

11.1 Lists

Survey

©zyBooks 09/27/22 12:00 469702

Steven Cameron

WGUC859v4

The following questions are part of a zyBooks survey to help us improve our content so we can offer the best experience for students. The survey can be taken anonymously and takes just 3-5 minutes. Please take a short moment to answer by clicking the following link.

Link: [Student survey](#)

List basics

The **list** object type is one of the most important and often used types in a Python program. A list is a **container**, which is an object that groups related objects together. A list is also a sequence; thus, the contained objects maintain a left-to-right positional ordering. Elements of the list can be accessed via indexing operations that specify the position of the desired element in the list. Each element in a list can be a different type such as strings, integers, floats, or even other lists.

The animation below illustrates how a list is created using brackets [] around the list elements. The animation also shows how a list object contains references to the contained objects.

PARTICIPATION
ACTIVITY

11.1.1: Lists contain references to other objects.



Animation content:

undefined

Animation captions:

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

1. The user creates a new list.
2. The interpreter creates a new object for each list element.
3. 'my_list' holds references to objects in list.

A list can also be created using the built-in `list()` function. The **list()** function accepts a single iterable object argument, such as a string, list, or tuple, and returns a new list object. Ex: `list('abc')`

creates a new list with the elements [a', 'b', 'c'].

Accessing list elements

An **index** is a zero-based integer matching to a specific position in the list's sequence of elements. A programmer can reference a specific element in a list by providing an index. Ex: `my_list[4]` uses an integer to access the element at index 4 (the 5th element) of `my_list`.

©zyBooks 09/27/22 12:00 469702

Steven Cameron

WGUC859v4

An index can also be an expression, as long as that expression evaluates into an integer. Replacing the index with an integer variable, such as in `my_list[i]`, allows the programmer to quickly and easily lookup the $(i+1)$ th element in a list.

zyDE 11.1.1: List's ith element can be directly accessed using [i]: Oldest people program.

Consider the following program that allows a user to print the age of the Nth oldest k person to have ever lived. Note: The ages are in a list sorted from oldest to youngest.

1. Modify the program to print the correct ordinal number ("1st", "2nd", "3rd", "4th", "5th" of "1th", "2th", "3th", "4th", "5th").
2. For the oldest person, remove the ordinal number (1st) from the print statement to "oldest person lived 122 years".

Reminder: List indices begin at 0, not 1, thus the print statement uses `oldest_people[nth_person-1]`, to access the nth_person element (element 1 at index 0 2 at index 1, etc.).

```
Load default template...  
1 oldest_people = [122, 119, 117, 117, 11  
2  
3 nth_person = int(input('Enter N (1-5):  
4  
5 if (nth_person >= 1) and (nth_person <=  
6     print('The {}th oldest person lived  
7         .format(nth_person, oldest_peop  
8 |
```

3

Run

©zyBooks 09/27/22 12:00 469702

Steven Cameron

WGUC859v4

The program can quickly access the Nth oldest person's age using `oldest_people[nth_person-1]`. Note that the index is `nth_person-1` rather than just `nth_person` because a list's indices start at 0, so the first age is at index 0, the second at index 1, etc.

A list's index must be an integer type. The index cannot be a floating-point type, even if the value is 0.0, 1.0, etc.

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4



PARTICIPATION
ACTIVITY

11.1.2: List indices.

Given the following code, what is the output of each code segment?

```
animals = ['cat', 'dog', 'bird', 'raptor']
print(animals[0])
```

1) `print(animals[0])`

Check

[Show answer](#)



2) `print(animals[2])`

Check

[Show answer](#)



3) `print(animals[0 + 1])`

Check

[Show answer](#)



4)

```
i = 3
print(animals[i])
```

Check

[Show answer](#)



©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Modifying a list and common list operations

Unlike the string sequence type, a list is **mutable** and is thus able to grow and shrink without the program having to replace the entire list with an updated copy. Such growing and shrinking capability is called ***in-place modification***. The highlighted lines in the list below indicate ways to perform an in-place modification of the list:

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Table 11.1.1: Some common list operations.

Operation	Description	Example code	Example output
my_list = [1, 2, 3]	Creates a list.	my_list = [1, 2, 3] print(my_list)	[1, 2, 3]
list(iter)	Creates a list.	my_list = list('123') print(my_list)	['1', '2', '3']
my_list[index]	Get an element from a list.	my_list = [1, 2, 3] print(my_list[1])	2
my_list[start:end]	Get a new list containing some of another list's elements.	my_list = [1, 2, 3] print(my_list[1:3])	[2, 3]
my_list1 + my_list2	Get a new list with elements of my_list2 added to end of my_list1.	my_list = [1, 2] + [3] print(my_list)	[1, 2, 3]
my_list[i] = x	Change the value of the ith element in-place.	my_list = [1, 2, 3] my_list[2] = 9 print(my_list)	[1, 2, 9]
my_list[len(my_list):] = [x]	Add the elements in [x] to the end of my_list. The append(x) method (explained in another section) may be preferred for clarity.	my_list = [1, 2, 3] my_list[len(my_list):] = [9] print(my_list)	[1, 2, 3, 9]
del my_list[i]	Delete an element from a list.	my_list = [1, 2, 3] del my_list[1] print(my_list)	[1, 3]

Some of the operations in the table might be familiar to the reader because they are sequence type operations also supported by strings. The dark-shaded rows highlight in-place modification

operations.

The below animation illustrates how a program can use in-place modifications to modify the contents of a list.

**PARTICIPATION
ACTIVITY****11.1.3: In-place modification of a list.**

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4**Animation captions:**

1. A list, `my_list`, is created with the chars '`h`', '`e`', '`l`', '`l`', and '`o`'.
2. Characters '`w`', '`o`', '`r`', '`l`', '`d`', and '`u`' are added to the end of `my_list`.
3. Index 11 is changed to '`!`' character.
4. Element at index 5 is removed from `my_list`.

The difference between in-place modification of a list and an operation that creates an entirely new list is important. In-place modification affects any variable that references the list, and thus can have unintended side-effects. Consider the following code in which the variables `your_teams` and `my_teams` reference the same list (via the assignment `your_teams = my_teams`). If either `your_teams` or `my_teams` modifies an element of the list, then the change is reflected in the other variable as well.

The below Python Tutor tool executes a Python program and visually shows the objects and variables of a program. The tool shows names of variables on the left, with arrows connecting to bound objects on the right. Note that the tool does not show each number or string character as unique objects to improve clarity. The Python Tutor tool is available at www.pythontutor.com.

**PARTICIPATION
ACTIVITY****11.1.4: In-place modification of a list.**

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

In the above example, changing the elements of `my_teams` also affects the contents of `your_teams`. The change occurs because `my_teams` and `your_teams` are bound to the same list object. The code `my_teams[1] = 'Lakers'` modifies the element in position 1 of the shared list object, thus changing the value of both `my_teams[1]` and `your_teams[1]`.

The programmer of the above example probably meant to only change `my_teams`. The correct approach would have been to create a *copy* of the list instead. One simple method to create a copy is to use slice notation with no start or end indices, as in `your_teams = my_teams[:]`.

PARTICIPATION
ACTIVITY

11.1.5: In-place modification of a copy of a list.



©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

On the other hand, assigning a variable with an element of an existing list, and then reassigning that variable with a different value does not change the list.

PARTICIPATION
ACTIVITY

11.1.6: List indexing.



Animation content:

undefined

Animation captions:

1. Indexing operation changes list elements.
2. The list does not change when fav_color is updated.

To change the value in the list above, the programmer would have to do an in-place modification operation, such as `colors[1] = 'orange'`.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

PARTICIPATION
ACTIVITY

11.1.7: List basics.



- 1) A program can modify the elements of an existing list.



- True
- 2) The size of a list is determined when the list is created and cannot change.



- True
- False
- 3) All elements of a list must have the same type.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

- True
- False
- 4) The statement `del my_list[2]` produces a new list without the element in position 2.



- True
- False
- 5) The statement `my_list1 + my_list2` produces a new list.



CHALLENGE ACTIVITY

11.1.1: Enter the output for the list.



334598.939404.qx3zqy7

Start

Type the program's output

```
user_values = [2, 5, 7]
print(user_values)
```

[2, 5, 7]

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

1

2

3

4

Check

Next

**CHALLENGE
ACTIVITY**

11.1.2: Modify a list.



Modify short_names by deleting the first element and changing the last element to Joe.

Sample output with input: 'Gertrude Sam Ann Joseph'

```
[ 'Sam', 'Ann', 'Joe' ]
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

334598.939404.qx3zqy7

```
1 user_input = input()
2 short_names = user_input.split()
3
4 ''' Your solution goes here '''
5
6 print(short_names)
```

Run

View your last submission ▾

11.2 List methods

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Common list methods

A **list method** can perform a useful operation on a list such as adding or removing elements, sorting, reversing, etc.

The table below shows the available list methods. Many of the methods perform in-place modification of the list contents — a programmer should be aware that a method that modifies the list in-place changes the underlying list object, and thus may affect the value of a variable that

references the object.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Table 11.2.1: Available list methods.

List method	Description	Code example	Final my_list value
©zyBooks 09/27/22 12:00 469702 Steven Cameron WGUC859v4			
Adding elements			
list.append(x)	Add an item to the end of list.	<code>my_list = [5, 8] my_list.append(16)</code>	[5, 8, 16]
list.extend([x])	Add all items in [x] to list.	<code>my_list = [5, 8] my_list.extend([4, 12])</code>	[5, 8, 4, 12]
list.insert(i, x)	Insert x into list <i>before</i> position i.	<code>my_list = [5, 8] my_list.insert(1, 1.7)</code>	[5, 1.7, 8]

Removing elements			
list.remove(x)	Remove first item from list with value x.	<code>my_list = [5, 8, 14] my_list.remove(8)</code>	[5, 14]
list.pop()	Remove and return last item in list.	<code>my_list = [5, 8, 14] val = my_list.pop()</code>	[5, 8] val is 14
list.pop(i)	Remove and return item at position i in list.	<code>my_list = [5, 8, 14] val = my_list.pop(0)</code>	[8, 14] val is 5

Modifying elements			
list.sort()	Sort the items of list in-place.	<code>my_list = [14, 5, 8] my_list.sort()</code>	[5, 8, 14]
list.reverse()	Reverse the elements of list in-place	<code>my_list = [14, 5, 8] my_list.reverse()</code>	[8, 5, 14]

In-place modification			
Miscellaneous			
list.index(x)	Return index of first item in list with value x.	<pre>my_list = [5, 8, 14] print(my_list.index(14))</pre>	Prints "2"
list.count(x)	Count the number of times value x is in list.	<pre>my_list = [5, 8, 5, 5, 14] print(my_list.count(5))</pre>	Prints "3"

Good practice is to use list methods to add and delete list elements, rather than alternative add/delete approaches. Alternative approaches include syntax such as

`my_list[len(my_list):] = [val]` to add to a list, or `del my_list[0]` to remove from a list. Generally, using a list method yields more readable code.

The `list.sort()` and `list.reverse()` methods rearrange a list element's ordering, performing in-place modification of the list.

The `list.index()` and `list.count()` return information about the list and do not modify the list.

The below interactive tool shows a few of the list methods in action:

PARTICIPATION ACTIVITY

11.2.1: In-place modification using list methods.

**Animation content:**

undefined

Animation captions:

1. `vals` is a list containing elements 1, 4, and 16.
2. The statement `vals.append(9)` appends element 9 to the end of the list.
3. The statement `vals.insert(2, 18)` inserts element 18 into position 2 of the list.
4. The statement `vals.pop()` removes the last element, 9, from the list.
5. The statement `vals.remove(4)` removes the first instance of element 4 from the list.
6. The statement `vals.remove(55)` removes the first instance of element 55 from the list. The list does not contain the element 55 so `vals` is the same.

zyDE 11.2.1: Amusement park ride reservation system.

The following (unfinished) program implements a digital line queuing system for an amusement park ride. The system allows a rider to reserve a place in line without actually having to wait. The rider simply enters a name into a program to reserve a place. Riders who purchase a VIP pass get to skip past the common riders up to the last VIP rider in line before they board the ride first. (Considering the average wait time for a Disneyland ride is **about 45 minutes**, this might be a useful program.) For this system, an employee manually selects which rider to dispatch when the ride is dispatched (thus removing the next riders from the front of the line).

Complete the following program, as described above. Once finished, add the following commands:

- The rider can enter a name to find the current position in line. (Hint: Use the `list.index()` method.)
- The rider can enter a name to remove the rider from the line.

[Load default template](#)

```
1 riders_per_ride = 3 # Num riders per ride to dispatch
2
3 line = [] # The Line of riders
4 num_vips = 0 # Track number of VIPs at front of line
5
6 menu = ('(1) Reserve place in line.\n' # Add rider to line
7         '(2) Reserve place in VIP line.\n' # Add VIP
8         '(3) Dispatch riders.\n' # Dispatch next ride car
9         '(4) Print riders.\n'
10        '(5) Exit.\n\n')
11
12 user_input = input(menu).strip().lower()
13
14 while user_input != '5':
15     if user_input == '1': # Add rider
16         name = input('Enter name:').strip().lower()
17         print(name)
```

```
1
Frank
4
```

Run

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

**PARTICIPATION
ACTIVITY**

11.2.2: List methods.



- 1) What is the output of the following program?

