Data Science

Manipulation and Data Analysis with Pandas
Introduction

Ricardo Campos

Acknowledgements and Authorship

This presentation was developed by Ricardo Campos, Assistant Professor at the University of Beira Interior and researcher of LIAAD - INESC TEC. Part of the slides used in this presentation were adapted from presentations found in internet and from reference bibliography:

- https://www.bu.edu/tech/files/2017/09/Python-for-Data-Analysis.pptx by Katia Oleinik
- https://github.com/nickeubank/practicaldatascience/
- https://towardsdatascience.com/how-to-reshape-a-pandas-dataframe-98b42c428a8
- https://practicaldatascience.co.uk/data-science/how-to-import-data-into-pandas-dataframes

Acknowledgements and Authorship

Please refer to the following when using this presentation:

Campos, Ricardo. (2024). Manipulation and Data Analysis with Pandas (Introduction).

A .ppt version of this presentation can be provided upon request by sending an email to [ricardo.campos@ubi.pt]

Objetivos de Aprendizagem

Learning objectives

No final desta apresentação o aluno deverá saber distinguir os diferentes tipos de dados, dominar o conceito de series e de dataframes, bem como saber criar e importar dados a partir do Pandas.

At the end of this presentation, the student should know how to distinguish between different types of data, master the concept of series and dataframes, as well as know how to create and import data from Pandas.

Sumário

Manipulação e Análise de Dados com Pandas Dataframes

Introdução dos alunos aos conceitos fundamentais da criação e importação de dataframes

- Tipos de dados: numéricos e categóricos.
- Séries de dados em Pandas.
- Criação de DataFrames.
- Importação exportação de dados.

Class Summary

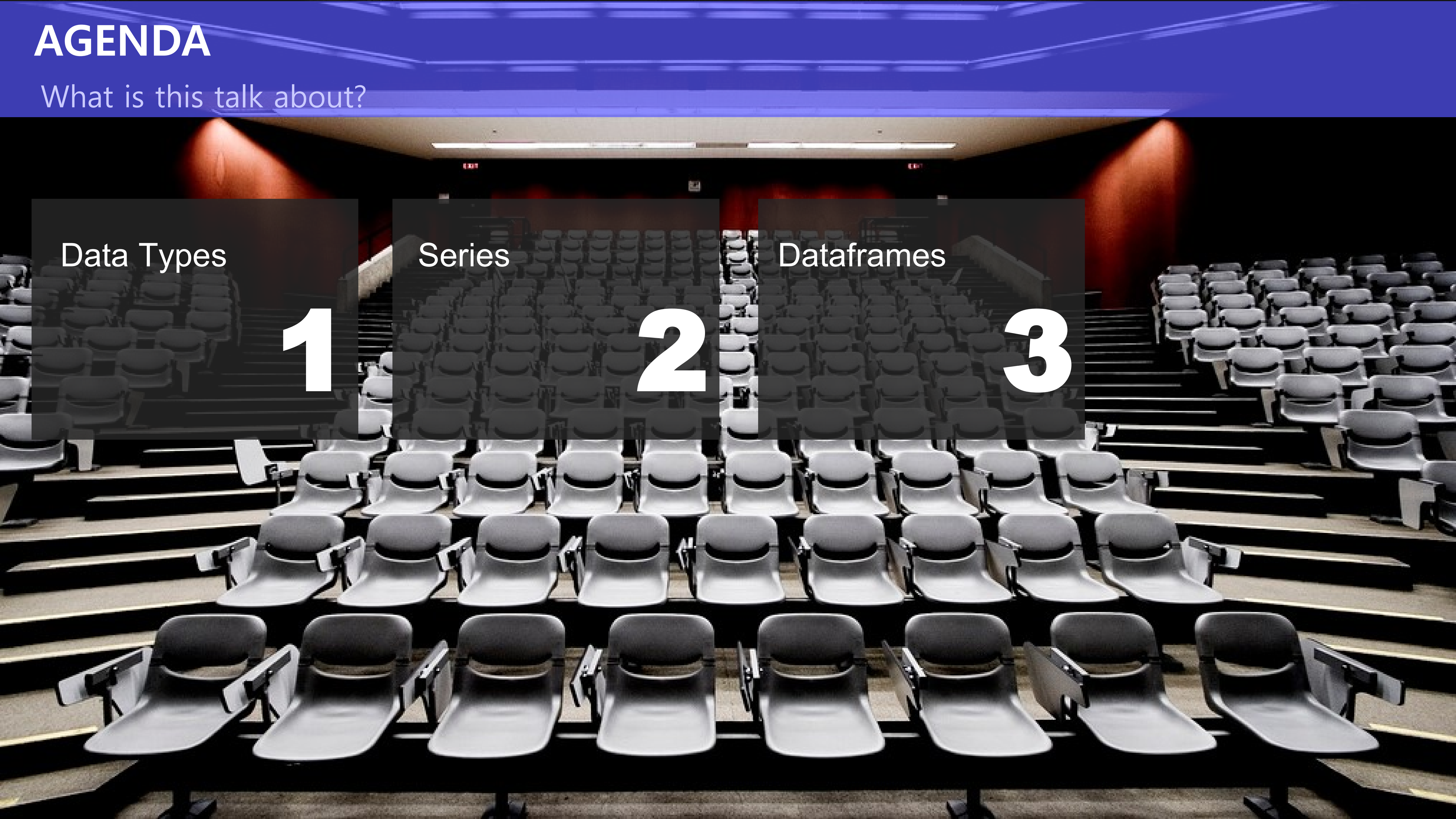
Manipulation and Data Analysis with Pandas Dataframes

Introduction to the fundamental concepts of dataframes creation and ingestion

- Data types: numerical and categorical.
- Series in pandas.
- Dataframes creation.
- Data ingestion.

AGENDA

What is this talk about?



Data Types 1

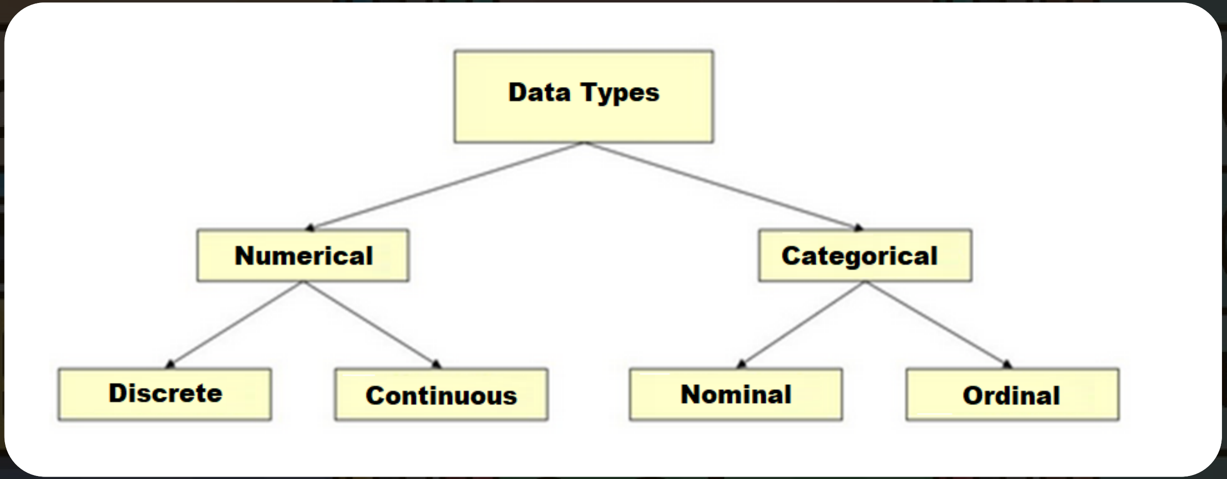
Series 2

Dataframes 3

Data Types

Overview

By data types, we don’t mean the way the data is formatted or stored. Here we are referring to the statistical properties of the data. These properties affect what sorts of statistical operations can be meaningfully applied to the data. There are two types of data: numerical and categorical data, which are further classified into four types data: continuous, discrete, nominal and ordinal,



Data Types

Numerical

Categorical

Discrete

Continuous

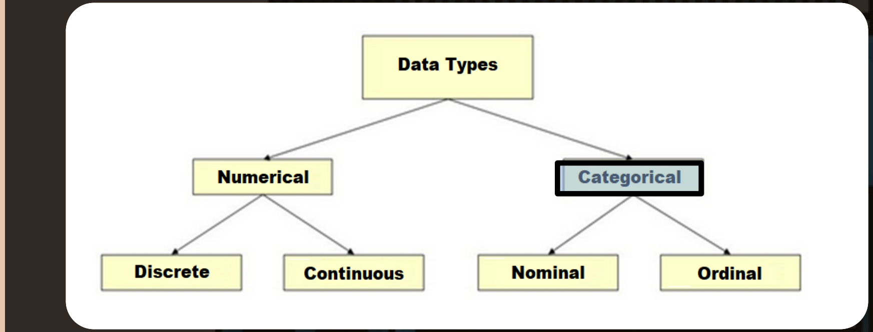
Nominal

Ordinal

Source: https://www.casadocodigo.com.br/products/livro-pandas-python

Data Types

Categorical



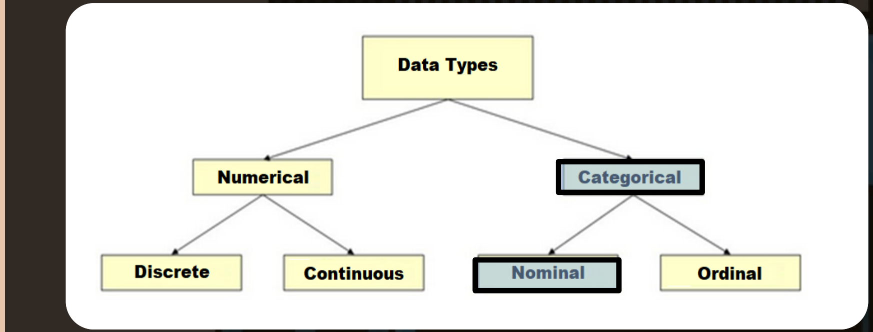
Categorical data is data that can’t be measured or counted in the form of numbers. These types of data are sorted by category, not by number.

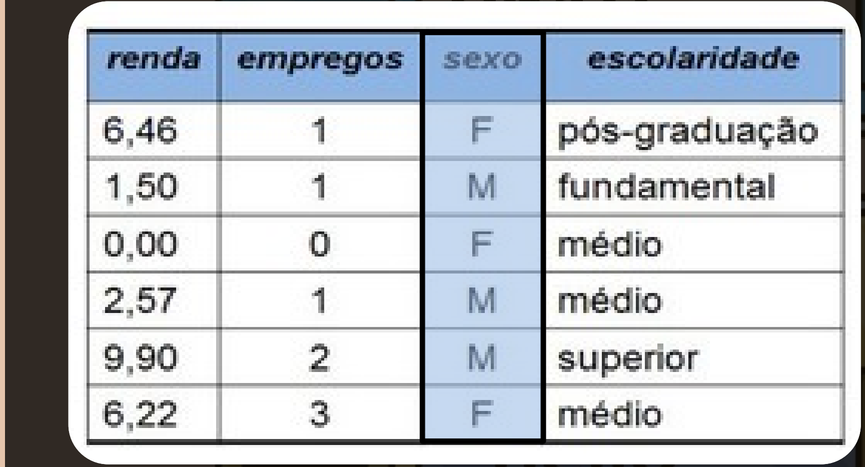
Examples of Categorical Data:

- Colors
- Grades (A, B, C, etc.)

Data Types

Categorical - Nominal





renda	empregos	sexo	escolaridade
6,46	1	F	pós-graduação
1,50	1	M	fundamental
0,00	0	F	médio
2,57	1	M	médio
9,90	2	M	superior
6,22	3	F	médio

A nominal variable is one that has two or more categories, but there is no intrinsic ordering to the categories. For example, a binary variable (such as yes/no question) is a categorical variable having two categories (yes or no) and there is no intrinsic ordering to the categories.

A purely nominal variable is one that simply allows you to assign categories, but you cannot clearly order the categories. If the variable has a clear ordering, then that variable would be an ordinal variable, as described below.

