14.1 Procedural vs Object Oriented Programming

- Large software can be built in C. e.g: Linux Kernel
- OOP: Easier to design, reuse components and build large software
- SDE: should know OOP as they will be working in teams building large software
- ML & DS roles: Should know the basics to use various modules and extend them when needed.
- The true power of OOP and OOD can be understood when designing a large piece of software like a library for plotting

14.2 Classes and Objects (real-world example: the TinyURL service)

```
3 import random
4 import string
5 d = dict();
6 # given a long URL, get a short URL
7 def getShortURL(longURL):
8     # length = random value in 6-10
9     l = random.randint(6,10);
10
11     # generate random characters into a string of length l
12     chars = string.ascii_lowercase
13     shortURL = ''.join(random.choice(chars) for i in range(l))
14
15     # check if this string is already present in dict d
16     if shortURL in d:
17         return getShortURL(longURL);
18     else:
19         d[shortURL] = longURL;
20
21
22     r = "https://www.shortURL.com/"+shortURL
23     return r;
24
25 def getLongURL(shortURL):
26
27     # extarct key from URL https://www.shortURL.com/mxzmuis ---> mxzmuis
28     k = shortURL[25:];
29
30     if k in d:
31         return d[k];
32     else:
33         return None;
```

- We have defined a dictionary two functions getLongURL( ) and getShortURL( ) as shown above.
- A class is basically a logical grouping of all the related functions and members together,we convert the above procedural implementation into object oriented implementation as shown below.

```
# Class: group all variables/attributes and functions/methods into a single logical unit

class ShortURL:

    def __init__(self): # constructor; not must, but, good to have;  initialize all attributes here
        self.d=dict();

    # given a long URL, get a short URL
    def getShortURL(self, longURL): # first argument to all methods is "self" => this object
        # length = random value in 6-10
        l = random.randint(6,10);


        # generate random characters into a string of length l
        chars = string.ascii_lowercase
        shortURL = ''.join(random.choice(chars) for i in range(l))


        # check if this string is already present in dict d
        if shortURL in self.d:
            return getShortURL(longURL);
        else:
            self.d[shortURL] = longURL;


        r = "https://www.shortURL.com/"+shortURL
        return r;

    def getLongURL(self, shortURL):

        # print(self.d); # print statemnt for debugging

         # extarct key from URL https://www.shortURL.com/mxzmuis ---> mxzmuis
         k = shortURL[25:];


         if k in self.d:
             return self.d[k];
         else:
             return None;

```

Timestamp 2.39

Class: datatype

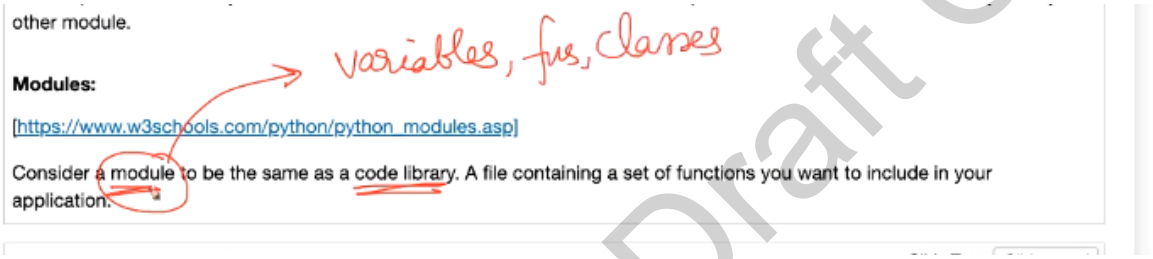
Object: variable

self represents the instance of the class. By using the “self” keyword we can access the attributes and methods of the class in python. It binds the attributes with the given arguments.

```
# Class: datatype/DS & Object: variable of a class
s = ShortURL() # constructor being called; memory allocated.

print(type(s))

```

What does this "*_main_*" mean?

The script invoked directly is considered to be in the **main** module. It can be imported and accessed the same way as any other module.



A module will have variables,functions,classes.

Modules:

Consider a module to be the same as a code library. A file containing a set of functions you want to include in your application.

```
1 print(s.shortURL("appliedaicourse"))
2 print(s.shortURL("gate.appliedcourse.com"))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-52-84023b7d1bbb> in <module>
----> 1 print(s.shortURL("appliedaicourse"))
      2 print(s.shortURL("gate.appliedcourse.com"))

AttributeError: 'ShortURL' object has no attribute 'shortURL'
```

SEARCH STACK OVERFLOW

```
1 print(s.getShortURL("appliedaicourse.com"))
2 print(s.getShortURL("gate.appliedcourse.com"))
3
```

https://www.shortURL.com/alrpoy
https://www.shortURL.com/dlfjxuxhff

Timestamp 19.45

Using the object we have created we call the function as shown above.

