Curso Programação Backend com Python

Recursos Básicos

Python

É preciso saber o histórico da linguagem Python, que atualmente está na **versão 3** (3.9 em Dezembro de 2020). Para saber a versão mais atual da Linguagem:

https://www.python.org/downloads/release

A versão 2.x do Python ainda existe e tem atualizações de segurança e bug fixes. A última versão é:

https://www.python.org/downloads/release/python-2715/

Devo usar Python 2 ou 3?

Boa pergunta! Você deve aprender Python 3, que é a versão atual da linguagem, e utilizá-la em seus projetos. Python 2 só faz sentido se for um projeto legado. Depois de aprender Python 3, é interessante ver as diferenças entre as duas versões, pois sempre pode ser interessante.

https://wiki.python.org/moin/Python2orPython3

A Wikipedia tem um excelente artigo histórico sobre Python:

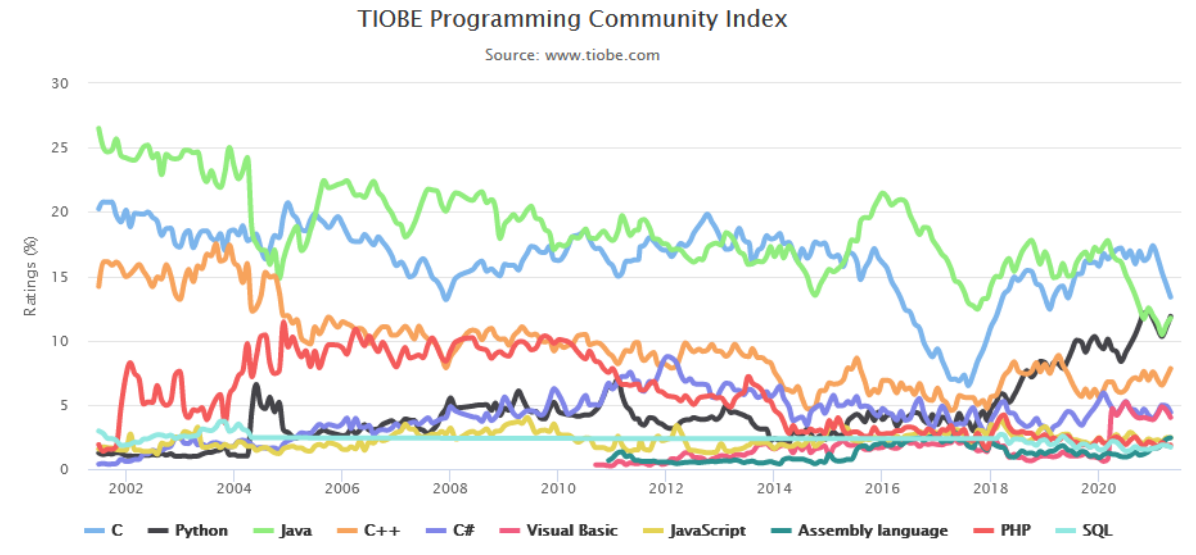
*“Python é uma linguagem de programação de alto nível, interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por **Guido van Rossum** em 1991. Atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation. Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem como um todo não é formalmente especificada. O padrão de facto é a implementação **CPython**.*

A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional. Prioriza a legibilidade do código sobre a velocidade ou expressividade. Combina uma sintaxe concisa e clara com os recursos poderosos de sua biblioteca padrão e por módulos e frameworks desenvolvidos por terceiros.

Python é uma linguagem de propósito geral de alto nível, multiparadigma, suporta o paradigma orientado a objetos, imperativo, funcional e procedural. Possui tipagem dinâmica e uma de suas principais características é permitir a fácil leitura do código e exigir poucas linhas de código se comparado ao mesmo programa em outras linguagens. Devido às suas características, ela é principalmente utilizada para processamento de textos, dados científicos e criação de CGIs para páginas dinâmicas para a web. Foi considerada pelo público a 3ª linguagem "mais amada", de acordo com uma pesquisa conduzida pelo site Stack Overflow em 2018, e está entre as 5 linguagens mais populares, de acordo com uma pesquisa conduzida pela RedMonk."

