Object Oriented Programming (OOP)

multi-paradigm programming language

- پایتون یک زبان برنامه نویسی multi-paradigm است و از الگوهای مختلف برنامه نویسی پشتیبانی می کند.
- یکی از مهمترین و محبوب ترین الگوها برای حل مسائل، **برنامه نویسی شی گرا** می باشد.
- در برنامه نویسی شی گرا، هر شی دو مشخصه اصلی دارد:
  - ویژگی (attribute)
  - رفتار (behavior) ← **method**
- تمرکز اصلی OOP در زبان پایتون بر ایجاد کد قابل استفاده مجدد (reusable) می باشد.
- بعنوان مثال: طوطی یک شی است:
  - نام، سن و رنگ (ویژگی)
  - آواز خواندن و رقصیدن (رفتار)

Python Data Types

- پایتون دارای انواع متنوعی از data type است. که بعضی ابتدایی (primitive) و بعضی هم پیچیده می باشند.
- در زبان پایتون ما می توانیم data type مخصوص خودمان را بسازیم!
- این یکی از اهداف مهم OOP می باشد.
- یک **شی (object)** می تواند به سادگی یک متغیر باشد.
- یک **کلاس (class)** می تواند به سادگی یک نوع داده ای برای یک شی باشد.
- برای مثال:
  - str یک نوع داده ای می باشد. (کلاس)
  - ما می توانیم اشیایی از نوع (کلاس) رشته بسازیم

```
Animal = “cow”
Fruit = “apple”
```

شی و کلاس

- **مقایسه شی و متغیر**
  - اشیا می توانند شامل متغیرهایی باشند که به آنها attribute گفته می شود.
  - همچنین اشیا می توانند شامل method باشند.
- **مقایسه کلاس و نوع**
  - برنامه نویسان می توانند کلاس های مربوط به خودشان را تعریف کنند.
  - Attributeها و methodها درون کلاس تعریف می شوند.

تعریف کلاس

- کلاس یک **طرح کلی** (blueprint) برای شی است.
- بعنوان مثال کلاس طوطی، یک طرح کلی از ویژگی ها و رفتارهای طوطی می باشد.
- در صورتیکه بخواهیم در مورد یک طوطی خاص صحبت کنیم می بایست ویژگی ها و رفتارهای آن طوطی را بر اساس طرح کلی (کلاس) بیان کنیم. در انجا طوطی یک شی می باشد.
- کلاس یک موجودیت **بالقوه** و شی یک موجودیت **بالفعل** می باشد.
- ساختار کلی تعریف کلاس

```
Class NewClass:
    <statement>
    <statement>
    . . .
```

تعریف کلاس - ادامه

- بعنوان مثال داریم:

```
Class Parrot:
    pass
```

- از کلمه کلیدی class برای تعریف یک کلاس خالی برای طوطی استفاده می کنیم.
- زمانیکه بخواهیم در مورد یک طوطی خاص صحبت کنیم، می بایست از کلاس طوطی یک نمونه (instance) بسازیم.

تعریف شی

- شی یک **نمونه** (**instance**) از یک **کلاس** است.
- در زمان تعریف کلاس هیچ حافظه ای برای آن در نظر گرفته نمی شود. ولی زمانیکه یک شی تعریف می کنیم برای آن حافظه مناسبش در نظر گرفته می شود.
- در زیر می توانید تعریف یک شی از کلاس طوطی را ملاحظه نمایید.

```
obj = Parrot()
```

پیاده سازی کلاس

- ساختار تعریف کلاس در پایتون

```
Class NewClass:
    <statement>
    <statement>
    . . .
```

- **متد initializer** یا **سازنده (constructor)**، متدی است که در زمان ایجاد شی از روی کلاس یکبار اجرا می شود. (بسیار کاربردی برای مقداردهی اولیه صفات کلاس)
- عنوان سازنده در زبان پایتون **__inti__** می باشد.
- همه متدها در کلاس می بایست حداقل یک پارامتر داشته باشند. این پارامتر به شی کلاس اشاره می کند و با **self** نشان داده می شود.

تعریف کلاس و شی - مثال

```
class Parrot:
    # class attribute
    species = "bird"

    # instance attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age

# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))
 # access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

Output:

Blu is a bird
Woo is also a bird
Blu is 10 years old
Woo is 15 years old

متد (method)

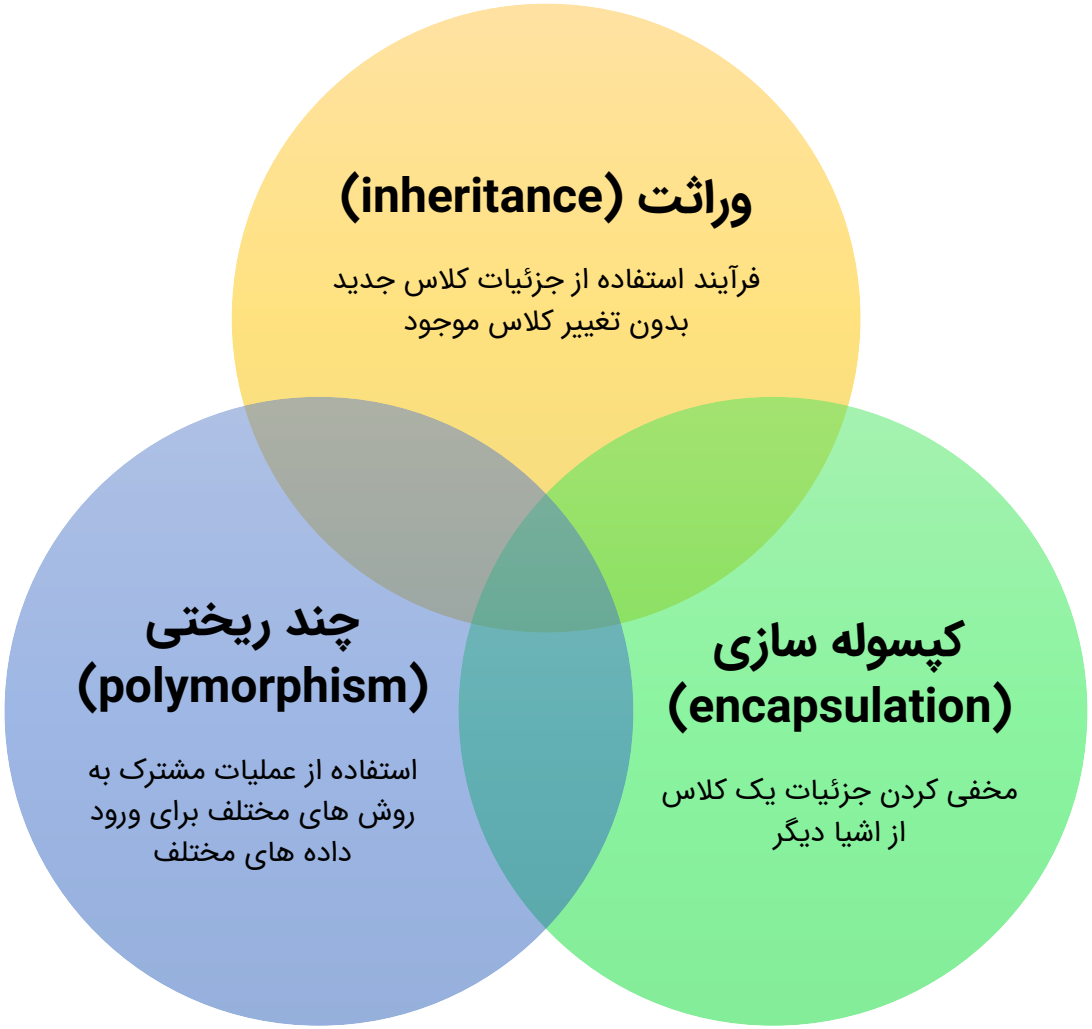
- متدها توابعی هستند که درون کلاس تعریف شده و برای تعریف رفتار یک شی مورد استفاده قرار می گیرند.

```
class Parrot:
    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def sing(self, song):
        return "{} sings {}".format(self.name, song)

    def dance(self):
        return "{} is now dancing".format(self.name)

# instantiate the object
blu = Parrot("Blu", 10)
# call our instance methods
print(blu.sing("'Happy'"))
print(blu.dance())
```

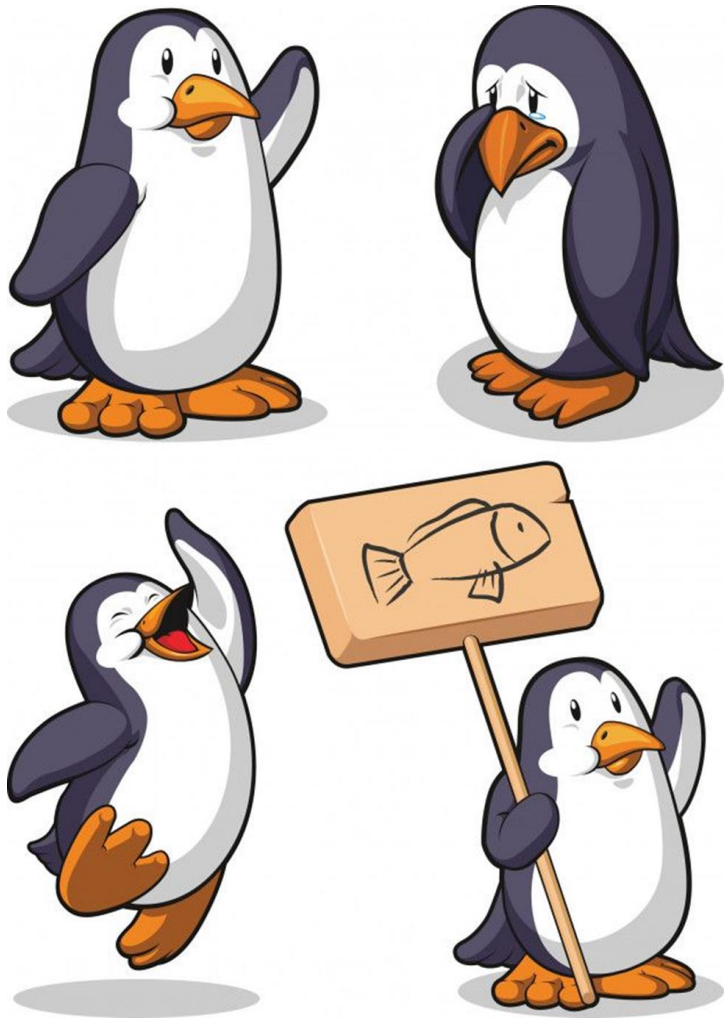
Output:

Blu sings ‘Happy’
Blu is now dancing

چند مفهوم اساسی در OOP



وراثت (inheritance)



- وراثت این امکان را به ما می دهد تا کلاسی ایجاد کنیم تا از همه و یا قسمتی از جزئیات کلاس دیگر استفاده نماید.
- به کلاس تازه شکل گرفته کلاس **مشتق شده** یا **child** و به کلاس موجود، **کلاس پایه** یا **parent** گفته می شود
- وراثت یک ویژگی مهم در زبان های برنامه نویسی شی گرا می باشد.

ساختار وراثت در پایتون

- تعریف دو کلاس child و parent بصورت زیر انجام می شود.

```
Class BaseClass:
    <statement>
    . . .

Class DerivedClass(BaseClass):
    <statement>
    . . .
```

- کلاس مشتق شده ویژگی هایی را از کلاس پایه به ارث می برد و همچنین می تواند ویژگی های جدیدی را هم برای خود داشته باشد.
- این ویژگی (وراثت) منجر به قابلیت **استفاده مجدد** از کد می شود.

ساختار وراثت - مثال

- می خواهیم یک کلاس چند ضلعی تعریف کنیم به نام Polygon

```
class Polygon:
    def __init__(self, no_of_sides):
        self.n = no_of_sides
        self.sides = [0 for i in range(no_of_sides)]


    def inputSides(self):
        self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i in range(self.n)]


    def dispSides(self):
        for i in range(self.n):
            print("Side",i+1,"is",self.sides[i])
```

ساختار وراثت - مثال

- می دانیم مثلث یک چندضلعی ست با ۳ ضلع. پس برای تعریف کلاس مثلث کافیست که مشخص کنیم کلاس مثلث از polygon ارث می برد. با این کار تمام ویژگی های موجود در کلاس چندضلعی در مثلث نیز قابل دسترس می شوند و دیگر نیازی به تعریف مجدد آنها وجود ندارد.

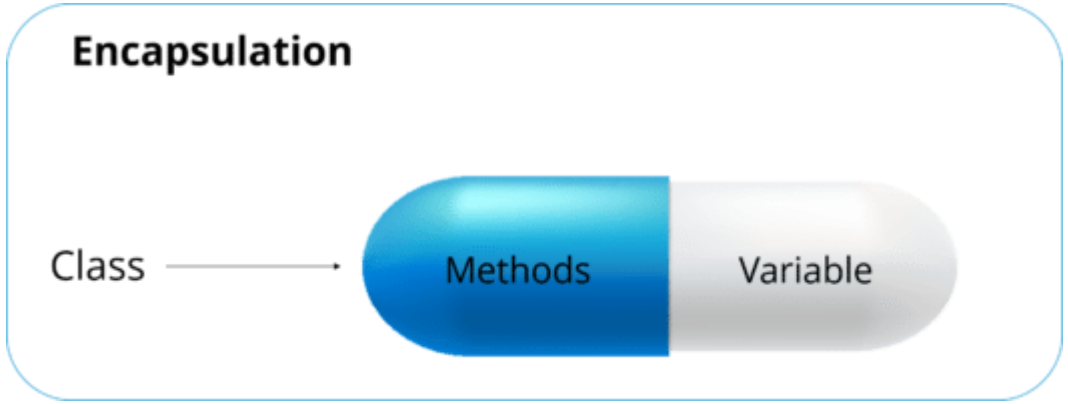
```
Class Triangle(Polygon):
    def __init__(self):
        Polygon.__init__(self,3)


    def findArea(self):
        a, b, c = self.sides
        # calculate the semi-perimeter
        s = (a + b + c) / 2
        area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
         print('The area of the triangle is %0.2f' %area
```

ساختار وراثت - مثال

```
>>> t = Triangle()

>>> t.inputSides()
Enter side 1 : 3
Enter side 2 : 5
Enter side 3 : 4

>>> t.dispSides()
Side 1 is 3.0
Side 2 is 5.0
Side 3 is 4.0

>>> t.findArea()
The area of the triangle is 6.00
```

کپسوله سازی (encapsulation)

- با استفاده از OOP، می توانیم دسترسی به متغیرها و متدهای کلاس را محدود کنیم. به عمل محدود کردن دسترسی **کپسوله سازی** گفته می شود.
- در زبان پایتون صفات خصوصی با _ و یا __ مشخص می شوند.



کپسوله سازی (encapsulation)

- مثال

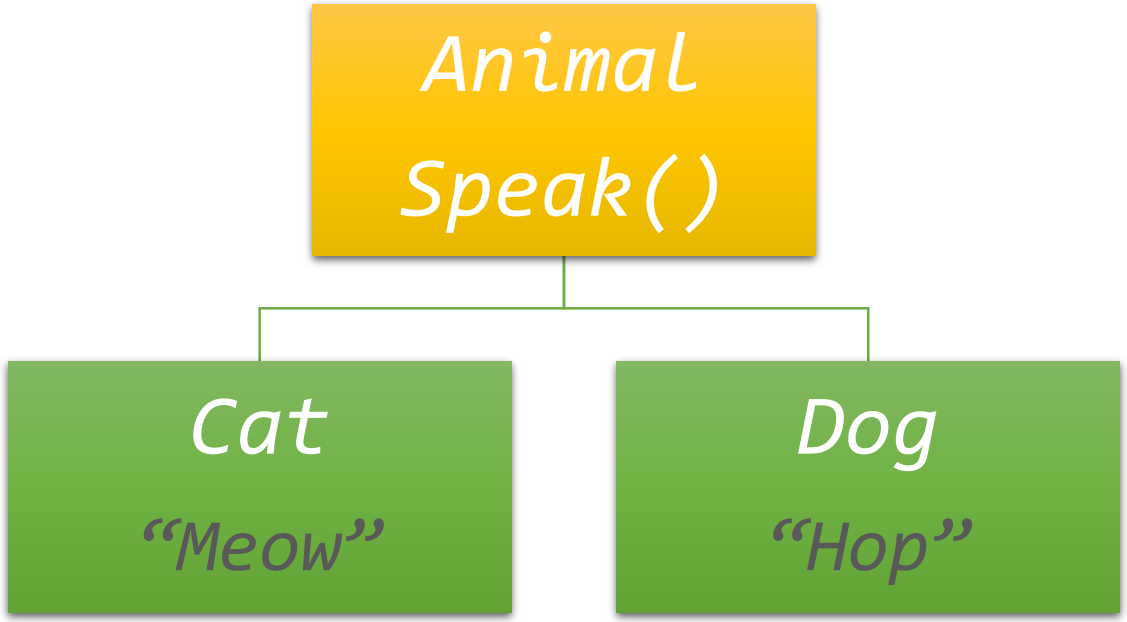
```
class Computer:

    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

Output:

```
Selling Price: 900
Selling Price: 900
Selling Price: 1000
```

چندریختی (polymorphism)

- پلی مورفیسم یک ویژگی مهم در OOP بوده که توانایی استفاده از یک رابط مشترک برای فرم های مختلف را فراهم می سازد.
- فرض کنید که می خواهیم یک شکل را رنگ آمیزی کنیم. این شکل می تواند مستطیل، مربع و دایره باشد. با این وجود می توانیم از همان روش برای رنگ آمیزی هر شکلی استفاده کنیم. به این مفهوم **پلی مورفیسم** یا **چندریختی** گفته می شود.



چندریختی (polymorphism)

- مثال

```
class Parrot:

    def fly(self):
        print("Parrot can fly")

    def swim(self):
        print("Parrot can't swim")

class Penguin:

    def fly(self):
        print("Penguin can't fly")

    def swim(self):
        print("Penguin can swim")

# common interface
def flying_test(bird):
    bird.fly()

#instantiate objects
blu = Parrot()
peggy = Penguin()

# passing the object
flying_test(blu)
flying_test(peggy)
)
```

Output:

Parrot can fly

Penguin can’t fly

حذف صفات و اشیا

- با استفاده از دستور del می توانیم ویژگی های یک شی و یا خود شی را حذف کنیم.
- مثال: فرض کنید کلاسی به شکل زیر تعریف شود

```
class ComplexNumber:
    def __init__(self,r = 0,i = 0):
        self.real = r
        self.imag = i

    def getData(self):
        print("{0}+{1}j".format(self.real,self.imag))
```

حذف صفات و اشیا - ادامه

```
>>> c1 = ComplexNumber(2,3)
>>> del c1.imag
>>> c1.getData()
Traceback (most recent call last):
...
AttributeError: 'ComplexNumber' object has no attribute 'imag'

>>> del ComplexNumber.getData
>>> c1.getData()
Traceback (most recent call last):
...
AttributeError: 'ComplexNumber' object has no attribute
'getData'.format(self.real,self.imag))

>>> c1 = ComplexNumber(1,3)
>>> del c1
>>> c1
Traceback (most recent call last):
...
NameError: name 'c1' is not defined
```

حذف صفات و اشیا - ادامه