Motivation for dictionaries

Exercise

To see why dictionaries are useful, have a look at the two lists defined in the script. countries contains the names of some European countries. capitals lists the corresponding names of their capital.

Instructions

Use the index() method on countries to find the index of 'germany'. Store this index as ind_ger.

Use ind_ger to access the capital of Germany from the capitals list. Print it out.

In [ ]:

```
# Definition of countries and capital
countries = ['spain', 'france', 'germany', 'norway']
capitals = ['madrid', 'paris', 'berlin', 'oslo']

# Get index of 'germany': ind_ger

# Use ind_ger to print out capital of Germany

#_____________________________________________#
#Solutions

# Definition of countries and capital
countries = ['spain', 'france', 'germany', 'norway']
capitals = ['madrid', 'paris', 'berlin', 'oslo']

# Get index of 'germany': ind_ger
ind_ger= countries.index('germany')

# Use ind_ger to print out capital of Germany
print(capitals[ind_ger])
```

Create dictionary

Exercise

The countries and capitals lists are again available in the script. It's your job to convert this data to a dictionary where the country names are the keys and the capitals are the corresponding values. As a refresher, here is a recipe for creating a dictionary:

my_dict = { "key1":"value1", "key2":"value2", }

In this recipe, both the keys and the values are strings. This will also be the case for this exercise.

Instructions

With the strings in countries and capitals, create a dictionary called europe with 4 key:value pairs. Beware of capitalization!

Make sure you use lowercase characters everywhere.

Print out europe to see if the result is what you expected.

In [ ]:

```
# Definition of countries and capital
countries = ['spain', 'france', 'germany', 'norway']
capitals = ['madrid', 'paris', 'berlin', 'oslo']

# From string in countries and capitals, create dictionary europe
europe = { 'spain':'madrid', ___, ___, ___ }

# Print europe

#_____________________________________________#
#Solutions

# Definition of countries and capital
countries = ['spain', 'france', 'germany', 'norway']
capitals = ['madrid', 'paris', 'berlin', 'oslo']

# From string in countries and capitals, create dictionary europe
europe = { 'spain':'madrid','france':'paris', 'germany':'berlin', 'norway':'oslo'}

# Print europe
print(europe)
```

Access dictionary

Exercise

If the keys of a dictionary are chosen wisely, accessing the values in a dictionary is easy and intuitive. For example, to get the capital for France from europe you can use:

europe['france']

Here, 'france' is the key and 'paris' the value is returned.

Instructions

Check out which keys are in europe by calling the keys() method on europe. Print out the result.

Print out the value that belongs to the key 'norway'

In [ ]:

```
# Definition of dictionary
europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo' }

# Print out the keys in europe

# Print out value that belongs to key 'norway'

#_____________________________________________#
#Solutions

# Definition of dictionary
europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo' }

# Print out the keys in europe
print( europe.keys() )

# Print out value that belongs to key 'norway'
print( europe['norway'])
```

Dictionary Manipulation (1)

Exercise

If you know how to access a dictionary, you can also assign a new value to it. To add a new key-value pair to europe you can use something like this:

europe['iceland'] = 'reykjavik'

Instructions

Add the key 'italy' with the value 'rome' to europe.

To assert that 'italy' is now a key in europe, print out 'italy' in europe.

Add another key:value pair to europe: 'poland' is the key, 'warsaw' is the corresponding value.

Print out europe.

In [ ]:

```
# Definition of dictionary
europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo' }

# Add italy to europe

# Print out italy in europe

# Add poland to europe

# Print europe

#_____________________________________________#
#Solutions

# Definition of dictionary
europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo' }

# Add italy to europe
europe['italy']='rome'

# Print out italy in europe
print('italy' in europe)

# Add poland to europe
europe['poland']='warsaw'


# Print europe
print(europe)
```

Dictionary Manipulation (2)

Exercise

Somebody thought it would be funny to mess with your accurately generated dictionary. An adapted version of the europe dictionary is available in the script.

Can you clean up? Do not do this by adapting the definition of europe, but by adding Python commands to the script to update and remove key:value pairs.

Instructions

The capital of Germany is not 'bonn'; it's 'berlin'. Update its value.

Australia is not in Europe, Austria is! Remove the key 'australia' from europe.

Print out europe to see if your cleaning work paid off.

In [ ]:

```
# Definition of dictionary
europe = {'spain':'madrid', 'france':'paris', 'germany':'bonn',
          'norway':'oslo', 'italy':'rome', 'poland':'warsaw',
          'australia':'vienna' }

# Update capital of germany

# Remove australia

# Print europe

#_____________________________________________#
#Solutions
# Definition of dictionary
europe = {'spain':'madrid', 'france':'paris', 'germany':'bonn',
          'norway':'oslo', 'italy':'rome', 'poland':'warsaw',
          'australia':'vienna' }

# Update capital of germany
europe['germany']='berlin'

# Remove australia
del(europe['australia'])

# Print europe
print(europe)
```

Dictionariception

Exercise

Remember lists? They could contain anything, even other lists. Well, for dictionaries the same holds. Dictionaries can contain key:value pairs where the values are again dictionaries.

As an example, have a look at the script where another version of europe - the dictionary you've been working with all along - is coded. The keys are still the country names, but the values are dictionaries that contain more information than just the capital.

It's perfectly possible to chain square brackets to select elements. To fetch the population for Spain from europe, for example, you need:

europe['spain']['population']

Instructions

Use chained square brackets to select and print out the capital of France.

Create a dictionary, named data, with the keys 'capital' and 'population'. Set them to 'rome' and 59.83, respectively.

Add a new key-value pair to europe; the key is 'italy' and the value is data, the dictionary you just built.

In [ ]:

```
# Print out the capital of France

# Create sub-dictionary data

# Add data to europe under key 'italy'

# Print europe

#_____________________________________________#
#Solutions

# Dictionary of dictionaries
europe = { 'spain': { 'capital':'madrid', 'population':46.77 },
           'france': { 'capital':'paris', 'population':66.03 },
           'germany': { 'capital':'berlin', 'population':80.62 },
           'norway': { 'capital':'oslo', 'population':5.084 } }

# Print out the capital of France
print( europe['france']['capital'] )

# Create sub-dictionary data
data={'capital':'rome', 'population':59.83  }

# Add data to europe under key 'italy'
europe['italy']=data

# Print europe
print(europe)
```

Dictionary to DataFrame (1)

Exercise

Pandas is an open source library, providing high-performance, easy-to-use data structures and data analysis tools for Python. Sounds promising!

The DataFrame is one of Pandas' most important data structures. It's basically a way to store tabular data where you can label the rows and the columns. One way to build a DataFrame is from a dictionary.

In the exercises that follow you will be working with vehicle data from different countries. Each observation corresponds to a country and the columns give information about the number of vehicles per capita, whether people drive left or right, and so on.

Three lists are defined in the script:

names, containing the country names for which data is available.

dr, a list with booleans that tells whether people drive left or right in the corresponding country.

cpc, the number of motor vehicles per 1000 people in the corresponding country.

Each dictionary key is a column label and each value is a list which contains the column elements.

Instructions

Import pandas as pd.

Use the pre-defined lists to create a dictionary called my_dict. There should be three key value

pairs:

key 'country' and value names.

key 'drives_right' and value dr.

key 'cars_per_cap' and value cpc.

Use pd.DataFrame() to turn your dict into a DataFrame called cars.

Print out cars and see how beautiful it is.# Dictionary of dictionaries

europe = { 'spain': { 'capital':'madrid', 'population':46.77 }, 'france': { 'capital':'paris', 'population':66.03 }, 'germany': { 'capital':'berlin', 'population':80.62 }, 'norway': { 'capital':'oslo', 'population':5.084 } }

In [ ]:

```
# Pre-defined lists
names = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco', 'Egypt']
dr =  [True, False, False, False, True, True, True]
cpc = [809, 731, 588, 18, 200, 70, 45]

# Import pandas as pd

# Create dictionary my_dict with three key:value pairs: my_dict

# Build a DataFrame cars from my_dict: cars

# Print cars

#_____________________________________________#
#Solutions

# Pre-defined lists
names = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco', 'Egypt']
dr =  [True, False, False, False, True, True, True]
cpc = [809, 731, 588, 18, 200, 70, 45]

# Import pandas as pd
import pandas as pd

# Create dictionary my_dict with three key:value pairs: my_dict
my_dict={'country':names, 'drives_right':dr, 'cars_per_cap':cpc  }

# Build a DataFrame cars from my_dict: cars
cars=pd.DataFrame(my_dict)

# Print cars
print(cars)
```

Dictionary to DataFrame (2)

Exercise

The Python code that solves the previous exercise is included in the script. Have you noticed that the row labels (i.e. the labels for the different observations) were automatically set to integers from 0 up to 6?

To solve this a list row_labels has been created. You can use it to specify the row labels of the cars DataFrame. You do this by setting the index attribute of cars, that you can access as cars.index.

Instructions

Hit Run Code to see that, indeed, the row labels are not correctly set.

Specify the row labels by setting cars.index equal to row_labels.

Print out cars again and check if the row labels are correct this time.

In [ ]:

```
import pandas as pd

# Build cars DataFrame
names = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco', 'Egypt']
dr =  [True, False, False, False, True, True, True]
cpc = [809, 731, 588, 18, 200, 70, 45]
cars_dict = { 'country':names, 'drives_right':dr, 'cars_per_cap':cpc }
cars = pd.DataFrame(cars_dict)
print(cars)

# Definition of row_labels
row_labels = ['US', 'AUS', 'JPN', 'IN', 'RU', 'MOR', 'EG']

# Specify row labels of cars

# Print cars again

#_____________________________________________#
#Solutions

import pandas as pd

# Build cars DataFrame
names = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco', 'Egypt']
dr =  [True, False, False, False, True, True, True]
cpc = [809, 731, 588, 18, 200, 70, 45]
cars_dict = { 'country':names, 'drives_right':dr, 'cars_per_cap':cpc }
cars = pd.DataFrame(cars_dict)
print(cars)

# Definition of row_labels
row_labels = ['US', 'AUS', 'JPN', 'IN', 'RU', 'MOR', 'EG']

# Specify row labels of cars
cars.index= row_labels

# Print cars again
print(cars)
```

CSV to DataFrame (1)

Exercise

Putting data in a dictionary and then building a DataFrame works, but it's not very efficient. What if you're dealing with millions of observations? In those cases, the data is typically available as files with a regular structure. One of those file types is the CSV file, which is short for "comma-separated values".

To import CSV data into Python as a Pandas DataFrame you can use read_csv().

Let's explore this function with the same cars data from the previous exercises. This time, however, the data is available in a CSV file, named cars.csv. It is available in your current working directory, so the path to the file is simply 'cars.csv'.

Instructions

To import CSV files you still need the pandas package: import it as pd.

Use pd.read_csv() to import cars.csv data as a DataFrame. Store this DataFrame as cars.

Print out cars. Does everything look OK?

In [ ]:

```
# Import pandas as pd

# Import the cars.csv data: cars

# Print out cars

#_____________________________________________#
#Solutions

# Import pandas as pd
import pandas as pd

# Import the cars.csv data: cars
cars=pd.read_csv('cars.csv')

# Print out cars
print(cars)
```

CSV to DataFrame (2)

Exercise

Your read_csv() call to import the CSV data didn't generate an error, but the output is not entirely what we wanted. The row labels were imported as another column without a name.

Remember index_col, an argument of read_csv(), that you can use to specify which column in the CSV file should be used as a row label? Well, that's exactly what you need here!

Python code that solves the previous exercise is already included; can you make the appropriate changes to fix the data import?

Instructions

Run the code with Run Code and assert that the first column should actually be used as row labels.

Specify the index_col argument inside pd.read_csv(): set it to 0, so that the first column is used as row labels.

In [ ]:

```
# Import pandas as pd
import pandas as pd

# Fix import by including index_col
cars = pd.read_csv('cars.csv')

# Print out cars
print(cars)
#_____________________________________________#
#Solutions

# Import pandas as pd
import pandas as pd

# Fix import by including index_col
cars = pd.read_csv('cars.csv',index_col=0)

# Print out cars
print(cars)
```

Square Brackets (1)

Exercise

In the video, you saw that you can index and select Pandas DataFrames in many different ways. The simplest, but not the most powerful way, is to use square brackets.

In the sample code, the same cars data is imported from a CSV files as a Pandas DataFrame. To select only the cars_per_cap column from cars, you can use:

cars['cars_per_cap']

cars[['cars_per_cap']]

The single bracket version gives a Pandas Series, the double bracket version gives a Pandas DataFrame.

Instructions

Use single square brackets to print out the country column of cars as a Pandas Series.

Use double square brackets to print out the country column of cars as a Pandas DataFrame.

Use double square brackets to print out a DataFrame with both the country and drives_right columns of cars, in this order.

