



Chapter 28



LISTS AND BOOLEAN EXPRESSIONS

LISTS

A **LIST** is a variable that stores multiple values. This is useful when you have several pieces of information that you want to store in one place. Lists can store all types of values including numbers, strings, and other lists.

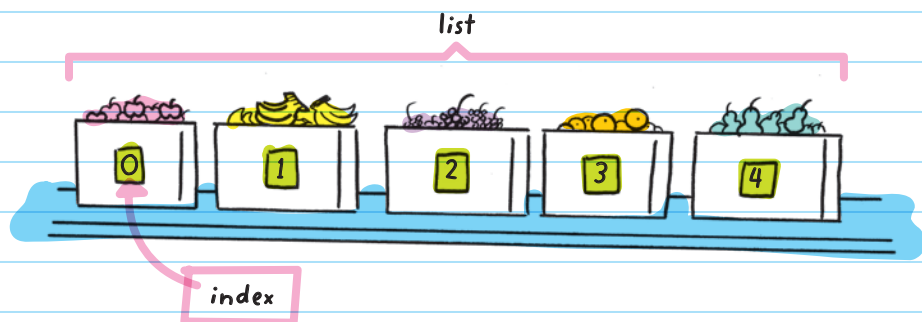
In Python, lists are formatted using square brackets (and sometimes quotation marks), with a comma between each item. For example, we can make a list called "fruits" and assign it the values apples, bananas, grapes, oranges, and pears:



```
fruits = ['apples', 'bananas', 'grapes', 'oranges', 'pears']
```

Note: Each item is separated by a comma, but make sure the commas are on the outside of the quotation marks so they are not included as part of the string.

Think of a list as a row of boxes, where each box holds a value. Every box on the list gets a number, starting with 0. This number is called the **INDEX**.



The index is used to find or change specific values, or items, within the list. In the fruit example, apples is located at index location 0, and the index location of pears is 4.

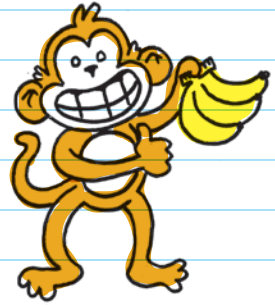
A specific item in a list is located using the format `list[#]`, where "list" is the name of the list and "#" is the index location of the item.

So, `fruits[0]` is apples and `fruits[4]` is pears.

To display the second item of the fruits list, use the print function, enter "fruits" as the name of the list, and "1" for the index location of the item:

```
print(fruits[1])
```

Because the index starts at 0, the second item has an index of 1.



This will print: **bananas**

You can also print out a section of the list by naming the range of indexes to print, like this:

```
print(fruits[1:4])
```

ending index

starting index



This will print: **['bananas', 'grapes', 'oranges']**

The ending index (4: pears) is not included.



When you use **print()** to display more than one item in a list, the output will include the brackets and quotation marks. That's because you're printing a list, not just items in a list.

There are lots of different ways to update a list.

The last item isn't listed—only the items *up to* the ending index position are included in the list.

You can replace an item in a list after the list has been created.

■ To replace a value of an item within a list,

1. reference (name) the index location
2. reassign the value

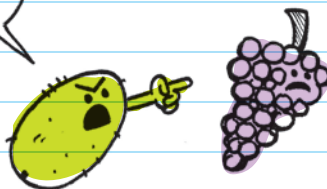
To replace grapes with kiwi: `['apples', 'bananas', 'grapes', 'oranges', 'pears']`

The index location of grapes is 2. You can replace grapes by assigning index 2 of the fruits list to kiwi:

```
fruits[2] = "kiwi"
```

GET OUTTA
HERE!

The list is now: `['apples', 'bananas', 'kiwi', 'oranges', 'pears']`



- To add an item to the end of a list, use the `append()` function.

Append means
"to add"

If the list of fruits was a shopping list, the `append()` function would add items to the end of it.

You can add "cherries" to the list by applying the `append` function to the fruits list like this:

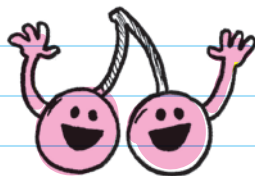
```
fruits.append("cherries")
```

The list is now: `['apples', 'bananas', 'kiwi', 'oranges', 'pears', 'cherries']`

Commas are
added automatically
when you add new
items to a list.

- To insert an item between two other values in a list, use the `insert()` function.

Inside the parentheses, tell the function where to insert the new value and what the value should be.



Add "peaches" to index location 2 of the fruits list using the `insert()` function like this:

```
fruits.insert(2, "peaches")
```

new item index location

value to insert

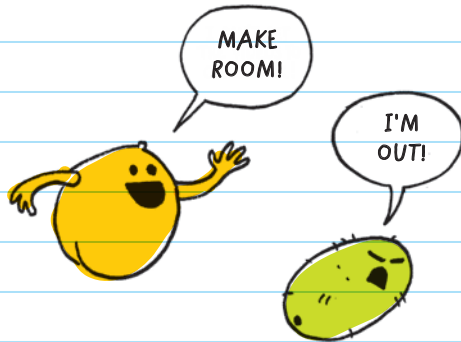
The list is now: ['apples', 'bananas', 'peaches', 'kiwi', 'oranges', 'pears', 'cherries']

- To remove an item from a list, use the `remove()` function. Inside the parentheses, tell the function which item you want removed by entering the value (the name of the item, not the index number):

```
fruits.remove("kiwi")
```

The list is now:

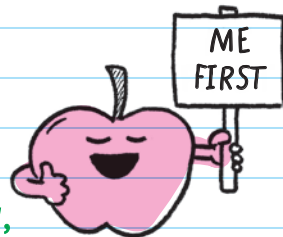
```
['apples', 'bananas',  
'peaches', 'oranges',  
'pears', 'cherries']
```



- To sort a list,
 - use either the `sort()` function, which will put the list in numerical or alphabetical order (A-Z), or
 - use the `reverse()` function, which will put the list in reverse order.

```
fruits.sort()
```

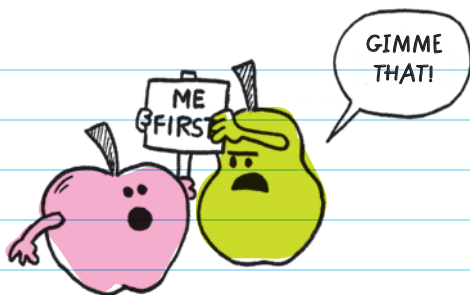
orders the list to: ['apples', 'bananas', 'cherries', 'oranges', 'peaches', 'pears']



```
fruits.reverse()
```

orders the list to:

```
['pears', 'peaches',  
'oranges', 'cherries',  
'bananas', 'apples']
```



- To get the number of items in a list, use the `len()` function:

```
fruits = ['pears', 'peaches', 'oranges', 'cherries',  
'bananas', 'apples']  
fruit_length = len(fruits)  
print(fruit_length)
```

(Put the name of the list
inside the parentheses.)

prints: 6

- To add one list to the end of another list, use the `+` operator:

```
fruits = ['apples', 'bananas', 'kiwi']  
vegetables = ['carrots', 'peas', 'onions']  
produce = fruits + vegetables  
print(produce)
```

Give the new variable (the
combined list) a name.

prints: ['apples', 'bananas', 'kiwi', 'carrots', 'peas',
'onions']

LIST FUNCTIONS

list[#]: References a specific item in a list, where “#” is the index number of the item and “list” is the name of the list

append(“item”): Adds an item to the end of a list

insert(#, “item”): Inserts an item in a specific index location in a list

remove(“item”): Removes an item from a list

sort(): Sorts a list by numerical or alphabetical order

reverse(): Sorts a list by reverse order

len(): Provides the number of items found in a list

Lists Within Lists

A list can be set within another list. The second list is called the **INNER LIST**. You can use this list to show a subcategory—a more specialized group. The format for making a list within a list is a second set of brackets within the first:

Outer list Outer list continued

```
list = ["A", "B", "C", ["D1", "D2", "D3"], "E"]
```

Inner list, stored in fourth item location of outer list

FOR EXAMPLE, to list favorite apples within the list of favorite fruits:

1. enter the opening bracket
2. add the types of apples in quotation marks, and end with a closing bracket
3. continue with the rest of the list

The diagram shows the code `fruits = ["bananas", ["Gala", "Empire", "McIntosh", "Golden Delicious"], "kiwi", "oranges", "pears"]`. Brackets are used to identify parts of the list: a top bracket from the second opening quote to the closing quote of the inner list is labeled "Outer list"; a bottom bracket from the opening quote to the closing quote of the inner list is labeled "Inner list"; a bottom bracket from the opening quote to the closing quote of the first item "bananas" is labeled "Inner list"; and a bottom bracket from the opening quote of the inner list to the closing quote of the last item "pears" is labeled "Outer list continued".

```
fruits = ["bananas", ["Gala", "Empire", "McIntosh", "Golden  
Delicious"], "kiwi", "oranges", "pears"]
```

The list of apples is stored as the second item of the fruits list.

To print just the favorite apples (the inner list) from the list of favorite fruits, use the print function with the second item (index location 1) of the favorite fruits list:

The diagram shows the code `print(fruits[1])`. An arrow points from a box labeled "2nd item" to the index `[1]`. Another arrow points from a box labeled "list name" to the variable `fruits`.

