**Introduction to Python #2 Python List**

**Create a list**

As opposed to int, bool etc., a list is a **compound data type**; you can group values together:

a = "is"

b = "nice"

my\_list = ["my", "list", a, b]

After measuring the height of your family, you decide to collect some information on the house you're living in. The areas of the different parts of your house are stored in separate variables for now, as shown in the script.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Create a list, areas, that contains the area of the hallway (hall), kitchen (kit), living room (liv), bedroom (bed) and bathroom (bath), in this order. Use the predefined variables.
* Print areas with the [**print()**](https://docs.python.org/3/library/functions.html#print) function.

Script.py  
01 # area variables (in square meters)

02 hall = 11.25

03 kit = 18.0

04 liv = 20.0

05 bed = 10.75

06 bath = 9.50

07

08 # Create list areas

09 areas = [hall, kit, liv, bed, bath]

10

11 # Print areas

12 print(areas)

IPython Shell

In [1]: # area variables (in square meters)

hall = 11.25

kit = 18.0

liv = 20.0

bed = 10.75

bath = 9.50

# Create list areas

areas = [hall, kit, liv, bed, bath]

# Print areas

print(areas)

[11.25, 18.0, 20.0, 10.75, 9.5]

In [2]:

**Create list with different types**

A list can contain any Python type. Although it's not really common, a list can also contain a mix of Python types including strings, floats, booleans, etc.

The printout of the previous exercise wasn't really satisfying. It's just a list of numbers representing the areas, but you can't tell which area corresponds to which part of your house.

The code on the right is the start of a solution. For some of the areas, the name of the corresponding room is already placed in front. Pay attention here! "bathroom" is a string, while bath is a variable that represents the float 9.50 you specified earlier.

**Instructions**

**100 XP**

* Finish the line of code that creates the areas list. Build the list so that the list first contains the name of each room as a string and then its area. In other words, add the strings "hallway", "kitchen" and "bedroom" at the appropriate locations.
* Print areas again; is the printout more informative this time?

Script.py  
01 # area variables (in square meters)

02 hall = 11.25

03 kit = 18.0

04 liv = 20.0

05 bed = 10.75

06 bath = 9.50

07

08 # Adapt list areas

09 areas = ["hallway", hall, "kitchen", kit, "living room", liv, "bedroom", bed, "bathroom", bath]

10

11 # Print areas

12 print(areas)  
IPython Shell  
In [1]: # area variables (in square meters)

hall = 11.25

kit = 18.0

liv = 20.0

bed = 10.75

bath = 9.50

# Adapt list areas

areas = ["hallway", hall, "kitchen", kit, "living room", liv, "bedroom", bed, "bathroom", bath]

# Print areas

print(areas)

['hallway', 11.25, 'kitchen', 18.0, 'living room', 20.0, 'bedroom', 10.75, 'bathroom', 9.5]

In [2]:

**elect the valid list**

A list can contain any Python type. But a list itself is also a Python type. That means that a list can also contain a list! Python is getting funkier by the minute, but fear not, just remember the list syntax:

my\_list = [el1, el2, el3]

Can you tell which ones of the following lines of Python code are valid ways to build a list?

A. [1, 3, 4, 2] B. [[1, 2, 3], [4, 5, 7]] C. [1 + 2, "a" \* 5, 3]

**Instructions**

**50 XP**

**Possible Answers**

* 

A, B and C

* 

B

* 

B and C

* 

C

In [1]: a = [1, 3, 4, 2]

In [2]: b = [[1, 2, 3], [4, 5, 7]]

In [3]: c = [1 + 2, "a" \* 5, 3]

**List of lists**

As a data scientist, you'll often be dealing with a lot of data, and it will make sense to group some of this data.

Instead of creating a flat list containing strings and floats, representing the names and areas of the rooms in your house, you can create a list of lists. The script on the right can already give you an idea.

Don't get confused here: "hallway" is a string, while hall is a variable that represents the float 11.25 you specified earlier.

**Instructions**

**100 XP**

* Finish the list of lists so that it also contains the bedroom and bathroom data. Make sure you enter these in order!
* Print out house; does this way of structuring your data make more sense?
* Print out the type of house. Are you still dealing with a list?

