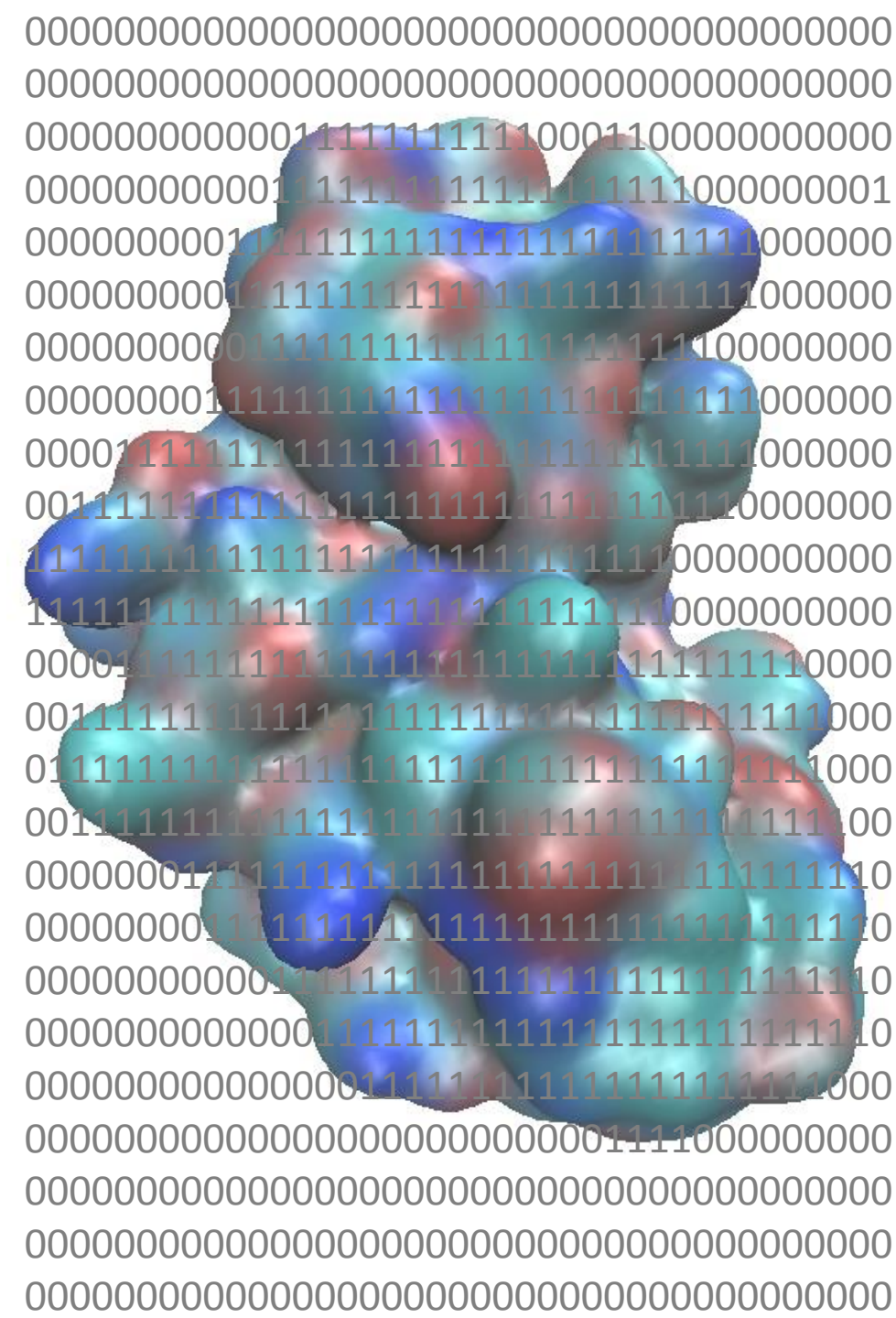
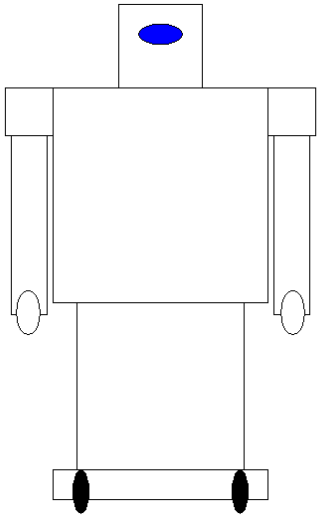


