**PYTHON**

**Subsetting List of Lists:**

You saw before that a Python list can contain practically anything; even other lists! To subset lists of lists, you can use the same technique as before: square brackets. Try out the commands in the following code sample in the IPython Shell:

x = [["a", "b", "c"],

["d", "e", "f"],

["g", "h", "i"]]

Print(x[2][0])

print(x[2][:2])

Ans: [“h”] [“g”,”h”]

**Replace list elements**

Replacing list elements is pretty easy. Simply subset the list and assign new values to the subset. You can select single elements or you can change entire list slices at once.

Use the IPython Shell to experiment with the commands below. Can you tell what's happening and why?

x = ["a", "b", "c", "d"]

x[1] = "r"

x[2:] = ["s", "t"]

**print(x)**

**Ans:** ["a", "r", "s", "t"]

# Extend a list

If you can change elements in a list, you sure want to be able to add elements to it, right? You can use the + operator:

x = ["a", "b", "c", "d"]

y = x + ["e", "f"]

**Ans:** ["a", "b", "c", "d", "e", "f"]

# Delete list elements

Finally, you can also remove elements from your list. You can do this with the del statement:

x = ["a", "b", "c", "d"]

del(x[1])

**Ans:** ["a", "c", "d"]

The ; sign is used to place commands on the same line. The following two code chunks are equivalent:

# Same line

command1; command2

# Separate lines

command1

command2

# Inner workings of lists

At the end of the video, Hugo explained how Python lists work behind the scenes. In this exercise you'll get some hands-on experience with this.

The Python code in the script already creates a list with the name areas and a copy named areas\_copy. Next, the first element in the areas\_copy list is changed and the areas list is printed out. If you hit Run Code you'll see that, although you've changed areas\_copy, the change also takes effect in the areas list. That's because areas and areas\_copy point to the same list.

If you want to prevent changes in areas\_copy from also taking effect in areas, you'll have to do a more explicit copy of the areas list. You can do this with [**list()**](https://docs.python.org/3/library/functions.html#func-list) or by using [:].

# Create list areas

areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Create areas\_copy

areas\_copy = areas

# Change areas\_copy

areas\_copy[0] = 5.0

# Print areas

print(areas)

[5.0, 18.0, 20.0, 10.75, 9.5]

# Create list areas

areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Create areas\_copy

areas\_copy = list(areas)

# Change areas\_copy

areas\_copy[0] = 5.0

# Print areas

print(areas)

[11.25, 18.0, 20.0, 10.75, 9.5]

# Functions

* Use [**print()**](https://docs.python.org/3/library/functions.html#print) in combination with [**type()**](https://docs.python.org/3/library/functions.html#type) to print out the type of var1.
* Use **[len()](https://docs.python.org/3/library/functions.html" \l "len" \t "_blank)** to get the length of the list var1. Wrap it in a [**print()**](https://docs.python.org/3/library/functions.html#print) call to directly print it out.
* Use [**int()**](https://docs.python.org/3/library/functions.html#int) to convert var2 to an integer. Store the output as out2.
* # Create variables var1 and var2
* var1 = [1, 2, 3, 4]
* var2 = True
* # Print out type of var1
* print(type(var1))
* # Print out length of var1
* print(len(var1))
* # Convert var2 to an integer: out2
* out2=int(var2)

<class 'list'>

4

# Help!

Maybe you already know the name of a Python function, but you still have to figure out how to use it. Ironically, you have to ask for information about a function with another function: [**help()**](https://docs.python.org/3/library/functions.html#help). In IPython specifically, you can also use ? before the function name.

To get help on the [**max()**](https://docs.python.org/3/library/functions.html#max) function, for example, you can use one of these calls:

help(max)

?max

# Multiple arguments

In the previous exercise, you identified optional arguments by viewing the documentation with help(). You'll now apply this to change the behavior of the sorted() function.

Have a look at the documentation of [**sorted()**](https://docs.python.org/3/library/functions.html#sorted) by typing help(sorted) in the IPython Shell.

You'll see that [**sorted()**](https://docs.python.org/3/library/functions.html#sorted) takes three arguments: iterable, key, and reverse.

key=None means that if you don't specify the key argument, it will be None. reverse=False means that if you don't specify the reverse argument, it will be False, by default.

In this exercise, you'll only have to specify iterable and reverse, not key. The first input you pass to [**sorted()**](https://docs.python.org/3/library/functions.html#sorted) will be matched to the iterable argument, but what about the second input? To tell Python you want to specify reverse without changing anything about key, you can use = to assign it a new value:

sorted(\_\_\_\_, reverse=\_\_\_\_)

Two lists have been created for you. Can you paste them together and sort them in descending order?

* Use + to merge the contents of first and second into a new list: full.
* Call [**sorted()**](https://docs.python.org/3/library/functions.html#sorted) on full and specify the reverse argument to be True. Save the sorted list as full\_sorted.
* Finish off by printing out full\_sorted.

# Create lists first and second

first = [11.25, 18.0, 20.0]

second = [10.75, 9.50]

# Paste together first and second: full

full=first+second

# Sort full in descending order: full\_sorted

full\_sorted=sorted(full,reverse=True)

# Print out full\_sorted

print(full\_sorted)

[20.0, 18.0, 11.25, 10.75, 9.5]

Max(first)

20.0

# String Methods

* Use the [**upper()**](https://docs.python.org/3/library/stdtypes.html#str.upper) method on place and store the result in place\_up. Use the syntax for calling methods that you learned in the previous video.
* Print out place and place\_up. Did both change?
* Print out the number of o's on the variable place by calling [**count()**](https://docs.python.org/3/library/stdtypes.html#str.count) on place and passing the letter 'o' as an input to the method. We're talking about the variable place, not the word "place"!
* Use the [**upper()**](https://docs.python.org/3/library/stdtypes.html#str.upper) method on place and store the result in place\_up. Use the syntax for calling methods that you learned in the previous video.
* Print out place and place\_up. Did both change?
* Print out the number of o's on the variable place by calling [**count()**](https://docs.python.org/3/library/stdtypes.html#str.count) on place and passing the letter 'o' as an input to the method. We're talking about the variable place, not the word "place"!
* # string to experiment with: place
* place = "poolhouse"
* # Use upper() on place: place\_up
* place\_up=place.upper()
* # Print out place and place\_up
* print(place)
* print(place\_up)
* # Print out the number of o's in place
* print(place.count('o'))

poolhouse

POOLHOUSE

3

# List Methods

Strings are not the only Python types that have methods associated with them. Lists, floats, integers and booleans are also types that come packaged with a bunch of useful methods. In this exercise, you'll be experimenting with:

* [**index()**](https://docs.python.org/3/library/stdtypes.html#str.index), to get the index of the first element of a list that matches its input and
* [**count()**](https://docs.python.org/3/library/stdtypes.html#str.count), to get the number of times an element appears in a list.

You'll be working on the list with the area of different parts of a house: areas.

* Use the [**index()**](https://docs.python.org/3/library/stdtypes.html#str.index) method to get the index of the element in areas that is equal to 20.0. Print out this index.
* Call [**count()**](https://docs.python.org/3/library/stdtypes.html#str.count) on areas to find out how many times 9.50 appears in the list. Again, simply print out this number.

# Create list areas

areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Print out the index of the element 20.0

print(areas.index(20.0))

# Print out how often 9.50 appears in areas

print(areas.count(9.50))

2

1

# List Methods (2)

Most list methods will change the list they're called on. Examples are:

* [**append()**](https://docs.python.org/3/library/stdtypes.html#typesseq-mutable), that adds an element to the list it is called on,
* [**remove()**](https://docs.python.org/3/library/stdtypes.html#typesseq-mutable), that removes the first element of a list that matches the input, and
* [**reverse()**](https://docs.python.org/3/library/stdtypes.html#typesseq-mutable), that reverses the order of the elements in the list it is called on.

You'll be working on the list with the area of different parts of the house: areas.

* Use [**append()**](https://docs.python.org/3/library/stdtypes.html#typesseq-mutable) twice to add the size of the poolhouse and the garage again: 24.5 and 15.45, respectively. Make sure to add them in this order.
* Print out areas
* Use the [**reverse()**](https://docs.python.org/3/library/stdtypes.html#typesseq-mutable) method to reverse the order of the elements in areas.
* Print out areas once more.

# Create list areas

areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Use append twice to add poolhouse and garage size

areas.append(24.5)

areas.append(15.45)

# Print out areas

print(areas)

# Reverse the orders of the elements in areas

areas.reverse()

# Print out areas

print(areas)

[11.25, 18.0, 20.0, 10.75, 9.5, 24.5, 15.45]

[15.45, 24.5, 9.5, 10.75, 20.0, 18.0, 11.25]

# Packages