**Intermediate Python for Data Science**

**#4 Loops**

**while: warming up**

The while loop is like a repeated if statement. The code is executed over and over again, as long as the condition is True. Have another look at its recipe.

while condition :

expression

Can you tell how many printouts the following while loop will do?

x = 1

while x < 4 :

print(x)

x = x + 1

**Instructions**

**50 XP**

**Instructions**

**50 XP**

**Possible Answers**

* 

0

* 

1

* 

2

* 

3

* 

4

**Basic while loop**

Below you can find the example from the video where the errorvariable, initially equal to 50.0, is divided by 4 and printed out on every run:

error = 50.0

while error > 1 :

error = error / 4

print(error)

This example will come in handy, because it's time to build a while loop yourself! We're going to code a while loop that implements a very basic control system for an [**inverted pendulum**](https://en.wikipedia.org/wiki/Inverted_pendulum). If there's an offset from standing perfectly straight, the while loop will incrementally fix this offset.

Note that if your while loop takes too long to run, you might have made a mistake!

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Create the variable offset with an initial value of 8.
* Code a while loop that keeps running as long as offset is not equal to 0. Inside the while loop:
* Print out the sentence "correcting...".
* Next, decrease the value of offset by 1. You can do this with offset = offset - 1.
* Finally, print out offset so you can see how it changes.

Script.py  
1 # Initialize offset

2 offset = 8

3

4 # Code the while loop

5 while offset != 0 :

6 print("correcting...")

7 offset = offset - 1

8 print(offset)

IPytheon Shell  
In [1]: # Initialize offset

offset = 8

# Code the while loop

while offset != 0 :

print("correcting...")

offset = offset - 1

print(offset)

correcting...

7

correcting...

6

correcting...

5

correcting...

4

correcting...

3

correcting...

2

correcting...

1

correcting...

0

<script.py> output:

correcting...

7

correcting...

6

correcting...

5

correcting...

4

correcting...

3

correcting...

2

correcting...

1

correcting...

0

In [2]:

**Add conditionals**

The while loop that corrects the offset is a good start, but what if offset is negative? You can try to run the following code where offset is initialized to -6:

# Initialize offset

offset = -6

# Code the while loop

while offset != 0 :

print("correcting...")

offset = offset - 1

print(offset)

but your session will be disconnected. The while loop will never stop running, because offset will be further decreased on every run. offset != 0 will never become False and the whileloop continues forever.

Fix things by putting an if-else statement inside the whileloop. If your code is still taking too long to run, you probably made a mistake!

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* **Inside** the while loop, replace offset = offset - 1 by an if-else statement:
  + If offset is greater than zero, you should decrease offset by 1.
  + Else, you should increase offset by 1.
* If you've coded things correctly, hitting *Submit Answer* should work this time. If your code is still taking too long to run, you probably made a mistake!

Script.py  
01 # Initialize offset

02 offset = -6

03

04 # Code the while loop

05 while offset != 0 :

06 print("correcting...")

07 if offset > 0 :

08 offset = offset -1

09 else :

10 offset = offset + 1

11 print(offset)

IPython Shell  
In [1]: # Initialize offset

offset = -6

# Code the while loop

while offset != 0 :

print("correcting...")

if offset > 0 :

offset = offset -1

else :

offset = offset + 1

print(offset)

correcting...

-5

correcting...

-4

correcting...

-3

correcting...

-2

correcting...

-1

correcting...

0

<script.py> output:

correcting...

-5

correcting...

-4

correcting...

-3

correcting...

-2

correcting...

-1

correcting...

0

In [2]:

# Loop over a list

Have another look at the for loop that Filip showed in the video:

fam = [1.73, 1.68, 1.71, 1.89]

for height in fam :

print(height)

As usual, you simply have to indent the code with 4 spaces to tell Python which code should be executed in the for loop.

The areas variable, containing the area of different rooms in your house, is already defined.

##### Instructions

**100 XP**

##### Instructions

**100 XP**

Write a for loop that iterates over all elements of the areas list and prints out every element separately.

Script.py  
1 # areas list

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

3

4 # Code the for loop

5 for height in areas :

6 print(height)

IPytheon Shell  
In [1]: # areas list

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

# Code the for loop

for height in areas :

print(height)

11.25

18.0

20.0

10.75

9.5

<script.py> output:

11.25

18.0

20.0

10.75

9.5

In [2]:

**Indexes and values (1)**

Using a for loop to iterate over a list only gives you access to every list element in each run, one after the other. If you also want to access the index information, so where the list element you're iterating over is located, you can use [**enumerate()**](https://docs.python.org/3/library/functions.html#enumerate).

