Lecture 15 - Cool Little Objects

YouTube

Python Objects - https://youtu.be/pCYMwbcnRdM

From Amazon S3 - for download (same as youtube videos)

Python Objects

An object is a mix of data and code.

Let’s model vehicles.

How do we associate code and data in a simple class.

“words” that you have to use:

class - the thing that defines that this is an “object”

init - the thing that sets up a class - called a “constructor”

main - the special name that allows you to build tests in the same file as a class.

self - the name that allows you to access the data associated with this “instance” of an object.

an “instances” is the “type” of a class with its data.

This is the definition or “type” for the object.

vehicle1.py:

```
class Vehicle:

    """ this is a constructor """
    def __init__ ( self, manufacturer, vtype, year ):
        self.year = year
        self.vtype = vtype
        self.manufacturer = manufacturer
        self.odometer = 0


    """ this is a Method """
    def SetMilage ( self, n ):
        if n < 0 :
            print ( "Error: can't take milage off" )
        else:
            self.odomoter = self.odomoter + n

v1 = Vehicle( 'Tonka', 'truck', 2020 )
v2 = Vehicle( 'NASA', 'moon rover', 1969 )

print ( f"v1 = {v1}" )
print ( f"v2 = {v2}" )
```

Ouptut:

```
v1 = <__main__.Vehicle object at 0x7f46c1a2c8b0>
v2 = <__main__.Vehicle object at 0x7f46c06b9190>
```

When we run it we don’t get a useful output for the objects this can be fixed with a string conversion. That is the function `__str__` .

```
class Vehicle:

    """ this is a constructor """
    def __init__ ( self, manufacturer, vtype, year ):

        self.year = year
        self.vtype = vtype
        self.manufacturer = manufacturer
        self.odometer = 0


    """ this is a Method """
    def SetMilage ( self, n ):
        if n < 0 :
            print ( "Error: can't take milage off" )
        else:
            self.odomoter = self.odomoter + n

    """ Print out the class """
    def __str__ ( self ) :
        return ( f"Vehicle: {self.vtype} built in {self.year} by {self.manufacturer}" )

v1 = Vehicle( 'Tonka', 'truck', 2020 )
v2 = Vehicle( 'NASA', 'moon rover', 1969 )

print ( f"v1 = {v1}" )
print ( f"v2 = {v2}" )
```

Ouptut:

```
v1 = Vehicle: truck built in 2020 by Tonka
v2 = Vehicle: moon rover built in 1969 by NASA
```

So what are the parts.

Now let’s increment the odometer by 2 miles.

```
class Vehicle:

    """ this is a constructor """
    def __init__ ( self, manufacturer, vtype, year ):
        self.year = year
        self.vtype = vtype
        self.manufacturer = manufacturer
        self.odometer = 0


    """ this is a Method """
    def SetMilage ( self, n ):
        if n < 0 :
            print ( "Error: can't take milage off" )
        else:
            self.odometer = self.odometer + n

    def GetMilage ( self ):
        return self.odometer

    """ Print out the class """
    def __str__ ( self ) :
        return ( f"Vehicle: {self.vtype} built in {self.year} by {self.manufacturer} mi

v1 = Vehicle( 'Tonka', 'truck', 2020 )
v2 = Vehicle( 'NASA', 'moon rover', 1969 )

print ( f"v1 = {v1}" )
print ( f"v2 = {v2}" )

v2.SetMilage(2)

print ( f"v2 = {v2}" )
```

And the output is:

```
v1 = Vehicle: truck built in 2020 by Tonka milage 0
v2 = Vehicle: moon rover built in 1969 by NASA milage 0
v2 = Vehicle: moon rover built in 1969 by NASA milage 2
```

Objects are useful concept in describing a set of data.

Objects group into a hierarchy.

