•••

Unit 2: Design Principles

Session 1

Lesson Outcome:

Understanding Objects and Classes in Object Oriented Programming

Object Oriented Design

Topic 1: Objects and Classes

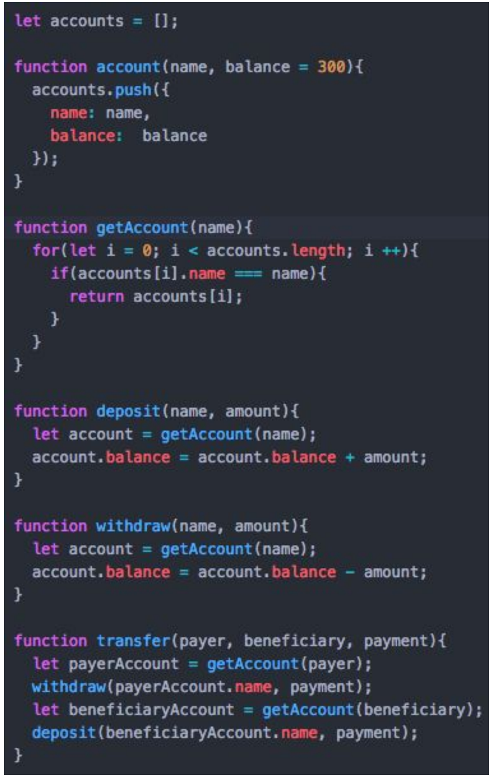
DESIGN PRINCIPLES

These are guidelines that aid the software design process.

Procedural Programming

- In procedural programming, a program is divided into functions that perform specific tasks

- Data is global, which means that all the functions can access global data



```
let accounts = [];

function account(name, balance = 300){
  accounts.push({
    name: name,
    balance:  balance
  });
}

function getAccount(name){
  for(let i = 0; i < accounts.length; i ++){
    if(accounts[i].name === name){
      return accounts[i];
    }
  }
}

function deposit(name, amount){
  let account = getAccount(name);
  account.balance = account.balance + amount;
}

function withdraw(name, amount){
  let account = getAccount(name);
  account.balance = account.balance - amount;
}

function transfer(payer, beneficiary, payment){
  let payerAccount = getAccount(payer);
  withdraw(payerAccount.name, payment);
  let beneficiaryAccount = getAccount(beneficiary);
  deposit(beneficiaryAccount.name, payment);
}
```

Object Oriented Programming

- Object-oriented programming (OOP) is a way of structuring a program by bundling related properties and behaviors into individual objects.

- The concept of OOP in Python focuses on creating reusable code.
  - This concept is also known as DRY (Don't Repeat Yourself).

Objects

- Objects are like the components of a system
- An object contains data
- What if objects can be more than just a data type? What if we can actually get objects to do things?

Objects

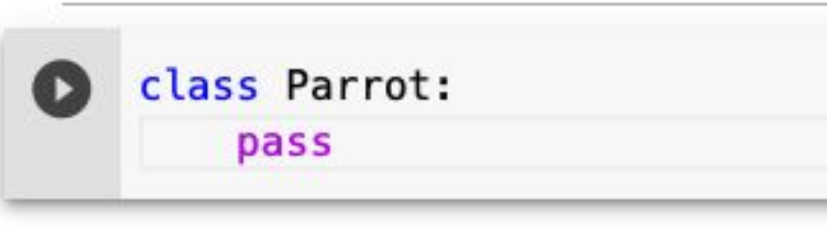
- An object has two characteristics:
  - attributes
  - behavior
- A parrot is an object, as it has the following properties:
  - name, age, color as attributes
  - singing, dancing as behavior

Class

- In Python, the concept of OOP follows some basic principles:
  - A class is a blueprint for the object.
  - We can think of class as a sketch of a parrot with labels. It contains all the details about the name, colors, size etc.
  - Here, a parrot is an object.

Class

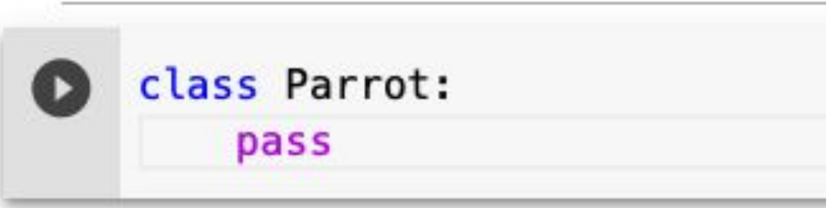
- The example for class of parrot can be:



```
class Parrot:
    pass
```

Class

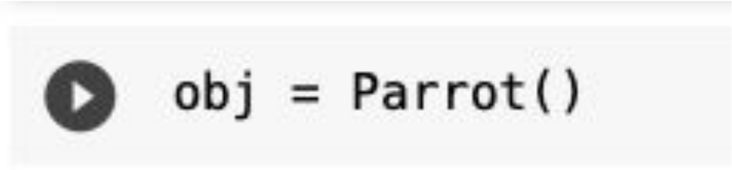
- Here, we use the class keyword to define an empty class Parrot. From class, we construct instances. An instance is a specific object created from a particular class



```
class Parrot:
    pass
```

Object

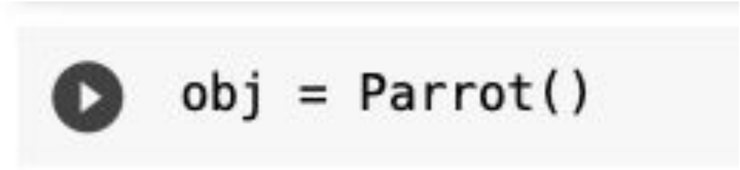
- An object is an **instance** of a class.
- When a class is defined, only the description for the object is defined.
- We have to actually CREATE the object from the class.
- The example for object of parrot class :



```
obj = Parrot()
```

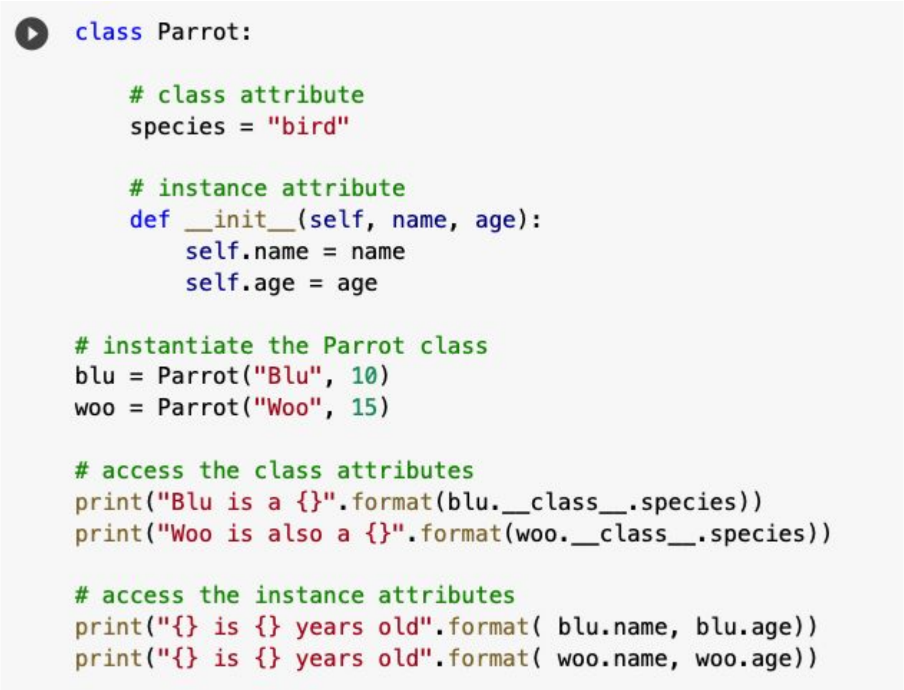
Object

- Here, ‘obj’ is an object of class Parrot.



