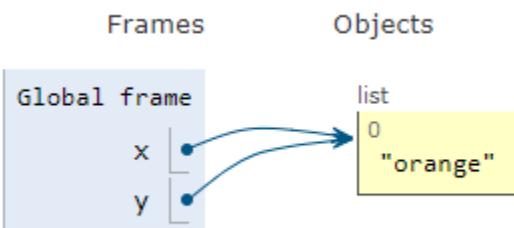
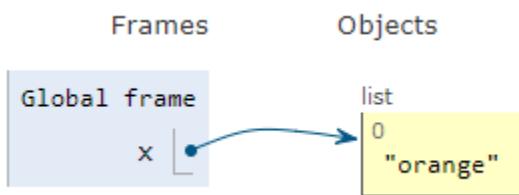
List Copy: Quiz (cont'd)

```
>>> x = ['orange']
>>> y = x
>>> x[0] = 'apple'
>>> print(x[0], y[0])
>>> ('apple', 'orange')
```



List Copy: Quiz (cont'd)

```
>>> x = ['orange']
>>> y = x
>>> x[0] = 'apple'
>>> print(x[0], y[0])
>>> ('apple', 'apple')
```



Review Problems

Interview Problem

- explain the difference between list *append()* and list *extend()* methods

Interview Problem

- what is *list comprehension*?

Interview Problem

- given the list `x = ["One", "Two", "Three"]`
- convert `x` to string `"One, Two, Three"`
- show two different ways to do so

Interview Problem

- how to count the occurrences of each item present in the list without explicitly mentioning them?

Interview Problem

- sort integers in list x

```
>>> x = [6, 9, 3, 91, 54, 122, 3]
```

Interview Problem

- given the list with odd and even numbers

```
>>> x = [1,3,8,3,6,8,11,19, 22]
```

- create two lists:
 - (1) x_even with even numbers from x
 - (2) x_odd with odd numbers from x

Interview Problem

- remove duplicates from a list

Interview Problem

- the following code is supposed to remove numbers less than 5 from list n, but there is a bug. Fix the bug.

```
n = [1,2,5,10,3,100,9,24]
```

```
for e in n:  
    if e < 5:  
        n.remove(e)  
print n
```

Interview Problem

- remove duplicates from a list

Interview Problem

- given the list
`>>> x = ['1', '13', '61', '99', '17']`
- convert to list of numbers:
`y = [1, 13, 61, 99, 17]`
- show two different ways to do so
(including list comprehension)

Interview Problem

- how to generate random numbers?

Interview Problem

- what built-in function can be used to iterate over a number sequence?

Interview Problem

- what is the difference between `range()` and `xrange()` functions?

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' in green rounded rectangles, followed by a 'Workbench' button, a help icon (a question mark), and a settings gear icon. Below these are two tabs: 'WORK AREA' (highlighted in blue) and 'SOLUTIONS'. A 'Content Support' button is located above a large text area. The main text area contains the following problem statement:

Given a variable temps that refers to a list, all of whose elements refer to values of type float, representing temperature data, compute the average temperature and assign it to a variable named avg_temp. Besides temps and avg_temp, you may use two other variables -- k and total.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons labeled 'PREV' and 'NEXT' with arrows, followed by the title 'Workbench'. To the right of the title are two icons: a question mark and a gear. Below the title, the text 'Exercise 51211 —' is displayed. Underneath this, there are two tabs: 'WORK AREA' and 'SOLUTIONS', with 'WORK AREA' being the active tab. In the center of the main area, there is a green button labeled 'Content Support'. The main content area contains the following text and list:

We informally define the term **corresponding element** as follows:

- The first element and the last element of a list are corresponding elements.
- Similarly, the second element and the element just before the last element are corresponding elements.
- The third element and the element just before the element just before the last element are corresponding elements -- and so on.

Given that the variable `a` is associated with a list, write an **expression** for the corresponding element of `a[i]`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, and 'Workbench' in the center. To the right of 'Workbench' are two icons: a question mark and a gear. Below these buttons, the title 'Exercise 51212 –' is displayed. Underneath the title are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located below the tabs. The main content area contains the following text: 'Given that a refers to a list, write the necessary code to reverse the elements of the list.'

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a 'Workbench' interface for a programming exercise. At the top, there are 'PREV' and 'NEXT' buttons, the title 'Workbench', and a help/gear icon. Below the title, it says 'Exercise 51213 —'. There are two tabs: 'WORK AREA' (which is selected) and 'SOLUTIONS'. In the 'WORK AREA' section, there is a 'Content Support' button. The main content area contains the following text:

Given:

- a variable `current_members` that refers to a list, and
- a variable `member_id` that has been defined.

Write some code that assigns True to a variable `is_a_member` if the value associated with `member_id` can be found in the list associated with `current_members`, but that otherwise assigns False to `is_a_member`. Use only `current_members`, `member_id`, and `is_a_member`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a programming exercise interface. At the top, there are navigation buttons: 'PREV' and 'NEXT' in green rounded rectangles, followed by a 'Workbench' button in blue, and a help icon with a question mark and gear symbol. Below these are two tabs: 'WORK AREA' (highlighted in blue) and 'SOLUTIONS'. A 'Content Support' button is located above the main area. The main content area contains the following text:

You are given a variable `zipcode_list` that refers to a list.

Write some code that assigns True to a variable `duplicates` if there are two adjacent elements in the list that have the **same value**, but that otherwise assigns False to `duplicates` otherwise. In order to accomplish this, you may, if you wish, use one other variable, `k`.

Use only `k`, `zipcode_list`, and `duplicates`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' in green rounded rectangles, followed by a 'Workbench' button in blue, and a help icon ('?') and settings icon ('⚙️'). Below these is the title 'Exercise 51215 —'. Underneath the title are two tabs: 'WORK AREA' and 'SOLUTIONS', with 'WORK AREA' being the active tab. A 'Content Support' button is located above the main content area. The main content area contains the following text:

You are given a variable named `zipcode_list` that has been defined and refers to a list of postal codes.

Write some code that assigns True to duplicates if **any** two elements in the list have the **same value**, but that otherwise assigns False to duplicates. You may, if you wish, use two other variables, `j` and `k`.

Use only `j`, `k`, `zipcode_list`, and `duplicates`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons labeled 'PREV' and 'NEXT' with arrows, followed by the word 'Workbench'. To the right are help and settings icons. Below the title 'Exercise 51216' are two tabs: 'WORK AREA' and 'SOLUTIONS', with 'WORK AREA' being the active tab. A green button labeled 'Content Support' is located in the top right of the main area. The main content area contains the following text:

Given:

- a variable named incompletes that refers to a list of student ids, and
- a variable student_id

Write some code that **counts** the number of times the value associated with student_id appears in the list associated with incompletes and **assigns** this value to number_of_incompletes. You may use, if you wish, an additional variable, k.

You may use only k, incompletes, student_id, and number_of_incompletes.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

Workbench

PREV NEXT

Exercise 51219 —

WORK AREA SOLUTIONS

Content Support

A list named `parking_tickets` has been defined to be the number of parking tickets given out by the city police each day since the beginning of the current year. (Thus, the first element of the list contains the number of tickets given on January 1; the last element contains the number of tickets given today.)

Write some code that associates `most_tickets` with the **largest** value found in `parking_tickets`. You may, if you wish, use one additional variable, `k`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, and 'Workbench' in the center. To the right of 'Workbench' are two green icons: a question mark and a gear. Below these buttons, the text 'Exercise 51256 —' is displayed. Underneath this, there are two tabs: 'WORK AREA' and 'SOLUTIONS'. The 'WORK AREA' tab is currently selected. In the main content area, there is a button labeled 'Content Support'. The main text area contains the following instruction: 'Reverse the list associated with the variable words.'