```
temps = [65, 67, 72, 75]
temps.append(77)
print(temps[-1])
```

Check**Show answer**

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

- 2) What is the output of the following program?

```
actors = ['Pitt', 'Damon']
actors.insert(1, 'Affleck')
print(actors[0], actors[1],
      actors[2])
```

Check**Show answer**

- 3) Write the simplest two statements that first sort my_list, then remove the largest value element from the list, using list methods.

```
my_list.sort()
my_list.pop()
```

Check**Show answer**

- 4) Write a statement that counts the number of elements of my_list that have the value 15.

```
len([x for x in my_list if x == 15])
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Check**Show answer****CHALLENGE
ACTIVITY**

11.2.1: Reverse sort of list.



Sort short_names in reverse alphabetic order.

Sample output with input: 'Jan Sam Ann Joe Tod'

```
[ 'Tod', 'Sam', 'Joe', 'Jan', 'Ann' ]
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

334598.939404.qx3zqy7

```
1 user_input = input()  
2 short_names = user_input.split()  
3  
4 ''' Your solution goes here '''  
5  
6 print(short_names)
```

Run

11.3 Iterating over a list

List iteration

A programmer commonly wants to access each element of a list. Thus, learning how to iterate through a list using a loop is critical.

©zyBooks 09/27/22 12:00 469702

Steven Cameron

Looping through a sequence such as a list is so common that Python supports a construct called a **for loop**, specifically for iteration purposes. The format of a for loop is shown below.

Figure 11.3.1: Iterating through a list.

```
for my_var in my_list:  
    # Loop body statements go  
    here
```

©zyBooks 09/27/22 12:00 469702

Steven Cameron

WGUC859v4

Each iteration of the loop creates a new variable by binding the next element of the list to the name my_var. The loop body statements execute during each iteration and can use the current value of my_var as necessary.¹

Programs commonly iterate through lists to determine some quantity about the list's items. Ex: The following program determines the value of the maximum even number in a list:

©zyBooks 09/27/22 12:00 469702

Steven Cameron

WGUC859v4

Figure 11.3.2: Iterating through a list example: Finding the maximum even number.

```
# User inputs string w/ numbers: '203 12 5 800 -10'
user_input = input('Enter numbers:')

tokens = user_input.split() # Split into separate strings

# Convert strings to integers
nums = []
for token in tokens:
    nums.append(int(token))

# Print each position and number
print() # Print a single newline
for index in range(len(nums)):
    value = nums[index]
    print('{0}: {1}'.format(index, value))

# Determine maximum even number
max_num = None
for num in nums:
    if (max_num == None) and (num % 2 == 0):
        # First even number found
        max_num = num
    elif (max_num != None) and (num > max_num) and (num % 2 == 0):
        # Larger even number found
        max_num = num

print('Max even #: ', max_num)
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

```
Enter numbers:3 5 23 -1 456 1 6 83
0: 3
1: 5
2: 23
3: -1
4: 456
5: 1
6: 6
7: 83
Max even #: 456
...
Enter numbers:-5 -10 -44 -2 -27 -9 -27 -9
0:-5
1:-10
2:-44
3:-2
4:-27
5:-9
6:-27
7:-9
Max even #: -2
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

If the user enters the numbers 7, -9, 55, 44, 20, -400, 0, 2, then the program will output **Max even #: 44**. The code uses three for loops. The first loop converts the strings obtained from the split() function into integers. The second loop prints each of the entered numbers. Note that the first and second loops could easily be combined into a single loop, but the example uses two loops

for clarity. The third loop evaluates each of the list elements to find the maximum even number.

Before entering the first loop, the program creates the list `nums` as an empty list with the statement `nums = []`. The program then appends items to the list inside the first loop. Omitting the initial empty list creation would cause an error when the `nums.append()` function is called, because `nums` would not actually exist yet.

The main idea of the code is to use a variable `max_num` to maintain the largest value seen so far as the program iterates through the list. During each iteration, if the list's current element value is even and larger than `max_num` so far, then the program assigns `max_num` with current value. Using a variable to track a value while iterating over a list is very common behavior.

PARTICIPATION ACTIVITY

11.3.1: Using a variable to keep track of a value while iterating over a list.

**Animation content:**

undefined

Animation captions:

1. Loop iterates over all elements of list `nums`.
2. Only larger even numbers update the value of `max_num`.
3. Odd numbers, or numbers smaller than `max_num`, are ignored.
4. When the loop ends, `max_num` is set to the largest even number 456.

A logic error in the above program would be to set `max_even` initially to 0, because 0 is not in fact the largest value seen so far. This would result in incorrect output (of 0) if the user entered all negative numbers. Instead, the program sets `max_even` to None.

PARTICIPATION ACTIVITY

11.3.2: List iteration.



Fill in the missing field to complete the program.

- 1) Count how many odd numbers (`cnt_odd`) there are.

`cnt_odd =`

```
for i in num:  
    if i % 2 == 1:  
        cnt_odd += 1
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Check**Show answer**



- 2) Count how many negative numbers (cnt_neg) there are.

```
cnt_neg = 0
for i in num:
    if i < 0:
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Check**Show answer**

- 3) Determine the number of elements in the list that are divisible by 10. (Hint: the number x is divisible by 10 if $x \% 10$ is 0.)

```
div_ten = 0
for i in num:
    if
        :
    div_ten += 1
```

**Check****Show answer**

IndexError and enumerate()

A common error is to try to access a list with an index that is out of the list's index range, e.g., to try to access `my_list[8]` when `my_list`'s valid indices are 0-7. Accessing an index that is out of range causes the program to automatically abort execution and generate an **IndexError**. Ex: For a list `my_list` containing 8 elements, the statement `my_list[10] = 42` produces output similar to:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

Iterating through a list for various purposes is an extremely important programming skill to master.

Steven Cameron
WGUC859v4

zyDE 11.3.1: Iterating through a list example: Finding the sum of a list's elements.

Here is another example computing the sum of a list of integers. Note that the code is somewhat different than the code computing the max even value. For computing the program initializes a variable sum to 0, then simply adds the current iteration's list element value to that sum.

Run the program below and observe the output. Next, modify the program to calculate the following:

- Compute the average, as well as the sum. Hint: You don't actually have to change the loop, but rather change the printed value.
- Print each number that is greater than 21.

```
Load default template...  
  
1 # User inputs string w/ numbers: '20  
2 user_input = input('Enter numbers: '  
3  
4 tokens = user_input.split() # Split  
5  
6 # Convert strings to integers  
7 print()  
8 nums = []  
9 for pos, token in enumerate(tokens):  
10    nums.append(int(token))  
11    print('{}: {}'.format(pos, token)  
12  
13 sum = 0  
14 for num in nums:  
15    sum += num  
16  
17 print('Sum:', sum)
```

203 12 5 800 -10

Run

The built-in **enumerate()** function iterates over a list and provides an iteration counter. The program above uses the `enumerate()` function, which results in the variables `pos` and `token` being assigned the current loop iteration element's index and value, respectively. Thus, the first iteration of the loop assigns `pos` with 0 and `token` with the first user number; the second iteration assigns `pos` with 1 and `token` with the second user number; and so on.

Built-in functions that iterate over lists

Iterating through a list to find or calculate certain values like the minimum/maximum or sum is so common that Python provides built-in functions as shortcuts. Instead of writing a for loop and tracking a maximum value, or adding a sum, a programmer can use a statement such as

`max(my_list)` or `sum(my_list)` to quickly obtain the desired value.

Table 11.3.1: Built-in functions supporting list objects.

Function	Description	Example code	Example output
all(list)	True if every element in list is True ($\neq 0$), or if the list is empty.	<code>print(all([1, 2, 3]))</code> <code>print(all([0, 1, 2]))</code>	<code>True</code> <code>False</code>
any(list)	True if any element in the list is True.	<code>print(any([0, 2]))</code> <code>print(any([0, 0]))</code>	<code>True</code> <code>False</code>
max(list)	Get the maximum element in the list.	<code>print(max([-3, 5, 25]))</code>	<code>25</code>
min(list)	Get the minimum element in the list.	<code>print(min([-3, 5, 25]))</code>	<code>-3</code>
sum(list)	Get the sum of all elements in the list.	<code>print(sum([-3, 5, 25]))</code>	<code>27</code>

zyDE 11.3.2: Using built-in functions with lists.

Complete the following program using functions from the table above to find some statistics about basketball player Lebron James. The code below provides lists of various statistics for the years 2003-2013. Compute and print the following statistics:

- Total career points
- Average points per game
- Years of the highest and lowest scoring season

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Use loops where appropriate.

[Load default template...](#)

[Run](#)

```
1
2 #Lebron James: Statistics for 2003/2
3 games_played = [79, 80, 79, 78, 75,
4 points = [1654, 2175, 2478, 2132, 22
5 assists = [460, 636, 814, 701, 771,
6 rebounds = [432, 588, 556, 526, 592,
7
8 # Print total points
9
10 # Print Average PPG
11
12 # Print best scoring years (Ex: 2004)
13
14 # Print worst scoring years (Ex: 200
15
16
17
```

PARTICIPATION ACTIVITY

11.3.3: Lists and built-in functions.



Assume that `my_list` is `[0, 5, 10, 15]`.

- 1) What value is returned by
`sum(my_list)?`

[Check](#)

[Show answer](#)

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

- 2) What value is returned by
`max(my_list)?`



Check**Show answer**

- 3) What value is returned by
any(my_list)?

**Check****Show answer**

- 4) What value is returned by
all(my_list)?

**Check****Show answer**

- 5) What value is returned by
min(my_list)?

**Check****Show answer****CHALLENGE ACTIVITY**

11.3.1: Get user guesses.



Write a loop to populate the list user_guesses with a number of guesses. The variable num_guesses is the number of guesses the user will have, which is read first as an integer. Read each guess (an integer) one at a time using int(input()).

Sample output with input:

3
9
5
2

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

user_guesses: [9, 5, 2]

334598.939404.qx3zqy7

```
1 num_guesses = int(input())
2 user_guesses = []
3
4 ''' Your solution goes here '''
5
6 print('user_guesses:', user_guesses)
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Run**CHALLENGE ACTIVITY**

11.3.2: Sum extra credit.



Assign sum_extra with the total extra credit received given list test_grades. Iterate through the list with `for grade in test_grades:`. The code uses the Python `split()` method to split a string at each space into a list of string values and the `map()` function to convert each string value to an integer. Full credit is 100, so anything over 100 is extra credit.

Sample output for the given program with input: '101 83 107 90'

Sum extra: 8

(because $1 + 0 + 7 + 0$ is 8)

334598.939404.qx3zqy7

```
1 user_input = input()
2 test_grades = list(map(int, user_input.split())) # test_grades is an integer list
3
4
5 sum_extra = 0 # Initialize 0 before your loop
6
7 ''' Your solution goes here '''
8
9 print('Sum extra:', sum_extra)
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Run

View your last submission ▾

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4**CHALLENGE
ACTIVITY**

11.3.3: Hourly temperature reporting.



Write a loop to print all elements in hourly_temperature. Separate elements with a -> surrounded by spaces.

Sample output for the given program with input: '90 92 94 95'

90 -> 92 -> 94 -> 95

Note: 95 is followed by a space, then a newline.

334598.939404.qx3zqy7

```
1 user_input = input()
2 hourly_temperature = user_input.split()
3
4 ''' Your solution goes here '''
5
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4**Run**

View your last submission ▾

(*1) Actually, a for loop works on any iterable object. An iterable object is any object that can access each of its elements one at a time -- most sequences like lists, strings, and tuples are iterables. Thus, for loops are not specific to lists.

