Dictionaries, Part 1

INTERMEDIATE PYTHON

Hugo Bowne-Anderson
Data Scientist at DataCamp

datacamp

List

```
pop = [30.55, 2.77, 39.21]
countries = ["afghanistan", "albania", "algeria"]
ind_alb = countries.index("albania")
ind_alb
```

```
1
```

```
pop[ind_alb]
```

```
2.77
```

- Not convenient
- Not intuitive

Dictionary

```
pop = [30.55, 2.77, 39.21]
countries = ["afghanistan", "albania", "algeria"]

...

        {                                                    }
```

Dictionary

```
pop = [30.55, 2.77, 39.21]
countries = ["afghanistan", "albania", "algeria"]

...

        {"afghanistan":30.55,                                    }
```

Dictionary

```
pop = [30.55, 2.77, 39.21]
countries = ["afghanistan", "albania", "algeria"]

...

world = {"afghanistan":30.55, "albania":2.77, "algeria":39.21}
world["albania"]
```

```
2.77
```

Let's practice!

INTERMEDIATE PYTHON

Dictionaries, Part 2

INTERMEDIATE PYTHON

Hugo Bowne-Anderson
Data Scientist at DataCamp

datacamp

Recap

```
world = {"afghanistan":30.55, "albania":2.77, "algeria":39.21}
world["albania"]
```

```
2.77
```

```
world = {"afghanistan":30.55, "albania":2.77,
         "algeria":39.21, "albania":2.81}
world
```

```
{'afghanistan': 30.55, 'albania': 2.81, 'algeria': 39.21}
```

Recap

- Keys have to be "immutable" objects

```
{0:"hello", True:"dear", "two":"world"}
```

```
{0: 'hello', True: 'dear', 'two': 'world'}
```

```
{["just", "to", "test"]: "value"}
```

```
TypeError: unhashable type: 'list'
```

Principality of Sealand

[1] Source: Wikipedia

Dictionary

```
world["sealand"] = 0.000027
world
```

```
{'afghanistan': 30.55, 'albania': 2.81,
        'algeria': 39.21, 'sealand': 2.7e-05}
```

```
"sealand" in world
```

```
True
```

Dictionary

```
world["sealand"] = 0.000028
world
```

```
{'afghanistan': 30.55, 'albania': 2.81,
        'algeria': 39.21, 'sealand': 2.8e-05}
```

```
del(world["sealand"])
world
```

```
{'afghanistan': 30.55, 'albania': 2.81, 'algeria': 39.21}
```

List vs Dictionary

List vs Dictionary

List vs Dictionary

List	Dictionary
Select, update and remove: [ ]	Select, update and remove: [ ]

List vs Dictionary

List	Dictionary
Select, update and remove: [ ]	Select, update and remove: [ ]

List vs Dictionary

List	Dictionary
Select, update and remove: [ ]	Select, update and remove: [ ]
Indexed by range of numbers	

List vs Dictionary

List	Dictionary
Select, update and remove: [ ]	Select, update and remove: [ ]
Indexed by range of numbers	Indexed by unique keys

List vs Dictionary

List	Dictionary
Select, update and remove: [ ]	Select, update and remove: [ ]
Indexed by range of numbers	Indexed by unique keys
Collection of values order matters select entire subsets	

List vs Dictionary

List	Dictionary
Select, update and remove: [ ]	Select, update and remove: [ ]
Indexed by range of numbers	Indexed by unique keys
Collection of values order matters select entire subsets	Lookup table with unique keys

Let's practice!

INTERMEDIATE PYTHON

Pandas, Part 1

INTERMEDIATE PYTHON

Hugo Bowne-Anderson
Data Scientist at DataCamp

datacamp

Tabular dataset examples

temperature	measured_at	location
76	2016-01-01 14:00:01	valve
86	2016-01-01 14:00:01	compressor
72	2016-01-01 15:00:01	valve
88	2016-01-01 15:00:01	compressor
68	2016-01-01 16:00:01	valve
78	2016-01-01 16:00:01	compressor

Tabular dataset examples

temperature	measured_at	location
76	2016-01-01 14:00:01	valve
86	2016-01-01 14:00:01	compressor
72	2016-01-01 15:00:01	valve
88	2016-01-01 15:00:01	compressor
68	2016-01-01 16:00:01	valve
78	2016-01-01 16:00:01	compressor

row = observations
column = variable

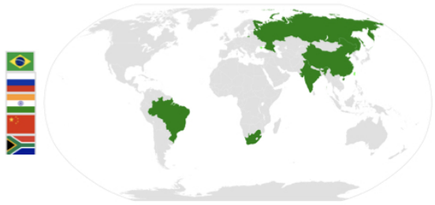
2. Tabular dataset examples
Suppose you're working in a chemical plant and have a ton of temperature measurements to analyze.

Tabular dataset examples

temperature	measured_at	location
76	2016-01-01 14:00:01	valve
86	2016-01-01 14:00:01	compressor
72	2016-01-01 15:00:01	valve
88	2016-01-01 15:00:01	compressor
68	2016-01-01 16:00:01	valve
78	2016-01-01 16:00:01	compressor

row = observations
column = variable

country	capital	area	population
Brazil	Brasilia	8.516	200.4
Russia	Moscow	17.10	143.5
India	New Delhi	3.286	1252
China	Beijing	9.597	1357
South	Pretoria	1.221	52.98



Another example: you have collected information on the so-called BRICS countries, Brazil, Russia, India, China and South Africa. You can again build a table with this data,

Datasets in Python

- 2D Numpy array?
  - One data type

Datasets in Python

country	capital	area	population
Brazil	Brasilia	8.516	200.4
Russia	Moscow	17.10	143.5
India	New Delhi	3.286	1252
China	Beijing	9.597	1357
South	Pretoria	1.221	52.98
		float	float

5. Datasets in Python
To start working on this data in Python, you'll need some kind of rectangular data structure.

7. Datasets in Python
the country and capital are strings, for example. Your datasets will typically comprise different data types, so we need a tool that's better suited for the job. To easily and efficiently handle this data, there's the Pandas package.

Datasets in Python

country	capital	area	population
Brazil	Brasilia	8.516	200.4
Russia	Moscow	17.10	143.5
India	New Delhi	3.286	1252
China	Beijing	9.597	1357
South	Pretoria	1.221	52.98
str	str	float	float

- pandas!
  - High level data manipulation tool
  - Wes McKinney
  - Built on Numpy
  - DataFrame

DataFrame

```
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
```

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
```

Pandas assigned some automatic row labels, 0 up to 4. To specify them manually, you can set the index attribute of brics to a list with the correct labels.

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
```

- CSV = comma-separated values

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   Brasilia   8.516      200.40
1         RU        Russia     Moscow  17.100      143.50
2         IN         India  New Delhi   3.286     1252.00
3         CH         China    Beijing   9.597     1357.00
4         SA  South Africa   Pretoria   1.221       52.98
```

12. DataFrame from CSV file
If you now print brics, there's still something wrong. The row labels are seen as a column in their own right. To solve this, You do this by setting the index_col argument, like this.

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   1221037   Pretoria
```

Let's practice!

INTERMEDIATE PYTHON

Pandas, Part 2

INTERMEDIATE PYTHON

Hugo Bowne-Anderson
Data Scientist at DataCamp

brics

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

Index and select data

- Square brackets
- Advanced methods
  - loc
  - iloc

3. Index and select data
There are numerous ways in which you can index and select data from DataFrames, so we'll take this step by step. First, I'm going to talk about how to use square brackets; next, I'm going to tell you about advanced data access methods, loc and iloc, that make Pandas extra powerful.

Column Access [ ]

```
         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["country"]
```

```
BR          Brazil
RU          Russia
IN           India
CH           China
SA    South Africa
Name: country, dtype: object
```

4. Column Access [ ]
Suppose that you only want to select the country column from brics. How to do this with square brackets? Well, you type brics, and then the column label inside square brackets. Python prints out the entire column, together with the row labels. But there's something strange here. The last line says Name: country, dtype: object. We're clearly not dealing with a regular DataFrame here. Let's find out about the type of the object that gets returned, with the type function,

Column Access [ ]

```
         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
```

```
type(brics["country"])
```

```
pandas.core.series.Series
```

- 1D labelled array

5. Column Access [ ]
with the type function, as follows. In a simplified sense, you can think of the Series as a 1-dimensional array that can be labeled, just like the DataFrame. Otherwise put, if you paste together a bunch of Series, you can create a DataFrame.