As an example, have a look at how the for loop from the video was converted:

fam = [1.73, 1.68, 1.71, 1.89]

for index, height in enumerate(fam) :

print("person " + str(index) + ": " + str(height))

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Adapt the for loop in the sample code to use [**enumerate()**](https://docs.python.org/3/library/functions.html#enumerate)and use two iterator variables.
* Update the print() statement so that on each run, a line of the form "room x: y" should be printed, where x is the index of the list element and y is the actual list element, i.e. the area. Make sure to print out this exact string, with the correct spacing.

Script.py  
1 # areas list

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

3

4 # Change for loop to use enumerate() and update print()

5 for index, area in enumerate(areas) :

6 print("room " + str(index) + ": " + str(area))

IPython Shell  
In [1]: # areas list

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

# Change for loop to use enumerate() and update print()

for index, area in enumerate(areas) :

print("room " + str(index) + ": " + str(area))

room 0: 11.25

room 1: 18.0

room 2: 20.0

room 3: 10.75

room 4: 9.5

<script.py> output:

room 0: 11.25

room 1: 18.0

room 2: 20.0

room 3: 10.75

room 4: 9.5

In [2]:

# Indexes and values (2)

For non-programmer folks, room 0: 11.25 is strange. Wouldn't it be better if the count started at 1?

##### Instructions

**100 XP**

Adapt the [**print()**](https://docs.python.org/3/library/functions.html#print) function in the for loop on the right so that the first printout becomes "room 1: 11.25", the second one "room 2: 18.0" and so on.

Script.py  
1 # areas list

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

3

4 # Code the for loop

5 for index, area in enumerate(areas) :

6 print("room " + str(index + 1) + ": " + str(area))

IPython Shell  
In [1]: # areas list

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

# Code the for loop

for index, area in enumerate(areas) :

print("room " + str(index + 1) + ": " + str(area))

room 1: 11.25

room 2: 18.0

room 3: 20.0

room 4: 10.75

room 5: 9.5

<script.py> output:

room 1: 11.25

room 2: 18.0

room 3: 20.0

room 4: 10.75

room 5: 9.5

In [2]:

# Loop over list of lists

Remember the house variable from the Intro to Python course? Have a look at its definition on the right. It's basically a list of lists, where each sublist contains the name and area of a room in your house.

It's up to you to build a for loop from scratch this time!

##### Instructions

**100 XP**

Write a for loop that goes through each sublist of house and prints out the x is y sqm, where x is the name of the room and y is the area of the room.

Script.py  
01 # house list of lists

02 house = [["hallway", 11.25],

03 ["kitchen", 18.0],

04 ["living room", 20.0],

05 ["bedroom", 10.75],

06 ["bathroom", 9.50]]

07

08 # Build a for loop from scratch

09 for x in house :

10 print("the " + x[0] + " is " + str(x[1]) + " sqm")

IPython Shell  
In [1]: # house list of lists

house = [["hallway", 11.25],

["kitchen", 18.0],

["living room", 20.0],

["bedroom", 10.75],

["bathroom", 9.50]]

# Build a for loop from scratch

for x in house :

print("the " + x[0] + " is " + str(x[1]) + " sqm")

the hallway is 11.25 sqm

the kitchen is 18.0 sqm

the living room is 20.0 sqm

the bedroom is 10.75 sqm

the bathroom is 9.5 sqm

<script.py> output:

the hallway is 11.25 sqm

the kitchen is 18.0 sqm

the living room is 20.0 sqm

the bedroom is 10.75 sqm

the bathroom is 9.5 sqm

In [2]:

# Loop over dictionary

In Python 3, you need the [**items()**](https://docs.python.org/3/library/stdtypes.html#dict.items) method to loop over a dictionary:

world = { "afghanistan":30.55,

"albania":2.77,

"algeria":39.21 }

for key, value in world.items() :

print(key + " -- " + str(value))

Remember the europe dictionary that contained the names of some European countries as key and their capitals as corresponding value? Go ahead and write a loop to iterate over it!

