



# Noções de Orientação a Objetos

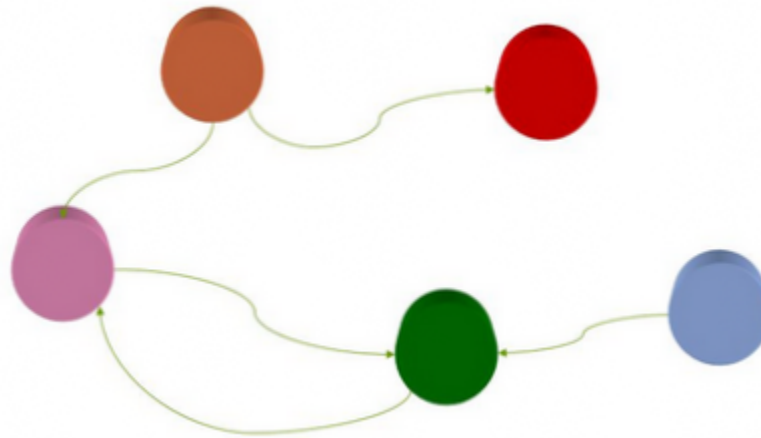
Prof.:

Edemberg Rocha/Thiago Moura

# Orientação a Objetos

---

- ▶ Paradigma que introduz uma abordagem na qual o programador visualiza seu programa em execução como uma coleção de objetos cooperantes que se comunicam.



- ▶ Os objetos conhecem muito bem a si mesmos e respondem as mensagens de acordo com suas características (atributos) e com seus próprios métodos.

# Orientação a Objetos

---

## ▶ Vantagens

- ▶ Facilidade de manutenção;
- ▶ Reusabilidade;
- ▶ Simplicidade;
- ▶ Agilidade.



# Conceito de Orientação a Objetos

---

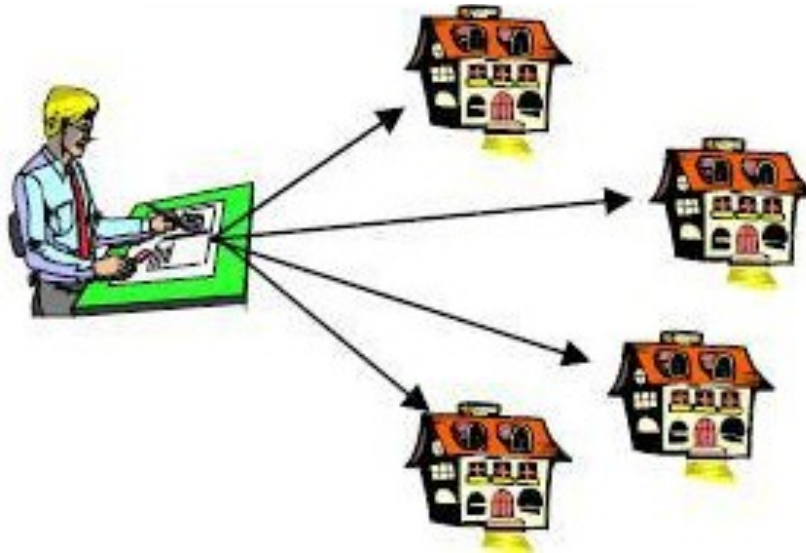
- ▶ Exemplos de objetos do mundo bancário:
  - ▶ `conta numero=1, agencia=1234; conta numero=2, agencia=1234; conta numero=3, agencia=1234`
  - ▶ `Cliente Edemberg, CPF=12345, Cliente Thiago, CPF=67890`
- ▶ Objetos podem ser agrupados em classes: `Conta Corrente`, `Cliente` etc.
- ▶ Observe que existem várias contas correntes de uma mesma classe “`Conta Corrente`”.



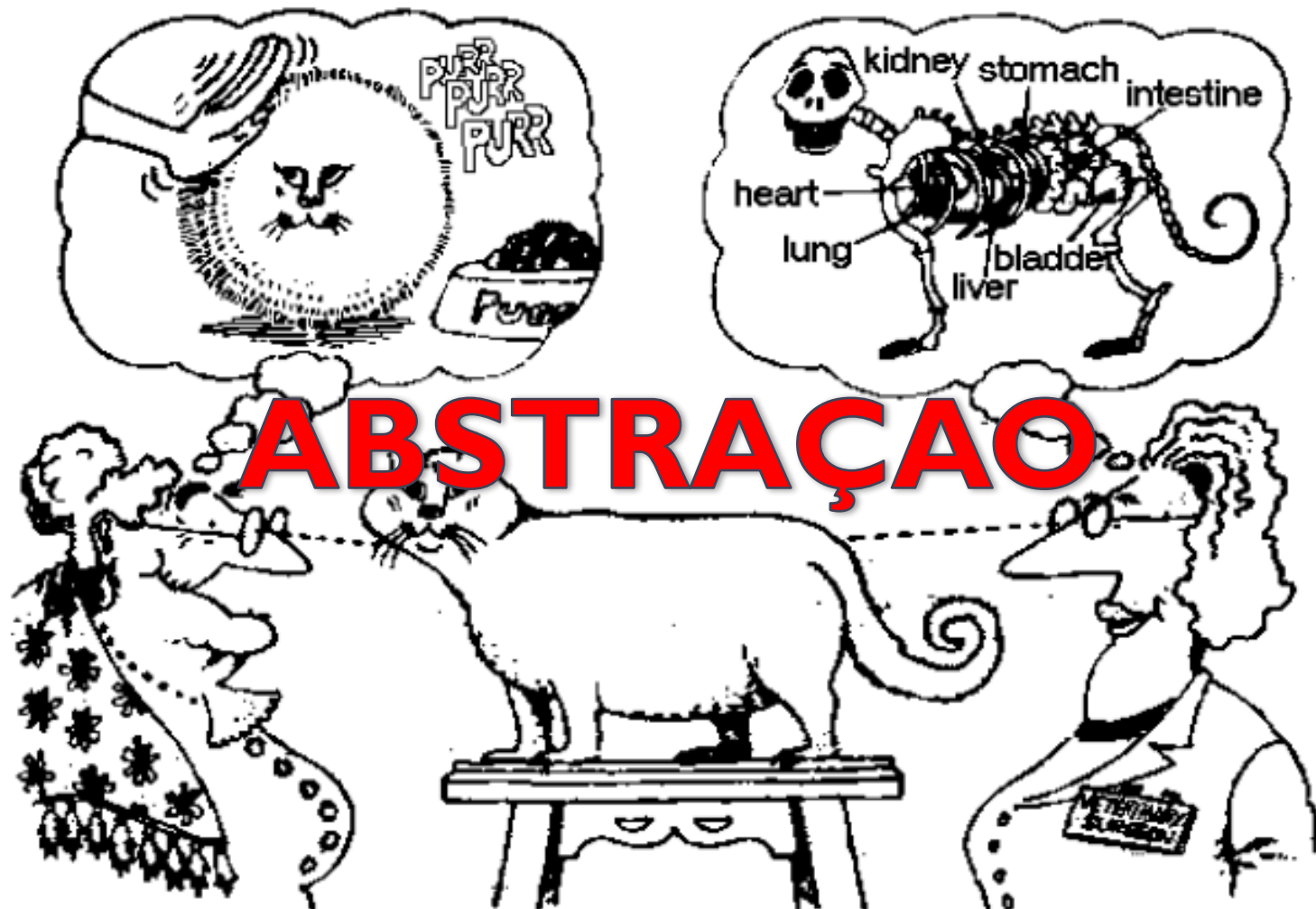
# Orientação a Objetos

---

- ▶ A diferença entre classe e objeto
  - ▶ Classe é um gabarito (como uma planta de uma casa).
  - ▶ Objeto é a concretização do gabarito (casas feitas a partir da mesma planta), i.e., uma representação de algo que seja real ou abstrato que contenha traços bem definidos.



# Como classificar objetos???



# Abstração

---

- ▶ “Processo mental que consiste em escolher ou isolar um aspecto determinado de um estado de coisas relativamente complexo, a fim de simplificar a sua avaliação, classificação ou para permitir a comunicação do mesmo” - Dicionário Houaiss.



# Abstração

---

- ▶ Na programação...
  - ▶ Abstrair um conceito é limitar-se a **representar** este conceito em um linguagem de programação **apenas** em seus **detalhes necessários** à **aplicação** em curso.
  - ▶ Por exemplo, um sistema bancário abstrai a entidade real cliente em apenas alguns dados do cliente que são importantes (nome, cpf, endereço, salário, etc).





# Abstração

---

- ▶ **Objetos** com os mesmo traços recebem a mesma **classificação**.
- ▶ Os objetos podem ser concretos ou abstratos:
  - ▶ Concretos: pessoa, carro, casa etc;
  - ▶ Abstratos: conta, música, disciplina etc.



# Classes e Objetos em Python

Noções de Orientação a Objetos

# Classes e Objetos em Python

---

- ▶ A sintaxe básica para definição de uma classe:

Palavra reservada



**class** Nome\_Da\_Classe :

Exemplo a seguir....



# Classes e Objetos em Python

---

- ▶ Exemplo: definir uma classe que representa pontos cartesianos, com coordenadas  $x$  e  $y$ .

```
1  class Ponto:
2      |      pass
3
4  ponto1 = Ponto()
5  ponto2 = Ponto()
```

Na linha 1, a palavra reservada **class** indica a criação da classe chamada **Ponto**.

Nas linhas 4 e 5 são criados (instanciados) objetos da classe. Logo, **ponto1** e **ponto2** são dois objetos.

E as coordenadas  $X$  e  $Y$  dos objetos?



# Classes e Objetos

---

- ▶ **Objetos** de uma certa classe **têm atributos**
  - ▶ Cada casa tem seu número, sua cor, sua localização;
  - ▶ Cada conta tem um número, saldo, histórico;
  - ▶ Cada pessoa tem nome, endereço;
  - ▶ Cada cheque tem uma numeração e valor;
- ▶ No caso, cada ponto cartesiano tem coordenada X e Y, logo nossa classe...



# Objetos e Atributos em Python

---

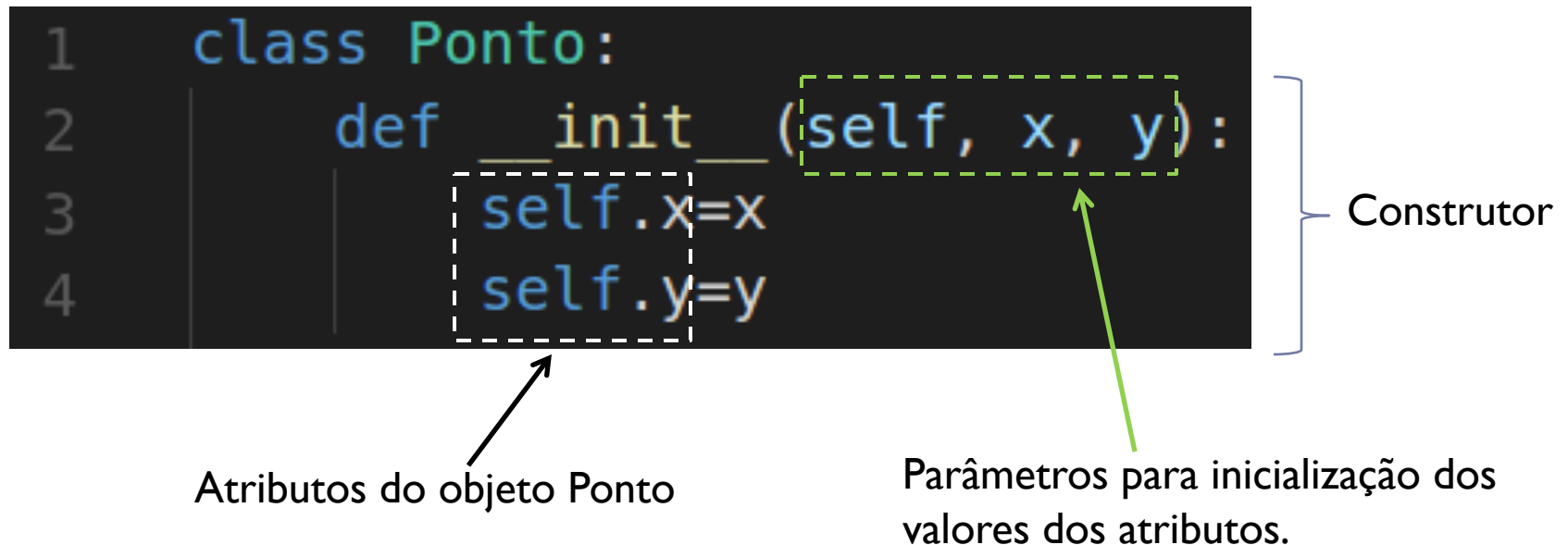
## ► Exemplo:

```
1 class Ponto:
2     def __init__(self, x, y):
3         self.x=x
4         self.y=y
```

Construtor

Atributos do objeto Ponto

Parâmetros para inicialização dos valores dos atributos.

The diagram shows a Python class definition for 'Ponto'. The class has a constructor method named '\_\_init\_\_'. The parameters 'self', 'x', and 'y' are enclosed in a green dashed box, with a green arrow pointing to it from the text 'Parâmetros para inicialização dos valores dos atributos.'. The body of the method contains two lines of code: 'self.x=x' and 'self.y=y', which are enclosed in a white dashed box. A black arrow points from the text 'Atributos do objeto Ponto' to this white dashed box. A blue bracket on the right side of the code block groups the entire method definition under the label 'Construtor'.

Mais à frente falaremos do self!!!

# Objetos e Atributos em Python

---

## ► Exemplo:

```
1 class Ponto:
2     def __init__(self, x, y):
3         self.x=x
4         self.y=y
5
6 ponto1 = Ponto(1,2)
7 print ('Coordenadas do Ponto 1 (%d,%d)'%(ponto1.x,ponto1.y))
8 ponto2 = Ponto(5,5)
9 print ('Coordenadas do Ponto 2 (%d,%d)'%(ponto2.x,ponto2.y))
```

Saída:

Coordenadas do Ponto 1 (1,2)

Coordenadas do Ponto 2 (5,5)

Acesso aos atributos dos  
objetos através do operador “.”