11.4 List games

The following activities can help one become comfortable with iterating through lists. Challenge yourself with these list games.

PARTICIPATION ACTIVITY

11.4.1: Find the maximum value in the list.

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4



If a new maximum value is seen, then click 'Store value'. Try again to get your best time.

Start**max**

X	X	X	X	X	X	X
---	---	---	---	---	---	---

Next value**Store value**

Time - Best time -

Clear best**PARTICIPATION ACTIVITY**

11.4.2: Negative value counting in list.



If a negative value is seen, then click 'Increment'. Try again to get your best time.

Start**Counter**

X	X	X	X	X	X	X
---	---	---	---	---	---	---

0

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4**Next value****Increment**

Time - Best time -

Clear best

PARTICIPATION ACTIVITY**11.4.3: Manually sorting largest value.**

Move the largest value to the right-most position of the list. If the larger of the two current values is on the left, then swap the values. Try again to get your best time.

Start

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

**Next value****Swap values**

Time - Best time -

Clear best

11.5 List nesting

Since a list can contain any type of object as an element, and a list is itself an object, a list can contain another list as an element. Such embedding of a list inside another list is known as **list nesting**. Ex: The code `my_list = [[5, 13], [50, 75, 100]]` creates a list with two elements that are each another list.

Figure 11.5.1: Multi-dimensional lists.

```
my_list = [[10, 20], [30, 40]]
print('First nested list:', my_list[0])
print('Second nested list:', my_list[1])
print('Element 0 of first nested list:',
      my_list[0][0])
```

First nested list: [10, 20]
Second nested list: [30, 40]
Element 0 of first nested list:
10

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

The program accesses elements of a nested list using syntax such as `my_list[0][0]`.

PARTICIPATION ACTIVITY**11.5.1: List nesting.**

Animation captions:

1. The nested lists can be accessed using a single access operation.
2. The elements of each nested list can be accessed using two indexing operations.

PARTICIPATION
ACTIVITY

11.5.2: List nesting.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4



- 1) Given the list `nums = [[10, 20, 30], [98, 99]]`, what does `nums[0][0]` evaluate to?

Check

[Show answer](#)

- 2) Given the list `nums = [[10, 20, 30], [98, 99]]`, what does `nums[1][1]` evaluate to?

Check

[Show answer](#)

- 3) Given the list `nums = [[10, 20, 30], [98, 99]]`, what does `nums[0]` evaluate to?

Check

[Show answer](#)

- 4) Create a nested list `nums` whose only element is the list `[21, 22, 23]`.

Check

[Show answer](#)



©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

A list is a single-dimensional sequence of items, like a series of times, data samples, daily

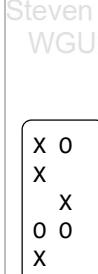
temperatures, etc. List nesting allows for a programmer to also create a **multi-dimensional data structure**, the simplest being a two-dimensional table, like a spreadsheet or tic-tac-toe board. The following code defines a two-dimensional table using nested lists:

Figure 11.5.2: Representing a tic-tac-toe board using nested lists.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

```
tic_tac_toe = [
    ['X', 'O', 'X'],
    [' ', 'X', ' '],
    ['O', 'O', 'X']
]

print(tic_tac_toe[0][0], tic_tac_toe[0][1], tic_tac_toe[0]
[2])
print(tic_tac_toe[1][0], tic_tac_toe[1][1], tic_tac_toe[1]
[2])
print(tic_tac_toe[2][0], tic_tac_toe[2][1],
tic_tac_toe[2][2])
```



The example above creates a variable `tic_tac_toe` that represents a 2-dimensional table with 3 rows and 3 columns, for $3 \times 3 = 9$ total table entries. Each row in the table is a nested list. Table entries can be accessed by specifying the desired row and column: `tic_tac_toe[1][1]` accesses the middle square in row 1, column 1 (starting from 0), which has a value of 'X'. The following animation illustrates:

PARTICIPATION
ACTIVITY

11.5.3: Two-dimensional list.



Animation captions:

1. New list object contains other lists as elements.
2. Elements accessed by [row][column].

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

zyDE 11.5.1: Two-dimensional list example: Driving distance between cities.

The following example illustrates the use of a two-dimensional list in a distance between cities example.

Run the following program, entering the text '1 2' as input to find the distance between Chicago. Try other pairs. Next, try modifying the program by adding a new city. Anchorage is 3400, 3571, and 4551 miles from Los Angeles, Chicago, and Boston, respectively.

Note that the styling of the nested list in this example makes use of indentation to clearly indicate the elements of each list -- the spacing does not affect how the interpreter evaluates the list contents.

```
Load default template...  
  
1 # direct driving distances between cities  
2 # 0: Boston    1: Chicago    2: Los Angeles  
3  
4 distances = [  
5     [  
6         0,  
7         960,    # Boston-Chicago  
8         2960 # Boston-Los Angeles  
9     ],  
10    [  
11        960,    # Chicago-Boston  
12        0,  
13        2011 # Chicago-Los Angeles  
14    ],  
15    [  
16        2960,    # Los Angeles-Boston  
17        2011,    # Los-Angeles-Chicago
```

1 2

Run

The level of nested lists is arbitrary. A programmer might create a three-dimensional list structure as follows:

Figure 11.5.3: The level of nested lists is arbitrary.

```
nested_table = [
    [
        [10, 0,
    55],
        [0, 4, 16]
    ],
    [
        [0, 0, 1],
        [1, 20, 2]
    ]
]
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

A number from the above three-dimensional list could be accessed using three indexing operations, as in `nested_table[1][1][1]`.

PARTICIPATION
ACTIVITY

11.5.4: Multi-dimensional lists.



Assume the following list has been created

```
scores = [
    [75, 100, 82, 76],
    [85, 98, 89, 99],
    [75, 82, 85, 5]
]
```

- 1) Write an indexing expression that gets the element from scores whose value is 100.



Check

[Show answer](#)

- 2) How many elements does scores contain? (The result of `len(scores)`)



Check

[Show answer](#)

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

As always with lists, a typical task is to iterate through the list elements. A programmer can access all of the elements of nested lists by using **nested for loops**. The first for loop iterates through the elements of the outer list (rows of a table), while the nested loop iterates through the inner list

elements (columns of a table). The code below defines a 3x3 table and iterates through each of the table entries:

PARTICIPATION ACTIVITY**11.5.5: Iterating over multi-dimensional lists.****Animation content:**

undefined

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Animation captions:

1. Each iteration row is assigned the next list element from currency. Each item in a row is printed in the inner loop.

The outer loop assigns the variable row with one of the list elements. The inner loop then iterates over the elements in that list. Ex: On the first iteration of the outer loop row is [1, 5, 10]. The inner loop then assigns cell 1 on the first iteration, 5 on the second iteration, and 10 on the last iteration.

Combining nested for loops with the enumerate() function gives easy access to the current row and column:

Figure 11.5.4: Iterating through multi-dimensional lists using enumerate().

```
currency = [
    [1, 5, 10],  # US Dollars
    [0.75, 3.77, 7.53],  #Euros
    [0.65, 3.25, 6.50]  # British pounds
]
for row_index, row in enumerate(currency):
    for column_index, item in enumerate(row):
        print('currency[{}][{}] is {:.2f}'.format(row_index,
            column_index, item))
```

```
currency[0][0] is
1.00
currency[0][1] is
5.00
currency[0][2] is
10.00
currency[1][0] is
0.75
currency[1][1] is
3.77
currency[1][2] is
7.53
currency[2][0] is
0.65
currency[2][1] is
3.25
currency[2][2] is
6.50
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

PARTICIPATION ACTIVITY**11.5.6: Find the error.**

The desired output and actual output of each program is given. Find the error in each program.

- 1) Desired output:

0	2	4	6
0	3	6	9
12			

Actual output:

[0, 2, 4, 6]	[0, 3, 6, 9, 12]
[0, 2, 4, 6]	[0, 3, 6, 9, 12]

```
nums = [  
    [0, 2, 4, 6],  
    [0, 3, 6, 9, 12]  
]
```

```
for n1  
in nums  
:  
    for n2  
in nums  
:  
        print(n2, end=' ')  
print()
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

- 2) Desired output:

X wins!

Actual output:

Cat's game!

```
tictactoe = [  
    ['X', 'O', 'O'],  
    ['O', 'O', 'X'],  
    ['X', 'X', 'X']  
]
```

```
# Check for 3 Xs in one row  
# (Doesn't check columns or diagonals)  
for row in tictactoe  
:  
    num_X_in_row = 0  
    for square in row  
    :  
        if square == 'X'  
        :  
            num_X_in_row += 1  
        if num_X_in_row == square  
        :  
            print('X wins!')  
            break  
    else:
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

```
print("Cat's game!")
```

**CHALLENGE
ACTIVITY****11.5.1: Print multiplication table.**

Print the two-dimensional list `mult_table` by row and column. On each line, each character is separated by a space. Hint: Use nested loops.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Sample output with input: '1 2 3,2 4 6,3 6 9':

1	2	3
2	4	6
3	6	9

334598.939404.qx3zqy7

```
1 user_input= input()
2 lines = user_input.split(',')
3
4 # This line uses a construct called a List comprehension, introduced elsewhere,
5 # to convert the input string into a two-dimensional list.
6 # Ex: 1 2, 2 4 is converted to [ [1, 2], [2, 4] ]
7
8 mult_table = [[int(num) for num in line.split()] for line in lines]
9
10 ''' Your solution goes here '''
11
```

Run

View your last submission ▾

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

11.6 List slicing

A programmer can use ***slice notation*** to read multiple elements from a list, creating a new list that contains only the desired elements. The programmer indicates the start and end positions of a range of elements to retrieve, as in `my_list[0:2]`. The 0 is the position of the first element to read,

and the 2 indicates last element. Every element between 0 and 2 from my_list will be in the new list. The end position, 2 in this case, is *not* included in the resulting list.

Figure 11.6.1: List slice notation.

```
boston_bruins = ['Tyler', 'Zdeno',
'Patrice']
print('Elements 0 and 1:',
boston_bruins[0:2])
print('Elements 1 and 2:',
boston_bruins[1:3])
```

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4

```
Elements 0 and 1: ['Tyler',
'Zdeno']
Elements 1 and 2: ['Zdeno',
'Patrice']
```

The slice `boston_bruins[0:2]` produces a new list containing the elements in positions 0 and 1: ['Tyler', 'Zdeno']. The end position is *not* included in the produced list – to include the final element of a list in a slice, specify an end position past the end of the list. Ex: `boston_bruins[1:3]` produces the list ['Zdeno', 'Patrice'].

PARTICIPATION
ACTIVITY

11.6.1: List slicing.



Animation captions:

1. The list object is created.
2. The list is sliced from 0 to 3, and then printed out.
3. The list is sliced from 1 up to 2.

Negative indices can also be used to count backwards from the end of the list.

Figure 11.6.2: List slicing: Using negative indices.

```
election_years = [1992, 1996, 2000, 2004, 2008]
print(election_years[0:-1]) # Every year except the last
print(election_years[0:-3]) # Every year except the last
three
print(election_years[-3:-1]) # The third and second to last
years
```

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4

```
[1992, 1996, 2000,
2004]
[1992, 1996]
[2000, 2004]
```

A position of -1 refers to the last element of the list, thus `election_years[0:-1]` creates a slice

containing all but the last election year. Such usage of negative indices is especially useful when the length of a list is not known, and is simpler than the equivalent expression `election_years[0:len(election_years)-1]`.

PARTICIPATION ACTIVITY**11.6.2: List slicing.**

Assume that the following code has been evaluated:

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

```
nums = [1, 1, 2, 3, 5, 8, 13]
```

- 1) What is the result of `nums[1:5]`?

Check**Show answer**

- 2) What is the result of `nums[5:10]`?

Check**Show answer**

- 3) What is the result of `nums[3:-1]`?

