Object Oriented Design

CS 177 – Recitation 13

Table of Contents

OOP Terminology

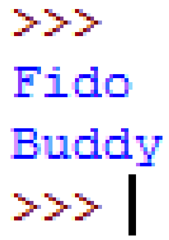
Class Definition

```
class Dog:

  kind = "canine"

  def __init__(self, name):
    self.name = name

  def printName(self):
    print(self.name)
```

Usage of Class

```
d = Dog("Fido")
e = Dog("Buddy")

d.printName()
e.printName()
```

Output



```
>>>
Fido
Buddy
>>>
```

OOP Terminology

Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

```
class Dog:
```

$\longrightarrow$ A class called Dog is defined

```
    kind = "canine"

    def __init__(self, name):
        self.name = name

    def printName(self):
        print(self.name)
```

OOP Terminology

Object : A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.

```
d = Dog("Fido")
e = Dog("Buddy")
```

Here both d and e are objects of class

OOP Terminology

Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods.

```
class Dog:

    kind = "canine"

    def __init__(self, name):
        self.name = name

    def printName(self):
        print(self.name)
```

kind is a class variable

How to use a class variable :

```
        print(Dog.kind)
        print(d.kind)
        print(e.kind)
```

OOP Terminology

Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

```
class Dog:

    kind = "canine"

    def __init__(self, name):
        self.name = name

    def printName(self):
        print(self.name)
```

self.name is an instance variable

How to use an instance variable :

```
        print(d.name)
        print(e.name)
```

Note: You will get an error if you use Dog.name

OOP Terminology

- **Data member:** A class variable or instance variable that holds data associated with a class and its objects.

OOP Terminology

Method: A function that is defined inside a class definition.

```
class Dog:

    kind = "canine"

    def __init__(self, name):
        self.name = name

    def printName(self):
        print(self.name)
```

printName is a method of the class

How to use a method :

```
d.printName()
e. printName()
```

Note: You will get an error if you use Dog.printname()

Global and Shared data

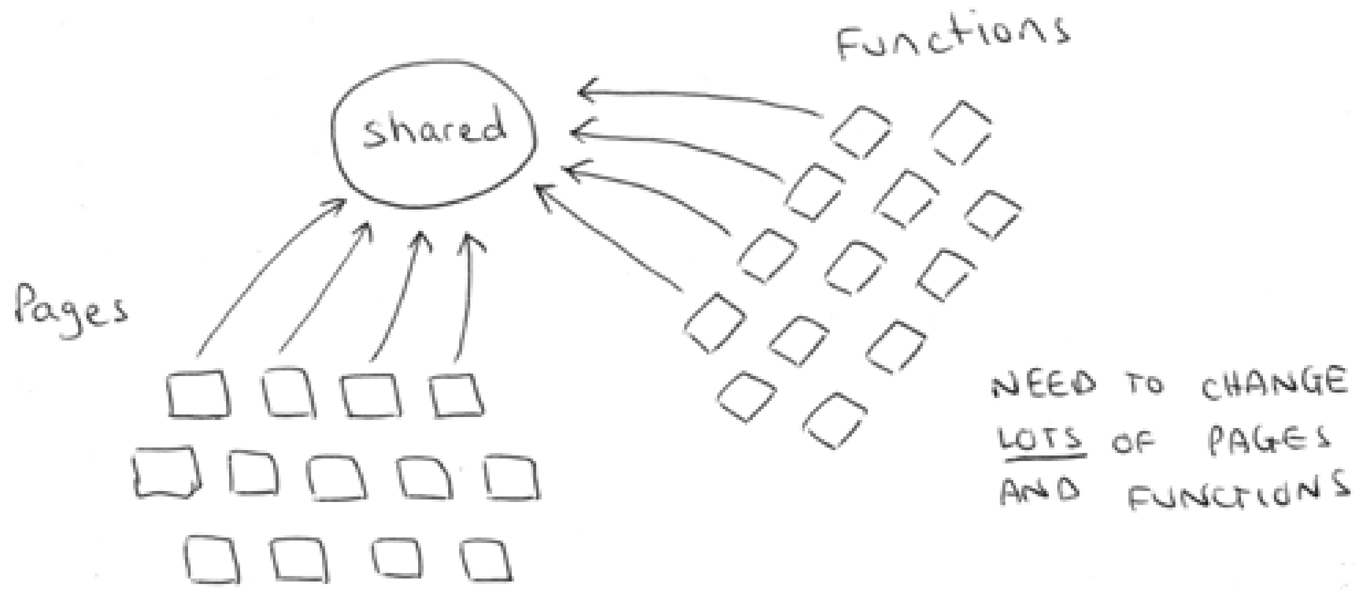
- We can see that one of the principle differences is that procedural systems make use of **shared** and **global data**, while object oriented systems lock their data privately away in objects.
- Let's consider a scenario where you need to change a shared variable in a procedural system. Perhaps you need to rename it, change it from a string to a numeric, change it from a struct to an array, or even remove it completely.

Global and Shared data

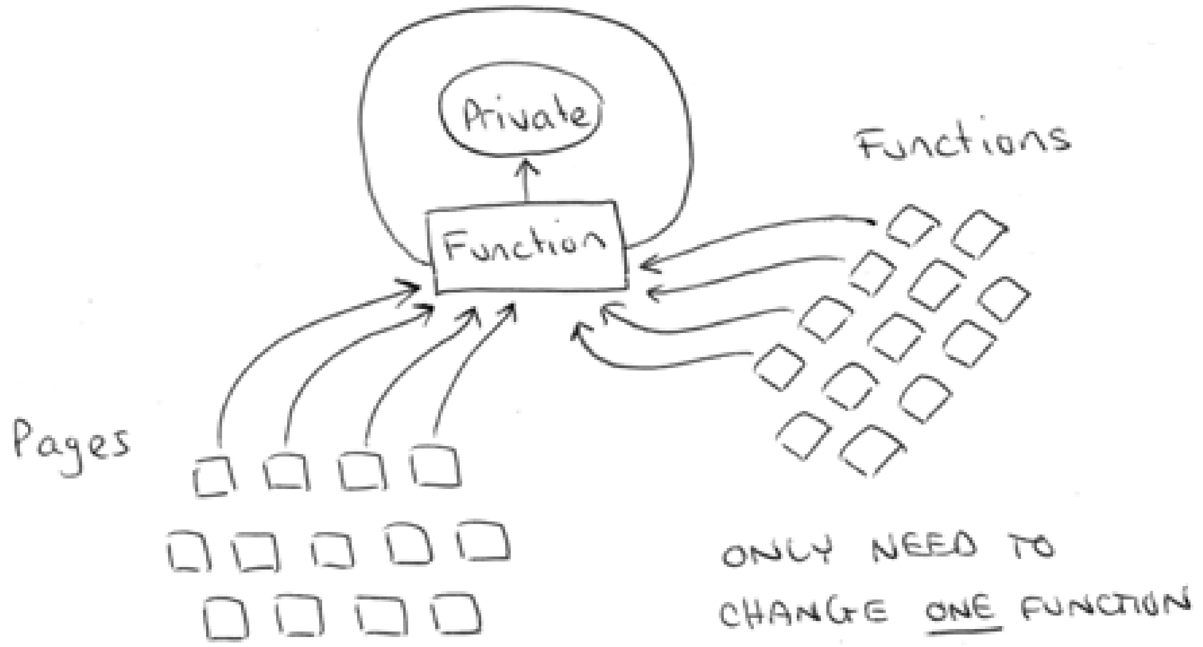
In a procedural application you would need to find and change each place in the code where that variable is referenced. In a large system this can be a widespread and difficult change to make.

In an object oriented system we know that all variables are inside objects and that only functions within those objects can access or change those variables. When a variable needs to be changed then we only need to change the functions that access those variables

Procedural approach



Object Oriented Design



Bank account System

- Suppose we want to model a bank account with support for deposit and withdraw operations.
- There are different approaches to do it.

Bank Account System (A naïve way)

- One way to do that is by using global variable
  - Notice that you can use a global variable in other functions by declaring it as **global** in each function that assigns to it

```
balance = 0

def deposit(amount):
    global balance
    balance += amount
    return balance

def withdraw(amount):
    global balance
    balance -= amount
    return balance
```

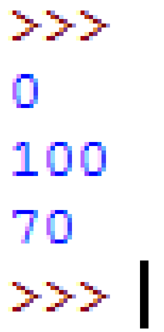
balance += amount is equivalent to:

balance = balance + amount

Bank Account System (A naïve way)

```
balance = 0

def deposit(amount):
    global balance
    balance += amount
    return balance

def withdraw(amount):
    global balance
    balance -= amount
    return balance

print(balance)

deposit(100)
print(balance)

withdraw(30)
print(balance)
```

Output



```
>>>
0
100
70
>>> |
```

Bank Account System (A naïve way)

- The previous example is good enough only if we want to have just a single account. Things start getting complicated if want to model multiple accounts.
- Let’s do it in another way …

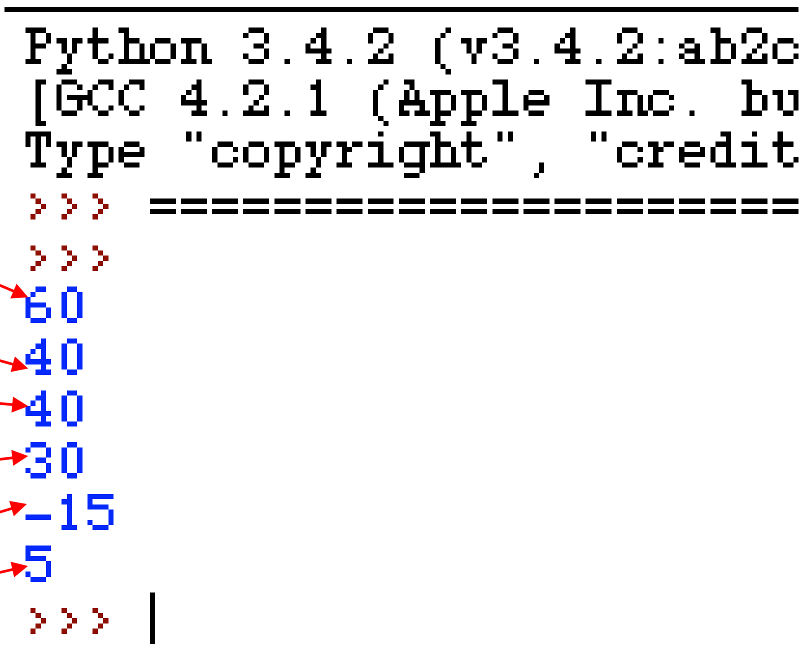
Bank Account System (OOP)

```
class BankAccount:
    def __init__(self, initial_balance):
        #Create an account with the given balance
        self.balance = initial_balance
        self.penalty = 0

    def deposit(self, amount):
        #deposit the amount into the account
        self.balance += amount
        return self.balance

    def withdraw(self, amount):
        #Withdraw the amount from the account. Each withdrawal resulting in a
        #negative balance also deducts a penalty fee of 5$ from the balance
        if self.balance - amount < 0:
            self.balance -= amount +5
            self.penalty += 5
        else:
            self.balance -= amount
        return self.balance

    def get_balance(self):
        #returns the current balance in the account
        return self.balance

    def get_fees(self):
        #returns the total fees ever dedcuted from the account
        return self.penalty
```

Bank Account System (OOP)

```python
def main():
    account1 = BankAccount(10)
    account2 = BankAccount(20)

    account2.deposit(40)
    print(account2.get_balance())

    account2.withdraw(20)
    print(account2.get_balance())

    account1.deposit(30)
    print(account1.get_balance())

    account1.withdraw(10)
    print(account1.get_balance())

    account1.withdraw(40)
    print(account1.get_balance())
    print(account1.get_fees())

main()
```

Output:



```
Python 3.4.2 (v3.4.2:ab2c
[GCC 4.2.1 (Apple Inc. bu
Type "copyright", "credit
>>> ======================
>>>
60
40
40
30
-15
5
>>> |
```

Encapsulation

- In OOP, the data and related functions are bundled together into an "object". Ideally, the data inside an object can only be manipulated by calling the object's functions.

Bank Account

deposit()

withdraw()

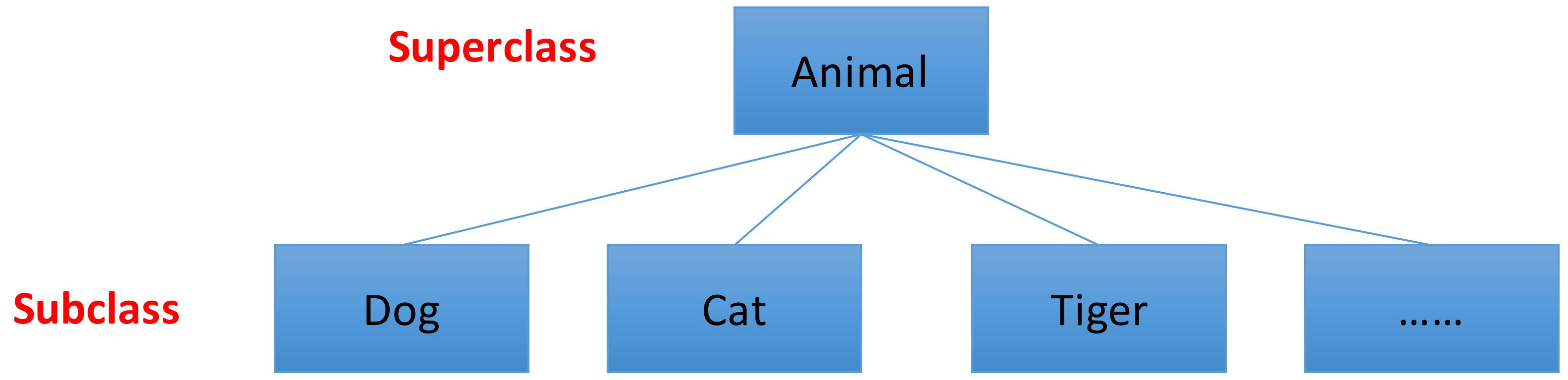
get_balance()

get_fees()

It is like a black box…….
We only need to worry about what objects can do (via the functions provided), and not about how they are implemented.

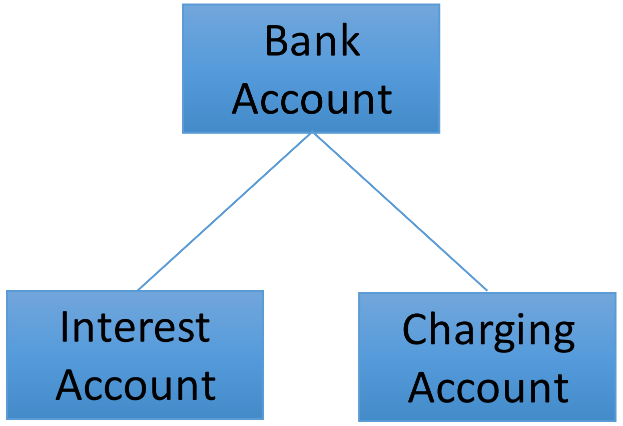
Inheritance

- Inheritance is used to indicate that one class will get most or all of its features from a parent class.
  - One example:



Inheritance

- Bank account system
  - **InterestAccount:** add interest (we'll assume 3%) on every deposit.
  - **ChargingAccount**: charge $3 for every withdrawal.



Inheritance

- Now we use inheritance to provide an **InterestAccount** class**.** It will be identical to the standard BankAccount class except for the **deposit** method. So we simply override that.
- For **ChargingAccount**, we can create a new class inheriting from BankAccount and modifying the **withdraw** method

Inheritance

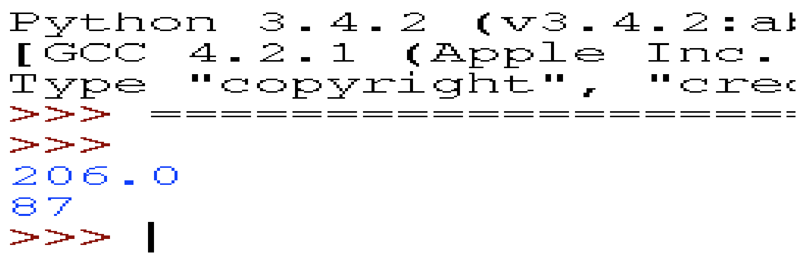
```
class InterestAccount(BankAccount):
    def deposit(self, amount):
        BankAccount.deposit(self, amount)
        self.balance = self.balance*1.03
        return self.balance

class ChargingAccount(BankAccount):
    def __init__(self, initial_balance):
        BankAccount.init(self, initial_balance)
        self.fee = 3

    def withdraw(self, amount):
        BankAccount.withdraw( self, amount + self.fee)
        return self.balance
```

Inheritance

```
class InterestAccount(BankAccount):
    def deposit(self, amount):
        BankAccount.deposit(self, amount)
        self.balance = self.balance*1.03
        return self.balance

class ChargingAccount(BankAccount):
    def __init__(self, initial_balance):
        BankAccount.init(self, initial_balance)
        self.fee = 3

    def withdraw(self, amount):
        BankAccount.withdraw( self, amount + self.fee)
        return self.balance
```

```
def main():
    Interest_AC = InterestAccount(100)
    Interest_AC.deposit(100)
    print(Interest_AC.getBalance())

    Charging_AC = ChargingAccount(100)
    Charging_AC.withdraw(10)
    print(Charging_AC.getBalance())

main()
```

Output



```
Python 3.4.2 (v3.4.2:ab
[GCC 4.2.1 (Apple Inc.
Type "copyright", "crea
>>> ====================
>>>
206.0
87
>>> 
```

Polymorphism

- In OOP, polymorphism is the provision of a single interface to entities of different types

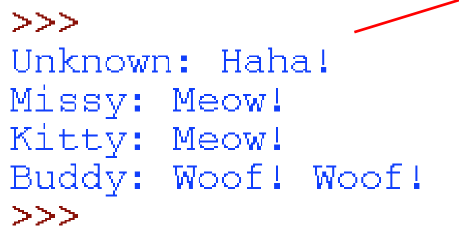
```
class Animal:
    def __init__(self, name):
        self.name = name

    def talk(self):
        return "Haha!"

class Cat(Animal):
    def talk(self):
        return "Meow!"

class Dog(Animal):
    def talk(self):
        return "Woof! Woof!"
```

Polymorphism

```
class Animal:
    def __init__(self, name):
        self.name = name

    def talk(self):
        return "Haha!"

class Cat(Animal):
    def talk(self):
        return "Meow!"

class Dog(Animal):
    def talk(self):
        return "Woof! Woof!"
```

```
animals = [Animal("Unknown"),
Cat("Missy"), Cat("Kitty"),
Dog("Buddy")]

for animal in animals:
    print(animal.name + ':' +
animal.talk())
```

Output



```
>>>
Unknown: Haha!
Missy: Meow!
Kitty: Meow!
Buddy: Woof! Woof!
>>>
```

Project-4 (RGB Model)

The RGB color model is an additive color model in which red, green and blue light are added together in various ways to reproduce a broad array of colors.

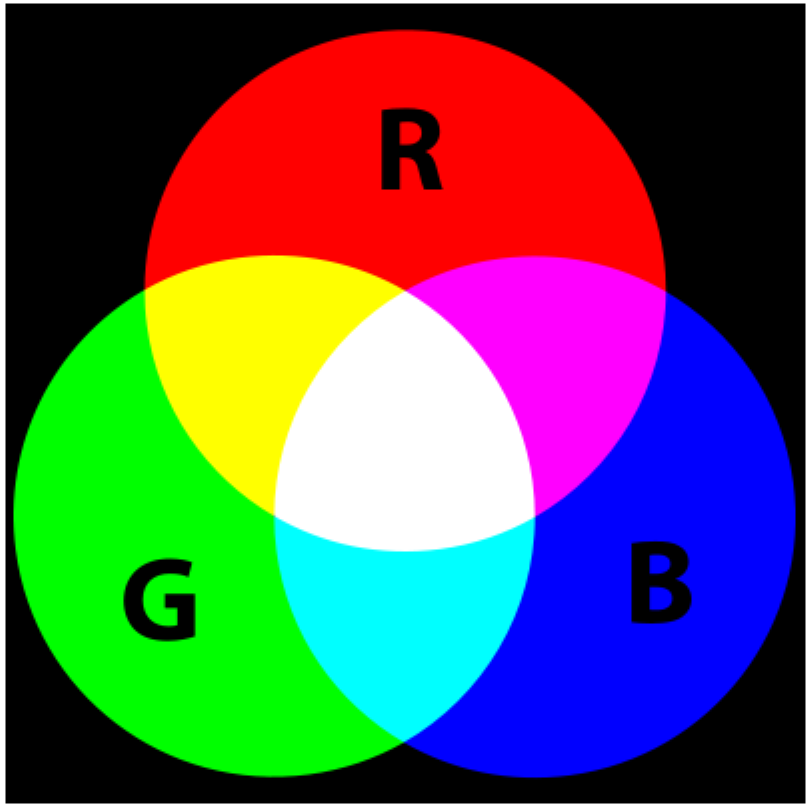
The name of the model comes from the initials of the three primary colors, red, green, and blue

To form a color with RGB, three light beams (one red, one green, and one blue) must be superimposed

Each of the three beams is called a *component* of that color, and each of them can have an arbitrary intensity, from fully off to fully on, in the mixture.

Zero intensity for each component gives the darkest color (no light, considered the *black*), and full intensity of each gives a white

Project-4 (RGB Model)



QUESTIONS?