Script.py  
01 # area variables (in square meters)

02 hall = 11.25

03 kit = 18.0

04 liv = 20.0

05 bed = 10.75

06 bath = 9.50

07

08 # house information as list of lists

09 house = [["hallway", hall],

10 ["kitchen", kit],

11 ["living room", liv],

12 ["bedroom", bed],

13 ["bathroom", bath]]

14

15 # Print out house

16 print(house)

17

18 # Print out the type of house

19 print(type(house))  
IPython Shell  
In [1]: # area variables (in square meters)

hall = 11.25

kit = 18.0

liv = 20.0

bed = 10.75

bath = 9.50

# house information as list of lists

house = [["hallway", hall],

["kitchen", kit],

["living room", liv],

["bedroom", bed],

["bathroom", bath]]

# Print out house

print(house)

# Print out the type of house

print(type(house))

[['hallway', 11.25], ['kitchen', 18.0], ['living room', 20.0], ['bedroom', 10.75], ['bathroom', 9.5]]

<class 'list'>

In [2]:

**Subset and conquer**

Subsetting Python lists is a piece of cake. Take the code sample below, which creates a list x and then selects "b" from it. Remember that this is the second element, so it has index 1. You can also use negative indexing.

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

x[1]

x[-3] # same result!

Remember the areas list from before, containing both strings and floats? Its definition is already in the script. Can you add the correct code to do some Python subsetting?

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Print out the second element from the areas list (it has the value 11.25).
* Subset and print out the last element of areas, being 9.50. Using a negative index makes sense here!
* Select the number representing the area of the living room (20.0) and print it out.

Script.py

1 # Create the areas list

2 areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

3

4 # Print out second element from areas

5 print(areas[1])

IPython Shell  
In [1]: # Create the areas list

areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

# Print out second element from areas

print(areas[1])

11.25

# Subset and calculate

After you've extracted values from a list, you can use them to perform additional calculations. Take this example, where the second and fourth element of a list x are extracted. The strings that result are pasted together using the + operator:

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

print(x[1] + x[3])

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Using a combination of list subsetting and variable assignment, create a new variable, eat\_sleep\_area, that contains the sum of the area of the kitchen and the area of the bedroom.
* Print the new variable eat\_sleep\_area.

Script.py  
1 # Create the areas list

2 areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

3

4 # Sum of kitchen and bedroom area: eat\_sleep\_area

5 eat\_sleep\_area = areas[3] + areas[7]

6

7 # Print the variable eat\_sleep\_area

8 print(eat\_sleep\_area)

IPython Shell  
In [1]: # Create the areas list

areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

# Sum of kitchen and bedroom area: eat\_sleep\_area

eat\_sleep\_area = areas[3] + areas[7]

# Print the variable eat\_sleep\_area

print(eat\_sleep\_area)

28.75

In [2]:

**Slicing and dicing**

Selecting single values from a list is just one part of the story. It's also possible to *slice* your list, which means selecting multiple elements from your list. Use the following syntax:

my\_list[start:end]

The start index will be included, while the end index is *not*.

The code sample below shows an example. A list with "b" and "c", corresponding to indexes 1 and 2, are selected from a list x:

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

x[1:3]

The elements with index 1 and 2 are included, while the element with index 3 is not.

**Instructions**

**100 XP**

* Use slicing to create a list, downstairs, that contains the first 6 elements of areas.
* Do a similar thing to create a new variable, upstairs, that contains the last 4 elements of areas.
* Print both downstairs and upstairs using [**print()**](https://docs.python.org/3/library/functions.html#print).

Script.py  
01 # Create the areas list

02 areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

03

04 # Use slicing to create downstairs

05 downstairs = areas[:6]

06

07 # Use slicing to create upstairs

08 upstairs = areas[-4:]

09

10 # Print out downstairs and upstairs

11 print(downstairs)

12 print(upstairs)

IPython Shell  
In [1]: # Create the areas list

areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

# Use slicing to create downstairs

downstairs = areas[:6]

# Use slicing to create upstairs

upstairs = areas[-4:]

# Print out downstairs and upstairs

print(downstairs)

print(upstairs)

['hallway', 11.25, 'kitchen', 18.0, 'living room', 20.0]

['bedroom', 10.75, 'bathroom', 9.5]

In [2]:

**Slicing and dicing (2)**

In the video, Filip first discussed the syntax where you specify both where to begin and end the slice of your list:

my\_list[begin:end]

However, it's also possible not to specify these indexes. If you don't specify the begin index, Python figures out that you want to start your slice at the beginning of your list. If you don't specify the end index, the slice will go all the way to the last element of your list. To experiment with this, try the following commands in the IPython Shell:

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

x[:2]

x[2:]

x[:]

**Instructions**

**100 XP**

* Create downstairs again, as the first 6 elements of areas. This time, simplify the slicing by omitting the begin index.
* Create upstairs again, as the last 4 elements of areas. This time, simplify the slicing by omitting the end index.