Check**Show answer**

An optional component of slice notation is the **stride**, which indicates how many elements are skipped between extracted items in the source list. Ex: The expression `my_list[0:5:2]` has a stride of 2, thus skipping every other element, and resulting in a slice that contains the elements in positions 0, 2, and 4. The default stride value is 1 (the expressions `my_list[0:5:1]` and `my_list[0:5]` being equivalent).

©zyBooks 09/27/22 12:00 469702
Steven Cameron

If the reader has studied string slicing, then list slicing should be familiar. In fact, slicing has the same semantics for most sequence type objects.

PARTICIPATION ACTIVITY**11.6.3: List slicing.**

Given the following code:

```
nums = [0, 25, 50, 75, 100]
```

- 1) The result of evaluating `nums[0:5:2]` is
[25, 75].

True
 False

- 2) The result of evaluating `nums[0:-1:3]`
is [0, 75].

True
 False

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4



A table of common list slicing operations is given below. Note that omission of the start or end positions, such as `my_list[:2]` or `my_list[4:]`, has the same meaning as in string slicing. `my_list[:2]` includes every element up to position 2. `my_list[4:]` includes every element following position 4 (including the element at position 4).

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Table 11.6.1: Some common list slicing operations.

Operation	Description	Example code	Example output
my_list[start:end]	Get a list from start to end (minus 1).	©zyBooks 09/27/22 12:00 469702 Steven Cameron WGUC859v4 <pre>my_list = [5, 10, 20] print(my_list[0:2])</pre>	[5, 10]
my_list[start:end:stride]	Get a list of every stride element from start to end (minus 1).	<pre>my_list = [5, 10, 20, 40, 80] print(my_list[0:5:3])</pre>	[5, 40]
my_list[start:]	Get a list from start to end of the list.	<pre>my_list = [5, 10, 20, 40, 80] print(my_list[2:])</pre>	[20, 40, 80]
my_list[:end]	Get a list from beginning of list to end (minus 1).	<pre>my_list = [5, 10, 20, 40, 80] print(my_list[:4])</pre>	[5, 10, 20, 40]
my_list[:]	Get a copy of the list.	<pre>my_list = [5, 10, 20, 40, 80] print(my_list[:])</pre>	[5, 10, 20, 40, 80]

The interpreter handles incorrect or invalid start and end positions in slice notation gracefully. An end position that exceeds the length of the list is treated as the end of the list. If the end position is less than the start position, an empty list is produced.

PARTICIPATION ACTIVITY

11.6.4: Match the expressions to the list.



Match the expression on the left to the resulting list on the right. Assume that my_list is

the following Fibonacci sequence:

```
my_list = [1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Mouse: Drag/drop. Refresh the page if unable to drag and drop.

my_list[4:] my_list[3:1] my_list[3:6]

©zyBooks 09/27/22 12:00 469702

Steven Cameron

my_list[len(my_list)//2:(len(my_list)//2) + 1] my_list[:20] my_list[2:5]

B59v4

[5, 8, 13, 21, 34]

[]

[1, 1, 2, 3, 5, 8, 13, 21, 34]

[5]

[2, 3, 5]

[3, 5, 8]

Reset

**CHALLENGE
ACTIVITY**

11.6.1: List slicing.



334598.939404.qx3zqy7

Start

Type the program's output

```
my_list = [13, 14, 15, 16, 17, 18, 19]
new_list = my_list[0:2]
print(new_list)
```

©zyBooks 09/27/22 12:00 469702

[13, 14]

WGUC859v4

1

2

3

4

Check

Next

11.7 Loops modifying lists

Sometimes a program iterates over a list while modifying the elements, such as by changing some elements' values, or by moving elements' positions.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Changing elements' values

The below example of changing element's values combines the `len()` and `range()` functions to iterate over a list and increment each element of the list by 5.

Figure 11.7.1: Modifying a list during iteration example.

```
my_list = [3.2, 5.0, 16.5,  
          12.25]  
  
for i in range(len(my_list)):  
    my_list[ i ] += 5
```

The figure below shows two programs that each attempt to convert any negative numbers in a list to 0. The program on the right is incorrect, demonstrating a common logic error.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Figure 11.7.2: Modifying a list during iteration example: Converting negative values to 0.

Correct way to modify the list.

```
user_input = input('Enter numbers: ')  
  
tokens = user_input.split()  
  
# Convert strings to integers  
nums = []  
for token in tokens:  
    nums.append(int(token))  
  
# Print each position and number  
print()  
for pos, val in enumerate(nums):  
    print('{}: {}'.format(pos, val))  
  
# Change negative values to 0  
for pos in range(len(nums)):  
    if nums[pos] < 0:  
        nums[pos] = 0  
  
# Print new numbers  
print('New numbers: ')  
for num in nums:  
    print(num, end=' ')
```

```
Enter numbers:5 67 -5 -4 5 6 6 4  
0: 5  
1: 67  
2: -5  
3: -4  
4: 5  
5: 6  
6: 6  
7: 4  
New numbers:  
5 67 0 0 5 6 6 4
```

Incorrect way: list not modified.

```
user_input = input('Enter numbers: ')  
  
tokens = user_input.split()  
  
# Convert strings to integers  
nums = []  
for token in tokens:  
    nums.append(int(token))  
  
# Print each position and number  
print()  
for pos, val in enumerate(nums):  
    print('{}: {}'.format(pos, val))  
  
# Change negative values to 0  
for num in nums:  
    if num < 0:  
        num = 0 # Logic error: temp variable  
num set to 0  
  
# Print new numbers  
print('New numbers: ')  
for num in nums:  
    print(num, end=' ')
```

```
Enter numbers:5 67 -5 -4 5 6 6 4  
0: 5  
1: 67  
2: -5  
3: -4  
4: 5  
5: 6  
6: 6  
7: 4  
New numbers:  
5 67 -5 -4 5 6 6 4
```

The program on the right illustrates a common logic error. A common error when modifying a list during iteration is to update the loop variable instead of the list object. The statement `num = 0` simply binds the name `num` to the integer literal value 0. The reference in the list is never changed.

In contrast, the program on the left correctly uses an index operation `nums[pos] = 0` to modify to 0 the reference held by the list in position `pos`. The below activities demonstrate further; note that only the second program changes the list's values.

**PARTICIPATION
ACTIVITY**

11.7.1: Incorrect list modification example.



©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

**PARTICIPATION
ACTIVITY**

11.7.2: Corrected list modification example.

**CHALLENGE
ACTIVITY**

11.7.1: Iterating through a list using range().

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4



334598.939404.qx3zqy7

Start

Type the program's output

```
user_values = [1, 5, 8]
for i in range(len(user_values)):
    print(user_values[i])
```

1
5
8

1

2

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Check

Next

PARTICIPATION
ACTIVITY

11.7.3: List modification.



Consider the following program:

```
nums = [10, 20, 30, 40, 50]
for pos in range(len(nums)):
    tmp = nums[pos] / 2
    if (tmp % 2) == 0:
        nums[pos] = tmp
```

- 1) What's the final value of
nums[1]?



Check

Show answer

Changing list size

A common error is to add or remove a list element while iterating over that list. Such list modification can lead to unexpected behavior if the programmer is not careful. Ex: Consider the following program that reads in two sets of numbers and attempts to find numbers in the first set that are not in the second set.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Figure 11.7.3: Modifying lists while iterating: Incorrect program.

```
nums1 = []
nums2 = []

user_input = input('Enter first set of numbers: ')
tokens = user_input.split() # Split into separate strings

# Convert strings to integers
print()
for pos, val in enumerate(tokens):
    nums1.append(int(val))
    print('{}: {}'.format(pos, val))

user_input = input('Enter second set of numbers: ')
tokens = user_input.split()

# Convert strings to integers
print()
for pos, val in enumerate(tokens):
    nums2.append(int(val))
    print('{}: {}'.format(pos, val))

# Remove elements from nums1 if also in nums2
print()
for val in nums1:
    if val in nums2:
        print('Deleting {}'.format(val))
        nums1.remove(val)

# Print new numbers
print('\nNumbers only in first set:', end=' ')
for num in nums1:
    print(num, end=' ')
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

```
Enter first set of numbers:5 10 15
20
0: 5
1: 10
2: 15
3: 20
Enter second set of numbers:15 20
25 30
0: 15
1: 20
2: 25
3: 30

Deleting 15

Numbers only in first set: 5 10 20
```

The above example iterates over the list `nums1`, deleting an element from the list if the element is also found in the list `nums2`. The programmer expected a certain result, namely that after removing an element from the list, the next iteration of the loop would reference the next element as normal. However, removing the element shifts the position of each following element in the list to the left by one. In the example above, removing 15 from `nums1` shifts the value 20 left into position 2. The loop, having just iterated over position 2 and removing 15, moves to the next position and finds the end of the list, thus never evaluating the final value 20.

©zyBooks 09/27/22 12:00 469702
Steven Cameron

The problem illustrated by the example above has a simple fix: Iterate over a copy of the list instead of the actual list being modified. Copying the list allows a programmer to modify, swap, add, or delete elements without affecting the loop iterations. The easiest way to copy the iterating list is to use slice notation inside of the loop expression, as in:

Figure 11.7.4: Copy a list using [:].

```
for item in  
my_list[:]:  
    # Loop  
statements.
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

PARTICIPATION
ACTIVITY

11.7.4: List modification.



Animation captions:

1. The loop, having just iterated over position 1 and removing 10, moves to the next position and finds the end of the list, thus never evaluating the final value 15.
2. The problem illustrated by the example above can be fixed by iterating over a copy of the list instead of the actual list being modified.

zyDE 11.7.1: Modify the above program to work correctly.

Modify the program (copied from above) using slice notation to iterate over a copy of

[Load default template...](#)

Pre-enter any input for program, then run.

Run

```
1  
2 nums1 = []  
3 nums2 = []  
4  
5 user_input = input('Enter first set')  
6 tokens = user_input.split() # Split  
7  
8 # Convert strings to integers  
9 for pos, val in enumerate(tokens):  
10     nums1.append(int(val))  
11     print('{}: {}'.format(pos, val))  
12  
13 user_input = input('Enter second set')  
14 tokens = user_input.split()  
15  
16 # Convert strings to integers  
17 print()
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

PARTICIPATION
ACTIVITY

11.7.5: Modifying a list while iterating.





1) Iterating over a list and deleting elements from the original list might cause a logic program error.

- True
- False

2) A programmer can iterate over a copy of a list to safely make changes to that list.

- True
- False

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4



11.8 List comprehensions

A programmer commonly wants to modify every element of a list in the same way, such as adding 10 to every element. The Python language provides a convenient construct, known as **list comprehension**, that iterates over a list, modifies each element, and returns a new list consisting of the modified elements.

A list comprehension construct has the following form:

Construct 11.8.1: List comprehension.

```
new_list = [expression for loop_variable_name in iterable]
```

A list comprehension has three components:

1. An *expression component* to evaluate for each element in the iterable object.
2. A *loop variable component* to bind to the current iteration element.
3. An *iterable object component* to iterate over (list, string, tuple, enumerate, etc)

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

A list comprehension is always surrounded by brackets, which is a helpful reminder that the comprehension builds and returns a new list object. The loop variable and iterable object components make up a normal for loop expression. The for loop iterates through the iterable object as normal, and the expression operates on the loop variable in each iteration. The result is a new list containing the values modified by the expression. The below program demonstrates a simple

list comprehension that increments each value in a list by 5.

Figure 11.8.1: List comprehension example: A first look.

```
my_list = [10, 20, 30]
list_plus_5 = [(i + 5) for i in
my_list]

print('New list contains:',
list_plus_5)
```

©zyBooks 09/27/22 12:00 469702
New list contains: [15, 25, Cameron
35] WGUC859v4

The following animation illustrates:

PARTICIPATION
ACTIVITY

11.8.1: List comprehension.



Animation captions:

1. My list is created and holds integer values.
2. Loop variable i set to each element of my_list.

Programmers commonly prefer using a list comprehension rather than a for loop in many situations. Such preference is due to less code and due to more-efficient execution by the interpreter. The table below shows various for loops and equivalent list comprehensions.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Table 11.8.1: List comprehensions can replace some for loops.

