

# LINGUAGENS DE PROGRAMAÇÃO

Engenharia Informática




# Nesta Aula...

- Métodos
  - *Instância/Objeto*
  - *Classe*
  - *Estáticos*
- Atributos
  - *Instância*
  - *Classe*
- Construtor

# Class vs Objecto

- Classe define atributos e métodos da classe
  - *nomes e verbos*
- Objecto é a instânciação de uma classe.

**class** Pessoa():  Definição  
**pass**

pessoa\_1 = Pessoa()  Instânciação  
 variável  atribuição

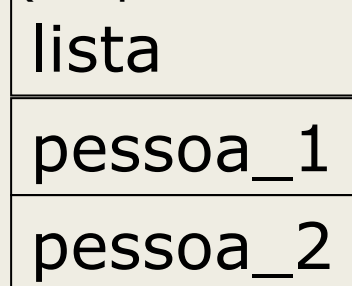
# Instância - Referências

```
lista = []
```

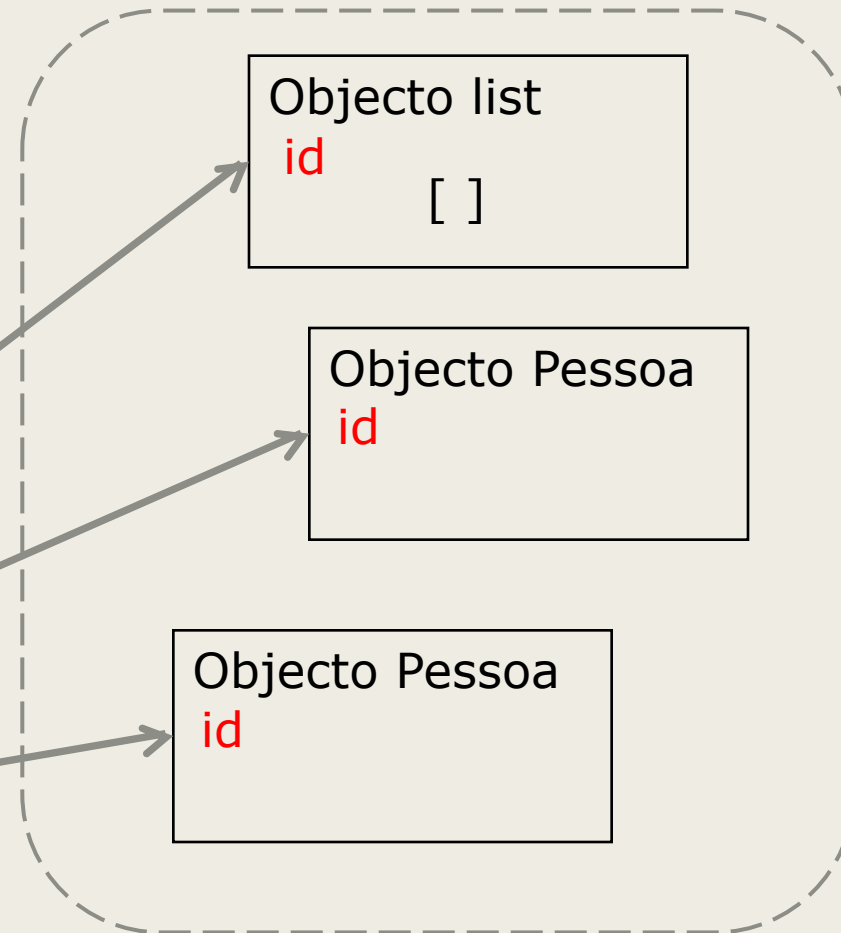
```
class Pessoa():  
    pass
```

```
pessoa_1 = Pessoa()  
pessoa_2 = Pessoa()
```

