Class 3:
Python Data Types 2/2

Python for Data Analysts: Method & Tools

Felipe Dominguez - Adjunct Professor - Jan 17, 2023

Python Data Types

- There are **7 fundamental** data types in Python.

- Everything developed in Python **is built on these data types**.

- It is **crucial to have a good understanding** of how to work with each of these data types.

Data Types	Description
Numeric	Holds numeric values (int, float, complex)
String	Sequence of characters wrapped by "" or '' that can be read as text
Boolean	Two constant values that represent truth (True and False)
Sequence Types	Store multiple values in an organized and efficient way (Lists, tuples, and range)
Binary	Allows to manipulate binary data in Python (bytes, bytearray, and memoryview)
Set	Unordered collection of distinct hashable objects
Mapping	A mapping object maps hashable values to arbitrary objects (dictionaries)

Table 1. Python fundamental data types.

Today’s Class

- Sequence Data types
  - Range
  - Numpy Arrays
- Dictionaries
- Game Theory: The Python Equilibrium

Sequence Data types: Ranges

Range - Syntax

Range: **Immutable** sequence of numbers. It creates a **list-type** of numbers within a specified range.

- It only include the lower limit (**Upper limit excluded**)

Declare a new range

```
my_range = range(lower_limit, upper_limit, step =)
```

Transform an string to an integer

```
my_range = range(0, 100)
print(my_range)
print(type(my_range))
```

```
>>> range(0, 100)
>>> <class 'range'>
```

Range - Accessing

Range can be accessed similar to tuples and lists.

- If you slice a range, Python will return a **new range** data type with the lower and upper limits defined by the slice limits.

Access first range element

```
my_range = range(0, 100)
print(my_range[0])
```

```
>>> 0
```

Slice a range

```
my_range_2 = my_range[0:50]
print(my_range_2)
print(my_range_2[-1])
```

```
>>> range(0, 50)
>>> 49
```

Range - Iterate

Ranges are mostly used to iterate a specific number of times.

Range iteration

```
for i in range(0, 100):
    print(i)
```

```
>>> 1
>>> 2
>>> ...
>>> 99
```

Let’s practice range

Sequence Data types: Numpy Arrays

Numpy Arrays - What is it?

Numpy: **Fundamental** python library for scientific programming.

- It contains **multidimensional** objects called numpy arrays, ndarrays, or n-dimensional arrays.
- Can only contain **one type** of data

numpy array syntax

```
import numpy as np

np_array = np.array([[elements]])
```

Create numpy array

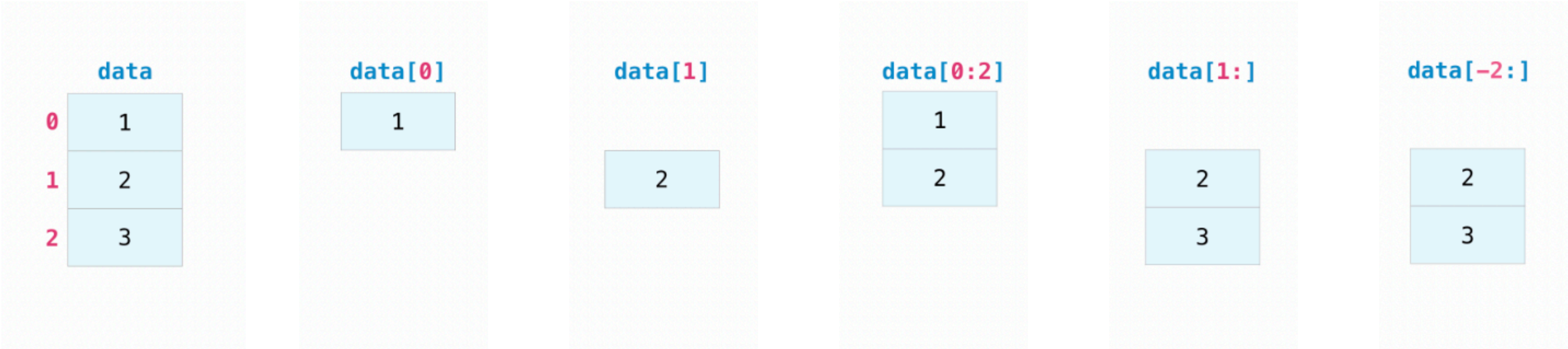
```
np_array = np.array([ [1, 2], [3,4] ])

print(np_array)

print(type(np_array))
```

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

Numpy Arrays - Accessing



Numpy Arrays - Accessing

np_array

$$\begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 1 & 2 \\ 1 & 3 & 4 \\ 2 & 5 & 6 \end{array}$$

np_array[0]

$$\begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 1 & 2 \end{array}$$

np_array[1]

$$\begin{array}{c|cc} & 0 & 1 \\ \hline 1 & 3 & 4 \end{array}$$

np_array[0, 1]

$$\begin{array}{c|c} & 1 \\ \hline 0 & 2 \\ 1 & \end{array}$$

np_array[0:3, :1]

$$\begin{array}{c|c} & 0 \\ \hline 0 & 1 \\ 1 & 3 \\ 2 & 5 \end{array}$$

Numpy Arrays - Advantages

- Pandas library is built-on Numpy.
- Provides easy use of algebraic operations (E.g.,cross product).
- Faster and more compact than lists.
- Similar access than lists. Additionally, it allows **conditions.**

Numpy Arrays - Conditions

Numpy: You can access numpy arrays similar to lists and tuples. Additionally, you can slice based on **conditions.**

Slice a ndarray syntax

```
import numpy as np

np_array = np.array([[elements]])

new_array = np_array[ condition ]
```

Slice a ndarray example

```
condition = np_array > 2

print(condition)
```

```
>>> [[False False]
     [ True  True]]
```

Numpy Arrays - Conditions

Slice a ndarray example

```
new_array = np_array[np_array > 2]

print(new_array)
```

```
>>> [3 4]
```

Let’s practice numpy arrays

Dictionaries

Dictionaries - What are they?

Dictionaries: Is an **unordered** collection of data which is changeable and do not allow duplicates.

- Allow you to store data in **key:value** pairs.

Dictionaries syntax

```
my_dict = { key1 : value1,
            key2: value2,
            …,
            keyn: valuen}
```

Create a dictionary

```
my_dict = {“brand”: “Mazda”,
           “model”: “Mazda6”,
           “year”: “2020”}
print(my_dict)
print(type(my_dict))
```

```
>>> {'brand': 'Mazda', 'model': 'Mazda6', 'year': 2020}
>>> <class 'dict'>
```

Dictionaries - Advantages

- Fast look-up: Map between key-value pairs.
- Flexible data types: Can store any data type.
- Dictionaries are mutable.
- Efficient in terms of memory usage.
- Easy to create and manipulate: Popular choice for storing data.

Dictionaries - Accessing

- Access values by parsing its key
- Access values by parsing its key and its index.

Slicing dictionary key

```
my_dict = {“brand”: “Mazda”,
          “model”: [“Mazda6”, “Mazda3”],
          “year”: “2020”}
print(my_dict[“model”])
```

```
>>>[‘Mazda6’, ‘Mazda 3’]
```

Slicing dictionary key & index

```
my_dict = {“brand”: “Mazda”,
          “model”: [“Mazda6”, “Mazda3”],
          “year”: “2020”}
print(my_dict[“model”][1])
```

```
>>>‘Mazda 3’
```

Dictionaries - Accessing

- Access all keys, values, and key:values pair of a dictionary

Accessing all keys & values

```
print(my_dict.keys())

print(my_dict.values())
```

```
>>> dict_keys(['brand', 'model', ‘year'])

>>> dict_values(['Mazda', ['Mazda6', 'Mazda3'], '2020'])
```

Accessing all key:value pairs.

```
print(my_dict.items())
```

```
>>>dict_items([('brand', 'Mazda'), ('model', ['Mazda6', 
'Mazda3']), ('year', '2020')])
```

Dictionaries - Add new values

Add New keys and New values

```
my_dict[“color”] = “blue”
print(my_dict)
```

```
>>>{'brand': 'Mazda', 'model': [‘Mazda6’, ’Mazda3’],
'year': 2020, 'color': 'blue'}
```

Add New values into an existing key

```
my_dict[“model”] = “CX-5”
print(my_dict)
```

```
>>>{'brand': 'Mazda', 'model': ‘CX-5’, 'year': 2020,
'color': 'blue'}
```

Let’s practice Dictionaries

Game Theory: The Python Equilibrium

Prisoner's Dilemma

- Paradox in decision analysis. It represents that two individuals acting by their own self-interest won't produce the optimal outcome.
- Several examples of it in the real-world.

\ Individual 1 Individual 2. \	**Cooperate** **(Stay Silent)**	**Not cooperate** **(Betray)**
Cooperate **(Stay Silent)**	Both 1 year of prison	Ind 1. Free Ind 2. 10 years
Not cooperate **(Betray)**	Ind 1. 10 years Ind 2. Free	Both 25

Prisoner's Dilemma

- Let's see a real world example
  - Company A sells product A and Company B sells product B.
  - Product A and B are substitutes.

\ Company A Company B \	**Cooperate**	**Not cooperate**
Cooperate	Profit A & B: $100	Profit A: $200 Profit B: $0
Not cooperate	Profit A: $0 Profit B: $200	Profit A & B: $50

Prisoner’s Dilemma

- Other real world examples:
  - Negotiating
  - Cartels organization (OPEC)
  - Pricing in a marketplace
  - Marketing expenses

Assignment Example