Num	Description	For loop	Equivalent list comprehension	Output of both programs
1	Add 10 to every element.	<pre>my_list = [5, 20, 50] for i in range(len(my_list)): my_list[i] += 10 print(my_list)</pre>	<pre>my_list = [5, 20, 50] my_list = [(i+10) for i in my_list] print(my_list)</pre>	©zyBooks 09/27/22 12:00 469702 Steven Cameron WGUC859v4 [15, 30, 60]
2	Convert every element to a string.	<pre>my_list = [5, 20, 50] for i in range(len(my_list)): my_list[i] = str(my_list[i]) print(my_list)</pre>	<pre>my_list = [5, 20, 50] my_list = [str(i) for i in my_list] print(my_list)</pre>	['5', '20', '50']
3	Convert user input into a list of integers.	<pre>inp = input('Enter numbers:') my_list = [] for i in inp.split(): my_list.append(int(i)) print(my_list)</pre>	<pre>inp = input('Enter numbers:') my_list = [int(i) for i in inp.split()] print(my_list)</pre>	Enter numbers: 7 9 3 [7, 9, 3]
4	Find the sum of each row in a two-dimensional list.	<pre>my_list = [[5, 10, 15], [2, 3, 16], [100]] sum_list = [] for row in my_list: sum_list.append(sum(row)) print(sum_list)</pre>	<pre>my_list = [[5, 10, 15], [2, 3, 16], [100]] sum_list = [sum(row) for row in my_list] print(sum_list)</pre>	[30, 21, 100]
5	Find the sum of the row with the smallest sum in a two-dimensional table.	<pre>my_list = [[5, 10, 15], [2, 3, 16], [100]] sum_list = [] for row in my_list: sum_list.append(sum(row)) min_row = min(sum_list) print(min_row)</pre>	<pre>my_list = [[5, 10, 15], [2, 3, 16], [100]] min_row = min([sum(row) for row in my_list]) print(min_row)</pre>	©zyBooks 09/27/22 12:00 469702 Steven Cameron WGUC859v4 21

Note that list comprehension is not an exact replacement of for loops, because list

comprehensions create a new list object, whereas the typical for loop is able to modify an existing list.

The third row of the table above has an expression in place of the iterable object component of the list comprehension, `inp.split()`. That expression is evaluated first, and the list comprehension will loop over the list returned by `split()`.

The last example from above is interesting because the list comprehension is wrapped by the built-in function `min()`. List comprehension builds a new list when evaluated, so using the new list as an argument to `min()` is allowed – conceptually the interpreter is just evaluating the more familiar code: `min([30, 21, 100])`.

PARTICIPATION
ACTIVITY

11.8.2: List comprehension examples.



For the following questions, refer to the table above.

- 1) What's the output of the list comprehension program in row 1 if my_list is [-5, -4, -3]?

Check

[Show answer](#)



- 2) Alter the list comprehension from row 2 to convert each number to a float instead of a string.

```
my_list = [5, 20, 50]
my_list = [
     for i
    in my_list]
print(my_list)
```



Check

[Show answer](#)



- 3) What's the output of the list comprehension program from row 3 if the user enters "4 6 100"?

Check**Show answer**

- 4) What's the output of the list comprehension program in row 4 if my_list is [[5, 10], [1]]?

**Check****Show answer**

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

- 5) Alter the list comprehension from row 5 to calculate the sum of every number contained by my_list.

```
my_list = [[5, 10, 15], [2,  
3, 16], [100]]  
sum_list =  
      
([sum(row) for row in  
my_list])  
print(sum_list)
```

Check**Show answer****PARTICIPATION
ACTIVITY**

11.8.3: Building list comprehensions.



Write a list comprehension that contains elements with the desired values. Use the name 'i' as the loop variable.

- 1) Twice the value of each element in the list variable x.

**Check****Show answer**

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

- 2) The absolute value of each element in x. Use the abs() function to find the absolute value of a number.



Check**Show answer**

- 3) The largest square root of any element in x. Use math.sqrt() to calculate the square root.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Check**Show answer**

Conditional list comprehensions

A list comprehension can be extended with an optional conditional clause that causes the statement to return a list with only certain elements.

Construct 11.8.2: Conditional list comprehensions.

```
new_list = [expression for name in iterable if  
           condition]
```

Using the above syntax will only add an element to the resulting list if the condition evaluates to True. The following program demonstrates list comprehension with a conditional clause that returns a list with only even numbers.

Figure 11.8.2: Conditional list comprehension example: Return a list of even numbers.

```
# Get a list of integers from the user  
numbers = [int(i) for i in input('Enter  
numbers:').split()]  
  
# Return a list of only even numbers  
even_numbers = [i for i in numbers if (i % 2) == 0]  
print('Even numbers only:', even_numbers)
```

```
Enter numbers: 5 52 16 7 25  
Even numbers only: [52, 16]  
... ©zyBooks 09/27/22 12:00 469702  
Enter numbers: 8 12 -14 9 0  
Even numbers only: [8, 12, -14,  
0]
```

PARTICIPATION
ACTIVITY

11.8.4: Building list comprehensions with conditions.

Write a list comprehension that contains elements with the desired values. Use the name 'i' as the loop variable. Use parentheses around the expression or condition as necessary.

- 1) Only negative values from the list x



numbers =

Check

Show answer

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

- 2) Only negative odd values from the list x



numbers =

Check

Show answer

CHALLENGE ACTIVITY

11.8.1: List comprehensions.



334598.939404.qx3zqy7

Start

Type the program's output

```
my_list = [-1, 0, 1, 2]
new_list = [ number - 4 for number in my_list ]
print(new_list)
```

[-5, -4, -3,

1

2

3

Check

Next

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

11.9 Sorting lists

One of the most useful list methods is **sort()**, which performs an in-place rearranging of the list elements, sorting the elements from lowest to highest. The normal relational equality rules are followed: numbers compare their values, strings compare ASCII/Unicode encoded values, lists compare element-by-element, etc. The following animation illustrates.



Animation captions:

1. The list my_list is created and holds integer values.
2. The list is sorted in-place.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

The sort() method performs element-by-element comparison to determine the final ordering. Numeric type elements like int and float have their values directly compared to determine relative ordering, i.e., 5 is less than 10.

The below program illustrates the basic usage of the list.sort() method, reading book titles into a list and sorting the list alphabetically.

Figure 11.9.1: list.sort() method example: Alphabetically sorting book titles.

```
books = []
prompt = 'Enter new book: '
user_input = input(prompt).strip()

while (user_input.lower() != 'exit'):
    books.append(user_input)
    user_input =
        input(prompt).strip()

books.sort()

print('\nAlphabetical order:')
for book in books:
    print(book)
```

```
Enter new book: Pride, Prejudice, and
Zombies
Enter new book: Programming in Python
Enter new book: Hackers and Painters
Enter new book: World War Z
Enter new book: exit

Alphabetical order:
Hackers and Painters
Pride, Prejudice, and Zombies
Programming in Python
World War Z
```

The sort() method performs in-place modification of a list. Following execution of the statement my_list.sort(), the contents of my_list are rearranged. The **sorted()** built-in function provides the same sorting functionality as the list.sort() method, however, sorted() creates and returns a new list instead of modifying an existing list.

©zyBooks 09/27/22 12:00 469702
WGUC859v4

Figure 11.9.2: Using sorted() to create a new sorted list from an existing list without modifying the existing list.

```
numbers = [int(i) for i in input('Enter numbers:').split()]
sorted_numbers = sorted(numbers)
print('\nOriginal numbers:', numbers)
print('Sorted numbers:', sorted_numbers)
```

Enter numbers: -5 5 -100 23 4 5
Original numbers: [-5, 5, -100,
23, 4, 5]
Sorted numbers: [-100, -5, 4, 5,
5, 23]

PARTICIPATION
ACTIVITY

11.9.2: list.sort() and sorted().



- 1) The sort() method modifies a list in-place.



- True
 False

- 2) The output of the following is [13, 7,



```
primes = [5, 13, 7]
5]: primes.sort()
print(primes)
```

- True
 False

- 3) The output of `print(sorted([-5, 5, 2]))` is [2, -5, 5].



- True
 False

Both the list.sort() method and the built-in sorted() function have an optional **key** argument. The key specifies a function to be applied to each element prior to being compared. Examples of key functions are the string methods str.lower, str.upper, or str.capitalize.

Consider the following example, in which a roster of names is sorted alphabetically. If a name is mistakenly uncapitalized, then the sort algorithm places the name at the end of the list, because lower-case letters have a larger encoded value than upper-case letters. Ex: 'a' maps to the ASCII decimal value of 97 and 'A' maps to 65. Specifying the key function as str.lower (note the absence of parentheses) automatically converts the elements to lower-case before comparison, thus

placing the lower-case name at the appropriate position in the sorted list.

Figure 11.9.3: Using the key argument.

```
names = []
prompt = 'Enter name: '

user_input = input(prompt)

while user_input != 'exit':
    names.append(user_input)
    user_input = input(prompt)

no_key_sort = sorted(names)
key_sort = sorted(names, key=str.lower)

print('Sorting without key:', no_key_sort)
print('Sorting with key: ', key_sort)
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

```
Enter name: Serena Williams
Enter name: Venus Williams
Enter name: rafael Nadal
Enter name: john McEnroe
Enter name: exit
Sorting without key: ['Serena Williams', 'Venus Williams', 'john McEnroe', 'rafael Nadal']
Sorting with key:  ['john McEnroe', 'rafael Nadal', 'Serena Williams', 'Venus Williams']
```

The key argument can be assigned any function, not just string methods like str.upper and str.lower. Ex: A programmer might want to sort a two-dimensional list by the max of the rows, which can be accomplished by assigning key with the built-in function max, as in: sorted(x, key=max).

Figure 11.9.4: The key argument to list.sort() or sorted() can be assigned any function.

```
my_list = [[25], [15, 25, 35], [10,
15]]
sorted_list = sorted(my_list, key=max)
print('Sorted list:', sorted_list)
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Sorted list: [[10, 15], [25], [15, 25, 35]]

Sorting also supports the **reverse** argument. The reverse argument can be set to a Boolean value,

either True or False. Setting `reverse=True` flips the sorting from lower-to-highest to highest-to-lowest. Thus, the statement `sorted([15, 20, 25], reverse=True)` produces a list with the elements [25, 20, 15].

PARTICIPATION ACTIVITY**11.9.3: Sorting.**

Provide an expression using `x.sort` that sorts the list `x` accordingly.

©zyBooks 09/27/22 12:00 469702

Steven Cameron

WGUC859v4

- 1) Sort the elements of `x` such that the greatest element is in position 0.

Check**Show answer**

- 2) Arrange the elements of `x` from lowest to highest, comparing the upper-case variant of each element in the list.

Check**Show answer**

11.10 Command-line arguments

Command-line arguments are values entered by a user when running a program from a command line. A command line exists in some program execution environments, wherein a user can run a program by typing at a command prompt. Ex: To run a Python program named "myprog.py" with an argument specifying the location of a file named "myfile1.txt", the user would enter the following at the command prompt:

©zyBooks 09/27/22 12:00 469702

Steven Cameron

WGUC859v4

```
> python myprog.py myfile1.txt
```

The contents of this command line are automatically stored in the list `sys.argv`, which is stored in the standard library `sys` module. `sys.argv` consists of one string element for each argument typed on the command line.

When executing a program, the interpreter parses the entire command line to find all sequences of characters separated by whitespace, storing each as a string within list variable `argv`. As the entire command line is passed to the program, the name of the program executable is always added as

the first element of the list. Ex: For a command line of `python myprog.py myfile1.txt`, argv has the contents ['myprog.py', 'myfile1.txt'].

The following animation further illustrates.

PARTICIPATION ACTIVITY**11.10.1: Command-line arguments.**

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4**Animation captions:**

1. Whitespace separates arguments.
2. User text is stored in sys.argv list.

The following program illustrates simple use of command-line arguments, where the program name is myprog, and two additional arguments should be passed to the program.

