python-funcional

Quase um livro, quase um tutorial, quase qualquer coisa

Project maintained by dunossauro
Hosted on GitHub Pages — Theme by mattgraham

Continue lendo: »> 01 - Funções

0. Saindo da zona de conforto

Sinta-se um vencedor, se você chegou até aqui, isso significa que quer aprender mais sobre o mundo da programação.

Aprender novos paradígmas podem te trazer muitas coisas positivas, assim como aprender linguagens diferentes, pois paradígmas e linguagens transpõem maneiras, estruturas e métodos de implementação completamente diferentes. Com isso você pode ter mais ferramentas para usar no dia a dia. Você pode aumentar sua capacidade de expressar ideias de diferentes maneiras. Eu penso que o maior limitador de um programador é a linguagem de programação em que ele tem domínio. Quando você aprende linguagens imperativas, como C, Python, Java e etc..., você se vê limitado ao escopo de criar e manipular variáveis. Não que isso seja uma coisa ruim, porém existem outras maneiras de resolver problemas e quando você tem conhecimento disso consegue avaliar melhor quando implementar cada tipo de coisa.

Você pode me dizer que aprender diferentes tipos de estruturas e maneiras de computar é uma coisa negativa pois tudo é variável nesse contexto. Mas eu penso exatamente o contrário, quanto mais você aprender da sua língua nativa, no caso estamos falando em português, maior o campo de domínio que você tem sobre como se comunicar e expressar ideias. Assim como aprender outras línguas te darão mais fundamentos para expressar ideias em outros idiomas, que não são melhores que os seu, mas diferentes e compõem diferentes estruturas, e isso pode ser libertador. Não quero me prolongar nesse assunto, mas dizer que isso pode acrescentar muito na suas habilidades cognitivas, até mesmo para usar ferramentas que você já usa no seu dia a dia.

Vamos começar fazendo uma tentativa de entender os paradígmas de programação, sem muito falatório e complicações. Um exemplo muito legal é do David Mertz em “Functional

- Usa-se programação funcional quando se programa em Lisp, Haskell, Scala, Erlang, F# etc..

- Do mesmo modo que se usa programação imperativa quando se programada C/C++, Pascal, Java, Python etc...

- Também quando se programa Prolog estamos programando usando o paradígma lógico.

Apesar de não ser uma definição muito elegante, talvez seja a melhor a ser dada em muitas ocasiões. Vamos tentar ser um pouco mais objetivos em relação ao estilo de computação, embora essa discussão não tenha fim:

- O foco de usar programação imperativa está no ato de mudar variáveis. A computação se dá pela modificação dos estados das variáveis iniciais. Sendo assim, vamos pensar que tudo é definido no início e vai se modificando até que o resultado esperado seja obtido.

- Na programação funcional, se tem a noção de que o estado deve ser substituído, no caso da avaliação, para criação de um novo 'objeto' que no caso são funções.

0.1 Mas de onde vem a programação funcional?

O florescer da programação funcional nasce no Lisp (acrônomo para List Processing) para tentar resolver alguns problemas de inteligência artificial que eram provenientes da linguística, que tinha foco em processamento de linguagem natural que por sua vez eram focados em processamento de listas em geral. Isso justifica uma grande parte do conteúdo que vamos ver aqui e seus tipos de dados variam somente entre listas e átomos. E assim foi mantido o foco de processamento de listas em todas as linguagens funcionais e suas funções e abstrações para resolver problemas relativos a listas e estruturas iteráveis. Uma curiosidade é que para quem não sabe porque em lisp existem tantos parênteses é que ele é baseado em s-expression, uma coisa que temos um "equivalente" evoluído em python, que parte dos teoremas de gramáticas livres de contexto:

```
(+ 4 5)
```

Sim, isso é uma soma em lisp. Diferente das linguagens imperativas como costumamos ver:

Uma assertiva pode ser feita dessa maneira:

- Funcional (ou declarativa)

```
(= 4 (+ 2 2))
```

- Imperativa

```
(2 + 2) == 4
```

Chega de enrolação e vamos correr com essa introdução, não viemos aqui para aprender Lisp ou C. Mas acho que parte desse contexto pode nos ajudar e muito quando formos nos aprofundar em alguns tópicos. Pretendo sempre que iniciar uma nova ferramenta da programação funcional ao menos explicar em que contexto ela foi desenvolvida e para resolver cada tipo de problema.

0.2 Técnicas usadas por linguagens funcionais

Vamos tentar mapear o que as linguagens funcionais fazem de diferente das linguagens imperativas, mas não vamos nos aprofundar nesse tópicos agora, pois são coisas às vezes complexas sem o entendimento prévio de outros contextos, mas vamos tentar só explanar pra que você se sinta empolgado por estar aqui:

- Funções como objetos de primeira classe:
  - São funções que podem estar em qualquer lugar (em estruturas, declaradas em tempo de execução).

- Funções de ordem superior:
  - São funções que podem receber funções como argumentos e retornar funções.

- Funções puras:
  - São funções que não sofrem interferências de meios externos (variáveis de fora). Evita efeitos colaterais.

- Recursão, como oposição aos loops:
  - Frequentemente a recursão na matemática é uma coisa mais intuitiva e é só chamar tudo outra vez, no lugar de ficar voltando ao ponto inicial da iteração.

- Foco em processamento de iteráveis:
  - Como dito anteriormente, pensar em como as sequências podem nos ajudar a

- O que deve ser computado, não como computar:
  - Não ser tão expressivo e aceitar que as intruções não tem necessidade de estar explicitas todas as vezes, isso ajuda em legibilidade.

- Lazy evaluation:
  - Criar sequências infinitas sem estourar nossa memória.

0.3 Python é uma linguagem funcional?

Não. Mas é uma linguagem que implementa muitos paradígmas e porque não usar todos de uma vez?

O objetivo desse 'conjunto de tópicos' é escrever código que gere menos efeito colateral e código com menos estados. Só que isso tudo feito na medida do possível, pois Python não é uma linguagem funcional. Porém, podemos contar o máximo possível com as features presentes do paradígma em python.

Exemplos de funcional (básicos) em python:

```
# Gerar uma lista da string # Imperativo
string = 'Python'
lista = [] # estado inicial

for l in string:
    lista.append(l) # cada iteração gera um novo estado

print(lista) # ['P', 'y', 't', 'h', 'o', 'n']
```

```
# Gerar uma lista da string # Funcional
string = lambda x: x

lista = list(map(str, string('Python'))) # atribuição a um novo objeto

print(lista) # ['P', 'y', 't', 'h', 'o', 'n']
```

Como você pode ver, depois de uma explanação básica das técnicas, a segunda implementação não sofre interferência do meio externo (Funções puras), evita loops e sua saída sem o construtor de list é lazy. Mas não se assuste, vamos abordar tudo isso com calma.

Primeiramente gostaria de dizer que roubei essa ideia dos infinitos livros da O'Reilly, que sempre exibem esse tópico. Mas vamos ao assunto. Este curso é para você que sabe o básico de Python, e quando digo básico quero dizer que consegue fazer qualquer coisa com um pouco de pesquisa na internet. O básico de programação se reduz a isso. Vamos falar sobre coisas simples e coisas mais complexas, mas pretendo manter o bom senso para que todos possam absorver o máximo de conteúdo possível.

Então, caso você venha do Python (OO ou procedural) você vai encontrar aqui uma introdução a programação funcional descontraída e sem uma tonelada de material difícil de entender. Caso você venha de linguagens funcionais como Haskell e Lisp, você pode se sentir um pouco estranho com tantas declarações, mas aprenderá a se expressar em Python. Caso você venha de linguagens funcionais modernas como Clojure e Scala, as coisas são bem parecidas por aqui.

Então tente tirar o máximo de proveito. Vamos nos divertir.

0.5 Apresentando o Jaber

Jaber é nosso aluno de mentira, mas vamos pensar que ele é um aluno que senta na primeira fileira e pergunta de tudo, sempre que acha necessário. Roubei essa ideia do livro de expressões regulares do Aurélio. Ele tem um personagem, Piazinho, e acho que toda interação com ele é sempre graciosa e tira dúvidas quando tudo parece impossível.

0.6 Sobre as referências

Não gosto muito de citar referências pois procurei não copiar texto dos livros, mas muita coisa contida neles serve de base para o entendimento de certos tópicos. Outro motivo é o nível de complexidade dos exemplos ou explicações que tentei reduzir ao máximo enquanto escrevia esses roteiros. Para um exemplo, você pode olhar o livro do Steven Lott, cheio de fórmulas e abstrações matemáticas que em certo ponto acabam comprometendo o entendimento de quem não tem uma sólida base em computação teórica ou matemática.

Como um todo, as referências serviram como guia, foi o que lí quando dúvidas para explicações surgiram. Não tiro nenhum crédito delas e as exponho para que todos saibam que existem muitos livros bons e que boa parte do que é passado aqui, foi aprendido neles.

0.7 Mais sobre o histórico das linguagens funcionais

Se você pretende realmente se aprofundar no assunto enquanto acompanha esse curso, fazer uma imersão ou coisa parecida. Tudo começa com o cálculo lambda mentalizado pelo incrível Alonzo Church. Caso você não o conheça, ele foi um matemático fantástico e teve

Raymond Smullyan etc...

Outro grande homem e que vale a pena mencionar e ser buscado é o Haskell Curry, um lógico que trouxe excelentes contribuições para o que chamamos hoje de programação funcional.

A primeira linguagem funcional 'oficial' (não gosto muito de dizer isso) é o Lisp (List Processing) criada pelo fenomenal John McCarthy que também vale a pena ser pesquisado e estudado.

Veremos o básico sobre os tipos de função no próximo tópico.

Continue lendo: »> 01 - Funções