

Python: An Introduction

- 1 Introduction
- 2 Creating a Vector in Python
- 3 Creating A Matrix in Python
- 4 Dataframe and Importing Data in Python
- 5 Logical Variables in Python
- 6 Loops
- 7 Functions
- 8 Saving Output to a Text File

- 1 Introduction
- 2 Creating a Vector in Python
- 3 Creating A Matrix in Python
- 4 Dataframe and Importing Data in Python
- 5 Logical Variables in Python
- 6 Loops
- 7 Functions
- 8 Saving Output to a Text File

The History of Python

- Python is a general-purpose programming language in a similar vein to other programming languages that you might have heard of such as C++, JavaScript or Microsoft's C#.
- It has been originally conceived back in the 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands.
- The language is named after a BBC TV show (Guido's favorite program) "Monty Pythons Flying Circus".
- Python reached version 1.0 in January 1994. Python 2.0 was released on October 16, 2000.
- Python 3.0, backwards-incompatible, was released on 3 December 2008.

The Goods of Python

- Its flexibility and simplicity which makes it easy to learn.
- Its use by the Data Science community where it provides a more standard programming language than some rivals (such as R).
- Its suitability as a scripting language for those working in the DevOps field where it provides a higher level of abstraction than alternative languages traditionally used.
- Its ability to run on (almost) any operating system, but particularly the big three operating systems Windows, MacOS and Linux.
- The availability of a wide range of libraries (modules) that can be used to extend the basic features of the language.
- It is free!

How to Start Python

- Download Python 3: <https://www.python.org/downloads/>
- You will need an editor to write your code. **For this course, we'll use Jupyter Notebook** (web-based).
- You can get Jupyter Notebook from Anaconda:
<https://www.anaconda.com/products/individual>.

Python Code Using Jupyter Notebook


- After downloading Python 3, you can download Anaconda (which can provide you Jupyter Notebook, and R Studio also).
- After downloading Anaconda, double click on its icon (green color), then launch Jupyter Notebook, you will have a web link with "localhost:8888/notebooks/". (Number 8888 can be other number, like 8890)
- From this, you can start a new file of code.
- Why Jupyter Notebook? Because if you get used to R, then this will help you to see the output when running the code easier than PyCharm.

- 1 Introduction
- 2 Creating a Vector in Python**
- 3 Creating A Matrix in Python
- 4 Dataframe and Importing Data in Python
- 5 Logical Variables in Python
- 6 Loops
- 7 Functions
- 8 Saving Output to a Text File

To Create a Vector in Python (1)

 jupyter Topic2_Py Last Checkpoint: 01/28/2021 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

        Run    Code 

```
In [1]: #To create a vector:
x = [1,2,3,4,5,6,7,8,9,10]
print(x)

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
In [2]: print(x*2) #this will print vector 'x' two times

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
In [3]: print(x[1])

2
```

```
In [4]: name = ['Daisy', 'Julie', 'Sophie']
print(name)

['Daisy', 'Julie', 'Sophie']
```

To Create a Vector in Python (2)

- The code line: `x = [1,2,3,4,5,6,7,8,9,10]` will create a vector “x” that has 10 strings/characters, not numbers.
- We'll need to convert this vector of strings into numeric. To do so, there are different ways, but they need some packages such as “pandas” or “numpy”.
- The code should be:

```
import numpy as np
x = [1,2,3,4,5,6,7,8,9,10]
x = np.array(x) # this will turn the vector 'x' above to a numeric vector.
print(x*2) # we'll get [ 2 4 6 8 10 12 14 16 18 20]
```

Creating a Vector in Python: using “Numpy”

- ```
import numpy as np
array = np.arange(20)
this will create a list of 20 numbers (from 0 to 19).
```
- Or, you can create a list then convert it to numeric by `np.array` like in the previous slide.