Data Types

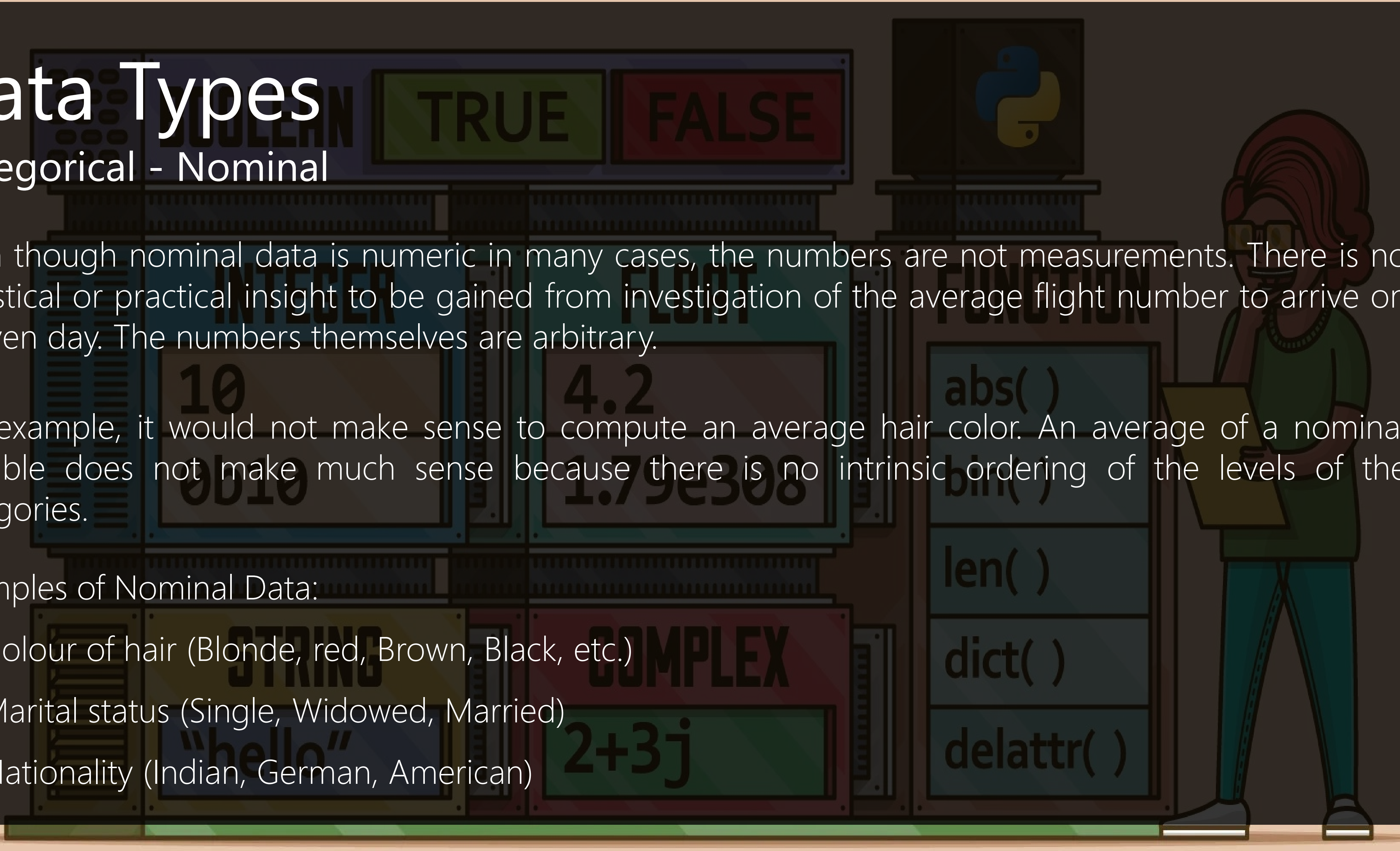
Categorical - Nominal

Even though nominal data is numeric in many cases, the numbers are not measurements. There is no statistical or practical insight to be gained from investigation of the average flight number to arrive on a given day. The numbers themselves are arbitrary.

For example, it would not make sense to compute an average hair color. An average of a nominal variable does not make much sense because there is no intrinsic ordering of the levels of the categories.

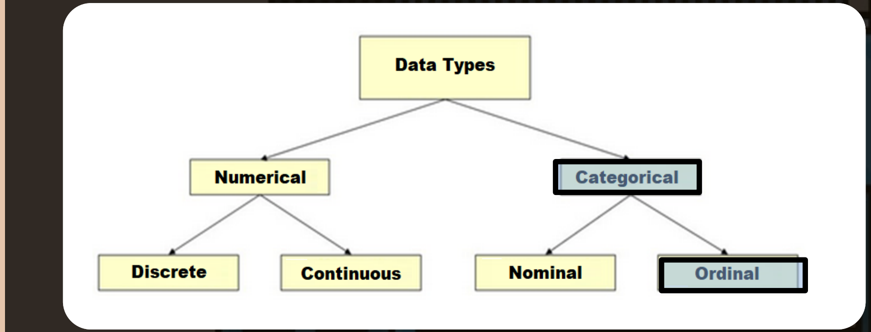
Examples of Nominal Data:

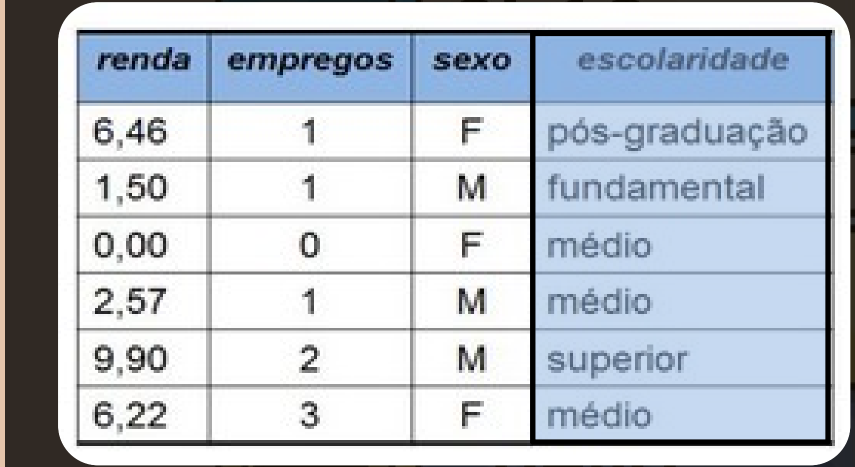
- Colour of hair (Blonde, red, Brown, Black, etc.)
- Marital status (Single, Widowed, Married)
- Nationality (Indian, German, American)



Data Types

Categorical - Ordinal





renda	empregos	sexo	escolaridade
6,46	1	F	pós-graduação
1,50	1	M	fundamental
0,00	0	F	médio
2,57	1	M	médio
9,90	2	M	superior
6,22	3	F	médio

An ordinal variable is similar to a nominal variable. The difference between the two is that there is a clear ordering of the categories. For example, suppose you have a variable, economic status, with three categories (low, medium and high). In addition to being able to classify people into these three categories, you can order the categories as low, medium and high.

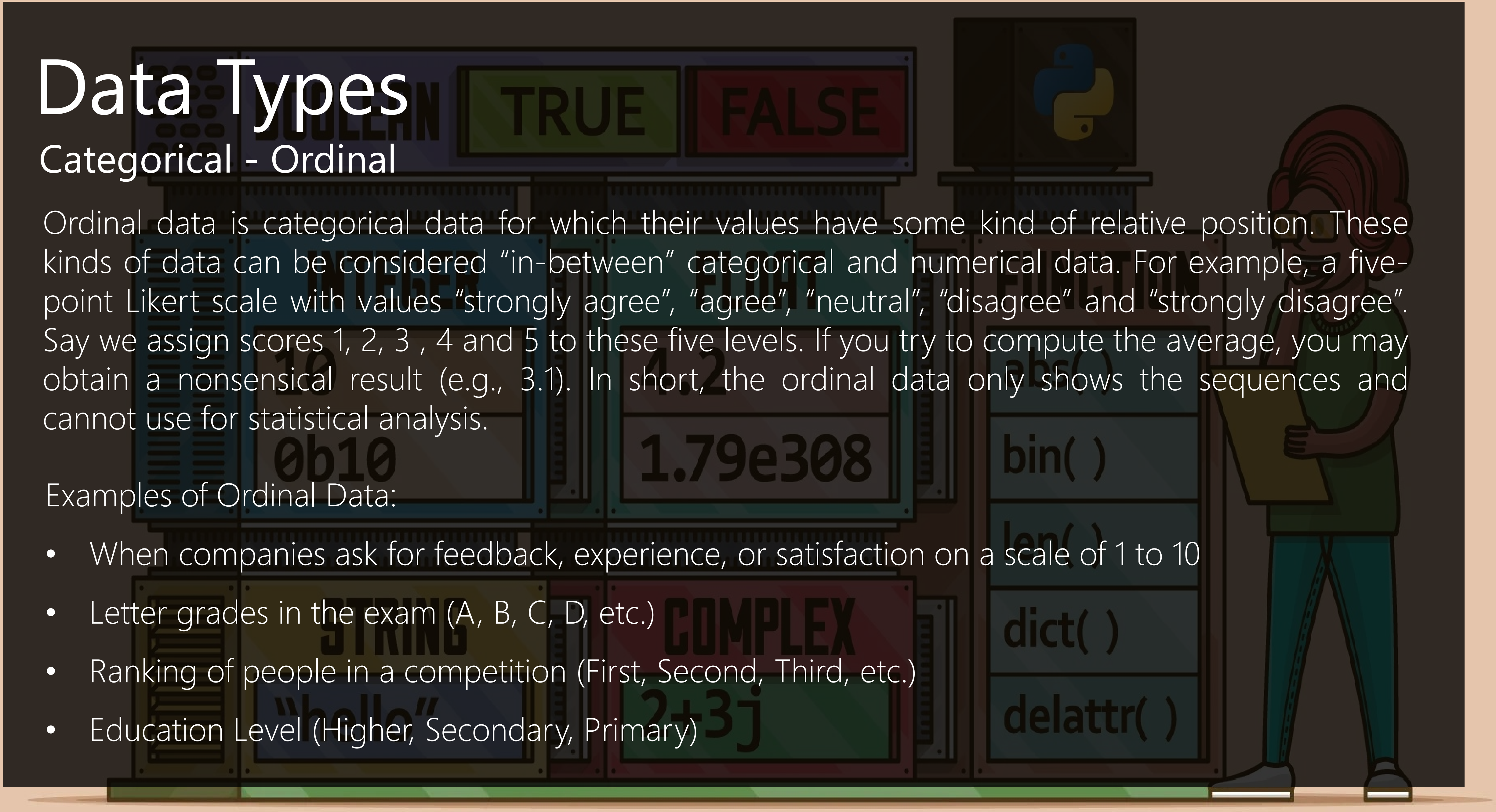
Data Types

Categorical - Ordinal

Ordinal data is categorical data for which their values have some kind of relative position. These kinds of data can be considered “in-between” categorical and numerical data. For example, a five-point Likert scale with values “strongly agree”, “agree”, “neutral”, “disagree” and “strongly disagree”. Say we assign scores 1, 2, 3 , 4 and 5 to these five levels. If you try to compute the average, you may obtain a nonsensical result (e.g., 3.1). In short, the ordinal data only shows the sequences and cannot use for statistical analysis.

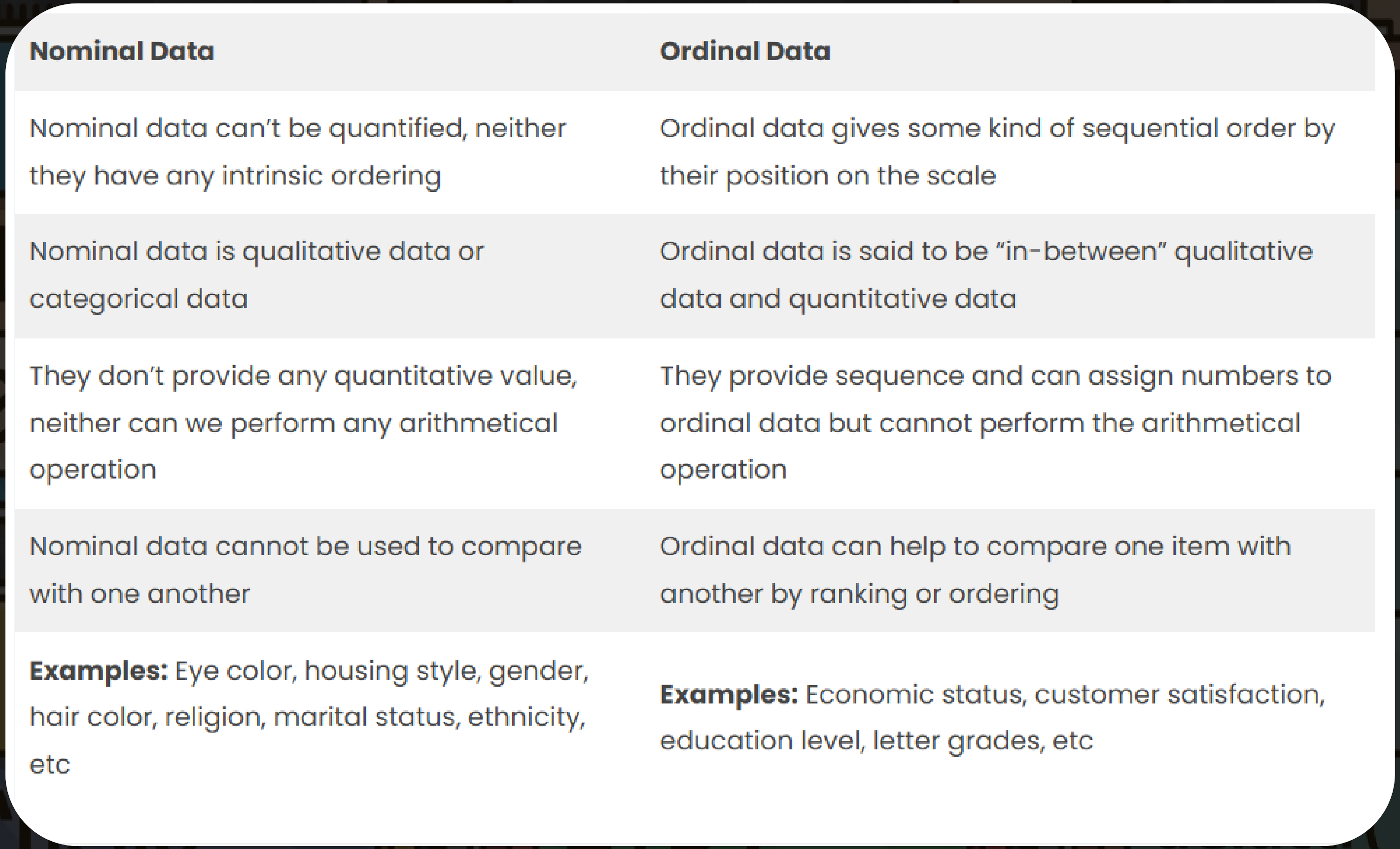
Examples of Ordinal Data:

- When companies ask for feedback, experience, or satisfaction on a scale of 1 to 10
- Letter grades in the exam (A, B, C, D, etc.)
- Ranking of people in a competition (First, Second, Third, etc.)
- Education Level (Higher, Secondary, Primary)



Data Types

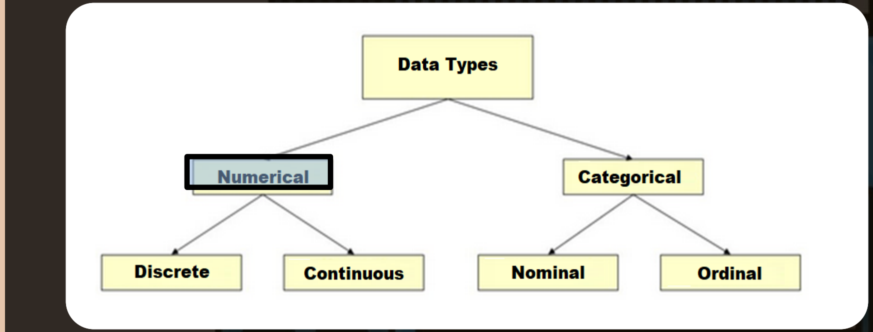
Categorical – Nominal vs Ordinal



Nominal Data	Ordinal Data
Nominal data can't be quantified, neither they have any intrinsic ordering	Ordinal data gives some kind of sequential order by their position on the scale
Nominal data is qualitative data or categorical data	Ordinal data is said to be "in-between" qualitative data and quantitative data
They don't provide any quantitative value, neither can we perform any arithmetical operation	They provide sequence and can assign numbers to ordinal data but cannot perform the arithmetical operation
Nominal data cannot be used to compare with one another	Ordinal data can help to compare one item with another by ranking or ordering
Examples: Eye color, housing style, gender, hair color, religion, marital status, ethnicity, etc	**Examples:** Economic status, customer satisfaction, education level, letter grades, etc

Data Types

Numerical



Data Types

Numerical

Categorical

Discrete

Continuous

Nominal

Ordinal

Numerical data can be expressed in numerical values, making it countable and including statistical data analysis. It answers the questions like "how much," "how many," and "how often." For example, the price of a phone, the computer's ram, the height or weight of a person, etc., falls under quantitative data.

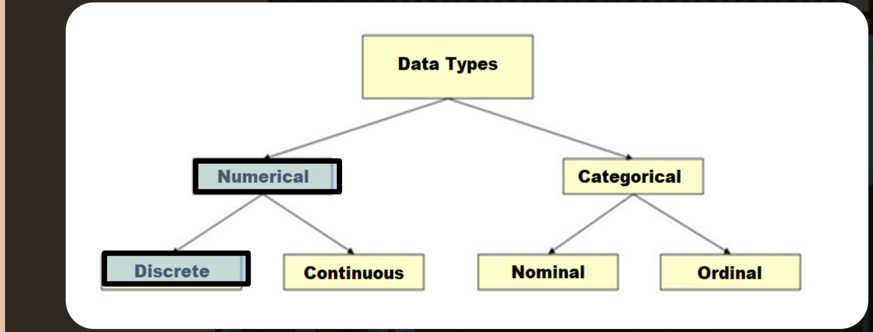
Quantitative data can be used for statistical manipulation. These data can be represented on a wide variety of graphs and charts, such as bar graphs, histograms, scatter plots, boxplots, pie charts, line graphs, etc.

Examples of Numerical Data:

- Room Temperature
- Scores and Marks (Ex: 59, 80, 60, etc.)

Data Types

Numerical - Discrete



Data Types

Numerical

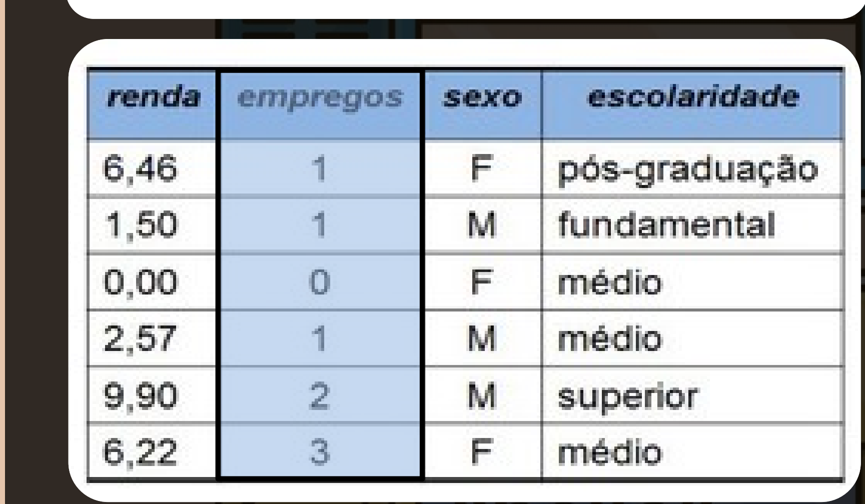
Categorical

Discrete

Continuous

Nominal

Ordinal



renda	empregos	sexo	escolaridade
6,46	1	F	pós-graduação
1,50	1	M	fundamental
0,00	0	F	médio
2,57	1	M	médio
9,90	2	M	superior
6,22	3	F	médio

The term discrete means distinct or separate. The discrete data contain the values that fall under integers or whole numbers. The total number of students in a class is an example of discrete data. These data can’t be broken into decimal or fraction values.

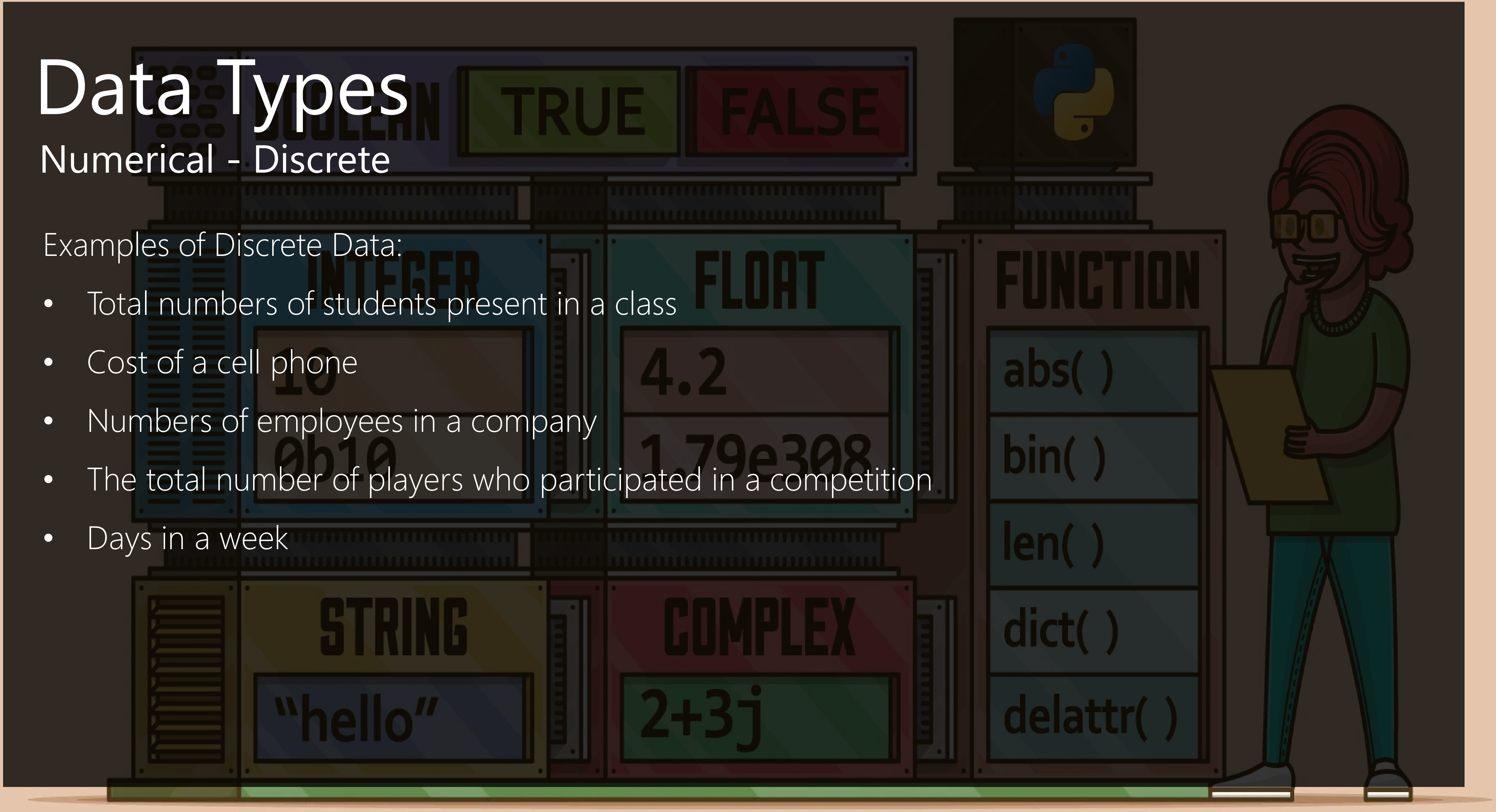
The discrete data are countable and have finite values; their subdivision is not possible. These data are represented mainly by a bar graph, number line, or frequency table.

Data Types

Numerical - Discrete

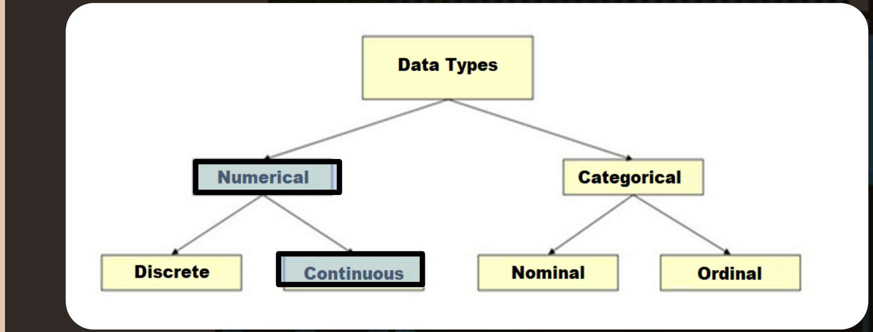
Examples of Discrete Data:

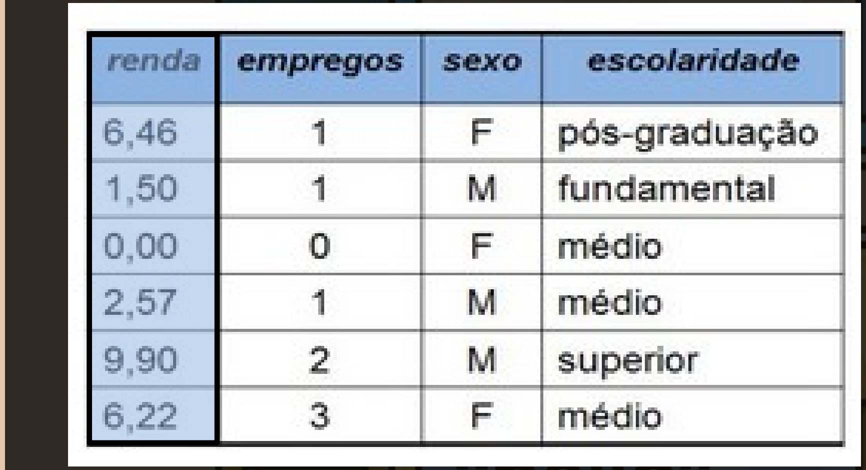
- Total numbers of students present in a class
- Cost of a cell phone
- Numbers of employees in a company
- The total number of players who participated in a competition
- Days in a week



Data Types

Numerical - Continuous





renda	empregos	sexo	escolaridade
6,46	1	F	pós-graduação
1,50	1	M	fundamental
0,00	0	F	médio
2,57	1	M	médio
9,90	2	M	superior
6,22	3	F	médio

Continuous data are in the form of fractional numbers. It can be the version of an android phone, the height of a person, the length of an object, etc. Continuous data represents information that can be divided into smaller levels. The continuous variable can take any value within a range.