14.3 Empty Classes and documentation

In [58]:

Slide Type Slide

```
# No need to have any atributes or methods
class emptyClass:
    pass

e = emptyClass();
print(type(e))
```

```
<class '__main__.emptyClass'>
```

Slide Type Slide

- We can create empty classes in python as shown above.
- There are lots of internals and boundary cases for which reading documentation is a good idea.
- https://docs.python.org/3/tutorial/classes.html
- We will focus more on applied aspects in the context of ML/AI.

14.4 Class Variables and Private Members

Private members of the class are denied access from the environment outside the class. They can be handled only from within the class.
Public members (generally methods declared in a class) are accessible from outside the class. The object of the same class is required to invoke a public method.

```
class ShortURL1:

    URLPrefix = "https://www.shortURL.com/"; # class variable shared by all objects

    def __init__(self): # constructor; not must, but, good to have;  initialize all attributes here
        self.d=dict();

    # given a long URL, get a short URL
    def getShortURL(self, longURL): # first argument to all methods is "self" => this object
        # length = random value in 6-10
        l = random.randint(6,10);


        # generate random characters into a string of length l
        chars = string.ascii_lowercase
        shortURL = ''.join(random.choice(chars) for i in range(l))


        # check if this string is already present in dict d
        if shortURL in self.d:
            return getShortURL(longURL);
        else:
            self.d[shortURL] = longURL;


        r = self.URLPrefix + shortURL
        return r;

    def getLongURL(self, shortURL):

       # print(self.d); # print statemnt for debugging
```

```

        # extarct key from URL https://www.shortURL.com/mxzmuis ---> mxzmuis
        k = shortURL[25:];


        if k in self.d:
            return self.d[k];
        else:
            return None;

```

Timestamp 1.35

The variable URLPrefix is a class variable

```
s1 = ShortURL1();

print(s1.getShortURL("appliedaicourse.com"))
print(s1.getShortURL("gate.appliedaicourse.com"))
```

```
https://www.shortURL.com/bqrkdpum
https://www.shortURL.com/jaxgluaeoh
```

9]: Slide Type Slide

```
print(s1.d)
print(s1.URLPrefix)
```

```
{'bqrkdpum': 'appliedaicourse.com', 'jaxgluaeoh': 'gate.appliedaicourse.com'}
https://www.shortURL.com/
```

1]: Slide Type Slide

```
s1a = ShortURL1();
print(s1a.URLPrefix);
print(s1a.d)
```

```
https://www.shortURL.com/
{}
```

Timestamp 3.21

Because URLprefix is a property of the class it can be shared by all objects of that class.

```
classVar = "test"; # Public => accesible directly from outside the class

__URLPrefix = "https://www.shortURL.com/"; # "Private" memebers. Member => attribute or method
```

Timestamp 8.47

- Adding two underlines(__) at the beginning makes a variable or a method **private** is the convention used by most **python** code
- If we convert URLprefix to a private variable we cannot access it from outside of class as shown below.

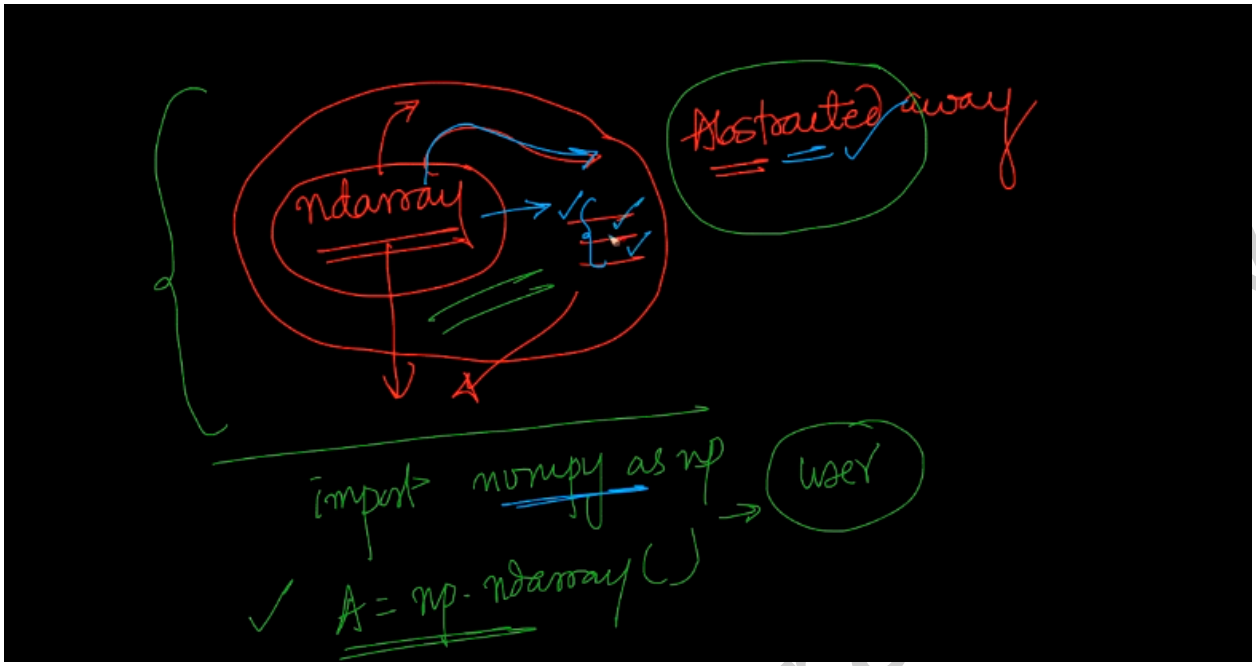
```
 1 s2 = ShortURL2();
 2 print(s2.classVar)
 3 print(s2.__URLPrefix)
```

```
test
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-90-4fd735b2fbbe> in <module>
      1 s2 = ShortURL2();
      2 print(s2.classVar)
----> 3 print(s2.__URLPrefix)

AttributeError: 'ShortURL2' object has no attribute '__URLPrefix'
```

SEARCH STACK OVERFLOW

14.5 Abstraction and Inheritance (with real-world examples)

- As a recommendation to the programmer, in its formulation by Benjamin C. Pierce in Types and Programming Languages (2002), the abstraction principle reads: “ Each significant piece of functionality in a program should be implemented in just one place in the source code. Where similar functions are carried out by distinct pieces of code, it is generally beneficial to combine them into one by abstracting out the varying parts. ”
- General concept in programming: libraries, classes
- "Its main goal is to handle complexity by hiding unnecessary details from the user. That enables the user to implement more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden complexity." [Source: https://stackify.com/oop-concept-abstraction/]



We import numpy and we create ndarray using it ,we don't really need to understand how numpy is implemented the variables and functions declared it numpy and internals of how nd array is implemented .The mathematical operations of ndarray are fairly complex and these things are abstracted away from user(as a programmer we don't need to know about this)

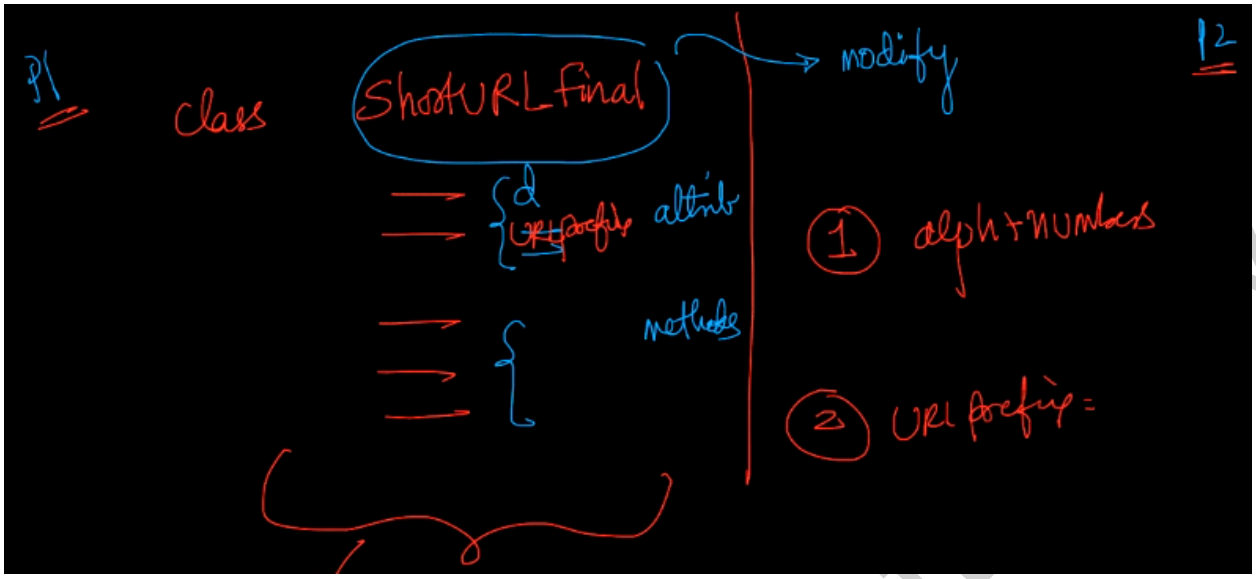
Encapsulation:

In object-oriented programming languages, and other related fields, encapsulation refers to one of two related but distinct notions, and sometimes to the combination thereof:

1. A language mechanism for restricting direct access to some of the object's components.
2. A language construct that facilitates the bundling of data with the methods (or other functions) operating on that data.