# Exercício 1

---

a) Defina classes que representem:

- ▶ Retângulo
- ▶ Aluno
- ▶ Conta Corrente

b) Crie objetos das classes do item (a), imprimindo o valor de seus atributos.





# Classes e Objetos

---

- ▶ Objetos de uma certa classe podem ter comportamentos (métodos);
- ▶ Sobre o comportamento...
  - ▶ É caracterizado pelo conjunto de operações que o objeto é capaz de executar (ou sua interface);
  - ▶ Podemos imaginar os objetos como prestadores de serviços;
  - ▶ Sempre que o objeto solicitar um serviço de outro objeto, deve enviar-lhe uma mensagem;
  - ▶ Se o objeto receptor for capaz de prestar este serviço, ele então será executado;
  - ▶ Podem afetar o estado (atributos) dos objetos.



# Classes e Objetos

---

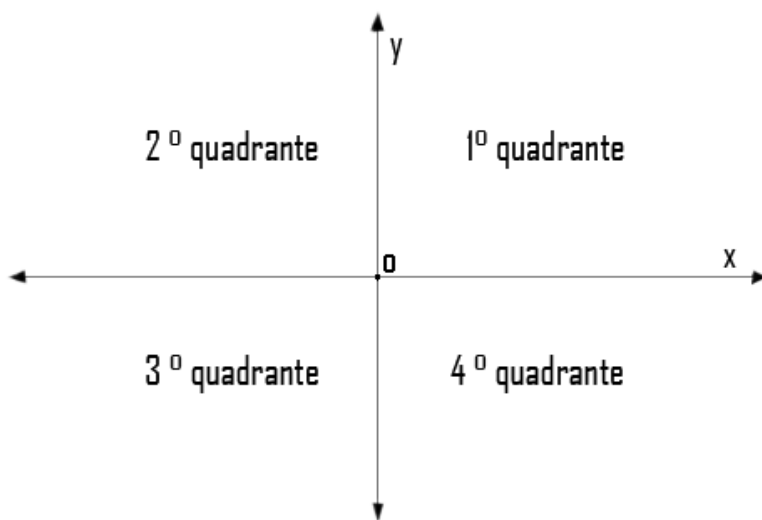
- ▶ **Objetos** de uma certa classe **têm comportamentos (métodos)**
  - ▶ Cada conta sofre a ação de um saque, depósito, extrato etc;
  - ▶ Cada pessoa pode correr, cantar etc;
- ▶ No caso, cada objeto da classe Ponto pode ter o método de identificar em qual quadrante ele se encontra no plano cartesiano.



# Classes e Objetos

---

- ▶ No caso, cada objeto da classe Ponto pode ter o comportamento de identificar em qual quadrante ele se encontra no plano cartesiano.



Mas por que a responsabilidade de saber o quadrante pertence ao objeto Ponto???



Porque apenas cada objeto ponto é que sabe os valores de suas coordenadas!!!!



# Métodos em Python

---

- ▶ São implementados como “funções” dentro da classe que representa seus objetos;
- ▶ Eles representam o comportamento do objeto;
- ▶ Veja o exemplo a seguir...



# Métodos em Python

```
1 class Ponto:
2     def __init__(self, x, y):
3         self.x=x
4         self.y=y
5     def quadrante(self):
6         if(self.x >0 and self.y>0):
7             return "1° quadrante"
8         elif (self.x <0 and self.y>0):
9             return "2° quadrante"
10        elif (self.x <0 and self.y<0):
11            return "3° quadrante"
12        elif (self.x >0 and self.y<0):
13            return "4° quadrante"
14
15        else:
16            return "Nenhum dos quadrantes"
17
18 ponto1 = Ponto(1,2)
19 print ('Coordenadas do Ponto 1 (%d,%d)'%(ponto1.x,ponto1.y))
20 print ('Ponto 1 pertence a(o) %s'%(ponto1.quadrante()))
```

Método que retorna a informação sobre qual quadrante o ponto pertence!!!

Enviando mensagem ao objeto ponto1, solicitando que ele informe seu quadrante!!!

# Métodos em Python

```
1 class Ponto:
2     def __init__(self, x, y):
3         self.x=x
4         self.y=y
5     def quadrante(self):
6         if(self.x > 0 and self.y > 0):
7             return "1° quadrante"
8         elif (self.x < 0 and self.y > 0):
9             return "2° quadrante"
10        elif (self.x < 0 and self.y < 0):
11            return "3° quadrante"
12        elif (self.x > 0 and self.y < 0):
13            return "4° quadrante"
14
15        else:
16            return "Nenhum dos quadrantes"
17
18 ponto1 = Ponto(1,2)
19 print ('Coordenadas do Ponto 1 (%d,%d)'%(ponto1.x,ponto1.y))
20 print ('Ponto 1 pertence a(o) %s'%(ponto1.quadrante()))
```

A definição do método informa que ele recebe um parâmetro (self), no entanto, a mensagem para ponto1 está sem argumentos. Por que???

# Métodos em Python

---

- ▶ Na verdade, está sendo passado sim um parâmetro, só que implícito...

```
1 class Ponto:
2     def __init__(self, x, y):
3         self.x=x
4         self.y=y
5     def quadrante(self):
```

```
ponto1.quadrante()
```

Ao invocar `quadrante()`, **ponto1** envia sua **referência**, implicitamente, ao método, e a mesma é **atribuída** ao parâmetro **self**.

## Exercício 2

---

a) Defina e implemente os métodos da classe Retangulo:

- ▶ calculaArea()
- ▶ ehQuadrado()

b) Invoque os métodos do item (a).





# Classes e Objetos

---

- ▶ Até agora vimos que as classes...
  - ▶ Descrevem os **atributos** (estados/características) e os **métodos** (comportamentos) de seus objetos;
  - ▶ Também possuem **métodos** especiais chamados de **Construtores!**



# Construtores

---

- ▶ Um construtor é um método especial que é executado todas as vezes que um objeto é criado;
- ▶ Ele é normalmente usado para fins de inicialização;
- ▶ Características de um construtor em Python:
  - ▶ O nome do construtor é `__init__` (com duplos *underscores* no início e fim);
  - ▶ Um construtor no mínimo possui um parâmetro (o *self*);
  - ▶ A chamada de um construtor é feita exclusivamente através do nome da classe, seguido pelos parâmetros definidos no construtor;
  - ▶ Um construtor não retorna valor.



# Construtores em Python

---

## ► Exemplo:

```
1  class Ponto:
2      def __init__(self, x, y):
3          self.x=x
4          self.y=y
```

Construtor

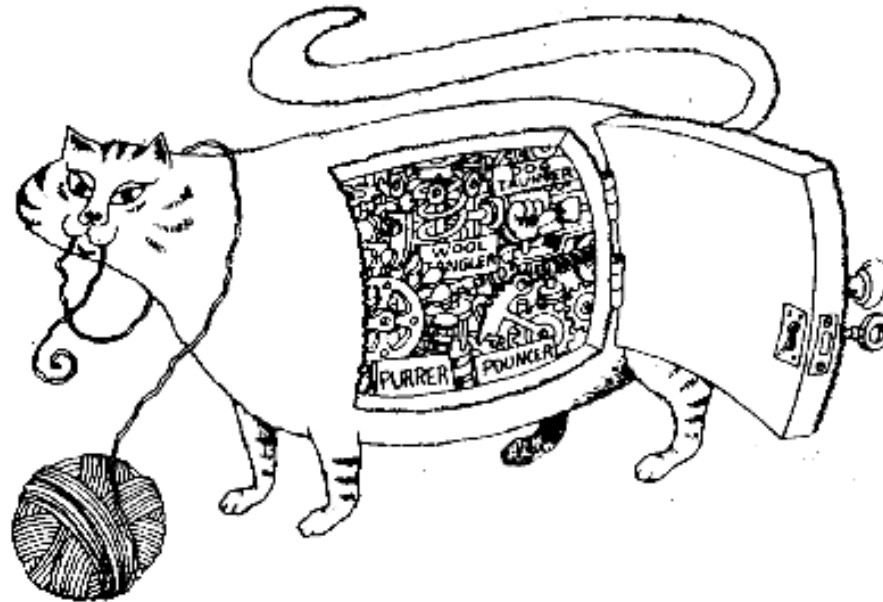
Atributos do objeto Ponto

Parâmetros para inicialização dos valores dos atributos. O **self** representa a auto-referência ao objeto criado.

