OBJECT ORIENTED
PROGRAMMING
memegenerator.net

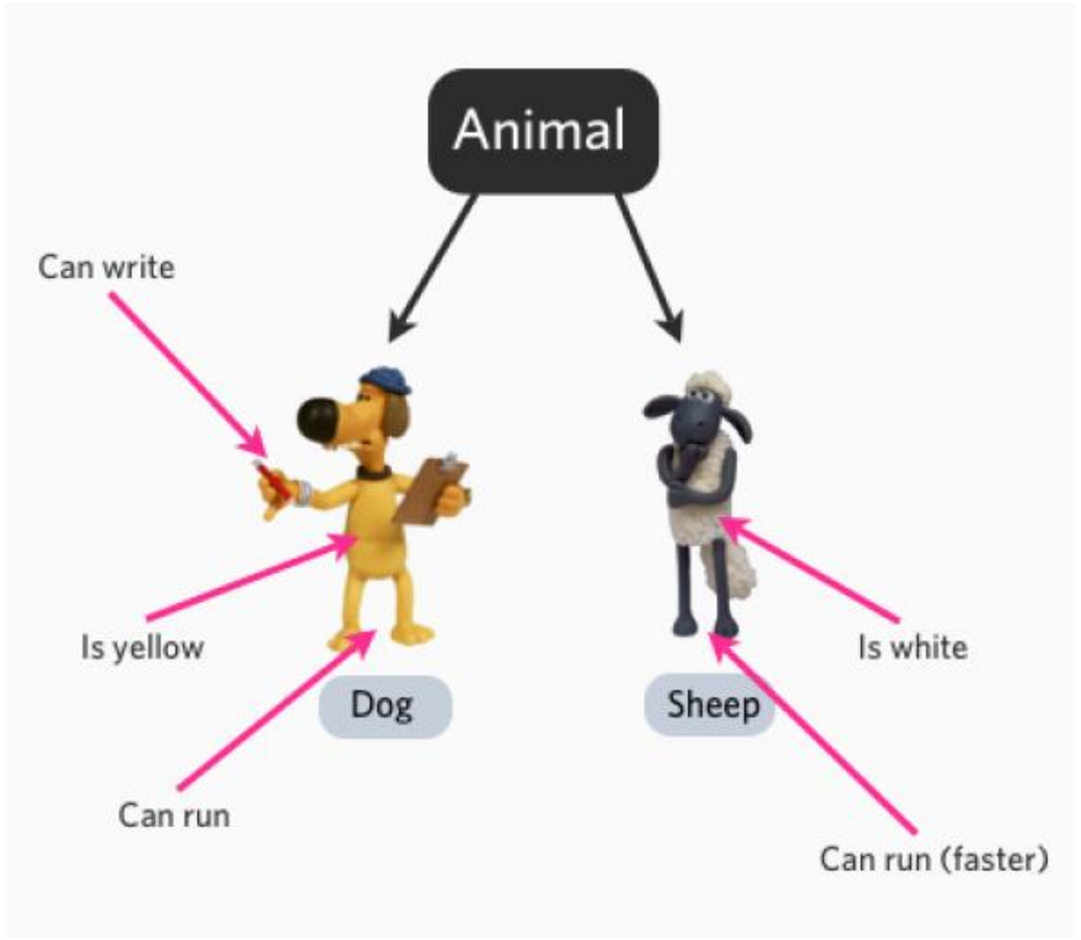
Effective Software Design

- Two simple general principles:
  - KIS (Keep It Simple)

    No Overengineering, no Spaghetti code.
  - DRY (Don’t Repeat Yourself)

    Code duplication equals bug reuse.
- Iterative Development: (Agile Development)
  - One cannot anticipate every detail of a complex problem.
  - Start simple (with something that works), then improve it.

Object Oriented Programming



Object Orientated Programming

- Objects

  Combine state (data) and behavior (algorithms).

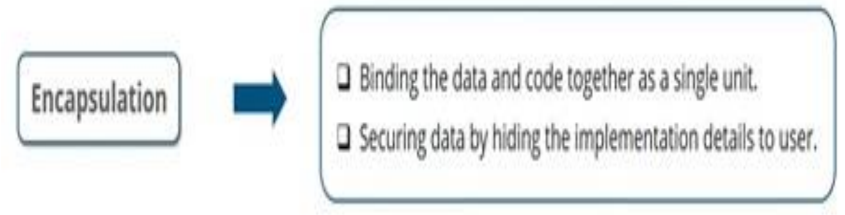
- Encapsulation

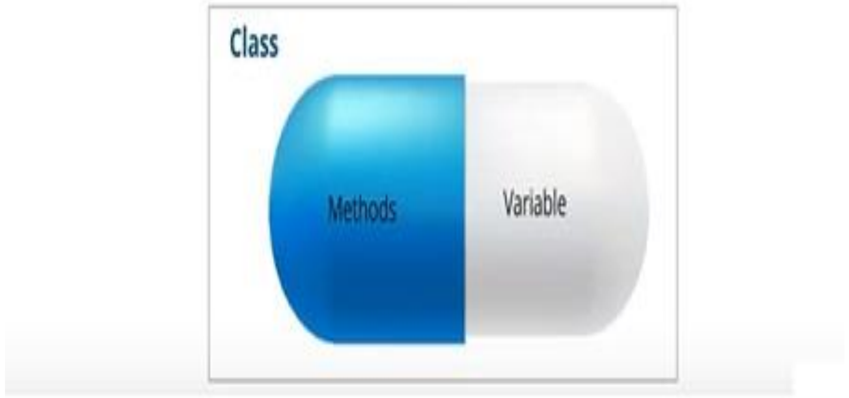
  Only what is necessary is exposed (public interface) to the outside. Implementation details are hidden to provide abstraction.

- Classes

  Define what is common for a whole class of objects.

  Define once how a car works and then reuse it for all cars.

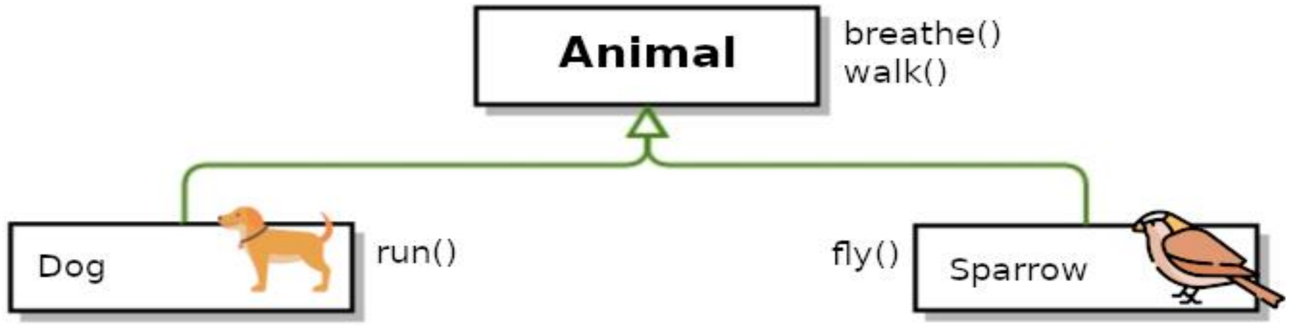

Encapsulation
❑ Binding the data and code together as a single unit.
❑ Securing data by hiding the implementation details to user.

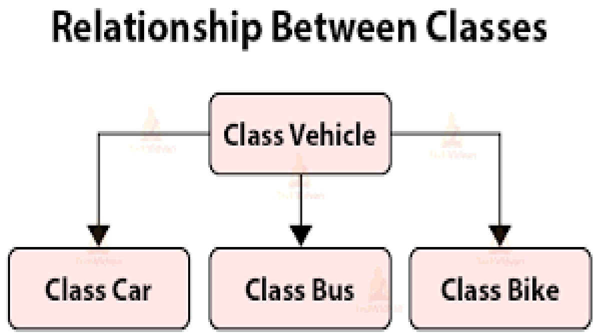


Object Orientated Programming

- Inheritance
  - Subclass is derived from / inherits / extends a parent class.

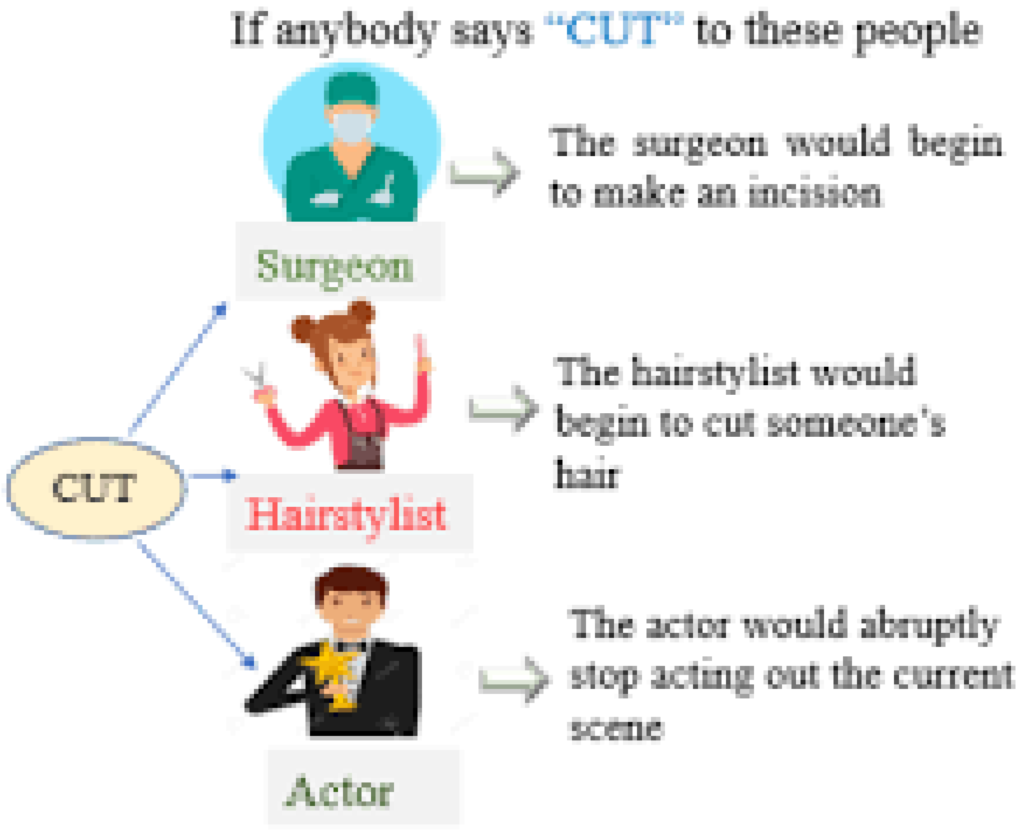
    "a dog (subclass) is a mammal (parent / superclass)"
  - Override parts with specialized behavior and extend it with additional functionality.
  - Liskov substitution principle: What works for the parent class should also work for any subclass.





Object Orientated Programming

- Polymorphism
  - Different subclasses can be treated like the parent class, but execute their specialized behavior.
  - Example:
    - 1 + 2 → 3,
    - "Hel" + "lo" → "Hello"



If anybody says "CUT" to these people
CUT
Surgeon
The surgeon would begin to make an incision
Hairstylist
The hairstylist would begin to cut someone's hair
Actor
The actor would abruptly stop acting out the current scene

A simple class : Account

```python
class Account:
    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder

    def deposit(self, amount):
        self.balance = self.balance + amount
        return self.balance

    def withdraw(self, amount):
        if amount > self.balance:
            return 'Insufficient funds'
        self.balance = self.balance - amount
        return self.balance
```

A simple class : Account

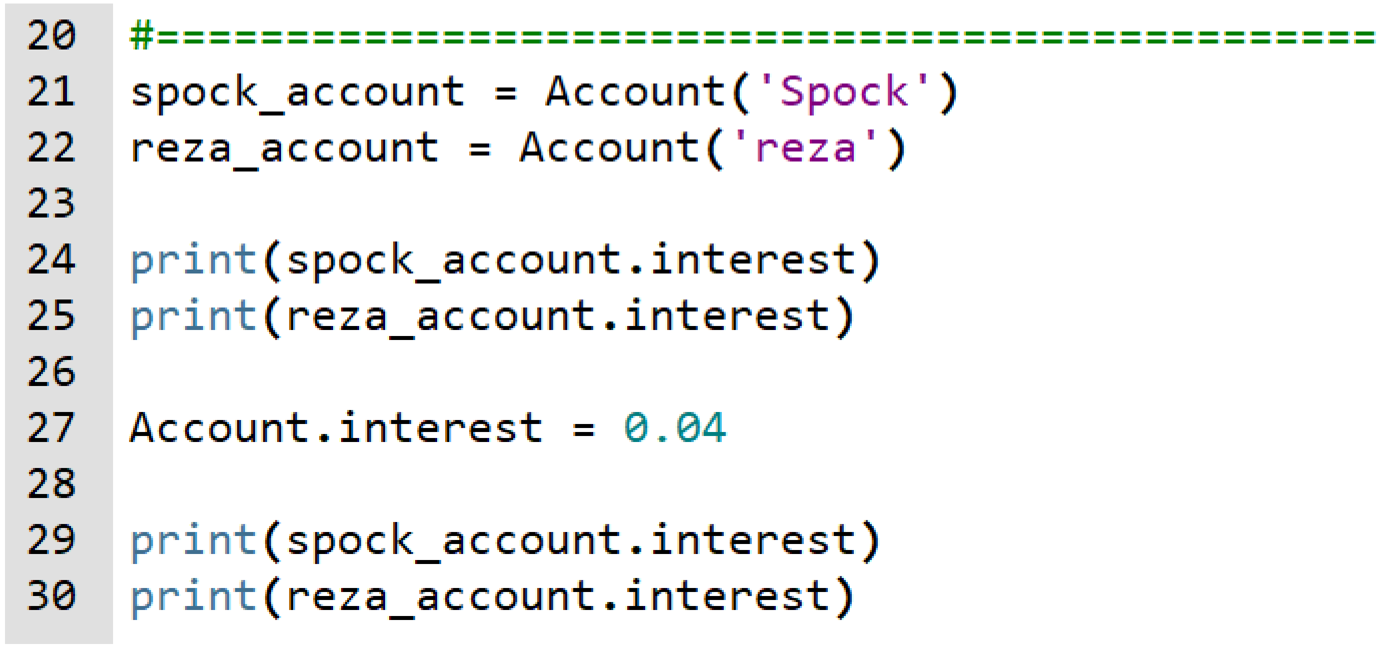
```python
class Account:
    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder

    def deposit(self, amount):
        self.balance = self.balance + amount

    def printBalance(self):
        print(self.balance)

    def withdraw(self, amount):
        if amount > self.balance:
            return 'Insufficient funds'
        self.balance = self.balance - amount
        return self.balance
#=========================================================
spock_account = Account('Spock')
spock_account.deposit(100)
spock_account.printBalance()
```

```
>>> [evaluate Account.py]
100
>>>
```

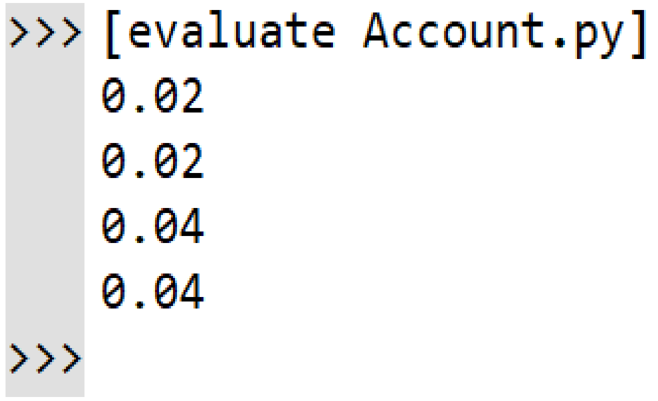
A simple class : Account

```python
class Account:

    interest = 0.02            # A class attribute

    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder

    def deposit(self, amount):
        self.balance = self.balance + amount

    def printBalance(self):
        print(self.balance)

    def withdraw(self, amount):
        if amount > self.balance:
            return 'Insufficient funds'
        self.balance = self.balance - amount
        return self.balance
```

A simple class : Account

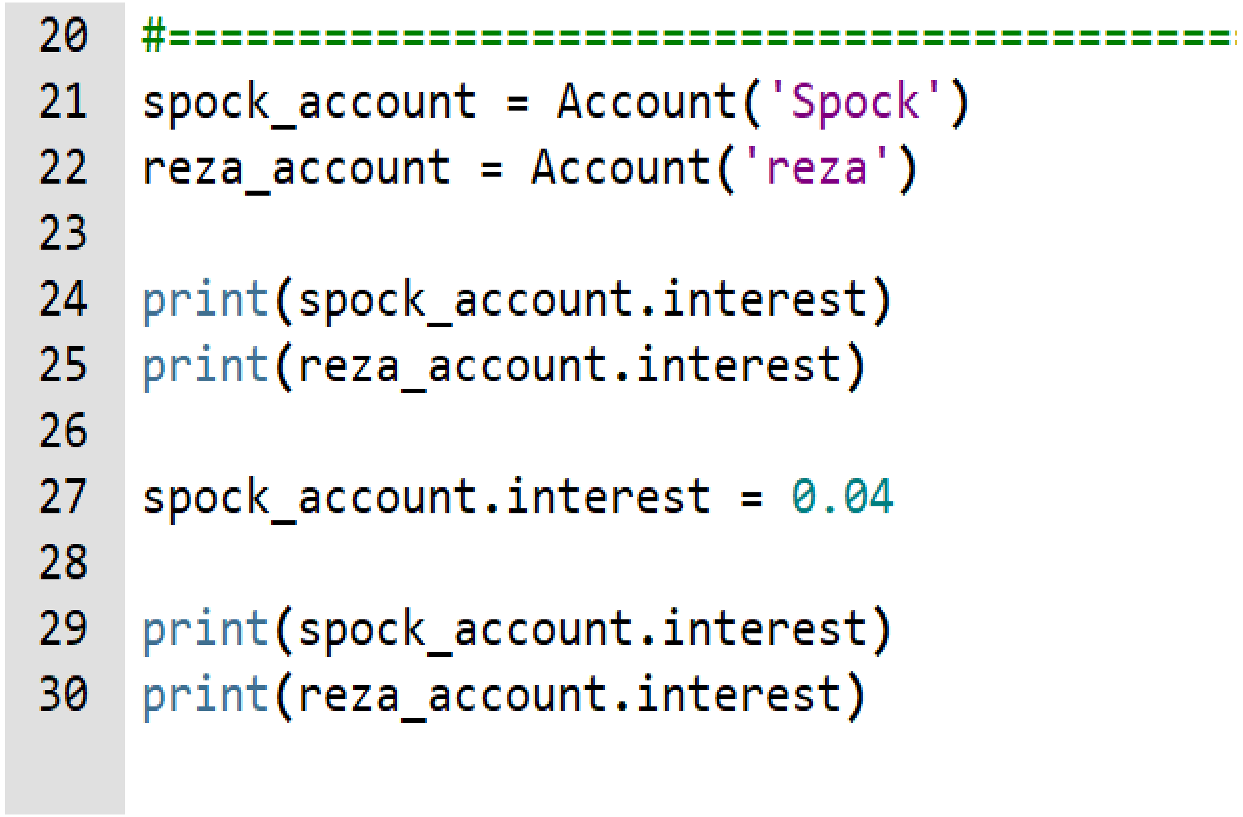


```python
#===============================================
spock_account = Account('Spock')
reza_account = Account('reza')

print(spock_account.interest)
print(reza_account.interest)

Account.interest = 0.04

print(spock_account.interest)
print(reza_account.interest)
```

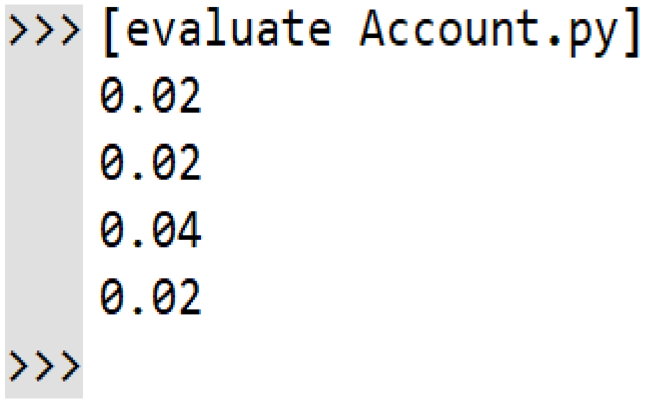


```
>>> [evaluate Account.py]
    0.02
    0.02
    0.04
    0.04
>>>
```

A simple class : Account



```python
#=========================================
spock_account = Account('Spock')
reza_account = Account('reza')

print(spock_account.interest)
print(reza_account.interest)

spock_account.interest = 0.04

print(spock_account.interest)
print(reza_account.interest)
```

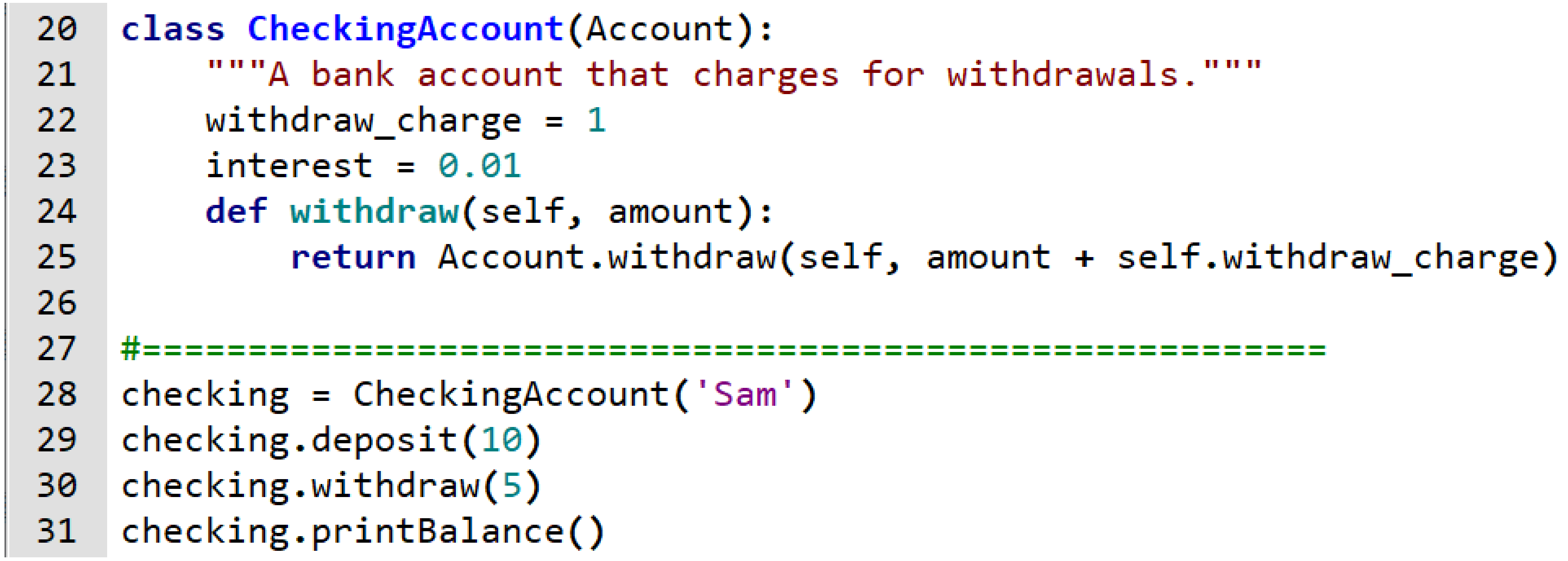


```
>>> [evaluate Account.py]
0.02
0.02
0.04
0.02
>>>
```

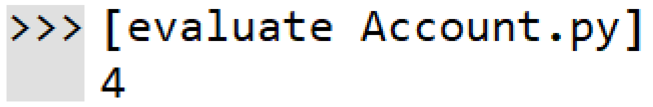
Inheritance : Account class

```python
class Account:
    interest = 0.02

    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder

    def deposit(self, amount):
        self.balance = self.balance + amount

    def printBalance(self):
        print(self.balance)

    def withdraw(self, amount):
        if amount > self.balance:
            return 'Insufficient funds'
        self.balance = self.balance - amount
        return self.balance
```

Inheritance : CheckingAccount class

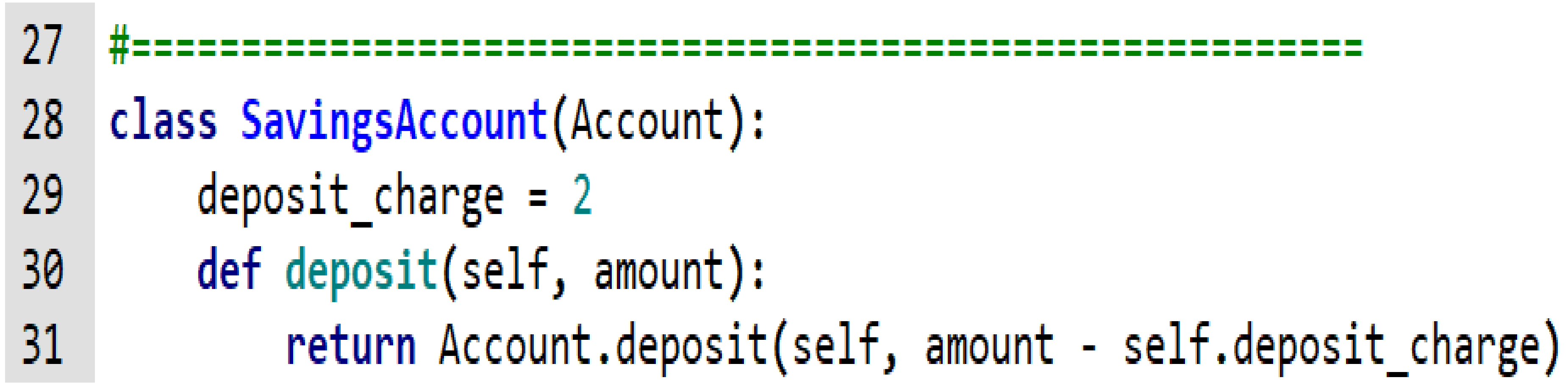


```python
class CheckingAccount(Account):
    """A bank account that charges for withdrawals."""
    withdraw_charge = 1
    interest = 0.01
    def withdraw(self, amount):
        return Account.withdraw(self, amount + self.withdraw_charge)

#==========================================================
checking = CheckingAccount('Sam')
checking.deposit(10)
checking.withdraw(5)
checking.printBalance()
```