In [ ]:

```
# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Print out country column as Pandas Series

# Print out country column as Pandas DataFrame

# Print out DataFrame with country and drives_right columns

#_____________________________________________#
#Solutions

# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Print out country column as Pandas Series
print(cars['country'])


# Print out country column as Pandas DataFrame
print(cars[['country']])


# Print out DataFrame with country and drives_right columns
print(cars[['country', 'drives_right']])
```

Square Brackets (2)

Exercise

Square brackets can do more than just selecting columns. You can also use them to get rows, or observations, from a DataFrame. The following call selects the first five rows from the cars DataFrame:

cars[0:5]

The result is another DataFrame containing only the rows you specified.

Pay attention: You can only select rows using square brackets if you specify a slice, like 0:4. Also, you're using the integer indexes of the rows here, not the row labels!

Instructions

Select the first 3 observations from cars and print them out.

In [ ]:

```
# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Print out first 3 observations

# Print out fourth, fifth and sixth observation

#_____________________________________________#
#Solutions

# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Print out first 3 observations
print(cars[0:3])

# Print out fourth, fifth and sixth observation
print(cars[3:6])
```

loc and iloc (1)

Exercise

With loc and iloc you can do practically any data selection operation on DataFrames you can think of. loc is label-based, which means that you have to specify rows and columns based on their row and column labels. iloc is integer index based, so you have to specify rows and columns by their integer index like you did in the previous exercise.

Try out the following commands in the IPython Shell to experiment with loc and iloc to select observations. Each pair of commands here gives the same result.

cars.loc['RU']

cars.iloc[4]

cars.loc[['RU']]

cars.iloc[[4]]

cars.loc[['RU', 'AUS']]

cars.iloc[[4, 1]]

As before, code is included that imports the cars data as a Pandas DataFrame.

Instructions

Use loc or iloc to select the observation corresponding to Japan as a Series. The label of this row is JPN, the index is 2. Make sure to print the resulting Series.

Use loc or iloc to select the observations for Australia and Egypt as a DataFrame. You can find out about the labels/indexes of these rows by inspecting cars in the IPython Shell. Make sure to print the resulting DataFrame.# Import cars data

In [ ]:

```
# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Print out observation for Japan

# Print out observations for Australia and Egypt

#_____________________________________________#
#Solutions

# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Print out observation for Japan
print(cars.loc['JPN'])

# Print out observations for Australia and Egypt
print(cars.loc[['AUS', 'EG']])
```

loc and iloc (2)

Exercise

loc and iloc also allow you to select both rows and columns from a DataFrame. To experiment, try out the following commands in the IPython Shell. Again, paired commands produce the same result.

cars.loc['IN', 'cars_per_cap']

cars.iloc[3, 0]

cars.loc[['IN', 'RU'], 'cars_per_cap']

cars.iloc[[3, 4], 0]

cars.loc[['IN', 'RU'], ['cars_per_cap', 'country']]

cars.iloc[[3, 4], [0, 1]]

Instructions

Print out the drives_right value of the row corresponding to Morocco (its row label is MOR)

Print out a sub-DataFrame, containing the observations for Russia and Morocco and the columns country and drives_right.

In [ ]:

```
# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Print out drives_right value of Morocco

# Print sub-DataFrame

#_____________________________________________#
#Solutions

# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Print out drives_right value of Morocco
print(cars.iloc[5, 2])

# Print sub-DataFrame
print(cars.loc[['RU', 'MOR'], ['country', 'drives_right']])
```

loc and iloc (3)

Exercise

It's also possible to select only columns with loc and iloc. In both cases, you simply put a slice going from beginning to end in front of the comma:

cars.loc[:, 'country']

cars.iloc[:, 1]

cars.loc[:, ['country','drives_right']]

cars.iloc[:, [1, 2]]

Instructions

Print out the drives_right column as a Series using loc or iloc.

Print out the drives_right column as a DataFrame using loc or iloc.

Print out both the cars_per_cap and drives_right column as a DataFrame using loc or iloc.

In [ ]:

```
# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Print out drives_right column as Series

# Print out drives_right column as DataFrame

# Print out cars_per_cap and drives_right as DataFrame

#_____________________________________________#
#Solutions

# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)
cars
# Print out drives_right column as Series
print(cars.loc[:, 'drives_right'])

# Print out drives_right column as DataFrame
print(cars.loc[:, ['drives_right']])

# Print out cars_per_cap and drives_right as DataFrame
print(cars.loc[:, ['cars_per_cap','drives_right']])
```