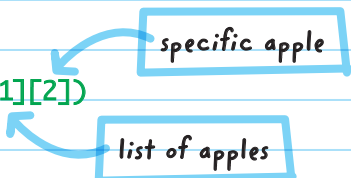
```
print(fruits[1])
```

prints: ["Gala", "Empire", "McIntosh", "Golden Delicious"]

You can print a single apple name by using the index location 1 to reference the list of apples and then adding the index location of the specific apple.

For example, to print "McIntosh" off the inner list of apples, use this code:

```
print(fruits[1][2])
```



prints: McIntosh

BOOLEAN EXPRESSIONS

In programming, it's very common to want to know if something is true or false. For example, in a game we want to know if the game is over, or if it is still going, or if a user got a question right or wrong.

Python automatically detects a Boolean-type variable when the value is set to "True" or "False," (a Boolean expression).

When you're setting a variable to "True" or "False," make sure you capitalize the T in True and the F in False. For example: `my_value = True`.

Boolean variables are like light switches. There are only two options: on or off (True or False).



When you assign a Boolean expression to a variable, Python will set the value to "True" or "False" depending on whether the Boolean expression is true or false. For example:

```
height = 58
```

Assigns "height" the value of 58.

```
meet_limit = height > 50
```

```
print(meet_limit)
```

Assigns "meet_limit" the value of "height > 50." This means that a height of greater than 50 will be considered true.

This will print: **True**

The expression is True because the given height is 58, and $58 > 50$, which means that `meet_limit` is true.

Number variables store the answer to a calculation and not the mathematical expression. Boolean variables are similar because they store "True" or "False" as the answer to a comparison expression instead of the comparison itself.

This example shows assigning the variable of `test1` to the Boolean expression `2 is equal to 4`, which is false.

```
test1 = 2 == 4
```

```
print(test1)
```

This will print: **False**

COMPARISON OPERATORS evaluate information to be true or false. They compare two values to each other.

Comparison operators are:

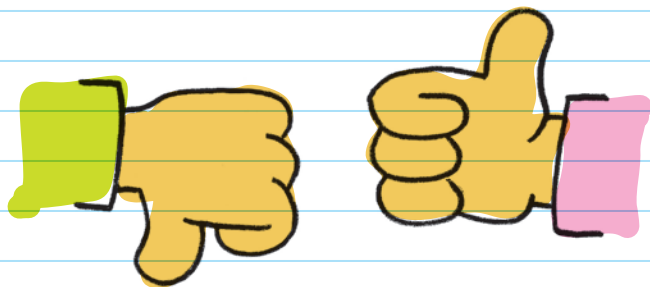
SYMBOL	MEANING	SYMBOL	MEANING
<code>==</code>	is equal to	<code>></code>	is greater than
<code>!=</code>	is not equal to	<code><=</code>	is less than or equal to
<code><</code>	is less than	<code>>=</code>	is greater than or equal to

Examples of expressions that evaluate to True:

EXPRESSION	MEANING	VALUE
<code>2 == 2</code>	2 is equal to 2	True
<code>2 != 3</code>	2 is not equal to 3	True
<code>2 < 3</code>	2 is less than 3	True
<code>4 > 3</code>	4 is greater than 3	True
<code>2 <= 2</code>	2 is less than or equal to 2	True
<code>5 >= 3</code>	5 is greater than or equal to 3	True

Examples of expressions that evaluate to False:

EXPRESSION	MEANING	VALUE
$2 == 5$	2 is equal to 5	False
$2 != 2$	2 is not equal to 2	False
$3 < 3$	3 is less than 3	False
$2 > 3$	2 is greater than 3	False
$5 \leq 3$	5 is less than or equal to 3	False
$2 \geq 3$	2 is greater than or equal to 3	False





CHECK YOUR KNOWLEDGE

1. _____ are used to store multiple values.
2. What function should you use to add an item to the end of a list?
3. How can you replace the second item in the list "cars" with "Porsche"?
4. What function should you use to add an item between two existing items in a list?
5. How do you store a list within a list?
6. Explain what the `sort()` function does.
7. Write the code that would print "bananas" given the list below:

```
fruits = ["apples", "bananas", "kiwi", "oranges", "pears"]
```

8. Which of the following does NOT evaluate to a Boolean value?

- A. True
- B. $3 ** 2$
- C. False
- D. $3 > 2$

9. What will the following program print?

```
score = 3  
game_over = score > 5  
print(game_over)
```

10. What will the following code print?

```
print(100 == 25)
```

11. How is "==" different from "=" in Python?

CHECK YOUR ANSWERS



1. Lists
2. `append()`
3. `cars[1] = "Porsche"`
4. `insert()`
5. Add a second set of brackets around the list inside another list. The inner list will take one item spot in the outer list.
6. The `sort()` function rearranges the list into numerical or alphabetical order.
7. `print(fruits[1])`
8. B
9. False
10. False
11. In Python, `"=="` is a comparison operator that checks if two values are exactly the same, but `"="` is the assignment operator and assigns a value to a variable.