Python Lists

INTRODUCTION TO PYTHON

Hoai Thuan TRAN
Gia Dinh Univerrsity

Python Data Types

- float - real numbers
- int - integer numbers
- str - string, text
- bool - True, False

```
height = 1.73
tall = True
```

- Each variable represents single value

Problem

- Data Science: many data points
- Height of entire family

```
height1 = 1.73
height2 = 1.68
height3 = 1.71
height4 = 1.89
```

- Inconvenient

Python List

- `[a, b, c]`

```
[1.73, 1.68, 1.71, 1.89]
```

```
[1.73, 1.68, 1.71, 1.89]
```

```
fam = [1.73, 1.68, 1.71, 1.89]
fam
```

```
[1.73, 1.68, 1.71, 1.89]
```

- Name a collection of values
- Contain any type
- Contain different types

Python List

- `[a, b, c]`

```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam2 = [["liz", 1.73],
        ["emma", 1.68],
        ["mom", 1.71],
        ["dad", 1.89]]
fam2
```

```
[['liz', 1.73], ['emma', 1.68], ['mom', 1.71], ['dad', 1.89]]
```

List type

```
type(fam)
```

```
list
```

```
type(fam2)
```

```
list
```

- Specific functionality
- Specific behavior

Let's practice!

INTRODUCTION TO PYTHON

Create a list

List is a compound data type; you can group values together:

```
a = "is"
b = "nice"
my_list = ["my", "list", a, b]
```

Instructions

- 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() function.

```
# area variables (in square meters)
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50

# Create list areas


# Print areas
```

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.

Instructions

- Finish the 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?

```
# area variables (in square meters)
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50

# Adapt list areas
areas = [hall, kit, "living room", liv, bed,
"bathroom", bath]

# Print areas
```

Select 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!

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]`

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 in the editor can already give you an idea.

Instructions

- 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?

```
# 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]]

# Print out house


# Print out the type of house
```

Subseting Lists

INTRODUCTION TO PYTHON

Hoai Thuan TRAN
Gia Dinh Univerrsity

Subseting lists

```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[3]
```

```
1.68
```

Subseting lists

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[6]
```

```
'dad'
```

```
fam[-1]
```

```
1.89
```

```
fam[7]
```

```
1.89
```

Subseting lists

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[6]
```

```
'dad'
```

```
fam[-1]   # <-
```

```
1.89
```

```
fam[7] # <-
```

```
1.89
```

List slicing

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[3:5]
```

```
[1.68, 'mom']
```

```
fam[1:4]
```

```
[1.73, 'emma', 1.68]
```

```
[ start : end ]
```

inclusive exclusive

List slicing

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[:4]
```

```
['liz', 1.73, 'emma', 1.68]
```

```
fam[5:]
```

```
[1.71, 'dad', 1.89]
```

Let's practice!

INTRODUCTION TO PYTHON

Subset list

To select “b” from the list x. We can do as follow:

```
x = ["a", "b", "c", "d"]
x[1]
x[-3] # same result!
```

Instructions

- 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.

```
# 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 out last element from areas

# Print out the area of the living room
```

Subset and calculate

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

- 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.

```
# 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


# Print the variable eat_sleep_area
```

Slicing and dicing

It's also possible to *slice* your list, which means selecting multiple elements from your list. Use the following syntax:

```
my_list[start:end]
```

Instructions

- 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.

```
# 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


# Use slicing to create upstairs


# Print out downstairs and upstairs
```

Slicing and dicing

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.

Try the following code:

```
x = ["a", "b", "c", "d"]
x[:2]
x[2:]
x[:]
```

Instructions

- 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.
- Print both downstairs and upstairs.

```
# 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

# Alternative slicing to create upstairs
```

Manipulating Lists

INTRODUCTION TO PYTHON

Hoai Thuan TRAN
Gia Dinh Univerrsity

List Manipulation

- Change list elements
- Add list elements
- Remove list elements

Changing list elements

```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[7] = 1.86
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
```

```
fam[0:2] = ["lisa", 1.74]
fam
```

```
['lisa', 1.74, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
```

Adding and removing elements

```
fam + ["me", 1.79]
```

```
['lisa', 1.74,'emma', 1.68, 'mom', 1.71, 'dad', 1.86, 'me', 1.79]
```

```
fam_ext = fam + ["me", 1.79]
del(fam[2])
fam
```

```
['lisa', 1.74, 1.68, 'mom', 1.71, 'dad', 1.86]
```

Behind the scenes (1)

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

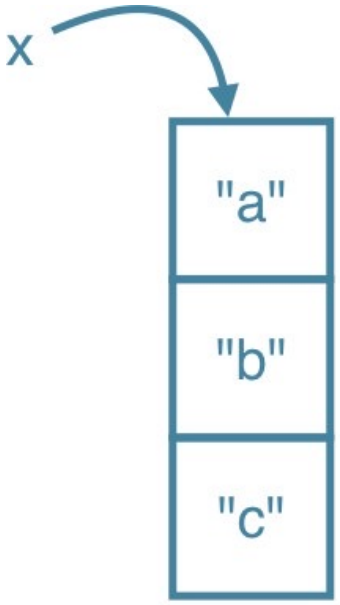
Behind the scenes (1)

```
x = ["a", "b", "c"]
y  =  x
y[1] = "z"
y
```

```
['a', 'z', 'c']
```

```
x
```

```
['a', 'z', 'c']
```



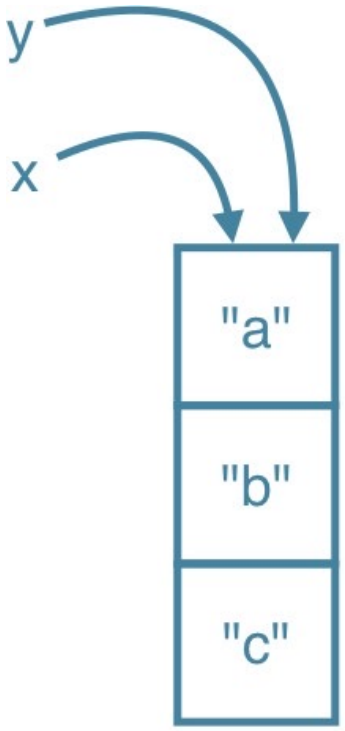
Behind the scenes (1)

```
x = ["a", "b", "c"]
y  =  x
y[1] = "z"
y
```

```
['a', 'z', 'c']
```

```
x
```

```
['a', 'z', 'c']
```



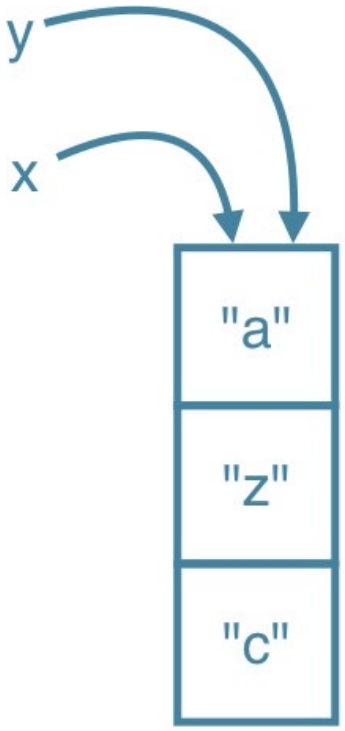
Behind the scenes (1)

```
x = ["a", "b", "c"]
y  =  x
y[1] = "z"
y
```

```
['a', 'z', 'c']
```

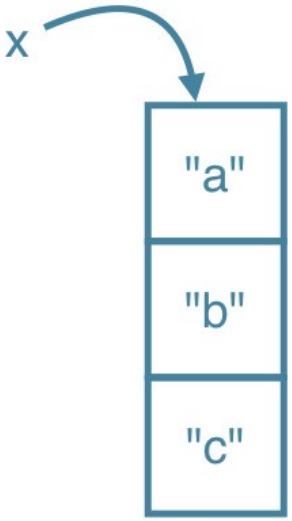
```
x
```

```
['a', 'z', 'c']
```



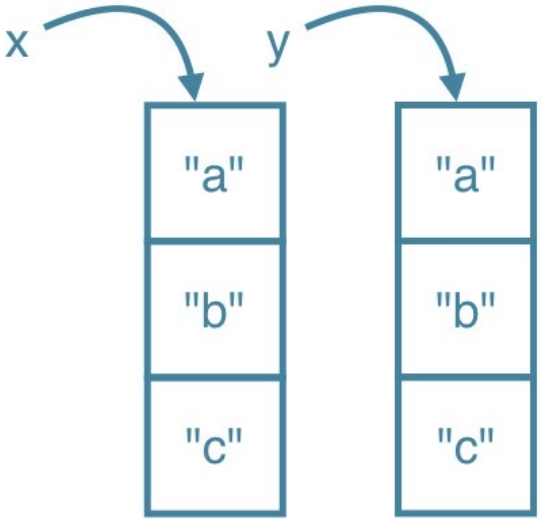
Behind the scenes (2)

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



Behind the scenes (2)

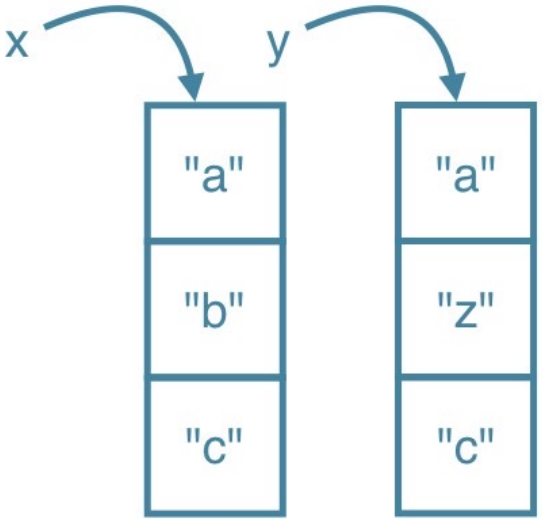
```
x = ["a", "b", "c"]
y = list(x)
y = x[:]
```



Behind the scenes (2)

```
x = ["a", "b", "c"]
y = list(x)
y = x[:]
y[1] = "z"
x
```

```
['a', 'b', 'c']
```



Let's practice!

INTRODUCTION TO PYTHON

Replace list elements

Subset the list and assign new values to the subset. You can select single elements or you can change entire list slices at once.

Try the following code:

```
x = ["a", "b", "c", "d"]
x[1] = "r"
x[2:] = ["s", "t"]
```

Instructions

- 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".

```
# 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


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

Extend a list

You can use the + operator to add elements to the list.

Try the following code:

```
x = ["a", "b", "c", "d"]
y = x + ["e", "f"]
```

Instructions

- 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.

```
# 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


# Add garage data to areas_1, new list is areas_2
```

Delete list elements

You can also remove elements from your list with the del statement:

Try the following code:

```
x = ["a", "b", "c", "d"]
del(x[1])
```

Instructions

- Remove the poolhouse and it's area from the areas list.

```
# 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,"poolhouse", 24.5,
"garage", 15.45]

# Remove the poolhouse and it area.
```

Inner workings of lists

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() or by using **[:]**.

Instructions

- 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_copy shouldn't affect areas.

```
# 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)
```