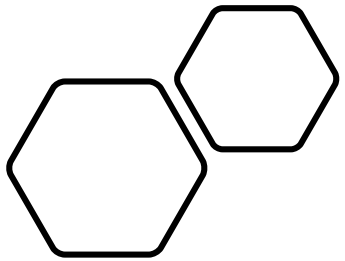
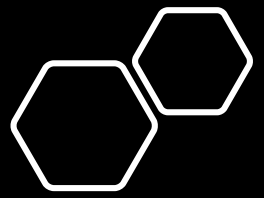
Python para Ciência de Dados e Inteligência Artificial



Aula 03: Função Lambda, Classes e Introdução ao Numpy

Prof. Anderson Harayashiki Moreira



Funções Lambda

Funções sem nome?

Funções Lambda

- **Lambda:** Funções anônimas, normalmente são simples e definidas em uma única linha. As funções lambda são usualmente utilizadas em conjunto com os métodos filter, reduce e map.

```
lambda argumento_1, ..., argumento_n: expressão
```

Funções Lambda

- **Exemplo 1:** Algo simples como x + 1

```
soma1 = lambda x: x + 1

print(soma1(1))
```

```
def soma_1(x):
    return x + 1

print(soma_1(1))
```

- **Exemplo 2:** Algo simples como x + y

```
somador = lambda x,y: x + y

print(somador(1,2))
```

```
def somador_f(x,y):
    return x + y

print(somador_f(1,2))
```

Funções Lambda

- **Exemplo 3:** Algo mais útil para nosso curso

```
lista = [(1,2), (-2,4), (3,-1), (5,4), (-10,1), (18,9)] # Lista de Tuplas
lista_ordenada = sorted(lista)

print(lista)
print(lista_ordenada)

lista_ordenada_2 = sorted(lista, key = lambda x: x[1])
print(lista_ordenada_2)

lista_ordenada_3 = sorted(lista, key = lambda x: sum(x)) # ou x[0] + x[1]
print(lista_ordenada_3)
```

Funções Lambda

- **Exemplo 4:** Vamos criar uma parábola?

```
dominio = list(range(-10,10))
imagem = map(lambda x: x**2 + 2*x + 1, dominio)
print(list(imagem))
```

- **Exemplo 5:** Vamos fazer uma filtragem?

```
lista = [1, 2, 3, 4, 5]
lista_filtrada = list(filter(lambda x: x > 2, lista))
print(lista)
print(lista_filtrada)
```

Funções Lambda

- **Exemplo 6:** Fatorial?

```
from functools import reduce

valores = [1, 2, 3, 4, 5, 6, 7]
produto = reduce (lambda x, y: x*y, valores)

print(produto)
```

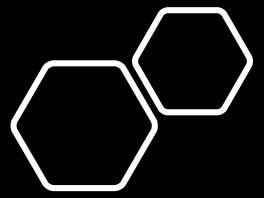
Funções Lambda

- **Exemplo 7:** Spoiler

```
import pandas as pd
import numpy as np

dados = {'City': ['São Paulo', 'Santo André', 'São Caetano do Sul'],
         'Population': [12252023, 718773, 160275],
         'Area': [1521.11, 174.840, 15.331]}

df = pd.DataFrame(data=dados)
df['Density'] = df.apply(lambda x: x['Population']/x['Area'], axis=1)
print(df)
```

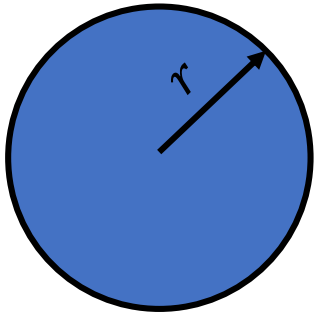


Classes

Introdução a Orientação a Objetos

Criando uma classe: Atributos e Método Construtor

- **Atributos:** Todas as características que determinam uma classe
- **Método construtor:** Responsável por inicializar uma classe



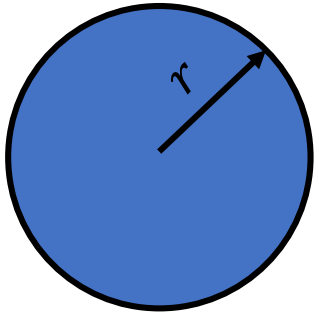
Classe:
- Círculo

Atributos:
- Cor
- Raio

```
class Circulo(object):
    # Construtor
    def __init__(self, raio = 3, cor = 'azul'):
        self.raio = raio
        self.cor = cor
```