Design a MyURLShortner built on top of ShortURLFinal (given in a module) with some changes

- Change getShortURL to have both alphabets and numbers
- Change the URLPrefix to my website (myurlshortner.com)

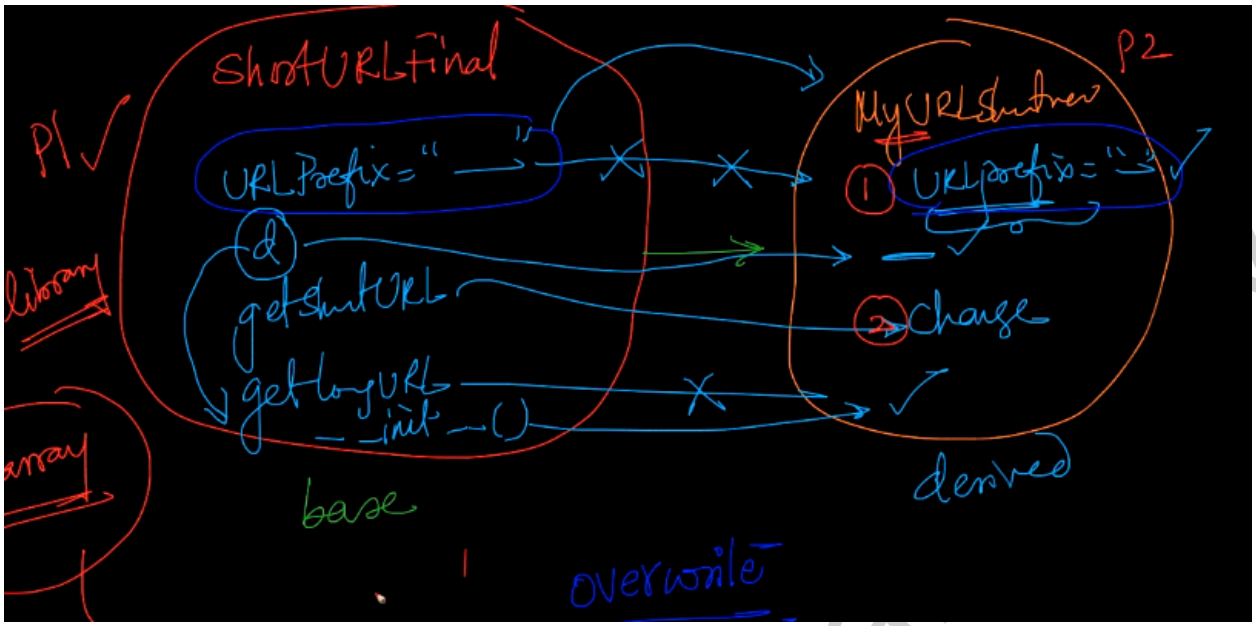


Timestamp 10.37

- Let's say we have implemented a class as shown and now we want to do some modifications as specified.We use inheritance for that as shown below.
- We derive all the functionalities in base class into our derived class.We do this by specifying base class in function definition as shown below .

```
# Lets define MyURLShortner based on ShortURLFinal

# Inheritence: baseclass, derived class [https://docs.python.org/3/tutorial/classes.html#inheritance]
class MyURLShortner(ShortURLFinal):
    URLPrefix = "www.myurlshortner.com/" # overriding as same name as base-class


    # given a long URL, get a short URL
    def getShortURL(self, longURL): # overriding as same name as base-class, use both digits and lowercase
        # length = random value in 6-10
        l = random.randint(6,10);


        # generate random characters into a string of length l
        chars = string.ascii_lowercase + string.digits # both digits and lowercase
        shortURL = ''.join(random.choice(chars) for i in range(l))


        # check if this string is already present in dict d
        if shortURL in self.d:
            return getShortURL(longURL);
        else:
            self.d[shortURL] = longURL;


        r = self.URLPrefix + shortURL
        return r;

    # getLongURL and dict "d" not changed
```

Here MyURLShortner is derived from ShortURLFinal base class.



