Ada Tech [DS-PY-002] Programação Orientada a Objetos (PY)

Apresentação

Rudiney Casali:

- Doutorado em Física (UFSC)
- Especialista em Dados (Mackenzie)
- Lead Senior Data Scientist (Kyndryl)
- linkedin.com/in/rcasali/

Requerimentos

- Todos aqui estão familiarizados com a ferramenta Jupyter Notebook?

- Você pode editar seu JN no browser, e usando anaconda. Usaremos o editor de texto Visual Studio Code (NNF).

- Os encontros com a língua inglesa serão inevitáveis daqui pra frente…

Conteúdos

- Aula 1 (Expositiva e Prática): Introdução, Paradigmas de Programação, Programação Orientada a Objetos (POO);

- Aulas 2 e 3 (Expositiva e Prática): POO -
  - Classes e Objetos em Python,
  - Modulos e Propriedades,
  - Atributos Privados e de classe,
  - Métodos Mágicos,
  - Herança,
  - Polimorfismo;

- Aulas 4 e 5 (Expositiva e Prática): Desenvolvimento de Classes em POO;

- Aulas 6, 7 e 8 (Prática): Prática de POO em grupo;

  Aula 9 (Prática): Apresentação do Trabalho em POO e Rubrica.

Aula 01 | Paradigmas de Programação

O problema da programação

Imagine que você tem uma tarefa repetitiva e metódica que precisa realizar, que envolve a leitura de dados, o renomeio de arquivos e tabelas, o cálculo de valores, o rearranjo dos números e outras operações… Para economizar tempo você resolve automatizar o processo por meio de uma sequência de procedimentos pré-determinadas. Mas como organizar a automatização, que recursos aplicar para a construção do algoritmo? Que paradigmas adotar para programar a seqüência de procedimentos?

O Paradigma de Programação

O advento da computação aperfeiçoou nossa capacidade de realizar processos longos, repetitivos e tediosos.

Uma língua especial binária (0,1) é utilizada para a comunicação com a máquina. Simplificada para computadores, não amigável para humanos.

Linguagens mais ricas em recursos léxicos, sintáticos e semânticos foram propostas. Diferentes paradigmas foram desenvolvidos.

Paradigmas de Programação

Um paradigma é definido como um conjunto das formas que servem de modelo de derivação ou de flexão.

Diferentes maneiras ou estilos em que um determinado programa ou linguagem de programação pode ser organizado.

Paradigmas de Programação

Cada paradigma consiste em certas estruturas, características e opiniões sobre como problemas comuns de programação devem ser resolvidos.

Cada paradigma é mais apropriado a uma determinada situação, de modo que sua aplicação é mais recomendada que outra para um determinado tipo de problema, como também ocorre com as diferentes linguagens de programação.

Paradigmas de Programação

Em programação, as linguagens se subdividem em paradigmas, de acordo com suas características (estrutura, execução…).

Atualmente há dois principais paradigmas de programação, o imperativo e o declarativo. Cada um se subdivide em diferentes tipos.

Paradigmas de Programação

Devido aos avanços em *software* e *hardware*, os paradigmas de programação sofreram, por sua vez, modificações e evoluções, ao longo do tempo.

O desenvolvimento das ferramentas criadas e sua adaptação a problemas específicos foram grandes inspirações para a criação de diferentes paradigmas.

Paradigmas de Programação

Um paradigma não é uma linguagem ou uma ferramenta por si, ele é um conjunto de ideais e diretrizes com os quais um grupo de pessoas concorda, segue e expande.

Há linguagens de programação que tiram melhor proveito de um tipo de paradigma (Haskell) e há linguagens "multi-paradigma", em que é possível adaptar o código para aceitar um ou mais paradigmas (JavaScript e Python).

Paradigmas de Programação

Paradigmas de programação não são mutuamente excludentes, é possível aplicar práticas de diferentes (multi)paradigmas no mesmo código.

Conhecer diferentes paradigmas e o jargão da área pode ajudar a resolver problemas cotidianos e a transitar no mundo da programação.

Paradigmas de Programação Populares

Alguns dos mais populares paradigmas são:

- Paradigma Declarativo;
  - Programação Funcional;
  - Programação Lógica
- Paradigma Imperativo;
  - Programação Estruturada/Modular;
  - Programação Procedural;
  - Requisição a bancos de dados;
  - Programação Orientada a Objetos.

Paradigma Declarativo

Na programação declarativa não há instruções sobre como executar uma tarefa, o código apenas declara qual é o resultado requisitado. Suponha que devemos filtrar uma lista de números:

```
cutoff = 5
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
results = []
print('Before:')
print('\t numbers:', numbers)
print('\t results:', results)
results = list(filter(lambda number: number > cutoff, numbers))
print('After:')
print('\t numbers:', numbers)
print('\t results:', results)
```

Paradigma Declarativo

Não dizemos ao programa para iterar cada elemento do arranjo, comparar o valor do item com cutoff = 5 e, se o item for maior que 5, colocá-lo em um novo arranjo. Apenas aplicamos a função de filtragem *filter()*.

A iteração ainda ocorre, mas implicitamente, e não de forma explícita.

Paradigma Declarativo : Programação Funcional

A programação funcional tenta resolver problemas compondo funções matemáticas como componentes do programa.

O código é criado junto com a aplicação e compondo as funções matemáticas. O programa escrito sempre produz uma saída exatamente similar a sua função matemática equivalente:

```
f(x) = x + 1
x = 1 : f(1) = 2
x = 2 : f(1) = 3
```

Paradigma Declarativo : Programação Funcional

A imutabilidade é uma das características mais importantes da programação funcional.

Códigos escritos sob o paradigma funcional podem ser executados em ambientes multi-núcleos (multi-core) e multi-lineares (multi-threaded).

Paradigma Declarativo : Programação Funcional

As funções são tratadas como objetos de primeira classe, em que uma função pode ser atribuída a uma variável, ser passada como argumento para outra função e também ser retornada de outra função, como um valor.

As funções puras e impuras são outro conceito importante da programação funcional. No **primeiro caso**, a função sempre retorna a mesma saída quando recebe a mesma entrada e não tem nenhum efeito colateral, não causando nenhuma modificação em um estado não-local. O inverso do que ocorre no **segundo caso**, quando os valores retornados podem variar para a mesma entrada e efeitos colaterais são gerados.

Paradigma Declarativo : Programação Funcional

A modularização e a ausência dos efeitos colaterais tornam o código mais fácil de ser lido e mantido. Retomando o exemplo do filtro para números:

```
def Filter_Numbers(cutoff, numbers):

    results = []
    print('Before:')
    print('\t numbers:', numbers)
    print('\t results:', results)

    results = list(filter(lambda number: number > cutoff, numbers))

    print('After:')
    print('\t numbers:', numbers)
    print('\t results:', results)
    return results
```

Paradigma Declarativo : Programação Lógica

Baseado em **lógica** (fatos e regras) e **controle** (ordem das regras)

A **lógica** define o que deve ser resolvido, enquanto o **controle** define como deve ser resolvido.

Paradigma Declarativo : Programação Lógica

Declarações em forma lógica, cada uma contendo fatos, regras, ordem das regras.

Modelo de computação abstrata (Abstract model of computation)

Baseado na lógica de predicados de primeira ordem (Horn clause)

Paradigma Declarativo : Programação Lógica

Um programa lógico é um conjunto de declarações.

```
def test_1(list_values, cutoff_value):
    if list_values[0] > cutoff_value:
        print(f"O primeiro número é maior que  {cutoff_value}")
    if list_values[0] > cutoff_value  and list_values[1] > cutoff_value:
        print(f"Os dois primeiros números são maiores que  {cutoff_value}")
    if list_values[0] > cutoff_value  or list_values[1] > cutoff_value:
        print(f"Pelo menos Um dos dois primeiros números é maior que  {cutoff_value}")
    if list_values[0] > cutoff_value  and list_values[1] > cutoff_value  and list_values[2] > cutoff_value:
        print(f"Os três primeiros números são maiores que  {cutoff_value}")
    if list_values[0] > cutoff_value  or list_values[1] > cutoff_value  or list_values[2] > cutoff_value:
        print(f"Pelo menos um dos dois primeiros números é maior que  {cutoff_value}")
```

Paradigma Imperativo

Consiste em conjuntos de instruções detalhadas que são fornecidas ao computador para executar em uma determinada ordem.

É chamado de "imperativo" porque, como programadores, ditamos exatamente o que o computador deve fazer, de uma forma muito específica. Descreve-se como um programa funciona, passo a passo. Como em uma receita de bolo.

1 - Coloque farinha em uma bacia;
2 - Adicione dois ovos;
3 - Adicione leita;
4 - Misture;
5 - Coloque na forma;
6 - Asse por 35 min;
7 - Aproveite.

Paradigma Imperativo

Para filtrar uma lista de números por aqueles acima de 5. Dizemos ao programa para iterar cada elemento do arranjo, comparar o valor do item com cutoff = 5 e, se o item for maior que 5, colocá-lo em um novo arranjo.

```
cutoff = 5
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
results = []
print('Before:')
print('\t numbers:', numbers)
print('\t results:', results)
for i in range(0, len(numbers)):
    print('i:', i, f'numbers[i]:', numbers[i])
    if numbers[i] > cutoff: results.append(numbers[i])
    else: pass
print('After:')
print('\t numbers:', numbers)
print('\t results:', results)
```

Paradigma Imperativo: Programação Estruturada/Modular

Visa otimizar o código, ao utilizar construções de controle de fluxo de programa, construtores de tomada de decisão (if, if then, else) e construtores de iteração (for, while), blocos e funções.

O principal objetivo da programação estruturada é a de melhorar a legibilidade.

Paradigma Imperativo: Programação Procedural (Processual)

Foca em subdividir um código, de uma simples sequência de instruções para uma coleção de sub-rotinas com instruções, estruturas e variáveis específicas.

Em um código não-procedural a repetição de porções específicas do código dependia de declarações do tipo ***go to***. Tornando o código mais complexo e de difícil de conservar e atualizar.

Paradigma Imperativo: Programação Procedural (Processual)

A partição de um código monolítico em sub rotinas melhora a modularidade do programa.

- **Declaração**: define o nome e os parâmetros necessários de sub-rotina.
- **Chamada**: instrução que dispara a execução de uma sub-rotina já declarada. Os dados correspondentes a cada parâmetro de uma sub-rotina são definidos dentro da instrução de chamada.
- **Retorno**: a instrução de retorno indica o resultado de uma sub-rotina. Algumas sub-rotinas podem não possuir retorno.

Paradigma Imperativo: Programação Procedural (Processual)

Boa usabilidade para uso geral, por sua re-usabilidade e de fácil rastreabilidade do fluxo do programa. Fomenta receios a respeito da segurança dos dados.

```
list1 = [10, 200, 50, 70]
list2 = [3, 26, 33, 13]

print('Sum(list1):', Sum(list1))
print('Sum(list2):', Sum(list2))
print('Subtract(list1):', Subtract(list1))
print('Subtract(list2):', Subtract(list2))
print('Multiply(list1):', Multiply(list1))
print('Multiply(list2):', Multiply(list2))
print('Divide(list1):', Divide(list1))
print('Divide(list2):', Divide(list2))
```

Paradigma Imperativo: Linguagem de Requisição a Bancos de Dados

A mais comummente usada das linguagens baseadas em modelo, o SQL é um exemplo de linguagem de consulta de banco de dados.

Uma base de dados é vital para qualquer projeto de dados, a maioria dos Sistema de Gerenciamento de Banco de Dados Relacionais.

Paradigma Imperativo: Linguagem de Requisição a Bancos de Dados

A sintaxe SQL foi modificada para simplificar sua compreensão, tornando a transição entre escrever código para o navegador e obter dados mais fácil.

```
query = '''
SELECT
   name, species, tank_number
FROM
   Fish
'''
rows = cursor.execute(query).fetchall()
print(rows)
```

Paradigma Imperativo: Linguagem de Requisição a Bancos de Dados

Atualmente existem muitas linguagens de consulta que podem fornecer vários recursos que o SQL não pode:

- PRQL: Pipelined Relational Query Language
- GraphQL
- Amazon RDS: Amazon Relational Database Service
- WebAssembly (Wasm)
- Gremlin: Graph Search
- N1QL
- Malloy
- Basis

Paradigma Imperativo: Programação Orientada a Objeto (POO)

O conceito central é entidades codificadas como objetos, que agrupam um determinado conjunto de informações (**propriedades**) e ações (**métodos**) que podem ser realizadas pela entidade.

Faz uso de **classes**, uma forma de criar novos objetos a partir de um projeto ou modelo definido pelo programador. **Objetos** criados pela **classe** são chamados de **instâncias.**

Paradigma Imperativo: Programação Orientada a Objeto (POO)

Retomando a receita de bolo, o código contém a declaração de uma classe:

```
class Bolo:

    def __init__(self, sabor, tamanho, cobertura):
        self.sabor = sabor
        self.tamanho = tamanho
        self.cobertura = cobertura

    def cozinhar(self):
        print(f"Preparando um bolo de tamanho {self.tamanho} de sabor {self.sabor}...")

    def adicionar_cobertura(self):
        print(f"Adicionando {self.cobertura} cobertura ao bolo...")
```

Paradigma Imperativo: Programação Orientada a Objeto (POO)

A criação de objetos como instâncias de classe:

```
bolo_chocolate = Bolo("chocolate", "médio", "chocolate")
print('bolo_chocolate:', bolo_chocolate)
bolo_baunilha = Bolo("baunilha", "pequeno", "baunilha")
print('bolo_baunilha:', bolo_baunilha)

bolo_chocolate.cozinhar()
bolo_chocolate.adicionar_cobertura()

bolo_baunilha.cozinhar()
bolo_baunilha.adicionar_cobertura()
```

Exercício proposto:

Escolha um ou mais dos paradigmas discutidos acima e crie um pequeno código para realizar alguma tarefa. Depois conte-nos sobre o que você fez e quais paradigmas de programação você aplicou e porquê.

Vamos ao Notebook

Referências

- Introduction To Compilers
- Types and Tokens
- Understanding Semantic Analysis – NLP
- Introduction to Syntax Analysis in Compiler Design
- Introduction of Lexical Analysis

Referências

- Programming Paradigms – Paradigm Examples for Beginners
- What is Binary Code?
- haskell.org
- Paradigms in JavaScript
- Perceiving Python programming paradigms
- Imperative and Declarative Programming Paradigms
- What is Imperative Programming?
- Functional Programming in Python: When and How to Use It
- Revisiting our “Bake the cake” recipe – objects and classes in programming languages

Referências

- Multiprocessing in Python: A Guide to Using Multiple CPU Cores
- Multithreading Principles and Patterns
- Functional Python: Embracing a New Paradigm for Better Code
- Functional Programming: Pure and Impure Functions
- What are Side-Effects in Programming?
- What is modular programming?
- Difference between Procedural and Non-Procedural language
- Lógica de Primeira Ordem: Sentenças, Proposições, Conectivos Lógicos, Princípios Fundamentais e Tabelas-Verdade

Referências

- Predicate Logic
- First-Order Logic
- First-Order Logic
- Abstract Machine
- Basics of Automata Theory
- Turing Machines
- Logical Operators
- Understanding Declarative Programming: A Comprehensive Guide
- Horn Clause
- Object-Oriented Programming in Python For Absolute Beginners
- Control flow statements in Programming

Referências

- Go Beyond SQL! 10 SQL Alternatives That You Should Use
- RDBMS (relational database management system)
- 5 NoSQL Query Languages for Efficient Data Retrieval and Manipulation
- LMQL — SQL for Language Models