# Creating a Vector in Python: using “Pandas” (1)

- The code is

```
import pandas as pd
data = {'X': [1,2,3,4,5,6] }
df = pd.DataFrame(data, columns =['X'])
print(df)
print(df/2)
```

- The out put of `print(df/2)` is

|   | X   |
|---|-----|
| 0 | 0.5 |
| 1 | 1.0 |
| 2 | 1.5 |
| 3 | 2.0 |
| 4 | 2.5 |
| 5 | 3.0 |

# Tuple vs List (1)

```
In [24]: t = (2,3,'abs',5)
 t[0]
```

```
Out[24]: 2
```

```
In [27]: t = (2,3,'abs',5)
 t[0] = 1 # change the first element to '1'
 print(t)
```

```

TypeError Traceback (most recent call last)
<ipython-input-27-722938644ac6> in <module>
 1 t = (2,3,'abs',5)
----> 2 t[0] = 1 # change the first element to '1'
 3 print(t)
```

**TypeError:** 'tuple' object does not support item assignment

```
In [26]: t = [2,3,'abs',5]
 t[0] = 1 # change the first element to '1'
 print(t)
```

```
[1, 3, 'abs', 5]
```

## Tuple vs List (2)

- Check the figure the the previous slide, you will notice the big difference caused by the signs: `[]` and `()`.
- `t = ()` is a tuple, it is immutable while `t = []` is a list and it is mutable.
- Tuple (the one uses the round brackets) does not allow us to change its elements while a list (the one uses the square brackets) does.

- 1 Introduction
- 2 Creating a Vector in Python
- 3 Creating A Matrix in Python**
- 4 Dataframe and Importing Data in Python
- 5 Logical Variables in Python
- 6 Loops
- 7 Functions
- 8 Saving Output to a Text File

# Creating a Matrix: Using `numpy.array`

Example 1: using `np.array`

```
import numpy as np
array1 = np.array([1,2,3,4,5])
array2 = np.array([6,7,8,9,10])
matrix = np.array([array1,array2])
print(matrix)
```

```
[[1 2 3 4 5]
 [6 7 8 9 10]]
```



# Creating a Matrix: Using `numpy.asmatrix`

Example 2: using `np.asmatrix`

```
In [9]: row1 = [1,2,3]
 row2 = [4,5,6]
 row3 = [7,8,9]

 matrix2 = np.asmatrix([row1,row2,row3])
 print(matrix2)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
In [10]: print(matrix2*2)
```

```
[[2 4 6]
 [8 10 12]
 [14 16 18]]
```

```
In [11]: print(matrix2.shape) #dimension
```

```
(3, 3)
```

# Creating a Matrix: Using `numpy.asmatrix`

Example 2 (cont): Transpose and inverse of a matrix.

```
print(matrix2.T) #transpose
```

```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

```
print(matrix2.I) # Inverse
```

```
[[3.15251974e+15 -6.30503948e+15 3.15251974e+15]
 [-6.30503948e+15 1.26100790e+16 -6.30503948e+15]
 [3.15251974e+15 -6.30503948e+15 3.15251974e+15]]
```

# Creating a Matrix: Combining Rows or Columns

Example 3: using `np.vstack` and `np.column_stack`

```
print(np.vstack((matrix, array1)))
```

```
[[1 2 3 4 5]
 [6 7 8 9 10]
 [1 2 3 4 5]]
```

```
print(np.column_stack((array1, array2)))
```

```
[[1 6]
 [2 7]
 [3 8]
 [4 9]
 [5 10]]
```

# Creating a Matrix in Python: using “Pandas”

- The code is

```
import pandas as pd
data = {'X': [1,2,3,4,5,6], 'Y': [6,5,4,3,2,1]}
df = pd.DataFrame(data, columns=['X', 'Y'])
print(df**2)
```

- Output

|   | X  | Y  |
|---|----|----|
| 0 | 1  | 36 |
| 1 | 4  | 25 |
| 2 | 9  | 16 |
| 3 | 16 | 9  |
| 4 | 25 | 4  |
| 5 | 36 | 1  |

- 1 Introduction
- 2 Creating a Vector in Python
- 3 Creating A Matrix in Python
- 4 Dataframe and Importing Data in Python**
- 5 Logical Variables in Python
- 6 Loops
- 7 Functions
- 8 Saving Output to a Text File

# Dataframe in Python (1)

- The form of a dataframe in Python is similar as in R: rows for the observations and columns for the variables.
- The first observation starts with index “0”, not by “1” as in R.
- In R, once we have run the command `attach(data)` then you can assess every variable in this “data”. Unlike in R, in Python, there is no command that is equivalent to the command `attach` in R, and we cannot assess a variable in a dataframe directly.

# Dataframe in Python (2)

Example: variable 'X' in dataframe "df" is not accessible directly.

```
In [6]: data = {'X': [1,2,3,4,5,6], 'Y': [6,5,4,3,2,1]}
```

```
df = pd.DataFrame(data, columns=['X', 'Y'])
print(df**2) # this has 2 columns
print(X)
```

|   | X  | Y  |
|---|----|----|
| 0 | 1  | 36 |
| 1 | 4  | 25 |
| 2 | 9  | 16 |
| 3 | 16 | 9  |
| 4 | 25 | 4  |
| 5 | 36 | 1  |

```

NameError Traceback (most recent call last)
<ipython-input-6-f8f714ba7afe> in <module>
 3 df = pd.DataFrame(data, columns=['X', 'Y'])
 4 print(df**2) # this has 2 columns
----> 5 print(X)
 6
```

```
NameError: name 'X' is not defined
```

## Dataframe in Python (3)

Getting all variables' name in a data:

```
In [13]: data = {'X': [1,2,3,4,5,6], 'Y': [6,5,4,3,2,1]}

df = pd.DataFrame(data, columns=['X', 'Y'])
#print(df**2) # this has 2 columns
#print(X)
list(df)
```

```
Out[13]: ['X', 'Y']
```

Changing column's name:

```
#df_new = df.rename({'X': 'NewX', 'Y': 'NewY'}, axis='columns')
df_new = df.rename({'X': 'NewX', 'Y': 'NewY'}, axis=1)
print(df_new)
```

|   | NewX | NewY |
|---|------|------|
| 0 | 1    | 6    |
| 1 | 2    | 5    |
| 2 | 3    | 4    |
| 3 | 4    | 3    |
| 4 | 5    | 2    |
| 5 | 6    | 1    |



# Reading Data File (1)

## Importing a CSV file, using "Pandas"

```
In [19]: ##### IMPORTING DATA:
CSV DATA

import pandas as pd
data = pd.read_csv (r"C:\Data\Delivery_Time.csv")
print (data) # the given names of the columns are too long
```

|    | Observation | Delivery Time, y | Number of Cases, x1 | Distance, x2 (ft) |
|----|-------------|------------------|---------------------|-------------------|
| 0  | 1           | 16.68            | 7                   | 560               |
| 1  | 2           | 11.50            | 3                   | 220               |
| 2  | 3           | 12.03            | 3                   | 340               |
| 3  | 4           | 14.88            | 4                   | 80                |
| 4  | 5           | 13.75            | 6                   | 150               |
| 5  | 6           | 18.11            | 7                   | 330               |
| 6  | 7           | 8.00             | 2                   | 110               |
| 7  | 8           | 17.83            | 7                   | 210               |
| 8  | 9           | 79.24            | 30                  | 1460              |
| 9  | 10          | 21.50            | 5                   | 605               |
| 10 | 11          | 40.33            | 16                  | 688               |
| 11 | 12          | 21.00            | 10                  | 215               |
| 12 | 13          | 13.50            | 4                   | 255               |

## Reading Data File (2)

Importing a Text file, using “Pandas” and assessing a particular column:

```
#####3 ANOTHER DATA (TEXT data):

text = pd.read_csv(r"C:\Data\ex_1_name.txt", header = 0, sep = " ")
print(text)
print(text['CA2'])
```

|   | Subject | Gender | CA1 | CA2 | HW |
|---|---------|--------|-----|-----|----|
| 0 | 10      | M      | 80  | 84  | A  |
| 1 | 7       | M      | 85  | 89  | A  |
| 2 | 4       | F      | 90  | 86  | B  |
| 3 | 20      | M      | 82  | 85  | B  |
| 4 | 25      | F      | 94  | 94  | A  |
| 5 | 14      | F      | 88  | 84  | C  |

|   |    |
|---|----|
| 0 | 84 |
| 1 | 89 |
| 2 | 86 |
| 3 | 85 |
| 4 | 94 |
| 5 | 84 |

Name: CA2, dtype: int64

Other type of data (Excel, SPSS, SAS, Rdata...) also can be read by Python.  
Useful link:

<https://www.listendata.com/2017/02/import-data-in-python.html>.

## Reading Data File (3)

One of easy way to ask Python to read a data file is using function “open” which does not require any module (like “pandas”), however be careful with this “open” function:

```
In [62]: f = open("ex_1_name.txt", "r")
f = f.read()
print(f)
print(f[0:7])
```

```
Subject Gender CA1 CA2 HW
10 M 80 84 A
7 M 85 89 A
4 F 90 86 B
20 M 82 85 B
25 F 94 94 A
14 F 88 84 C
Subject
```

```
In [61]: f = open("ex_1_name.txt", "r")
f = f.read()
print(f)
print(f[6:19])
```

```
Subject Gender CA1 CA2 HW
10 M 80 84 A
7 M 85 89 A
4 F 90 86 B
20 M 82 85 B
25 F 94 94 A
14 F 88 84 C
t Gender CA1
```

# Reading Data File (4)

```
In [63]: f = open("ex_1_name.txt", "r")
f = f.read()
print(f)
print(f['Subject'])
```

```
Subject Gender CA1 CA2 HW
10 M 80 84 A
7 M 85 89 A
4 F 90 86 B
20 M 82 85 B
25 F 94 94 A
14 F 88 84 C
```

-----  
**TypeError**

Traceback (most recent call last)

```
<ipython-input-63-0a81e6c8a89a> in <module>
 2 f = f.read()
 3 print(f)
----> 4 print(f['Subject'])
```

**TypeError:** string indices must be integers

## Reading Data File (5)

Adding a new column/variable to an existing data:

```
text = pd.read_csv(r"C:\Data\ex_1_name.txt", header = 0, sep = " ")
#print(text)
#print(text['CA2'])
CA3 = [1,2,3,4,5,6] #creating a new vector

text['CA3'] = CA3 # adding the new vector to the existing data as a new column
print(text)
```

|   | Subject | Gender | CA1 | CA2 | HW | CA3 |
|---|---------|--------|-----|-----|----|-----|
| 0 | 10      | M      | 80  | 84  | A  | 1   |
| 1 | 7       | M      | 85  | 89  | A  | 2   |
| 2 | 4       | F      | 90  | 86  | B  | 3   |
| 3 | 20      | M      | 82  | 85  | B  | 4   |
| 4 | 25      | F      | 94  | 94  | A  | 5   |
| 5 | 14      | F      | 88  | 84  | C  | 6   |

## Reading Data File (5)

Adding a new column/variable from another dataframe to an existing dataframe (merging by column when using "axis = 1"):

```
In [74]: text = pd.read_csv(r"C:\Data\ex_1_name.txt", header = 0, sep = " ")
#print(text)
new= pd.read_csv(r"C:\Data\ex_1_IQ.txt", header = 0, sep = " ")
print(new['IQ'])
frames = [text, new['IQ']]
result = pd.concat(frames, axis=1)
print(result)
```

```
0 106
1 112
2 119
3 102
4 125
5 101
Name: IQ, dtype: int64
```

|   | Subject | Gender | CA1 | CA2 | HW | IQ  |
|---|---------|--------|-----|-----|----|-----|
| 0 | 10      | M      | 80  | 84  | A  | 106 |
| 1 | 7       | M      | 85  | 89  | A  | 112 |
| 2 | 4       | F      | 90  | 86  | B  | 119 |
| 3 | 20      | M      | 82  | 85  | B  | 102 |
| 4 | 25      | F      | 94  | 94  | A  | 125 |
| 5 | 14      | F      | 88  | 84  | C  | 101 |

- 1 Introduction
- 2 Creating a Vector in Python
- 3 Creating A Matrix in Python
- 4 Dataframe and Importing Data in Python
- 5 Logical Variables in Python**
- 6 Loops
- 7 Functions
- 8 Saving Output to a Text File

# Logical Tests in Python (1)

```
In [2]: x = 1
 y = 2
 print(x>y)
```

False

```
In [3]: print('x > y is',x>y)
```

x > y is False

```
In [6]: x = 1
 y = 2
 print(x<y)
```

True

```
In [5]: print('x < y is',x<y)
```

x < y is True



# Logical Tests in Python (2)