Script.py  
1 # Create the areas list

2 areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

3

4 # Alternative slicing to create downstairs

5 downstairs = areas[:6]

6

7 # Alternative slicing to create upstairs

8 upstairs = areas[-4:]

IPython Shell  
In [1]: # Create the areas list

areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

# Alternative slicing to create downstairs

downstairs = areas[:6]

# Alternative slicing to create upstairs

upstairs = areas[-4:]

In [2]:

**Subsetting lists 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"]]

x[2][0]

x[2][:2]

x[2] results in a list, that you can subset again by adding additional square brackets.

What will house[-1][1] return? house, the list of lists that you created before, is already defined for you in the workspace. You can experiment with it in the IPython Shell.

**Instructions**

**50 XP**

**Instructions**

**50 XP**

**Possible Answers**

* 

A float: the kitchen area

* 

A string: "kitchen"

* 

A float: the bathroom area

* 

A string: "bathroom"

IPython Shell  
In [1]: print(house[-1][1])

9.5

In [2]: print(house[-1][0])

Bathroom

**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"]

For this and the following exercises, you'll continue working on the areas list that contains the names and areas of different rooms in a house.

**Instructions**

**100 XP**

* Update the area of the bathroom area to be 10.50 square meters instead of 9.50.
* Make the areas list more trendy! Change "living room" to "chill zone".

Script.py  
1 # Create the areas list

2 areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

3

4 # Correct the bathroom area

5 areas[-1] = 10.50

6

7 # Change "living room" to "chill zone"

8 areas[4] = "chill zone"  
IPython Shell

In [1]: # Create the areas list

areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

# Correct the bathroom area

areas[-1] = 10.50

# Change "living room" to "chill zone"

areas[4] = "chill zone"

In [2]:

**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"]

You just won the lottery, awesome! You decide to build a poolhouse and a garage. Can you add the information to the areas list?

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Use the + operator to paste the list ["poolhouse", 24.5] to the end of the areas list. Store the resulting list as areas\_1.
* Further extend areas\_1 by adding data on your garage. Add the string "garage" and float 15.45. Name the resulting list areas\_2.

Script.py

1 # Create the areas list and make some changes

2 areas = ["hallway", 11.25, "kitchen", 18.0, "chill zone", 20.0,

3 "bedroom", 10.75, "bathroom", 10.50]

4

5 # Add poolhouse data to areas, new list is areas\_1

6 areas\_1 = areas + ["poolhouse", 24.5]

7

8 # Add garage data to areas\_1, new list is areas\_2

9 areas\_2 = areas\_1 + ["garage", 15.45]

IPython Shell

In [1]: # Create the areas list and make some changes

areas = ["hallway", 11.25, "kitchen", 18.0, "chill zone", 20.0,

"bedroom", 10.75, "bathroom", 10.50]

# Add poolhouse data to areas, new list is areas\_1

areas\_1 = areas + ["poolhouse", 24.5]

# Add garage data to areas\_1, new list is areas\_2

areas\_2 = areas\_1 + ["garage", 15.45]

In [2]:

# 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])

Pay attention here: as soon as you remove an element from a list, the indexes of the elements that come after the deleted element all change!

The updated and extended version of areas that you've built in the previous exercises is coded below. You can copy and paste this into the IPython Shell to play around with the result.

areas = ["hallway", 11.25, "kitchen", 18.0,

"chill zone", 20.0, "bedroom", 10.75,

"bathroom", 10.50, "poolhouse", 24.5,

"garage", 15.45]

There was a mistake! The amount you won with the lottery is not that big after all and it looks like the poolhouse isn't going to happen. You decide to remove the corresponding string and float from the areas list.

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

Which of the code chunks will do the job for us?

**Instructions**

**50 XP**

**Possible Answers**

* 

del(areas[10]); del(areas[11])

* 

del(areas[10:11])

* 

del(areas[-4:-2])

* 

del(areas[-3]); del(areas[-4])

# Inner workings of lists

At the end of the video, Filip 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 [:].

##### Instructions

**100 XP**

Change the second command, that creates the variable areas\_copy, such that areas\_copy is an explicit copy of areas. After your edit, changes made to areas\_copyshouldn't affect areas. Hit Submit Answer to check this.

Script.py  
01 # Create list areas

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

03

04 # Create areas\_copy

05 areas\_copy = list(areas)

06

07 # Change areas\_copy

08 areas\_copy[0] = 5.0

09

10 # Print areas

11 print(areas)

IPython Shell

In [1]: # 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]

In [2]: