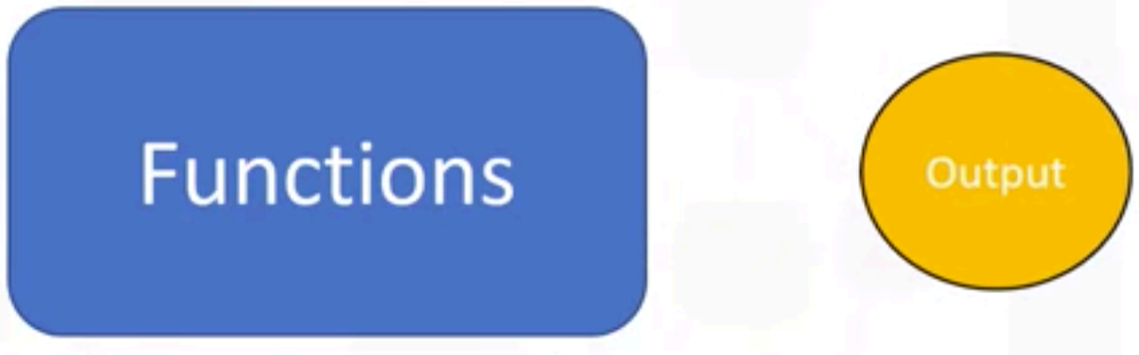
PYTHON
Functions

Functions




```
def  function(a):
    """add 1 to a"""
    b=a+1;

    print(a, " +1 = ",b)
    return b
```

Python Built-in functions

Len

album_ratings = [10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]

L=len(album_ratings)

len

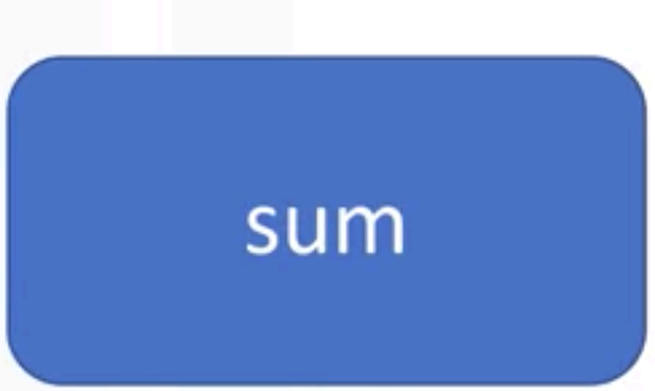
Sum

album_ratings = [10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]

10.0+8.5+9.5+7.0+7.0+9.5+9.0+9.5

S=sum(album_ratings)

S:70



70

Sorted vs Sort

album_ratings = [10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]

sorted_album_rating = sorted (album_ratings)

sorted_album_rating:

[7.0, 7.0, 8.5, 9.0, 9.5, 9.5, 9.5, 10.0]

[7.0, 7.0, 8.5, 9.0, 9.5, 9.5, 9.5, 10.0]

COGNITIVE CLASS.ai

Sorted vs Sort

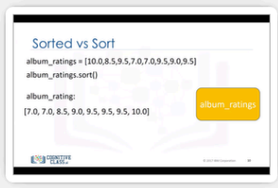
```
album_ratings = [10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]
album_ratings.sort()
```

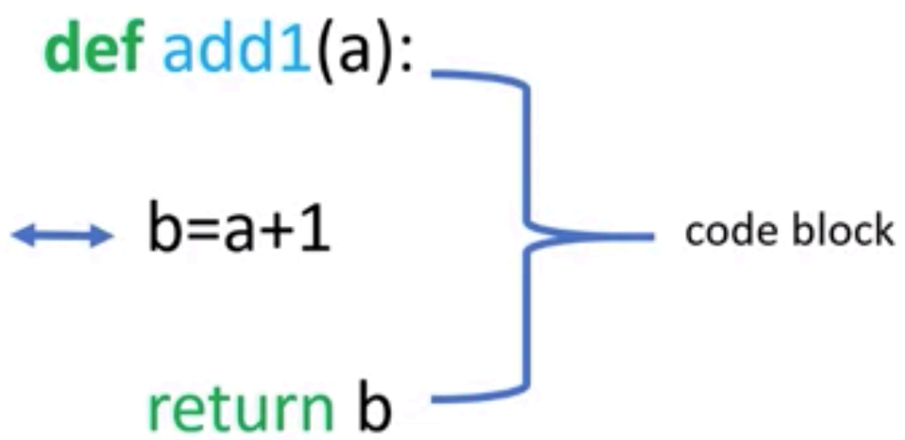
album_rating:

[7.0, 7.0, 8.5, 9.0, 9.5, 9.5, 9.5, 10.0]

album_ratings

Making Functions





```
def add1(a):
    """
    add 1 to a
    """
    b=a+1;
    return b

help(add1)
```

Documentation String

```
Help on function add1 in module __main__: add1(a) add 1 to a
```

Multiple Parameters

- A function can have multiple parameters

```
def Mult(a,b):
        c=a*b
        return c
```

```
Mult(2,3)
```

6

```
Mult(10,3.14)
```

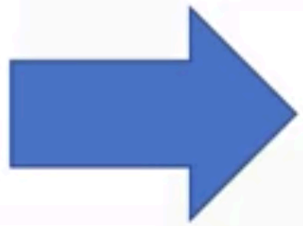
```
def Mult(a,b):
    c=a*b
    return c
```

Mult(2,"Michael Jackson ")

"Michael Jackson Michael Jackson "

```
def  NoWork():
        pass
print(NoWork())
```

None



```
def  NoWork():
        pass
        return None
```

```
def  add1(a):

    b=a+1;

    print(a, “plus 1 equals ”,b)

    return b
```

add1(2)

2 plus 1 equals 3

3

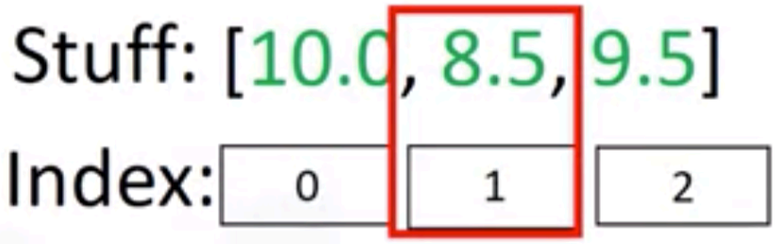
a	2
b	3
output of print(...	2 plus 1 equals 3
value returned	3

```
def  printStuff(Stuff):

    for i,s in enumerate(Stuff):

        print("Album", i ,"Rating is ", s)

album_ratings = [10.0,8.5,9.5]
printstuff(album_ratings)
```




```
Album  0  Rating is  10
Album  1  Rating is  8.5
```

Collecting arguments

```
def  ArtistNames (*names):

    for name in names:

            print(name)


ArtistNames("Michael Jackson","AC/DC")
```

Scope: Local Variables

Global Scope

```
def Thriller():
    Date=1982
    return (Date)

Date=2017

print(Thriller())
```

1982

```
print(Date)
```

2017

Scope Thriller

Date

1982

Date

2017

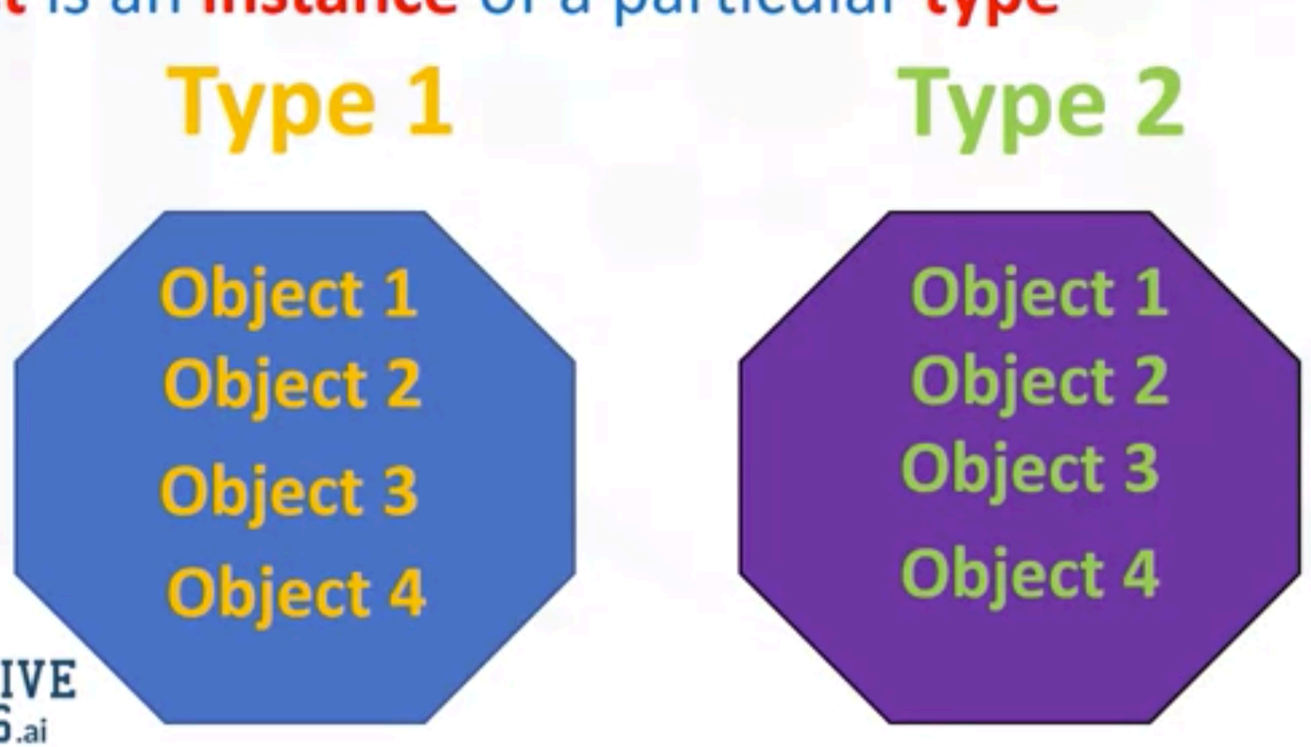
PYTHON
Objects and Classes

Built-in Types in Python

- Python has lots of data types
- Types:
  - int: `1, 2,567..`
  - float: `1.2,0.62..`
  - String: 'a', 'abc', 'The cat is yellow'
  - List: [1, 2, 'abc']
  - Dictionary: {"dog": 1, "Cat": 2}
  - Bool: **`False,True`**
- Each is an **Object**

Built-in Types in Python

- every **object** has:
  - a **type**
  - an internal data representation (a blueprint)
  - a set of procedures for interacting with the object (**methods**)
- an **object** is an **instance** of a particular **type**



Objects: Type

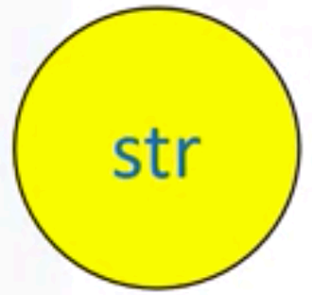
- You can find the type of a object by using the command `type()`

```
>>type([1,34,3])
<class 'list'>
```

Instance of type List


```
>>type('The cat is yellow' )
<class 'str'>
```

Instance of type str




```
>>type(1)
<class 'int'>
```

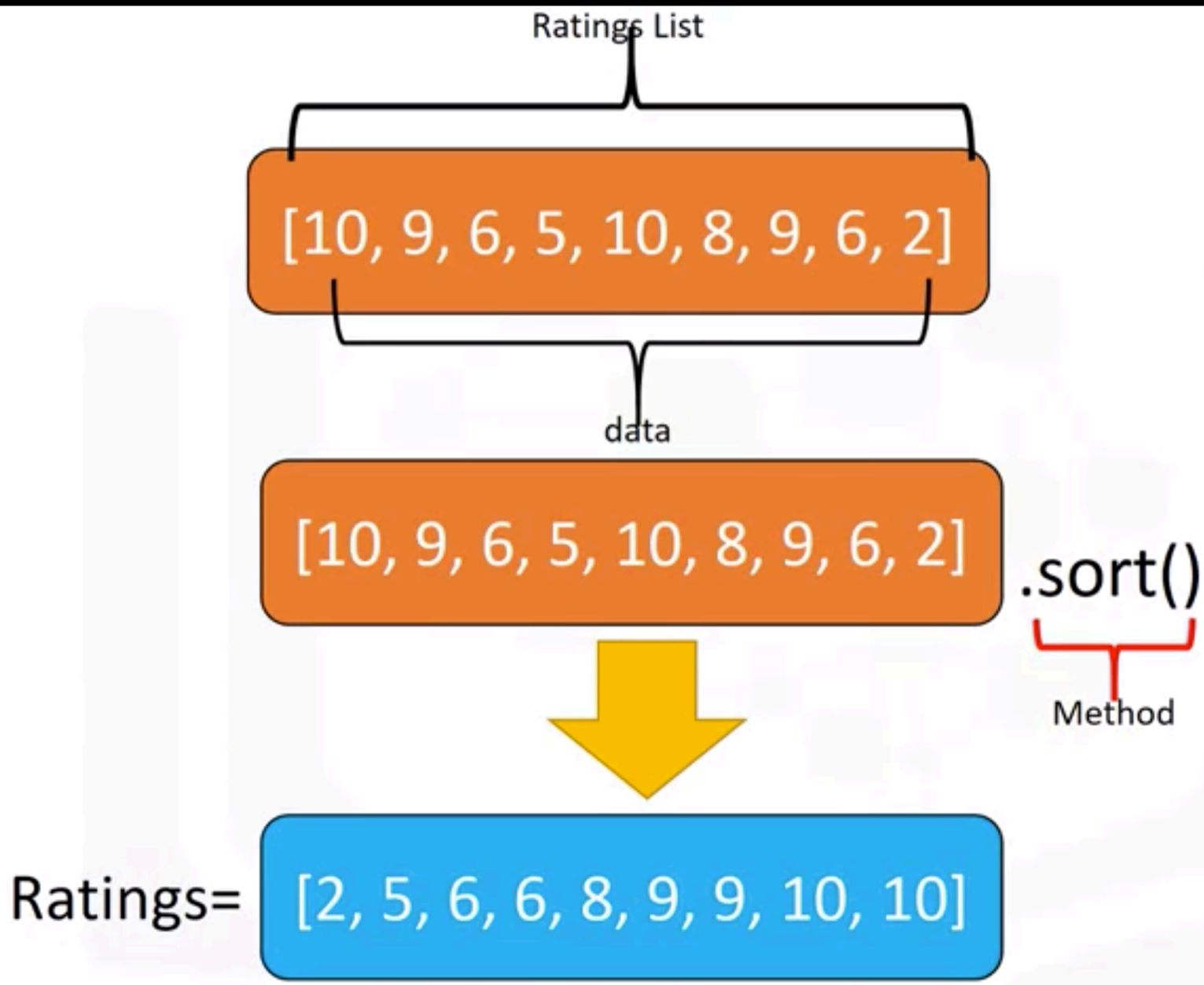
Instance of type int

Methods

- A class or type's methods are functions that every instance of that class or type provides
- It's how you interact with the data in a object
- Sorting is an example of a method that interacts with the data in the object

Ratings=[10,9,6,5,10,8,9,6,2]

Ratings.sort()



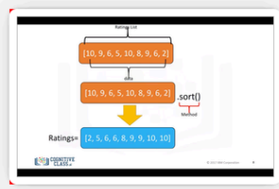
Ratings= [2, 5, 6, 6, 8, 9, 9, 10, 10]

[2, 5, 6, 6, 8, 9, 9, 10, 10] .reverse()

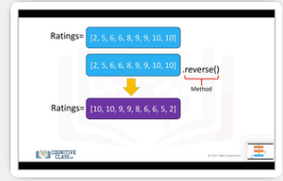
Method

Ratings= [10, 10, 9, 9, 8, 6, 6, 5, 2]

COGNITIVE CLASS.ai

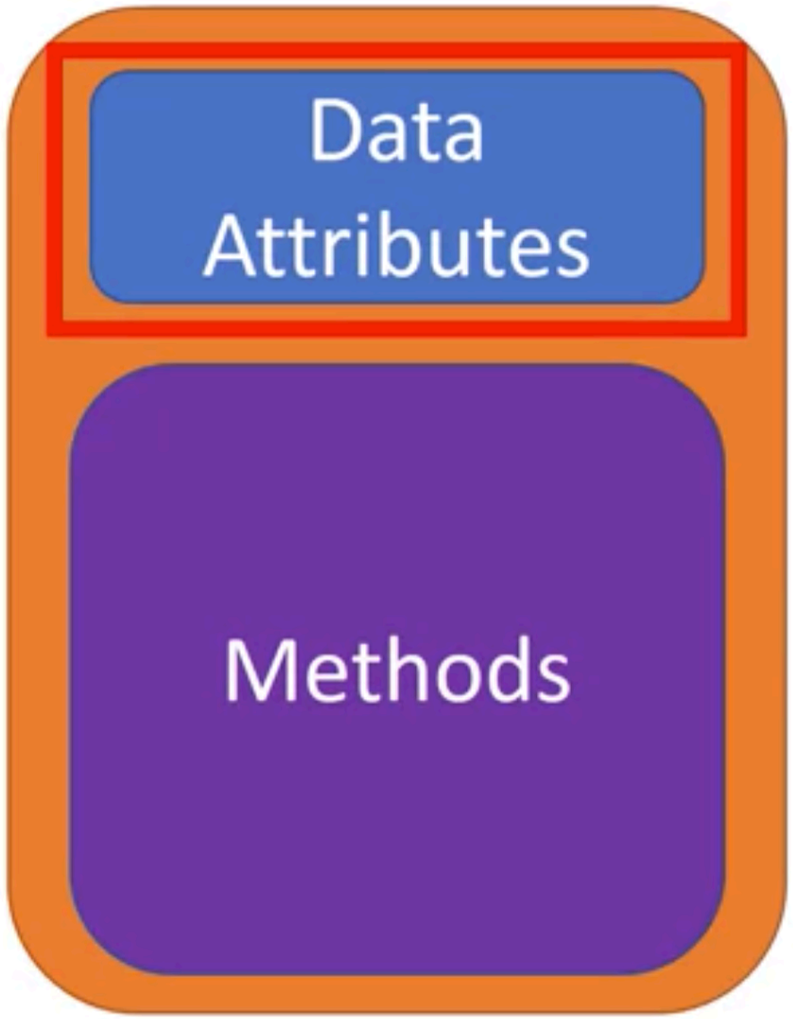


Creating Your Own Types: Defining Classes

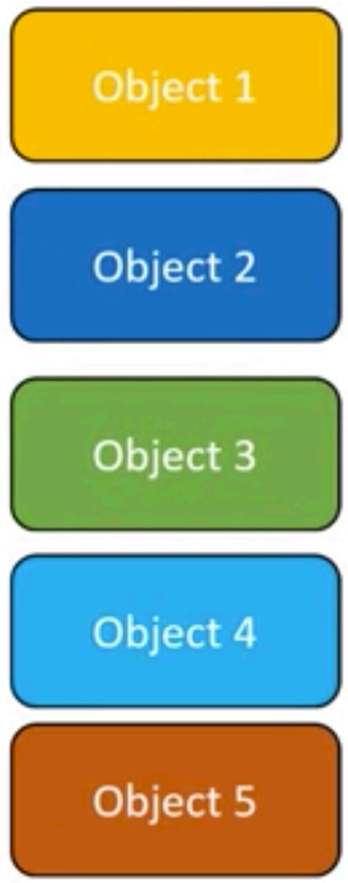


COGNITIVE CLASS.ai

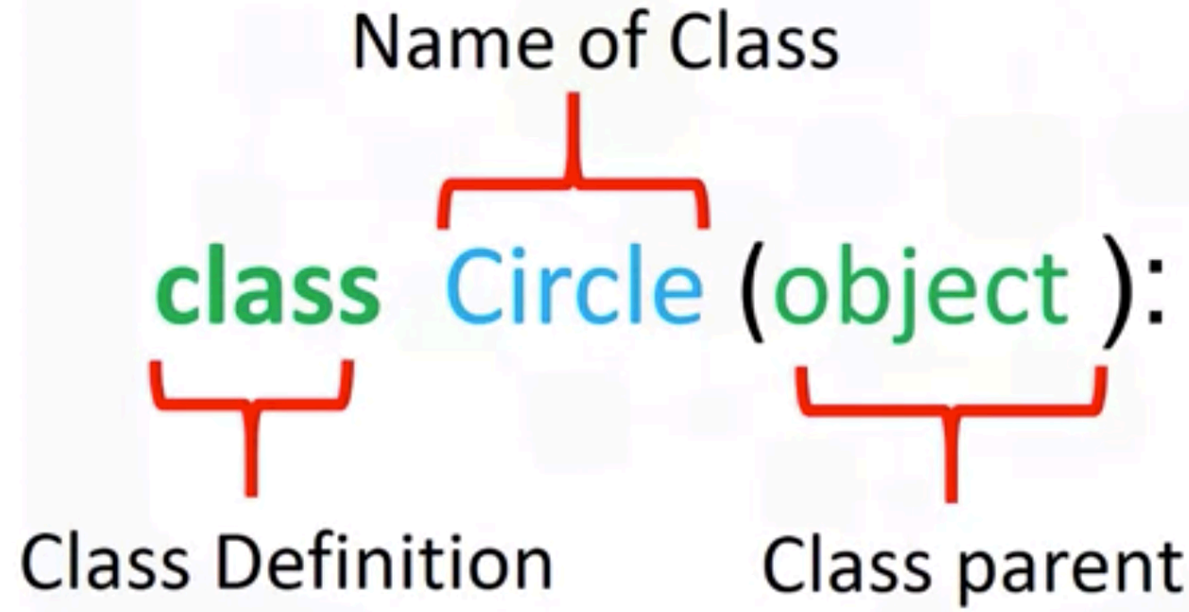
Class



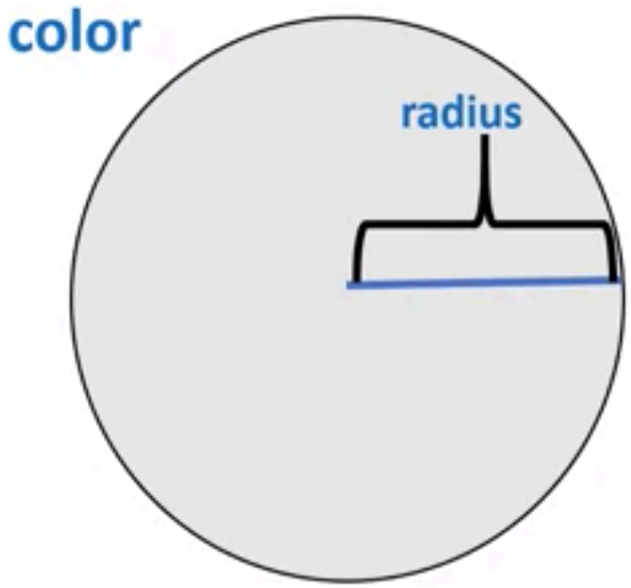
Objects or Instances of that Class



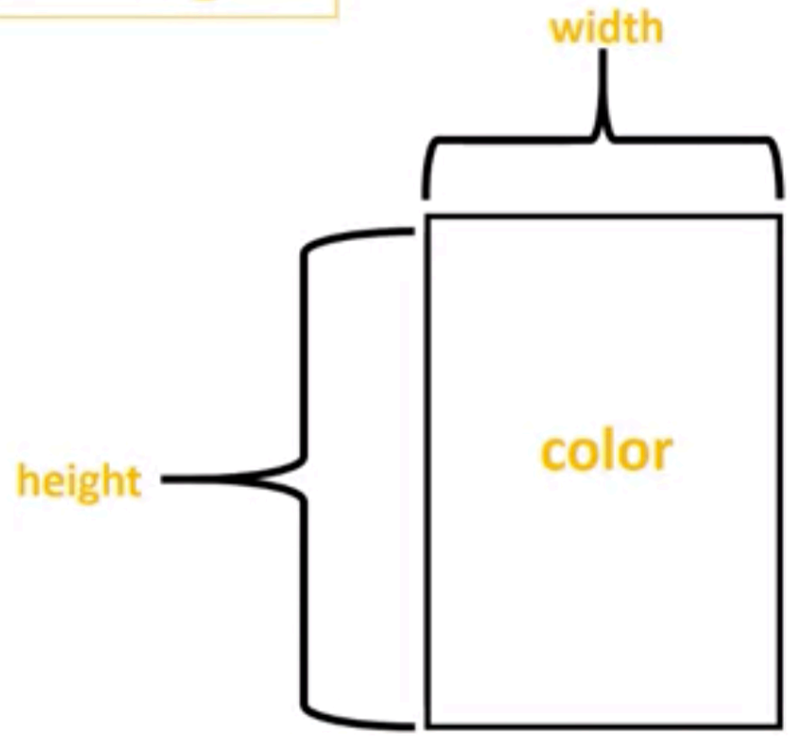
Create a class: Circle



Class Circle

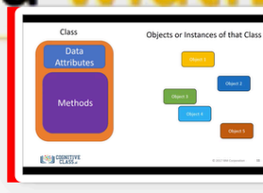


Class Rectangle



Data Attributes: radius, color

Data Attributes: color, height and width



COGNITIVE CLASS.ai

Create a class: Circle

```
class  Circle (object ):
def   __init__(self, radius , color):
    self.radius = radius;
    self.color = color;
```

Define your class

Data attributes used to Initialize each instance of the class

Create a class: Circle

special method or constructor used to initialize data attributes

parameters

```
def __init__(self, radius , color):
    self.radius = radius;
    self.color = color;
```

The self parameter

Create a class: Rectangle

```
class  Rectangle (object ):

    def   __init__(self, color, height , width):
        self.height = height;
        self.width = width
        self.color = color;
```

Define your class

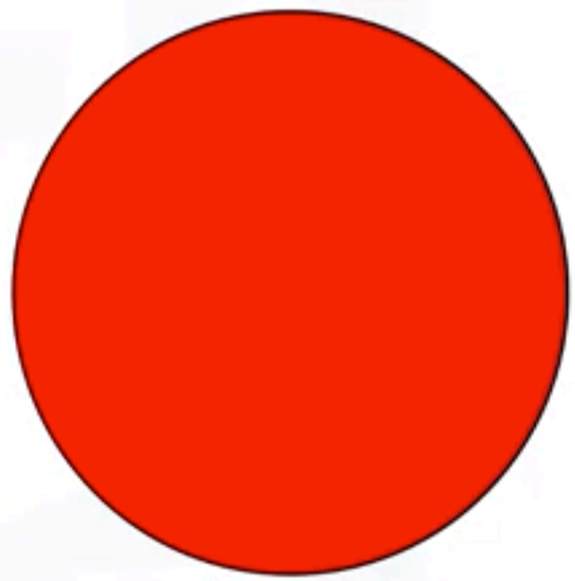
Initialize the object's Data attributes

Create an Instance of a Class: Circle

C1=Circle (10,'red')

```
class  Circle (object ):

    def__init__(self,  10,  'red'):
            self.radius = 10;
            self.color = 'red';
```



self .radius = 10;
self. color = 'red';

Create an Instance of a Class: Circle

C1=Circle (10, “red”)

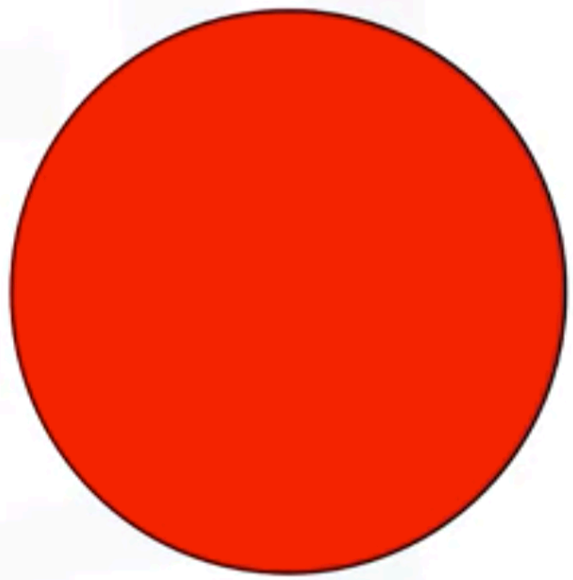
C1.radius

10

C1.color

“red”

self → self .radius, self. color



self.radius = 10;
self.color = ‘red’;

Methods

Create a class: Circle

```
class  Circle (object ):
    def   __init__(self, radius , color):
        self.radius = radius;
        self.color = color;
def  add_radius(self,r):
        self.radius= self.radius +r
```

Method used to add r to radius

Create a class: Circle

```
class  Circle (object ):
    def  __init__(self, radius=3 , color='red'):
        self.radius = radius;
        self.color = color;

def  add_radius(self,r):
        self.radius= self.radius +r


def  drawCircle(self):
```

We can add default values for parameters

New method

Create a class: Circle

Name of object

dir (Nameofobject):

```
['__class__',
 '__delattr__',
 '__dict__',
 '__doc__',
 '__format__',
 '__getattribute__',
 '__hash__',
 '__init__',
 '__module__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'add_radius',
 'color',
 'drawCircle',
 'radius']
```

COGNITIVE CLASS.ai

PYTHON

Reading files with open

```
File1 = open(“/resources/data/Example2.txt",”w")
```

Mode

```
with open("Example1.txt","r") as File1:

    file_stuff=File1.read()

    print(file_stuff)
```

End of block, close file object

```
print(File1.closed)
print(file_stuff)
```

```
print(file_stuff)
This is line 1
This is line 2
This is line 3
```

file_stuff:

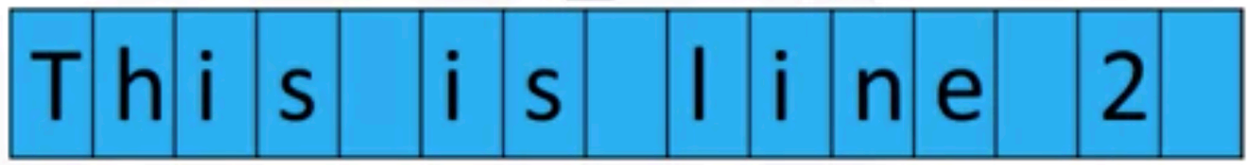
This is line 1 \n This is line 2 \n This is line 3

```
with open("Example1.txt","r") as File1:

    file_stuff=File1.readlines()
    print(file_stuff)
```

file_stuff[0]

file_stuff[1]