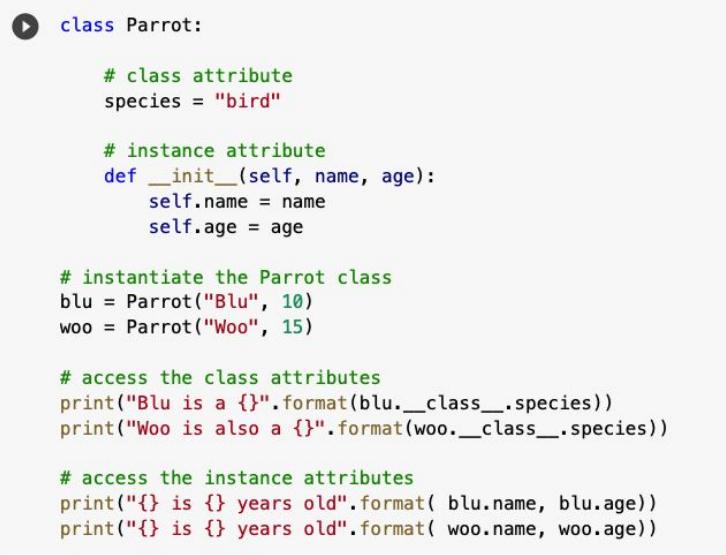
```
obj = Parrot()
```

Classes and Objects



```
class Parrot:

    # class attribute
    species = "bird"

    # instance attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age

# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))

# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

Classes and Objects

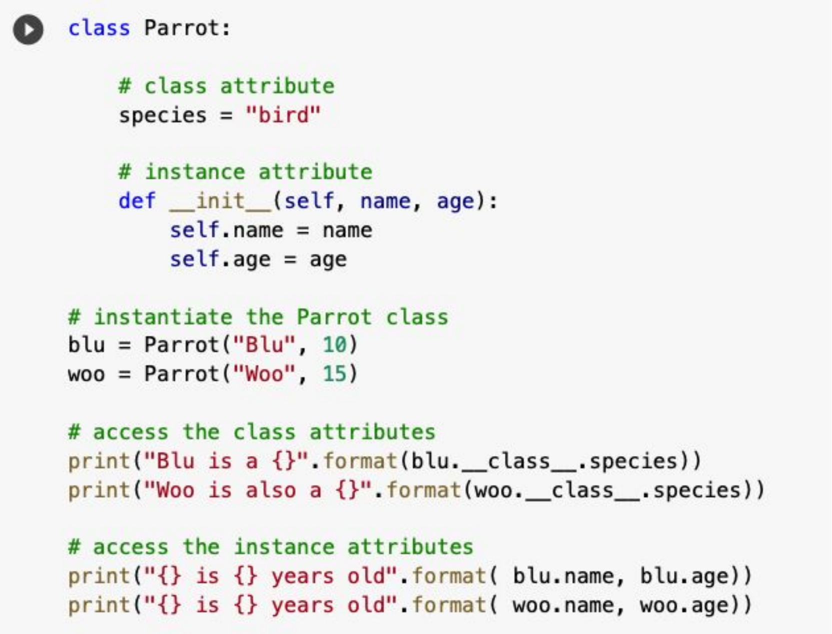


```python
class Parrot:

    # class attribute
    species = "bird"

    # instance attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age

# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))

# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

- Write out the code. What output do you get?