Criando uma classe: Métodos

- **Métodos:** Ações realizadas por uma classe



Classe:
- Círculo

Atributos:
- Cor
- Raio

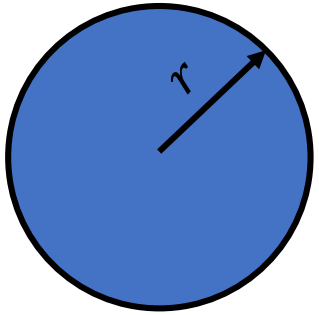
Métodos:
- incrementaRaio
- desenhaCirculo

```
import matplotlib.pyplot as plt

class Circulo(object):
    # Método Construtor
    def __init__(self, raio = 3, cor = 'blue'):
        self.raio = raio
        self.cor = cor
    def incrementaRaio(self, inc):
        """Incrementa o valor de inc no atributo raio"""
        self.raio += inc
        return self.raio
    def desenhaCirculo(self):
        plt.gca().add_patch(plt.Circle((0, 0), radius=self
.raio, fc=self.cor))
        plt.axis('scaled')
        plt.show()
```

Criando uma classe: Instâncias

- **Instâncias:** É uma representação da classe



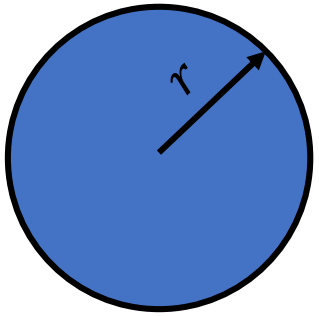
Classe:
- Círculo

Atributos:
- Cor
- Raio

Métodos:
- incrementaRaio
- desenhaCirculo

```
vermelho = Circulo(3, "red")
vermelho.desenhaCirculo()
```

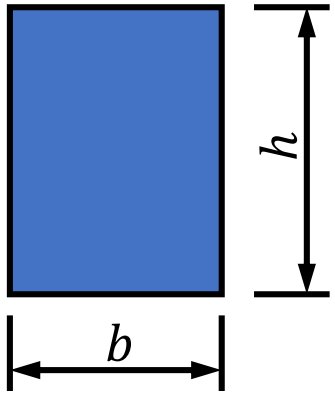
Criando uma classe: Exercício



Classe:
- Círculo

Atributos:
- Cor
- Raio

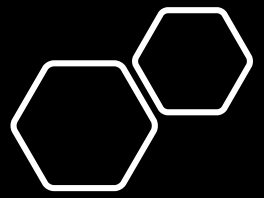
Métodos:
- incrementaRaio
- desenhaCirculo



Classe:
- Retângulo

Atributos:
- Cor
- Base
- Altura
- Ângulo

Métodos:
- Método construtor
- incrementaBase
- incrementaAltura
- incrementaAngulo
- desenhaRetangulo



Numpy

Introdução ao Numpy

Exemplo: Problema Motivacional

- Considere a lista `L = [1, 2, 3, 4, 5, 6, 7]`, deseja-se multiplicar cada item da lista por `7`, escreva o código em Python para realizar tal tarefa:

```
L = [1, 2, 3, 4, 5, 6, 7]
```

Numpy: Criando Arrays (1D)

- Criando Arrays (1D):

```
import numpy as np

a = np.array([2, 3, 4])
print(a)
print(a.dtype)

b = np.array([1.2, 3.5, 5.1])
print(b)
print(b.dtype)

c = np.array([1, 2, 3, 4], dtype=complex )
print(c)
print(c.dtype)
```

Numpy: Criando Arrays (2D)

- Criando Arrays (2D):

```
import numpy as np

a = np.array([[11, 12, 13], [21, 22, 23], [31, 32, 33]])
print(a)
print(a.dtype)

b = np.array([[1.2], [3.5], [5.1]])
print(b)
print(b.dtype)

c = np.array([[1,2],[3,4]], dtype=complex)
print(c)
print(c.dtype)
```

Numpy: Criando Arrays

- Criando Arrays pré-definidos:

```
import numpy as np

z = np.zeros((3, 4))
print(z)

o = np.ones((2,3,4), dtype=np.int16) # Com definição do tipo de dado
print(o)

i = np.identity(3)
print("\ni = \n", i)

v = np.empty((2,3))
print(v)
```

Numpy: Criando Arrays

- Criando Arrays pré-definidos:

```
import numpy as np

x1 = np.arange( 10, 30, 5 )
print("\nx1 = \n", x1)

x2 = np.arange( 0, 2, 0.3 ) # Também funciona para incremento real (float)
print("\nx2 = \n", x2)

x3 = np.linspace(0, 2, 9) # 9 elementos de 0 a 2
print("\nx3 = \n", x3)

x4 = np.linspace(0, 2*np.pi, 100)
print("\nx4 = \n", x4)
```

