Python level 1

Introductory course
Class 2

Fábio Neves & Luis Dias

September 11

AGENDA

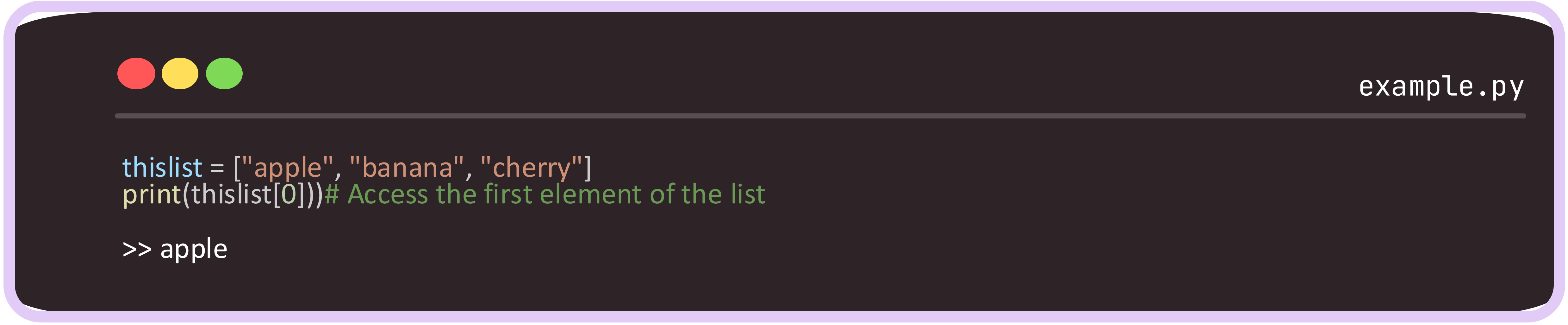
Part 1 - Introduction to python

1. **Data Structures**
   a. Lists
   b. Sets
   c. Numpy Arrays
   d. Dictionary

Data structures

Data Structures – Lists

Lists are used to store multiple items in a single variable. **Lists** are created using square brackets. List items are indexed, the first item has index **[0]**, the second item has index **[1]** etc.

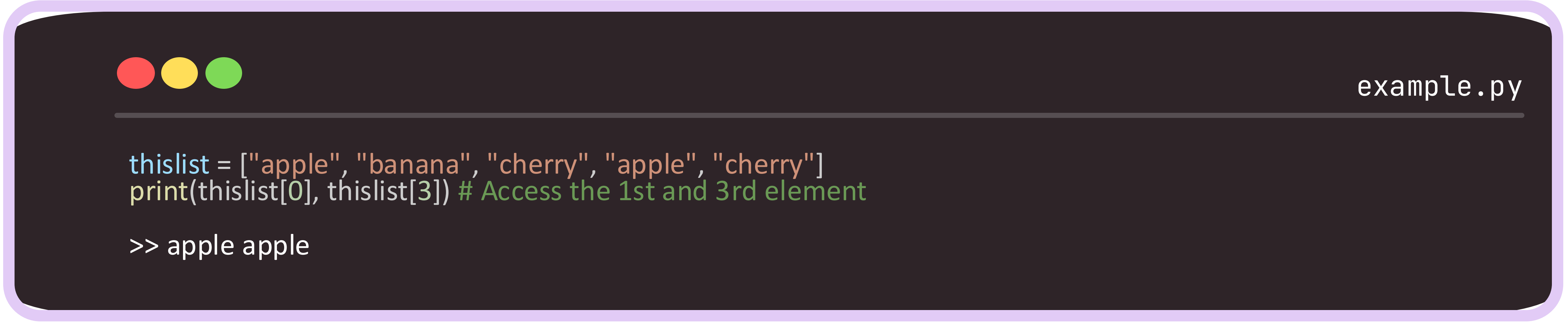


```python
thislist = ["apple", "banana", "cherry"]
print(thislist[0]))# Access the first element of the list

>> apple
```

The list items are:

- **Ordered:** it means that the items have a defined order, and that order will not change. If you add new items to a list, the new items will be placed at the end of the list
- **Changeable:** we can change, add, and remove items in a list after it has been created
- **Allow duplicate values**: Since lists are indexed, lists can have items with the same value



```python
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist[0], thislist[3]) # Access the 1st and 3rd element

>> apple apple
```

Data Structures – Lists

To determine how many items a list has, use the **len()** function.

```
example.py

thislist = ["apple", "banana", "cherry"]
print(len(thislist))

>> 3
```

List items can be of any **data type.**

```
example.py

list1 = ["apple", "banana", "cherry"] # List of strings
list2 = [1, 5, 7, 9, 3] # List of ints
list3 = [True, False, False] # List of Booleans
list1 = ["abc", 34, True, 40, "male"] # A list with strings, integers and boolean
```

Data Structures – Lists

To **change the value** of a specific item, refer to the index number.

```
example.py

thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)

>> ['apple', 'blackcurrant', 'cherry']
```

To add an item to the end of the list, use the **append()** method.

```
thislist = ["apple", "banana", "cherry"]
thislist.append('orange')
print(thislist)

>> ['apple', 'banana', 'cherry', 'orange']
```

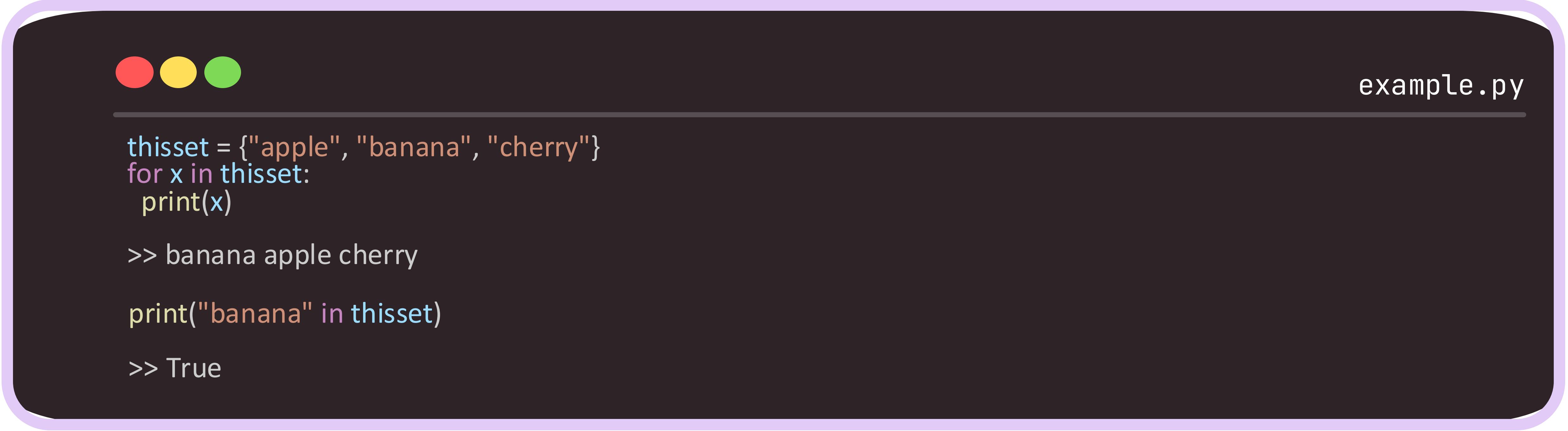
The **remove()** method removes the specified item.

```
thislist = ["apple", "banana", "cherry"]
thislist.remove('banana')
print(thislist)

>> ['apple', 'cherry']
```

Data Structures – Sets

Sets are used to store multiple items in a single variable. Sets are written with **curly brackets** and you **cannot access items in a set by referring to an index** or a key. But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.



example.py

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)

>> banana apple cherry

print("banana" in thisset)

>> True
```

Sets are:

- **Unordered:** means that the items in a set do not have a defined order. Set items can appear in a different order every time you use them and cannot be referred to by index or key
- **Unchangeable:** we cannot change the items after the set has been created
- **Do not allow duplicate values**: Sets cannot have two items with the same value

Data Structures – Sets

To determine how many items a set has, use the **len()** function.

example.py

```python
thisset = {"apple", "banana", "cherry"}
print(len(thisset))