Classes and Objects

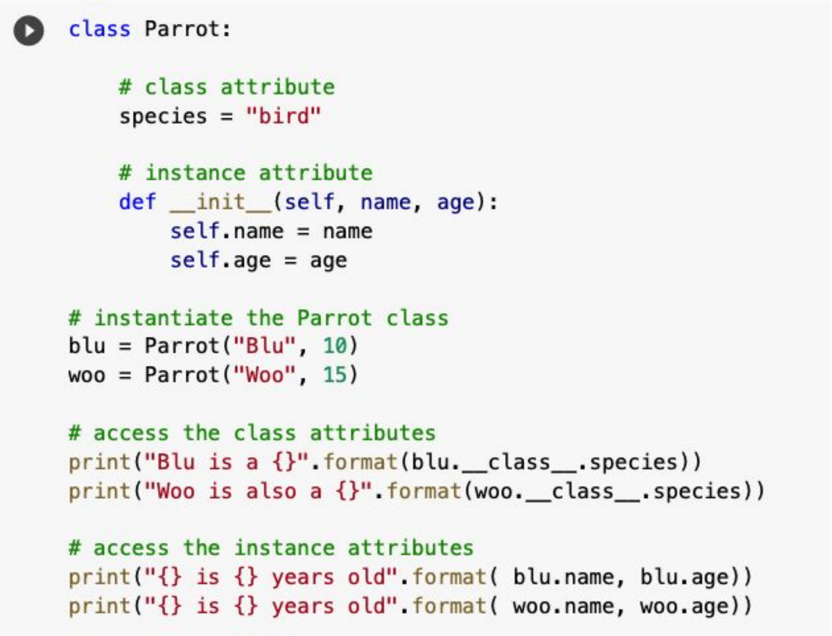
- In the program, we create a class with the name Parrot. Then, we define attributes. The attributes are a characteristic of an object.
- These attributes are defined inside the __init__ method of the class. It is the initializer method that is first run as soon as the object is created.



```python
class Parrot:

    # class attribute
    species = "bird"

    # instance attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age

# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))

# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

Classes and Objects

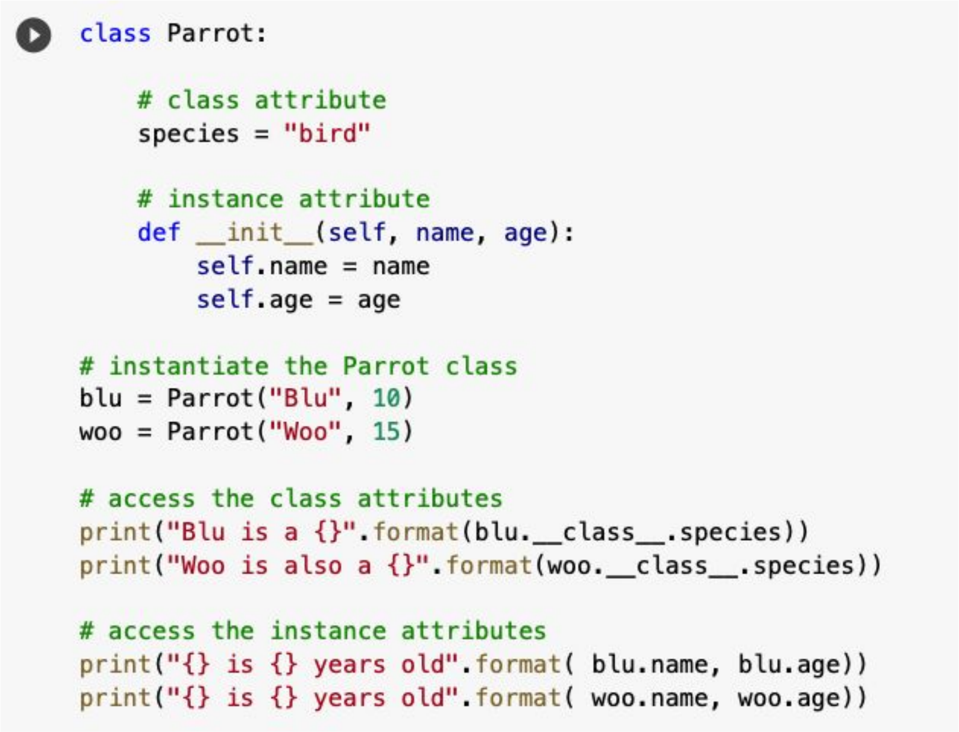
- Then, we create instances of the Parrot class. Here, blu and woo are references (value) to our new objects.
- We can access the class attribute using __class__.species. Class attributes are the same for all instances of a class.