```
In [1]: x= 1
 y = 2
 print('x > y is',x>y)

 print('x < y is',x<y)

 print('x == y is',x==y)

 print('x != y is',x!=y)

 print('x >= y is',x>=y)

 print('x <=y is', x<=y)

x > y is False
x < y is True
x == y is False
x != y is True
x >= y is False
x <=y is True
```

- 1 Introduction
- 2 Creating a Vector in Python
- 3 Creating A Matrix in Python
- 4 Dataframe and Importing Data in Python
- 5 Logical Variables in Python
- 6 Loops**
- 7 Functions
- 8 Saving Output to a Text File

# If () else in Python (1)

Generally, it has format as:

```
if <condition>:
 <expression>

if <condition>:
 <expression>
else :
 <expression>

if <condition>:
 <expression>
elif: <condition>:
 <expression>
else :
 <expression>
```

## If () else in Python (2)

```
In [7]: x = 1
 y = 2

 if x < y:
 print('x is less than y')
 else:
 print('x is not less than y')
```

x is less than y

```
In [6]: x = 1
 y = 0

 if x < y:
 print('x is less than y')
 else:
 print('x is not less than y')
```

x is not less than y

- The statement in <condition> must produce a True or a False value.
- The indentation is very important! After the colon of the “if”, all the statements that are indented are under the “if” condition.

## If () else in Python (3)

```
In [8]: x = 1
 y = 2

 if x>0:
 print('x is positive')
 elif x>100:
 print('x is greater than 100')
```

x is positive

```
In [9]: x = 1
 y = 2

 if x<=0:
 print('x is negative')
 elif x<5:
 print('x is positive but less than 5')
```

x is positive but less than 5

```
In [10]: x = 10

 if x<=0:
 print('x is negative')
 elif x<=5:
 print('x is positive but less than 5')
 else:
 print('x is greater than 5')
```

x is greater than 5

## while Loop (1)

- Format:  
while <condition>:  
    <expression>
- The idea of while loop (and for loop also) is similar as in R.
- The while loop can be stopped even when the condition of the loop is met; or we can skip some iterations in between the loop (see next slide).
- A significant difference from the while loop in R is the **indentation** when writing the code.

```
In [11]: i = 1
while i<=5:
 print('i = ',i, ' less than 5')
 i = i+1
print('i = ',i, ' is larger than 5')
```

```
i = 1 less than 5
i = 2 less than 5
i = 3 less than 5
i = 4 less than 5
i = 5 less than 5
i = 6 is larger than 5
```

## while Loop (2)

- With the `break` statement we can stop the loop even if the condition of `while` is true

```
In [13]: i = 1
 while i<=100:
 print('i = ',i, ' less than 5')
 if i == 3:
 break
 i = i+1
```

```
i = 1 less than 5
i = 2 less than 5
i = 3 less than 5
```

## while Loop (3)

- With the `continue` statement we can stop the current iteration and continue with the next one.

```
In [21]: i = 0
s = 0
while i < 6:
 s = s + i
 i += 1
 if i == 3:
 print('skip')
 continue
 print(s)
```

```
0
1
skip
6
10
15
```



## while Loop (4)

An example of while loop and if statement (equivalent to the example in slide 43 in Topic 1: print all the squares of integers from 1 to 5)

```
In [22]: x = 0
 test = True
 while test == True:
 x = x+1
 if x<6:
 test = True
 else:
 test = False
 print(x**2, test)
```

```
1 True
4 True
9 True
16 True
25 True
36 False
```

## for Loop (1)

- Similar idea as the for loop in R.
- Similar as the while loop, we have the break to stop the loop and continue to skip some iterations in the loop.
- Form:  
for <variable> in range(<some\_num>):  
    <expression>
- The indentation really matters.

```
In [30]: s = 0
 for i in range(10): # i from 0 to 9
 s = s+i
 print(s)
```

45

## for Loop (2)

- The range(start, stop, step)
- range(10) means start = 0, step = 1 and stop = 9.