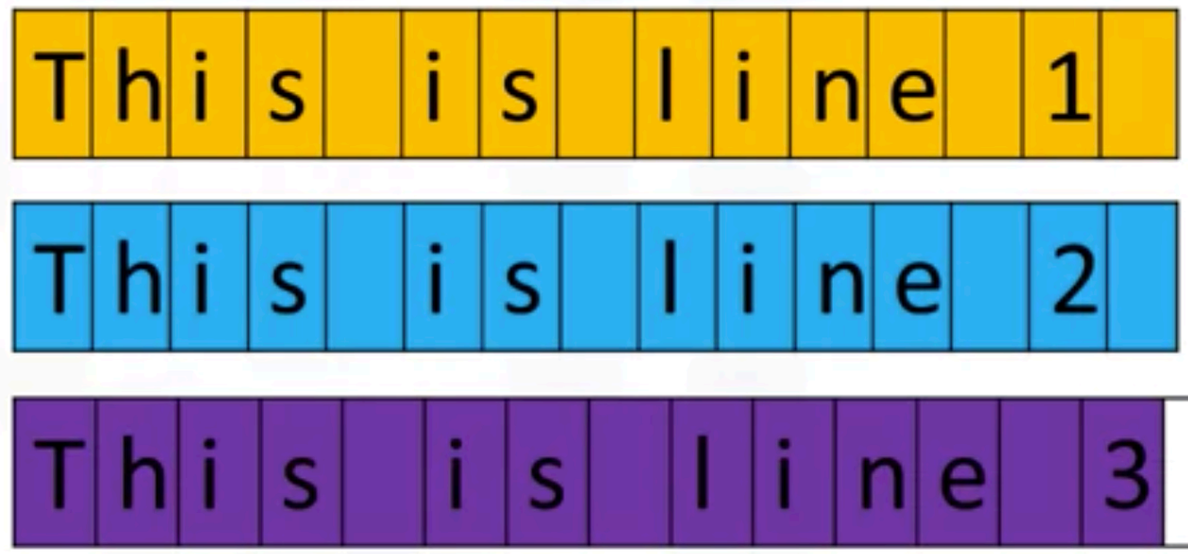


file_stuff: ['This is line 1 \n', 'This is line 2 \n', 'This is line 3']

```
with open("Example1.txt","r") as File1:

    for line in File1:
        print(line)
```

```
This is line 1
This is line 2
```



```
with open("Example1.txt","r") as File1:

    file_stuff=File1.readlines(16)
    print(file_stuff)
    file_stuff=File1.readlines(5)
    print(file_stuff)
    file_stuff=File1.readlines(9)
    print(file_stuff)
```

T	h	i	s		i	s		l	i	n	e		1	

T	h	i	s	

i	s		l	i	n	e		2

T	h	i	s		i	s		l	i	n	e		1	

T	h	i	s		i	s		l	i	n	e		2	

T	h	i	s		i	s		l	i	n	e		3	

COGNITIVE CLASS.ai

PYTHON

Writing files with open

```
with open(“/resources/data/Example2.txt”, “w”) as File1:
```

```
File1.write ("This is line A\n" )
File1.write (“This is line B\n”)
```

This is line A
This is line B

Example2.txt

```
Lines=["This is line A\n","This is line B\n","This is line C\n"]
with open(“/resources/data/Example2.txt”, "w") as File1:
    for line in Lines:
```

File1.write(line)

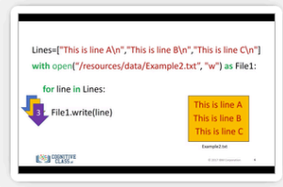
This is line A
This is line B
This is line C

Example2.txt

```
with open(“/resources/data/Example2.txt”, “a”) as File1:
    File1.write ("This is line C" )
```

```
This is line A
This is line B
This is line C
```

Example2.txt



```
with open(“Example1.txt”, ”r") as readfile :
    with open(“Example3.txt”, ”w") as writefile:
            for line in readfiles:
                    writefile.write(line)
```

This is line A
This is line B
This is line C

Example1.txt

This is line A
This is line B

Example3.txt