python™
www.python.org

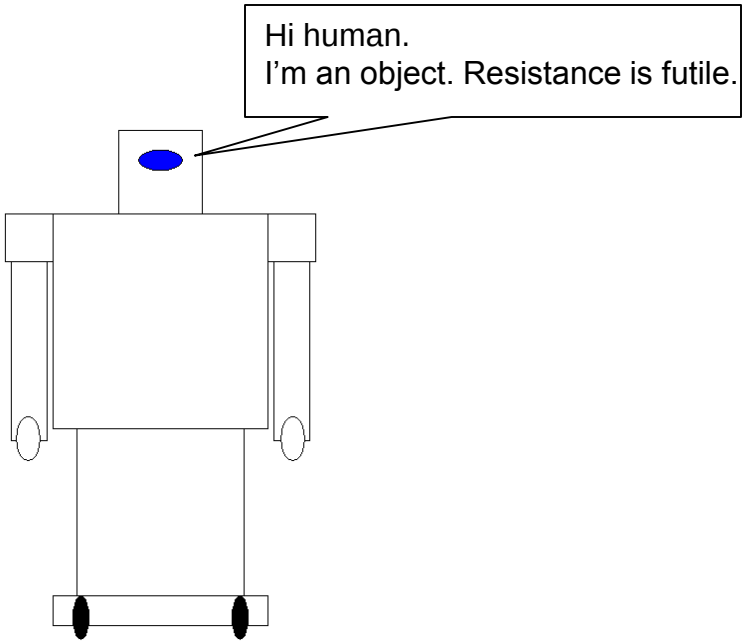
Bioinformática Aplicada

Aula 14

Hoje veremos o uso da abordagem de **programação orientada a objeto** (*object-oriented programming (OOP)*).
A aplicação da *OOP* baseia-se no conceito de **objeto**. O uso desta abordagem permite a representação de objetos reais como objetos do programa. Em *OOP* montamos **classes** que são usadas para criarmos objetos.



Objetos reais têm características que chamamos em *OOP* de **atributos**. Esses objetos reais podem apresentar comportamentos, que em *OOP* chamamos de **métodos**. Por exemplo, um programa para simular um robô pode ter capacidade memória, potência do motor de movimento, autonomia da bateria como atributos. Os métodos podem ser falar, andar, responder perguntas etc...
Vamos discutir os principais conceitos de OOP a partir do estudo de programas simples.



Abaixo temos o código do programa *robot1.py*. O programa inicia com a definição da **classe** *Robot*. Usamos o comando *class* seguido do nome da classe. Não é obrigatório, mas é comum usarmos nomes de classe que iniciam com letra maiúscula.
Após o nome da classe, colocamos a palavra reservada *object.* No presente caso, a classe *Robot* é baseada no objeto, um tipo fundamental interno de dado da linguagem Python. Uma classe pode ser baseada numa outra classe previamente definida.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
     """Simple class for a robot"""
     # Constructor method
     def __init__(self, name):
          print("\nA new robot has been built!")
          self.name = name
     # Method talk()
     def talk(self):
          print("\nHi. I'm an instance of class Robot.")

def main():
     # Instantiating an object of the Robot class and assigns it to the variable r2d2
     r2d2 = Robot("R2D2")
     # Invoking the method talk()
     r2d2.talk()
```

Em seguida definimos a *docstring* com uma breve descrição da classe. A *docstring* é montada de forma similar ao que foi visto para funções.
Agora temos a definição do método construtor. Este método é invocado toda vez que é criado um objeto da classe *Robot*. Veja que a definição do método construtor ocorre com o comando *def __init__()*. Usamos o comando *def* para definição de métodos de uma classe. O método construtor é muito útil, sendo frequente seu uso para cada classe. Um dos usos do método construtor é na atribuição de valores iniciais a atributos de um objeto. No programa *robot1.py* não usamos esse recurso. Aqui usamos o método construtor para mostrarmos uma mensagem na tela.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
     """Simple class for a robot"""
     # Constructor method
     def __init__(self, name):
          print("\nA new robot has been built!")
          self.name = name
     # Method talk()
     def talk(self):
          print("\nHi. I'm an instance of class Robot.")

def main():
     # Instantiating an object of the Robot class and assigns it to the variable r2d2
     r2d2 = Robot("R2D2")
     # Invoking the method talk()
     r2d2.talk()
```

Podemos ter os atributos de um objeto automaticamente criados e inicializados logo depois que sua criação. A classe *Robot* tem um método construtor que cria e inicializa o atributo *name.* O método construtor mostra a mensagem "*\nA new robot has been built!*". Em seguida é criado um novo atributo, *name,* para o novo objeto que usa o valor do parâmetro *name* como valor. Assim, a linha de código no programa principal, *r2d2 = Robot("R2D2"),* tem como resultado a criação de um novo objeto da classe *Robot*, com o atributo *name* definido como "R2D2". Por último, o objeto é atribuído à variável *r2d2*.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
     """Simple class for a robot"""
     # Constructor method
     def __init__(self, name):
          print("\nA new robot has been built!")
          self.name = name
     # Method talk()
     def talk(self):
          print("\nHi. I'm an instance of class Robot.")

def main():
     # Instantiating an object of the Robot class and assigns it to the variable r2d2
     r2d2 = Robot("R2D2")
     # Invoking the method talk()
     r2d2.talk()
```

A palavra reservada *self* tem como função receber a referência a um objeto invocando um método. Assim, por meio do *self*, um método pode acessar o objeto invocando ele e acessar os atributos e métodos do objeto.
No método construtor, o parâmetro *self* recebe uma referência ao novo objeto da classe *Robot,* enquanto o parâmetro *name* recebe a string *"R2D2"*.
A linha de código *self.name = name* cria o atributo *name* para o objeto e define o seu valor com o valor do parâmetro *name,* que é *"R2D2"*.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
     """Simple class for a robot"""
     # Constructor method
     def __init__(self, name):
          print("\nA new robot has been built!")
          self.name = name
     # Method talk()
     def talk(self):
          print("\nHi. I'm an instance of class Robot.")

def main():
     # Instantiating an object of the Robot class and assigns it to the variable r2d2
     r2d2 = Robot("R2D2")
     # Invoking the method talk()
     r2d2.talk()
```

O método *talk()* é definido usando-se o comando *def.* Veja que temos *self* como parâmetro do método, com a função já descrita. O método *talk()* é simplesmente uma função *print().* Veja que podemos pensar nos métodos como funções associadas a um objeto. Os métodos de uma dada classe têm que necessariamente ter o parâmetro *self*. Que no método *talk()* não é usado, mas tem que estar presente.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
     """Simple class for a robot"""
     # Constructor method
     def __init__(self, name):
          print("\nA new robot has been built!")
          self.name = name
     # Method talk()
     def talk(self):
          print("\nHi. I'm an instance of class Robot.")

def main():
     # Instantiating an object of the Robot class and assigns it to the variable r2d2
     r2d2 = Robot("R2D2")
     # Invoking the method talk()
     r2d2.talk()
```

No programa principal, a linha de código *r2d2 = Robot("R2D2")* cria um novo objeto da classe *Robot* e atribui este objeto à variável *r2d2.*
Você pode atribuir o novo objeto a qualquer variável, desde que siga as regras de variáveis do Python.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
     """Simple class for a robot"""
     # Constructor method
     def __init__(self, name):
          print("\nA new robot has been built!")
          self.name = name
     # Method talk()
     def talk(self):
          print("\nHi. I'm an instance of class Robot.")

def main():
     # Instantiating an object of the Robot class and assigns it to the variable r2d2
     r2d2 = Robot("R2D2")
     # Invoking the method talk()
     r2d2.talk()
```

Por último, o programa principal invoca o método *talk().* Usamos a notação *dot* (.) como vista para métodos de bibliotecas do Python.
A linha de código *r2d2.talk()* simplesmente invoca o método *talk()* de um objeto da classe *Robot*, que foi previamente atribuído à variável *r2d2.*

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
     """Simple class for a robot"""
     # Constructor method
     def __init__(self, name):
          print("\nA new robot has been built!")
          self.name = name
     # Method talk()
     def talk(self):
          print("\nHi. I'm an instance of class Robot.")

def main():
     # Instantiating an object of the Robot class and assigns it to the variable r2d2
     r2d2 = Robot("R2D2")
     # Invoking the method talk()
     r2d2.talk()
```

Abaixo temos o resultado de rodar o programa *robot1.py*.

```
A new robot has been built!


Hi. I'm an instance of class Robot.
```

No programa *robot2.py* modificamos o método *talk()* de forma que podemos mostrar o que foi atribuído ao parâmetro *name.* A modificação do método está em destaque em vermelho.

```
# Program to illustrate the basic concepts of class, objects, and methods

class Robot(object):
     """Simple class for a robot"""
     # Constructor method
     def __init__(self, name):
          print("\nA new robot has been built!")
          self.name = name
     # Method talk()
     def talk(self):
          print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
     # Instantiating an object of the Robot class and assigns it to the variable r2d2
     r2d2 = Robot("R2D2")
     # Instantiating an object of the Robot class and assigns it to the variable c3p0
     c3p0 = Robot("C3P0")
     # Invoking the method talk()
     r2d2.talk()
     # Invoking the method talk()
     c3p0.talk()

main()
```

Para criarmos múltiplos objetos da classe *Robot*, temos que ter uma linha de código para cada novo objeto a ser criado. Obviamente usamos como argumento strings distintas, como indicado no código abaixo.
Após a criação dos objetos, podemos invocar o método *talk()* para cada objeto criado.

```
# Program to illustrate the basic concepts of class, objects, and methods

class Robot(object):
     """Simple class for a robot"""
     # Constructor method
     def __init__(self, name):
          print("\nA new robot has been built!")
          self.name = name
     # Method talk()
     def talk(self):
          print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
     # Instantiating an object of the Robot class and assigns it to the variable r2d2
     r2d2 = Robot("R2D2")
     # Instantiating an object of the Robot class and assigns it to the variable c3p0
     c3p0 = Robot("C3P0")
     # Invoking the method talk()
     r2d2.talk()
     # Invoking the method talk()
     c3p0.talk()

main()
```

Abaixo temos o resultado de rodar o programa *robot2.py*.

```
A new robot has been built!

A new robot has been built!

Hi. I'm an instance of class Robot. My name is R2D2

Hi. I'm an instance of class Robot. My name is C3P0
```

Por meio do método *__str__()*, podemos criar uma string que será mostrada toda vez que usarmos a função *print()* para um objeto da classe. No exemplo em vermelho abaixo, temos a definição da string. Veja que usarmos o comando *return*.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
      """Simple class for a robot"""
      # Constructor method
      def __init__(self, name):
            print("\nA new robot has been built!")
            self.name = name
      def __str__(self):
            rep = "Robot object\n"
            rep += "name: " + self.name + "\n"
            return rep
      # Method talk()
      def talk(self):
            print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
      # Instantiating an object of the Robot class and assigns it to the variable r2d2
      r2d2 = Robot("R2D2")
      # Instantiating an object of the Robot class and assigns it to the variable c3p0
      c3p0 = Robot("C3P0")
      # Invoking the method talk()
      r2d2.talk()
      # Invoking the method talk()
      c3p0.talk()
      print("\nPrinting r2d2:")
      print(r2d2)
      print("\nDirectly accessing r2d2.name:")
      print(r2d2.name)
main()
```

No programa principal, a linha de código *print(r2d2)* garante que a string definida no método *__str__()* seja mostrada na tela.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
      """Simple class for a robot"""
      # Constructor method
      def __init__(self, name):
            print("\nA new robot has been built!")
            self.name = name
      def __str__(self):
            rep = "Robot object\n"
            rep += "name: " + self.name + "\n"
            return rep
      # Method talk()
      def talk(self):
            print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
      # Instantiating an object of the Robot class and assigns it to the variable r2d2
      r2d2 = Robot("R2D2")
      # Instantiating an object of the Robot class and assigns it to the variable c3p0
      c3p0 = Robot("C3P0")
      # Invoking the method talk()
      r2d2.talk()
      # Invoking the method talk()
      c3p0.talk()
      print("\nPrinting r2d2:")
      print(r2d2)
      print("\nDirectly accessing r2d2.name:")
      print(r2d2.name)
main()
```

Para acessarmos um atributo da classe *Robot*, fora da definição da classe, podemos usar a notação *dot*, como indicada na linha de código *print(r2d2.name)*, onde acessamos o valor atribuído ao *name*.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
      """Simple class for a robot"""
      # Constructor method
      def __init__(self, name):
            print("\nA new robot has been built!")
            self.name = name
      def __str__(self):
            rep = "Robot object\n"
            rep += "name: " + self.name + "\n"
            return rep
      # Method talk()
      def talk(self):
            print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
      # Instantiating an object of the Robot class and assigns it to the variable r2d2
      r2d2 = Robot("R2D2")
      # Instantiating an object of the Robot class and assigns it to the variable c3p0
      c3p0 = Robot("C3P0")
      # Invoking the method talk()
      r2d2.talk()
      # Invoking the method talk()
      c3p0.talk()
      print("\nPrinting r2d2:")
      print(r2d2)
      print("\nDirectly accessing r2d2.name:")
      print(r2d2.name)
main()
```

Abaixo temos o resultado de rodar o programa *robot3.py*.

```
A new robot has been built!
A new robot has been built!
Hi. I'm an instance of class Robot. My name is R2D2
Hi. I'm an instance of class Robot. My name is C3P0
Printing r2d2:
Robot object
name: R2D2
Directly accessing r2d2.name:
R2D2
```

O programa *robot4.py* define um atributo da classe *Robot* que conta o número de objetos da classe *Robot* que são criados.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
      """Simple class for a robot"""
      total = 0
      # Define a static method
      @staticmethod
      def status():
            print("\nThe total number of robots is", Robot.total)
      # Constructor method
      def __init__(self, name):
            print("\nA new robot has been built!")
            self.name = name
            Robot.total += 1
      def __str__(self):
            rep = "Robot object\n"
            rep += "name: " + self.name + "\n"
            return rep
      # Method talk()
      def talk(self):
            print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
      # Instantiating an object of the Robot class and assigns it to the variable r2d2
      r2d2 = Robot("R2D2")
      # Instantiating an object of the Robot class and assigns it to the variable c3p0
      c3p0 = Robot("C3P0")
      Robot.status()
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
main()
```

A linha de código *total = 0* cria um atributo *total* e atribui o valor 0 a ele. Variáveis definidas fora dos métodos são atributos da classe.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
      """Simple class for a robot"""
      total = 0
      # Define a static method
      @staticmethod
      def status():
            print("\nThe total number of robots is", Robot.total)
      # Constructor method
      def __init__(self, name):
            print("\nA new robot has been built!")
            self.name = name
            Robot.total += 1
      def __str__(self):
            rep = "Robot object\n"
            rep += "name: " + self.name + "\n"
            return rep
      # Method talk()
      def talk(self):
            print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
      # Instantiating an object of the Robot class and assigns it to the variable r2d2
      r2d2 = Robot("R2D2")
      # Instantiating an object of the Robot class and assigns it to the variable c3p0
      c3p0 = Robot("C3P0")
      Robot.status()
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
main()
```

Esta linha de código é executada uma vez, antes mesmo da criação do objeto. Os atributos de classes podem ser usados mesmo sem objetos.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
      """Simple class for a robot"""
      total = 0
      # Define a static method
      @staticmethod
      def status():
            print("\nThe total number of robots is", Robot.total)
      # Constructor method
      def __init__(self, name):
            print("\nA new robot has been built!")
            self.name = name
            Robot.total += 1
      def __str__(self):
            rep = "Robot object\n"
            rep += "name: " + self.name + "\n"
            return rep
      # Method talk()
      def talk(self):
            print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
      # Instantiating an object of the Robot class and assigns it to the variable r2d2
      r2d2 = Robot("R2D2")
      # Instantiating an object of the Robot class and assigns it to the variable c3p0
      c3p0 = Robot("C3P0")
      Robot.status()
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
main()
```

A linha de código *print(Robot.total)* acessa o valor do atributo *Robot.total,* antes da criação de um objeto da classe *Robot.*

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
      """Simple class for a robot"""
      total = 0
      # Define a static method
      @staticmethod
      def status():
            print("\nThe total number of robots is", Robot.total)
      # Constructor method
      def __init__(self, name):
            print("\nA new robot has been built!")
            self.name = name
            Robot.total += 1
      def __str__(self):
            rep = "Robot object\n"
            rep += "name: " + self.name + "\n"
            return rep
      # Method talk()
      def talk(self):
            print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
      # Instantiating an object of the Robot class and assigns it to the variable r2d2
      r2d2 = Robot("R2D2")
      # Instantiating an object of the Robot class and assigns it to the variable c3p0
      c3p0 = Robot("C3P0")
      Robot.status()
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
main()
```

A linha de código *print("\nThe total number of robots is", Robot.total)* acessa o valor do atributo *Robot.total*, no método estático *status()*

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
      """Simple class for a robot"""
      total = 0
      # Define a static method
      @staticmethod
      def status():
            print("\nThe total number of robots is", Robot.total)
      # Constructor method
      def __init__(self, name):
            print("\nA new robot has been built!")
            self.name = name
            Robot.total += 1
      def __str__(self):
            rep = "Robot object\n"
            rep += "name: " + self.name + "\n"
            return rep
      # Method talk()
      def talk(self):
            print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
      # Instantiating an object of the Robot class and assigns it to the variable r2d2
      r2d2 = Robot("R2D2")
      # Instantiating an object of the Robot class and assigns it to the variable c3p0
      c3p0 = Robot("C3P0")
      Robot.status()
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
main()
```

A linha de código *Robot.total* += *1* atualiza o valor do atributo *Robot.total*, no método construtor.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
      """Simple class for a robot"""
      total = 0
      # Define a static method
      @staticmethod
      def status():
            print("\nThe total number of robots is", Robot.total)
      # Constructor method
      def __init__(self, name):
            print("\nA new robot has been built!")
            self.name = name
            Robot.total += 1
      def __str__(self):
            rep = "Robot object\n"
            rep += "name: " + self.name + "\n"
            return rep
      # Method talk()
      def talk(self):
            print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
      # Instantiating an object of the Robot class and assigns it to the variable r2d2
      r2d2 = Robot("R2D2")
      # Instantiating an object of the Robot class and assigns it to the variable c3p0
      c3p0 = Robot("C3P0")
      Robot.status()
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
main()
```

Na definição do método estático *status()* usamos o decorador *@staticmethod* que indica que as linhas seguintes definem um método estático.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
      """Simple class for a robot"""
      total = 0
      # Define a static method
      @staticmethod
      def status():
            print("\nThe total number of robots is", Robot.total)
      # Constructor method
      def __init__(self, name):
            print("\nA new robot has been built!")
            self.name = name
            Robot.total += 1
      def __str__(self):
            rep = "Robot object\n"
            rep += "name: " + self.name + "\n"
            return rep
      # Method talk()
      def talk(self):
            print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
      # Instantiating an object of the Robot class and assigns it to the variable r2d2
      r2d2 = Robot("R2D2")
      # Instantiating an object of the Robot class and assigns it to the variable c3p0
      c3p0 = Robot("C3P0")
      Robot.status()
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
main()
```

O método estático *status()* mostra na tela o valor atualizado do atributo de classe *Robot.total.* O método estático não pode modificar o estado do objeto *(self.name*).

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
      """Simple class for a robot"""
      total = 0
      # Define a static method
      @staticmethod
      def status():
            print("\nThe total number of robots is", Robot.total)
      # Constructor method
      def __init__(self, name):
            print("\nA new robot has been built!")
            self.name = name
            Robot.total += 1
      def __str__(self):
            rep = "Robot object\n"
            rep += "name: " + self.name + "\n"
            return rep
      # Method talk()
      def talk(self):
            print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
      # Instantiating an object of the Robot class and assigns it to the variable r2d2
      r2d2 = Robot("R2D2")
      # Instantiating an object of the Robot class and assigns it to the variable c3p0
      c3p0 = Robot("C3P0")
      Robot.status()
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
main()
```

No programa principal, a linha de código *Robot.status()* invoca o método estático *status()*, que mostra o valor atualizado do atributo *Robot.status().*

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
      """Simple class for a robot"""
      total = 0
      # Define a static method
      @staticmethod
      def status():
            print("\nThe total number of robots is", Robot.total)
      # Constructor method
      def __init__(self, name):
            print("\nA new robot has been built!")
            self.name = name
            Robot.total += 1
      def __str__(self):
            rep = "Robot object\n"
            rep += "name: " + self.name + "\n"
            return rep
      # Method talk()
      def talk(self):
            print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
      # Instantiating an object of the Robot class and assigns it to the variable r2d2
      r2d2 = Robot("R2D2")
      # Instantiating an object of the Robot class and assigns it to the variable c3p0
      c3p0 = Robot("C3P0")
      Robot.status()
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
main()
```

Na última linha de código do programa principal, acessamos o valor do atributo *Robot.total,* após a criação de dois objetos.

```
# Program to illustrate the basic concepts of class, objects, and methods
class Robot(object):
      """Simple class for a robot"""
      total = 0
      # Define a static method
      @staticmethod
      def status():
            print("\nThe total number of robots is", Robot.total)
      # Constructor method
      def __init__(self, name):
            print("\nA new robot has been built!")
            self.name = name
            Robot.total += 1
      def __str__(self):
            rep = "Robot object\n"
            rep += "name: " + self.name + "\n"
            return rep
      # Method talk()
      def talk(self):
            print("\nHi. I'm an instance of class Robot. My name is ",self.name)
def main():
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
      # Instantiating an object of the Robot class and assigns it to the variable r2d2
      r2d2 = Robot("R2D2")
      # Instantiating an object of the Robot class and assigns it to the variable c3p0
      c3p0 = Robot("C3P0")
      Robot.status()
      print("\nAccessing the class attribute Robot.total:", end=" ")
      print(Robot.total)
main()
```

Abaixo temos o resultado de rodar o programa *robot4.py*.

```
Accessing the class attribute Robot.total: 0

A new robot has been built!

A new robot has been built!

The total number of robots is 2

Accessing the class attribute Robot.total: 2
```

Pegue os programas *fibonacci.py* e *scatter_plot6.py* e use o paradigma de programação orientada a objeto para gerar novos códigos. Os novos programas serão chamados.:

fibonacci2.py
scatter_plot7.py

-BRESSERT, Eli. **SciPy and NumPy**. Sebastopol: O'Reilly Media, Inc., 2013. 57 p. Link

-DAWSON, Michael. **Python Programming, for the Absolute Beginner.** 3ed. Boston: Course Technology, 2010. 455 p.

-HACKELING G. **Mastering Machine Learning with scikit-learn**. Birmingham: Packt Publishing Ltd., 2014. 221 p. Link

-HETLAND, Magnus Lie. **Python Algorithms. Mastering Basic Algorithms in the Python Language.** 2ed. Nova York: Springer Science+Business Media LLC, 2010. 294 p. Link

-IDRIS, Ivan. **NumPy 1.5. An action-packed guide dor the easy-to-use, high performance, Python based free open source NumPy mathematical library using real-world examples. Beginner's Guide.** Birmingham: Packt Publishing Ltd., 2011. 212 p. Link

-LUTZ, Mark. **Programming Python.** 4ed. Sebastopol: O'Reilly Media, Inc., 2010. 1584 p. Link

-TOSI, Sandro. **Matplotlib for Python Developers.** Birmingham: Packt Publishing Ltd., 2009. 293 p. Link

Última atualização: 08 de julho de 2019.