Column Access [ ]

```
         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[["country"]]
```

```
         country
BR        Brazil
RU        Russia
IN         India
CH         China
SA  South Africa
```

6. Column Access [ ]
If you want to select the country column but keep the data in a DataFrame, you'll need double square brackets, like this. If you check out the type of this fellow,

Column Access [ ]

```
         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
```

```
type(brics[["country"]])
```

```
pandas.core.frame.DataFrame
```

8. Column Access [ ]
You can perfectly extend this call to select two columns, country and capital, for example. If you look at it from a different angle, you're actually putting a list with column labels inside another set of square brackets, and end up with a 'sub data frame', containing only the country and capital columns. You can also use the same square brackets to select rows from a DataFrame.

Column Access [ ]

```
         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[["country", "capital"]]
```

```
         country    capital
BR        Brazil   Brasilia
RU        Russia     Moscow
IN         India  New Delhi
CH         China    Beijing
SA  South Africa   Pretoria
```

Row Access [ ]

```
         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
```

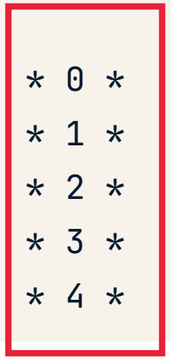
```
brics[1:4]
```

```
   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
```

9. Row Access [ ]
The way to do it is by specifying a slice. To get the second, third and fourth rows of brics, we use the slice 1 colon 4. Remember that the end of the slice is exclusive and that the index starts at zero.

Row Access [ ]

```
         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[1:4]
```

```
   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
```

Discussion [ ]

- Square brackets: limited functionality
- Ideally
  - 2D Numpy arrays
  - `my_array[rows, columns]`
- pandas
  - `loc` (label-based)
  - `iloc` (integer position-based)

Row Access 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"]
```

```
country       Russia
capital       Moscow
area            17.1
population     143.5
Name: RU, dtype: object
```

- Row as pandas Series

Row Access 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"]]
```

```
   country capital  area  population
RU  Russia  Moscow  17.1       143.5
```

- DataFrame

Row Access 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    area  population
RU  Russia     Moscow  17.100       143.5
IN   India  New Delhi   3.286      1252.0
CH   China    Beijing   9.597      1357.0
```

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 & 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[:, ["country", "capital"]]
```

```
         country    capital
BR        Brazil   Brasilia
RU        Russia     Moscow
IN         India  New Delhi
CH         China    Beijing
SA  South Africa   Pretoria
```

Recap

- 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"]
]
```

17. Recap
So, let's take a step back: simple square brackets work fine if you want to get columns; to get rows, you can use slicing. The loc function is more versatile: you can select rows, columns, but also rows and columns at the same time. When you use loc, subsetting becomes remarkable similar to how you subsetted 2D Numpy arrays. The only difference is that you use row labels with loc, not the positions of the

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.loc[["RU"]]
```

```
   country capital  area  population
RU  Russia  Moscow  17.1       143.5
```

```
brics.iloc[[1]]
```

```
   country capital  area  population
RU  Russia  Moscow  17.1       143.5
```

18. Row Access iloc
In loc, you use the "RU" string in double square brackets, to get a DataFrame, like here. In iloc, you use the index 1 instead of RU. **The results are exactly the same.**

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.loc[["RU", "IN", "CH"]]
```

```
   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
```

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
```

Row & Column 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.loc[["RU", "IN", "CH"], ["country", "capital"]]
```

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

24. Row & Column iloc
For iloc, it's like this. loc and iloc are pretty similar, the only difference is how you refer to columns and rows.

Row & Column 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], [0, 1]]
```

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

Row & Column 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.loc[:, ["country", "capital"]]
```

```
         country    capital
BR        Brazil   Brasilia
RU        Russia     Moscow
IN         India  New Delhi
CH         China    Beijing
SA  South Africa   Pretoria
```

Row & Column 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[:, [0,1]]
```

```
         country    capital
BR        Brazil   Brasilia
RU        Russia     Moscow
IN         India  New Delhi
CH         China    Beijing
SA  South Africa   Pretoria
```

Let's practice!

INTERMEDIATE PYTHON