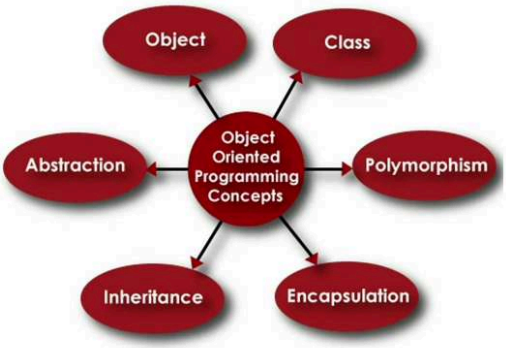
Concept of OOP

- Provides a means of structuring programs so that properties and behaviors are bundled into individual objects.
- OOP reflects the real world behavior of how things work
- It make visualization easier because it is closest to real world scenarios.
- We can reuse the code through inheritance , this saves time, and shrinks our project.
- There are flexibility through polymorphism

No connection

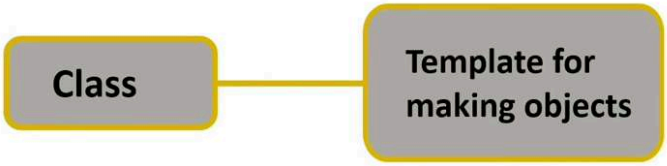
Pillars of OOP

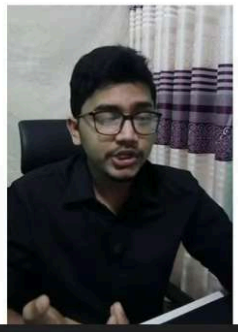


No connection

Class

- A class is the blueprint for the objects created from that class
- Each class contains some data definitions(called fields), together with methods to manipulate that data
- When the object is instantiated from the class, an instance variable is created for each field in the class

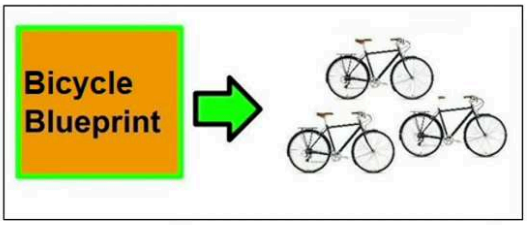




No connection

Object

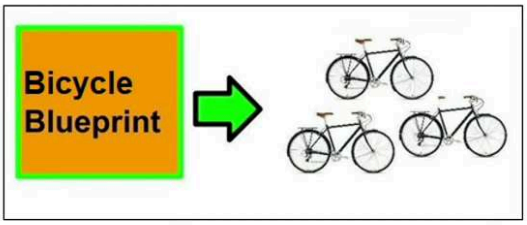
- Object are the basic run time entities in an object oriented system
- It is an instance of class
- We can say that objects are the variables of the type class



No connection

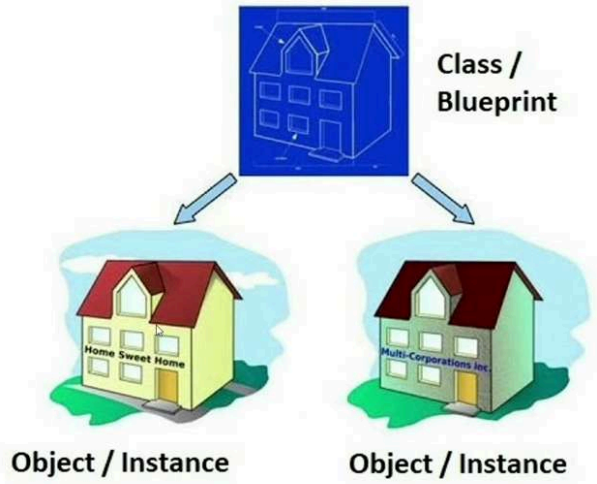
Object

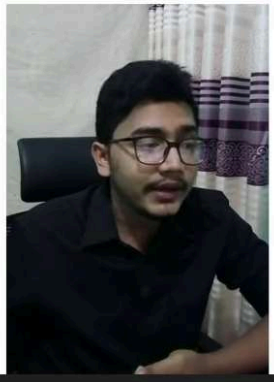
- Object are the basic run time entities in an object oriented system
- It is an instance of class
- We can say that objects are the variables of the type class



No connection

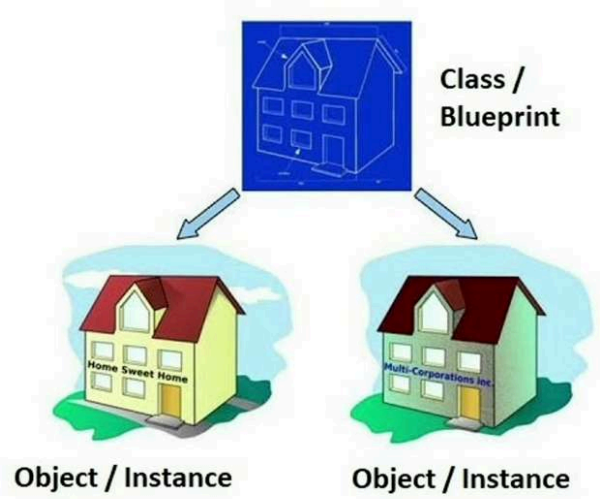
Class & Object

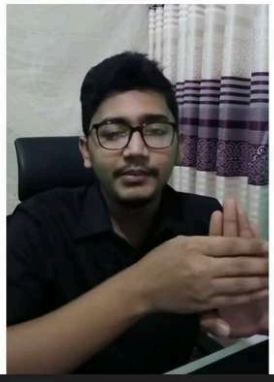




No connection

Class & Object



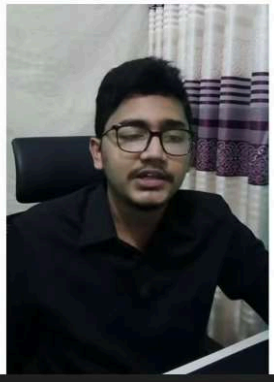


No connection

Class Components

- Data (the attributes about it)
- Behavior (the methods)

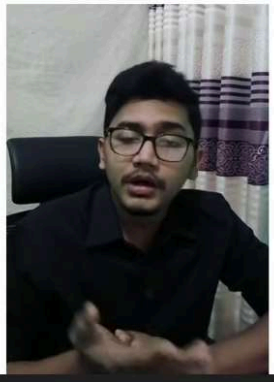
Data	Method()
• driver_name • num_passenger	• pick_up_passenger() • drop_off_passenger()



No connection

Method

- A Python method is like a Python function
- It must be called on an object.
- It must put it inside a class
- A method has a name, and may take parameters and have a return statement



No connection

Class Components

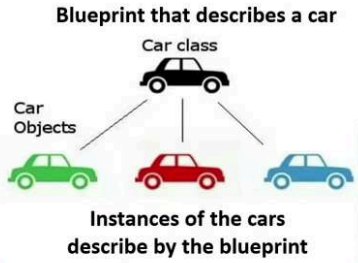
Taxi
• driver_name: **string** • num_passenger: **int**
• Pick_up_passenger() • Drop_off_passenger()

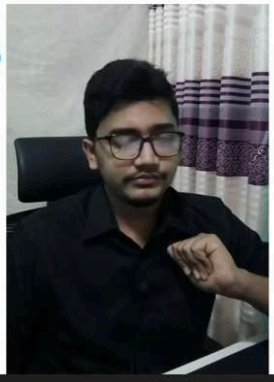
No connection

Class and Objects

	CLASS
Name	Car
Attributes	- Color - Brand
Behaviors	*Display()

OBJECT - 1	OBJECT - 2	OBJECT - 3
Mustang Car	Prius Car	Golf Car
- Green - Ford	- Red - Toyota	- Blue - Volkswagen
*Display()	*Display()	*Display()

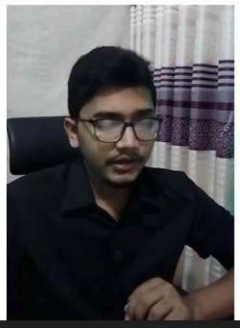




No connection

Constructor

- A special kind of method we use to initialize instance members of that class
- It is used for initializing the instance members when we create the object of a class.
- If you create four objects, the class constructor is called four times.
- Every class must have a constructor, even if it simply relies on the default constructor.
- Constructors can be of two types.
    - * Non-parameterized Constructor (Default constructor)
    - * Parameterized Constructor



No connection

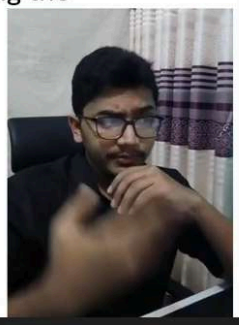
Python __init__

- "__init__" is a reserved method in python classes.
- It is known as a constructor in OOP concepts.
- This method is called when an object is created from the class and it allows the class to initialize the attributes of a class.
- It accepts the `self` -keyword as a first argument which allows accessing the attributes or method of the class.
- We can pass any number of arguments at the time of creating the class object, depending upon the __init__() definition.

No connection

Python __init__

- "__init__" is a reserved method in python classes.
- It is known as a constructor in OOP concepts.
- This method is called when an object is created from the class and it allows the class to initialize the attributes of a class.
- It accepts the `self` -keyword as a first argument which allows accessing the attributes or method of the class.
- We can pass any number of arguments at the time of creating the class object, depending upon the __init__() definition.

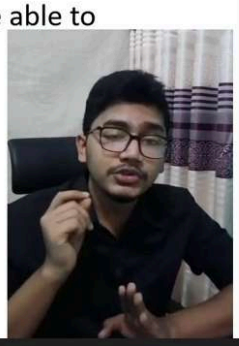


No connection

Non-parameterized Constructor (default constructor)

- When we do not include the constructor in the class or forget to declare it, then that becomes the default constructor.
- It does not perform any task but initializes the objects
- In the following example, we do not have a constructor but still we are able to create an object for the class.

```
class Employee:
    pass

emp1 = Employee()
emp2 = Employee()
```

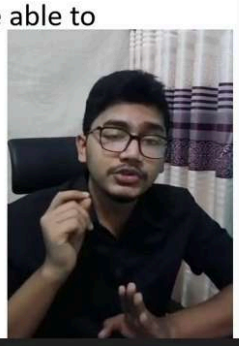


No connection

Non-parameterized Constructor (default constructor)

- When we do not include the constructor in the class or forget to declare it, then that becomes the default constructor.
- It does not perform any task but initializes the objects
- In the following example, we do not have a constructor but still we are able to create an object for the class.

```
class Employee:
    pass

emp1 = Employee()
emp2 = Employee()
```



No connection

Non-parameterized Constructor (default constructor)

```
class Employee:
    def __init__(self):
        print("Employee object created")

emp1 = Employee()
emp2 = Employee()
```

Output:

```
Employee object created
Employee object created
```

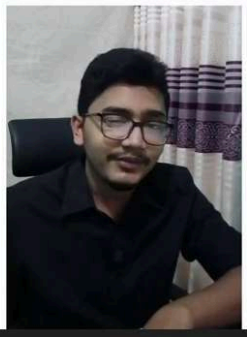
No connection

Python Parameterized Constructor

- The parameterized constructor has multiple parameters along with the **self**.
- It accepts the arguments during object creation

```
class Employee:

    #parameterized constructor
    def __init__(self, name):
        self.name = name  #instance variable
        print(self.name, "created")

emp1 = Employee("John")  #instance 1
emp2 = Employee("David") #instance 2
```

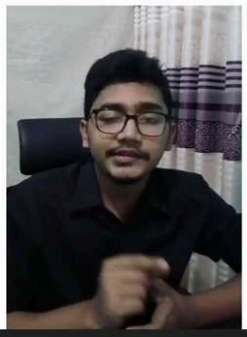
Output:
John created
David created



No connection

Instance Method

- Instance method are methods which require an object of its class to be created before it can be called.
- Instance methods need a class instance and can access the instance through `self`.
- Instance method takes more that one parameter, `self`, which points to an instance of class when the method is called
- The `self` parameter, instance methods can freely access attributes and other methods on the same object



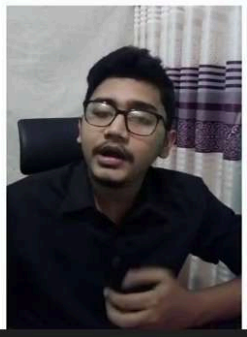
No connection

Class design example

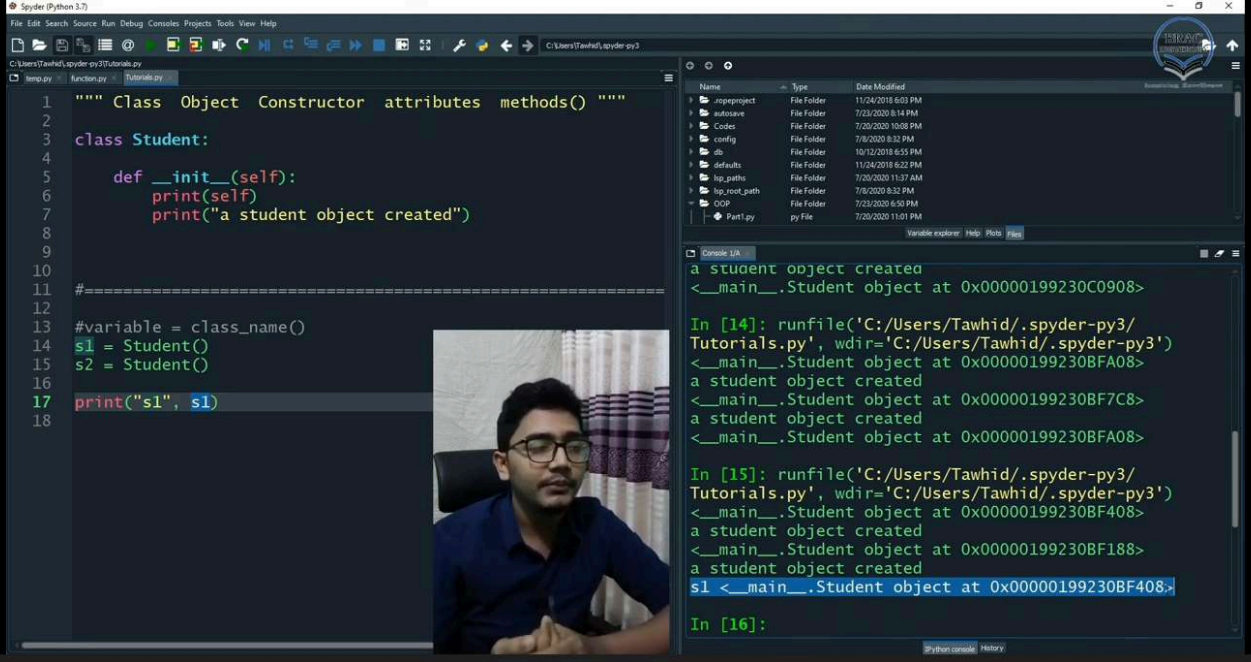
```
class Employee:

    #parameterized constructor
    def __init__(self, name, no):
        self.no = no        #instance variable
        self.name = name    #instance variable

    #instance method
    def display(self):
        print(self.name, self.no)

emp1 = Employee("John", 11)  #instance 1
emp2 = Employee("David", 12) #instance 2
emp1.display()
emp2.display()
```

Here, no and name are instance variables and these get initialized when we create an object for the Employee. If we want to call display() method which is an instance method, we must create an object for the class.

Output:
John 11
David 12



No connection

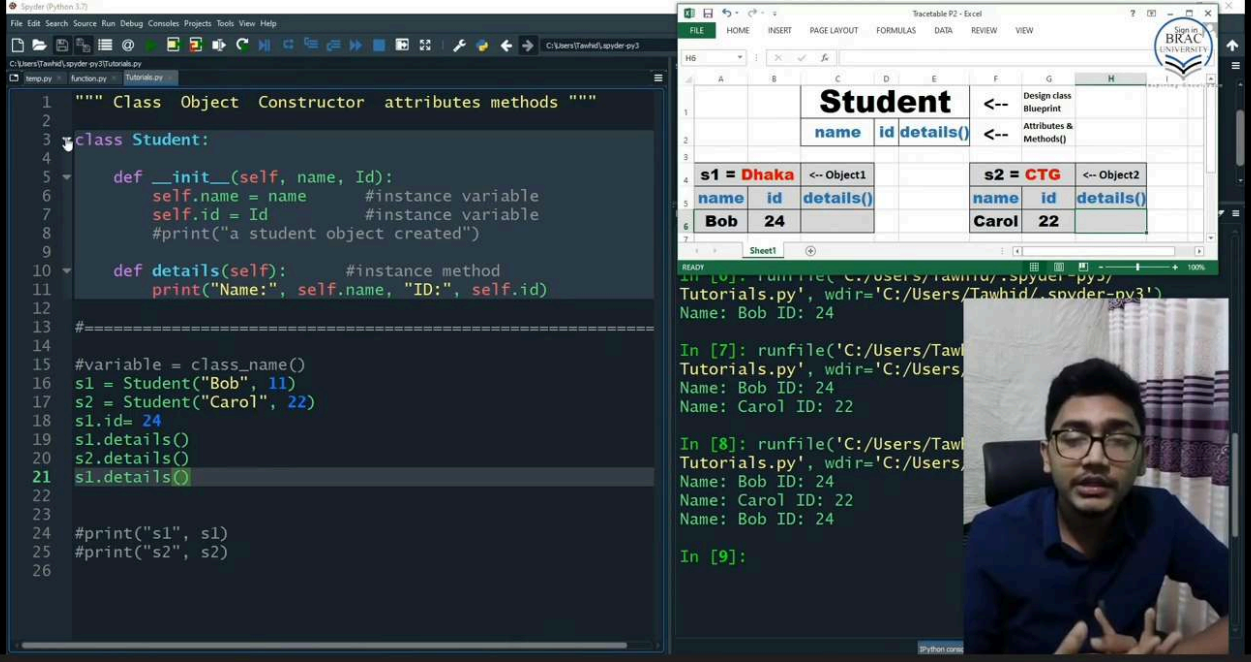


No connection

```python
""" Class  Object  Constructor  attributes """

class Student:

    def __init__(self, name, Id):
        self.name = name      #instance variable
        self.id = Id          #instance variable
        #print("a student object created")



#==========================================================

#variable = class_name()
s1 = Student("Bob", 11)
s2 = Student("Carol", 22)
print(s1.id)  #11
s1.id = 33
print(s1.id) #33

#print("s1", s1)
#print("s2", s2)
```

```
In [28]: runfile('C:/Users/Tawhid/.spyder-py3/
Tutorials.py', wdir='C:/Users/Tawhid/.spyder-py3')
11
33

In [29]:
```

No connection



No connection

```python
class Student:    #class / design/ blueprint

    def __init__(self, nm, iD):             #constructor
        self.name = nm
        self.Id = iD


#==========================================================

#variable = classname()   ebahbe object create kore

s1 = Student("Pranto", 1)    #1st object / instance
print(s1.name)
s1.name = "Roy"
print(s1.name)

print("---------------")

s2 = Student("ABC", 2)    #2nd object / instance
print(s2.name)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

```
In [12]: runfile('C:/Users/Tawhid/.spyder-py3/
csell1_final.py', wdir='C:/Users/Tawhid/.spyder-py3')
Pranto
1
---------------

In [13]: runfile('C:/Users/Tawhid/.spyder-py3/
csell1_final.py', wdir='C:/Users/Tawhid/.spyder-py3')
Pranto
---------------
ABC

In [14]: runfile('C:/Users/Tawhid/.spyder-py3/
csell1_final.py', wdir='C:/Users/Tawhid/.spyder-py3')
Pranto
Roy
---------------
ABC

In [15]:
```

```python
class House:

    def __init__(self):
        self.window = 4   #instacne variable
        self.door = 2      #instacne variable

    def view(self):
        print(self.window, "windows", self.door, "doors")

#==============================================================

h1 = House()
h2 = House()
h1.window = 6
h2.door = 3
h1.view()
h2.view()
```

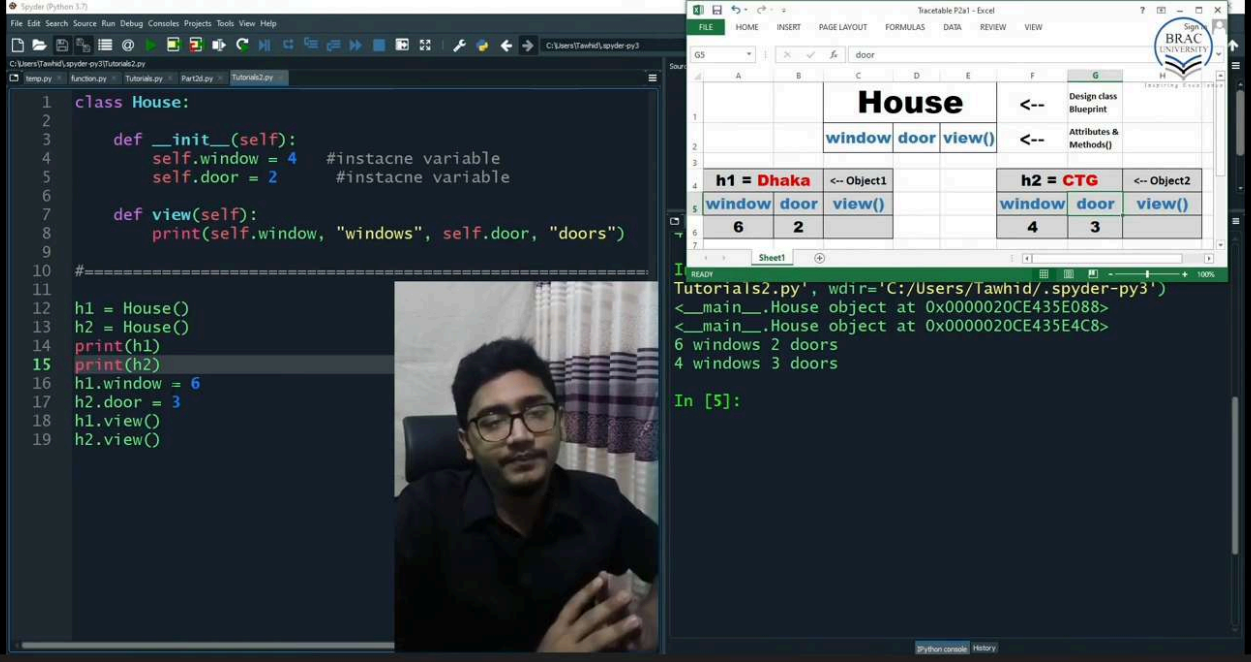
House			<--	Design class Blueprint
window	door	view()	<--	Attributes & Methods()

h1 = Dhaka		<-- Object1
window	door	view()
4	2	

h2 = CTG		<-- Object2
window	door	view()
4	2	

```
In [3]: runfile('C:/Users/Taw
Tutorials2.py', wdir='C:/User
6 windows 2 doors
4 windows 3 doors

In [4]:
```

No connection



No connection

```python

class Car:

    def __init__(self, name, model):
        self.name = name            #instance variable
        self.model_year = model      #instance variable
        self.wheel = 4               #instance variable

    def view(self):                  #instance method
        print("The model year of this", self.name,
              "is", self.model_year)
        print("It is a", self.wheel, "wheel car")

#==========================================================

c1 = Car("BMW", 2016)

c1.view()
```

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
In [28]: runfile('C:/Users/Tawhid/.spyder-py3/
Tutorials.py', wdir='C:/Users/Tawhid/.spyder-py3')
The model year of this BMW is 2016
It is a 4 wheel car

In [29]:
```

No connection

```python

class Dog:

    def __init__(self, name, color):
        self.name = name
        self.color = color

    def update_color(self, color):
        self.color = color

    def poke(self):
        print(self.color, self.name, "is smiling")

#==========================================================

d1 = Dog("rover", "Brown")
d2 = Dog("tomy", "White")

d1.poke()
d1.update_color("Black")
d1.poke()
#d2.poke()
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
In [19]: runfile('C:/Users/Tawhid/.spyder-py3/
Tutorials.py', wdir='C:/Users/
Brown rover is smiling
Black rover is smiling

In [20]:
```

```python
class Dog:

    def __init__(self, name, color):
        self.name = name
        self.color = color

    def update_color(self, color):
        self.color = color

    def poke(self):
        print(self.color, self.name, "is smiling")

#==========================================================

d1 = Dog("rover", "Brown")
d2 = Dog("tomy", "White")

#d1.poke()
d1.update_color("Black")
#d1.poke()

print(d2.__dict__)
print(dir(d1))
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
In [27]: runfile('C:/Users/Tawhid/.spyder-py3/
Tutorials.py', wdir='C:/Users/Tawhid/.spyder-py3')
{'name': 'tomy', 'color': 'White'}
['__class__', '__delattr__', '__dict__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '__weakref__', 'color', 'name',
'poke', 'update_color']

In [28]:
```

No connection

```python
#from folder_name import class_name
import book

b1 = book.Book("Opekkha", "Humayun Ahmed")
b1.set_price(255)
b1.details()


#tester class
```

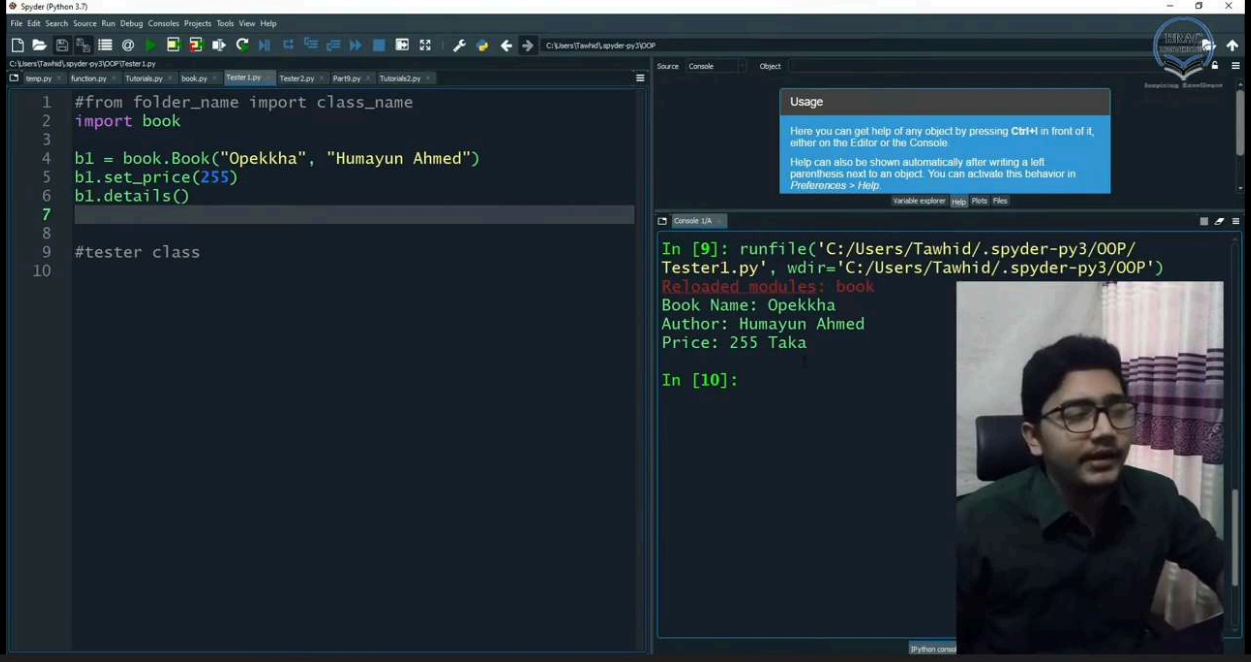
Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

```
In [7]: runfile('C:/Users/Tawhid/.spyder-py3/OOP/
Tester1.py', wdir='C:/Users/Tawhid/.spyder-py3/OOP')
Reloaded modules: book
Book Name: Opekkha
Author: Humayun Ahmed
Price: 255 Taka

In [8]:
```

No connection



No connection

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Tawhid\.spyder-py3\Tester2.py

temp.py | function.py | Tutorials.py | book.py | Tester1.py | Tester2.py* | Part9.py | Tutorials2.py

```
#from folder_name import class_name
from OOP import book

b1 = book.Book("Opekkha", "Humayun Ahmed")
b1.set_price(255)
b1.details()
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

```
In [9]: runfile('C:/Users/Tawhid/.spyder-py3/OOP/
Tester1.py', wdir='C:/Users/Tawhid/.spyder-py3/OOP')
Reloaded modules: book
Book Name: Opekkha
Author: Humayun Ahmed
Price: 255 Taka

In [10]:
```

No connection

```python

class Cat:

    def __init__(self, color, action):
        self.color = color
        self.action = action

    def view(self):
        print(self.color, "cat is", self.action)

    def compare(self, ct):
        if self.action == ct.action:
            print("Both cats are", self.action)
        else:
            print("They are different")

#==========================================================

c1 = Cat("White", "jumping")
c2 = Cat("Brown", "jumping")
c1.view()
c2.view()

c1.compare(c2)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

```
They are different

In [33]: runfile('C:/Users/Tawhid/.spyder-py3/
Tutorials.py', wdir='C:/Users
White cat is jumping
Brown cat is jumping
Both cats are jumping

In [34]:
```

No connection

```python
class Cat:

    def __init__(self, color, action):
        self.color = color
        self.action = action

    def view(self):
        print(self.color, "cat is", self.action)

    def compare(self, ct):
        if self.action == ct.action:
            print("Both cats are", ct.action)
        else:
            print("They are different")

#==============================================================

c1 = Cat("White", "jumping")
c2 = Cat("Brown", "jumping")
c1.view()
c2.view()

c1.compare(c2)
```

Cat					
				<--	Design class Blueprint
color	action	view()	cmpr()	<--	Attributes & Methods()

c1 = Dhaka			
color	action	view()	cmpr()
white	jumping		ct
			CTG

c2 = CTG			
color	action	view()	cmpr()
Brown	jumping		ct

```
In [34]: runfile('C:/Users/Ta
Tutorials.py', wdir='C:/Users
White cat is jumping
Brown cat is jumping
Both cats are jumping

In [35]:
```

No connection

```python
class Cat:
    def __init__(self, color, action):
        self.color = color
        self.action = action

    def view(self, num, clr):
        num = num + 5 #60
        clr1 = clr
        clr1[0] = "Green"
        print("Method inside:", num)
        print("Method inside:", clr1)

#==========================================================
colors = ["Black", "Red", "Yellow", "Blue"]
c1 = Cat("White", "jumping")
x = 55
c1.view(x, colors)
#print("Method outside", x)
#print("Method outside", colors)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

Variable explorer | Help | Plots | Files

Console 1/A

```
In [7]: runfile('C:/Users/Tawhid/.spyder-py3/
Tutorials2.py', wdir='C:/Users/Tawhid/.spyder-py3')
Method inside: 60
Method inside: ['Green', 'Red', 'Yellow', 'Blue']