```
class Parrot:

    # class attribute
    species = "bird"

    # instance attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age

# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))

# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

Classes and Objects

- Similarly, we access the instance attributes using blu.name and blu.age. However, instance attributes are different for every instance of a class.



```
class Parrot:

    # class attribute
    species = "bird"

    # instance attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age

# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))

# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

How are you feeling?

RED

I have no idea what you're talking about

YELLOW

I have some questions but feel like I understand some things

GREEN

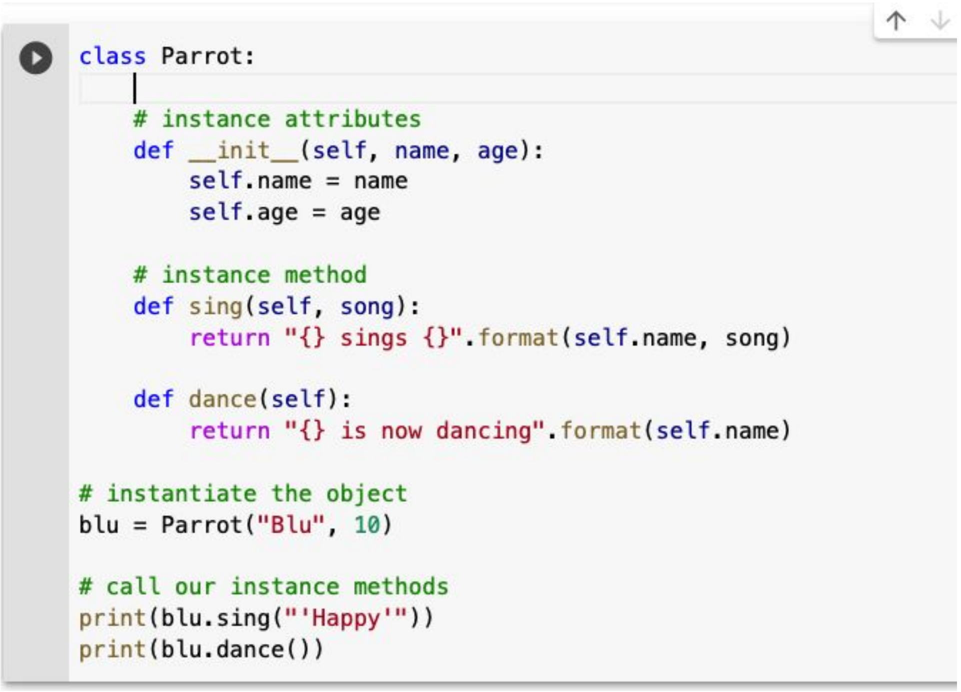
I feel comfortable with everything you've said

Topic 2: Methods

Methods

- Methods are functions defined inside the body of a class.
- They are used to define the behaviors of an object.

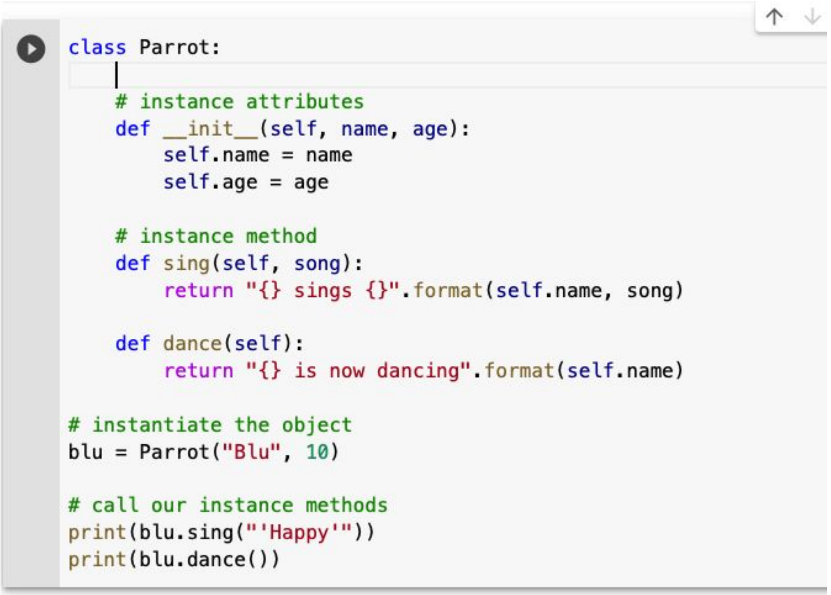
Methods