Figure 11.10.1: Simple use of command line arguments.

```
import sys

name = sys.argv[1]
age = int(sys.argv[2])

print('Hello {}'.format(name))
print('{} is a great'.format(age))
age.\n'.format(age))
```

```
> python myprog.py Tricia 12
Hello Tricia.
12 is a great age.

> python myprog.py Aisha 30
Hello Aisha.
30 is a great age.

> python myprog.py Franco
Traceback (most recent call last):
  File "myprog.py", line 4, in <module>
    age = sys.argv[2]
IndexError: list index out of range
```

While a program may expect the user to enter certain command-line arguments, there is no guarantee that the user will do so. A common error is to access elements within argv without first checking the length of argv to ensure that the user entered enough arguments, resulting in an IndexError being generated. In the last example above, the user did not enter the age argument, resulting in an IndexError when accessing argv. Conversely, if a user entered too many arguments, extra arguments will be ignored. Above, if the user typed `python myprog.py Alan 70 pizza`, "pizza" will be stored in argv[3] but will never be used by the program.

Thus, when a program uses command-line arguments, a good practice is to always check the length of argv at the beginning of the program to ensure that the user entered the correct number of arguments. The following program uses the statement `if len(sys.argv) != 3` to check for the correct number of arguments, the three arguments being the program, name, and age. If the

number of arguments is incorrect, the program prints an error message, referred to as a **usage message**, that provides the user with an example of the correct command-line argument format. A good practice is to always output a usage message when the user enters incorrect command-line arguments.

Figure 11.10.2: Checking for proper number of command-line arguments.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

```
import sys

if len(sys.argv) != 3:
    print('Usage: python myprog.py name age\n')
    sys.exit(1) # Exit the program, indicating an error
with 1.

name = sys.argv[1]
age = int(sys.argv[2])

print('Hello {}'.format(name))
print('{} is a great age.\n'.format(age))
```

```
> python myprog.exe Tricia
12
Hello Tricia. 12 is a great
age.

> python myprog.py Franco
Usage: python myprog.py name
age

> python myprog.py Alan 70
pizza
Usage: python myprog.py name
age
```

Note that all command-line arguments in argv are strings. If an argument represents a different type like a number, then the argument needs to be converted using one of the built-in functions such as int() or float().

A single command-line argument may need to include a space. Ex: A person's name might be "Mary Jane". Recall that whitespace characters are used to separate the character typed on the command line into the arguments for the program. If the user provided a command line of **python myprog.py Mary Jane 65**, the command-line arguments would consist of four arguments: "myprog.py", "Mary", "Jane", and "65". When a single argument needs to contain a space, the user can enclose the argument within quotes "" on the command line, such as the following, which will result in only 3 command-line arguments, where sys.argv has the contents ['myprog.py', 'Mary Jane', '65'].

```
> python myprog.py "Mary Jane" 65
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

PARTICIPATION
ACTIVITY

11.10.2: Command-line arguments.



- 1) What is the value of sys.argv[1] given the following command-line input (include quotes in your answer): **python prog.py**



Tricia Miller 26

- 2) What is the value of sys.argv[1] given the following command-line input (include quotes in your answer): python prog.py
'Tricia Miller' 26



©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Exploring further:

Command-line arguments can become quite complicated for large programs with many options. There are entire modules of the standard library dedicated to aiding a programmer develop sophisticated argument parsing strategies. The reader is encouraged to explore modules such as argparse and getopt.

- **argparse**: Parser for command-line options, arguments, and sub-commands
- **getopt**: C-style parser for command-line options

11.11 Additional practice: Engineering examples

The following is a sample programming lab activity; not all classes using a zyBook require students to fully complete this activity. No auto-checking is performed. Users planning to fully complete this program may consider first developing their code in a separate programming environment.

©zyBooks 09/27/22 12:00 469702
WGUC859v4

A list can be useful in solving various engineering problems. One problem is computing the voltage drop across a series of resistors. If the total voltage across the resistors is V, then the current through the resistors will be $I = V/R$, where R is the sum of the resistances. The voltage drop V_x across resistor x is then $V_x = I \cdot R_x$.

zyDE 11.11.1: Calculate voltage drops across series of resistors.

The following program uses a list to store a user-entered set of resistance values and computes I.

Modify the program to compute the voltage drop across each resistor, store each in a list `voltage_drop`, and finally print the results in the following format:

```
5 resistors are in series.  
This program calculates the voltage drop across each resistor.  
Input voltage applied to circuit: 12.0  
Input ohms of 5 resistors  
1) 3.3  
2) 1.5  
3) 2.0  
4) 4.0  
5) 2.2  
Voltage drop per resistor is  
1) 3.0 V  
2) 1.4 V  
3) 1.8 V  
4) 3.7 V  
5) 2.0 V
```

[Load default template](#)

```
1 num_resistors = 5  
2 resistors = []  
3 voltage_drop = []  
4  
5  
6 print(f'{num_resistors} resistors are in the series.')  
7 print('This program calculates the')  
8 print('voltage drop across each resistor.')  
9  
10 input_voltage = float(input('Input voltage applied to circuit: '))  
11 print (input_voltage)  
12  
13  
14 print(f'Input ohms of {num_resistors} resistors')  
15 for i in range(num_resistors):  
16  
17     res = float(input('{i + 1}'))
```

```
12  
3.3  
1.5
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Run

Engineering problems commonly involve matrix representation and manipulation. A matrix can be captured using a two-dimensional list. Then matrix operations can be defined on such lists.

zyDE 11.11.2: Matrix multiplication of 4x2 and 2x3 matrices.

The following illustrates matrix multiplication for 4x2 and 2x3 matrices captured as two-dimensional lists.

©zyBooks 09/27/22 12:00 469702
Steven Cameron

Run the program below. Try changing the size and value of the matrices and compute the values.

[Load default terms](#)

```
1 m1_rows = 4
2 m1_cols = 2
3 m2_rows = m1_cols # Must have same value
4 m2_cols = 3
5
6 m1 = [
7     [3, 4],
8     [2, 3],
9     [1, 2],
10    [0, 2]
11 ]
12
13 m2 = [
14     [5, 4, 4],
15     [0, 2, 3]
16 ]
17
```

[Run](#)

11.12 Dictionaries

©zyBooks 09/27/22 12:00 469702
Steven Cameron

A dictionary is another type of container object that is different from sequences like strings, tuples, and lists. Dictionaries contain references to objects as key-value pairs – each key in the dictionary is associated with a value, much like each word in an English language dictionary is associated with a definition. As of Python 3.7, dictionary elements maintain their insertion order. The **dict** type implements a dictionary in Python.



Animation captions:

1. An English dictionary associates words with definitions.
2. A Python dictionary associates keys with values.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

There are several approaches to create a dict:

- The first approach wraps braces {} around key-value pairs of literals and/or variables:
`{'Jose': 'A+', 'Gino': 'C-'}` creates a dictionary with two keys 'Jose' and 'Gino' that are associated with the grades 'A+' and 'C-', respectively.
- The second approach uses **dictionary comprehension**, which evaluates a loop to create a new dictionary, similar to how list comprehension creates a new list. Dictionary comprehension is out of scope for this material.
- Other approaches use the **dict()** built-in function, using either keyword arguments to specify the key-value pairs or by specifying a list of tuple-pairs. The following creates equivalent dictionaries:
 - `dict(Bobby='805-555-2232', Johnny='951-555-0055')`
 - `dict([('Bobby', '805-555-2232'), ('Johnny', '951-555-0055')])`

In practice, a programmer first creates a dictionary and then adds entries, perhaps by reading user-input or text from a file. Dictionaries are mutable, thus entries can be added, modified, or removed in-place. The table below shows some common dict operations.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Table 11.12.1: Common dict operations.

Operation	Description	Example code
my_dict[key]	Indexing operation – retrieves the value associated with key.	jose_grade = my_dict['Jose'] ©zyBooks 09/27/22 12:00 469702 Steven Cameron WGUC859v4
my_dict[key] = value	Adds an entry if the entry does not exist, else modifies the existing entry.	my_dict['Jose'] = 'B+'
del my_dict[key]	Deletes the key from a dict.	del my_dict['Jose']
key in my_dict	Tests for existence of key in my_dict.	if 'Jose' in my_dict: # ...

Dictionaries can contain objects of arbitrary type, even other containers such as lists and nested dictionaries. Ex: `my_dict['Jason'] = ['B+', 'A-']` creates an entry in my dict whose value is a list containing the grades of the student 'Jason'.

zyDE 11.12.1: Dictionary example: Gradebook.

The following program implements a gradebook. The student_grades dictionary contains entries whose keys are student names and whose values are lists of student scores. The inputs to change the dictionary.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

```
1 student_grades = {} # Create an empty dict
2 grade_prompt = "Enter name and grade (Ex. 'Bob A+'):"
3 del_prompt = "Enter a name to delete:"
4 menu_prompt = ("1. Add/modify student grade\n"
5                 "2. Delete student grade\n"
6                 "3. Print student grades\n"
7                 "4. Quit\n")
8
9 while True: # Exit when user enters no input
10    command = input(menu_prompt).lower().strip()
11    if command == '1':
12        name, grade = input(grade_prompt).split()
13        student_grades[name] = grade
14    elif command == '2':
15        name = input(del_prompt)
16        del student_grades[name]
17    elif command == '3':
```

```
1
Bob A+
1
.
```

Run

PARTICIPATION
ACTIVITY

11.12.2: Dictionaries.



- 1) Dictionary entries can be modified in-place – a new dictionary does not need to be created every time an element is added, changed, or removed.

- True
- False

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4





- 2) The variable my_dict created with the following code contains two keys, 'Bob' and 'A+'.

```
my_dict = dict(name='Bob', grade='A+')
```

- True
 False

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4

CHALLENGE ACTIVITY

11.12.1: Delete from dictionary.



Delete Prussia from country_capital.

Sample output with input: 'Spain:Madrid,Togo:Lome,Prussia: Konigsberg'

Prussia deleted? Yes.

Spain deleted? No.

Togo deleted? No.

334598.939404.qx3zqy7

```
1 user_input = input()
2 entries = user_input.split(',')
3 country_capital = {}
4
5 for pair in entries:
6     split_pair = pair.split(':')
7     country_capital[split_pair[0]] = split_pair[1]
8     # country_capital is a dictionary, Ex. { 'Germany': 'Berlin', 'France': 'P
9
10    ''' Your solution goes here '''
11
12 print('Prussia deleted?', end=' ')
13 if 'Prussia' in country_capital:
14     print('No.')
15 else:
16     print('Yes.')
17
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Run

View your last submission ▾

**CHALLENGE
ACTIVITY**

11.12.2: Enter the output of dictionaries.



334598.939404.qx3zqy7

Start

Type the program's output

2 12:00 469702
Steven Cameron
WGUC859v4

```
airport_codes = {}  
airport_codes['San Jose'] = 'SJC'  
airport_codes['Cincinnati'] = 'CVG'  
airport_codes['Dallas'] = 'DAL'  
  
print(airport_codes['Dallas'])  
print(airport_codes['Cincinnati'])
```

**DAL
CVG****1**

2

3

Check**Next**

11.13 Dictionary methods

A **dictionary method** is a function provided by the dictionary type (`dict`) that operates on a specific dictionary object. Dictionary methods can perform some useful operations, such as adding or removing elements, obtaining all the keys or values in the dictionary, merging dictionaries, etc.

Below are a list of common dictionary methods:

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Table 11.13.1: Dictionary methods.

Dictionary method	Description	Code example	Output
my_dict.clear()	Removes all items from the dictionary.	<pre>my_dict = {'Ahmad': 1, 'Jane': 42} my_dict.clear() print(my_dict)</pre> <p style="text-align: right;">©zyBooks 09/27/22 12:00 469702 Steven Cameron WGUC859v4</p>	<div style="border: 1px solid black; padding: 2px; display: inline-block;">{} 42 N/A</div>
my_dict.get(key, default)	Reads the value of the key from the dictionary. If the key does not exist in the dictionary, then returns default.	<pre>my_dict = {'Ahmad': 1, 'Jane': 42} print(my_dict.get('Jane', 'N/A')) print(my_dict.get('Chad', 'N/A'))</pre>	<div style="border: 1px solid black; padding: 2px; display: inline-block;">42 N/A</div>
my_dict1.update(my_dict2)	Merges dictionary my_dict1 with another dictionary my_dict2. Existing entries in my_dict1 are overwritten if the same keys exist in my_dict2.	<pre>my_dict = {'Ahmad': 1, 'Jane': 42} my_dict.update({'John': 50}) print(my_dict)</pre> <p style="text-align: right;">©zyBooks 09/27/22 12:00 469702 Steven Cameron WGUC859v4</p>	<div style="border: 1px solid black; padding: 2px; display: inline-block;">{'Ahmad': 1, 'Jane': 42, 'John': 50}</div>
	Removes and returns the		

```
my_dict.pop(key, default)
```

key value from the dictionary. If key does not exist, then default is returned.