##### Instructions

**100 XP**

Write a for loop that goes through each key:value pair of europe. On each iteration, "the capital of x is y" should be printed out, where x is the key and y is the value of the pair.

Script.py  
1 # Definition of dictionary

2 europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin',

3 'norway':'oslo', 'italy':'rome', 'poland':'warsaw', 'austria':'vienna' }

4

5 # Iterate over europe

6 for key, value in europe.items() :

7 print('the capital of ' + key + ' is ' + value)

IPython Shell  
In [1]: # Definition of dictionary

europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin',

'norway':'oslo', 'italy':'rome', 'poland':'warsaw', 'austria':'vienna' }

# Iterate over europe

for key, value in europe.items() :

print('the capital of ' + key + ' is ' + value)

the capital of france is paris

the capital of spain is madrid

the capital of germany is berlin

the capital of poland is warsaw

the capital of norway is oslo

the capital of austria is vienna

the capital of italy is rome

<script.py> output:

the capital of france is paris

the capital of spain is madrid

the capital of germany is berlin

the capital of poland is warsaw

the capital of norway is oslo

the capital of austria is vienna

the capital of italy is rome

In [2]:

**Loop over Numpy array**

If you're dealing with a 1D Numpy array, looping over all elements can be as simple as:

for x in my\_array :

...

If you're dealing with a 2D Numpy array, it's more complicated. A 2D array is built up of multiple 1D arrays. To explicitly iterate over all separate elements of a multi-dimensional array, you'll need this syntax:

for x in np.nditer(my\_array) :

...

Two Numpy arrays that you might recognize from the intro course are available in your Python session: np\_height, a Numpy array containing the heights of Major League Baseball players, and np\_baseball, a 2D Numpy array that contains both the heights (first column) and weights (second column) of those players.

**Instructions**

**100 XP**

* Import the numpy package under the local alias np.
* Write a for loop that iterates over all elements in np\_heightand prints out "x inches" for each element, where x is the value in the array.
* Write a for loop that visits every element of the np\_baseball array and prints it out.

Script.py  
01 # Import numpy as np

02 import numpy as np

03

04 # For loop over np\_height

05 for x in np\_height :

06 print(str(x) + " inches")

07

08 # For loop over np\_baseball

09 for x in np.nditer(np\_baseball) :

10 print(x)

# Loop over DataFrame (1)

Iterating over a Pandas DataFrame is typically done with the **[iterrows()](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.iterrows.html" \t "_blank)** method. Used in a for loop, every observation is iterated over and on every iteration the row label and actual row contents are available:

for lab, row in brics.iterrows() :

...

In this and the following exercises you will be working on the cars DataFrame. It contains information on the cars per capita and whether people drive right or left for seven countries in the world.

##### Instructions

**100 XP**

Write a for loop that iterates over the rows of cars and on each iteration perform two [**print()**](https://docs.python.org/3/library/functions.html#print) calls: one to print out the row label and one to print out all of the rows contents.

Script.py  
1 # Import cars data

2 import pandas as pd

3 cars = pd.read\_csv('cars.csv', index\_col = 0)

4

5 # Iterate over rows of cars

6 for lab, row in cars.iterrows() :

7 print(lab)

8 print(row)

IPython Shell  
In [1]: # Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Iterate over rows of cars

for lab, row in cars.iterrows() :

print(lab)

print(row)

US

cars\_per\_cap 809

country United States

drives\_right True

Name: US, dtype: object

AUS

cars\_per\_cap 731

country Australia

drives\_right False

Name: AUS, dtype: object

JAP

cars\_per\_cap 588

country Japan

drives\_right False

Name: JAP, dtype: object

IN

cars\_per\_cap 18

country India

drives\_right False

Name: IN, dtype: object

RU

cars\_per\_cap 200

country Russia

drives\_right True

Name: RU, dtype: object

MOR

cars\_per\_cap 70

country Morocco

drives\_right True

Name: MOR, dtype: object

EG

cars\_per\_cap 45

country Egypt

drives\_right True

Name: EG, dtype: object

In [2]:

# Loop over DataFrame (2)

The row data that's generated by **[iterrows()](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.iterrows.html" \t "_blank)** on every run is a Pandas Series. This format is not very convenient to print out. Luckily, you can easily select variables from the Pandas Series using square brackets:

for lab, row in brics.iterrows() :

print(row['country'])

##### Instructions

**100 XP**

##### Instructions

**100 XP**

Adapt the code in the for loop such that the first iteration prints out "US: 809", the second iteration "AUS: 731", and so on. The output should be in the form "country: cars\_per\_cap". Make sure to print out this exact string, with the correct spacing.

