Class 2:
Python Data Types 1/2
Python for Data Analysts: Method & Tools

Felipe Dominguez - Adjunct Professor - Jan 11, 2023

Introduce yourself!

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

- Numbers & Strings
- Booleans
- Sets
- Sequence Data types
  - Lists
  - Tuples

Numbers & Strings

Numbers - Integer

Integer: Whole number, positive or negative, without decimals, of unlimited length.

Transform an object to an integer

```
integer_number = int(2.5)
print(integer_number)
print(type(integer_number))
```

```
>>> 2
>>> <class 'int'>
```

Transform an string to an integer

```
integer_number = int("seven")
print(integer_number)
print(type(integer_number))
```

```
>>> ValueError: invalid literal for int() with base
10: 'seven'
```

Numbers - Float

Float: Number, positive or negative, containing one or more decimals.

Transform an object to float

```
float_number = float(2.5)
print(float_number)
print(type(float_number))
```

```
>>> 2.5
>>> <class ‘float'>
```

Transform an string to an integer

```
float_number = float(“seven”)
print(float_number)
print(type(float_number))
```

```
>>> ValueError: could not convert string to float: 'seven'
```

Numbers - Complex

Complex: Number of the form a + bj, where a and b are real numbers.

Transform an object to an integer

```
complex_number = complex(2, 3)
print(complex_number)
print(type(complex_number))
```

```
>>> (2+3j)
>>> <class ‘complex'>
```

Transform an string to an integer

```
complex_number = complex(“two”, “three”)
print(complex_number)
print(type(complex_number))
```

```
>>> TypeError: complex() can't take second arg if first is a string
```

Numbers - Operations

Operations:

- Most common mathematical operations can be performed as expected (+, -, *, /, etc)
- For more specific operations you’ll need to import the package math (logarithm, trigonometry, combinatory, euclidean distance, and more.)

Numbers - Shortcuts

Shortcuts: Python provide mathematical abbreviations for repetitive tasks.

Operand	Description
+=	addition and overwrite
-=	subtraction and overwrite
*=	multiplication and overwrite
/=	division and overwrite
%=	exponentiation and overwrite
**=	Unordered collection of distinct hashable objects

Table 2. Mathematical abbreviations.

Addition and overwrite (+=)

```
x = 10
x+=10      # instead of x = x + 10
print(x)
```

```
>>> 20
```

Numbers - Randomness

Randomness: refers to the lack of order, pattern, or purpose.

-1/3, +1/5, −1/7, +1/9, −1/11, +1/13, −1/15,...

What is the next number?

5, 2, 3, 5, 9, 8, 7, 7, 5, 5, ...

What is the next number?

Numbers - Randomness

randint(): Return random integer in range [a, b], including both start and end points.

Select a random number between 0 and 10

```
random_number = random.randint(0, 10)
print(random_number)
```

```
>>> 5
```

Set a seed number to 1

```
random.seed(1)
random_number = random.randint(0, 10)
print(random_number)
```

```
>>> 2
```

Strings - Syntax

Syntax - Python understand strings as any character wrapped by any of the following options:

- “This is a string 1” — Single line
- ‘This is a string 2’ — Single line
- “””This is a string 3””” — Multiple lines.

Numbers & Strings - Operations

- Different data types have different behaviors in Python.
- Mathematical operations work as expected when applied to numeric values.
- The only two mathematical operations that works with strings is addition, +, (concatenation) and multiplication (repetition), *.

Concatenate Text (use +)

```
var_1 = “Hello”

var_2 = “World!”

print(var_1 + var_2)
```

```
>>> “Hello World!”
```

Operates between str and int

```
print(2 + “2”)
```

```
>>> TypeError: can only concatenate str (not "int") to str
```

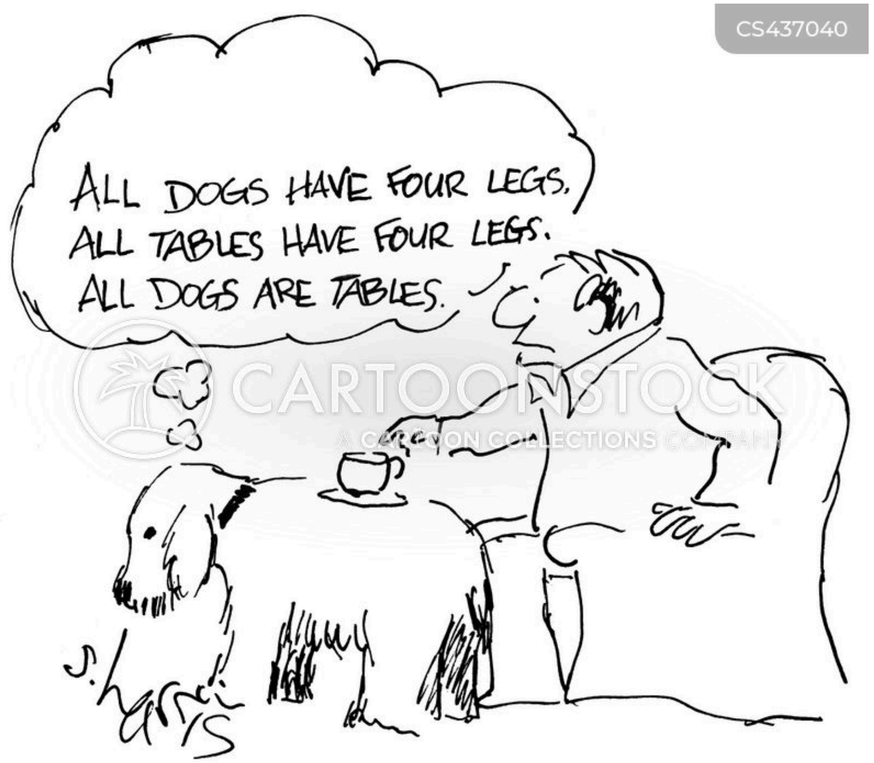
Let’s practice Numbers & Strings

Booleans

Booleans

Logic: The study of correct reasoning.

- There are two fundamental Truth values: **True (1, “ ”)** and **False (0, “”)**



Booleans

Truth Terms

Term	Example
==	2 == 2 —> **True**
!=	2!=2 —> **False**
not	not 2==3 —> **True**
>=	5 >= 7 —> **False**
<=	5 <= 7 —> **True**
and	2 == 2 and 1 == 1 —> **True**
or	2 == 2 or 1 == 12 —> **True**

Table 3. Truth Terms

Booleans - Example

Transform boolean into int

```
int_true = int(True)

print(int_true)
```

```
>>>1
```

Transform a string into boolean

```
str_false = bool(“”)

print(str_false)
```

```
>>> False
```

Booleans - Example

Boolean expressions

```
x = 10

y = 20

z = 30

print(x != y)

print(y == z)

print(x + y == z)
```

```
>>> True

>>> False

>>> True
```

Let’s practice Booleans

Sets

Sets:

- Store various **unordered** items.
- Store only **unique** values (Do not allow duplicates)
- Sets are **mutable.**
- Use cases: remove duplicate from sequences or computing mathematical operations like intersections, joins, etc.

Sets - Declare

Syntax

Declare a new Set

my_set = {item_0, item_1, ..., item_n}

OR

my_set = set({item_0, item_1, ..., item_n})

Declare a new Set

```
my_set = {“a”, 2, True}
my_set.add(9)
print(my_set)
```

```
>>> {“a”, 2, True, 9}
```

Sets - frozenset

Frozensets

- Similar characteristics as Sets.
- **Immutable** (It cannot be altered after created).
- Usually, used as keys in dictionary or as elements of another set.

Sets - frozenset

Syntax

Declare a new frozenset

```
my_fset = frozenset({item_1, item_2, ..., item_n})
```

Declare a new frozenset

```
my_fset = frozenset({“a”, 2, True})
my_fset.add(9)
```

```
>>>AttributeError: 'frozenset' object has no attribute 'add'
```

Let’s practice Sets

Sequence Data types: Lists

Lists - What are they?

- **Mutable and versatile** data type (Similar to arrays in other languages).
- Allows to store and hold **different data types** in the same list.

Syntax

```
my_list = [item_0, item_1, ..., item_n]
```

Lists - What are they?

- **Lists operates different** compared to other data types

Compare int vs lists

```
list_1 = [5]

number_1 = 5

print(list_1*5)

print(number_1*5)
```

```
>>> [5, 5, 5, 5, 5]

>>>25
```

Lists - What are they?

my_list = [item_0, item_1, …, item_n]

	A	B	C	D	E
1	Company	2015 Revenues (SMM)	Return on Assets	Return on Sales	Return on Equity
2	Wal-Mart Stores	$ 482,130	7.4%	3.0%	18.2%
3	Exxon Mobil	$ 246,204	4.8%	6.6%	9.5%
4	Apple	$ 233,715	18.4%	22.8%	44.7%
5	Berkshire Hathaway	$ 210,821	4.4%	11.4%	9.4%
6	McKesson	$ 181,241	2.7%	0.8%	18.4%
7	Unitedhealth Group	$ 157,107	5.2%	3.7%	17.2%
8	CVS Health	$ 153,290	5.6%	3.4%	14.1%
9	General Motors	$ 152,356	5.0%	6.4%	24.3%
10	Ford Motor	$ 149,558	3.3%	4.9%	25.7%

list_1 = [“Wal-Mart Stores”, 482130, 0.074, 0.03, 0.182]

Index 0 Index 1 ●●● Index 4

Lists - What are they?

- Each row can be represented as an individual list

my_list = [item_0, item_1, …, item_n]

	A	B	C	D	E
1	**Company**	**2015 Revenues (SMM)**	**Return on Assets**	**Return on Sales**	**Return on Equity**
2	Wal-Mart Stores	$ 482,130	7.4%	3.0%	18.2%
3	Exxon Mobil	$ 246,204	4.8%	6.6%	9.5%
4	Apple	$ 233,715	18.4%	22.8%	44.7%
5	Berkshire Hathaway	$ 210,821	4.4%	11.4%	9.4%
6	McKesson	$ 181,241	2.7%	0.8%	18.4%
7	Unitedhealth Group	$ 157,107	5.2%	3.7%	17.2%
8	CVS Health	$ 153,290	5.6%	3.4%	14.1%
9	General Motors	$ 152,356	5.0%	6.4%	24.3%
10	Ford Motor	$ 149,558	3.3%	4.9%	25.7%

list_1 = [“Wal-Mart Stores”, 482130, 0.074, 0.03, 0.182]

list_2 = [“Exxon Mobil”, 246204, 0.048, 0.066, 0.095]

Lists - Accessing

- There are three ways to access a list:
  - Indexing
  - Slicing
  - Subsetting

Lists - Accessing

Indexing

- You can access a list elements by calling the **index** (position) of the element.
  - Remember: First element is stored at position 0.

print element 0 of list_1 (*index forward*)

```
print(list_1[0])
```

>>> “Wal-Mart Stores”

print last element of list_1 (*index backward*)

```
print(list_1[-1])
```

>>> 0.182

Lists - Accessing

Slicing

- Returns a list with the elements between lower and upper limits selected. **Upper limit is not included.**

print all elements of list_2

```
print(list_2[:])
```

```
>>> [“Exxon Mobil”, 246204, 0.048, 0.066, 0.095]
```

print elements 2 and 3 of list_2

```
print(list_2[1:3])
```

```
>>> [246204, 0.048]
```

print the last two elements

```
print(list_2[-2:])
```

```
>>> [0.066, 0.095]
```

Lists - Fundamental Functions

- append()
- extend()
- insert()
- remove()
- pop()
- reverse()

Lists - Fundamental Functions

append()

- Most common method to add new element to a existing list. It **appends** a **single** new element at the end of the list.

Add the year at the end of list_1

```
list_1.append(2015)
print(list_1)
```

```
>>> [“Wal-Mart Stores”, 482130, 0.074, 0.03, 0.182, 2015]
```

Lists - Fundamental Functions

append()

- What happened if you append one list to another?

Append car_list2 to car_list1

```
car_list1 = [“Mazda”, “BMW”, “Tesla”]

car_list2 = [“Jeep”, “Audi”]

car_list1.append(car_list2)

print(car_list1)
```

```
>>> [“Mazda”, “BMW”, “Tesla”, [“Jeep”, “Audi”]]
```

Lists - Fundamental Functions

extend()

- Add elements from one list to another one. It **extends** each element of the second list at the end of the original one.

Extend car_list2 to car_list1

```
car_list1 = [“Mazda”, “BMW”, “Tesla”]

car_list2 = [“Jeep”, “Audi”]

car_list1.extend(car_list2)

print(car_list1)
```

```
>>> [“Mazda”, “BMW”, “Tesla”, “Jeep”, “Audi”]
```

Lists - Operations

insert()

- Insert a single element into a specific index (position) of a list.
- list.insert(index, object)

Add the year at the second position of list_1

```
list_1.insert(1, 2015)
print(list_1)
```

```
>>> [“Wal-Mart Stores”, 2015, 482130, 0.074, 0.03, 0.182, 2015]
```

Lists - Operations

remove()

- Remove a specific object from a list. It deletes the first occurrence of that object. This method do not return any value.

Remove the revenue of list_1

```
list_1.remove(482130)
print(list_1)
```

```
>>> [“Wal-Mart Stores”, 0.074, 0.03, 0.182, 2015]
```

Lists - Operations

remove()

- Remove a specific object from a list. It deletes the first occurrence of that object. This method do not return any value.

Remove the Walmart from list_stores

```
list_stores = [“Walmart”, “Target”, “Costco”, “Walmart”]
list_stores.remove(“Walmart”)
print(list_stores)
```

```
>>> [“Target”, “Costco”, “Walmart”]
```

Lists - Operations

pop()

- Remove a specific object from a list based on its position. This method returns the element removed.

Remove the Walmart from list_stores

```
list_stores = [“Walmart”, “Target”, “Costco”, “Walmart”]

object_removed = list_stores.pop(3)

print(object_removed)

print(list_stores)
```

```
>>> “Walmart”

>>> [“Walmart”, “Target”, “Costco”]
```

Lists - Operations

reverse()

- This method reverse the order of the items within a list.
- Usually, used when you need to add multiple values at the beginning of a list.

Lists of lists

Syntax

- Let’s go back to the companies dataset.

	A	B	C	D	E
1	**Company**	**2015 Revenues (SMM)**	**Return on Assets**	**Return on Sales**	**Return on Equity**
2	Wal-Mart Stores	$ 482,130	7.4%	3.0%	18.2%
3	Exxon Mobil	$ 246,204	4.8%	6.6%	9.5%
4	Apple	$ 233,715	18.4%	22.8%	44.7%
5	Berkshire Hathaway	$ 210,821	4.4%	11.4%	9.4%
6	McKesson	$ 181,241	2.7%	0.8%	18.4%
7	Unitedhealth Group	$ 157,107	5.2%	3.7%	17.2%
8	CVS Health	$ 153,290	5.6%	3.4%	14.1%
9	General Motors	$ 152,356	5.0%	6.4%	24.3%
10	Ford Motor	$ 149,558	3.3%	4.9%	25.7%

Lists of lists

Syntax

- So far, we created two lists (list_1 & list_2) for the first two rows. You'll need to create *n* lists variables to store n rows.
- Creating a list of lists allows you to store multiple lists in one list object.

	A	B	C	D	E
1	**Company**	**2015 Revenues (SMM)**	**Return on Assets**	**Return on Sales**	**Return on Equity**
2	Wal-Mart Stores	$ 482,130	7.4%	3.0%	18.2%
3	Exxon Mobil	$ 246,204	4.8%	6.6%	9.5%
4	Apple	$ 233,715	18.4%	22.8%	44.7%
5	Berkshire Hathaway	$ 210,821	4.4%	11.4%	9.4%
6	McKesson	$ 181,241	2.7%	0.8%	18.4%
7	Unitedhealth Group	$ 157,107	5.2%	3.7%	17.2%
8	CVS Health	$ 153,290	5.6%	3.4%	14.1%
9	General Motors	$ 152,356	5.0%	6.4%	24.3%
10	Ford Motor	$ 149,558	3.3%	4.9%	25.7%

Lists of lists

Syntax

- Let’s store the companies dataset into a list of list structure.

```
full_list = [
            [“Wal-Mart Stores”, 0.074, 0.03, 0.182, 2015],
            [“Exxon Mobil”, 246204, 0.048, 0.066, 0.095],
            ...,
           [“Ford Motor”, 149558, 0.039, 0.049, 0.257]
]
```

Lists of lists — Access

Index first list element of **full_list**

```
print(full_list[0])
```

```
>>> [“Wal-Mart Stores”, 0.074, 0.03, 0.182, 2015]
```

Index first element of first list of **full_list**

```
print(full_list[0][0])
```

```
>>> “Wal-Mart Stores”
```

Lists of lists — Access

Slice first 3 list objects of **full_list**

```
print(full_list[0:3])
```

```
>>> [ [“Wal-Mart Stores”, 0.074, 0.03, 0.182, 2015],
    [“Exxon Mobil”, 246204, 0.048, 0.066, 0.095],
    [“Apple”, 233715, 0.184, 0.228, 0.447] ]
```

Slice first two objects of first list of **full_list**

```
print(full_list[0][0:2])
```

```
>>> ['Wal-Mart Stores', 0.074]
```

Let’s practice Lists

Sequence Data types: Tuples

Tuples - What are they?

- Tuples are similar to lists
- Tuples are **immutable**, meaning that their values cannot be modified or changed once they have been created

Syntax

```
my_tuple = (item_0, item_1, ..., item_n)
```

Tuples - What are they?

Create a new tuple object

```
my_tuple = (1, 2, 3)
print(type(my_tuple))
print(my_tuple[0])
```

```
>>> <class 'tuple'>
>>> 1
```

Try replace an element to a tuple

```
my_tuple = (1, 2, 3)
my_tuple[1] = 0
```

```
>>> TypeError: 'tuple' object does not support item assignment
```

Let’s practice Tuples