

Intermediate Python

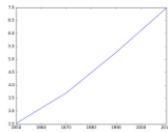
Cheatsheet & Final Project

Matplotlib

- mother of all visualization packages in Python

Matplotlib

```
import matplotlib.pyplot as plt
year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]
plt.plot(year, pop)
plt.show()
```



Scatter plot

```
import matplotlib.pyplot as plt
year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.692, 5.263, 6.972]
plt.scatter(year, pop)
plt.show()
```



The resulting scatter plot simply plots all the individual data points; Python doesn't connect the dots with a line.



- `plt.plot(x,y)` creates a line graph ^^
- Scatter plot is better because it only shows points.

Python for Data Science Cheatsheet

Python For Data Science Cheat Sheet

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
5
```

Calculations With Variables

Code	Description
<code>>>> x+2</code>	Sum of two variables
<code>>>> x-2</code>	Subtraction of two variables
<code>>>> x*2</code>	Multiplication of two variables
<code>>>> x**2</code>	Exponentiation of a variable
<code>>>> x%2</code>	Remainder of a variable
<code>>>> x/float(2)</code>	Division of a variable

Types and Type Conversion

Function	Example	Description
<code>str()</code>	<code>'5', '3.45', 'True'</code>	Variables to strings
<code>int()</code>	<code>5, 3, 1</code>	Variables to integers
<code>float()</code>	<code>5.0, 1.0</code>	Variables to floats
<code>bool()</code>	<code>True, True, True</code>	Variables to booleans

Asking For Help

```
>>> help(str)
```

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

String Methods

Method	Description
<code>my_string.upper()</code>	String to uppercase
<code>my_string.lower()</code>	String to lowercase
<code>my_string.count('w')</code>	Count String elements
<code>my_string.replace('e', 'l')</code>	Replace String elements
<code>my_string.strip()</code>	Strip whitespaces

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Code	Description
<code>my_list[1]</code>	Select item at index 1
<code>my_list[-3]</code>	Select 3rd last item
<code>my_list[1:3]</code>	Select items at index 1 and 2
<code>my_list[1:]</code>	Select items after index 0
<code>my_list[:3]</code>	Select items before index 3
<code>my_list[:]</code>	Copy my_list
<code>my_list2[1][0]</code>	my_list[list][itemOfList]
<code>my_list2[1][:2]</code>	

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

Method	Description
<code>my_list.index(a)</code>	Get the index of an item
<code>my_list.count(a)</code>	Count an item
<code>my_list.append('l')</code>	Append an item at a time
<code>my_list.remove('l')</code>	Remove an item
<code>del my_list[0:1]</code>	Remove an item
<code>my_list.reverse()</code>	Reverse the list
<code>my_list.extend('l')</code>	Append an item
<code>my_list.pop(-1)</code>	Remove an item
<code>my_list.insert(0, 'l')</code>	Insert an item
<code>my_list.sort()</code>	Sort the list

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
>>> from math import pi
```

Install Python

ANACONDA
Leading open data science platform powered by Python

SPYDER
Free IDE that is included with Anaconda

JUPYTER
Create and share documents with live code, visualizations, text, ...

NumPy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Code	Description
<code>my_array[1]</code>	Select item at index 1
<code>my_array[0:2]</code>	Select items at index 0 and 1
<code>my_2darray[rows, columns]</code>	

NumPy Array Operations

```
>>> my_array > 3
array([False,  True,  True,  True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

NumPy Array Functions

Function	Description
<code>my_array.shape</code>	Get the dimensions of the array
<code>np.append(other_array)</code>	Append items to an array
<code>np.insert(my_array, 1, 5)</code>	Insert items in an array
<code>np.delete(my_array, [1])</code>	Delete items in an array
<code>np.mean(my_array)</code>	Mean of the array
<code>np.median(my_array)</code>	Median of the array
<code>my_array.corrcoef()</code>	Correlation coefficient
<code>np.std(my_array)</code>	Standard deviation

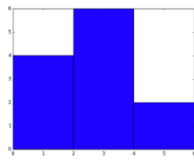
<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/f70dceca-c7ce-482a-85f8-8f6b0363aaba/e30fbcd9-f595-4a9f-803d-05ca5bf84612.pdf>

Histograms

- If there is no bin specified, the 10 bins will be the default.

Matplotlib example

```
values = [0,0.6,1.4,1.6,2.2,2.5,2.6,3.2,3.5,3.9,4.2,6]
plt.hist(values, bins=3)
plt.show()
```



Histograms are really useful to give a bigger picture.



Customization

Add historical data

population.py

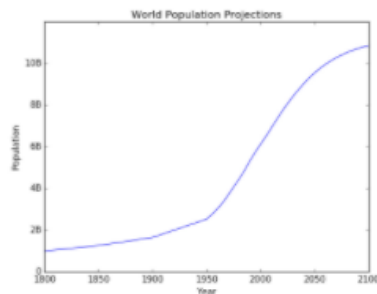
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

# Add more data
year = [1800, 1850, 1900] + year
pop = [1.0, 1.262, 1.650] + pop

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0, 2, 4, 6, 8, 10],
           ['0', '28', '48', '68', '88', '108'])

plt.show()
```



three data points are added to the graph, giving a more complete picture.



Dictionaries

Dictionary Manipulation (1)

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 `europa` you can use something like this:

```
europa['iceland'] = 'reykjavik'
```

Instructions

100 XP

- Add the key `'italy'` with the value `'rome'` to `europa`.
- To assert that `'italy'` is now a key in `europa`, print out `'italy' in europa`.
- Add another key-value pair to `europa`: `'poland'` is the key, `'warsaw'` is the corresponding value.
- Print out `europa`.

Take Hint (-30 XP)

Incorrect Submission

```
1 # Definition of dictionary
2 europa = {'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo' }
3
4 # Add italy to europa
5 europa['italy'] = 'rome'
6 print('italy' in europa)
7
8 europa['poland'] = 'warsaw'
9 print(europa)
10
11 # Print out italy in europa
12
13
14 # Add poland to europa
15
16
17
```



Run Code

Submit Answer

IPython Shell

Slides

```
True
{'spain': 'madrid', 'norway': 'oslo', 'france': 'paris', 'italy': 'rome', 'poland': 'warsaw'}
```

<script.py> output:

```
True
{'spain': 'madrid', 'norway': 'oslo', 'france': 'paris', 'italy': 'rome', 'poland': 'warsaw'}
```

Dictionary Manipulation (2)

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

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

Instructions

100 XP

- 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 `europa`.
- Print out `europa` to see if your cleaning work paid off.

Take Hint (-30 XP)

```
1 # Definition of dictionary
2 europa = {'spain':'madrid', 'france':'paris', 'germany':'bonn',
3           'norway':'oslo', 'italy':'rome', 'poland':'warsaw',
4           'australia':'vienna' }
5
6 # Update capital of germany
7
8 europa["germany"] = "berlin"
9
10 del(europa["australia"])
11 print(europa)
12
13 # Remove australia
14
15 # Print europa
16
```



Run Code

Submit Answer

IPython Shell

Slides

```
# Print europa
```

```
{'spain': 'madrid', 'norway': 'oslo', 'france': 'paris', 'italy': 'rome', 'poland': 'warsaw'}
```

<script.py> output:

```
{'spain': 'madrid', 'norway': 'oslo', 'france': 'paris', 'italy': 'rome', 'poland': 'warsaw'}
```

Dictionaryception

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 `europa` - 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 `europa`, for example, you need:

```
europa['spain']['population']
```

Instructions

100 XP

- 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 `europa`; the key is `'italy'` and the value is `data`, the dictionary you just built.

Take Hint (-30 XP)

```
1 # Dictionary of dictionaries
2 europa = { 'spain': { 'capital':'madrid', 'population':46.77 },
3           'france': { 'capital':'paris', 'population':66.03 },
4           'germany': { 'capital':'berlin', 'population':80.62 },
5           'norway': { 'capital':'oslo', 'population':5.084 } }
6
7 # Print out the capital of France
8 print(europa["france"]["capital"])
9
10 data = {'capital': 'rome', 'population': 59.83}
11
12 europa['italy'] = data
13 print(europa)
14 # Create sub-dictionary data
15 # Add data to europa under key 'italy'
16 # Print europa
17
```



Run Code

Submit Answer

IPython Shell

Slides

```
# Add data to europa under key 'italy'
```

```
# Print europa
```

```
paris
{'spain': {'population': 46.77, 'capital': 'madrid'}, 'norway': {'population': 5.084, 'capital': 'oslo'}}
```

Pandas

- part of the numpy package

Creating your own dataframe from Dictionary

DataFrame from Dictionary

```
dict = {
    "country":["Brazil", "Russia", "India", "China", "South Africa"],
    "capital":["Brasilia", "Moscow", "New Delhi", "Beijing", "Pretoria"],
    "area":[8.516, 17.10, 3.286, 9.597, 1.221]
    "population":[200.4, 143.5, 1252, 1357, 52.98] }
```

- keys (column labels)
- values (data, column by column)

```
import pandas as pd
brics = pd.DataFrame(dict)
```

DataFrame from Dictionary (2)

```
brics
```

	area	capital	country	population
0	8.516	Brasilia	Brazil	200.40
1	17.100	Moscow	Russia	143.50
2	3.286	New Delhi	India	1252.00
3	9.597	Beijing	China	1357.00
4	1.221	Pretoria	South Africa	52.98

```
brics.index = ["BR", "RU", "IN", "CH", "SA"]
```

```
brics
```

	area	capital	country	population
BR	8.516	Brasilia	Brazil	200.40
RU	17.100	Moscow	Russia	143.50
IN	3.286	New Delhi	India	1252.00
CH	9.597	Beijing	China	1357.00
SA	1.221	Pretoria	South Africa	52.98

Creating DataFrame from CSV file

DataFrame from CSV file

- brics.csv

```
,country,capital,area,population
BR,Brazil,Brasilia,8.516,200.4
RU,Russia,Moscow,17.10,143.5
IN,India,New Delhi,3.286,1252
CH,China,Beijing,9.597,1357
SA,South Africa,Pretoria,1.221,52.98
```

```
brics = pd.read_csv("path/to/brics.csv")
brics
```

Unnamed: 0	country	capital	area	population
0	BR	Brazil	8.516	200.40
1	RU	Russia	17.100	143.50
2	IN	India	3.286	1252.00
3	CH	China	9.597	1357.00
4	SA	South Africa	1.221	52.98

DataFrame from CSV file

```
brics = pd.read_csv("path/to/brics.csv", index_col = 0)
brics
```

	country	population	area	capital
BR	Brazil	200	8515767	Brasilia
RU	Russia	144	17098242	Moscow
IN	India	1252	3287590	New Delhi
CH	China	1357	9596961	Beijing
SA	South Africa	55	1221837	Pretoria

loc() vs iloc()

- loc is for words, iloc is for integers

- Square brackets
 - Column access `brics["country", "capital"]`
 - Row access: only through slicing `brics[1:4]`
- loc (label-based)
 - Row access `brics.loc["RU", "IN", "CH"]`
 - Column access `brics.loc[:, ["country", "capital"]]`
- Row & Column access
 - `brics.loc[["RU", "IN", "CH"], ["country", "capital"]]`

Row & Column loc

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```
brics.loc[["RU", "IN", "CH"], ["country", "capital"]]
```

	country	capital
RU	Russia	Moscow
IN	India	New Delhi
CH	China	Beijing

Row Access iloc

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```
brics.iloc[[1,2,3]]
```

	country	capital	area	population
RU	Russia	Moscow	17.100	143.5
IN	India	New Delhi	3.286	1252.0
CH	China	Beijing	9.597	1357.0

Array Equivalents of and, or, not

NumPy

- `logical_and()`
- `logical_or()`
- `logical_not()`

```
np.logical_and(bmi > 21, bmi < 22)
```

```
array([True, False, True, False, True], dtype=bool)
```

```
bmi[np.logical_and(bmi > 21, bmi < 22)]
```

```
array([21.852, 21.75, 21.441])
```

Also, AND, OR and NOT operators are spelled out on Python

if, elif, else

```

if condition :
    expression
elif condition :
    expression
else :
    expression

```

control.py

```

z = 6
if z % 2 == 0 :
    print("z is divisible by 2")    # True
elif z % 3 == 0 :
    print("z is divisible by 3")    # Never reached
else :
    print("z is neither divisible by 2 nor by 3")

```

Filtering Pandas from DataFrames

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```

is_huge = brics["area"] > 8
brics[is_huge]

```

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.4
RU	Russia	Moscow	17.100	143.5
CH	China	Beijing	9.597	1357.0

```

brics[brics["area"] > 8]

```

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.4
RU	Russia	Moscow	17.100	143.5
CH	China	Beijing	9.597	1357.0

Boolean operators

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```

import numpy as np
np.logical_and(brics["area"] > 8, brics["area"] < 10)

```

```

BR    True
RU    False
IN    False
CH    True
SA    False
Name: area, dtype: bool

```

```

brics[np.logical_and(brics["area"] > 8, brics["area"] < 10)]

```

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.4
CH	China	Beijing	9.597	1357.0

Remember about `np.logical_and()`, `np.logical_or()` and `np.logical_not()`, the Numpy variants of the `and`, `or` and `not` operators? You can also use them on Pandas Series to do more advanced filtering operations.

Take this example that selects the observations that have a `cars_per_cap` between 10 and 80. Try out these lines of code step by step to see what's happening.

```
cpc = cars['cars_per_cap']
between = np.logical_and(cpc > 10, cpc < 80)
medium = cars[between]
```

Instructions

100 XP

- Use the code sample provided to create a DataFrame `medium`, that includes all the observations of `cars` that have a `cars_per_cap` between 100 and 500.
- Print out `medium`.

```
cars = pd.read_csv('cars.csv', index_col = 0)
4
5 # Import numpy, you'll need this
6 import numpy as np
7
8 # Create medium: observations with cars_per_cap between 100 and 500
9 medium = cars[np.logical_and(cars["cars_per_cap"] > 100, cars["cars_per_cap"] < 500)]
10 print(medium)
11
12
13 # Print medium
14
```

Python Shell Slides

Print medium

Run Code Submit

While loop

- repeating action until condition is met

While

```
while condition :
    expression
```

while_loop.py

```
error = 50.0
# 0.78125
while error > 1: # False
    error = error / 4
    print(error)
```

```
12.5
3.125
0.78125
```

For loops


```
for var in seq :
    expression
```

family.py

```
fam = [1.73, 1.68, 1.71, 1.89]
for index, height in enumerate(fam) :
    print("index " + str(index) + ": " + str(height))
```

```
index 0: 1.73
index 1: 1.68
index 2: 1.71
index 3: 1.89
```

```
for c in "family" :
    print(c.capitalize())
```

```
F
A
M
I
L
Y
```

Loop over list of lists

Remember the `house` variable from the Intro to Python course? Have a look at its definition in the script. 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.

```
1 # house list of lists
2 house = [["hallway", 11.25],
3          ["kitchen", 18.0],
4          ["living room", 20.0],
5          ["bedroom", 10.75],
6          ["bathroom", 9.50]]
7
8 # Build a for loop from scratch
9
10 for key, value in house:
11     print("the " + str(key) + " is " + str(value) + " sqm")
```

Loop Data Structures

- Dictionary
 - `for key, val in my_dict.items() :`
- Numpy array
 - `for val in np.nditer(my_array) :`

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) :
```

Dictionary

```
for var in seq :
```

dictloop.py

```
world = { "afghanistan":30.55,
          "albania":2.77,
          "algeria":39.21 }
for k, v in world.items() :
    print(k + " -- " + str(v))
```

```
algeria -- 39.21
afghanistan -- 30.55
albania -- 2.77
```

2D Numpy Arrays

nploop.py

```
import numpy as np
np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])
np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])
meas = np.array([np_height, np_weight])
for val in np.nditer(meas) :
    print(val)
```

```
1.73
1.68
1.71
1.89
1.79
65.4
...

```

...

iterrows

dfloop.py

```
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
for lab, row in brics.iterrows():
    print(lab)
    print(row)
```

```
BR
country      Brazil
capital      Brasilia
area         8.516
population   200.4
Name: BR, dtype: object
...
RU
country      Russia
capital      Moscow
area         17.1
population   143.5
Name: RU, dtype: object
IN ...
```

Selective print

dfloop.py

```
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
for lab, row in brics.iterrows():
    print(lab + ": " + row["capital"])
```

```
BR: Brasilia
RU: Moscow
IN: New Delhi
CH: Beijing
SA: Pretoria
```

Adding columns

Add column

dfloop.py

```
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
for lab, row in brics.iterrows():
    # - Creating Series on every iteration
    brics.loc[lab, "name_length"] = len(row["country"])
print(brics)
```

	country	capital	area	population	name_length
BR	Brazil	Brasilia	8.516	200.40	6
RU	Russia	Moscow	17.100	143.50	6
IN	India	New Delhi	3.286	1252.00	5
CH	China	Beijing	9.597	1357.00	5
SA	South Africa	Pretoria	1.221	52.98	12

apply

dfloop.py

```
import pandas as pd
brics = pd.read_csv("brics.csv", index_col = 0)
brics["name_length"] = brics["country"].apply(len)
print(brics)
```

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

- this is better compared the one on the left

Another example using .apply(len)

Compare the `iterrows()` version with the `apply()` 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 `apply()` method on every name in the `country` column. However, `apply()` is a method, so we'll need a slightly different approach:

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 `brics`, containing an uppercase version of the country names.
- As usual, print out `brics` to see the fruits of your hard labor

```
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 for lab, row in cars.iterrows():
7     cars['COUNTRY'] = cars['country'].apply(str.upper)
```

Python Shell

Output

Etc:

A typical way to solve problems like this is by using `max()`. If you pass `max()` two arguments, the biggest one gets returned. For example, to make sure that a variable `x` never goes below `10` when you decrease it, you can use:

```
x = max(10, x - 1)
```