Aula 3

Mudando paradigmas

* Transpondo conceitos da vida real em código
* Programação Orientada a Objetos POO

Um problema para ser resolvido pela Programação Orientada a Objetos

PROGRAMAÇÃO ESTRUTURADA

* Método até agora: programação estruturada
* Receita de bolo
* Funções
* Estruturas de seleção
* Estrutura de repetição
* Entrada e saída de dados

PROGRAMAÇÃO ORIENTADA A OBJETOS

* Segurança
* Confiança
* Sistemas corporativos
* Sistemas corporativos
* Internet Banking
* E-commerce
* Sistemas contábeis
* Sistemas corporativos (ERPs)

NOÇÕES DE PROGRAMAÇÃO ORIENTADA A OBJETOS

POO

* Unified Modeling Language UML
* Programação Orientada a Objetos POO
* Structured Query Language SQL

OBJETOS

* “Coisas” da vida real
* Dois objetos não ocupam o mesmo lugar ao mesmo tempo
* Estado
* Características

ESTADOS

* Variáveis: atributos
* Comportamentos

>> Ações: funções (métodos)

CLASSES

* “Manual” (blueprint) para se criarem objetos
* Receita de bolo, não bolo
* Planta da casa, não a casa
* Instância de classe: objeto

Exemplo

* Classe “Cachorro”
* Métodos

>> Latir

>>Comer

>>Dormir

* Atributos

>>Cor

>>Nome

>>Brinquedo Favorito

* Objeto “Kora”: instância de “Cachorro”
* Objeto “Pipoca”: instância de “Cachorro”
* Objeto “Dexter”: instância de “Cachorro”

HERANÇAS

* Heranças de classes  
  >>Classes especializadas

>>Herdam métodos e atributos

* Carro

>>Caminhonete

>>Furgão

>>Minivam

* Bicicleta

>>BMX

>>Bicicleta de Carga

>>Bicicleta de Pista

* Residência
* Sobrado
* Casa de Praia
* Apartamento
* Cliente de Internet Banking
* Conta simples
* Conta premium
* Conta empresarial
* Classe “Cachorro”
* Métodos

Latir

Comer

Dormir

* Atributos
* Cor
* Nome
* Brinquedo favorito
* Classe “Cão Farejador”
* Métodos

Farejar

Sentar

* Atributos

Tempo de Treinamento

Aeroporto

Raça

INTERFACES

* Classes podem usar interfaces
* Interfaces listam os métodos que uma classe que a usa deve ter
* Interfaces não implementam métodos

CLASSES E OBJETOS EM PYTHON

CRIANDO UMA CLASSE

* Class [nome da classe]
* Atributos
* Métodos
* Self

Exemplo

class Bolo:

    peso = 0

    def atribuir\_peso(self, peso\_a\_atribuir):

        self.peso = peso\_a\_atribuir

bolo\_aniversario = Bolo()

print(bolo\_aniversario.peso)

bolo\_aniversario.atribuir\_peso (2)

print(bolo\_aniversario.peso)

INICIALIZANDO UMA CLASSE

* def\_\_init\_\_

class Bolo:

    peso = 0

    def\_\_init\_\_ (self, peso\_a\_atribuir):

        self.peso = peso\_a\_atribuir

    def atribuir\_peso(self, peso\_a\_atribuir):

        self.peso = peso\_a\_atribuir

bolo\_aniversario = Bolo()

print(bolo\_aniversario.peso)

bolo\_aniversario.atribuir\_peso (2)

print(bolo\_aniversario.peso)

POLIMORFISMO

* Implementações diferentes do mesmo método
* Parâmetros diferentes
* Parâmetros diferentes
* Resultados diferentes
* Veículo: estacionar
* Carro: estacionar
* Moto: estacionar
* Bicicleta: estacionar

CLOSURES E DECORATORS

ENCAPSULAMENTO

* Funções dentro de funções
* Reduções de casos de variáveis globais
* Encapsulamento e isolamento da lógica

CLOSURES

* Funções que:
* Ficam dentro de outras funções e Usam variáveis atribuídas a elas

CLOSURES

def calcular\_volume\_area\_base(area\_base):

    constante = 3

    def calcular\_volume\_piramide(altura):

        return (area\_base \* altura)/constante

    return calcular\_volume\_piramide

def calcular\_volume\_area\_base(area\_base):

    constante = 3

    def calcular\_volume\_piramide(altura):

        return (area\_base \* altura)/constante

    return calcular\_volume\_piramide

 # closure

 piramide = calcular\_volume\_area\_base(15) # area base

 piramide (2) # altura

DECORATOR

* Closures que podem ser inseridas no código sem alterações estruturais
* Uso do “@”
* @staticmethod
* @classmethod

class Bolo:

    peso = 0

    def\_\_init\_\_(self, peso\_inicial=0):

        self.peso = peso\_inicial

    def atribuir\_peso(self, peso\_a\_atribuir):

        self.peso = peso\_a\_atribuir

    @staticmethod

    def calcular\_porcoes\_por\_ovo(numero\_ovos):

        return round(numero\_ovos \* 3)