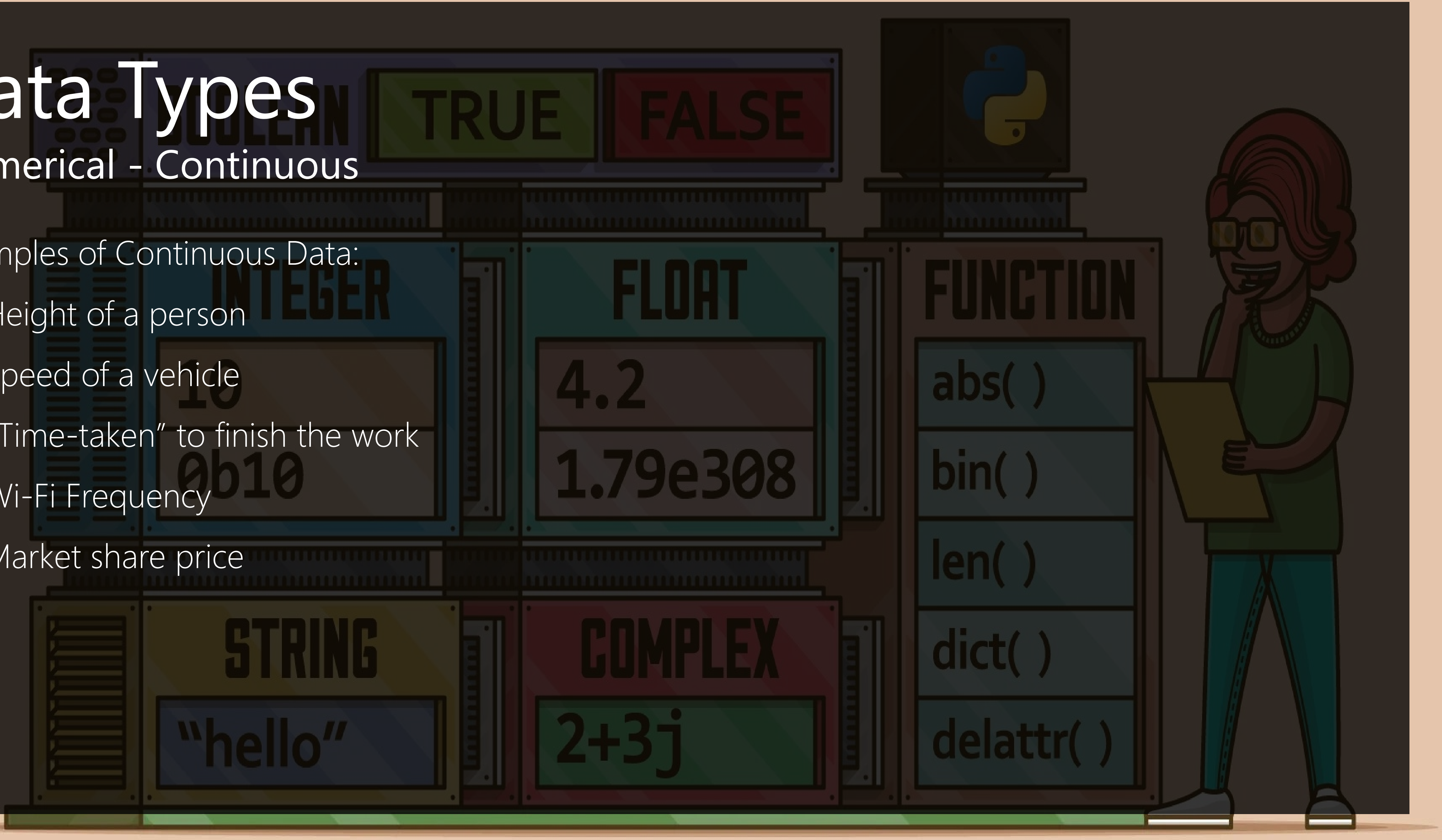
The key difference between discrete and continuous data is that discrete data contains the integer or whole number. Still, continuous data stores the fractional numbers to record different types of data such as temperature, height, width, time, speed, etc.

Data Types

Numerical - Continuous

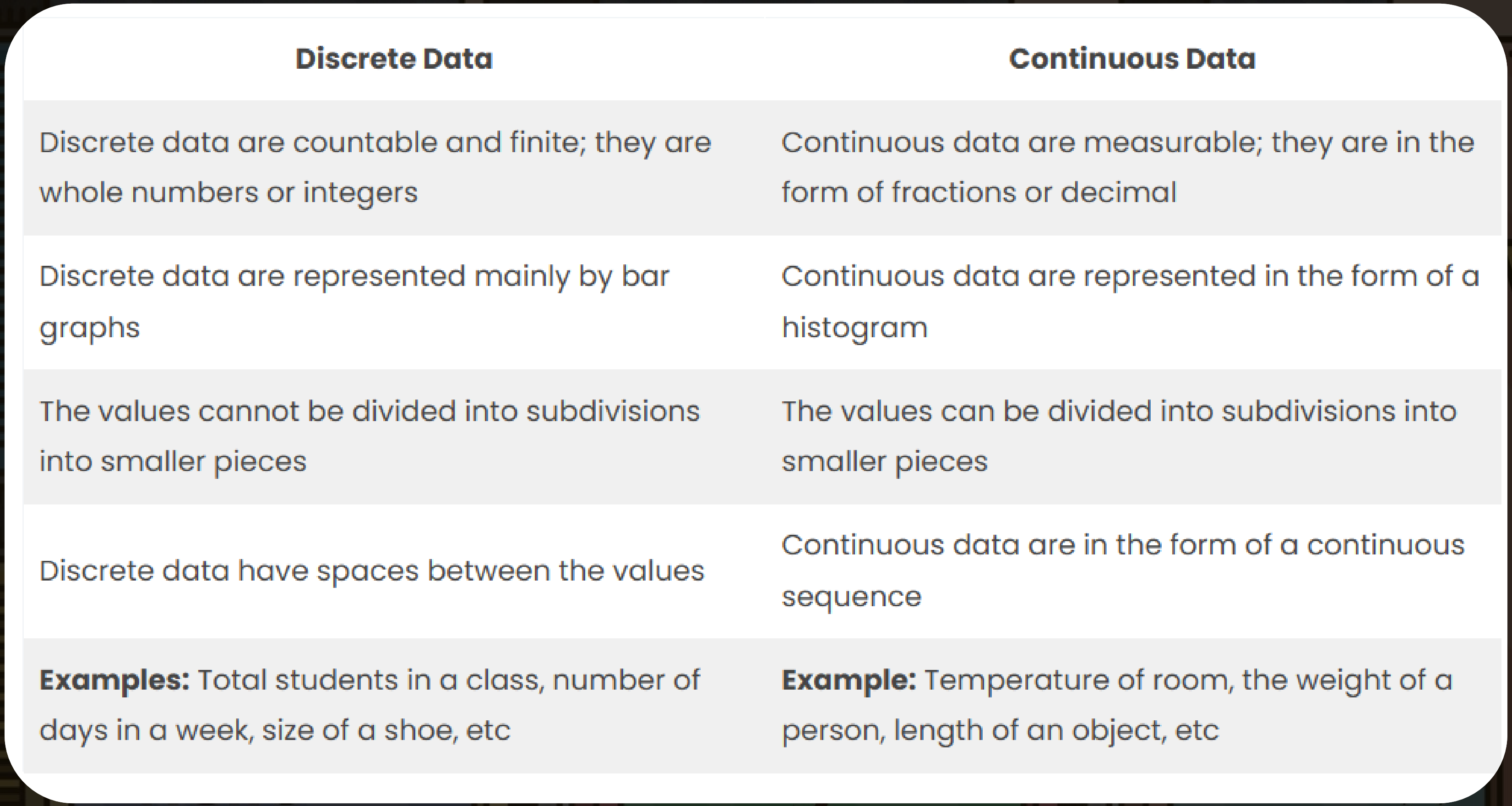
Examples of Continuous Data:

- Height of a person
- Speed of a vehicle
- “Time-taken” to finish the work
- Wi-Fi Frequency
- Market share price



Data Types

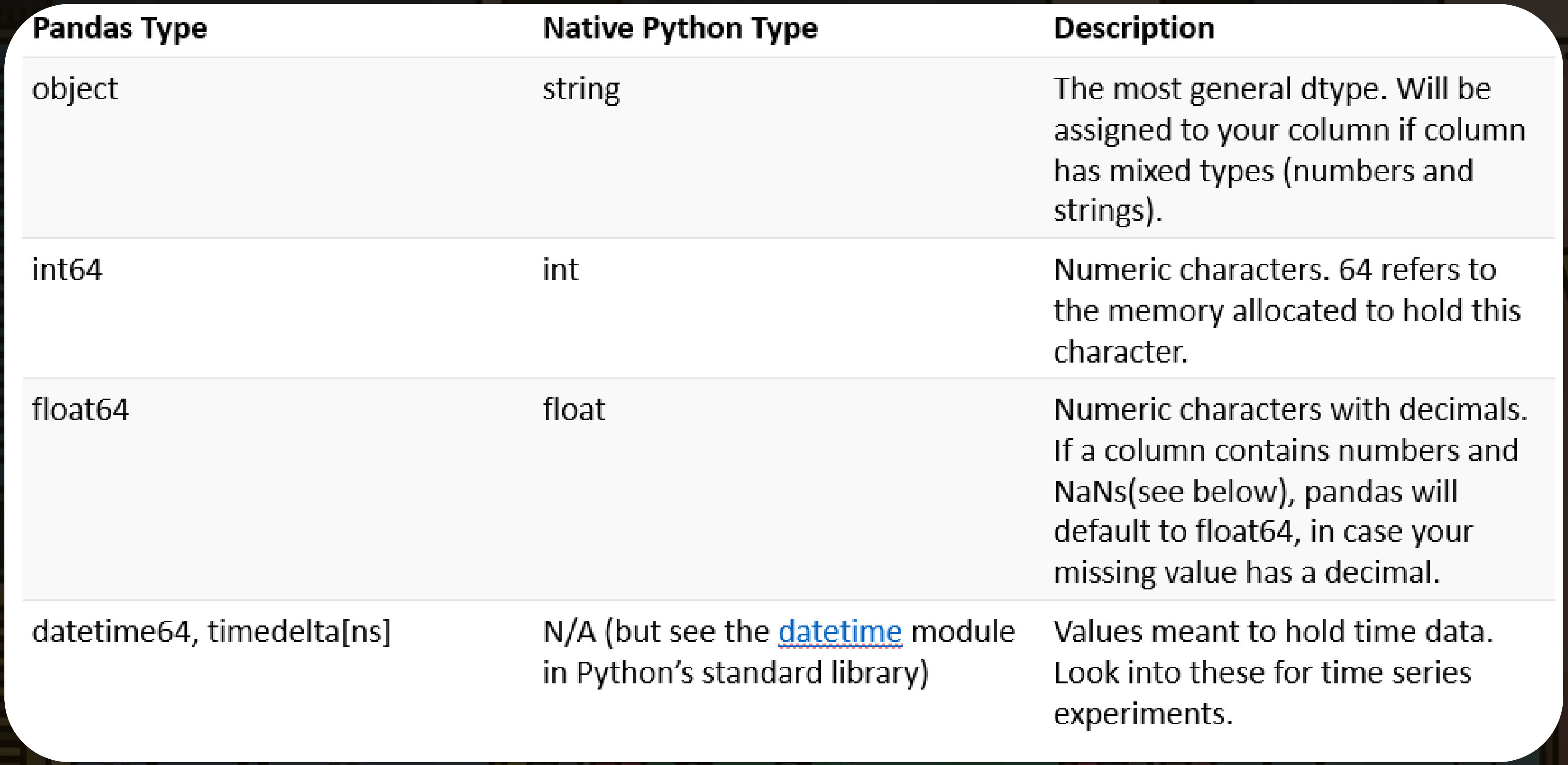
Numerical – Continuous vs Discrete



Discrete Data	Continuous Data
Discrete data are countable and finite; they are whole numbers or integers	Continuous data are measurable; they are in the form of fractions or decimal
Discrete data are represented mainly by bar graphs	Continuous data are represented in the form of a histogram
The values cannot be divided into subdivisions into smaller pieces	The values can be divided into subdivisions into smaller pieces
Discrete data have spaces between the values	Continuous data are in the form of a continuous sequence
Examples: Total students in a class, number of days in a week, size of a shoe, etc	**Example:** Temperature of room, the weight of a person, length of an object, etc

Data Types

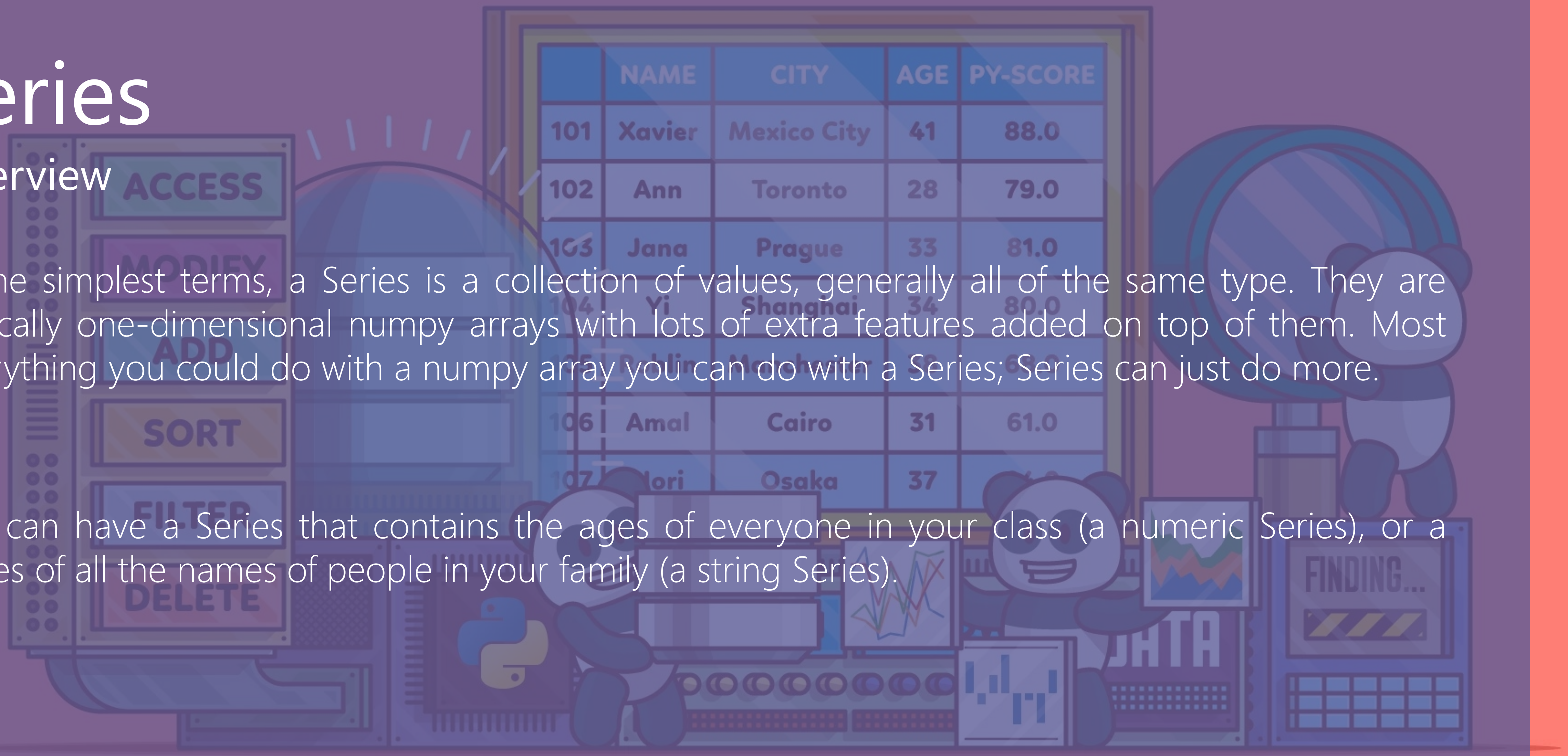
Pandas Data Types



Pandas Type	Native Python Type	Description
object	string	The most general dtype. Will be assigned to your column if column has mixed types (numbers and strings).
int64	int	Numeric characters. 64 refers to the memory allocated to hold this character.
float64	float	Numeric characters with decimals. If a column contains numbers and NaNs(see below), pandas will default to float64, in case your missing value has a decimal.
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	Values meant to hold time data. Look into these for time series experiments.

Series

Overview



In the simplest terms, a Series is a collection of values, generally all of the same type. They are basically one-dimensional numpy arrays with lots of extra features added on top of them. Most everything you could do with a numpy array you can do with a Series; Series can just do more.

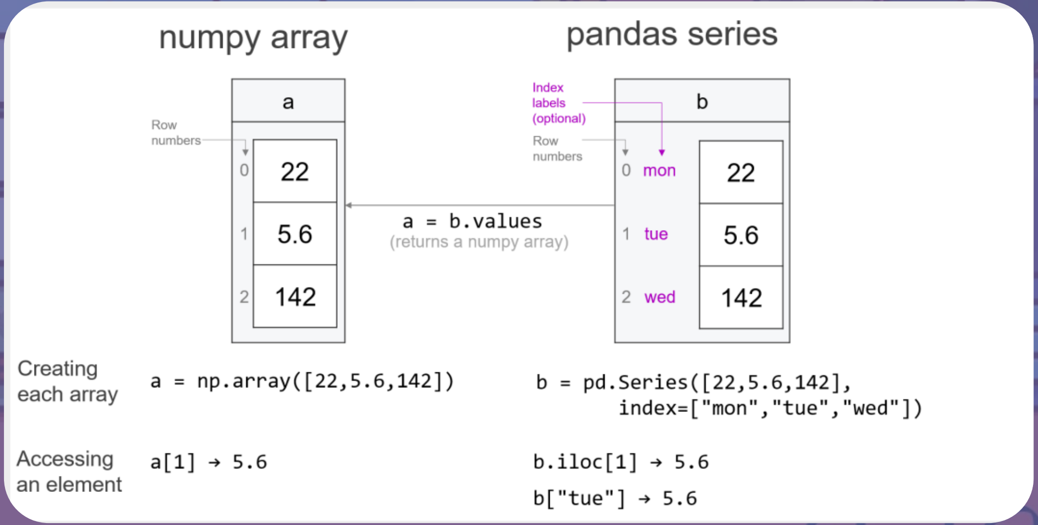
You can have a Series that contains the ages of everyone in your class (a numeric Series), or a Series of all the names of people in your family (a string Series).

Series

Overview

One of the fundamental differences between numpy arrays and Series is that all Series are associated with an index. An index is a set of labels for each observation in a Series. If you don't specify an index when you create a Series, pandas will create a default index that just labels each row with it's initial row number, but you can specify an index if you want.

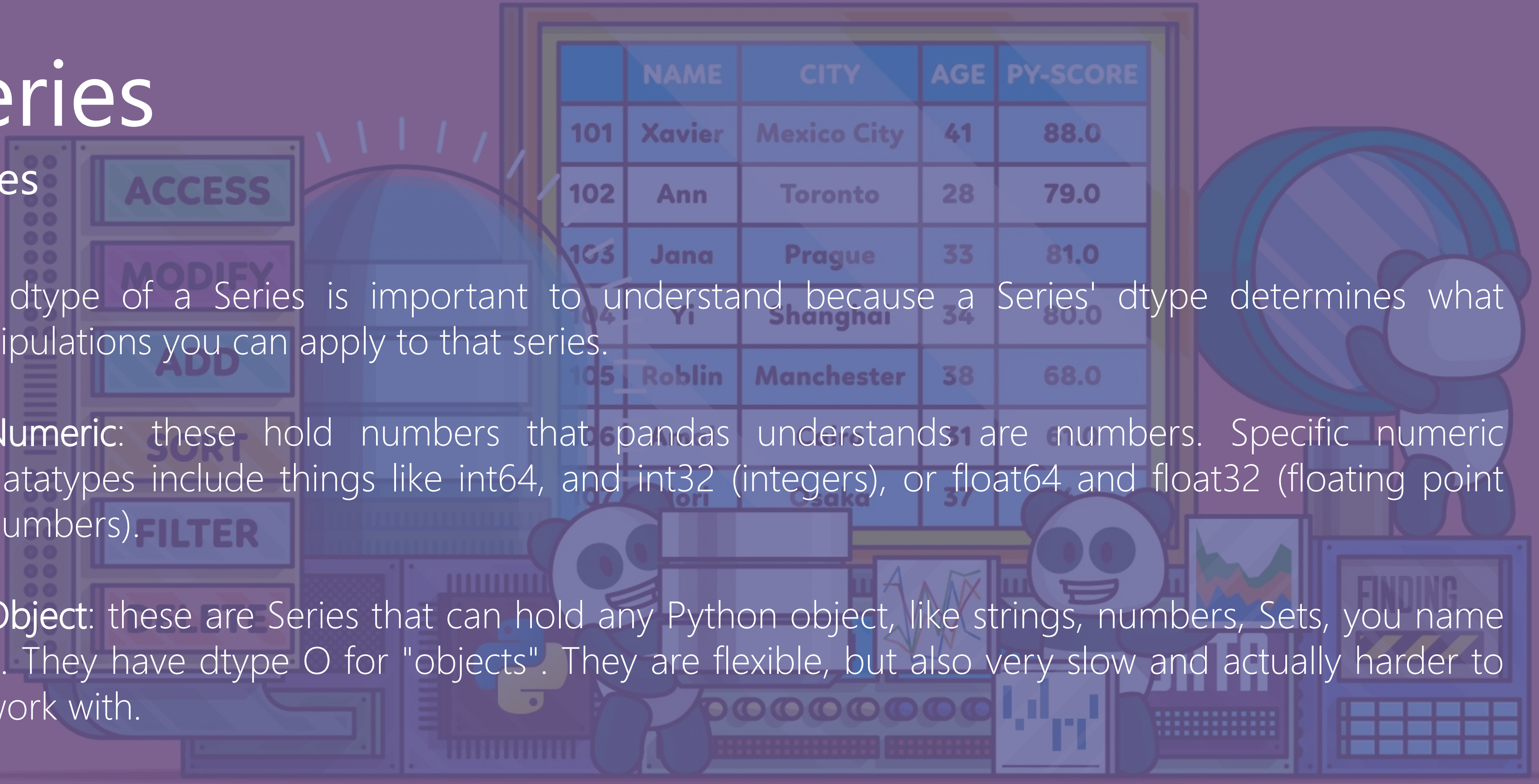
For example, if we were representing financial data for three days of a week: 'mon', tue', and 'wed', we may be able do this as shown in the figure below. Once we've created our pandas Series, we can return a numpy array. if b contains a pandas Series, then we can return a numpy array with b.values.



```
a = np.array([22,5.6,142])
b = pd.Series([22,5.6,142],
       index=["mon","tue","wed"])

a[1] → 5.6
b.iloc[1] → 5.6
b["tue"] → 5.6
```

Series

Types



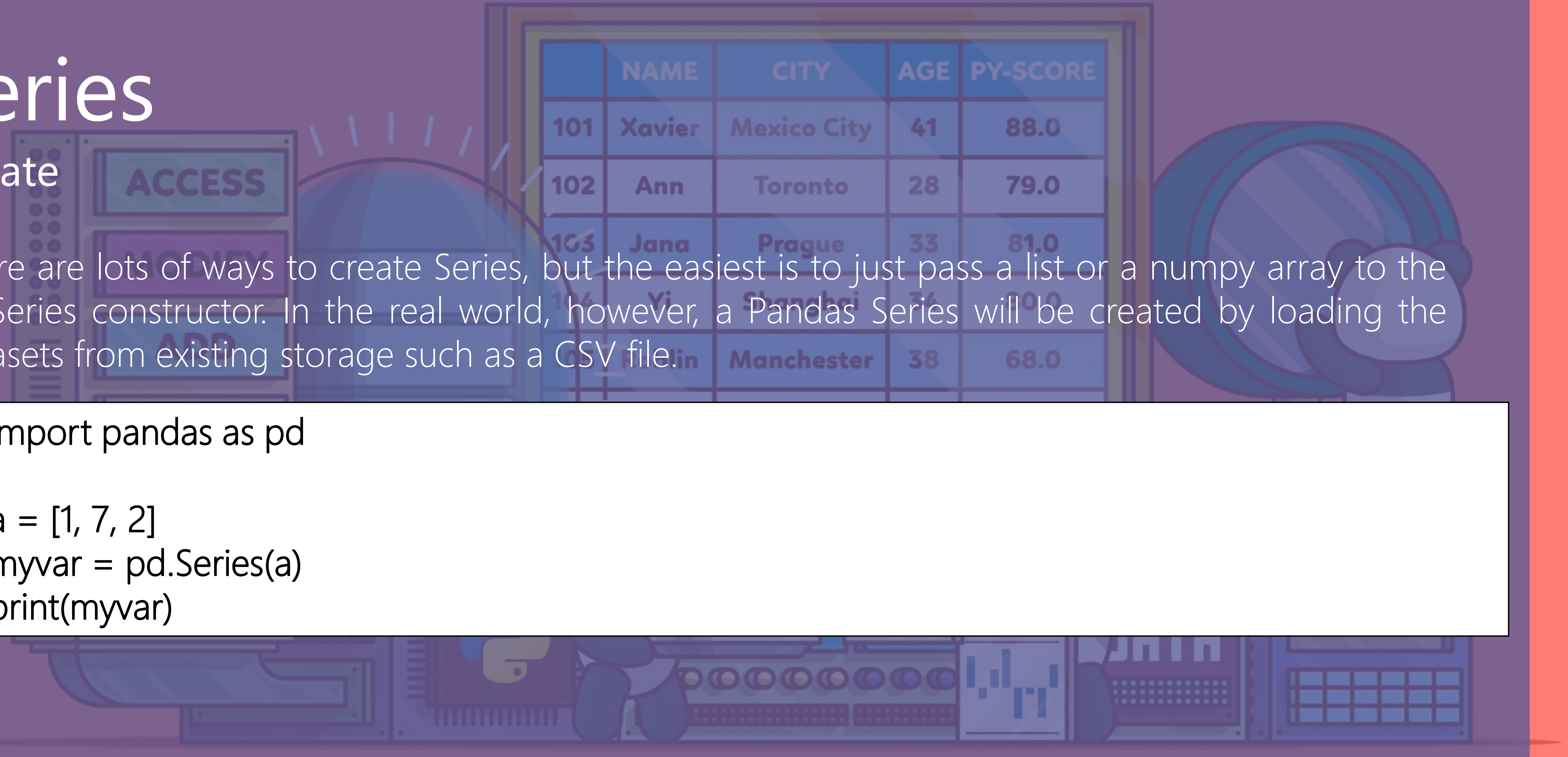
The dtype of a Series is important to understand because a Series' dtype determines what manipulations you can apply to that series.

- **Numeric**: these hold numbers that pandas understands are numbers. Specific numeric datatypes include things like int64, and int32 (integers), or float64 and float32 (floating point numbers).

- **Object**: these are Series that can hold any Python object, like strings, numbers, Sets, you name it. They have dtype O for "objects". They are flexible, but also very slow and actually harder to work with.

Series

Create

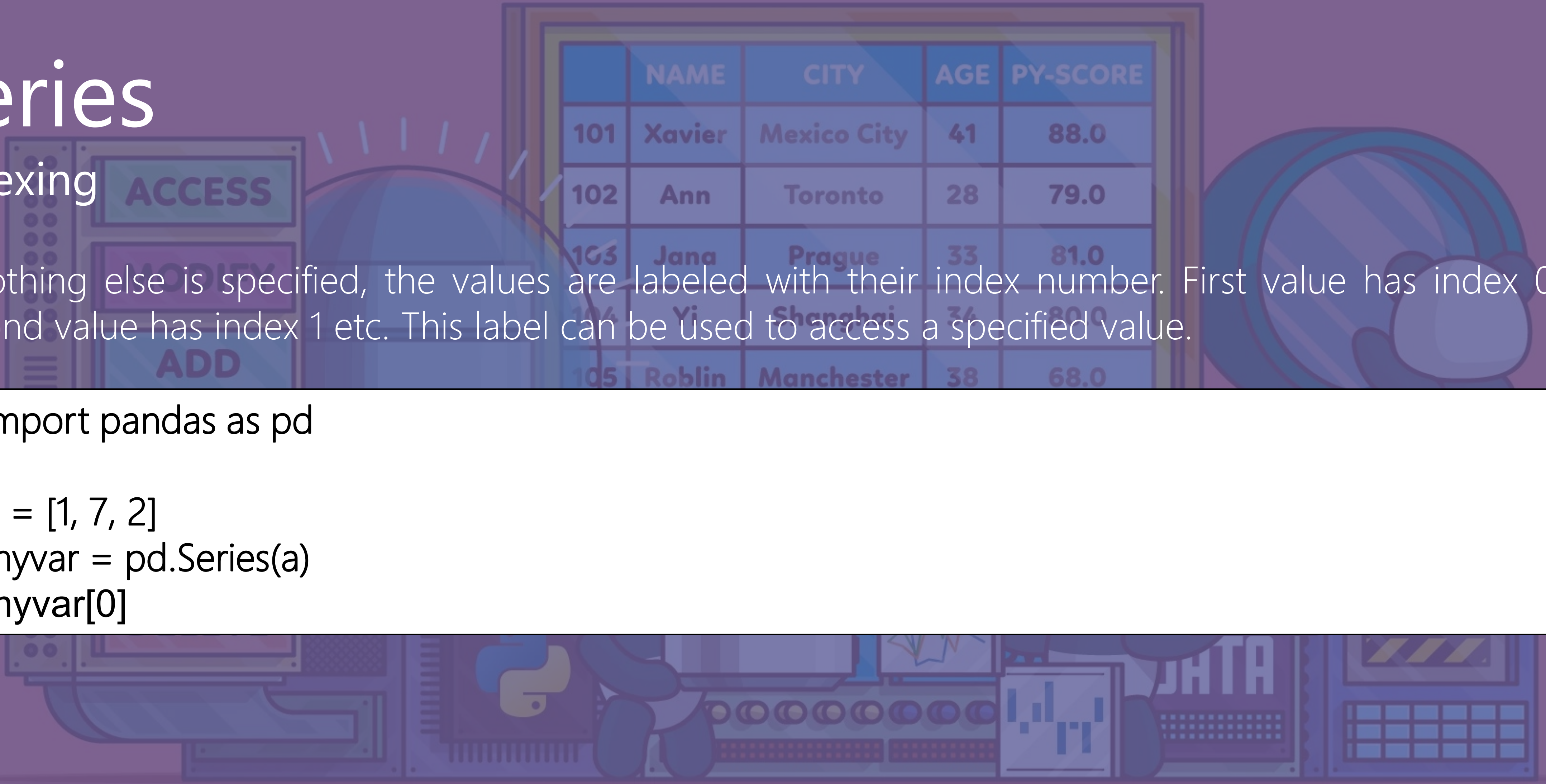


There are lots of ways to create Series, but the easiest is to just pass a list or a numpy array to the pd.Series constructor. In the real world, however, a Pandas Series will be created by loading the datasets from existing storage such as a CSV file.

```
import pandas as pd

a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

Series

Indexing

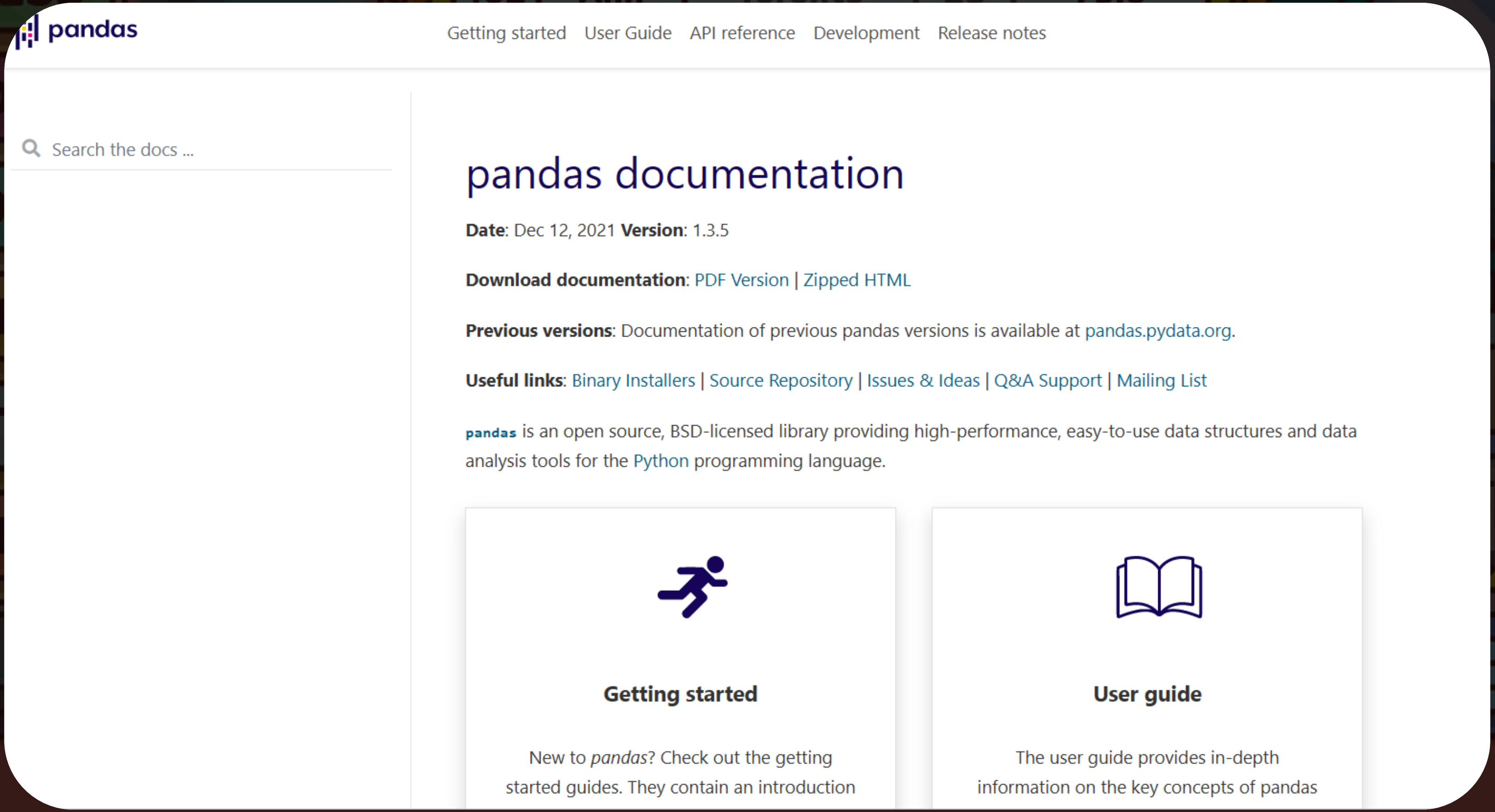


If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc. This label can be used to access a specified value.

```
import pandas as pd

a = [1, 7, 2]
myvar = pd.Series(a)
myvar[0]
```

Dataframes

Overview



pandas documentation

Date: Dec 12, 2021 **Version**: 1.3.5

Download documentation: PDF Version | Zipped HTML

Previous versions: Documentation of previous pandas versions is available at pandas.pydata.org.

Useful links: Binary Installers | Source Repository | Issues & Ideas | Q&A Support | Mailing List

`pandas` is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Getting started

New to *pandas*? Check out the getting started guides. They contain an introduction

User guide

The user guide provides in-depth information on the key concepts of pandas

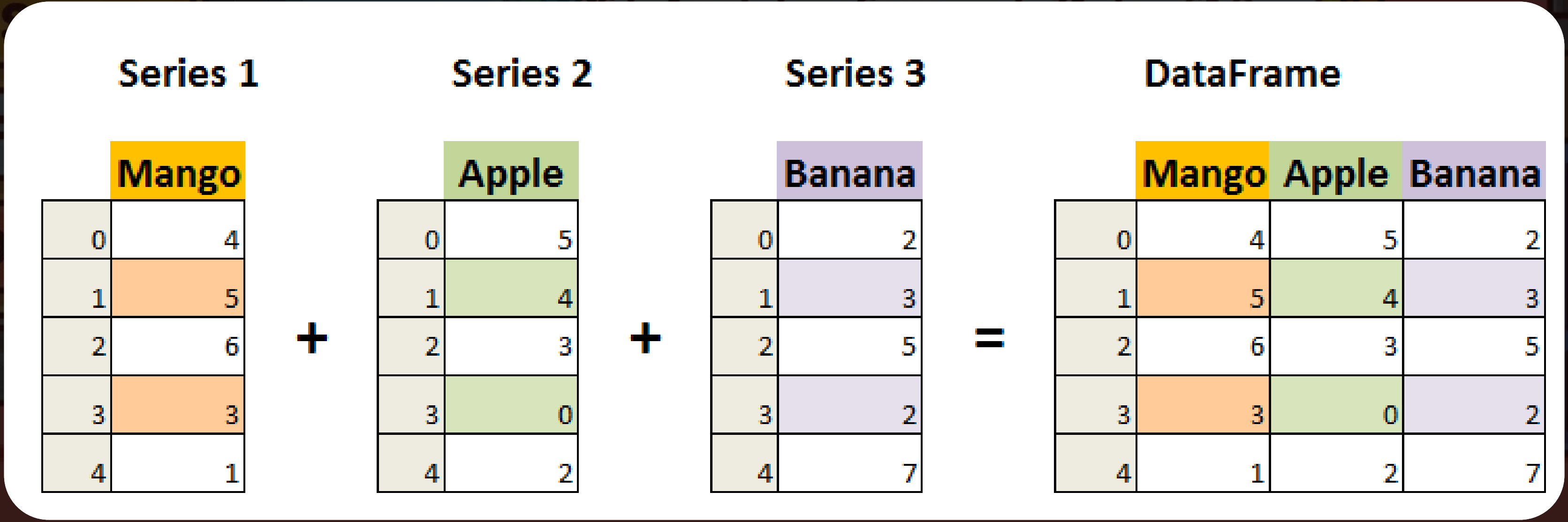
https://pandas.pydata.org/pandas-docs/stable/index.html

Real Python

Dataframes

Overview

Data sets in Pandas are usually multi-dimensional tables, called DataFrames. Series is like a column, a DataFrame is the whole table. Each column is actually a Series. More specifically, the DataFrame is a dictionary of Series, all the same size.



Series 1

	Mango
0	4
1	5
2	6
3	3
4	1

+

Series 2

	Apple
0	5
1	4
2	3
3	0
4	2

+

Series 3

	Banana
0	2
1	3
2	5
3	2
4	7

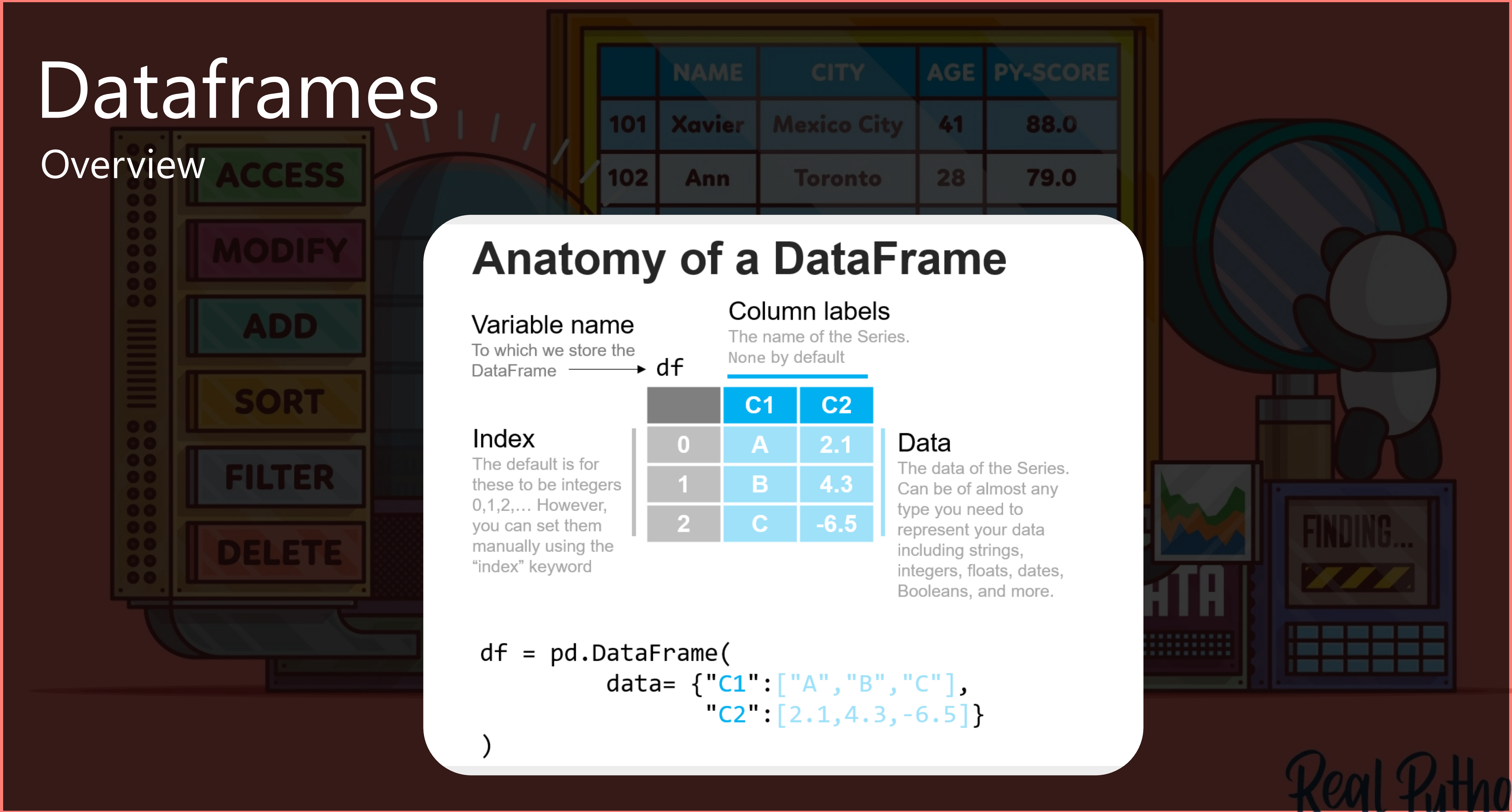
=

DataFrame

	Mango	Apple	Banana
0	4	5	2
1	5	4	3
2	6	3	5
3	3	0	2
4	1	2	7

Dataframes

Overview



Anatomy of a DataFrame

Variable name
To which we store the DataFrame → df

Column labels
The name of the Series.
None by default

	C1	C2
0	A	2.1
1	B	4.3
2	C	-6.5

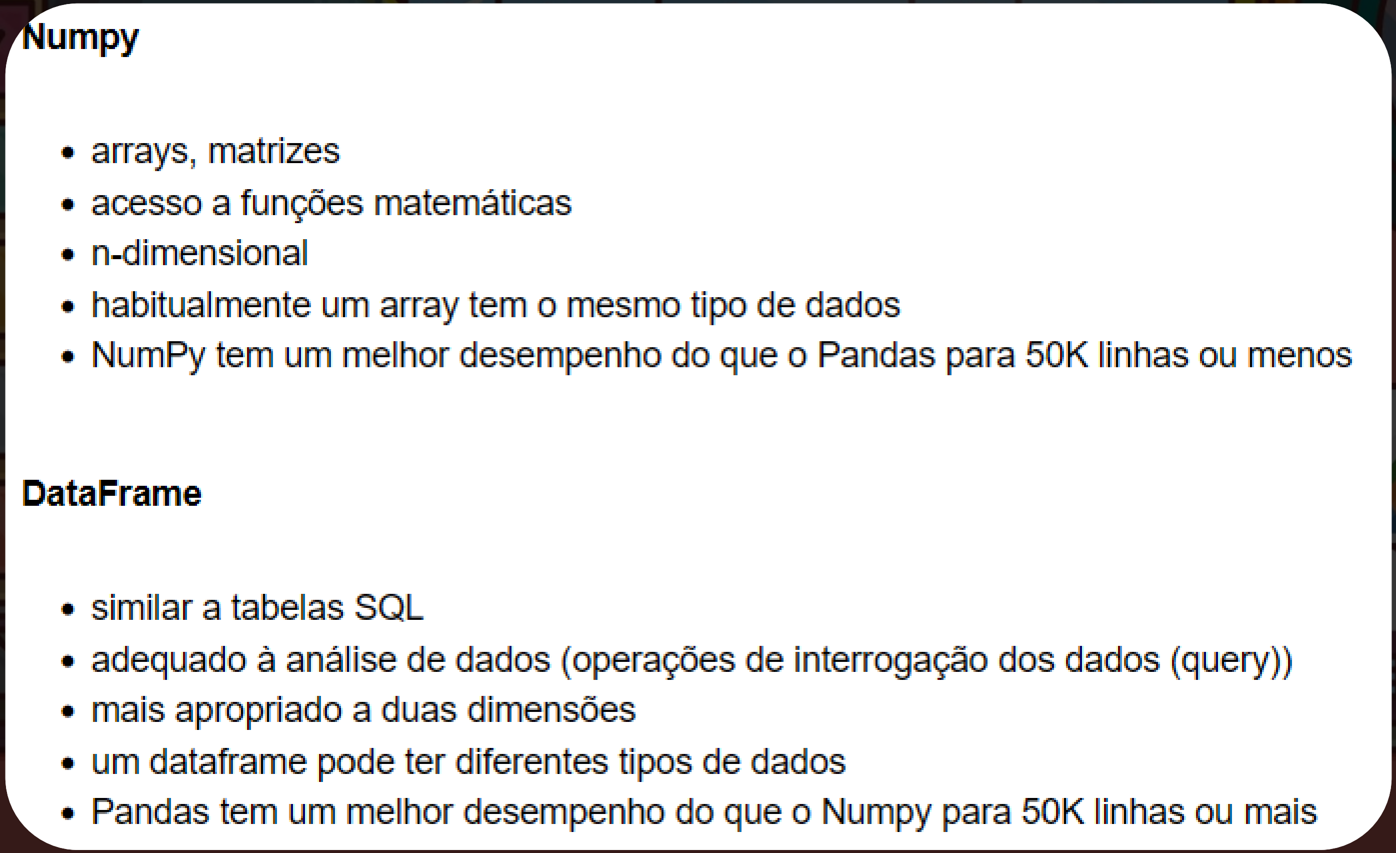
Index
The default is for these to be integers 0,1,2,… However, you can set them manually using the "index" keyword

Data
The data of the Series. Can be of almost any type you need to represent your data including strings, integers, floats, dates, Booleans, and more.