```
class Parrot:

    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def sing(self, song):
        return "{} sings {}".format(self.name, song)

    def dance(self):
        return "{} is now dancing".format(self.name)

# instantiate the object
blu = Parrot("Blu", 10)

# call our instance methods
print(blu.sing("'Happy'"))
print(blu.dance())
```

Methods

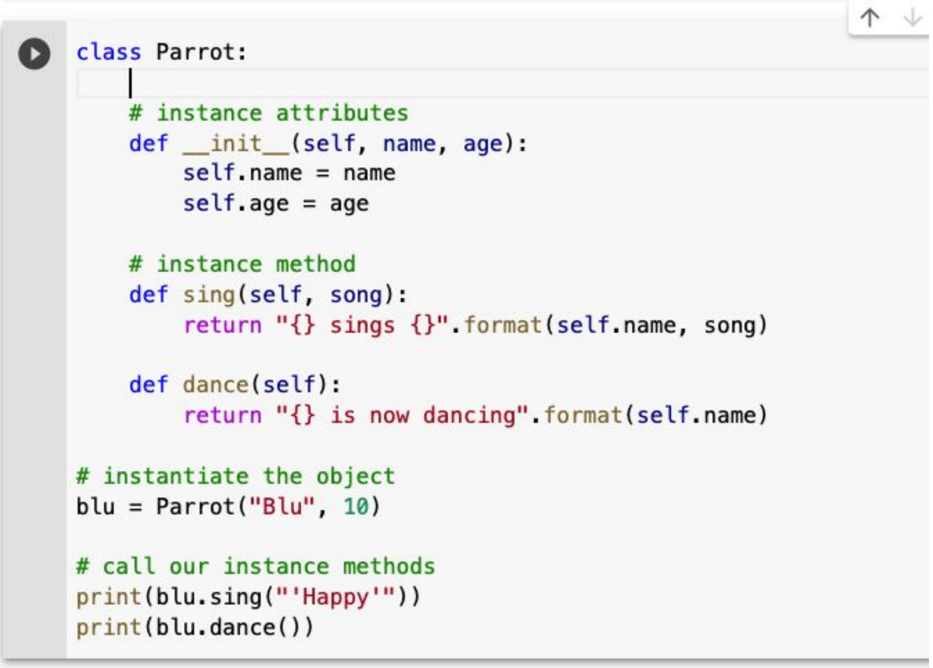
- In our code, we define two methods i.e sing() and dance().
- These are called instance methods because they are called on an instance object i.e blu.



```python
class Parrot:

    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def sing(self, song):
        return "{} sings {}".format(self.name, song)

    def dance(self):
        return "{} is now dancing".format(self.name)

# instantiate the object
blu = Parrot("Blu", 10)

# call our instance methods
print(blu.sing("'Happy'"))
print(blu.dance())
```

Methods

- What is the output when we call the methods?