Timestamp 19.08

As shown above we derive the properties and methods from base class and we override the methods that we want to modify.

```
m1 = MyURLShortner(); # base-class constructor is executed

print(m1.d)

```

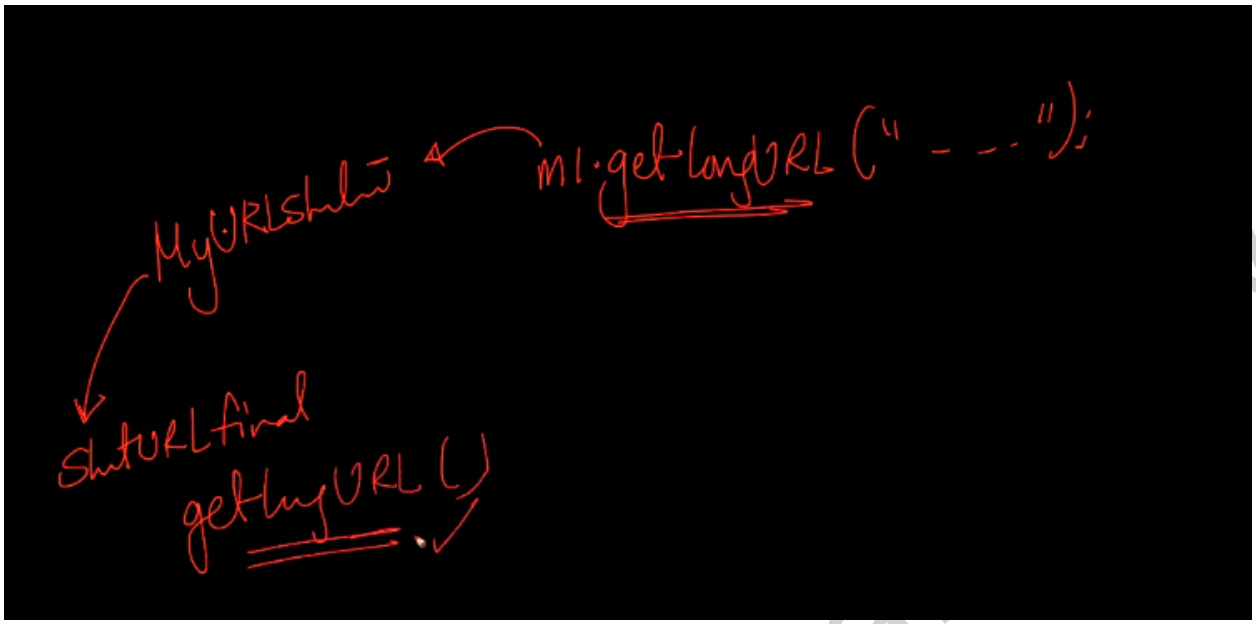
```
{}
```

```
print(m1.getShortURL("amazon.com"))
print(m1.getShortURL("google.com"))
```

```
www.myurlshortner.com/mt2e9h
www.myurlshortner.com/ymnsyufq3
```

```
print(m1.d)
```

```
{'mt2e9h': 'amazon.com', 'ymnsyufq3': 'google.com'}
```



Timestamp 23.25

When we create an object m1 and since we haven't defined a constructor in derived class the constructor in base class gets executed as shown above.

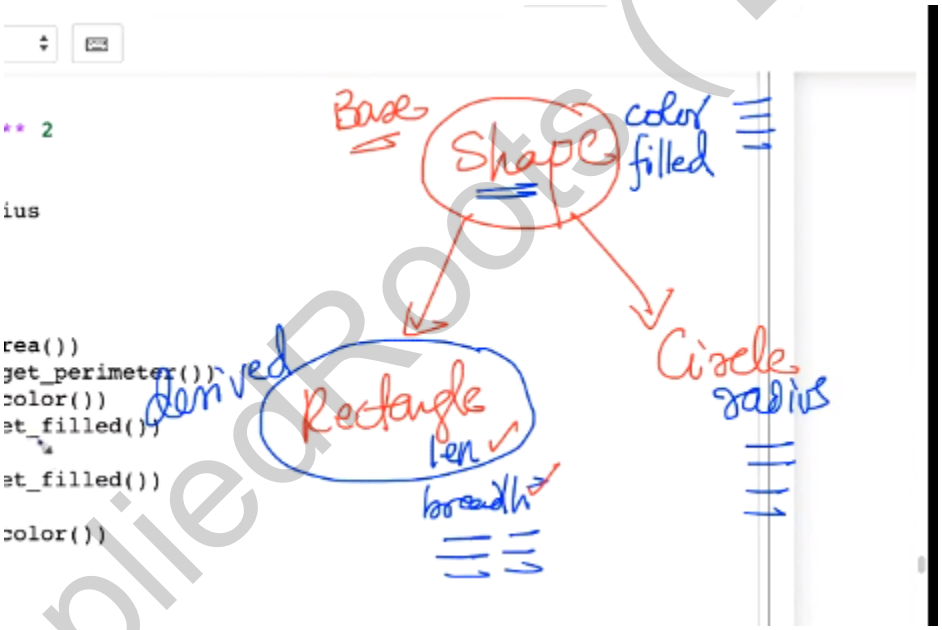
14.7 Example: OOP for representing Shapes

Below is an example of object oriented programming for representing shapes.

```python
# example from geometry: [Source: https://overiq.com/python-101/inheritanc
import math

class Shape:

    def __init__(self, color='black', filled=False):
        self.__color = color
        self.__filled = filled

    def get_color(self):
        return self.__color

    def set_color(self, color):
        self.__color = color

    def get_filled(self):
        return self.__filled

    def set_filled(self, filled):
        self.__filled = filled


```

```
23 class Rectangle(Shape):
24
25     def __init__(self, length, breadth):
26         super().__init__()
27         self.__length = length
28         self.__breadth = breadth
29
30     def get_length(self):
31         return self.__length
32
33     def set_length(self, length):
34         self.__length = length
35
36     def get_breadth(self):
37         return self.__breadth
38
39     def set_breadth(self, breadth):
40         self.__breadth = breadth
41
42     def get_area(self):
43         return self.__length * self.__breadth
44
45     def get_perimeter(self):
46         return 2 * (self.__length + self.__breadth)
47
```

```
49 class Circle(Shape):
50     def __init__(self, radius):
51         super().__init__()
52         self.__radius = radius
53 
54     def get_radius(self):
55         return self.__radius
56 
57     def set_radius(self, radius):
58         self.__radius = radius
59 
60     def get_area(self):
61         return math.pi * self.__radius ** 2
62 
63     def get_perimeter(self):
64         return 2 * math.pi * self.__radius
65 
```



Timestamp 6.56

- Classes Rectangle,Circle inherited from base class Shape

```python
### object is the base class for all classes in Python

class MyClass(object): # object is the base-class by default and implicitly.
    pass


# __new__() : creates a new object and calls the __init__()
# __init__() : default constructor
# __str__() : write code to convert object into string for printing.
```

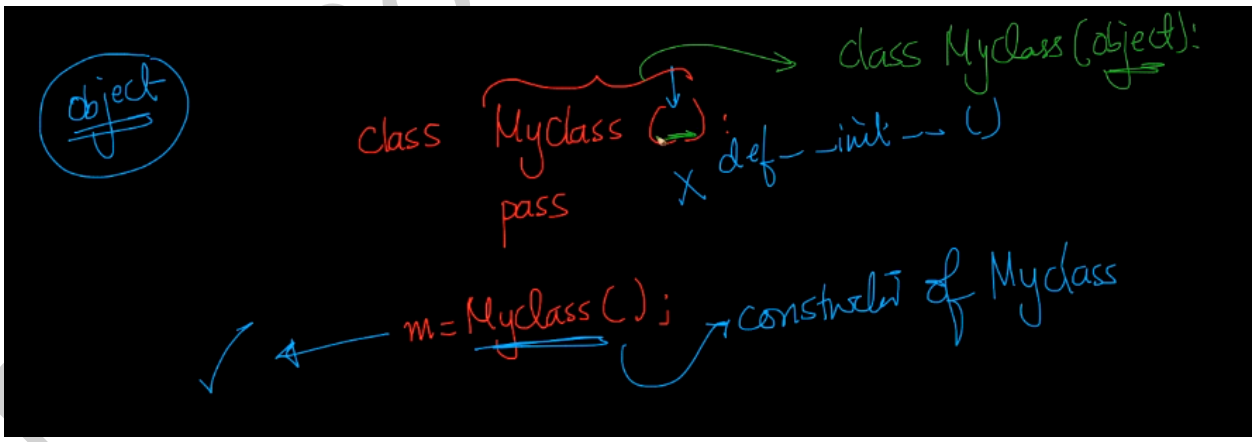
```python
a = [1,2,3,4];
print(type(a))

print(a)
```

```
<class 'list'>
[1, 2, 3, 4]
```

```python
class ClassWithStr(): #default object is the base class
    def __str__(self):
        return "any string representation of the object of this class that we want"

c1 = ClassWithStr();
print(c1)
```

```
any string representation of the object of this class that we want
```

