OBJECT ORIENTED PROGRAMMING IN PYTHON

Deep-Dive on Classes

Vicki Boykis
Senior Data Scientist

Working with DataFrames

```
df = pd.read_csv('mtcars.csv')
df
```

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4

Introducing the DataShell

DataFrame

DataShell

Full Class

```
class DataShell:

    #constructor
    def __init__(self, filename):
        self.filename = filename

    def create_datashell(self):
        self.array = np.genfromtxt(self.filename, delimiter=',', dtype=None)
        return self.array

    def rename_column(self, old_colname, new_colname):
        for index, value in enumerate(self.array[0]):
            if value == old_colname.encode('UTF-8'):
                self.array[0][index] = new_colname
        return self.array

    def show_shell(self):
        print(self.array)

    def five_figure_summary(self, col_pos):
        statistics =  stats.describe(self.array[1:,col_pos].astype(np.float))
        return f"Five-figure stats of column {col_position}: {statistics}"
```

Creating a New Class

```
class DataShell:
    pass
```

Parts of a Class in Detail

```
class DataShell:

    # constructor
    def __init__(self, filename):
        self.filename = filename # attribute

    # method
    def create_datashell(self):
        self.array = np.genfromtxt(self.filename, delimiter=',', dtype=None)
        return self.array

    # method
    def rename_column(self, old_colname, new_colname):

        for index, value in enumerate(self.array[0]):
            if value == old_colname.encode('UTF-8'):
                self.array[0][index] = new_colname

        return self.array
```

How to Call the Class

```
class DataShell():
 # some methods and attributes here
```

```
ourDataShell = DataShell('mtcars.csv')
```

```
>>>ourDataShell
<__main__.DataShell at 0x7f58df61de80>
```

OBJECT ORIENTED PROGRAMMING IN PYTHON

Let's practice!

OBJECT ORIENTED PROGRAMMING IN PYTHON

Initializing a class

Vicki Boykis
Senior Data Scientist

Understanding Constructors with Init

Empty Constructor

```
class Dinosaur:

    def __init__(self):
        pass
```

Constructor with Attributes

```
class Dinosaur:

    def __init__(self):
        self.tail = 'Yes'
```

Init and Our DataShell

```
# Modeled on Pandas read_csv
pandas.read_csv('mtcars.csv')
```

Creating the DataShell with a Constructor

```
class DataShell:

    def __init__(self, filename):
        self.filename = filename
```

Understanding Self

```
class DataShell:

    def __init__(self, filename):
        self.filename = filename
```

Initializing the Car DataShell

```
CarDataShell = DataShell('mtcars.csv')

    def __init__(CarDataShell, 'mtcars.csv'):
        self.filename = filename
```

Initializing the ForestFire DataShell

```
ForestFireDataShell = DataShell('forestfires.csv')

    def __init__(ForestFireDataShell, 'forestfires.csv'):
        self.filename = filename
```

Self is not a Python keyword but we use it like one

```
# Printing out the keyword list
>>>import keyword
>>>print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', '
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not',
'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

# Using this as an object referece
def __init__(this, filename):
        this.filename = filename
```

OBJECT ORIENTED PROGRAMMING IN PYTHON

Let's practice!

OBJECT ORIENTED PROGRAMMING IN PYTHON

Class and Instance Variables

Vicki Boykis
Senior Data Scientist

Class Variables

Our Dinosaur Class

```
class Dinosaur():

    eyes = 2

    def __init__(self, teeth):
        self.teeth = teeth
```

Building a Stegosaurus

```
Stegosaurus = Dinosaur(40)

Stegosaurus.teeth() = 40
>>> Stegosaurus.teeth
40
>>>Stegosaurus.eyes
2
```

Instance Variables

Our Dinosaur Class

```
class Dinosaur():
    eyes = 2

    def __init__(self, teeth):
        self.teeth = teeth
```

Building a Triceratops

```
Triceratops = Dinosaur(5)
Triceratops.teeth = 5
>>> Triceratops.teeth
5
>>>Triceratops.eyes
2
```

Passing in parameters to objects

```
class DataFrame(object):

    def __init__(self, filename):
        self.filename = fileanme
```

Results:

```
MyDataShell = DataShell('mtcars.csv')
print(MyDataShell.filename)

'mtcars.csv'
```

OBJECT ORIENTED PROGRAMMING IN PYTHON

Let's practice!

OBJECT ORIENTED PROGRAMMING IN PYTHON

Methods in Classes

Vicki Boykis
Senior Data Scientist

Methods

```
class DataShell:

    def __init__(self, filename):
        self.filename = filename

    def create_datashell(self):
        self.array = np.genfromtxt(self.filename, delimiter=',', dtype=None)
        return self.array

    def rename_column(self, old_colname, new_colname):

        for index, value in enumerate(self.array[0]):
            if value == old_colname.encode('UTF-8'):
                self.array[0][index] = new_colname
        return self.array
```

Initializing Methods in Classes

```
    def create_datashell(self):
        self.array = np.genfromtxt(self.filename, delimiter=',', dtype=None)
        return self.array
```

Methods with other parameters

```
    def rename_column(self, old_colname, new_colname):

        for index, value in enumerate(self.array[0]):
            if value == old_colname.encode('UTF-8'):
                self.array[0][index] = new_colname

        return self.array
```

How to call methods

```
# Calling without passing in a parameters
myDatashell.create_datashell()
```

```
# Calling by passing in a parameter
myDatashell.rename_column('cyl','cylinders')
```

OBJECT ORIENTED PROGRAMMING IN PYTHON

Let's practice!