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The image shows a screenshot of a programming exercise interface. At the top, there is a toolbar with buttons for 'PREV' and 'NEXT', a 'Workbench' button, and a help/gear icon. Below the toolbar, the text 'Exercise 51263 —' is displayed. Underneath this, there are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located above a large text box. The text box contains the following instruction:

Associate the sum of the non-negative values in the list numbers with the variable sum.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a 'Workbench' interface for a programming exercise. At the top, there are 'PREV' and 'NEXT' buttons, followed by the title 'Workbench' and a help icon. Below the title is the exercise identifier 'Exercise 51275'. The interface includes tabs for 'WORK AREA' and 'SOLUTIONS', and a 'Content Support' button. The main content area contains the following text:

Given the lists `list1` and `list2` that are of the same length, create a new list consisting of the first element of `list1` followed by the first element of `list2`, followed by the second element of `list1`, followed by the second element of `list2`, and so on (in other words the new list should consist of alternating elements of `list1` and `list2`). For example, if `list1` contained `[1, 2, 3]` and `list2` contained `[4, 5, 6]`, then the new list should contain `[1, 4, 2, 5, 3, 6]`. Associate the new list with the variable `list3`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons labeled 'PREV' and 'NEXT' with circular arrows, followed by the word 'Workbench'. To the right of 'Workbench' are two icons: a question mark and a gear. Below this header, the title 'Exercise 51276 —' is displayed. Underneath the title, there are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located below the tabs. The main content area contains a text box with the following instruction:

Given the lists `list1` and `list2` that are of the same length, create a new list consisting of the last element of `list1` followed by the last element of `list2`, followed by the second to last element of `list1`, followed by the second to last element of `list2`, and so on (in other words the new list should consist of alternating elements of the reverse of `list1` and `list2`). For example, if `list1` contained [1, 2, 3] and `list2` contained [4, 5, 6], then the new list should contain [3, 6, 2, 5, 1, 4]. Associate the new list with the variable `list3`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a help icon ('?') and settings icon ('⚙️') on the right. Below the navigation is the title 'Exercise 51277'. Underneath the title are two tabs: 'WORK AREA' (highlighted in blue) and 'SOLUTIONS'. A green button labeled 'Content Support' is located below the tabs. The main content area contains the following text:

Given the lists `list1` and `list2`, not necessarily of the same length, create a new list consisting of alternating elements of `list1` and `list2` (that is, the first element of `list1` followed by the first element of `list2`, followed by the second element of `list1`, followed by the second element of `list2`, and so on). Once the end of either list is reached, no additional elements are added. For example, if `list1` contained `[1, 2, 3]` and `list2` contained `[4, 5, 6, 7, 8]`, then the new list should contain `[1, 4, 2, 5, 3, 6]`. Associate the new list with the variable `list3`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a software interface for a programming exercise. At the top, there are navigation buttons labeled "PREV" and "NEXT", the title "Workbench", and a help icon. Below the title, it says "Exercise 51278 —". There are two tabs: "WORK AREA" (which is selected) and "SOLUTIONS". A "Content Support" button is located in the top right corner of the main content area. The main content area contains the following text:

Given the lists `list1` and `list2`, not necessarily of the same length, create a new list consisting of alternating elements of `list1` and `list2` (that is, the first element of `list1` followed by the first element of `list2`, followed by the second element of `list1`, followed by the second element of `list2`, and so on). Once the end of either list is reached, the remaining elements of the longer list is added to the end of the new list. For example, if `list1` contained [1, 2, 3] and `list2` contained [4, 5, 6, 7, 8], then the new list should contain [1, 4, 2, 5, 3, 6, 7, 8]. Associate the new list with the variable `list3`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons labeled 'PREV' and 'NEXT' with circular arrows, followed by the title 'Workbench' and icons for help ('?') and settings ('⚙️'). Below the title, the text 'Exercise 51284 —' is displayed. A horizontal bar contains two tabs: 'WORK AREA' (which is selected) and 'SOLUTIONS'. In the 'WORK AREA' section, there is a green button labeled 'Content Support'. The main content area contains the following text:

An *arithmetic progression* is a sequence of numbers in which the distance (or difference) between any two successive numbers is the same. This in the sequence 1, 3, 5, 7, ..., the distance is 2 while in the sequence 6, 12, 18, 24, ..., the distance is 6.

Given the positive integer distance and the non-negative integer n, create a list consisting of the arithmetic progression between (and including) 1 and n with a distance of distance. For example, if distance is 2 and n is 8, the list would be [1, 3, 5, 7].