```
my_dict = {'Ahmad': 1, 'Jane': 42}  
val = my_dict.pop('Ahmad')  
print(my_dict)
```

```
{'Jane'  
42}
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Modification of dictionary elements using the above methods is performed in-place. Ex: Following the evaluation of the statement `my_dict.pop('Ahmad')`, any other variables that reference the same object as `my_dict` will also reflect the removal of 'Ahmad'. As with lists, a programmer should be careful not to modify dictionaries without realizing that other references to the objects may be affected.

PARTICIPATION ACTIVITY

11.13.1: Dictionary methods.



Determine the output of each code segment. If the code produces an error, type None.

Assume that `my_dict` has the following entries:

```
my_dict = dict(bananas=1.59, fries=2.39, burger=3.50, sandwich=2.99)
```

1) `my_dict.update(dict(soda=1.49,
burger=3.69))
burger_price =
my_dict.get('burger', 0)
print(burger_price)`

```
_____
```

Check

[Show answer](#)



2) `my_dict['burger'] =
my_dict['sandwich']
val = my_dict.pop('sandwich')
print(my_dict['burger'])`

```
_____
```

Check

[Show answer](#)



©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

CHALLENGE ACTIVITY

11.13.1: Enter the output of dictionary methods.



334598.939404.qx3zqy7

Start

Type the program's output

```
airport_codes = {  
    'Dallas': 'DAL',  
    'Washington': 'IAD',  
    'Los Angeles': 'LAX',  
    'Cincinnati': 'CVG',  
    'Seattle': 'SEA'  
}  
  
print(airport_codes.get('Vancouver', 'na'))  
print(airport_codes.get('Seattle', 'na'))  
print(airport_codes.get('San Francisco', 'na'))
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

na
SEA
na

1

2

3

Check**Next**

11.14 Iterating over a dictionary

As usual with containers, a common programming task is to iterate over a dictionary and access or modify the elements of the dictionary. A for loop can be used to iterate over a dictionary object, the loop variable being set to a key of an entry in each iteration. The ordering in which the keys are iterated over is not necessarily the order in which the elements were inserted into the dictionary. The Python interpreter creates a hash of each key. A **hash** is a transformation of the key into a unique value that allows the interpreter to perform very fast lookup. Thus, the ordering is actually determined by the hash value, but such hash values can change depending on the Python version and other factors.

Construct 11.14.1: A for loop over a dictionary retrieves each key in the dictionary.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

```
for key in dictionary: # Loop  
    expression  
        # Statements to execute in the  
        loop  
  
    #Statements to execute after the loop
```

The dict type also supports the useful methods `items()`, `keys()`, and `values()` methods, which produce a view object. A **view object** provides read-only access to dictionary keys and values. A program can iterate over a view object to access one key-value pair, one key, or one value at a time, depending on the method used. A view object reflects any updates made to a dictionary, even if the dictionary is altered after the view object is created.

- `dict.items()` – returns a view object that yields (key, value) tuples.
- `dict.keys()` – returns a view object that yields dictionary keys.
- `dict.values()` – returns a view object that yields dictionary values.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

The following examples show how to iterate over a dictionary using the above methods:

Figure 11.14.1: Iterating over a dictionary.

dict.items()

```
num_calories = dict(Coke=90,  
Coke_zero=0, Pepsi=94)  
for soda, calories in  
num_calories.items():  
    print('{}: {}'.format(soda,  
calories))
```

```
Coke: 90  
Coke_zero: 0  
Pepsi: 94
```

dict.keys()

```
num_calories = dict(Coke=90,  
Coke_zero=0, Pepsi=94)  
for soda in num_calories.keys():  
    print(soda)
```

```
Coke  
Coke_zero  
Pepsi
```

dict.values()

```
num_calories = dict(Coke=90,  
Coke_zero=0, Pepsi=94)  
for soda in num_calories.values():  
    print(soda)
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

When a program iterates over a view object, one result is generated for each iteration as needed, instead of generating an entire list containing all of the keys or values. Such behavior allows the

interpreter to save memory. Since results are generated as needed, view objects do not support indexing. A statement such as `my_dict.keys()[0]` produces an error. Instead, a valid approach is to use the `list()` built-in function to convert a view object into a list, and then perform the necessary operations. The example below converts a dictionary view into a list, so that the list can be sorted to find the first two closest planets to Earth.

Figure 11.14.2: Use `list()` to convert view objects into lists.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

```
solar_distances = dict(mars=219.7e6, venus=116.4e6,  
jupiter=546e6, pluto=2.95e9)  
list_of_distances = list(solar_distances.values()) # Convert  
view to list  
  
sorted_distance_list = sorted(list_of_distances)  
closest = sorted_distance_list[0]  
next_closest = sorted_distance_list[1]  
  
print('Closest planet is {:.4e}'.format(closest))  
print('Second closest planet is {:.4e}'.format(next_closest))
```

Closest planet is
1.1640e+08
Second closest planet is
2.1970e+08

The `dict.items()` method is particularly useful, as the view object that is returned produces tuples containing the key-value pairs of the dictionary. The key-value pairs can then be unpacked at each iteration, similar to the behavior of `enumerate()`, providing both the key and the value to the loop body statements without requiring extra code.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

zyDE 11.14.1: Iterating over a dictionary example: Gradebook statistics.

Write a program that uses the keys(), values(), and/or items() dict methods to find stats about the student_grades dictionary. Find the following:

- Print the name and grade percentage of the student with the highest total of points.
- Find the average score of each assignment.
- Find and apply a curve to each student's total score, such that the best student gets 100% of the total points.

[Load default template...](#)

Run

```
1
2 # student_grades contains scores (out of 100) for 5 assignments
3 student_grades = {
4     'Andrew': [56, 79, 90, 22, 50],
5     'Nisreen': [88, 62, 68, 75, 78],
6     'Alan': [95, 88, 92, 85, 85],
7     'Chang': [76, 88, 85, 82, 90],
8     'Tricia': [99, 92, 95, 89, 99]
9 }
10
11
12 |
```

PARTICIPATION ACTIVITY

11.14.1: Iterating over dictionaries.



Fill in the code, using the dict methods items(), keys(), or values() where appropriate.

- 1) Print each key in the dictionary
my_dict.



```
for key in
    :
        print(key)
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Check

[Show answer](#)

- 2) Change all negative values in
my_dict to 0.



```
for key, value in
    :
        if value < 0:
            my_dict[key] = 0
```

Check**Show answer**

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4

- 3) Print twice the value of every value in my_dict.

```
for v in
    :
        print(2 * v)
```

Check**Show answer****CHALLENGE ACTIVITY**

11.14.1: Report country population.



Write a loop that prints each country's population in country_pop.

Sample output with input:

'China:1365830000,India:1247220000,United States:318463000,Indonesia:252164800':

United States has 318463000 people.
India has 1247220000 people.
Indonesia has 252164800 people.
China has 1365830000 people.

334598.939404.qx3zqy7

```
1 user_input = input()
2 entries = user_input.split(',')
3 country_pop = {}
4
5 for pair in entries:
6     split_pair = pair.split(':')
7     country_pop[split_pair[0]] = split_pair[1]
8     # country_pop is a dictionary, Ex: { 'Germany': '82790000', 'France': '67190000'
9
10    ''' Your solution goes here '''
11
12    print(country, 'has', pop, 'people.')
```

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4

 Run

View your last submission ▾

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

11.15 Dictionary nesting

A dictionary may contain one or more **nested dictionaries**, in which the dictionary contains another dictionary as a value. Consider the following code:

Figure 11.15.1: Nested dictionaries.

```
students = {}
students ['Jose'] = {'Grade': 'A+', 'StudentID':
22321}

print('Jose:')
print(' Grade: {}'.format(students ['Jose']
['Grade']))
print(' ID: {}'.format(students['Jose']
['StudentID']))
```

Jose:
Grade:
A+
ID:
22321

The variable `students` is first created as an empty dictionary. An indexing operation creates a new entry in `students` with the key '`'Jose'`' and the value of another dictionary. Indexing operations can be applied to the nested dictionary by using consecutive sets of brackets `[]`: The expression `students['Jose']['Grade']` first obtains the value of the key '`'Jose'`' from `students`, yielding the nested dictionary. The second set of brackets indexes into the nested dictionary, retrieving the value of the key '`'Grade'`'.

©zyBooks 09/27/22 12:00 469702

Nested dictionaries also serve as a simple but powerful data structure. A **data structure** is a method of organizing data in a logical and coherent fashion. Actually, container objects like lists and dicts are already a form of a data structure, but nesting such containers provides a programmer with much more flexibility in the way that the data can be organized. Consider the simple example below that implements a gradebook using nested dictionaries to organize students and grades.

Figure 11.15.2: Nested dictionaries example: Storing grades.

```

grades = {
    'John Ponting': {
        'Homeworks': [79, 80, 74],
        'Midterm': 85,
        'Final': 92
    },
    'Jacques Kallis': {
        'Homeworks': [90, 92, 65],
        'Midterm': 87,
        'Final': 75
    },
    'Ricky Bobby': {
        'Homeworks': [50, 52, 78],
        'Midterm': 40,
        'Final': 65
    },
}

user_input = input('Enter student name: ')

while user_input != 'exit':
    if user_input in grades:
        # Get values from nested dict
        homeworks = grades[user_input]['Homeworks']
        midterm = grades[user_input]['Midterm']
        final = grades[user_input]['Final']

        # print info
        for hw, score in enumerate(homeworks):
            print('Homework {}: {}'.format(hw, score))

        print('Midterm: {}'.format(midterm))
        print('Final: {}'.format(final))

        # Compute student total score
        total_points = sum([i for i in homeworks]) + midterm
+       final
        print('Final percentage: {:.1f}%'.format(100*(total_points / 500.0)))

    user_input = input('Enter student name: ')

```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

```

Enter student name: Ricky
Bobby
Homework 0: 50
Homework 1: 52
Homework 2: 78
Midterm: 40
Final: 65
Final percentage: 57.0%
...
Enter student name: John
Ponting
Homework 0: 79
Homework 1: 80
Homework 2: 74
Midterm: 85
Final: 92
Final percentage: 82.0%

```

Note the whitespace and indentation used to layout the nested dictionaries. Such layout improves the readability of the code and makes the hierarchy of the data structure obvious. The extra whitespace does not affect the dict elements, as the interpreter ignores indentation in a multi-line construct.

A benefit of using nested dictionaries is that the code tends to be more readable, especially if the keys are a category like 'Homeworks'. Alternatives like nested lists tend to require more code, consisting of more loops constructs and variables.

Dictionaries support arbitrary levels of nesting; Ex: The expression

`students['Jose']['Homeworks'][2]['Grade']` might be applied to a dictionary that has four levels of nesting.

zyDE 11.15.1: Nested dictionaries example: Music library.

The following example demonstrates a program that uses 3 levels of nested dictionaries to create a simple music library.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

The following program uses nested dictionaries to store a small music library. Extend the program such that a user can add artists, albums, and songs to the library. First, add a command that adds an artist name to the music dictionary. Then add commands for albums and songs. Take care to check that an artist exists in the dictionary before adding an album, and that an album exists before adding a song.

```
Load default template...  
  
1 music = {  
2     'Pink Floyd': {  
3         'The Dark Side of the Moon': {  
4             'songs': [ 'Speak to Me'  
5             'year': 1973,  
6             'platinum': True  
7         },  
8         'The Wall': {  
9             'songs': [ 'Another Brick in the Wall Part 2'  
10            'year': 1979,  
11            'platinum': True  
12        },  
13    },  
14    'Justin Bieber': {  
15        'My World': {  
16            'songs': [ 'One Time', 'Sorry'  
17            'year': 2010,
```

Pre-enter any input for program, then click Run.

Run

PARTICIPATION ACTIVITY

11.15.1: Nested dictionaries.



- 1) Nested dictionaries are a flexible way to organize data.

- True
- False

- 2) Dictionaries can contain, at most, three levels of nesting.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4



- 3) The expression `{'D1': {'D2': 'x'}}` is valid.
- True
 False
 True
 False



©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

11.16 LAB: Varied amount of input data

Statistics are often calculated with varying amounts of input data. Write a program that takes any number of integers as input, and outputs the average and max.

Ex: If the input is:

```
15 20 0 5
```

the output is:

```
10 20
```

Note: For output, round the average to the nearest integer.

334598.939404.qx3zqy7

LAB ACTIVITY

11.16.1: LAB: Varied amount of input data

10 / 10

main.py

Load default template...