```
class Parrot:

    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def sing(self, song):
        return "{} sings {}".format(self.name, song)

    def dance(self):
        return "{} is now dancing".format(self.name)

# instantiate the object
blu = Parrot("Blu", 10)

# call our instance methods
print(blu.sing("'Happy'"))
print(blu.dance())
```

Session 2

Practical Session:

Creating our own Objects and Classes. Independent work!

Creating custom modules in python

Modules

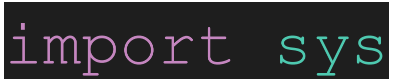
Modules refer to a file containing Python statements and definitions.

We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code.

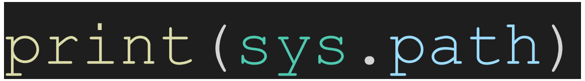
Modules

While importing a module, Python looks at several places. Interpreter first looks for a built-in module. Then(if built-in module not found), Python looks into a list of directories define in `sys.path`. The search is in this order.

- The current directory.
- `PYTHONPATH` (an environment variable with a list of directories).
- The installation-dependent default directory.



```
import sys
```



```
print(sys.path)
```

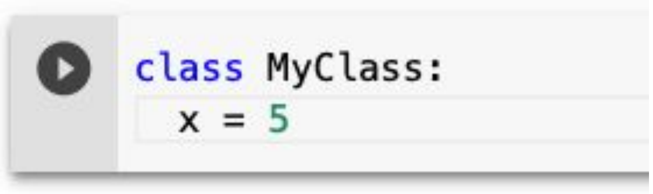
Modules

```
import sys
```

```
print(sys.path)
```

Create a Class:

- To create a class, use the keyword **class**:

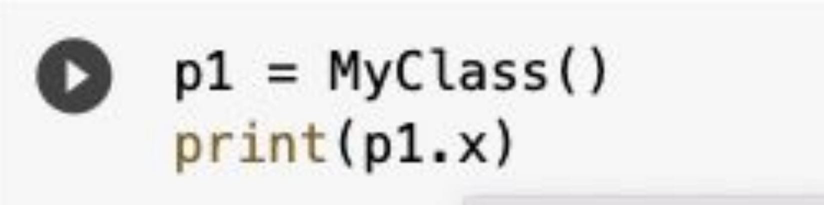


```
class MyClass:
  x = 5
```

- This class currently has a property named x.
- Add properties that are appropriate for your class.

Create Objects from your Class:

- Here I have created an object named p1, that prints the value of the attribute x:



```
p1 = MyClass()
print(p1.x)
```

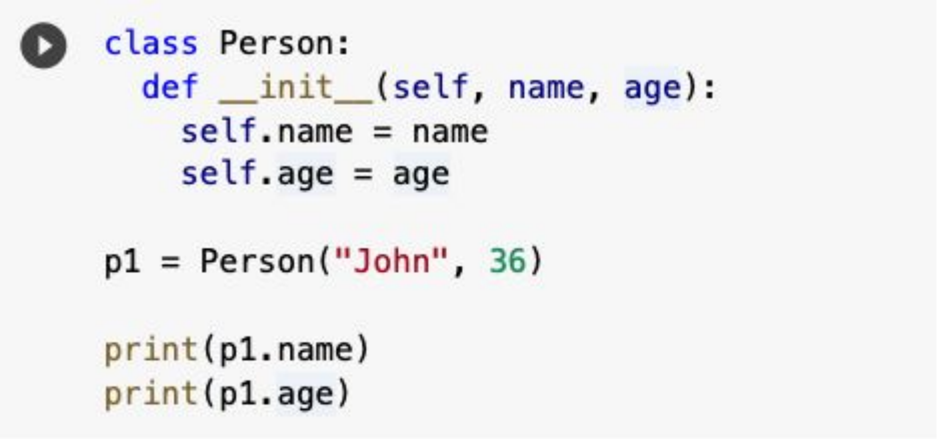
Similarly, create an object from your class and print out the value of the attribute that you created

The __init__() function

- To understand the meaning of classes we have to understand the built-in __init__() function.
- All classes have a function called __init__(), which is always executed when the class is being initiated.
- Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

The __init__() function

- In this example of a Person class, we use __init()___ to assign values for name and age.
- These values are assigned by default each time a Person object is made



```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Thank You