Associate the list with the variable `arith_prog`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a help icon ('?') and settings icon ('⚙️') on the right. Below these are two tabs: 'WORK AREA' (highlighted in blue) and 'SOLUTIONS'. A 'Content Support' button is located at the bottom right of the main content area. The main content area contains the following text:

An *arithmetic progression* is a sequence of numbers in which the distance (or difference) between any two successive numbers is the same. This in the sequence 1, 3, 5, 7, ..., the distance is 2 while in the sequence 6, 12, 18, 24, ..., the distance is 6.

Given the positive integer distance and the integers m and n, create a list consisting of the arithmetic progression between (and including) m and n with a distance of distance (if m > n, the list should be empty.) For example, if distance is 2, m is 5, and n is 12, the list would be [5, 7, 9, 11].

Associate the list with the variable `arith_prog`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a software interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a help icon ('?') and settings icon ('⚙️') on the right. Below the title, it says 'Exercise 51287 —'. There are two tabs: 'WORK AREA' (selected) and 'SOLUTIONS'. A green button labeled 'Content Support' is located above the main content area. The main content area contains the following text:

A *geometric progression* is a sequence of numbers in which each value (after the first) is obtained by multiplying the previous value in the sequence by a fixed value called the *common ratio*. For example the sequence 3, 12, 48, 192, ... is a geometric progression in which the common ratio is 4.

Given the positive integer ratio greater than 1, and the non-negative integer n, create a list consisting of the geometric progression of numbers between (and including) 1 and n with a common ratio of ratio. For example, if ratio is 2 and n is 8, the list would be [1, 2, 4, 8].

Associate the list with the variable geom_prog.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT'. To the right of these are links for 'Workbench', '?', and a gear icon. Below the navigation is the title 'Exercise 51288 —'. Underneath the title are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located above the main content area. The main content area contains the following text:

In the following sequence, each number (except the first two) is the sum of the previous two number: 0, 1, 1, 2, 3, 5, 8, 13, This sequence is known as the *Fibonacci sequence*.

Given the positive integer n create a list consisting of the portion of the Fibonacci sequence less than or equal to n . For example, if n is 6, then the list would be [0, 1, 1, 2, 3, 5] and if n is 1, then the list would be [0, 1, 1].

Associate the list with the variable `fib`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a software interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT'. To the right of these is the title 'Workbench' in red. Further to the right are two green icons: a question mark and a gear. Below the title, the text 'Exercise 51289 —' is displayed. Underneath this, there are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located on the right side of the main content area. The main content area contains the following text:

In the following sequence, each number (except the first two) is the sum of the previous two number: 0, 1, 1, 2, 3, 5, 8, 13, This sequence is known as the *Fibonacci sequence*.

Given the positive integers m and n (with $m < n$) create a list consisting of the portion of the Fibonacci sequence greater than or equal to m and less than or equal to n. For example, if m is 3 and n is 6, then the list would be [3, 5] and if m is 2 and n is 20, then the list would be [2, 3, 5, 8, 13].

Associate the list with the variable fib.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, and 'Workbench' with a gear icon on the right. Below these are two tabs: 'WORK AREA' (highlighted in blue) and 'SOLUTIONS'. A green button labeled 'Content Support' is located above the main work area. The main area contains the following text:

Sort the list, `lst` (use the `list sort` method).

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT'. To the right of these is the title 'Workbench'. Further right are icons for help ('?') and settings ('⚙️'). Below the title, the text 'Exercise 51295 —' is displayed. Underneath this, there are two tabs: 'WORK AREA' and 'SOLUTIONS'. In the bottom right corner of the main area, there is a button labeled 'Content Support'. The main content area contains the following text:

Given the lists, `lst1` and `lst2`, create a new sorted list consisting of all the elements of `lst1` that also appears in `lst2`. For example, if `lst1` is `[4, 3, 2, 6, 2]` and `lst2` is `[1, 2, 4]`, then the new list would be `[2, 2, 4]`. Note that duplicate elements in `lst1` that appear in `lst2` are also duplicated in the new list. Associate the new list with the variable `new_list`, and don't forget to sort the new list.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a software interface for a programming exercise. At the top, there are navigation buttons labeled "PREV" and "NEXT" in green circles, followed by the word "Workbench". To the right of "Workbench" are two icons: a question mark and a gear. Below the top bar, the text "Exercise 51296—" is displayed. Underneath this, there are two tabs: "WORK AREA" and "SOLUTIONS", with "WORK AREA" being the active tab. In the main content area, there is a "Content Support" button at the top right. The main text area contains the following problem statement:

Given the lists, `lst1` and `lst2`, create a new sorted list consisting of all the elements of `lst1` that do not appear in `lst2` together with all the elements of `lst2` that do not appear in `lst1`. For example, if `lst1` is `[4, 3, 2, 6, 2]` and `lst2` is `[1, 2, 4, 1, 5]`, then the new list would be `[1, 1, 3, 5, 6]`. Note that duplicate elements are also duplicated in the new list. Associate the new list with the variable `new_list`, and don't forget to sort the new list.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a question mark icon and a gear icon on the right. Below these are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located in the middle right. The main area contains the following text:

Given a list named `play_list`, write
an **expression** whose value is the
length of `play_list`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, a 'Workbench' tab in the center, and a help icon ('?') and settings icon ('⚙️') on the right. Below the navigation is the title 'Exercise 51603 —'. Underneath the title are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located below the tabs. The main area contains the following text:

Given a **variable** named `plist` that refers to a **list**, write a **statement** that adds another **element**, 5 to the end of the list.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' in green rounded rectangles, followed by a 'Workbench' button with blue and red text, and a help icon ('?') and settings icon ('⚙️'). Below these are two tabs: 'WORK AREA' and 'SOLUTIONS'. A 'Content Support' button is located in the top right corner of the main area. The main content area contains the following text:

Given a list named `alist`, write an **expression** that removes the last element of `alist`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, and 'Workbench' in the center. To the right of 'Workbench' are two icons: a question mark and a gear. Below these buttons, the title 'Exercise 51606 —' is displayed. Underneath the title, there are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located below the tabs. The main area contains the following text:

Given a variable `alist` that refers to a list, remove the list's last element and associates its value with a variable `k`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a question mark icon and a gear icon on the right. Below these is the title 'Exercise 51607 –'. Underneath the title are two buttons: 'WORK AREA' and 'SOLUTIONS'. A 'Content Support' button is located below the 'SOLUTIONS' button. The main area contains the following text:

Given that `alist` has been defined to be a non-empty list (that is with at least one element), write a **statement** that removes its first element.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' in green rounded rectangles, followed by a 'Workbench' button with blue and red text, and a help/gear icon. Below these are two tabs: 'WORK AREA' and 'SOLUTIONS'. A 'Content Support' button is located above a large text area. The main text area contains the following question:

Given that `alist` has been defined to be a list with at least 4 elements, write a **statement** that removes its 4th element.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a question mark icon and a gear icon on the right. Below these are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located below the tabs. The main area contains the following text:

Given that `k` refers to a non-negative int and that `alist` has been defined to be a list with at least `k+1` elements, write a **statement** that removes the element at index `k`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a help icon ('?') and settings icon ('⚙️') on the right. Below these is the title 'Exercise 51610 —'. Underneath the title is a tab bar with 'WORK AREA' and 'SOLUTIONS' buttons. A green button labeled 'Content Support' is located below the tabs. The main area contains the following text:

Given that `play_list` has been defined to be a list, write a **statement** that sorts the list.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a help icon ('?') and settings icon ('⚙️') on the right. Below these are two tabs: 'WORK AREA' (highlighted in blue) and 'SOLUTIONS'. A green button labeled 'Content Support' is located above the main content area. The main content area contains the following text:

Given that `plist1` and `plist2` both refer to lists, write an **expression** that evaluates to a list that **is the concatenation of `plist1` and `plist2`**. Do not modify `plist1` or `plist2`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' in green rounded rectangles, followed by a 'Workbench' button in blue text. To the right of 'Workbench' are two icons: a question mark and a gear. Below these buttons, the text 'Exercise 51612 —' is displayed. Underneath this, there are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located below the tabs. The main content area contains the following text:

Given that plist1 and plist2 both refer to lists, write a **statement** that defines plist3 as a new list that is the concatenation of plist1 and plist2. Do not modify plist1 or plist2.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a help icon with a question mark and a gear icon on the right. Below these are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located above the main content area. The main content area contains the following text:

Given that `play_list` has been defined to be a list, write an **expression** that evaluates to a new list containing the elements at index 0 through index 4 `play_list`. Do not modify `play_list`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' in green rounded rectangles, followed by a 'Workbench' button in blue, and a help icon with a question mark and a gear icon. Below these are two tabs: 'WORK AREA' and 'SOLUTIONS'. A 'Content Support' button is located above a large text area. The main text area contains the following problem statement:

Given that `k` and `j` each refer to a non-negative int and that `play_list` has been defined to be a list with at least `j+1` elements, write an expression that evaluates to a new list containing all the elements from the one at index `k` through the one at index `j` of list `play_list`. Do not modify `play_list`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' (with a left arrow), 'NEXT' (with a right arrow), 'Workbench' (in blue text), a help icon ('?'), and a settings icon ('gear'). Below these are two tabs: 'WORK AREA' (selected) and 'SOLUTIONS'. A 'Content Support' button is located at the bottom of the tabs. The main area contains the following text:

Given that `k` and `j` each refer to a non-negative int and that `play_list` has been defined to be a list with at least `j` elements, write an expression that evaluates to a new list containing all the elements from the one at index `k` through the one at index `j-1` of list `play_list`. Do not modify `play_list`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' in green rounded rectangles, followed by a 'Workbench' button in blue, and a help icon ('?') and settings icon ('⚙️') in green. Below these are two tabs: 'WORK AREA' and 'SOLUTIONS'. A 'Content Support' button is located in the top right of the main area. The main content area contains the following text:

Given that `play_list` has been defined to be a list, write a **statement** that makes the first 3 elements of `play_list` be "`spam`", "`eggs`" and "`vikings`" (in that order).

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' in green rounded rectangles, a 'Workbench' button in blue, and a help icon ('?') and settings icon ('⚙️') in green. Below these is a title 'Exercise 51617 —'. Underneath the title are two tabs: 'WORK AREA' and 'SOLUTIONS', with 'WORK AREA' being the active tab. A 'Content Support' button is located above a scrollable area containing the exercise text. The text reads: 'Given that L1 and L2 both refer to lists, write a **statement** that replaces the elements in L1 from index 5 through (and including) index 8 with all the elements of L2.'

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' in green rounded rectangles, followed by a 'Workbench' button in blue, and a help/gear icon. Below these are two tabs: 'WORK AREA' and 'SOLUTIONS'. A 'Content Support' button is located above a scrollable text area. The text area contains the following question:

Given that `worst_offenders` has been defined as a list with at least 6 elements, write a `statement` that defines `lesser_offenders` to be a new list that contains all the elements from index 5 of `worst_offenders` and beyond. Do not modify `worst_offenders`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a programming exercise interface. At the top, there are navigation buttons: 'PREV' and 'NEXT'. To the right of these are links for 'Workbench', '?', and a gear icon. Below the navigation is the title 'Exercise 51619 –'. Underneath the title are two tabs: 'WORK AREA' and 'SOLUTIONS'. A 'Content Support' button is located above the main content area. The main content area contains the following text:

Given that `k` refers to an `int` that is non-negative and that `plist1` has been defined to be a list with at least `k+1` elements, write a `statement` that defines `plist2` to be a new list that contains all the elements from index `k` of `plist1` and beyond. Do not modify `plist1`.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a question mark icon and gear icon on the right. Below these are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located above the main content area. The main content area contains the following text:

Given that plist has been defined to be a list, write an expression that evaluates to True if 3 is an element of plist.

Programming Exercise

Worksheet

this page is intentionally left blank

Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' in green rounded rectangles, followed by a 'Workbench' button in blue and red, and a help/symbol button. Below these are two tabs: 'WORK AREA' and 'SOLUTIONS'. A 'Content Support' button is located above the main content area. The main content area contains the following text:

Given that `k` refers to an `int` and that `play_list` has been defined to be a `list`, write a **expression** that evaluates to `True` if the value associated with `k` is an element of `play_list`.

Programming Exercise

Worksheet

this page is intentionally left blank