Numpy: Criando Arrays

- Criando Arrays pré-definidos:

```
import numpy as np

r1 = np.random.rand(2, 3)
print("\nr1 = \n", r1)

r2 = np.random.randint(1, 100, r1.shape)
print("\nr2 = \n", r2)
```

Numpy: Acesso a elementos em um Array

- Acessando elementos em um Array:

```
import numpy as np

v = np.array([1, 2, 3, 4, 5])
print(v)

M = np.array([[11, 12, 13], [21, 22, 23], [31, 32, 33]])
print(M)

print(v[0])
print(v[-1])
print(M[0]) # será que era isso que você queria?
print(M[0,0])
print(M[2][1])
print(M[-1][-2])
```

Numpy: Criando cópias

- Cuidado ao criar cópias de Arrays:

```
import numpy as np

A = np.arange(2,10).reshape(2,4)
print("A = \n", A)

B = A
B[1,2] = 0
print("\nA = \n", A)
print("\nB = \n", B)
print("\nA is B =", A is B)

C = A.copy()
C[1,2] = 2
print("\nA = \n", A)
print("\nC = \n", C)
print("\nA is C =", A is C)
```

Numpy: Manipulando o formato de Arrays

- Manipulando o formato de Arrays:

```
import numpy as np

a = np.arange(15)
print("a =\n", a)
print("\nshape =", a.shape)
print("ndim =", a.ndim)
print("itemsize =", a.itemsize)
print("size =", a.size)
a = a.reshape(3, 5)
print("\na =\n", a)
print("\nshape =", a.shape)
print("ndim =", a.ndim)
print("itemsize =", a.itemsize)
print("size =", a.size)

print("\na.T =\n", a.T)

a = a.ravel()
print("\na =\n", a)
```

Numpy: Funções e Métodos

- Funções e métodos mais comuns:

```
import numpy as np

A = np.array([[2, 3, 5], [1, 9, 7]])

print("len(A) = ", len(A)) # Sempre retorna o número de linhas
print("len(A[0, :]) = ", len(A[0, :])) # Retorna o número de colunas
print("A.max() = ", A.max())
print("A.min() = ", A.min())
print("A.sum() = ", A.sum())
print("A.sum(axis = 0) = ", A.sum(axis = 0))
print("A.sum(axis = 1) = ", A.sum(axis = 1))
print("A.prod() = ", A.prod())
print("np.where(A > 4) = ", np.where(A > 4, A, 0)) # Como isso funciona?
```

Numpy: Operações Básicas

- Operações básicas (Broadcasting):

```
import numpy as np

a = np.array([20,30,40,50])
b = np.arange(1,5)

print("a + b =", a + b)
print("a - b =", a - b)
print("a * b =", a * b) # Cuidado não é o prooduto de duas Matrizes
print("a / b =", a / b)
print("a @ b =", a @ b)
print("a.dot(b) =", a.dot(b)) # Produto escalar
print("a > b =", a > b)
```

Numpy: Operações Básicas

- No produto de duas matrizes cuidado com a dimensão:

```
import numpy as np

# Cuidado com a dimensão dos arrays na multiplicação de matrizes
A = np.array([[2, 3, 7], [1, 7, 5]])
B = np.array([[1, 5], [7, 8], [9, 8]])

print("\nA @ B =\n", A @ B)
print("\nA @ B.T =\n", A @ B.T) # B.T é a matriz transposta de B
```

Numpy: Indexação e Fatiamento

- Indexação para Arrays com uma dimensão:

```
import numpy as np

a = np.arange(10)**3

print("a =", a)
print("\na[2] =", a[2])
print("a[2:5] =", a[2:5])

a[:6:2] = 1000 # O que isso faz?
print("a =", a)
print("a[: : -1] =", a[ : :-1])

for i in a: # o Array continua sendo iterativo
    print(i**(1/3))
```

Numpy: Indexação e Fatiamento

- Indexação para Arrays com multidimensão:

```
import numpy as np

def f(x,y):
    return 10*x+y

b = np.fromfunction(f,(5,4),dtype=int)
print("b =\n", b)
print("\nb[2,3] =", b[2,3])
print("\nb[0:5, 1] =", b[0:5, 1])
print("\nb[:,1] =", b[:,1]) # Qual a diferença?
print("\nb[1:3,:] =\n", b[1:3,:])
```

Numpy: Estatística

- Análise exploratória simples:

```
import numpy as np

A = np.random.rand(10)

print(A)
print(np.mean(A))
print(np.std(A))
print(np.var(A))
```