In [8]:
```

No connection

```python
class Cat:
    def __init__(self, color, action):
        self.color = color
        self.action = action

    def view(self, num, clr):
        num = num + 5 #60
        clr1 = clr
        clr1[0] = "Green"
        print("Method inside:", num)
        print("Method inside:", clr1)

#==========================================================
colors = ["Black", "Red", "Yellow", "Blue"]
c1 = Cat("White", "jumping")
x = 55
c1.view(x, colors)
print("Method outside", x)
print("Method outside", colors)
```

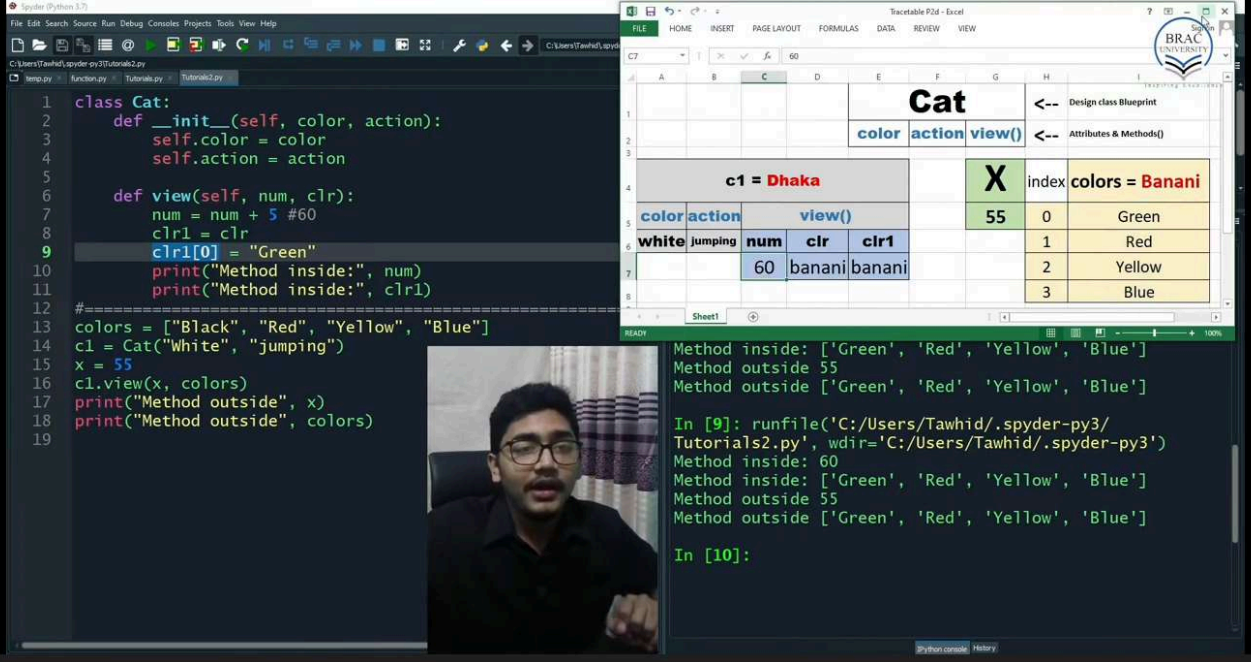
Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

```
Method inside: 60
Method inside: ['Green', 'Red', 'Yellow', 'Blue']
Method outside 55
Method outside ['Green', 'Red', 'Yellow', 'Blue']

In [9]:
```

No connection



No connection

```python

class my_calculator:

    def product(self, *nums):
        pro = 1
        print(nums)
        for x in nums:
            pro = pro * x
        print(pro)




#==========================================================

c1 = my_calculator()
c1.product(4)
c1.product(4, 5)
c1.product(4, 5, 6, 7, 5, 120)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in

```
In [13]: runfile('C:/Users/Tawhid/.spyder-py3/
Tutorials.py', wdir='C:/Users/Tawhid/.spyder-py3')
(4,)
4
(4, 5)
20
(4, 5, 6, 7, 5, 120)
504000

In [14]:
```

No connection

```python

class my_calculator:

    def product(self, num1, num2=None, num3=None):
        if num1 != None and num2 != None and num3 != None:
            print(num1 * num2 * num3)
        elif num1 != None and num2 != None:
            print(num1 * num2)
        else:
            print(1 * num1)

    def product(self, *nums):
        pro = 1
        for x in nums:
            pro = pro * x
        print(pro)




#==========================================================

c1 = my_calculator()
c1.product(4)
c1.product(4, 5)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in

```
In [14]:
```

No connection