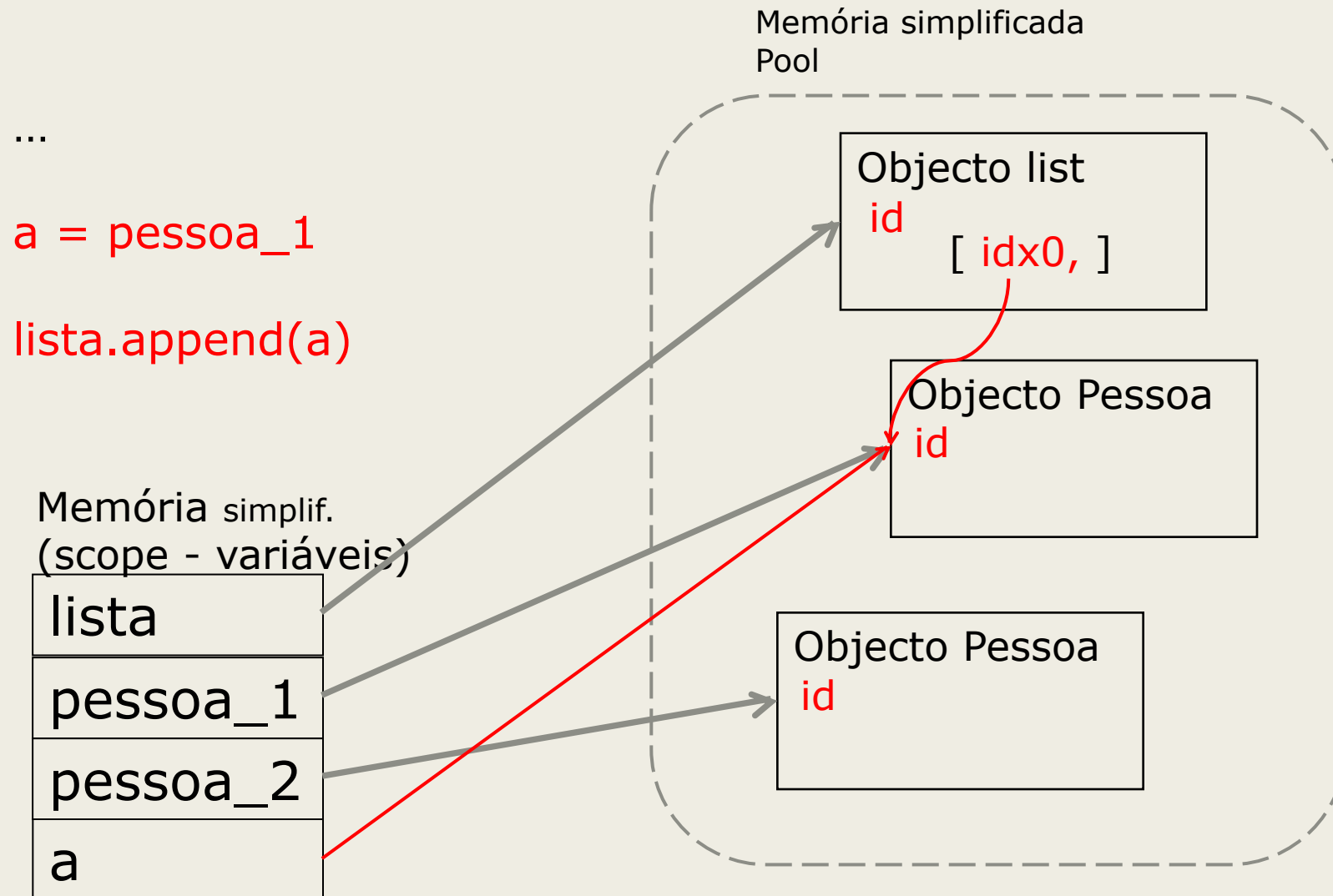
Memória simplif.  
(scope - variáveis)



Memória simplificada  
Pool



# Instância - Referências



# Métodos

- Método (method) é uma Função que é um membro de uma classe.
- Deve reflectir acções sobre com atributos dessa classe.
- O nome da função (normalmente) começa por um verbo
  - *Ex: fala, caminha, saca, codifica/descodifica*

# Métodos

```
class Pessoa():  
    def fala(self):  
        print ('Sou um objecto Pessoa id:', id(self))  
        print('Aka:', self.nome)
```

```
peessoa_1 = Pessoa()  
peessoa_1.nome = "Trump"  
peessoa_1.fala()
```

- Através de self é possível aceder a todos os membros e atributos associados (bounded) ao objecto instanciado.

# Métodos

```
class Pessoa():  
    def fala(self):  
        print ('Sou um objecto Pessoa id:', id(self))  
        print ('Aka:', self.nome)
```

```
peessoa_1 = Pessoa()  
peessoa_1.nome = "Trump"  
peessoa_1.fala()  
Pessoa.fala() ✖
```

TypeError: fala() missing 1 required positional argument: 'self'

- Através de self é possível aceder a todos os membros e atributos associados (bounded) ao objecto instanciado.



# Métodos estáticos

```
class Pessoa():  
    @staticmethod  
    def obter_numero_de_membros():  
        return 2 + 2
```

```
peessoa_1 = Pessoa()  
peessoa_1.obter_numero_de_membros()  
Pessoa.obter_numero_de_membros()
```

# Métodos estáticos

- Métodos sem o argumento `self`.
- Podiam ser colocados fora da classe como uma função genérica.
  - *Por estar dentro, mantêm a lógica da classe mais autocontida.*
- Métodos estáticos servem para organizar/agrupar código, relacionado com a class mas que não manipulam o objecto (`self`).

# Métodos estáticos

- Exemplo:

```
class Pizza(object):  
    @staticmethod  
    def mistura_ingredientes(x, y):  
        return x + y  
  
    def cozinha(self):  
        return self.mistura_ingredientes(self.queijo, self.vegetais)
```

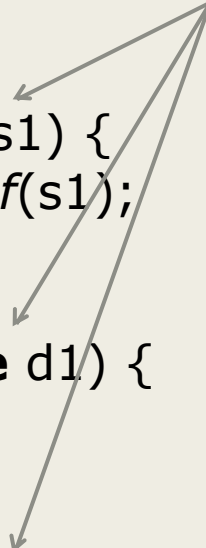
- Se `mistura_ingredientes` não fosse static, funcionaria da mesma forma:
- O *decorator* `staticmethod` permite retirar o `self`,
  - *poupa a associação de um método ao objecto*
  - *melhora a legibilidade do código*
- Permite o `reescrever` (override) polimorfismo...

# Métodos com assinatura múltipla

## ■ In Java:

```
public class Foo {  
  
    public int getInt(String s1) {  
        return Integer.valueOf(s1);  
    }  
  
    public int getInt(double d1) {  
        return (int) d1;  
    }  
  
    public int getInt(String s1, int dflt) {  
        if (s1 == null)  
            return dflt;  
        return Integer.valueOf(s1);  
    }  
}
```

Mesmo nome  
Assinaturas diferentes



# Métodos com assinatura múltipla

- In Python:

- *Não podemos ter vários métodos com o mesmo nome, ainda que tenham assinatura diferente.*

- *Mas temos:*

- *Required args*
    - *Keyword args*
    - *Default args*

```
class Foo():  
    @staticmethod  
    def getInt(s1, dflt=None):  
        if isinstance(s1, str):  
            if s1 is None:  
                return dflt  
            return int(s1)  
        if isinstance(s1, float):  
            return int(s1)
```

## ... por falar em **None**

- **None** é equivalente ao **null** do Java.
  - *Mais ou menos.*
  - *Simboliza que o conteúdo está vazio e que não tem nenhum objecto associado*
- Uma variável, índice de lista, valor de dicionário, argumento no tuple... quando existe em memória tem **SEMPRE** que referenciar um objecto.
  - *Nem que seja o objecto **None***
  - *Ou então não existe*

```
>>> id(None)
>>> sys.getrefcount(None)
>>> a = 5
>>> a = None
>>> del a
```

# Atributos

- Variáveis intrínsecas a um objecto!
  - *Já vimos na aula passada...*

```
class Pessoa():  
    pass
```

```
manel = Pessoa()  
manel.nome = "Manuel"  
manel.idade = 40
```

```
print(manel.nome)  
print(manel.__dict__)
```

**Atributos** do objecto  
referenciado por  
manel

Manuel  
{'nome': 'Manuel', 'idade': 40}

# Atributos da classe

- Variáveis intrínsecas a um objecto classe!
  - *Sim, objecto Classe!*
  - *Em Python TUDO é objecto, incluindo as classes.*
  - *Então a classe também pode ter os seus atributos próprios... e métodos.*
- Atributos da classe são partilhados pelo objecto classe e por TODOS as instâncias da classe.



# Atributos da classe

```
class Pessoa():  
    numero_bracos = 2  
pass
```

```
Pessoa.numero_pernas = 2
```

```
manel = Pessoa();  
xico = Pessoa()
```

```
print(manel.numero_bracos)  
print(xico.numero_bracos)  
print(Pessoa.numero_bracos)
```

```
print(manel.__dict__)  
print(Pessoa.__dict__)
```

**Atributos** da classe

2  
2  
2

{}

{'numero\_bracos': 2, ...}

# Atributos da classe

```
class Pessoa():  
    numero_bracos = 2  
pass
```

```
Pessoa.numero_pernas = 2
```

```
manel = Pessoa(); manel.numero_bracos = 3  
xico = Pessoa()
```

```
print(manel.numero_bracos)  
print(xico.numero_bracos)  
print(Pessoa.numero_bracos)
```

```
print(manel.__dict__)  
print(Pessoa.__dict__)
```

1º pesquisa no objecto  
2º pesquisa na classe  
apenas para leitura

**3**  
2  
2

{'numero\_bracos': 3}  
{'numero\_bracos': 2, ...}

# Métodos de classe

- Acessíveis pela classe.
- Podem aceder a atributos da classe.

```
class Pessoa():  
    numero_bracos = 2  
    numero_pernas = 2  
  
    @ classmethod  
    def get_membros(cls):  
        return cls.numero_bracos + cls.numero_pernas
```

```
manel = Pessoa();
```

```
Pessoa.get_membros() #Preferível  
manel.get_membros() #Funciona depois de esgotar pesquisa
```

# Construtor

- Em Java
  - *Pessoa a = new Pessoa();*
- Em Python
  - *a = Pessoa()*
- Estamos a invocar a construção de um novo objecto Pessoa.
  - *Alocar memória na pool. Atribuir um endereço (id)*
  - *Associar (bound) métodos e atributos.*

# Construtor

- Em Python há o método `__init__()`
  - *aka Método construtor*
  - *Que na verdade não é construtor, mas inicializador.*
  - *O método `__new__()` é que trata da construção.*
    - Pode ser útil em cenários avançados.

```
class Pessoa():
```

```
    def __init__(self, nome, idade):
```

```
        self.nome = nome
```

```
        self.idade = idade
```

← Inicialização  
dos atributos

```
manel = Pessoa("Manuel", 40)
```

# Documentários

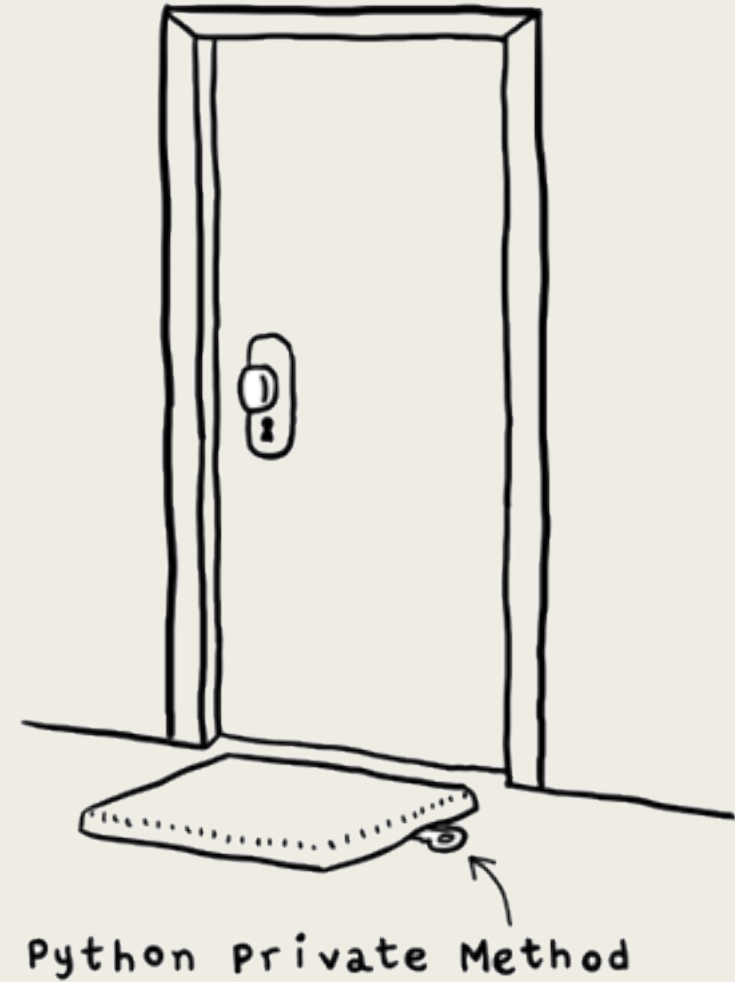
```
class Pessoa():  
    """ Pessoa encapsula um nome e uma idade """  
    def __init__(self, nome, idade):  
        """  
        Inicializa um novo objeto Pessoa  
        :param nome: o nome da pessoa  
        :param idade: a idade da pessoa  
        :returns nothing """  
        self.nome = nome  
        self.idade = idade  
  
help(Pessoa)
```

# Naming Conventions

- *PEP8 - Style Guide for Python Code*
- <https://www.python.org/dev/peps/pep-0008>
- Nomes de Classes
  - CapWords aka upper CamelCase
  - Pode seguir o estilo de funções quando a finalidade principal da classe é ser callable.
- Métodos
  - minúsculas\_separadas\_por\_underscore
  - mixedCase aka lower camelCase, pode utilizado quando é predominante no código existente.
    - Java style

# Próxima aula...

## ■ Atributos Privados





# Fim

- Questões?