# Encapsulamento

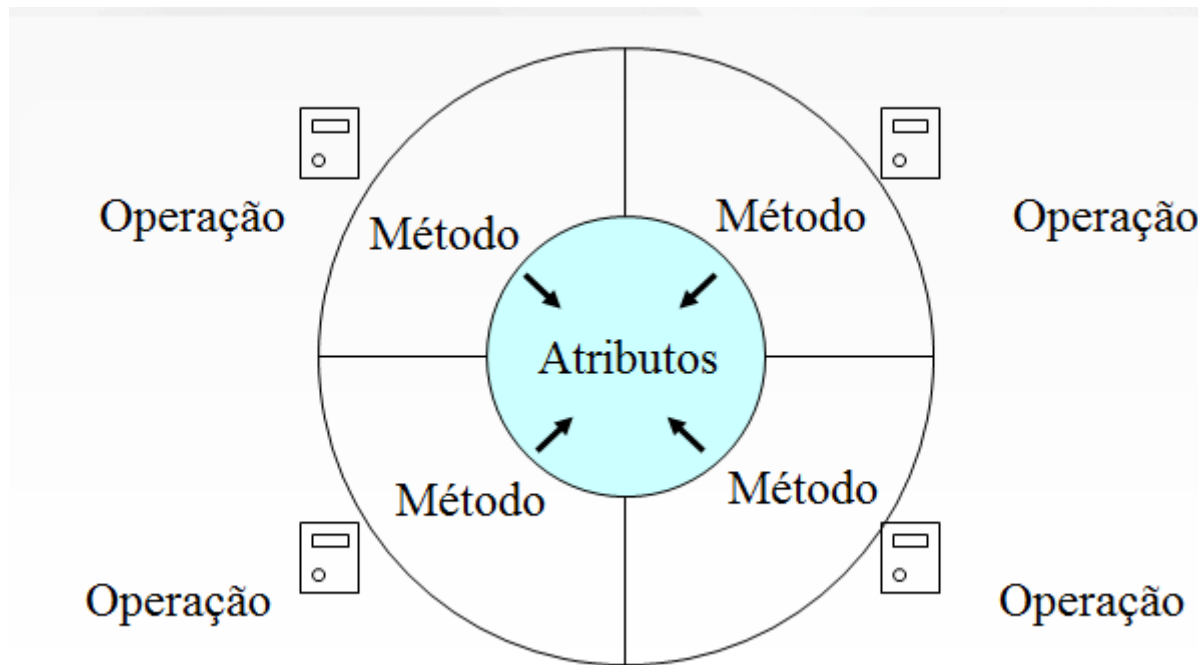
---

- ▶ É um conceito do paradigma OO que impede o acesso direto às propriedades dos objetos, visando aumentar o nível de abstração.



# Encapsulamento

---



# Encapsulamento em Python

```
1 class Ponto:
2     def __init__(self,x,y):
3         self.x = x
4         self.y = y
5
6 ponto1 = Ponto(1,2)
7 print("{}{}".format(ponto1.x,ponto1.y))
```

Atributos públicos!!!

- Usa-se um *underscore* prefixado ao nome do atributo, para “privá-lo”.

APENAS UMA CONVENÇÃO, com  
um único *underscore* prefixado!

```
1 class Ponto:
2     def __init__(self,x,y):
3         self._x = x
4         self._y = y
```

# Encapsulamento em Python

---

► Como ter acesso aos atributos privados, fora da classe???



**Através dos MÉTODOS!!!**



# Encapsulamento em Python

---

## ▶ Métodos acessadores

- ▶ Retornar os valores dos atributos privados;
- ▶ Métodos **get** (por convenção);
- ▶ Exemplo: `get_x`, `get_y`

## ▶ Métodos alteradores

- ▶ Alterar os valores dos atributos privados;
- ▶ Métodos **set** (por convenção);
- ▶ Exemplo: `set_x`, `set_y`

- ▶ Veja a seguir, tais métodos, na classe `Ponto`...





# Encapsulamento em Python

---

```
1  class Ponto:
2      def __init__(self,x,y):
3          self._x = x
4          self._y = y
5
6      # Métodos Acessadores
7      def get_x(self):
8          return self._x
9
10     def get_y(self):
11         return self._y
12
13     # Métodos Modificadores
14     def set_x(self,x):
15         self._x = x
16
17     def set_y(self,y):
18         self._y = y
```

```
22  p1 = Ponto(1,2)
23  print("{}{}".format(p1.get_x(),p1.get_y()))
```



# Encapsulamento em Python

- ▶ Há uma forma válida de privar os atributos, através do mecanismo chamado *name mangling*.
  - ▶ Existe com o intuito de evitar o conflito de nomes entre uma classe e subclasse;
  - ▶ Feito prefixando o nome dos atributos com duplo *underscore*.

```
1 class Ponto:
2     def __init__(self,x,y):
3         self.__x = x
4         self.__y = y
5
6 ponto1 = Ponto(1,2)
7 print("({},{})".format(ponto1.__x,ponto1.__y))
-
```

Name mangling ( duplo *underscore* prefixado)!

Tentativa de acesso ao atributo privado, *fora da classe*!

Saída:

Traceback (most recent call last):  
File "python", line 7, in <module>  
**AttributeError: 'Ponto' object has no attribute '\_\_x'**

# Exercício

---

Modifique a classe ContaCorrente...

1. Privando os atributos dos seus objetos;
2. Adicionando métodos modificadores (alteradores) e acessadores para tais atributos;
3. Fazendo alterações necessárias nos demais métodos.



# Relacionamento entre Objetos

---

- ▶ Até o presente momento, lidamos com objetos simples compostos por atributos de tipos primitivos ou que se relacionam apenas com outro objeto;
  - ▶ Os atributos das instâncias de Ponto, 'x' e 'y', são valores numéricos.
- ▶ Um sistema orientado a objetos é composto por dezenas de classes que se relacionam entre si para solucionar o problema proposto.



# Relacionamento entre Objetos

---

- ▶ Qualquer programa geralmente é composto por **diversos objetos**
  - ▶ **Eles relacionam entre si** para executar o propósito do programa;
- ▶ Suponha que temos uma classe Motor e uma classe Carro.
  - ▶ Essas classes relacionam-se entre si:
    - ▶ Um Carro possui um Motor;
    - ▶ De outra forma, um Motor pertence a um Carro.



# Relacionamento entre Objetos

---

```
1 class Motor:
2     #Construtor
3     def __init__(self, motorizacao, combustivel='flex'):
4         self.__motorizacao = motorizacao
5         self.__combustivel = combustivel
6
7     #Métodos Acessarores
8
9     def get_motorizacao(self):
10        return self.__motorizacao
11    def get_combustivel(self):
12        return self.__combustivel
13
14    #Métodos alteradores
15    def set_motorizacao(self, nova_motorizacao):
16        self.__motorizacao = nova_motorizacao
17
18    def set_combustivel(self, novo_combustivel):
19        self.__combustivel = novo_combustivel
20
21    #Método que retorna uma representação do objeto, em string.
22    def __str__(self):
23        return "Motor:{}L, Combustivel:{}".format(self.__motorizacao, self
            .__combustivel)
```

# Relacionamento entre Objetos

```
1 class Carro:
2     #Construtor
3     def __init__(self, cor, placa, motor, dimensao):
4         self.__cor = cor #string
5         self.__placa = placa #string
6         self.__motor = motor #instancia de Motor
7         self.__dimensao = dimensao #instancia de Dimensao
8
9     #Metodos acessadores
10
11     def get_cor(self):
12         return self.__cor
13     def get_placa(self):
14         return self.__placa
15     def get_motor(self):
16         return self.__motor
17     def get_dimensao(self):
18         return self.__dimensao
19
20     #Método alterador
21     def set_placa(self, nova_placa):
22         self.__placa = nova_placa
23
24     #Método mágico que retorna o objeto representado numa string
25     def __str__(self):
26         return 'Cor:{}, Placa: {}, \nMotorização: {}, \nDimensão: {}'
27         .format(
28             self.__cor, self.__placa, self.__motor, self.__dimensao)
```

Relação entre Carro e Motor, definido pelo atributo `__motor`

# Relacionamento entre Objetos

```
1 class Carro:
2     #Construtor
3     def __init__(self, cor, placa, motor, dimensao):
4         self.__cor = cor #string
5         self.__placa = placa #string
6         self.__motor = motor #instancia de Motor
7         self.__dimensao = dimensao #instancia de Dimensao
8
9     #Metodos acessadores
10
11     def get_cor(self):
12         return self.__cor
13     def get_placa(self):
14         return self.__placa
15     def get_motor(self):
16         return self.__motor
17     def get_dimensao(self):
18         return self.__dimensao
19
20     #Método alterador
21     def set_placa(self, nova_placa):
22         self.__placa = nova_placa
23
24     #Método mágico que retorna o objeto representado numa string
25     def __str__(self):
26         return 'Cor:{}, Placa: {}, \nMotorização: {}, \nDimensão: {}'
27         .format(
28             self.__cor, self.__placa, self.__motor, self.__dimensao)
```

Carro possui relação, também, com a classe Dimensão.



# Relacionamento entre Objetos

---

```
1 class Dimensao:
2     #Construtor
3     def __init__(self, altura, largura, comprimento):
4         self.__altura = altura;
5         self.__largura = largura
6         self.__comprimento = comprimento
7
8     #Métodos Acessadores
9     def get_altura(self):
10         return self.__altura
11     def get_largura(self):
12         return self.__largura
13     def get_comprimento(self):
14         return self.__comprimento
15
16     #Métodos Alteradores
17     def set_altura(self, nova_altura):
18         self.__altura = nova_altura
19     def set_largura(self, nova_largura):
20         self.__largura = nova_largura
21     def set_comprimento(self, novo_comprimento):
22         self.__comprimento = novo_comprimento
23
24     #Método mágico que retorna o objeto representado numa string
25     def __str__(self):
26         return "Altura: {}, Largura: {}, Comprimento:{}".format(
27             self.__altura, self.__largura, self.__comprimento)
```

# Criando e Relacionando os Objetos

---

```
1 from motor import Motor
2 from dimensao import Dimensao
3 from carro import Carro
4
5 #Criando um motor 1.6L a gasolina
6 motor = Motor('1.6','gasolina')
7
8 #Criando um objeto dimensão
9 dimensao = Dimensao(1.67,1.81,4.37)
10
11 #Criando um Carro, relacionando com os objetos motor e dimensao
12 carro = Carro('preto','XXX1234',motor,dimensao)
13 #Imprimindo o carro
14 print(carro)
```

Python 3.6.1 (default, Dec 2015, 13:05:11)

