3d-Object.Oriented.Programming

November 2, 2024

1 Basics of Object Oriented Programming

1.1 Why we care

Everything in python is an object: - strings - ints - lists - DataFrames - Numpy Arrays

We want to cover just enough OOP that you understand certain things about Python syntax.

1.2 Classes and Objects

Classes are like a blueprint. They define a type of object and the methods that can be used on that object.

Objects are specific instances of a class. For example, an actual house. You could create as many houses as you wanted from one blueprint.

For example, DataFrame is a class in Pandas. But when you create a DataFrame, that specific DataFrame is an obeject.

Classes allow us to create functionality that will be available to all objects that we make of that class.

For example, you are able to use head() with any DataFrame to get the first few entries.

1.3 Methods and Attributes

Methods are like functions, but they run on an object.
- For example, when you call head() on a dataframe it's df.head() not head(df).

Attributes are values for variables associated with an object.

[1]:

```python
import pandas as pd
df = pd.DataFrame([[0,2],[5,6],[9,0]])
df
```

[1]:

```
   0  1
0  0  2
1  5  6
2  9  0
```

[2]:

```python
# head() is a method we can call on our dataframe object
df.head(2)
```

[2]:

```
   0  1
0  0  2
1  5  6
```

[3]:

```
# shape is an attribute  - notice how it has no ()
df.shape
```

[3]:

```
(3, 2)
```

[4]:

```
print(df)
print(df.__class__)
print(df.__class__.__name__)
```

```
   0  1
0  0  2
1  5  6
2  9  0
<class 'pandas.core.frame.DataFrame'>
DataFrame
```

1.4 Making your own class

Animal **Class** Attributes: - Sound - What sound does it make? - Name - Color - Species -# of legs - eye color ….

Methods (What can an animal do? Or what can I do to it?): - Talk - Walk - …

Object - Specific instance of the class

Cat -

Sound - mreeow

Name - Head Cat

Color - White & grey

Make an animal class & an object of type Animal

[27]:

```
# Class names - first letter of words are capitalized - no spaces - i.e. pd.
 ↪DataFrame
class Animal:
  """This is the Animal Class"""

  # This is the constructor.  It is called when we make a new Animal.
  # For example, something like
  # head_cat = Animal('Head Cat', 'cat', 'white and grey', 'meow') goes to
  # head_cat = __init__('Head Cat', 'cat', 'white and grey', 'meow')
  def __init__(self, name, species, color, sound):
    # here we set the attributes based on the input from the constructor
    self.name = name
    self.species = species
    self.color = color
    self.sound = sound
    self.steps = 0

  # This is a dunder method - it returns a string as a representation of the object.
  # Creating a string that will be returned when print is used on the Animal
  # We'll make it something pretty and human-readable
  def __repr__(self):
    return(f"{self.__class__.__name__}(name={self.name}, species={self.species}, color={self.color}, sound={self.sound})")

  # This is a method
  def talk(self):  # head_cat.talk()  ->  talk(head_cat)
    """Print the animal's sound"""
    print(self.sound)  # print(head_cat.sound)

  def walk(self, num_steps):
    self.steps = self.steps + num_steps
    return num_steps
```

[28]:

```
our_dog = Animal(name = 'Zeus', species = 'dog', color = 'grey', sound = 'bark')
our_dog
```

[28]:

```
Animal(name=Zeus, species=dog, color=grey, sound=bark)
```

[29]:

```
our_dog.steps
```

[29]:

```
0
```

[30]:

```
our_dog.talk()
```

```
bark
```

[31]:

```
our_dog.walk(5)
```

[31]:

```
5
```

[32]:

```
our_dog.steps
```

[32]:

```
5
```

[33]:

```
#Animal?
help(Animal)
```

```
Help on class Animal in module __main__:

class Animal(builtins.object)
 |  Animal(name, species, color, sound)
 |
 |  This is the Animal Class
 |
 |  Methods defined here:
 |
 |  __init__(self, name, species, color, sound)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  talk(self)
 |      Print the animal's sound
 |
 |  walk(self, num_steps)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

[34]:

```python
# Head Cat is an Object of Type Animal.  Animal is the class.
head_cat = Animal('Head Cat', 'cat', 'white and grey', 'mrrrrr')
head_cat
```

[34]:

```
Animal(name=Head Cat, species=cat, color=white and grey, sound=mrrrrr)
```

[35]:

```python
head_cat.talk()
```

```
mrrrrr
```

[36]:

```python
print(head_cat)
```

```
Animal(name=Head Cat, species=cat, color=white and grey, sound=mrrrrr)
```

[37]:

```python
print(head_cat.species)
```

```
cat
```

[38]:

```python
head_cat.color
```

[38]:

```
'white and grey'
```

```
[39]: trip = Animal('Tripping Hazzard', 'cat', 'grey', 'purrr')
      trip.talk()
      print(f"Trip was a {trip.color} cat")
```

```
purrr
Trip was a grey cat
```

```
[40]: print(trip)
```

```
Animal(name=Tripping Hazzard, species=cat, color=grey, sound=purrr)
```

```
[41]: parrot = Animal("Timothy", "parrot", "blue", "Polly want a cracker!")
```

```
[42]: print(parrot)
```

```
Animal(name=Timothy, species=parrot, color=blue, sound=Polly want a cracker!)
```

```
[43]: parrot.talk()
```

```
Polly want a cracker!
```

```
[44]: parrot.walk(10)
      parrot.steps
```

```
[44]: 10
```

```
[45]: parrot.walk(5)
      parrot.steps
```

```
[45]: 15
```

```
[46]: print(parrot)
      print(parrot.color)
```

```
Animal(name=Timothy, species=parrot, color=blue, sound=Polly want a cracker!)
blue
```

1.5 Making a derived class

1.5.1 Creating a Queue

```
[47]: foo = [ 1, 2, 3]
      foo
```

```
[47]: [1, 2, 3]
```

```
[48]: foo.size
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-48-a76262c15f5a> in <cell line: 1>()
```

```
----> 1 foo.size

AttributeError: 'list' object has no attribute 'size'
```

[49]:
```
foo.pop()
```

[49]: 3

[50]:
```
foo.push(3)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-50-399717872260> in <cell line: 1>()
----> 1 foo.push(3)

AttributeError: 'list' object has no attribute 'push'
```

[52]:
```
class Queue(list):
  '''
  Implements a queue data structure with methods push(), pop(), and show(),
  and attribute size.
  '''

  def __init__(self, initial_list=None):
    '''Initialize a Queue given an optional list'''
    if initial_list is not None:
      super().__init__(initial_list)
      self.size = len(initial_list)
    else:
      super().__init__()
      self.size = 0

  def push(self, item):
    '''Push an element onto the Queue'''
    self.append(item)
    self.size += 1

  def pop(self):
    '''Remove an element from the Queue'''
    if self.size > 0:
      self.size -= 1
      return super().pop(0)
    else:
      return None

  def show(self):
    '''Show the elements in the Queue'''
    return(self)
```

[53]:

```
# Example usage:
q = Queue([1, 2, 3])
q
```

[53]:

```
[1, 2, 3]
```

[54]:

```
type(q)
```

[54]:

```
__main__.Queue
```

[55]:

```
print("Initial queue:", q)
```

```
Initial queue: [1, 2, 3]
```

[56]:

```
print("Size of the queue:", q.size)
```

```
Size of the queue: 3
```

[57]:

```
print("Popped item:", q.pop())
```

```
Popped item: 1
```

[58]:

```
print("Queue after popping:", q)
```

```
Queue after popping: [2, 3]
```

[59]:

```
print("Size after popping:", q.size)
```

```
Size after popping: 2
```

[60]:

```
q.push(100)
q
```

[60]:

```
[2, 3, 100]
```

[61]:

```
print(q.show())
type(q)
```

```
[2, 3, 100]
```

[61]:

```
__main__.Queue
```

[ ]:

```

```

1.6 Your turn

- Create your own class.
- List a few attributes (>2)
- Create at least 3 methods - init, repr, and at least 1 method of your own

- Create a new object of your class
- Print your object, print your object's attributes, & run your method

---

Ideas: - Car - Person - Plant

[62]:

```
# Solution
class Fibonacci:
  def __init__(self):
    self.memo = {}

  def fibonacci(self, n):
    if n in self.memo:
      return self.memo[n]
    if n <= 1:
      return n
    else:
      result = self.fibonacci(n - 1) + self.fibonacci(n - 2)
      self.memo[n] = result
      return result
```

[63]:

```
fib = Fibonacci()
print(fib.fibonacci(10))  # Output: 55
```

```
55
```

[ ]: