

BIG DATA



LABDATA



FUNDAÇÃO
INSTITUTO DE
ADMINISTRAÇÃO

Introdução ao Big Data

Tema da Aula: **Introdução ao Python**

Prof.: **Dino Magri**

24 de Outubro de 2018

Coordenação:

Prof. Dr. Adolpho Walter
Pimazzi Canton

Profa. Dra. Alessandra de
Ávila Montini

- Contatos:

- E-mail: professor.dinomagri@gmail.com
- Twitter: https://twitter.com/prof_dinomagri
- LinkedIn: <http://www.linkedin.com/in/dinomagri>
- Site: <http://www.dinomagri.com>

Coordenação:

Prof. Dr. Adolpho Walter
Pimazzi Canton

Profa. Dra. Alessandra de
Ávila Montini

Currículo

- **(2014-Presente)** – Professor no curso de Extensão, Pós e MBA na Fundação Instituto de Administração (FIA) – www.fia.com.br
- **(2013-Presente)** – Pesquisa e Desenvolvimento no Laboratório de Arquitetura e Redes de Computadores (LARC) na Universidade de São Paulo – www.larc.usp.br
- **(2013)** – Professor no MBA em Desenvolvimento de Inovações Tecnológicas para WEB na IMED Passo Fundo – RS – www.imed.edu.br
- **(2012)** – Bacharel em Ciência da Computação pela Universidade do Estado de Santa Catarina (UDESC) – www.cct.udesc.br
- **(2009/2010)** – Pesquisador e Desenvolvedor no Centro de Computação Gráfica – Guimarães – Portugal – www.ccg.pt
- **Lattes:** <http://lattes.cnpq.br/5673884504184733>

Material das aulas

- Material das aulas estão disponíveis em:
 - <https://urls.dinomagri.com/posmba-turma9>
 - **Senha:** turma9
 - **Data Expiração:** 31-12-2018

Material das aulas

- Caso esteja utilizando seu próprio computador, realize o download de todos os arquivos e salve na **Área de Trabalho** para facilitar o acesso.
 - Lembre-se de instalar os softwares necessários conforme descrito no documento de Instalação (**InstalaçãoPython3v1.1.pdf**).
- Nos computadores da FIA os arquivos já estão disponíveis, bem como a instalação dos softwares necessários.

Conteúdo da Aula

- Objetivo
- Funções
- Classes e Objetos
- Exercícios

Conteúdo da Aula

- **Objetivo**
- Funções
- Classes e Objetos
- Exercícios

Objetivo

- O objetivo dessa aula é **introduzir conceitos básicos sobre** a linguagem de programação **Python** para Big Data.

Conteúdo da Aula

- Objetivo
- **Funções**
- Classes e Objetos
- Exercícios

Funções

- Até agora, vimos diversos tipos de dados, atribuições, comparações e estruturas de controle.
- A ideia é **dividir para conquistar**, onde:
 - Um problema é dividido em diversos subproblemas
 - As soluções dos subproblemas são combinadas na solução do problema maior.
- Esses subproblemas têm o nome de **funções**.

Funções

- Funções possibilitam capturar a computação realizada e tratá-la como **primitiva**.
- Como exemplo, queremos que a variável **z** seja o máximo de dois números (x e y).
- Um programa simples seria:

```
>>> if x > y:
        z = x
    else:
        z = y
```

Funções

- A ideia da função é encapsular essa computação dentro de um escopo que pode ser tratado como primitiva.
 - Sendo que os detalhes internos estão escondidos dos usuários
 - Para utilizá-las, basta chamar o nome da função e fornecendo os parâmetros necessários.
- Uma função tem **3 partes importantes**:
 - Nome, parâmetros e corpo da função

Funções em Python

```
def <nome> ( <parametros> ) :
```

```
    <corpo da função>
```

- `def` é uma palavra chave e serve para definir uma função.
- `<nome>` é qualquer nome aceito pelo Python.
- `<parametros>` é a quantidade de parâmetros que será passado para a função (pode ser nenhum).
- `<corpo da função>` contém o código da função.

Funções

- Voltando ao nosso exemplo, podemos reescrever:

```
def maximo(x, y):
```

```
    if x > y:
```

```
        z = x
```

```
    else:
```

```
        z = y
```

- Ótimo **temos** uma **função** e podemos reaproveitá-la.
- Porém, para tratá-la como primitiva precisamos retornar o valor, desta forma, utilizamos o comando `return`

Funções

- Voltando ao nosso exemplo, podemos reescrever:

```
def maximo(x, y):
```

```
    if x > y:
```

```
        return x
```

```
    else:
```

```
        return y
```

- Agora sim! Já podemos reaproveitar nossa função!
- **E como podemos fazer isso?**

Funções

- Voltando ao nosso exemplo:

```
>>> def maximo(x, y):  
    if x > y:
```

```
        return x
```

```
    else:
```

```
        return y
```

Desta forma,
o retorno é
atribuído na
variável z

- Agora podemos chamar a função:

```
>>> z = maximo(3, 4)
```

```
>>> print(z)
```

```
4
```

2
Todas as expressões são avaliadas, e caso não se encontre correspondência, é retornado o valor `None`.

3
Ou até que encontre a palavra especial **return**

1
Quando chamamos a função `maximo(3, 4)` estamos definindo que `x = 3` e `y = 4`.

Funções

- Já entendemos o que é e como criar funções.
- Vamos testar e codificar um pouco mais 😊

 Abra o arquivo "**aula2-parte1-funcoes.ipynb**"

Exercício de 5 minutos

- Crie uma função que receba dois argumentos:
 - O primeiro argumento é o valor de um determinado serviço.
 - O segundo é a porcentagem da multa por atraso do pagamento. O valor padrão da porcentagem, se não passado, é de 7%.
 - A função deve retornar o valor final da conta com o juros. Lembre-se de converter possíveis valores.



Abra o arquivo **"aula2-parte1-funcoes.ipynb"**

Funções Embutidas

- Python já vem com diversas funções embutidas, que estão prontas para serem utilizadas.
 - Uma lista completa pode ser visualizada em <https://docs.python.org/3/library/functions.html>
- **Já utilizamos algumas delas! Quais?**

Funções Embutidas

Built-in Functions				
<code>abs()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	

Funções Embutidas

- Vamos **utilizar** duas funções:
 - `input`
 - `open`

input

- Essa função permite que o usuário digite uma entrada, por exemplo:

```
>>> idade = input('Digite sua idade:')
```

```
>>> print(idade)
```

input

- Essa função permite que o usuário digite uma entrada, por exemplo:

```
>>> idade = input('Digite sua idade:')
```

```
>>> print(idade)
```

```
>>> print(type(idade))
```

É importante notar que a função input retorna o valor digitado pelo usuário como sendo do tipo string.

open

- A função `open`, permite abrir um arquivo para leitura e escrita.

```
>>> arq = open(nome_arquivo, modo)
```

- Sendo que o `modo` pode ser:

r : abre o arquivo para leitura.

w : abre o arquivo para escrita.

a : abre o arquivo para escrita acrescentando os dados no final do arquivo.

+ : pode ser lido e escrito simultaneamente.

open - write

- O método `write` deve ser utilizado se o arquivo foi aberto para escrita. O código abaixo abre o `arquivo.txt` para escrita e adiciona 4 linhas:

```
>>> arq = open("arquivo.txt", "w")  
  
>>> for i in range(1, 5):  
    arq.write('{} . Escrevendo em arquivo\n'.format(i))  
  
>>> arq.close()
```

open - read

- O método `read` carrega todo o conteúdo do arquivo em uma única string.

```
>>> arq = open("arquivo.txt", "r")
```

```
>>> texto = arq.read()
```

```
>>> print(texto)
```

```
1. Escrevendo em arquivo  
2. Escrevendo em arquivo  
3. Escrevendo em arquivo  
4. Escrevendo em arquivo
```

```
>>> arq.close()
```

open - readlines

- O método `readlines` salva todo o conteúdo do arquivo em uma lista, onde cada linha do arquivo será um elemento da lista.

```
>>> arq = open("arquivo.txt", "r")
```

```
>>> texto = arq.readlines()
```

```
>>> print(texto)
```

```
['1. Escrevendo em arquivo\n', '2. Escrevendo em arquivo\n', '3.  
Escrevendo em arquivo\n', '4. Escrevendo em arquivo\n']
```

```
>>> arq.close()
```

Funções

- Vamos **testar** duas funções:
 - `input`
 - `open`



Abra o arquivo "**aula2-parte1-funcoes.ipynb**"

Conteúdo da Aula

- Objetivo
- Funções
- **Classes e Objetos**
- Exercícios

Classes e Objetos

- Até agora estamos utilizando a palavra objeto genericamente.
- Os códigos que escrevemos foram baseados em objeto, onde passamos **objetos** para os nossos programas.
- Utilizamos objetos em expressões, métodos entre outros.
- Para que o nosso código seja qualificado como **orientado à objeto (OO)**, precisamos que os objetos participem de uma estrutura chamada de **hierarquia de herança**.

Classes e Objetos

- **Cada objeto que iremos criar será uma instância de uma classe!**
- Uma classe Python é uma estrutura de codificação e dispositivo utilizado para **implementar novos tipos de objetos em Python.**
- Python tem como principal ferramenta **a programação Orientada à Objetos** e iremos aprender alguns conceitos envolvidos nesse paradigma.

Classes e Objetos

- A definição de classe tipicamente especifica as variáveis da instância, que é o **objeto que contém as ações (métodos/funções) que podem ser executados**.
- Os principais objetivos da programação orientada a objetos são:

Classes e Objetos

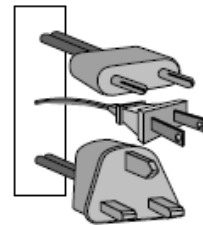
- **Robustez**

- Um programa deve **produzir a saída certa** para todas as entradas do programa. Além disso, é importante que o **software** seja capaz de **lidar com entradas inesperadas** que não são explicitamente definidas para sua aplicação.



- **Adaptabilidade**

- Software precisa ser capaz de **evoluir** junto as mudanças de ambiente.
- Também está relacionado a adaptabilidade e a **portabilidade**, que é a capacidade do software para executar com modificações mínimas em **diferentes plataformas.**



Classes e Objetos

- **Reusabilidade**

- O mesmo código pode ser utilizado como **um componente** de sistemas diferentes em várias aplicações.
- Desenvolver software é caro e portanto é importante **reutilizar códigos**.



- **Modularidade**

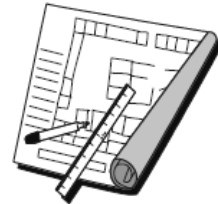
- Softwares consistem de **vários componentes diferentes** que devem interagir corretamente para que todo o sistema funcione.
- Modularidade refere-se a um princípio organizacional em que **diferentes componentes são divididos em unidades funcionais separadas**.



Classes e Objetos

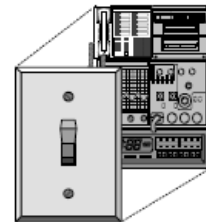
- **Abstração**

- A noção de abstração serve para **simplificar as partes cruciais de um sistema complexo**. Isso é feito, nomeando e explicando essas partes e como deve ser a operação de cada uma.



- **Encapsulamento**

- Diferentes componentes de um sistema de software não devem revelar os detalhes internos das suas respectivas implementações.
- **É importante manter uma interface pública para esse componente.**



Classes e Objetos



- Uma classe serve como o principal meio de abstração na programação orientada à objeto.
- A classe fornece um conjunto de comportamentos na forma de métodos (funções), com **implementações que são comuns a todas as instâncias** dessa classe.

Classes e Objetos

```
class Pessoa(object):  
    def __init__(self, nome, idade):  
        self.nome = nome  
        self.idade = idade
```

- Para criar uma classe em Python, temos as seguintes regras:
 - Deve iniciar com a palavra **class** seguida do nome da classe com a primeira letra em maiúscula.
 - Deve terminar com dois pontos no final da linha para abrir o bloco de código.
 - Toda classe associa atributos e métodos numa só estrutura. **Um objeto é uma variável cujo tipo é uma classe, ou seja, um objeto é um instância de uma classe.**
 - Deve conter os métodos que serão utilizados.

Construtor da classe

```
class Pessoa(object):  
    def __init__(self, nome, idade):  
        self.nome = nome  
        self.idade = idade
```

 Abra o arquivo **"aula2-parte2-classes.ipynb"**

Construtor da classe

```
class Pessoa(object):  
    def __init__(self, nome, idade):  
        self.nome = nome  
        self.idade = idade
```

- Quando instanciamos a classe, internamente, é realizado uma chamada do método especial `__init__` que serve como construtor da classe. A quantidade de parâmetros definidos no `__init__` deve ser passado como parâmetro quando a classe for instanciada.
- A principal responsabilidade é **estabelecer o estado do novo objeto** com a instância apropriada das variáveis.

Construtor da classe

```
class Pessoa(object):  
    def __init__(self, nome, idade):  
        self.nome = nome  
        self.idade = idade
```

- **O self, identifica a instância em que um método é invocado.**
- Todos os métodos de instâncias **devem declarar o self** como primeiro parâmetro!
- Todos os acessos a atributos (inclusive métodos) das instâncias devem ser feitos via **referência explícita a self**.

Herança

- O modelo de hierarquia é muito útil no desenvolvimento de software, pois promove a **reutilização de código**.
- Na programação orientada a objetos, esse modelo é conhecido como **Herança**.
- Isto permite que **uma nova classe seja definida com base em uma classe existente como o ponto de partida**.

Herança

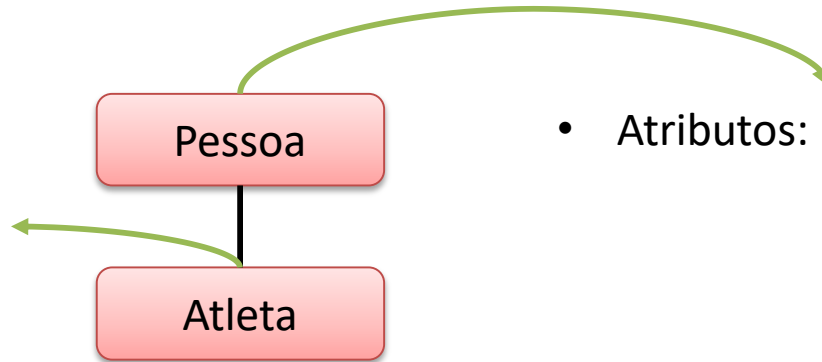
- Na POO, uma classe existente, pode ser descrita como:
 - Classe Base
 - Classe Pai
 - Classe Filha

Herança

- Uma classe filha pode **especializar** um comportamento existente, fornecendo uma nova aplicação que substitui um método existente.
- E também pode **estender** a classe pai, fornecendo novos métodos.

Herança

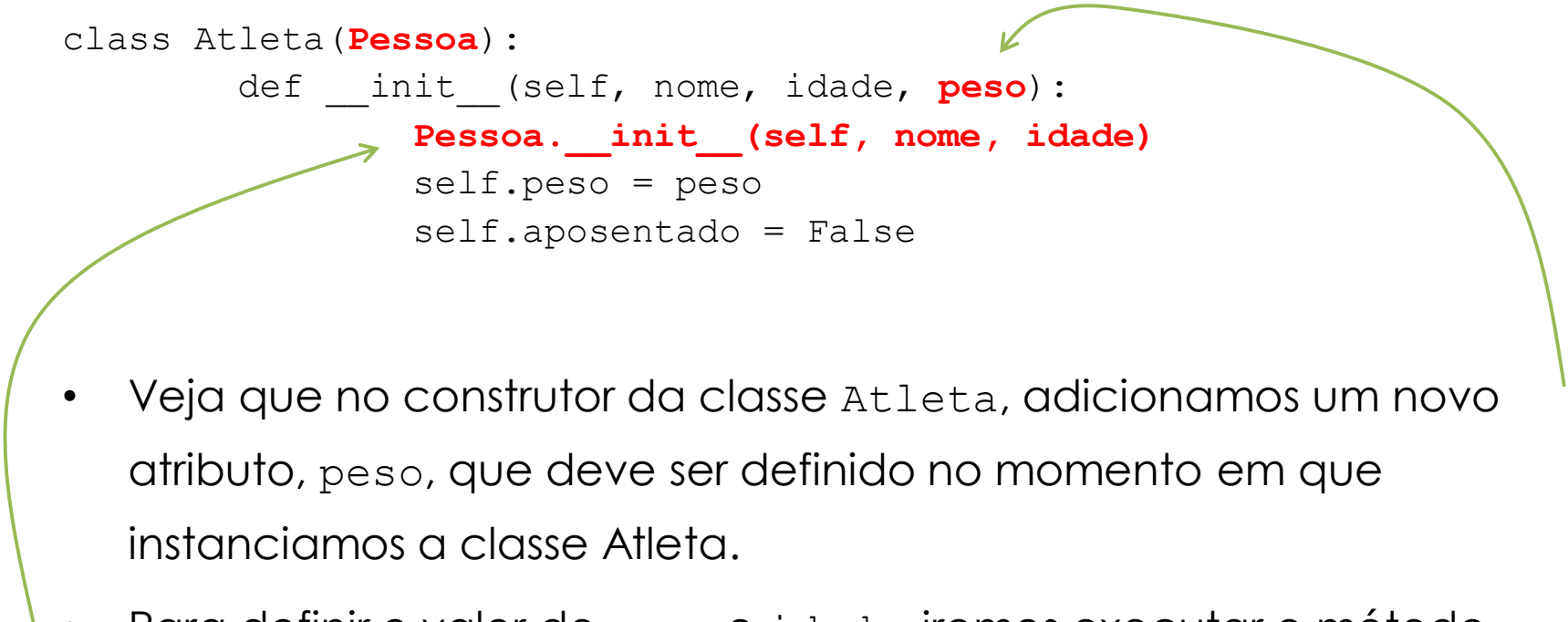
- Atributos: peso, **aposentado**
- Métodos: aquecer, aposentar



- Atributos: nome, idade

Herança

```
class Atleta(Pessoa):  
    def __init__(self, nome, idade, peso):  
        Pessoa.__init__(self, nome, idade)  
        self.peso = peso  
        self.aposentado = False
```



- Veja que no construtor da classe `Atleta`, adicionamos um novo atributo, `peso`, que deve ser definido no momento em que instanciamos a classe `Atleta`.
- Para definir o valor de `nome` e `idade`, iremos executar o método `__init__` da classe `Pessoa`.

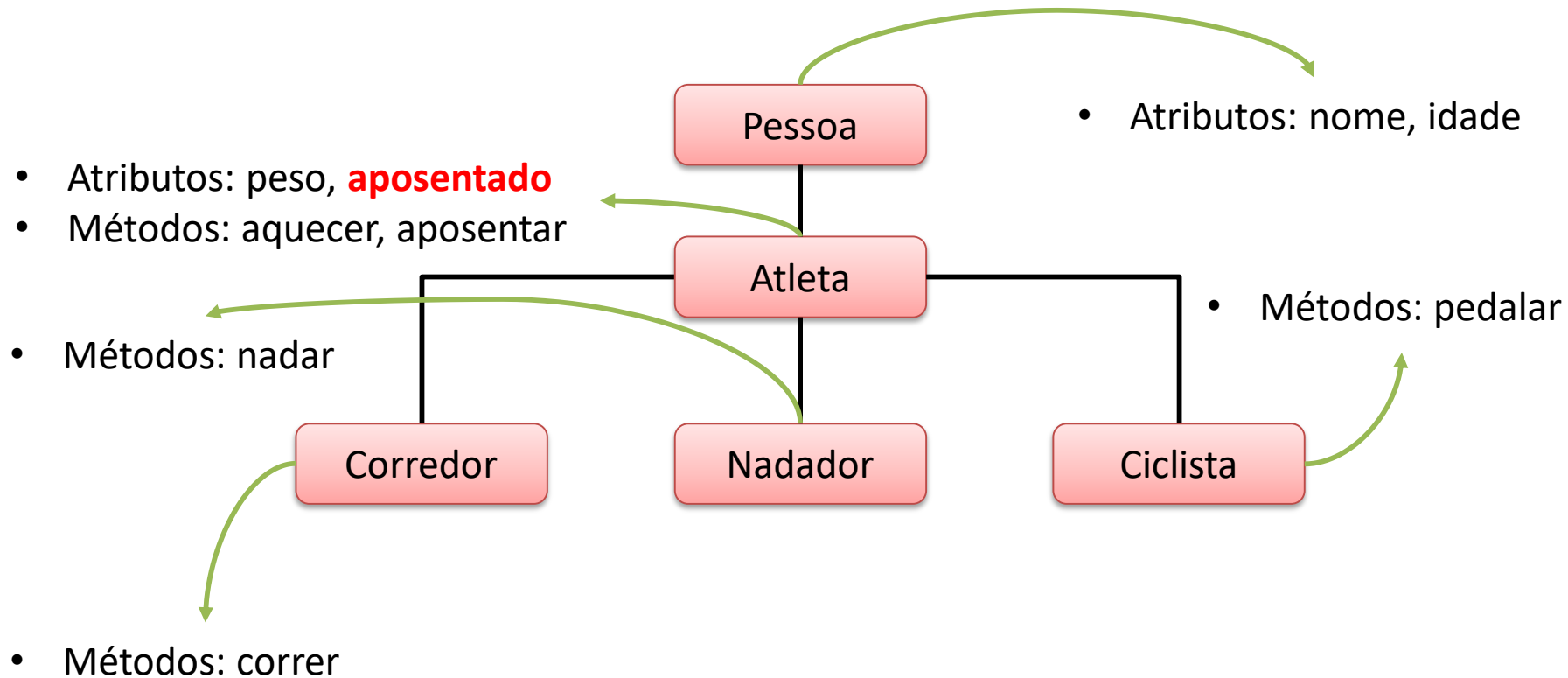
Herança

```
class Atleta(Pessoa):  
    def __init__(self, nome, idade, peso):  
        Pessoa.__init__(self, nome, idade)  
        self.peso = peso  
        self.aposentado = False  
  
    def aquecer(self):  
        print("Atleta Aquecido")  
  
    def aposentar(self):  
        self.aposentado = True
```



Abra o arquivo "aula2-parte2-classes.ipynb"

Herança



Herança

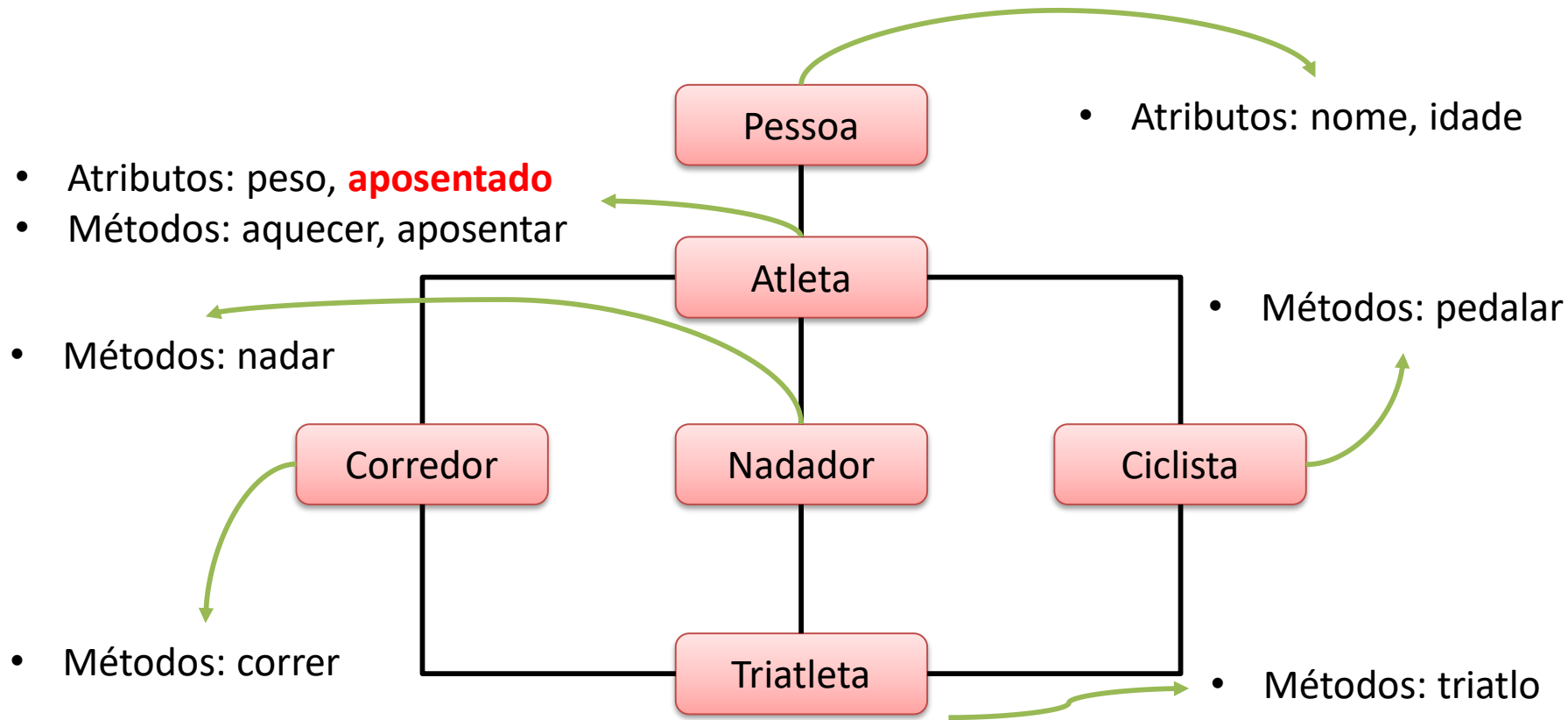
```
class Corredor(Atleta):  
    def correr(self):  
        print("Corredor correndo")
```

```
class Nadador(Atleta):  
    def nadar(self):  
        print("Nadador nadando")
```

```
class Ciclista(Atleta):  
    def pedalar(self):  
        print("Ciclista pedalando")
```

- As três classes Corredor, Nadador e Ciclista têm apenas um método cada e sem nenhum construtor.
- Porém todas herdam os atributos de **Atleta** que por sua vez herdam os atributos de **Pessoa**.

Herança Múltipla



Herança Múltipla

- É possível herdar atributos e métodos de múltiplas classes.
- Esse mecanismo é chamado de herança múltipla.
- **Por que quando instanciamos a classe Triatleta temos que definir os valores das variáveis nome, idade e peso?**

Herança Múltipla

```
class Triatleta(Corredor, Nadador, Ciclista):  
    def triatlo(self):  
        print("Iniciar prova")
```

 Abra o arquivo "**aula2-parte2-classes.ipynb**"

Exercício de 5 minutos

- Crie uma instancia da classe Pessoa com o nome de **joao**.
- Crie uma instancia da classe Atleta com o nome de **anderson**.
- Crie uma instancia da classe Corredor com o nome de **vanderlei**.
- Crie uma instancia da classe Triatleta com o nome de **silvia**.
- Análise o comportamento da criação e teste os métodos existentes em cada uma das instâncias criadas.

Conteúdo da Aula

- Objetivo
- Funções
- Classes e Objetos
- **Exercícios**

Exercícios

1) Escreva uma função chamada `carregar_arquivo`, que lê o conteúdo do arquivo `dados.txt`. Esse arquivo contém uma palavra em cada linha. Para cada linha lida, deve-se adicionar a palavra em uma lista. Ao final do código deve-se retornar a lista criada contendo todas as palavras. Lembre-se de remover o `\n`.

```
>>> def carregar_arquivo(nome_arquivo):
```



Abra o arquivo "[aula2-parte3-exercicios.ipynb](#)"

Exercícios

2) Crie uma função chamada **remover_repetidos**. Essa função deve receber uma lista como parâmetro. Deve-se remover todos as palavras repetidas. Utilize uma lista auxiliar para facilitar. Ao final retorne a lista.

```
>>> def remover_repetidos(dados):
```


Exercícios

3) Agora, crie uma função chamada **verificar_repetidos**. Essa função irá receber duas lista como parâmetro. Uma com todas as palavras lidas do arquivo e outra sem as palavras repetidas. Verifique a quantidade de vezes que as palavras aparecem. Ao final imprima a lista de palavras e a quantidade de vezes de cada palavra.

```
>>> def verificar_repetidos(dados, dados_unicos):
```

```
Saída:
```

```
Palavra1 - 10
```

```
Palavra2 - 1
```

Exercícios

4) Modifique a classe **Pessoa** vista em aula:

- a) Crie um método para calcular a idade em meses, chamado `calcular_meses`.
- b) Instancie a classe com os seguintes argumentos
 - Nome: 'João Silva'
 - Idade: 42
- c) Imprima a seguinte frase:
 - "Y tem X meses de vida", onde Y é o nome e X é o calculo da idade em meses.

Exercícios

5) Crie uma classe que modele um quadrado:

- Atributo: tamanho do lado (`tam_lado`)
 - Comportamentos:
 - Mudar valor do lado;
 - Retornar valor do lado; e
 - Calcular área;
- a) Instancie a classe com o valor de 4.
- b) Modifique o valor do lado para 8.
- c) Imprima a área calculada.

Referências Bibliográficas

- **Use a Cabeça! Python** – Paul Barry - Rio de Janeiro, RJ: Alta Books, 2012.
- **Use a Cabeça! Programação** – Paul Barry & David Griffiths – Rio de Janeiro RJ: Alta Books, 2010.
- **Aprendendo Python: Programação orientada a objetos** – Mark Lutz & David Ascher – Porto Alegre: Bookman, 2007

Referências Bibliográficas

- **Python for kids – A playful Introduction to programming** – Jason R. Briggs – San Francisco – CA: No Starch Press, 2013.
- **Python for Data Analysis** – Wes McKinney – USA: O'Reilly, 2013.
- **Python Cookbook** – David Beazley & Brian K. Jones – O'Reilly, 3th Edition, 2013.
- As referências de links utilizados podem ser visualizados em <http://urls.dinomagri.com/refs>