>> 3
```

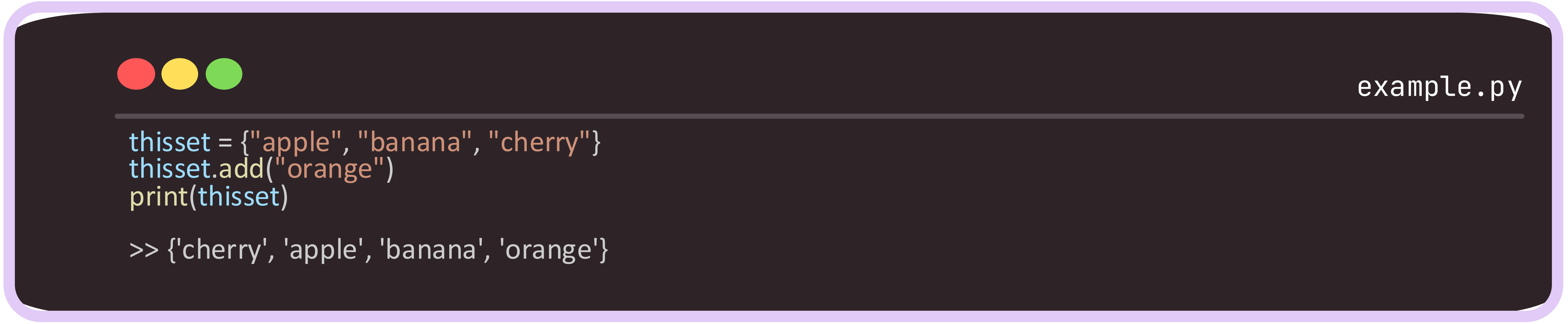
Set items can be of any **data type.**

example.py

```python
set1 = {"apple", "banana", "cherry"} # Set of strings
set2 = {1, 5, 7, 9, 3} # Set of ints
set3 = {True, False, False} # Set of Booleans
set1 = {"abc", 34, True, 40, "male"} # A set with strings, integers and boolean
```

Data Structures – Sets

Once a set is created, you cannot change its items, but you can add new items. To add an item to a set, use the **add()** method.

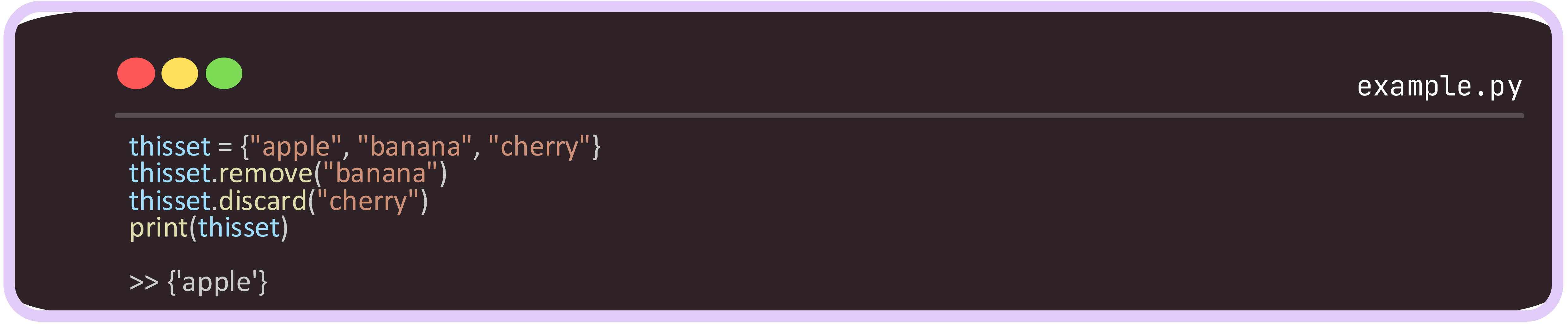


example.py

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)

>> {'cherry', 'apple', 'banana', 'orange'}
```

To remove an item in a set, use the **remove()**, or the **discard()** method.



example.py

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
thisset.discard("cherry")
print(thisset)

>> {'apple'}
```

Challenge: Why does Python have the **remove()** and **discard()** method?

Data Structures – Arrays

Python does not have built-in support for Arrays. Arrays are very similar to lists and Python has built-in support for lists. However, they are slow to process.

NumPy is a Python library, used for working with arrays. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

Installation of Numpy

terminal

