MR2020: Coding for METOC

Module 8: Classes

What is a Class?

Classes are the building blocks of object-oriented programming. Objects have attributes (things that describe them) and methods (things that can be done with them).

```
class Pair:
    def __init__(self,num1,num2):
        self.number1 = num1
        self.number2 = num2
    def sum(self):
        return self.number1 + self.number2
    def product(self):
        return self.number1 * self.number2
    def isequal(self):
        if self.number1 == self.number2:
            print('Numbers are equal.’)
        else:
            print('Numbers are not equal.')
```

Anatomy of a Class

```
class Pair:
    def __init__(self,num1,num2):
        self.number1 = num1
        self.number2 = num2
    def sum(self):
        return self.number1 + self.number2
    def product(self):
        return self.number1 * self.number2
    def isequal(self):
        if self.number1 == self.number2:
            print('Numbers are equal.’)
        else:
            print('Numbers are not equal.')
```

The name of the class is Pair.

Anatomy of a Class

The __init__ function is used to establish an instance and set its attributes. It is called automatically when an instance is created.

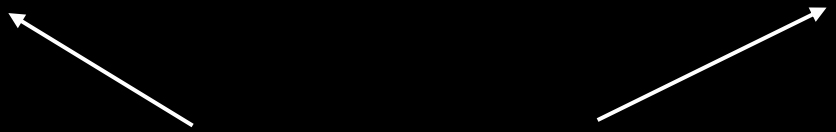
num1 and num2 are the inputs into the class

```
class Pair:
    def __init__(self,num1,num2):
        self.number1 = num1
        self.number2 = num2
    def sum(self):
        return self.number1 + self.number2
    def product(self):
        return self.number1 * self.number2
    def isequal(self):
        if self.number1 == self.number2:
            print('Numbers are equal.’)
        else:
            print('Numbers are not equal.')
```

Creating an Instance

leroyjenkins is the name of the new object.

leroyjenkins is made an instance of the class Pair with inputs 3 and 4.



```
leroyjenkins = Pair(3,4)
```

The 3 and 4 are assigned to attributes as described in the __init__ function for the Pair class.

`leroyjenkins` is an **instance** of the class Pair.

Anatomy of a Class

```
class Pair:
    def __init__(self,num1,num2):
        self.number1 = num1
        self.number2 = num2
    def sum(self):
        return self.number1 + self.number2
    def product(self):
        return self.number1 * self.number2
    def isequal(self):
        if self.number1 == self.number2:
            print('Numbers are equal.’)
        else:
            print('Numbers are not equal.')
```

This class has three methods, sum, product, and isequal. They are all written as functions inside the class.

Methods describe things a class can do or that can be done to it.

Anatomy of a Class

The sum method simply returns the sum of the two attributes.

```
class Pair:
    def __init__(self,num1,num2):
        self.number1 = num1
        self.number2 = num2
    def sum(self):
        return self.number1 + self.number2
    def product(self):
        return self.number1 * self.number2
    def isequal(self):
        if self.number1 == self.number2:
            print('Numbers are equal.’)
        else:
            print('Numbers are not equal.')
```

Anatomy of a Class

The product method returns the product of the two attributes.

```
class Pair:
    def __init__(self,num1,num2):
        self.number1 = num1
        self.number2 = num2
    def sum(self):
        return self.number1 + self.number2
    def product(self):
        return self.number1 * self.number2
    def isequal(self):
        if self.number1 == self.number2:
            print('Numbers are equal.’)
        else:
            print('Numbers are not equal.')
```

Anatomy of a Class

```
class Pair:
    def __init__(self,num1,num2):
        self.number1 = num1
        self.number2 = num2
    def sum(self):
        return self.number1 + self.number2
    def product(self):
        return self.number1 * self.number2
    def isequal(self):
        if self.number1 == self.number2:
            print('Numbers are equal.’)
        else:
            print('Numbers are not equal.')
```

The isequal method evaluates whether the two attributes are equal and uses an if-else control flow to print whether they are equal.

Calling Attributes and Methods

```
leroyjenkins = Pair(3,4)

# Calling attributes
leroyjenkins.number1
leroyjenkins.number2

# Calling methods
leroyjenkins.sum()
leroyjenkins.product()
leroyjenkins.isequal()
```

1) Attributes and methods are called by inserting a period after the object name.

2) Attributes are all the variables defined in the __init__ function of the class and can be called by inserting their name after the period.

3) Methods are all the functions (aside from __init__) that are defined inside the class. They can be accessed by inserting the function name after the period.

4) Parentheses must be inserted after a function name but not after an attribute.

Class and Instance Variables

Class variables are shared across *all* instances of a class. Example: All dogs are *canis familiaris*.

Instance variables are unique to each instance of a class and are established when the object is created and the __init__ function is executed. Example: Not every dog has the same name.

```
class Dog:
    species = 'Canis familiaris'
    def __init__(self, name):
        self.name = name
```

species is a class variable. All instances of Dog will have this attribute regardless of is passed to the __init__ function as input.

name is an instance variable. Different instances of dog might have different names depending on the input provided when the object is created.

Inheritance

```
# Base class
class Animal
    def __init__(self, name):
        self.name = name
    def speak(self):
        return "Some sound"

# Derived class
class Dog(Animal):
    def __init__(self, name, breed):
        # Call the __init__ method of the base class
        super().__init__(name)
        self.breed = breed

    # Override the speak method
    def speak(self):
        return "Woof!"

    # Additional method specific to Dog
    def fetch(self):
        return f"{self.name} is fetching the ball!"

# Create an instance of Dog
my_dog = Dog("Buddy", "Golden Retriever")

# Access attributes and methods
print(f"Name: {my_dog.name}") # Output: Name: Buddy
print(f"Breed: {my_dog.breed}") # Output: Breed: Golden Retriever
print(my_dog.speak()) # Output: Woof!
print(my_dog.fetch()) # Output: Buddy is fetching the ball!
```

Inheritance is a mechanism for creating a new class using details of an existing class without modifying it.

Dog inherits the attributes and methods of Animal here.

The speak method of Animal is overwritten by a speak method specific to Dog.

Dog is given a new method that the generic Animal doesn't have.