```
df = pd.DataFrame(
        data= {"C1":["A","B","C"],
               "C2":[2.1,4.3,-6.5]}
)
```

Dataframes

Overview



Numpy

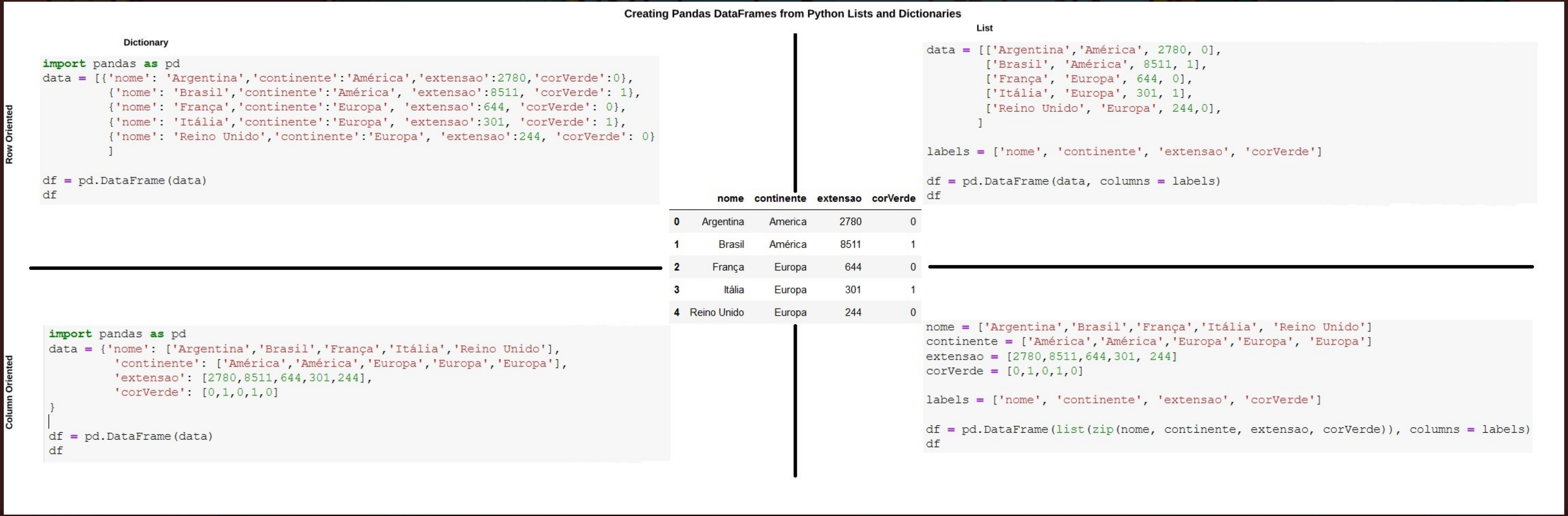
- arrays, matrizes
- acesso a funções matemáticas
- n-dimensional
- habitualmente um array tem o mesmo tipo de dados
- NumPy tem um melhor desempenho do que o Pandas para 50K linhas ou menos

DataFrame

- similar a tabelas SQL
- adequado à análise de dados (operações de interrogação dos dados (query))
- mais apropriado a duas dimensões
- um dataframe pode ter diferentes tipos de dados
- Pandas tem um melhor desempenho do que o Numpy para 50K linhas ou mais

Dataframes

Create



Creating Pandas DataFrames from Python Lists and Dictionaries

Dictionary

Row Oriented

```
import pandas as pd
data = [{'nome': 'Argentina','continente':'América','extensao':2780,'corVerde':0},
         {'nome': 'Brasil','continente':'América', 'extensao':8511, 'corVerde': 1},
         {'nome': 'França','continente':'Europa', 'extensao':644, 'corVerde': 0},
         {'nome': 'Itália','continente':'Europa', 'extensao':301, 'corVerde': 1},
         {'nome': 'Reino Unido','continente':'Europa', 'extensao':244, 'corVerde': 0}
         ]

df = pd.DataFrame(data)
df
```

List

```
data = [['Argentina','América', 2780, 0],
        ['Brasil', 'América', 8511, 1],
        ['França', 'Europa', 644, 0],
        ['Itália', 'Europa', 301, 1],
        ['Reino Unido', 'Europa', 244,0],
       ]

labels = ['nome', 'continente', 'extensao', 'corVerde']

df = pd.DataFrame(data, columns = labels)
df
```

	nome	continente	extensao	corVerde
0	Argentina	America	2780	0
1	Brasil	América	8511	1
2	França	Europa	644	0
3	Itália	Europa	301	1
4	Reino Unido	Europa	244	0

Column Oriented

```
import pandas as pd
data = {'nome': ['Argentina','Brasil','França','Itália','Reino Unido'],
         'continente': ['América','América','Europa','Europa','Europa'],
         'extensao': [2780,8511,644,301,244],
         'corVerde': [0,1,0,1,0]
}

df = pd.DataFrame(data)
df
```

```
nome = ['Argentina','Brasil','França','Itália', 'Reino Unido']
continente = ['América','América','Europa','Europa', 'Europa']
extensao = [2780,8511,644,301, 244]
corVerde = [0,1,0,1,0]

labels = ['nome', 'continente', 'extensao', 'corVerde']

df = pd.DataFrame(list(zip(nome, continente, extensao, corVerde)), columns = labels)
df
```

Real Python

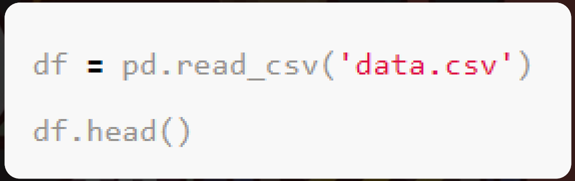
Dataframes

Import/Export (file formats)

Pandas allows you to import data from a wide range of data sources directly into a dataframe. These can be static files, such as CSV, TSV, Microsoft Excel, JSON, etc. You can even scrape data directly from web pages into Pandas dataframes.

CSV

Comma Separated Value or CSV files are likely to be the file format you encounter most commonly in data science. As the name suggests, these are simple text files in which the values are separated (usually) by commas



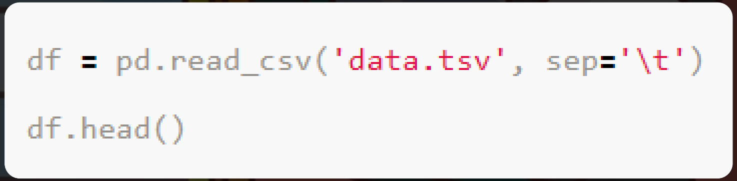
```
df = pd.read_csv('data.csv')
df.head()
```

You can also use read_csv() to read remote CSV files. Instead of passing in the path to the file you provide the full URL to the CSV file.

Dataframes

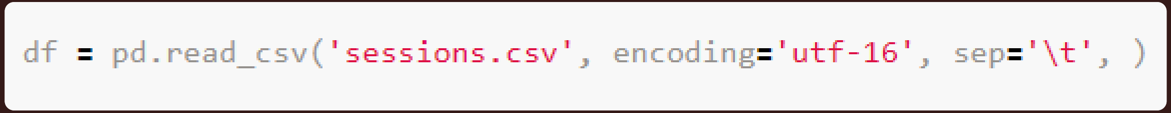
Import/Export (file formats)

Tab separated value files are just like CSVs, but the values are separated by a tab instead of a comma. They can also be read using the same read_csv() function, you just need to specify the separator character used. For tabs, this is \t.



```
df = pd.read_csv('data.tsv', sep='\t')
df.head()
```

Sometimes when importing data into Pandas things do not go to plan and Pandas will throw an error. There are two main reasons for this - at least in the files I regularly deal with. Firstly, the file encoding may not be set to utf-8, which causes Pandas to throw an error stating UnicodeDecodeError: 'utf-8' codec can't decode byte. You can usually resolve this by specifying the file encoding i.e. encoding='utf-16' and the problem should be resolved.



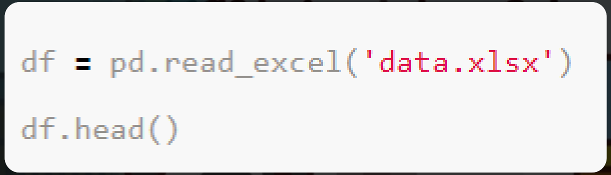
```
df = pd.read_csv('sessions.csv', encoding='utf-16', sep='\t', )
```

Dataframes

Import/Export (file formats)

Excel

Pandas includes built-in functionality for reading Microsoft Excel spreadsheet files via its read_excel() function. This works in the same way as read_csv() so can be used on local Excel documents as well as remote files,



```
df = pd.read_excel('data.xlsx')
df.head()
```

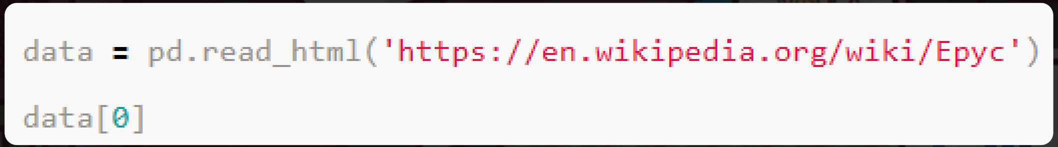
Dataframes

Import/Export (file formats)

HTML

One other handy feature of Pandas is the read_html() function. This allows you to parse HTML markup from remote web pages or local HTML documents and extract any tables present. In the example below I've extracted an HTML table from a Wikipedia page.

The read_html() function returns any tables it finds in a list, so if more than one is present, you'll need to define which one to display in your dataframe using its list index, which starts from zero.



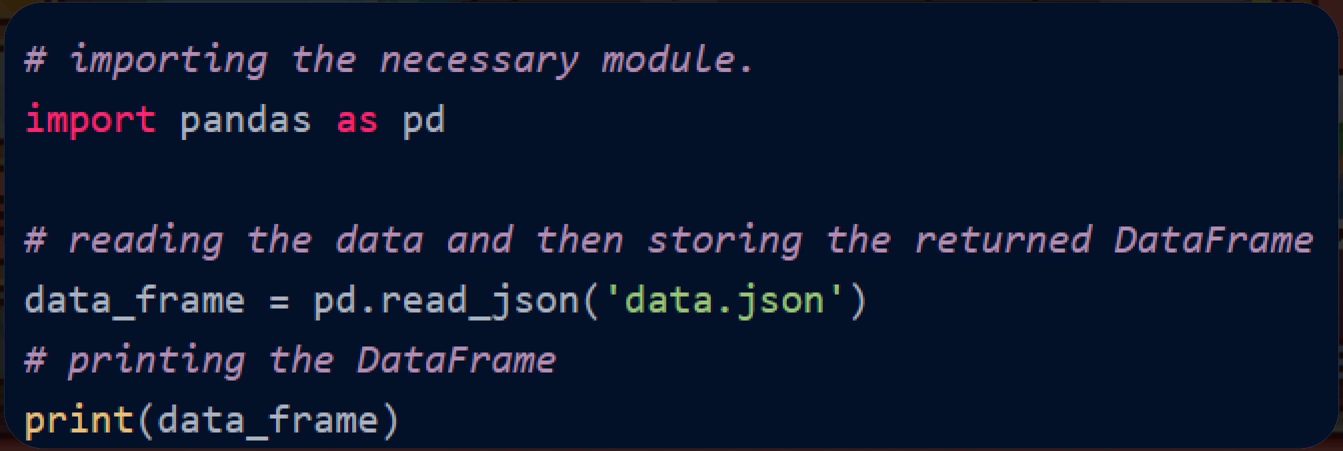
```
data = pd.read_html('https://en.wikipedia.org/wiki/Epyc')

data[0]
```

Dataframes

Import/Export (file formats)

JSON

Big data sets are often stored, or extracted as JSON. The read_json() is a function in the Pandas library that helps us read JSON. The read_json() function takes a JSON file and returns a DataFrame



```
# importing the necessary module.
import pandas as pd

# reading the data and then storing the returned DataFrame
data_frame = pd.read_json('data.json')
# printing the DataFrame
print(data_frame)
```

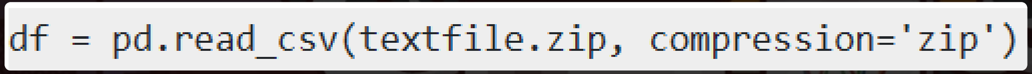
Dataframes

Import/Export (file formats)

ZIP

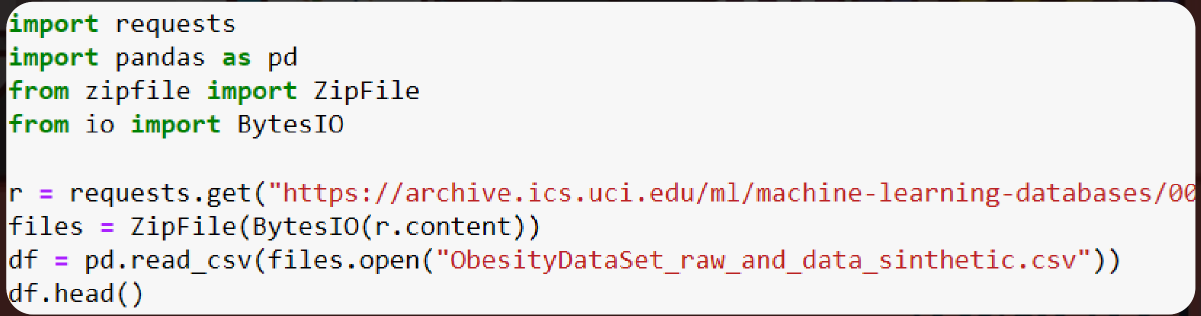
You can use ZIP files for bundling regular files together into a single archive, compressing your data to save some disk space, distributing your digital products, and more. Python’s zipfile is a standard library module intended to manipulate ZIP files.

If there is only one file per zip you could use the compression method with read_csv like below:



```
df = pd.read_csv(textfile.zip, compression='zip')
```

If the data file format is .zip, it might contain several files. To read such data files, we need to unzip it into a variable in memory and parse it using io.BytesIO.

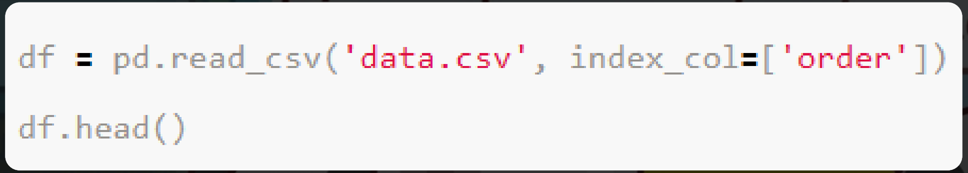


```
import requests
import pandas as pd
from zipfile import ZipFile
from io import BytesIO

r = requests.get("https://archive.ics.uci.edu/ml/machine-learning-databases/00
files = ZipFile(BytesIO(r.content))
df = pd.read_csv(files.open("ObesityDataSet_raw_and_data_sinthetic.csv"))
df.head()
```

Dataframes

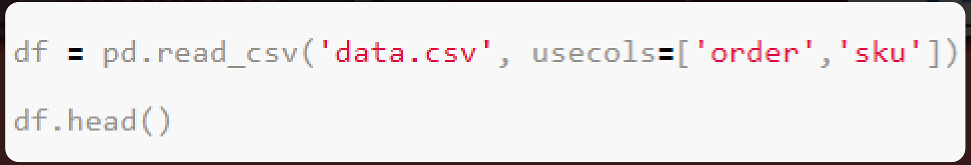
Import/Export (define specific fields)

Sometimes you may want to have one of your columns, such as the order ID, set as the index on your dataframe. Again, it's easy enough to do this after you've read the data, but it's much neater and quicker to do it during import. You can do this by passing a list of index columns to the index_col argument.



```
df = pd.read_csv('data.csv', index_col=['order'])
df.head()
```

If your data set includes hundreds of columns and you only need a specific subset of them, you can use the usecols argument to define the list of column names to import. This saves the hassle of importing all of the columns and then dropping the ones you don't need.

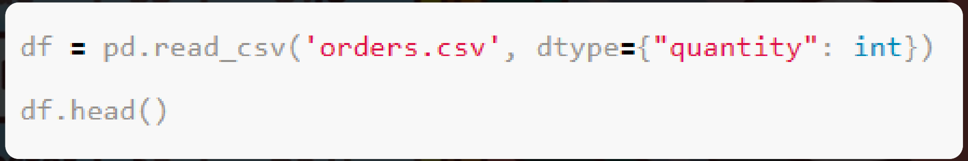


```
df = pd.read_csv('data.csv', usecols=['order','sku'])
df.head()
```

Dataframes

Import/Export (define specific fields)

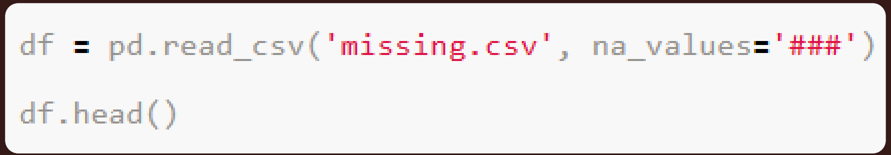
Another really common issue when dealing with data in Pandas is that the data you’re importing isn’t being identified with the correct data type for each column.



```
df = pd.read_csv('orders.csv', dtype={"quantity": int})
df.head()
```

Depending on the data source, missing values in a data set can be shown in a variety of ways. By default, Pandas recognises the presence of certain common missing value identifiers and replaces them with NaN. These values are: '', '#N/A', '#N/A N/A', '#NA', '-1.#IND', '-1.#QNAN', '-NaN', '-nan', '1.#IND', '1.#QNAN', '<NA>', 'N/A', 'NA', 'NULL', 'NaN', 'n/a', 'nan', and 'null’.

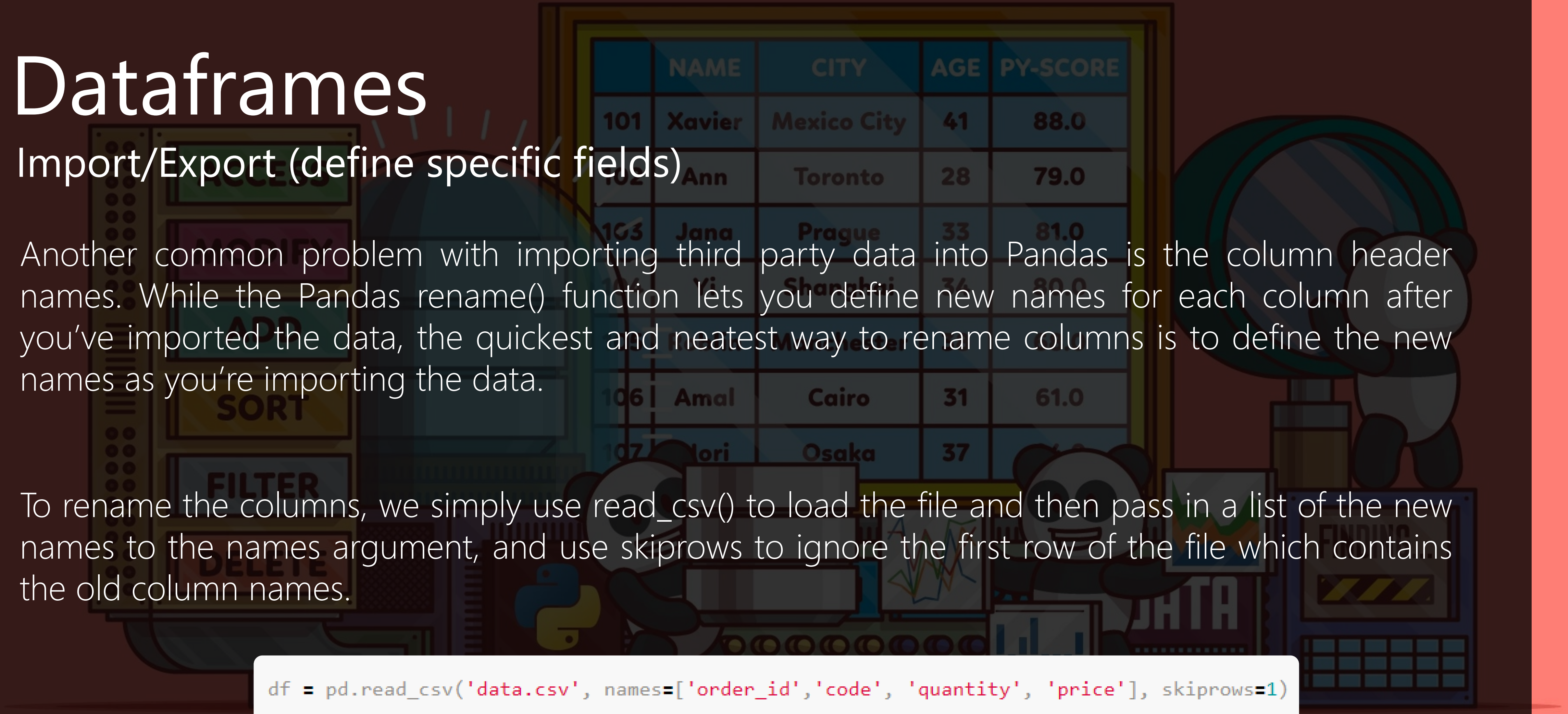
If missing values in your data set take some other form, you can specifically tell Pandas to interpret them as NaN values. In the example below, missing values are represented by ###.



```
df = pd.read_csv('missing.csv', na_values='###')
df.head()
```

Dataframes

Import/Export (define specific fields)



Another common problem with importing third party data into Pandas is the column header names. While the Pandas rename() function lets you define new names for each column after you've imported the data, the quickest and neatest way to rename columns is to define the new names as you're importing the data.

To rename the columns, we simply use read_csv() to load the file and then pass in a list of the new names to the names argument, and use skiprows to ignore the first row of the file which contains the old column names.

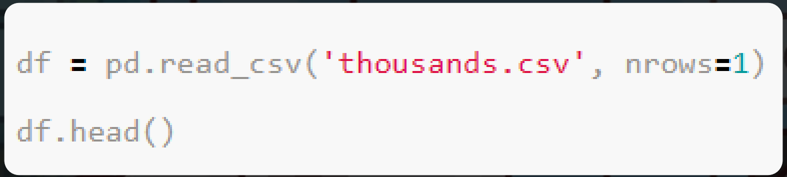
```
df = pd.read_csv('data.csv', names=['order_id','code', 'quantity', 'price'], skiprows=1)

df.head()
```

Real Python

Dataframes

Import/Export (define specific fields)

If you're dealing with massive datasets you may not always want to load the entire file. To restrict the number of rows that are read in you can pass an integer representing the number of rows to the nrows argument of read_csv().



```
df = pd.read_csv('thousands.csv', nrows=1)

df.head()
```

Real Python

Thank You

Ricardo Campos

[www.di.ubi.pt/~rcampos]