Script.py  
1 # Import cars data

2 import pandas as pd

3 cars = pd.read\_csv('cars.csv', index\_col = 0)

4

5 # Adapt for loop

6 for lab, row in cars.iterrows() :

7 print(lab + ": " + str(row['cars\_per\_cap']))

IPython Shell  
In [1]: # Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Adapt for loop

for lab, row in cars.iterrows() :

print(lab + ": " + str(row['cars\_per\_cap']))

US: 809

AUS: 731

JAP: 588

IN: 18

RU: 200

MOR: 70

EG: 45

In [2]:

**Add column (1)**

In the video, Filip showed you how to add the length of the country names of the brics DataFrame in a new column:

for lab, row in brics.iterrows() :

brics.loc[lab, "name\_length"] = len(row["country"])

You can do similar things on the cars DataFrame.

**Instructions**

**100 XP**

* Use a for loop to add a new column, named COUNTRY, that contains a uppercase version of the country names in the "country" column. You can use the string method [**upper()**](https://docs.python.org/2/library/stdtypes.html#str.upper)for this.
* To see if your code worked, print out cars. Don't indent this code, so that it's not part of the for loop.

Script.py  
01 # Import cars data

02 import pandas as pd

03 cars = pd.read\_csv('cars.csv', index\_col = 0)

04

05 # Code for loop that adds COUNTRY column

06 for lab, row in cars.iterrows() :

07 cars.loc[lab, 'COUNTRY'] = row['country'].upper()

08

09 # Print cars

10 print(cars)

IPython Shell  
In [1]: # Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Code for loop that adds COUNTRY column

for lab, row in cars.iterrows() :

cars.loc[lab, 'COUNTRY'] = row['country'].upper()

# Print cars

print(cars)

cars\_per\_cap country drives\_right COUNTRY

US 809 United States True UNITED STATES

AUS 731 Australia False AUSTRALIA

JAP 588 Japan False JAPAN

IN 18 India False INDIA

RU 200 Russia True RUSSIA

MOR 70 Morocco True MOROCCO

EG 45 Egypt True EGYPT

<script.py> output:

cars\_per\_cap country drives\_right COUNTRY

US 809 United States True UNITED STATES

AUS 731 Australia False AUSTRALIA

JAP 588 Japan False JAPAN

IN 18 India False INDIA

RU 200 Russia True RUSSIA

MOR 70 Morocco True MOROCCO

EG 45 Egypt True EGYPT

In [2]:

**Add column (2)**

Using **[iterrows()](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.iterrows.html" \t "_blank)** to iterate over every observation of a Pandas DataFrame is easy to understand, but not very efficient. On every iteration, you're creating a new Pandas Series.

If you want to add a column to a DataFrame by calling a function on another column, the **[iterrows()](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.iterrows.html" \t "_blank)** method in combination with a for loop is not the preferred way to go. Instead, you'll want to use [**apply()**](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.apply.html).

Compare the **[iterrows()](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.iterrows.html" \t "_blank)** version with the [**apply()**](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.apply.html) version to get the same result in the brics DataFrame:

for lab, row in brics.iterrows() :

brics.loc[lab, "name\_length"] = len(row["country"])

brics["name\_length"] = brics["country"].apply(len)

We can do a similar thing to call the [**upper()**](https://docs.python.org/2/library/stdtypes.html#str.upper) method on every name in the country column. However, [**upper()**](https://docs.python.org/2/library/stdtypes.html#str.upper) is a **method**, so we'll need a slightly different approach:

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Replace the for loop with a one-liner that uses .apply(str.upper). The call should give the same result: a column COUNTRY should be added to cars, containing an uppercase version of the country names.
* As usual, print out cars to see the fruits of your hard labor

Script.py  
1 # Import cars data

2 import pandas as pd

3 cars = pd.read\_csv('cars.csv', index\_col = 0)

4

5 # Use .apply(str.upper)

6 cars["COUNTRY"] = cars["country"].apply(str.upper)

IPython Shell  
In [1]: # Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Use .apply(str.upper)

cars["COUNTRY"] = cars["country"].apply(str.upper)

In [2]: