



# Programação Orientada a Objetos (POO)

- Utiliza o conceito de classes para encapsular dados e funções.

```
class Ponto(object):  
  
    def __init__(self, x,y):  
        self.x=x  
        self.y=y  
  
    def distancia(self,outro):  
        deltaX=(self.x-outro.x)  
        deltaY=(self.y-outro.y)  
        return (deltaX**2+deltaY**2)**0.5
```



# Programação Orientada a Objetos (POO)

- Objetos são instâncias de uma classe.

`p=Ponto(3,4)` ⇒ criou-se o objeto `p` como uma instância da classe `Ponto`.

`print(p.x,p.y)` ⇒ imprime 3 4, valores dos atributos `x` e `y` do objeto `p`.

`q=Ponto(4,8)` ⇒ outro objeto da classe `Ponto`.

`d= p.distancia(q)` ⇒ calcula distância entre `p` e `q`



# Programação Orientada a Objetos (POO)

- Para acessar diretamente um dado de um objeto:

`<objeto>.<atributo>`  $\Rightarrow$  `p.x` ou `p.y`

- Para acessar um método de um objeto:

`<objeto>.<método>`  $\Rightarrow$  `p.distancia(q)`



# Programação Orientada a Objetos (POO)

- Nós já estávamos manipulando objetos em Python:
  - `L=[10,20,30]` é um objeto que instancia um tipo lista.
  - `L.append(40)` ocorre onde `append()` é um método de lista.
  - `'João'` é uma instância do tipo string.
  - `2.5` é uma instância do tipo float.



# Programação Orientada a Objetos (POO)

- Utilizando o paradigma da POO podemos
  - Criar objetos de um novo tipo definindo novas classes.
  - Manipular tais objetos
  - Descartar tais objetos.



# Programação Orientada a Objetos (POO)

- Objetos podem ser vistos como abstrações para
  - atributos de dados
  - interface de interação através de métodos.
- As classes facilitam a reutilização de código.



# Programação Orientada a Objetos (POO)

- Criando uma classe
  - `class <nome da classe>(<classe pai>): class Ponto(object):`
  - Ponto herda todos os atributos da classe `object` definida em Python.
  - Ponto é uma subclasse de `object`
  - `object` é uma superclasse da classe Ponto.



# Programação Orientada a Objetos (POO)

- Os atributos de uma classe são representados pelos:
  - dados
  - Métodos 

```
def distancia(self,outro):  
    deltaX=(self.x-outro.x)  
    deltaY=(self.x-outro.y)  
    return (deltaX**2+deltaY**2)**0.5
```





# Programação Orientada a Objetos (POO)

- Um objeto é instanciado através de uma chamada ao método `__init__`.
- O método `__init__` inicia alguns dados do objeto.

```
def __init__(self, a,b):  
    self.a=a  
    self.b=b
```



# Programação Orientada a Objetos (POO)

```
def __init__(self, a,b):  
    self.a=a  
    self.b=b
```

- **self** se refere ao próprio objeto instanciado.
- `p=Ponto(3,4)`
  - `Ponto(3,4)` chama `__init__(self,3,4)`
  - **self** não precisa entrar como argumento em `Ponto(3,4)`.



# Programação Orientada a Objetos (POO)

```
def distancia(self,outro):  
    deltaX=(self.x-outro.x)  
    deltaY=(self.y-outro.y)  
    return (deltaX**2+deltaY**2)**0.5
```

- **self** se refere ao próprio objeto instanciado.
- outro é o argumento relativo ao outro objeto instanciado
- `d=p.distancia(q)`
  - **self** se refere ao objeto daquele método, ou seja, p.



# Programação Orientada a Objetos (POO)

- `_init_()` é um método reservado de Python.
- Um método reservado é disponibilizado pela linguagem e tem uma finalidade específica.
- Há outros métodos reservados:

```
__str__()  
__add__() +  
__sub__() -  
__eq__() ==  
__lt__() <
```

# Programação Orientada a Objetos (POO)

```
class Ponto(object):  
    def __init__(self, a,b):  
        self.a=a  
        self.b=b  
  
    def distancia(self,outro):  
        deltaX=(self.x-outro.x)  
        deltaY=(self.y-outro.y)  
        return (deltaX**2+deltaY**2)**0.5
```

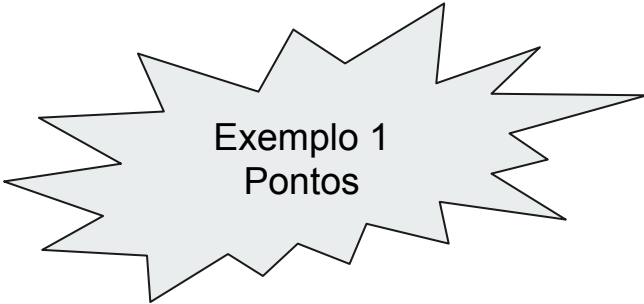
```
def __str__(self):  
    return '('+str(self.x)+','+str(self.y)+')
```



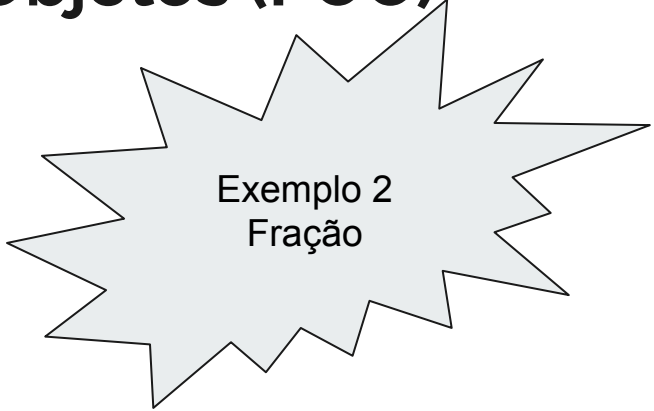
```
p=Ponto(3,4)  
print(p) ⇒ (3,4)
```



# Programação Orientada a Objetos (POO)



Exemplo 1  
Pontos



Exemplo 2  
Fração