OOP- II

- Multiple-inheritence, Polymorphism, iterator-objects
- Will answer a key questions from yesterday's live chat window.
- Will cover a few good ideas like dir() that others suggested in the chat window yesterday.

Pre-requisites: previous live-sessions in this series.

How is OOP typically used in an ML role:

- Using existing Classes.
- Reading documentation to understand how to use a fucntion/class/module.
- Fixing code bugs and understanding error messages.
- Extending existing classes to modify some fucntionality in an existing class
- Working with Software enigneers 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/ (https://learning.oreilly.com/library/view/head-first-design/0596007124/)

Multiple-inheritence

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

class DerivedClassName(Base1, Base2, Base3):
....
....

In [114]:

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

In [68]:

```
# 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
```

Polymorphism

- Different forms
- Operator level Polymorphism: 2+3, "abc" + "def"
- Function level Polymorphism: len([1,2,3]), len ("abcdef"), len({1,2,3,4})
- Class level Polymorphism

In [128]:

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

```
3
6
4
```

In [134]:

```
#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
```

In [138]:

```
# Polymorphism + Inheritence

# example seen earlier: [Source: https://overiq.com/python-101/inheritance-and-p
olymorphism-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


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
```

In [141]:

```
# 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}
```

Iterable objects

In [142]:

```
### Iteratable objects in Python
for i in [1,2,3,4]:
    print(i)
```

```
1
2
3
4
```

In [145]:

```
# How to make objects of a class iteratable?
# Source: https://docs.python.org/3/tutorial/classes.html#iterators

class Reverse:
    """Iterator for looping over a sequence backwards."""
    def __init__(self, data): # data can be list or tuple or string
        self.data = data
        self.index = len(data)

    def __iter__(self):
        return self

    def __next__(self):
        if self.index == 0:
            raise StopIteration
        self.index = self.index - 1
        return self.data[self.index]
```

In [146]:

```
r = Reverse([1,2,3,4]);
for i in r:
    print(i)
```

```
4
3
2
1
```

In [147]:

```
r = Reverse((1,2,3,4,5));
for i in r:
    print(i)
```

```
5
4
3
2
1
```

In [148]:

```
r = Reverse("abcdef");
for i in r:
    print(i)http://localhost:8888/notebooks/CodeWalkthroughSessions/LIVE_OOP_Bas
ics_II.ipynb#
```

```
f
e
d
c
b
a
```

In [149]:

```
r = Reverse({1,2,3,4,5});
for i in r:
    print(i)
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-149-28ae62154c2b> in <module>
      1 r = Reverse({1,2,3,4,5});
----> 2 for i in r:
      3     print(i)

<ipython-input-145-f14c16d18755> in __next__(self)
     15                 raise StopIteration
     16             self.index = self.index - 1
---> 17             return self.data[self.index]

TypeError: 'set' object is not subscriptable
```

Few questions from Yesterday's live session in the chat window

In [4]:

```
# Empty classes as a structure.

class E:
    pass

e1 = E();
e1.name="abc" # name and no attributes for e1
e1.no = 123


e2 = E(); # name and addr attrib for e2
e2.name = "xyz"
e2.addr = "abcdefghijklmnop"


print(e1.name, e1.no)
print(e2.name, e2.addr)
```

```
abc 123
xyz abcdefghijklmnop
```

In [9]:

```
# function within __init__
class A:
    def __init__(self):

        def function_within(x):
            return x+1

        i =10;
        print(function_within(i))


a = A();
```

```
11
```

In [11]:

```
# dir() is a powerful inbuilt function in Python3,
# which returns list of the attributes and methods of any object
# like classes , modules, strings, lists, dictionaries etc.

import math
print(dir(math)) # module
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__
spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh',
'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'e
xp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum',
'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isn
an', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'na
n', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'ta
n', 'tanh', 'tau', 'trunc']
```

In [14]:

```
class B:
    def f():
        print("Hi");

print(dir(B))
```

```
['__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__', '__weakr
ef__', 'f']
```

In [15]:

```
# use tab to autofill, differs from one IDE to another.
print(math.pi)
```

```
3.141592653589793
```

In [69]:

```
import sys
print(sys.version)
```

```
3.7.3 (default, Mar 27 2019, 09:23:15)
[Clang 10.0.1 (clang-1001.0.46.3)]
```

In [71]:

```
# private methods are just like private attributes that we saw earlier "__functi
on()"

class C:
    def __pr(self, i):
        return 2**i;

    def pu(self, x):
        return self.__pr(x)+1;

c = C();

print(c.pu(4))

print(c.__pr(4))
```

```
17

---------------------------------------------------------------------------
-------
AttributeError                            Traceback (most recent cal
l last)
<ipython-input-71-070461f56116> in <module>
     12 print(c.pu(4))
     13
---> 14 print(c.__pr(4))
     15

AttributeError: 'C' object has no attribute '__pr'
```

In [72]:

```
print(dir(C))
```

```
['_C__pr', '__class__', '__delattr__', '__dict__', '__dir__', '__doc
__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__
module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__re
pr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', 'pu']
```

In [73]:

```
print(c._C__pr(4))
```

```
16
```

In [76]:

```
# Inheritence of private members
# C++ has public, private, protected [accessible in all sub-classes but not outs
ide the class]

class C:
    def __pr(self, i): # private due to TWO underscores, implementation dependen
t, suggested
        return 2**i;

    def pu(self, x):
        return self.__pr(x)+1;

print(dir(C))
print("\n\n*********************************************************************
******\n\n")

class D(C):
    def f1(self, i):
        return self._C__pr(i) # accessing private member of base class

    def f2(self, i):
        return self.pu(i) # accessing private member of base class

d = D();
print(dir(D))

print(d.f2(4))
```

```
['_C__pr', '__class__', '__delattr__', '__dict__', '__dir__', '__doc
__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__
module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__re
pr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', 'pu']


*******************************************************************
*******


['_C__pr', '__class__', '__delattr__', '__dict__', '__dir__', '__doc
__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__
module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__re
pr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', 'f1', 'f2', 'pu']
17
```

In [77]:

```
print(d.f1(4))
```

```
16
```

In [58]:

```
# Protected in Python: ONE _
# Public: NO _
# Private: TWO _
# Convention and not a feature of the programming language. Implementation depen
dent
# Refer: https://docs.python.org/3/tutorial/classes.html#private-variables

class C:
    def _pr(self, i): # protected due to ONE underscore, implementation dependen
t, suggested
        return 2**i;

    def pu(self, x):
        return self._pr(x)+1;

print(dir(C))
print("\n\n*******************************************************************
******\n\n")

class D(C):
    def f1(self, i):
        return self._pr(i) # accessing private member of base class

    def f2(self, i):
        return self.pu(i) # accessing private member of base class

d = D();
print(dir(D))

print(d.f1(4)) # calling a protected member in base calss from derived class
```

```
['__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__', '__weakr
ef__', '_pr', 'pu']


*******************************************************************
*******


['__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__', '__weakr
ef__', '_pr', 'f1', 'f2', 'pu']
16
```

In [59]:

```
c = C();
print (c._pr(3))
```

```
8
```

In [42]:

```
# Constructor overloading:

class A:
    def __init__(self, i):
        self.var1 = i;
        self.var2 = 0;

    def __init__(self, i,j):
        self.var1 = i;
        self.var2 = j;

    def __str__(self):
        return "\ni="+str(self.var1)+"\t"+"j="+str(self.var2)

a1 = A(10);
print(a1)

# We can overload but can only use the most recent function definition
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-42-a4511922906c> in <module>
     13         return "\ni="+str(self.var1)+"\t"+"j="+str(self.var2
)
     14 
---> 15 a1 = A(10);
     16 print(a1)
     17 

TypeError: __init__() missing 1 required positional argument: 'j'
```

In [43]:

```
a1 = A(10,20);
print(a1)
```

```
i=10    j=20
```

In [61]:

```
# Can we call other functions inside __init__?

class A:
    def __init__(self, i,j):
        f(i,j)

    def f(self, i,j):
        self.var1 = i;
        self.var2 = j;

    def __str__(self):
        return "\ni="+str(self.var1)+"\t"+"j="+str(self.var2)

a1 = A(10,20);
print(a1)
```

```
---------------------------------------------------------------------
-------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-61-b8f3ad544ae8> in <module>
     12         return "\ni="+str(self.var1)+"\t"+"j="+str(self.var2
)
     13
---> 14 a1 = A(10,20);
     15 print(a1)

<ipython-input-61-b8f3ad544ae8> in __init__(self, i, j)
      3 class A:
      4     def __init__(self, i,j):
----> 5         f(i,j)
      6
      7     def f(self, i,j):

NameError: name 'f' is not defined
```

In [64]:

```
class A:

    def __init__(self, i,j):
        self.f(i,j)

    def f(self, i,j):
        self.var1 = i;
        self.var2 = j;

    def __str__(self):
        return "\ni="+str(self.var1)+"\t"+"j="+str(self.var2)

a1 = A(10,20);
print(a1)
```

```
i=10    j=20
```

- In later sections, we will use many major libraries where we will revisit OOP-concepts again and again
- We will extend some classes from major libraries (ML/DS/Plotting/Stats/DL) to suit our needs