```
In [32]: S = 0
 for i in range(10): # i from 0 to 9
 S = S+i
 print('The sum of the first ',i, 'numbers =',S)
```

```
The sum of the first 0 numbers = 0
The sum of the first 1 numbers = 1
The sum of the first 2 numbers = 3
The sum of the first 3 numbers = 6
The sum of the first 4 numbers = 10
The sum of the first 5 numbers = 15
The sum of the first 6 numbers = 21
The sum of the first 7 numbers = 28
The sum of the first 8 numbers = 36
The sum of the first 9 numbers = 45
```

```
In [88]: S = 0
 for i in range(1,10,2):
 S = S+i
 #print(S)
 print('sum of all the odd numbers less than 10 is ', S)
```

```
sum of all the odd numbers less than 10 is 25
```

## for Loop (3)

```
In [38]: S = 0
 for i in range(11):
 S = S + i
 if i == 6:
 break
 print(S)

 print('sum of first',i, 'integers is ',S)
```

0  
1  
3  
6  
10  
15  
sum of first 6 integers is 21

- 1 Introduction
- 2 Creating a Vector in Python
- 3 Creating A Matrix in Python
- 4 Dataframe and Importing Data in Python
- 5 Logical Variables in Python
- 6 Loops
- 7 Functions**
- 8 Saving Output to a Text File

# Built-in or Existing Functions in Python

```
import numpy as np
from statistics import mean
from statistics import variance
x = [20,3,4,5,8,12,1]
y = [5,6,3,2,80,10,9]
```

- `max(x)`: maximum value of `x`
- `min(x)`: minimum value of `x`
- `len(x)`: number of elements in `x`
- `sum(x)`: total of all the values in `x`
- `mean(x)`: arithmetic average values in `x`
- `median(x)`: median value of `x`
- `var(x)`: sample variance of `x`, with degrees of freedom = `length(x) - 1`
- `np.corrcoef(x, y)`: correlation matrix of `x` and `y`

# User-define Functions (1)

- Similar as in R, in Python, a function has a name, parameters (0 or more), a body and returns something.
- How to write/define:  

```
def name(parameters):
 <function body>
```
- At the end of the function body, some command to ask for evaluation and return should be included.

# User-define Functions: Examples (1)

- A function to find sum of all the values in a list:

```
def f_sum(x):
 return(sum(x))
x = [20,3,4,5,8,12,1]
print(f_sum(x))
```

- A function to find the standard error of the mean of a vector (SE of  $\bar{x}$ ):

```
import math
from statistics import mean
from statistics import variance
def f_se(x):
 se = math.sqrt(variance(x)/len(x))
 return(se)
x = [20,3,4,5,8,12,1]
print(f_se(x))
```



## User-define Functions: Examples (2)

```
In [77]: import math
from statistics import variance
import pandas as pd

def f_se(x):
 se = math.sqrt(variance(x)/len(x))
 return(se)

text = pd.read_csv(r"C:\Data\ex_1_name.txt", header = 0, sep = " ")
x = text['CA1']

print(f_se(x))
```

2.1252450839060106

- 1 Introduction
- 2 Creating a Vector in Python
- 3 Creating A Matrix in Python
- 4 Dataframe and Importing Data in Python
- 5 Logical Variables in Python
- 6 Loops
- 7 Functions
- 8 Saving Output to a Text File**

# Saving Output to a Text File (1)

“CA1” is stored in “x” which is the list that we want to save to a text file.

```
In [117]: text = pd.read_csv(r"C:\Data\ex_1_name.txt", header = 0, sep = " ")
print(text)
x = text['CA1'] # the list that we want to save to a text file
print(x)
```

|   | Subject | Gender | CA1 | CA2 | HW |
|---|---------|--------|-----|-----|----|
| 0 | 10      | M      | 80  | 84  | A  |
| 1 | 7       | M      | 85  | 89  | A  |
| 2 | 4       | F      | 90  | 86  | B  |
| 3 | 20      | M      | 82  | 85  | B  |
| 4 | 25      | F      | 94  | 94  | A  |
| 5 | 14      | F      | 88  | 84  | C  |

|   | CA1 |
|---|-----|
| 0 | 80  |
| 1 | 85  |
| 2 | 90  |
| 3 | 82  |
| 4 | 94  |
| 5 | 88  |

Name: CA1, dtype: int64

## Saving Output to a Text File (2)

The text file that we will save the output is named as “test.txt”.

```
In [123]: f = open('test.txt', 'w')
 for i in x:
 f.write(str(i) + '\n')
 f.close()
```

Opening the file “test.txt” as “y”.

```
In [124]: y = pd.read_csv("test.txt", header = None)
 print(y)
 y.columns = ['first CA'] #name the column
 print(y)
```

```
0
0 80
1 85
2 90
3 82
4 94
5 88
 first CA
0 80
1 85
2 90
3 82
4 94
5 88
```