

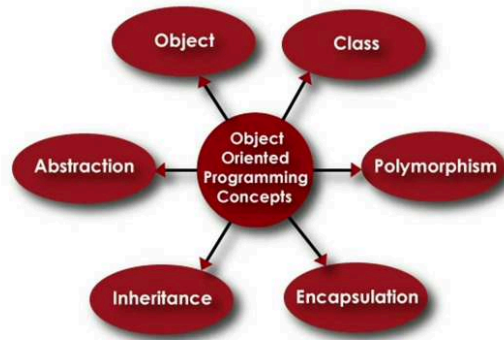
Concept of OOP

- 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

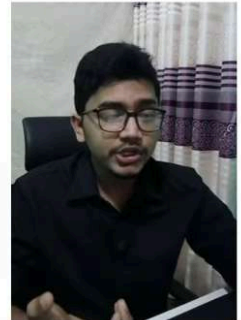
Pillars of OOP



No connection

Class

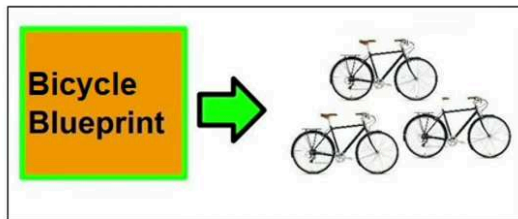
- A class is the blueprint for the objects created from that class
- Each class contains some data definitions(called fields), together with methods to manipulate that data
- When the object is instantiated from the class, an instance variable is created for each field in the class



No connection

Object

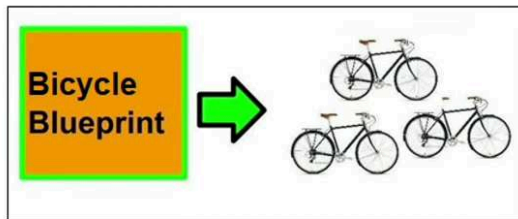
- Object are the basic run time entities in an object oriented system
- It is an instance of class
- We can say that objects are the variables of the type class



No connection

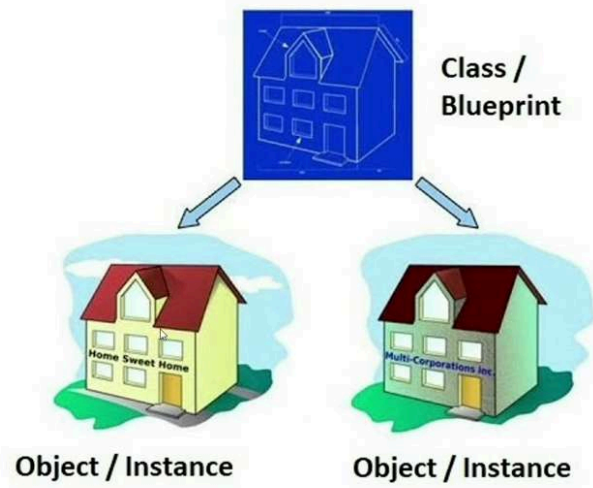
Object

- 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



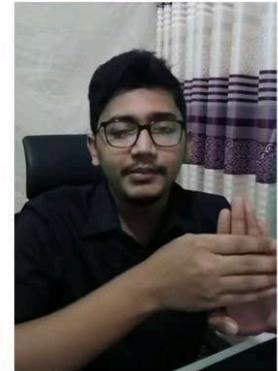
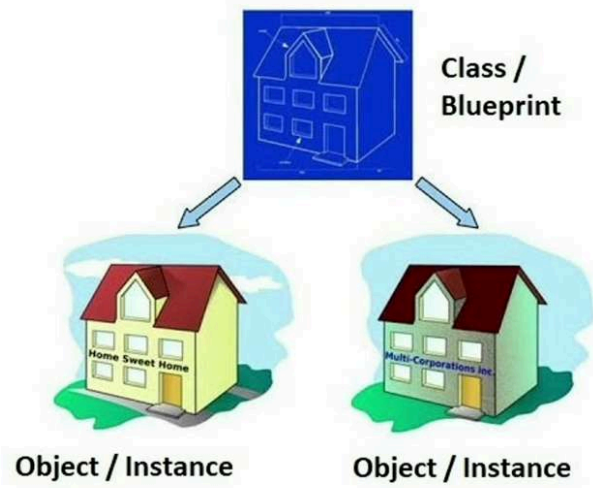
No connection

Class & Object



No connection

Class & Object



No connection

Class Components

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

Data
<ul style="list-style-type: none">• driver_name• num_passenger

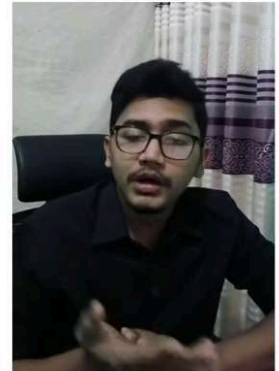
Method()
<ul style="list-style-type: none">• pick_up_passenger()• drop_off_passenger()



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

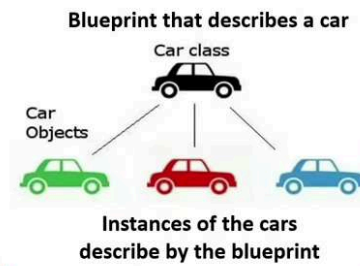
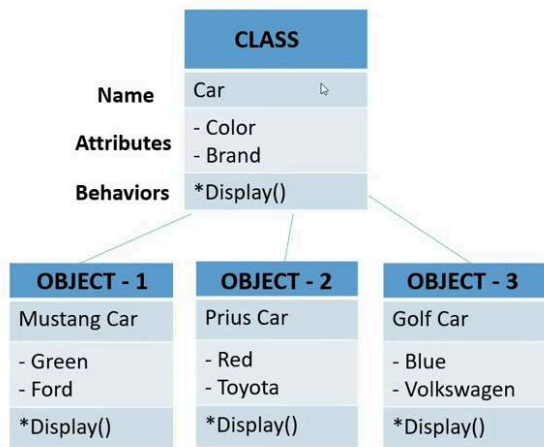
Taxi

- driver_name: **string**
- num_passenger: **int**
- Pick_up_passenger()
- Drop_off_passenger()



No connection

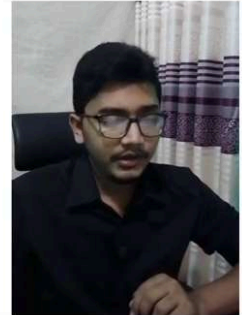
Class and Objects



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__`

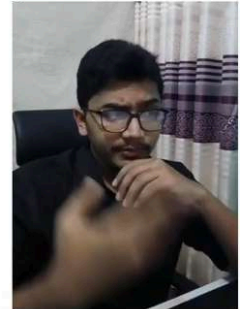
- "`__init__`" is a reserved method in python classes.
- It is known as a constructor in OOP concepts.
- This method is called when an object is created from the class and it allows the class to initialize the attributes of a class.
- It accepts the `self` -keyword as a first argument which allows accessing 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
```

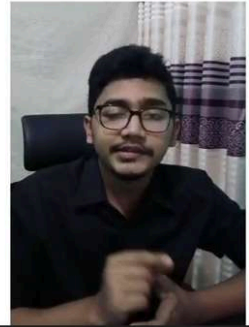
Output:
John created
David created



No connection

Instance Method

- Instance methods 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 methods take more than 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:

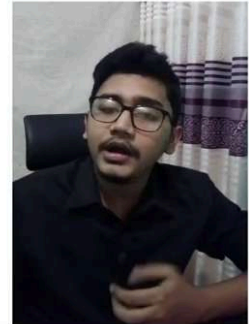
    #parameterized constructor
    def __init__(self, name, no):
        self.no = no        #instance variable
        self.name = name    #instance variable

    #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

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Tawhid\spyder-gp37\Tutorials.py

```
1 """ Class Object Constructor attributes methods() """
2
3 class Student:
4
5     def __init__(self):
6         print(self)
7         print("a student object created")
8
9
10
11 #=====
12
13 #variable = class_name()
14 s1 = Student()
15 s2 = Student()
16
17 print("s1", s1)
18
```


File Explorer

Name	Type	Date Modified
.jupyterproject	File Folder	11/24/2018 6:03 PM
.autosave	File Folder	7/20/2020 8:14 PM
.Cycles	File Folder	7/20/2020 10:08 PM
.config	File Folder	7/8/2020 8:32 PM
.ds	File Folder	10/12/2018 6:55 PM
.defaults	File Folder	11/24/2018 6:22 PM
.ipynb_path	File Folder	7/20/2020 11:37 AM
.ipy_note_path	File Folder	7/8/2020 8:32 PM
.ODP	File Folder	7/21/2020 6:50 PM
.Part1.py	py File	7/20/2020 11:01 PM

Console 1/A

```
a student object created
<__main__.Student object at 0x00000199230C0908>
In [14]: runfile('C:/Users/Tawhid/.spyder-py3/
Tutorials.py', wdir='C:/Users/Tawhid/.spyder-py3')
<__main__.Student object at 0x00000199230BFA08>
a student object created
<__main__.Student object at 0x00000199230BF7C8>
a student object created
<__main__.Student object at 0x00000199230BFA08>
In [15]: runfile('C:/Users/Tawhid/.spyder-py3/
Tutorials.py', wdir='C:/Users/Tawhid/.spyder-py3')
<__main__.Student object at 0x00000199230BF408>
a student object created
<__main__.Student object at 0x00000199230BF188>
a student object created
s1 <__main__.Student object at 0x00000199230BF408>
In [16]:
```

Python console History



No connection

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Tawhid\spyder\py27\Tutorial.py

```
1 """ Class Object Constructor attributes """
2
3 class Student:
4
5     def __init__(self, name, Id):
6         self.name = name #instance variable
7         self.id = Id #instance variable
8         #print("a student object created")
9
10
11
12 #=====
13
14 #variable = class_name()
15 s1 = Student("Bob", 11)
16 s2 = Student("Carol", 22)
17 print(s1.id) #11
18 s1.id = 33
19 print(s1.id) #33
20
21 #print("s1", s1)
22 #print("s2", s2)
23
```

Variables explorer: Help: Plots: Files


Name	Type	Date Modified
.ipynbproject	File Folder	11/24/2018 6:03 PM
.autosave	File Folder	7/23/2020 8:29 PM
.Cicles	File Folder	7/20/2020 10:08 PM
.config	File Folder	7/8/2020 8:32 PM
.ds	File Folder	10/12/2018 6:55 PM
.defaults	File Folder	11/24/2018 6:22 PM
.ipynb_path	File Folder	7/20/2020 10:37 AM
.ipynb_path	File Folder	7/8/2020 8:32 PM
.QDP	File Folder	7/23/2020 6:50 PM
.Part1.py	py File	7/20/2020 11:01 PM

Console 1A

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

In [29]:
```

Python console



No connection

```
1 """ Class Object Constructor attributes methods """
2
3 class Student:
4
5     def __init__(self, name, Id):
6         self.name = name #instance variable
7         self.id = Id #instance variable
8         #print("a student object created")
9
10    def details(self): #instance method
11        print("Name:", self.name, "ID:", self.id)
12
13    #=====
14
15    #variable = class_name()
16    s1 = Student("Bob", 11)
17    s2 = Student("Carol", 22)
18    s1.id= 24
19    s1.details()
20    s2.details()
21    s1.details()
22
23
24    #print("s1", s1)
25    #print("s2", s2)
26
```

Tracetable P2 - Excel

Student		
name	id	details()
s1 = Dhaka	<-- Object1	
name	id	details()
Bob	24	

Design class Blueprint
Attributes & Methods()

s2 = CTG <-- Object2		
name	id	details()
Carol	22	

Sheet1


READY

```
In [6]: runfile('C:/Users/Tawhid/.spyder-py3/Tutorials.py', wdir='C:/Users/Tawhid/.spyder-py3')
Name: Bob ID: 24
Name: Bob ID: 24

In [7]: runfile('C:/Users/Tawhid/.spyder-py3/Tutorials.py', wdir='C:/Users/Tawhid/.spyder-py3')
Name: Bob ID: 24
Name: Carol ID: 22
Name: Carol ID: 22

In [8]: runfile('C:/Users/Tawhid/.spyder-py3/Tutorials.py', wdir='C:/Users/Tawhid/.spyder-py3')
Name: Bob ID: 24
Name: Carol ID: 22
Name: Bob ID: 24

In [9]:
```



No connection

```

1 class Student:      #class / design/ blueprint
2
3     def __init__(self, nm, iD):      #constructor
4         self.name = nm
5         self.Id = iD
6
7
8
9
10 #=====
11 #variable = classname()   ebahbe object create kore
12
13 s1 = Student("Pranto", 1)    #1st object / instance
14 print(s1.name)
15 s1.name = "Roy"
16 print(s1.name)
17
18 print("-----")
19
20 s2 = Student("ABC", 2)    #2nd object / instance
21 print(s2.name)
22

```

Usage

Here you can get help of any object by pressing **Ctrl+H** 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/
cse111_final.py', wdir='C:/Users/Tawhid/.spyder-py3')
Pranto
1
-----

In [13]: runfile('C:/Users/Tawhid/.spyder-py3/
cse111_final.py', wdir='C:/Users/Tawhid/.spyder-py3')
Pranto
-----
ABC

In [14]: runfile('C:/Users/Tawhid/.spyder-py3/
cse111_final.py', wdir='C:/Users/Tawhid/.spyder-py3')
Pranto
Roy
-----
ABC

In [15]:

```


Top: Tutor (Python 3.7)

```
1 class House:
2
3     def __init__(self):
4         self.window = 4 #instancne variable
5         self.door = 2   #instancne variable
6
7     def view(self):
8         print(self.window, "windows", self.door, "doors")
9
10 #=====
11
12 h1 = House()
13 h2 = House()
14 h1.window = 6
15 h2.door = 3
16 h1.view()
17 h2.view()
```

Right: Tractable P2a1 - Excel

	A	B	C	D	E	F	G	H
1			House			<--	Design class Blueprint	
2			window	door	view()	<--	Attributes & Methods()	
3								
4			h1 = Dhaka <-- Object1				h2 = CTG <-- Object2	
5			window	door	view()		window	door
6			4	2			4	2

Bottom Left: In [3]: runfile('C:/Users/Taw... Tutorals2.py', wdir='C:/User...
6 windows 2 doors
4 windows 3 doors

Bottom Right: In [4]:

Bottom Center: No connection

Topyler (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Tawhid\spyder-qt\Tutorial2.py

```
1 class House:
2
3     def __init__(self):
4         self.window = 4 #instancne variable
5         self.door = 2   #instancne variable
6
7     def view(self):
8         print(self.window, "windows", self.door, "doors")
9
10 #=====
11
12 h1 = House()
13 h2 = House()
14 print(h1)
15 print(h2)
16 h1.window = 6
17 h2.door = 3
18 h1.view()
19 h2.view()
```

Tracetable P2a1 - Excel

FILE HOME INSERT PAGE LAYOUT FORMULAS DATA REVIEW VIEW

BRAC UNIVERSITY

	A	B	C	D	E	F	G	H	
			House			<--	Design class Blueprint		
1			window	door	view()	<--	Attributes & Methods()		
2									
3									
4			h1 = Dhaka	<-- Object1			h2 = CTG	<-- Object2	
5			window	door	view()		window	door	view()
6			6	2			4	3	
7									

Sheet1

Tutorials2.py, wdir='C:/Users/Tawhid/.spyder-py3')

```
<__main__.House object at 0x0000020CE435E088>
<__main__.House object at 0x0000020CE435E4C8>
6 windows 2 doors
4 windows 3 doors
```

In [5]:

No connection

```
1
2 class Car:
3
4     def __init__(self, name, model):
5         self.name = name           #instance variable
6         self.model_year = model    #instance variable
7         self.wheel = 4             #instance variable
8
9     def view(self):                 #instance method
10        print("The model year of this", self.name,
11              "is", self.model_year)
12        print("It is a", self.wheel, "wheel car")
13
14    #=====
15
16    c1 = Car("BMW", 2016)
17
18    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](#)

Variable explorer | Help | Plots | Files

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

```
1 class Dog:
2
3
4     def __init__(self, name, color):
5         self.name = name
6         self.color = color
7
8     def update_color(self, color):
9         self.color = color
10
11     def poke(self):
12         print(self.color, self.name, "is smiling")
13
14 #=====
15
16 d1 = Dog("rover", "Brown")
17 d2 = Dog("tomy", "White")
18
19 d1.poke()
20 d1.update_color("Black")
21 d1.poke()
22 #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](#)

Variable explorer Help Plots Files

Console 1(A)

Write something

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

```
In [20]:
```



```
1 class Dog:
2
3
4     def __init__(self, name, color):
5         self.name = name
6         self.color = color
7
8     def update_color(self, color):
9         self.color = color
10
11     def poke(self):
12         print(self.color, self.name, "is smiling")
13
14 #=====
15
16 d1 = Dog("rover", "Brown")
17 d2 = Dog("tomy", "White")
18
19 #d1.poke()
20 d1.update_color("Black")
21 #d1.poke()
22
23 print(d2.__dict__)
24 print(dir(d1))
25
```



No connection

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](#)

Console: JA

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]:

```
1 #from folder_name import class_name
2 import book
3
4 b1 = book.Book("Opekkha", "Humayun Ahmed")
5 b1.set_price(255)
6 b1.details()
7
8
9 #tester class
10
```

Usage

Here you can get help of any object by pressing **Ctrl+H** 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-py3/00P/
Tester1.py', wdir='C:/Users/Tawhid/.spyder-py3/00P')
Reloaded modules: book
Book Name: Opekkha
Author: Humayun Ahmed
Price: 255 Taka
```

```
In [8]:
```



No connection

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Tawhid\spyder-py3\OOP\Tester1.py

```
1 #from folder_name import class_name
2 import book
3
4 b1 = book.Book("Opekkha", "Humayun Ahmed")
5 b1.set_price(255)
6 b1.details()
7
8
9 #tester class
10
```

Usage

Here you can get help of any object by pressing **Ctrl+H** 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: UA

```
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]:
```

Python console



No connection

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Tawhid\spyder-py3\OOP

1 #from folder_name import class_name
2 from OOP import book
3
4 b1 = book.Book("Opekkha", "Humayun Ahmed")
5 b1.set_price(255)
6 b1.details()
7

Usage

Here you can get help of any object by pressing **Ctrl+H** 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*.

Console IJA

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]:

Python console



No connection


```
1
2 class Cat:
3
4     def __init__(self, color, action):
5         self.color = color
6         self.action = action
7
8     def view(self):
9         print(self.color, "cat is", self.action)
10
11    def compare(self, ct):
12        if self.action == ct.action:
13            print("Both cats are", self.action)
14        else:
15            print("They are different")
16
17    #=====
18
19    c1 = Cat("White", "jumping")
20    c2 = Cat("Brown", "jumping")
21    c1.view()
22    c2.view()
23
24    c1.compare(c2)
25
```

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 I/A

they are different

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

```
In [34]:
```



No connection

```
1 class Cat:
2
3
4     def __init__(self, color, action):
5         self.color = color
6         self.action = action
7
8     def view(self):
9         print(self.color, "cat is", self.action)
10
11     def compare(self, ct):
12         if self.action == ct.action:
13             print("Both cats are", ct.action)
14         else:
15             print("They are different")
16
17 #=====
18
19 c1 = Cat("White", "jumping")
20 c2 = Cat("Brown", "jumping")
21 c1.view()
22 c2.view()
23
24 c1.compare(c2)
25
```

Cat				<--	Design class Blueprint		
				<--	Attributes & Methods()		
color		action	view()	cmpr()			
c1 = Dhaka				c2 = CTG			
color	action	view()	cmpr()	color	action	view()	cmpr()
white	jumping		ct	Brown	jumping		ct
			CTG				

```
READY
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

```
1 class Cat:
2     def __init__(self, color, action):
3         self.color = color
4         self.action = action
5
6     def view(self, num, clr):
7         num = num + 5 #60
8         clr = clr
9         clr1[0] = "Green"
10        print("Method inside:", num)
11        print("Method inside:", clr1)
12
13    #=====
14    colors = ["Black", "Red", "Yellow", "Blue"]
15    c1 = Cat("White", "jumping")
16    x = 55
17    c1.view(x, colors)
18    #print("Method outside", x)
19    #print("Method outside", colors)
20
```

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*.

Console 1/A

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

In [8]:



No connection

```

1 class Cat:
2     def __init__(self, color, action):
3         self.color = color
4         self.action = action
5
6     def view(self, num, clr):
7         num = num + 5 #60
8         clr1 = clr
9         clr1[0] = "Green"
10        print("Method inside:", num)
11        print("Method inside:", clr1)
12
13    #=====
14    colors = ["Black", "Red", "Yellow", "Blue"]
15    c1 = Cat("White", "jumping")
16    x = 55
17    c1.view(x, colors)
18    print("Method outside", x)
19    print("Method outside", colors)
20

```

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*.

Console 1/A

```

tutorial12.py , wait= C:/users/tawhid/.spyder-pys
Method inside: 60
Method inside: ['Green', 'Red', 'Yellow', 'Blue']
Method outside 55
Method outside ['Green', 'Red', 'Yellow', 'Blue']

```

In [9]:



No connection

Spysder (Python 3.7)

```
1 class Cat:
2     def __init__(self, color, action):
3         self.color = color
4         self.action = action
5
6     def view(self, num, clr):
7         num = num + 5 #60
8         clr1 = clr
9         clr1[0] = "Green"
10        print("Method inside:", num)
11        print("Method inside:", clr1)
12
13    #=====
14    colors = ["Black", "Red", "Yellow", "Blue"]
15    c1 = Cat("White", "jumping")
16    x = 55
17    c1.view(x, colors)
18    print("Method outside", x)
19    print("Method outside", colors)
```

Tutorial2.py

Excel: Tacttable P2d - Excel

Cat			
color	action	view()	
c1 = Dhaka		X	index
55	0	Green	
1	Red		
2	Yellow		
3	Blue		

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

In [9]: runfile('C:/Users/Tawhid/.spyder-py3/Tutorials2.py', wdir='C:/Users/Tawhid/.spyder-py3')
Method inside: 60
Method inside: ['Green', 'Red', 'Yellow', 'Blue']
Method outside 55
Method outside ['Green', 'Red', 'Yellow', 'Blue']

In [10]:

BRAC UNIVERSITY

Python console History

No connection

```
1 class my_calculator:
2
3
4     def product(self, *nums):
5         pro = 1
6         print(nums)
7         for x in nums:
8             pro = pro * x
9         print(pro)
10
11
12
13
14
15 #=====
16
17 c1 = my_calculator()
18 c1.product(4)
19 c1.product(4, 5)
20 c1.product(4, 5, 6, 7, 5, 120)
```

Usage

Here you can get help of any object by pressing **Ctrl+H** 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

Variable explorer | [Help](#) | [Plots](#) | [Files](#)

```
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


```
1
2
3
4 class my_calculator:
5     def product(self, num1, num2=None, num3=None):
6         if num1 != None and num2 != None and num3 != None:
7             print(num1 * num2 * num3)
8         elif num1 != None and num2 != None:
9             print(num1 * num2)
10        else:
11            print(1 * num1)
12
13    def product(self, *nums):
14        pro = 1
15        for x in nums:
16            pro = pro * x
17        print(pro)
18
19
20
21
22 #=====
23
24 c1 = my_calculator()
25 c1.product(4)
26 c1.product(4, 5)
27 c1.product(4, 5, 6, 7)
```

Usage

Here you can get help of any object by pressing **Ctrl+H** 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

Variable explorer | [Help](#) | [Plots](#) | [Files](#)

In [14]:



No connection