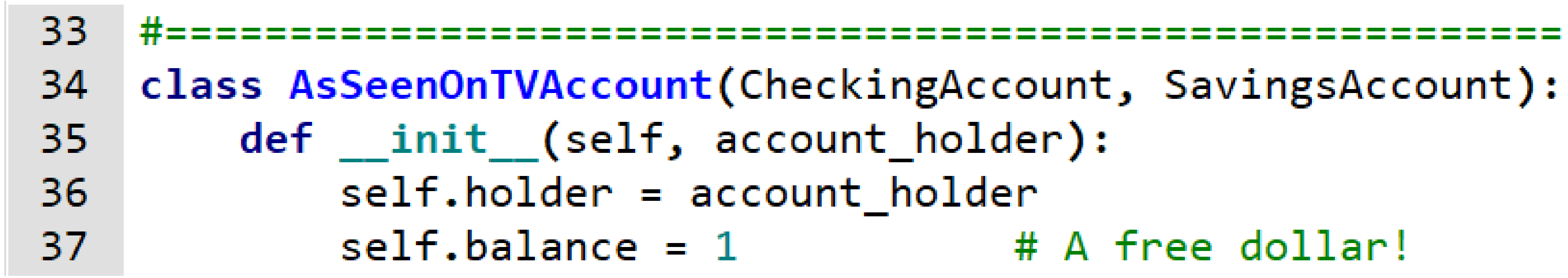


```
>>> [evaluate Account.py]
4
```

Multiple Inheritance : AsSeenOnTVAccount class

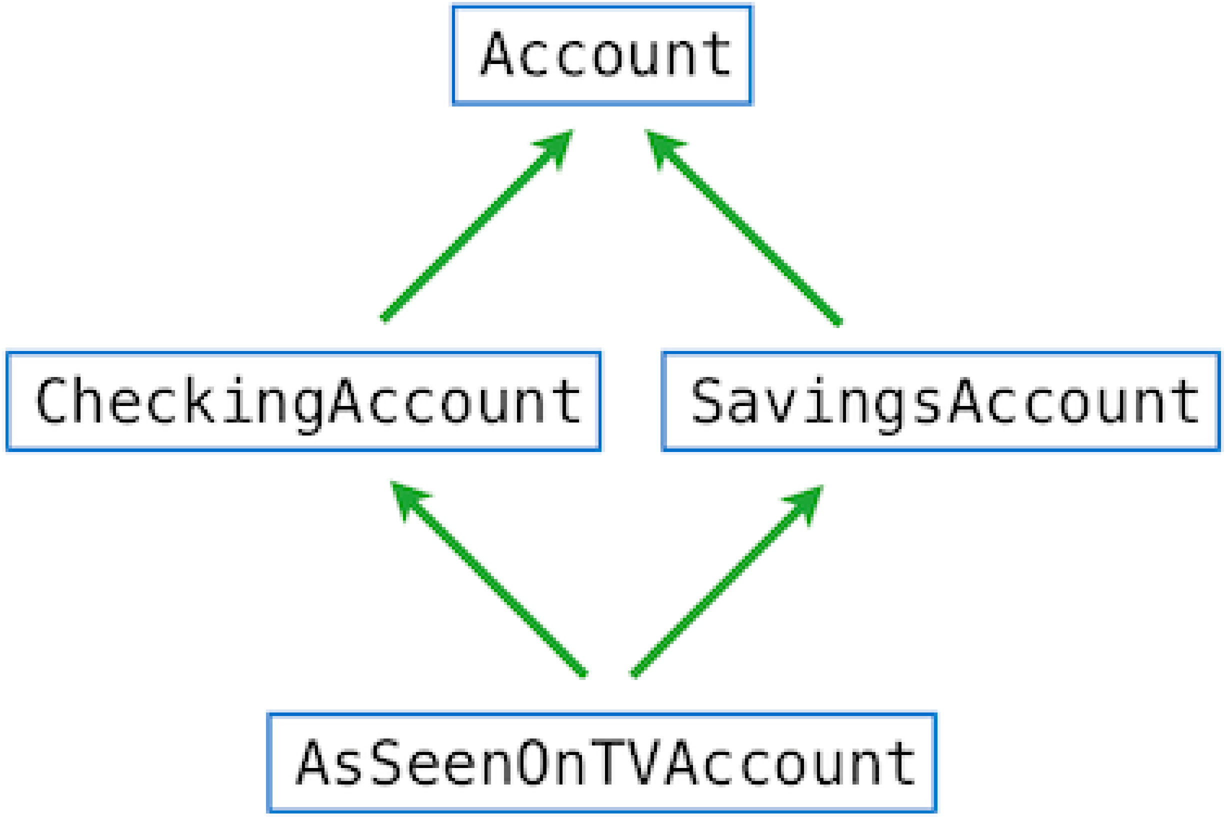


```python
#========================================================
class SavingsAccount(Account):
    deposit_charge = 2
    def deposit(self, amount):
        return Account.deposit(self, amount - self.deposit_charge)
```



```python
#========================================================
class AsSeenOnTVAccount(CheckingAccount, SavingsAccount):
    def __init__(self, account_holder):
        self.holder = account_holder
        self.balance = 1          # A free dollar!
```

Multiple Inheritance : AsSeenOnTVAccount class



Special Methods

```python
    def __str__(self):
        return self.holder + ' has ' + str(self.balance) + ' dollars'

    def __add__(self,other) :
        self.holder = self.holder + ' & ' + other.holder
        self.balance = self.balance + other.balance
```

```python
a = Account('Ali')
b = Account('Zahra')

print(a)

a.balance = 100
b.balance = 200
a+b
print(a.holder)
```

```
>>> [evaluate Account.py]
Ali has 0 dollars
Ali & Zahra
```

super() in Single Inheritance

```python
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * self.length + 2 * self.width

# ===============================================
class Square(Rectangle):
    def __init__(self, length):
        super().__init__(length, length)

#================================================
square = Square(4)
print(square.area())
```

```
>>> [evaluate rectangle.py]
16
```