```
1 raw_num = input().strip().split()  
2 #raw_num = ['15', '20', '0', '5']  
3 int_num = [int(i) for i in raw_num]  
4 avg_num = sum(int_num) / len(int_num)  
5  
6 print('{:.0f} {}'.format(avg_num, max(int_num)))  
7
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Develop mode**Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Run program

Input (from above)

**main.py**
(Your program)

0

Program output displayed here

Coding trail of your work [What is this?](#)

9/9 F-10 min:7

11.17 LAB: Filter and sort a list

Write a program that gets a list of integers from input, and outputs non-negative integers in ascending order (lowest to highest).

Ex: If the input is:

10 -7 4 39 -6 12 2

the output is:

2 4 10 12 39

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

For coding simplicity, follow every output value by a space. Do not end with newline.

334598.939404.qx3zqy7

**LAB
ACTIVITY**

11.17.1: LAB: Filter and sort a list

10 / 10

main.py

[Load default template...](#)

```
1 the_input = input().strip().split()
2 caw = [int(i) for i in the_input]
3 nums = []
4
5 for i in caw:
6     if i >= 0:
7         nums.append(i)
8
9 nums.sort()
10 str_it = [str(i) for i in nums]
11 print(' '.join(str_it), end=' ')
12
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

[Develop mode](#)[Submit mode](#)

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

[Run program](#)

Input (from above)



main.py
(Your program)



0

Program output displayed here

Coding trail of your work

[What is this?](#)

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4

9 / 9 F-----0, 0, 10 min:17

11.18 LAB: Elements in a range

Write a program that first gets a list of integers from input. That list is followed by two more integers representing lower and upper bounds of a range. Your program should output all integers from the list that are within that range (inclusive of the bounds).

Ex: If the input is:

```
25 51 0 200 33  
0 50
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

the output is:

```
25 0 33
```

The bounds are 0-50, so 51 and 200 are out of range and thus not output.

For coding simplicity, follow each output integer by a space, even the last one. Do not end with newline.

334598.939404.qx3zqy7

LAB
ACTIVITY

11.18.1: LAB: Elements in a range

10 / 10

main.py

[Load default template...](#)

```
1 the_input = list(map(int, input().strip().split()))  
2 ranges = list(map(int, input().strip().split()))  
3  
4 for i in range(len(the_input)):  
5     if (the_input[i] >= ranges[0]) and (the_input[i] <= ranges[1]):  
6         print(the_input[i], end=' ')  
7
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Develop mode

Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)

**main.py**
(Your program)

0

Program output displayed here

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Coding trail of your work [What is this?](#)

9/9 F-----10 min:17

11.19 LAB: Contact list

A contact list is a place where you can store a specific contact with other associated information such as a phone number, email address, birthday, etc. Write a program that first takes in word pairs that consist of a name and a phone number (both strings). That list is followed by a name, and your program should output the phone number associated with that name.

Ex: If the input is:

```
Joe 123-5432 Linda 983-4123 Frank 867-5309
Frank
```

the output is:

```
867-5309
```

334598.939404.qx3zqy7

LAB ACTIVITY

11.19.1: LAB: Contact list

©zyBooks 09/27/22 12:00 469702
10 / 10
Steven Cameron
WGUC859v4

main.py[Load default template...](#)

```
1 contacts = input().strip().split()
2 lookup = input().strip()
3 contact_dict = {contacts[i]: contacts[i + 1] for i in range(0, len(contacts), 2)}
4
5 print(contact_dict[lookup])
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Develop mode

Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)



main.py
(Your program)



0

Program output displayed here

Coding trail of your work [What is this?](#)

9/9 **F10** min:20

11.20 LAB: Check if list is sorted

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Write the `in_order()` function, which has a list of integers as a parameter, and returns True if the integers are sorted (in order from low to high) or False otherwise. The program outputs "In order" if the list is sorted, or "Not in order" if the list is not sorted.

Ex: If the list passed to the `in_order()` function is [5, 6, 7, 8, 3], then the function returns False and the program outputs:

Not in order

Ex: If the list passed to the `in_order()` function is [5, 6, 7, 8, 10], then the function returns True and the program outputs:

In order

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Note: Use a for loop. **DO NOT** use `sorted()` or `sort()`.

334598.939404.qx3zqy7

LAB
ACTIVITY

11.20.1: LAB: Check if list is sorted

10 / 10

main.py

[Load default template...](#)

```
1 def in_order(nums):
2     a = [nums[i - 1] <= nums[i] for i in range(1, len(nums))]
3     return all(x == a[0] for x in a)
4
5
6 if __name__ == '__main__':
7     # Test out-of-order example
8     nums1 = [5, 6, 7, 8, 3]
9     if in_order(nums1):
10         print('In order')
11     else:
12         print('Not in order')
13
14 # Test in-order example
15 nums2 = [5, 6, 7, 8, 10]
16 if in_order(nums2):
17     print('In order')
```

Develop mode

Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Run program

Input (from above)



main.py
(Your program)



0

Program output displayed here

Coding trail of your work [What is this?](#)

9/10 S10 min:11

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4

11.21 Lists and dictionaries practice

Learning resources

Following is a list of recommended resources for use when completing these exercises.

- [Data Structures](#) in the Python Tutorial
- [Built-in Types](#) in the Python Standard Library
- [Python Lists](#) from W3Schools.com
- [Python Dictionaries](#) from W3Schools.com

Tasks

WGU is providing additional practice exercises for all students. The best method of learning Python is to practice. This will also help you prepare for the assessment, which requires you to write code.

You may use a web-based version of Python such as [PyFiddle](#), [Repl.it](#), or [PythonFiddle](#) to complete the exercises. Just make sure you are using version 3 or greater. If you use a web-based Python environment, you can easily share code with course instructors if you need help. You may also use a local installation of Python.

Below is a sample exercise. You should copy the entire code snippet into your Python editor. Your code should be placed in the area that says "student code goes here" highlighted in yellow. This is inside the function. When you run this entire code snippet, there are test cases below your code. Your output should match the expected output.

```
# Complete the function to print the first X number of characters in the given
string
def printFirst(mystring, x):
# Student code goes here

# expected output: WGU
printFirst('WGU College of IT', 3)
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

```
# expected output: WGU College
printFirst('WGU College of IT', 11)
```

Task 1

Complete the function to return the first two items in the given list

```
# Complete the function to return the first two items in the given list
def getFirstTwo(mylist):
    # Student code goes here

# expected output: [8, 3]
print(getFirstTwo([8,3,5,2,10]))

# expected output: [15, 2]
print(getFirstTwo([15,2,10,12]))
```

Task 2

Complete the function to return the last two items in the given list

```
# Complete the function to return the last two items in the given list
def getLastTwo(mylist):
    # Student code goes here

# expected output: [2, 10]
printgetLastTwo([8,3,5,2,10])

# expected output: [9, 12]
printgetLastTwo([15,2,9,12]))
```

Task 3

Complete the function to add num1 to the end of the given list

```
# Complete the function to add num1 to the end of the given list
def addToEnd(mylist, num1):
    # Student code goes here

# expected output: [4, 5, 6, 7]
print(addToEnd([4,5,6], 7))

# expected output: [9, 8, 7, 6]
print(addToEnd([9,8,7], 6))
```

Task 4

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Complete the function to add num1 to the front of the given list

```
# Complete the function to add num1 to the front of the given list
def addToFront(mylist, num1):
# Student code goes here

# expected output: [3, 4, 5, 6]
print(addToFront([4,5,6], 3))

# expected output: [10, 9, 8, 7]
print(addToFront([9,8,7], 10))
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Task 5

Complete the function to return a new list containing the first two and last two items in the given list

```
# Complete the function to return a new list containing
# the first two and last two items in the given list
def getFirstTwoAndLastTwo(mylist):
# Student code goes here

# expected output: [8, 3, 19, 1]
print(getFirstTwoAndLastTwo([8,3,7,15,2,10,19,1]))

# expected output: [7, 15, 3, 5]
print(getFirstTwoAndLastTwo([7,15,2,10,19,1,3,5]))
```

Task 6

Complete the function to remove the first item in the given list

```
# Complete the function to remove the first item in the given list
def removeFirst(mylist):
# Student code goes here

# expected output: [7, 8, 9]
print(removeFirst([6,7,8,9]))

# expected output: [2, 3, 4]
print(removeFirst([1,2,3,4]))
```

Task 7

Complete the function to remove the third item in the given list

```
# Complete the function to remove the third item in the given list
def removeThird(mylist):
# Student code goes here

# expected output: [6, 7, 9]
print(removeThird([6,7,8,9]))

# expected output: [1, 2, 4]
print(removeThird([1,2,3,4]))
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Task 8

Complete the function to order the values in the list. If ascending is true then order lowest to highest otherwise sort highest to lowest

```
# Complete the function to order the values in the list
# if ascending is true then order lowest to highest
# if ascending is false then order highest to lowest
def sortList(mylist, ascending):
    # Student code goes here

# expected output: [4, 12, 19, 33]
print(sortList([19,4,33,12], True))

# expected output: [33, 19, 12, 4]
print(sortList([19,4,33,12], False))
```

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Task 9

Complete the function to return a dictionary using list1 as the keys and list2 as the values

```
# Complete the function to return a dictionary using
# list1 as the keys and list2 as the values
def createDict(list1, list2):
    # Student code goes here

# expected output: {'tomato': 'red', 'banana': 'yellow', 'lime': 'green'}
print(createDict(['tomato', 'banana', 'lime'], ['red','yellow','green']))

# expected output: {'Brazil': 'Brasilia', 'Ireland': 'Dublin', 'Indonesia': 'Jakarta'}
print(createDict(['Brazil', 'Ireland', 'Indonesia'], ['Brasilia','Dublin','Jakarta']))
```

Task 10

Complete the function to return a dictionary value if it exists or return Not Found if it doesn't exist

```
# Complete the function to return a dictionary value
# if it exists or return Not Found if it doesn't exist
def findDictItem(mydict, key):
    # Student code goes here

# expected output: yellow
print(findDictItem({'tomato': 'red', 'banana': 'yellow', 'lime': 'green'} , 'banana'))

# expected output: Not Found
print(findDictItem({'Brazil': 'Brasilia', 'Ireland': 'Dublin', 'Indonesia': 'Jakarta'},'Cameroon'))
```

Task 11

©zyBooks 09/27/22 12:00 469702
Steven Cameron
WGUC859v4

Complete the function to remove a dictionary item if it exists

```
# Complete the function to remove a dictionary item if it exists
def removeDictItem(mydict, key):
# Student code goes here

# expected output: {'tomato': 'red', 'lime': 'green'}
print(removeDictItem({'tomato': 'red', 'banana': 'yellow', 'lime': 'green'}, 'banana'))

# expected output: {'Brazil': 'Brasilia', 'Ireland': 'Dublin', 'Indonesia': 'Jakarta'}
print(removeDictItem({'Brazil': 'Brasilia', 'Ireland': 'Dublin', 'Indonesia': 'Jakarta'}, 'Cameroon'))
```

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4

Task 12

Complete the function to print every key and value

```
# Complete the function to print every key and value
def printDict(mydict):
# Student code goes here

# expected output:
#     tomato=red
#     banana=yellow
#     lime=green
printDict({'tomato': 'red', 'banana': 'yellow', 'lime': 'green'})

# expected output:
#     Brazil=Brasilia
#     Ireland=Dublin
#     Indonesia=Jakarta
printDict({'Brazil': 'Brasilia', 'Ireland': 'Dublin', 'Indonesia': 'Jakarta'})
```

©zyBooks 09/27/22 12:00 469702

Steven Cameron
WGUC859v4