```
pip install numpy
```

Once NumPy is installed, import it in your applications by adding the import keyword. NumPy is usually imported under the **np alias**. Create an alias with the as keyword while importing.

example.py

```
import numpy

# or

import numpy as np
```

Data Structures – Arrays

Create a NumPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called ndarray. We can create a NumPy ndarray object by using the **array()** function.

example.py

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))

>> [1 2 3 4 5]
>> <class 'numpy.ndarray'>
```

Access Array Elements

Array indexing is the same as accessing an array element. You can access an array element by referring to its index number. The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

example.py

```
print(arr[0])
print(arr[1:4])

>> 1
>> [2 3 4]
```

Data Structures – Arrays

To determine how many elements an array has, we use the **shape** function.

To **change the value** of a specific element, refer to the index number.

To add an item, use the **append()** method.

To remove an item, use the **delete()** method and pass the index of that element.

example.py

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr.shape)

>> (5,)
```

```
arr[0] = 2 # changing the first element
print(arr)

>> [2 2 3 4 5]
```

```
arr= np.append(arr, 3) # adding the element 3 to the array arr
print(arr)

>> [2 2 3 4 5 3]
```

```
arr= np.delete(arr, 0) # removing the first element from the array arr
print(arr)

>> [2 3 4 5 3]
```

Data Structures – Dictionaries

Dictionaries are used to store data values in **key:value pairs**. Dictionaries are written with **curly brackets and** have keys and values. You can access the items of a dictionary by referring to its key name, inside square brackets. **You can access the items of a dictionary by referring to its key name, inside square brackets.**

example.py

```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])

>> Ford
```

Dictionaries are:

- **Unordered:** means that the items in a set do not have a defined order, and thar order will not change. Dictionaries items cannot be referred to by index
- **Changeable:** we can change, add or remove items after the dictionary has been created
- **Do not allow duplicate keys**: Dictionaries cannot have two items with the same key

Data Structures – Dictionaries

To determine how many items a dictionary has, use the **len()** function.

example.py

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(len(thisdict))

>> 3
```

The dictionary items can be of any **data type.**

example.py

```
thisdict = {
"brand": "Ford",
"model": "Mustang",
"year": 1964,
"colors": ["red", "white", "blue"]
}
```

Data Structures – Dictionaries

To **change the value** of a specific item, by referring to its key name.

example.py

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2025
print(thisdict)

>> {'brand': 'Ford', 'model': 'Mustang', 'year': 2025}
```

Adding an item to the dictionary is done by using a new index key and assigning a value to it.

example.py

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict[”color"] = ["red", "white"]
print(thisdict)

>> {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'colors': ['red', 'white']}
```

Data Structures – Dictionaries

The **pop()** method removes the item with the specified key name.

```
example.py
```

```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)

>> {'brand': 'Ford', 'year': 2025}
```