[GCC 4.8.2] on linux

>

Cor:preto, Placa: XXX1234,

Motorização: Motor:1.6L, Combustivel:gasolina,

Dimensão: Altura: 1.67, Largura: 1.81, Comprimento:4.37

# Criando e Relacionando os Objetos

---

```
1  from motor import Motor
2  from dimensao import Dimensao
3  from carro import Carro
4
5  #Criando um motor 1.6L a gasolina
6  motor = Motor('1.6','gasolina')
7
8  #Criando um objeto dimensão
9  dimensao = Dimensao(1.67,1.81,4.37)
10
11 #Criando um Carro, relacionando com os objetos motor e dimensao
12 carro = Carro('preto','XXX1234',motor,dimensao)
13 #Imprimindo o carro
14 print(carro)
15
```

*Como alterar a motorização do Carro para 2.0?*



# Criando e Relacionando os Objetos

---

```
1  from motor import Motor
2  from dimensao import Dimensao
3  from carro import Carro
4
5  #Criando um motor 1.6L a gasolina
6  motor = Motor('1.6','gasolina')
7
8  #Criando um objeto dimensão
9  dimensao = Dimensao(1.67,1.81,4.37)
10
11 #Criando um Carro, relacionando com os objetos motor e dimensao
12 carro = Carro('preto','XXX1234',motor,dimensao)
13 #Imprimindo o carro
14 print(carro)
15
16 #Alterando o motor do carro
17 carro.get_motor().set_motorizacao('2.0')
```

Retorna uma instância de Motor



# Criando e Relacionando os Objetos

---

```
1  from motor import Motor
2  from dimensao import Dimensao
3  from carro import Carro
4
5  #Criando um motor 1.6L a gasolina
6  motor = Motor('1.6','gasolina')
7
8  #Criando um objeto dimensão
9  dimensao = Dimensao(1.67,1.81,4.37)
10
11 #Criando um Carro, relacionando com os objetos motor e dimensao
12 carro = Carro('preto','XXX1234',motor,dimensao)
13 #Imprimindo o carro
14 print(carro)
15
16 #Alterando o motor do carro
17 carro.get_motor().set_motorizacao('2.0')
```

*Como diminuir a altura do Carro para 1.65?*



# Criando e Relacionando os Objetos

---

```
1  from motor import Motor
2  from dimensao import Dimensao
3  from carro import Carro
4
5  #Criando um motor 1.6L a gasolina
6  motor = Motor('1.6','gasolina')
7
8  #Criando um objeto dimensão
9  dimensao = Dimensao(1.67,1.81,4.37)
10
11 #Criando um Carro, relacionando com os objetos motor e dimensao
12 carro = Carro('preto','XXX1234',motor,dimensao)
13 #Imprimindo o carro
14 print(carro)
15
16 #Alterando o motor do carro
17 carro.get_motor().set_motorizacao('2.0')
18
19 #Alterando a altura do carro
20 carro.get_dimensao().set_altura(1.65)
21
22 #Imprimindo novamente o carro
23 print(carro)
```



# Criando e Relacionando os Objetos

---

```
1 from motor import Motor
2 from dimensao import Dimensao
3 from carro import Carro
4
5 #Criando um motor 1.6L a gasolina
6 motor = Motor('1.6','gasolina')
7
8 #Criando um objeto dimensão
9 dimensao = Dimensao(1.67,1.81,4.37)
10
11 #Criando um Carro, relacionando com os objetos motor e dimensao
12 carro = Carro('preto','XXX1234',motor,dimensao)
13 #Imprimindo o carro
14 print(carro)
15
16 #Alterando o motor do carro
17 carro.get_motor().set_motorizacao('2.0')
18
19 #Alterando a altura do carro
20 carro.get_dimensao().set_altura(1.65)
21
22 #Imprimindo novamente o carro
23 print(carro)
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
```



```
Cor:preto, Placa: XXX1234,
Motorização: Motor:1.6L, Combustivel:gasolina,
Dimensão: Altura: 1.67, Largura: 1.81, Comprimento:4.37
Cor:preto, Placa: XXX1234,
Motorização: Motor:2.0L, Combustivel:gasolina,
Dimensão: Altura: 1.65, Largura: 1.81, Comprimento:4.37
```



# Exercício

---

1. Mais uma vez, modifique a classe ContaCorrente...
  - a) Removendo o atributo `__nomeTitular` (junto com os métodos que o manipulam).
  - b) Adicione o atributo `__cliente` (objeto da classe Cliente).
2. Crie uma classe Cliente, cujos atributos são `__nome` e `__cpf`. Defina na classe um construtor, métodos acessadores e alteradores, além do `__str__`;
3. Crie uma conta;
  1. através dela, altere o nome de seu titular;
  2. imprima o nome do titular.





# Bibliografia

---

- ▶ Python 3.7 Tutorial, disponível em :  
<https://docs.python.org/3/tutorial/classes.html>