Timestamp 7.58



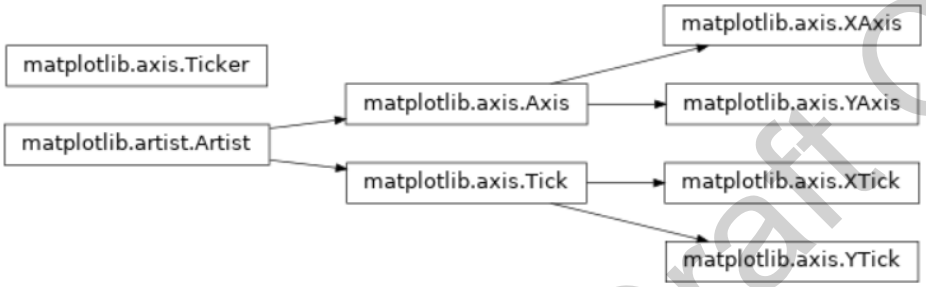
Timestamp 10.26

- Objectclass is baseclass for all the classes in python,every class in python inherits from object class
- Object class has the below functionalities .

1. `__new__() : creates a new object and calls the __init__()`
2. `__init__() : default constructor`
3. `__str__() : write code to convert object into string for printing`

Classes for the ticks and x and y axis.

Inheritance



- The above flowchart gives a nice intuition about inheritance.

14.8 How is OOP typically used in an ML role:

- Using existing Classes.
- Reading documentation to understand how to use a function/class/module.
- Fixing code bugs and understanding error messages.
- Extending existing classes to modify some functionality in an existing class
- Working with Software engineers to build some ML classes for them to use in the larger software.
- Do not perform OOD without understanding it well. Typically done by senior engineers/architects. A good beginner's book: https://learning.oreilly.com/library/view/head-first-design/0596007124/

14.9 Multiple Inheritance

https://docs.python.org/3/tutorial/classes.html#multiple-inheritance

class DerivedClassName(Base1, Base2, Base3):

....

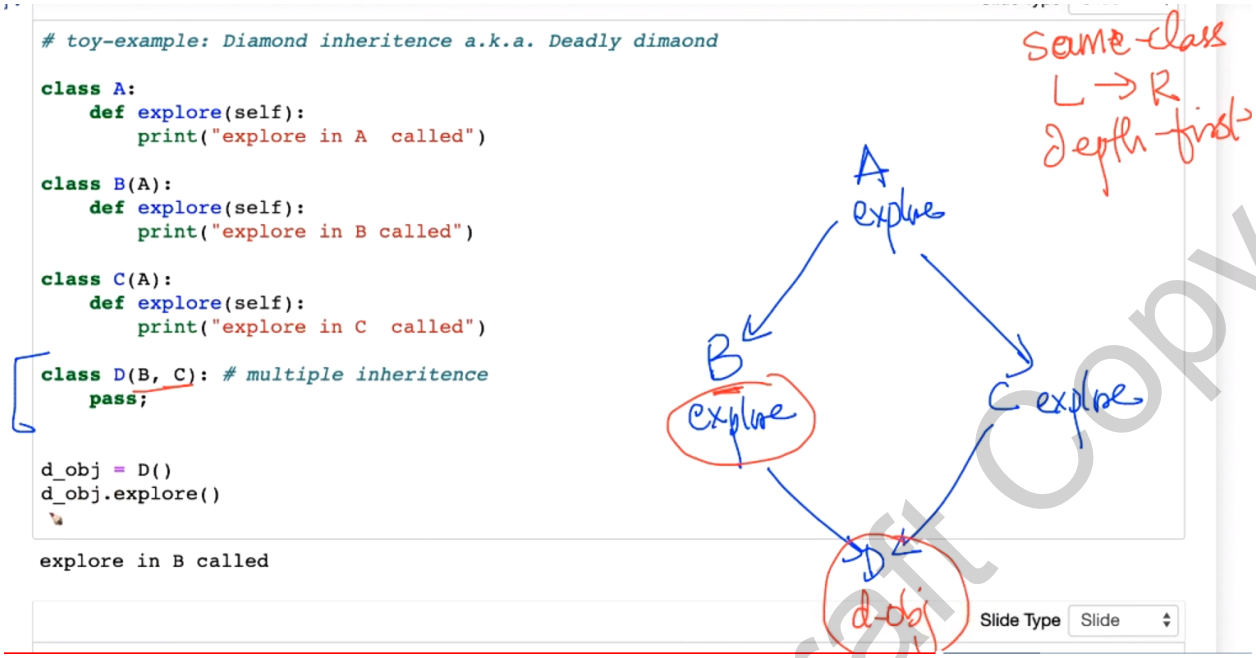
```
# toy-example: Modifications on https://overiq.com/python-101/inheritance-and-polymorphism-in-python/

class A:
    def explore(self):
        print("explore in A  called")
class B:
    def search(self):
        print("search in B  called")

    def explore(self):
        print("explore in B called")

class C:
    def discover(self):
        print("discover() in C  called")

class D(A, B, C): # multiple inheritence
    def test(self):
        print("test() in D called")


d_obj = D()
d_obj.explore()
d_obj.search()
d_obj.discover()
d_obj.test()
```

```
explore in A  called
search in B  called
discover() in C  called
test() in D called
```

Timestamp 4.26

Multiple inheritance is a feature of some object-oriented computer programming languages in which an object or class can **inherit** characteristics and features from more than one parent object or parent class.

In the above example you can clearly see that class D is inheriting from base classes A,B,C.The order is important because While inheriting from another class, the interpreter needs a way to resolve the methods that are being called via an instance. Thus we need the method resolution order




```
# toy-example: Diamond inheritence a.k.a. Deadly dimaond

class A:
    def explore(self):
        print("explore in A  called")

class B(A):
    def explore(self):
        print("explore in B called")

class C(A):
    def explore(self):
        print("explore in C  called")

class D(B, C): # multiple inheritence
    pass;


d_obj = D()
d_obj.explore()
```

```
explore in B called
```

Timestamp 9.37
There is a problem called the deadly diamond problem with Multiple inheritance as shown above.

14.10 Polymorphism (with real world examples)

- Different forms
- Operator level Polymorphism: 2+3, "abc" + "def"

  The + operator is performing addition operation and also string concatenation
- Function level Polymorphism: len([1,2,3]), len ("abcdef"), len({1,2,3,4})

  The function len( ) accepts lists,strings,set but it returns the length.

```
print(len([1,2,3]));
print(len("abcdef"))
print(len({1,2,3,4}))
```

```
3
6
4
```

- Below is an example of Class level Polymorphism

Timestamp 2.34

```
#class level Polymorphism

class A:
    def p(self):
        return "function p in A"

class B:
    def p(self):
        return "function p in B"


a = A();
b = B();

for i in (a,b):
    print(i.p()) # the function that runs depends on the object type making this code much more elegant and crisp

print("#########################")

x=a;
print(x.p());

x=b;
print(x.p());
```

```
function p in A
function p in B
#########################
function p in A
function p in B
```

- Below is an example where polymorphism and inheritance both are implemented,Class Shape is inherited by Rectangle and Circle classes and class level polymorphism is used to iterate over objects belonging to different classes..

Timestamp 9.18

```python
# Polymorphism + Inheritence

# example seen earlier: [Source: https://overiq.com/python-101/inheritance-and-polymorphism-in-python/]
import math

class Shape:

    def __init__(self, color='black', filled=False):
        self.__color = color
        self.__filled = filled

    def get_color(self):
        return self.__color

    def set_color(self, color):
        self.__color = color

    def get_filled(self):
        return self.__filled

    def set_filled(self, filled):
        self.__filled = filled

```

```python
class Rectangle(Shape):

    def __init__(self, length, breadth):
        super().__init__()
        self.__length = length
        self.__breadth = breadth

    def get_length(self):
        return self.__length

    def set_length(self, length):
        self.__length = length

    def get_breadth(self):
        return self.__breadth

    def set_breadth(self, breadth):
        self.__breadth = breadth

    def get_area(self):
        return self.__length * self.__breadth

    def get_perimeter(self):
        return 2 * (self.__length + self.__breadth)

```

```
class Circle(Shape):
    def __init__(self, radius):
        super().__init__()
        self.__radius = radius

    def get_radius(self):
        return self.__radius

    def set_radius(self, radius):
        self.__radius = radius

    def get_area(self):
        return math.pi * self.__radius ** 2

    def get_perimeter(self):
        return 2 * math.pi * self.__radius
s = Shape();
r = Rectangle(10,20);
c = Circle(2);

for i in (s, r,c):
    print(i.get_color())

for i in (r,c):
    print(i.get_area())
```

```
black
black
black
200
12.566370614359172
```

```python
# Polymorphism + Inheritence [inbuilt-DS]

d = {'a':1, 'b':2}
l = [1,2,3,4]
s = {1,2,3,4}

for i in (d,l,s):
    print(i) # polymorphism + inheritence [__str__ from object]

```

```
{'a': 1, 'b': 2}
[1, 2, 3, 4]
{1, 2, 3, 4}
```

Timestamp 15.53

- We can observe the implementation of polymorphism and inheritance using inbuilt data structures as well.