

OOP

OOP in Python

Everything is an object.

- The int object 10 is instantiated.
- The variable i is referencing the object.

```
i = 10 # Python
```

`__init__` as a Constructor

- Python does not have a constructor; instead, it has `init(self)`.

```
class ...:
    def __init__(self):
        self.speed = 0
        self.altitude = 0
        self.rollAngle = 0
        self.pitchAngle = 0
        self.yawAngle = 0
```

- We can use parameters and default values.

```
# when no argument is given,  
# default values are assigned  
def __init__(self, speed = 0,  
              altitude = 0,  
              rollAngle = 0):  
    self.speed = speed  
    self.altitude = altitude  
    self.rollAngle = rollAngle
```

Instantiation

- No `new` needed; everything in Python is an object.
- Objects are created, and references point to them.

```
# Airplane.__init__(self) is invoked.  
a = Airplane()
```

Overriding Python class methods

- Python's print shows an object's reference by default.
- Override `__str__` to print a custom string instead.

```
a = Airplane(speed = 100)
# <__main__.Airplane object at 0x106f13220>
print(a)

class Airplane(object):
    ...
    def __str__(self):
        return f"Speed: {self.speed} ..."

b = Airplane(speed = 100)
print(a) # Speed: 100 ...
```

Overriding `__eq__`

- When we override the `__eq__()` method, we can change how the `==` operator works.

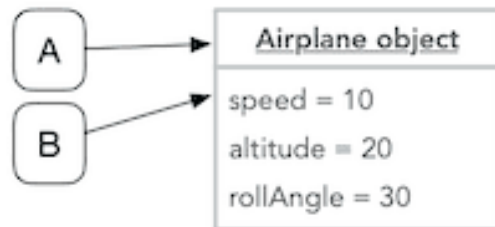
```
a = Airplane(speed = 100)
b = Airplane(speed = 100)
# False, as they are different object
print(a == b)
```



```
class Airplane(object):  
    ...  
    def __eq__(self, other):  
        return self.speed == other.speed  
  
a = Airplane(speed = 100)  
b = Airplane(speed = 100)  
  
# True because the speeds are the same  
print(a == b)
```

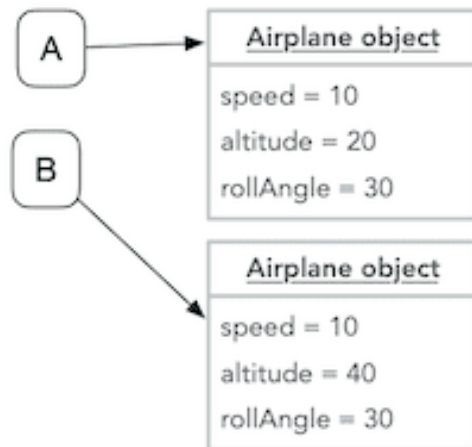
Shallow and Deep Copy

- In OOP, the = (assignment) operator copies the reference.



```
1 a = Airplane(speed = 100)
2 b = a
3 print(a == b) # True as same reference
4 b.speed = 500
5 print(a.speed) # 500
```

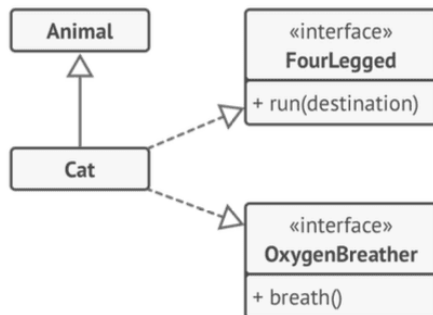
- To make a new object, we should use the `copy.deepcopy()` function.



```
1  from copy import deepcopy
2  ...
3  a = Airplane(speed = 100)
4  b = deepcopy(Airplane(speed = 100))
5  print(a == b) # False
6  b.speed = 500
7  print(a.speed) # 100
```

Python and Interface

- Python does not have an interface, but we can mimic an interface with an empty method.



```
1 class Animal(object): # Normal class
2     def makeSound(): print("")
3 class FourLegged(object): # Interface
4     def run(self, destination): pass
5 class OxygenBreather(object): # interface
6     def breathe(self): pass
7
8 # To python extends, and implements are the same
9 class Cat(Animal, FourLegged, OxygenBreather):
10     def run(self, destination):
11         print(f"I run to {destination}")
12     def breathe(self): print("I breathe")
13
14 c = Cat()
15 c.run("NKU")
16 c.breathe()
```

Python Duck Typing

- Duck typing means Python cares about an object's behavior, not its type.
- If an object has the required methods/attributes, it can be used regardless of its class.
- "If it walks like a duck and quacks like a duck, it's a duck."

- Both have the `__len__` method, so the `len` function invokes the method without knowing anything about the object.

```
# len is used for both data structures.  
a = []; len(a) # a.__len__()  
b = (); len(b) # b.__len__()
```

- We have the Duck and Tiger class that has the quack() method.
- The method is invoked in the same way.

```
class Duck(): def quack(self): print("Quack")
class Tiger(): def quack(self): print("Wow")
def quack_checker(q): q.quack()

quack_checker(Duck())
quack_checker(Tiger())
```

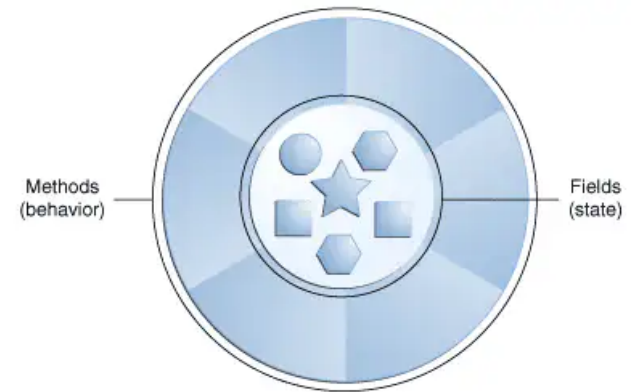
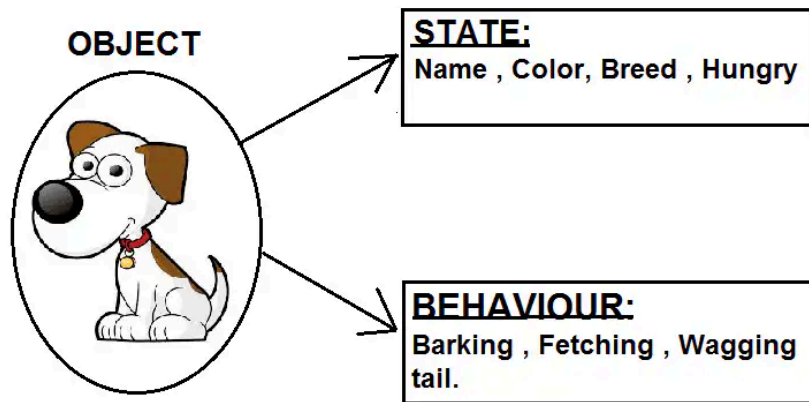
Python: Modeling with OOP

Python is an OOP language that can model the world effectively.

- Abstraction
- Polymorphism
- Interface
- Encapsulation

Abstraction

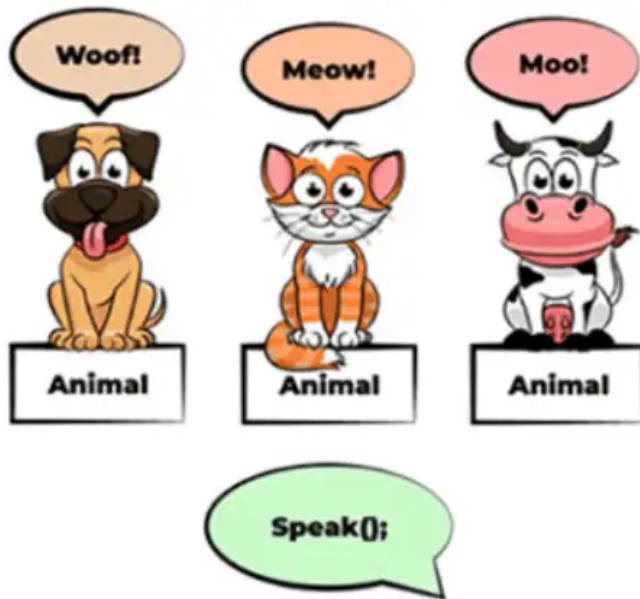
- Software modeling represents the real world.
- This is called abstraction.
- Abstraction captures an object's states (nouns) and behaviors (verbs).



- A Dog object has states and behaviors.
- The states/behaviors are called states/methods.

Polymorphism

- One message, different actions — this is polymorphism.



Inheritance

- We can model the real world with hierarchical general and specific concepts.
- This is called inheritance: superclasses are general, subclasses are specific, and creating one is called subclassing.

Super Class



Animals

Child Class



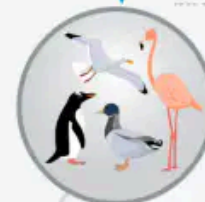
Amphibians



Reptiles



Mammals



Birds

Encapsulation

- In OOP, encapsulation means keeping fields (state) private and using public methods to access them.