The **popitem()** method removes the last inserted item.

```
example.py
```

```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["last"] = "of us"
thisdict.popitem()
print(thisdict)

>> {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

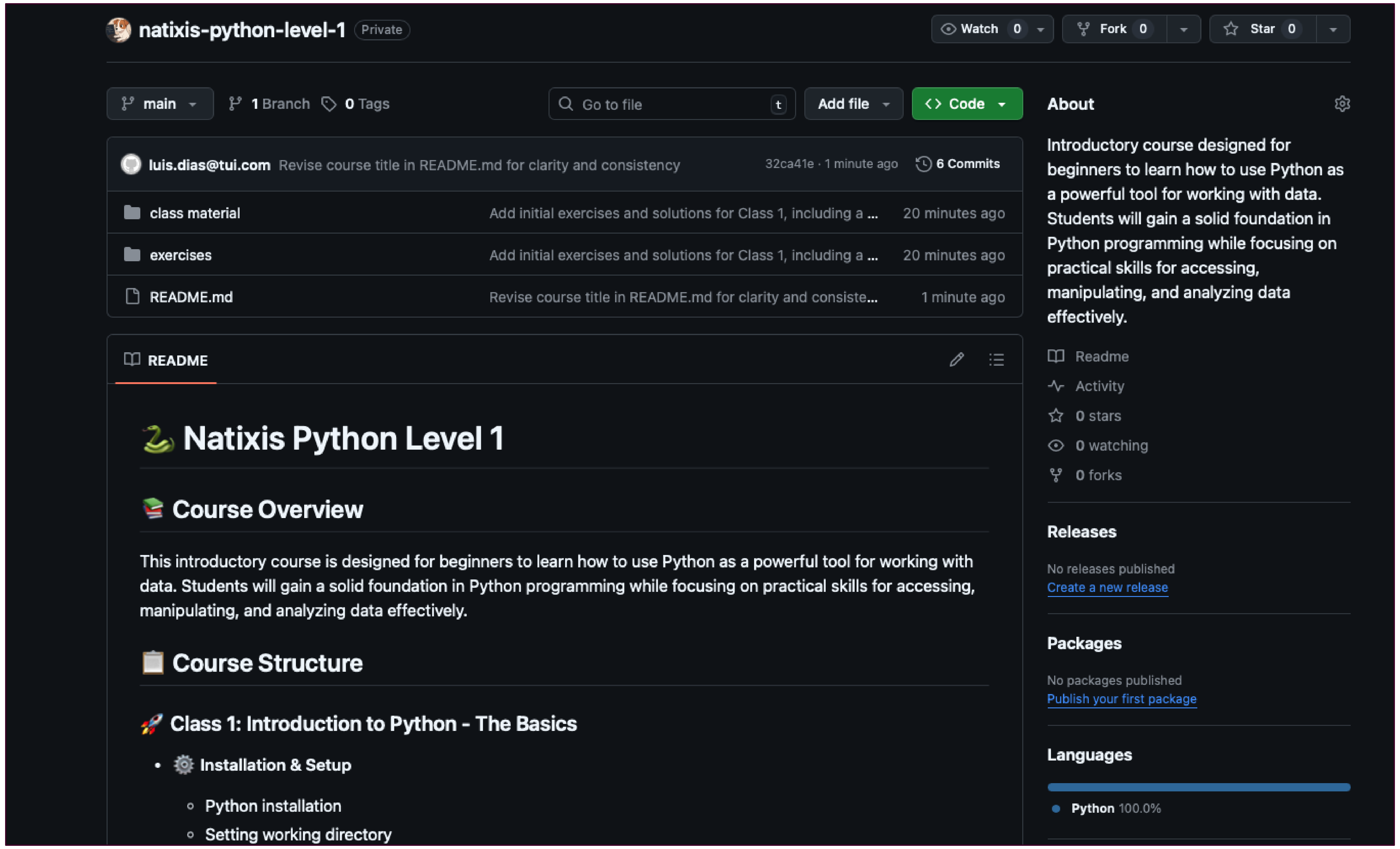
Class Wrap-up

What have we learned today?

- Learned about Python **data structures**, namely List, Dictionary, Set and Numpy array
- Learned what is the **purpose** of each **data structure**
- Learned to **create, read and write** to data structures
- Learned about **optimized data structures** made available through Python libraries

PRACTICE
PRACTICE
PRACTICE

Exercises for today



luis.dias@tui.com Revise course title in README.md for clarity and consistency 32ca41e · 1 minute ago 6 Commits

class material	Add initial exercises and solutions for Class 1, including a ...	20 minutes ago
exercises	Add initial exercises and solutions for Class 1, including a ...	20 minutes ago
README.md	Revise course title in README.md for clarity and consiste...	1 minute ago

🐍 Natixis Python Level 1

📚 Course Overview

This introductory course is designed for beginners to learn how to use Python as a powerful tool for working with data. Students will gain a solid foundation in Python programming while focusing on practical skills for accessing, manipulating, and analyzing data effectively.

📋 Course Structure

🚀 Class 1: Introduction to Python - The Basics

- ⚙️ **Installation & Setup**
  - Python installation
  - Setting working directory

About

Introductory course designed for beginners to learn how to use Python as a powerful tool for working with data. Students will gain a solid foundation in Python programming while focusing on practical skills for accessing, manipulating, and analyzing data effectively.

Readme
Activity
0 stars
0 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

Python 100.0%

Link to exercises: https://github.com/diaxz12/natixis-python-level-1/blob/main/exercises/Class2_exercises.py

THANK YOU 😊

Questions?

Fábio Neves fnssm26@gmail.com

Luis Dias luis.f.s.m.dias@gmail.com