The Term is “inheritance” - that one object can inherit the data and methods from another object.

```
class Vehicle:

    """ this is a constructor """
    def __init__ ( self, manufacturer, vtype, year ):
        self.year = year
        self.vtype = vtype
        self.manufacturer = manufacturer
        self.odometer = 0


    """ this is a Method """
    def SetMilage ( self, n ):
        if n < 0 :
            print ( "Error: can't take milage off" )
        else:
            self.odometer = self.odometer + n

    def GetMilage ( self ):
        return self.odometer

    """ Print out the class """
    def __str__ ( self ) :
        return ( f"Vehicle: {self.vtype} built in {self.year} by {self.manufacturer} mi

class Airplane(Vehicle):

    """ this is a constructor """
    def __init__ ( self, manufacturer, vtype, year, wingspan, maxSpeed ):
        super().__init__( manufacturer, vtype, year )
        self.wingspan = wingspan
        self.maxSpeed = maxSpeed
    """ Print out the class """
    def __str__ ( self ) :
        return ( f"Vehicle/Airplane: {self.vtype} built in {self.year} by {self.manufac

class Auto(Vehicle):

    """ this is a constructor """
    def __init__ ( self, manufacturer, vtype, year, maxSpeed ):
        super().__init__( manufacturer, vtype, year )
        self.maxSpeed = maxSpeed

    """ Print out the class """
    def __str__ ( self ) :
        return ( f"Vehicle/Automobile: {self.vtype} built in {self.year} by {self.manuf
```

```
v1 = Auto( 'Tonka', 'truck', 2020, 2 )
v2 = Auto( 'NASA', 'moon rover', 1969, 2 )
v3 = Airplane( 'Piper', 'Cherokee', 1969, 32, 185 )
v4 = Airplane( 'Pliatus', 'PC 12 NC', 1969, 41, 581 )

print ( f"v1 = {v1}" )
print ( f"v2 = {v2}" )
print ( f"v3 = {v3}" )
print ( f"v4 = {v4}" )


v2.SetMilage(2)
# v4.SetMilage(35200)

print ( f"v2 = {v2}" )
```

And the output is:

```
v1 = Vehicle/Automobile: truck built in 2020 by Tonka milage 0 Max Speed 2
v2 = Vehicle/Automobile: moon rover built in 1969 by NASA milage 0 Max Speed 2
v3 = Vehicle/Airplane: Cherokee built in 1969 by Piper milage 0 Max Speed 185 wingspan
v4 = Vehicle/Airplane: PC 12 NC built in 1969 by Pliatus milage 0 Max Speed 581 wingspa
v2 = Vehicle/Automobile: moon rover built in 1969 by NASA milage 2 Max Speed 2
```

This is *useful*! Lots of stuff can be modeled in this fusion. Lot's of software is done this way - this includes lots of libraries that form Python itself. This is why when we refer to strings we say `string.upper()` - we have a `string` object and it has a method `upper()` .

This isn't the only way to organize data. But if you have a system that deals with 50% or maybe 80% of data then that is a useful abstraction.

One of the most important object hierarchies is something that you use all the time. That is the screen and all of the "objects" in your user interface. When you build a user interface this system of organization is quite often critical to the entire process of building the software.

Example! - how a User Interface Works.

Now not all things fall into this kind of a category/hierarchy.

Example! - how a user / authentication system works.

class Vehicle:

```
""" this is a constructor """
def __init__ ( self, manufacturer, vtype, year ):
    self.year = year
    self.vtype = vtype
    self.manufacturer = manufacturer
    self.odometer = 0


""" this is a Method """
def SetMilage ( self, n ):
    if n < 0 :
        print ( "Error: can't take milage off" )
    else:
        self.odomoter = self.odomoter + n
```

v1 = Vehicle('Tonka', 'truck', 2020) v2 = Vehicle('NASA', 'moon rover', 1969)

print (f"v1 = {v1}") print (f"v2 = {v2}")

class Vehicle:

```
""" this is a constructor """
def __init__ ( self, manufacturer, vtype, year ):
    self.year = year
    self.vtype = vtype
```

```
        self.manufacturer = manufacturer
        self.odometer = 0


    """ this is a Method """
    def SetMilage ( self, n ):
        if n < 0 :
            print ( "Error: can't take milage off" )
        else:
            self.odomoter = self.odomoter + n

    """ Print out the class """
    def __str__ ( self ) :
        return ( f"Vehicle: {self.vtype} built in {self.year} by {self.manufacturer}" )
```

v1 = Vehicle(‘Tonka’, ‘truck’, 2020) v2 = Vehicle(‘NASA’, ‘moon rover’, 1969)

print (f”v1 = {v1}”) print (f”v2 = {v2}”)