Popularidade

De acordo com a lista TIOBE (https://www.tiobe.com/tiobe-index/) de Junho de 2021 Python ultrapassou Java e se tornou a segunda linguagem de programação mais popular do mundo.



Pontos negativos

Entenda que este tópico é apenas minha opinião, portanto, você deve formar a sua. Mas não posso deixar de expressar o que considero como pontos negativos na linguagem Python.

1. **Upgrade destrutivo**: Do Python 2 para o Python 3 muita coisa mudou, causando quebra de compatibilidade;
2. **Identação forçada**: Não adianta reclamar, pois em Python a identação é sintática! Embora isso seja um pouco desagradável e, por que não dizer, inseguro, deixa o código mais limpo, sem aquele monte de chaves (“{}”), muito populares em outras linguagens;
3. **Comunidade dogmática**: A comunidade Python é um pouco “dogmática” e segue preceitos quase religiosos, tendendo a desprezar quem não os segue. Isso não é característica única da linguagem Python, pois vejo o mesmo comportamento em outras linguagens, em especial: Lua e Java;
4. **Desempenho poderia melhorar**: Há muito questionamento e blá-blá-blá sobre esta questão, mas alguns sites de benchmark mostram que certas soluções em Python podem ter um desempenho pior do que em Java (e C++): https://benchmarksgame-team.pages.debian.net/benchmarksgame/faster/python.html

Nenhum destes pontos me impede de afirmar que hoje Python é a minha ferramenta favorita, tanto para aplicações de datascience (Bigdata, Machine Learning etc) como para aplicações IoT e Web.

O “jeitão” do Python

Ok, bora mostrar alguma coisa em Python? Para começar, vamos fazer o mesmo código em Java. Vamos supor que um Professor goste de arredondar as notas dos alunos da seguinte forma:

- Se a nota for menor ou igual a 60, nenhum arredondamento é feito;
- Se a diferença entre a nota e o próximo múltiplo de 5 superior a ela for menor que 3, então arredonde para o próximo múltiplo de 5;

Então, vejamos um exemplo de notas originais e arredondadas:

Nota original	Nota final
62	62
57	57
83	**85**
91	91
74	**75**
63	**65**

As notas são números inteiros, passados por argumento de linha de comando, para simplificar.

Em Java tradicional:

```
import java.util.Arrays;
public class Grade {
   public static void main(String [] args) {
      int [] notas = new int[args.length];
      for (int x=0; x<notas.length; x++) {
         int nota = Integer.parseInt(args[x]);
         if (nota > 60) {
            int multiplo = ((nota/5)+1)*5;
            if ((multiplo - nota)<3) {
               notas[x] = multiplo;
            }
            else {
               notas[x] = nota;
            }
         }
         else {
            notas[x] = nota;
         }
      }
      System.out.println(Arrays.toString(notas));
   }
}
```

Eis a compilação e execução deste código:

```
javac Grade.java
java Grade 62 57 83 91 74 63
[62, 57, 85, 91, 75, 65]
```

Ok. Agora, vamos ver como eu faria isso em Python:

```
import sys
notas = list(map(int,sys.argv[1:]))
novas = [(lambda ng: ((int(ng/5)+1)*5) if ng > 60 and ((int(ng/5)+1)*5) \
- ng < 3 else ng)(ng) for ng in notas]
print(novas)
```

Agora, vejamos a compilação e execução deste script:

```
python grade.py 62 57 83 91 74 63
[62, 57, 85, 91, 75, 65]
```

Hmmmmm. Gostou? Estranhou? O que achou?

A primeira coisa que quero te mostrar é o caractere de continuação contra-barra (“\”) no final da terceira linha. Em Python, podemos controlar a quebra de linha inserindo uma contra-barra onde queremos separar.

Olhando assim, parece que o código Java é mais simples de entender do que o código Python, que parece um pouco confuso, certo? Mas essa é a maneira “pythonica” de escrever código, utilizando expressão **lambda**, um recurso muito utilizado no estilo de **programação funcional**.

Wikipedia: Em ciência da computação, programação funcional é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis. Ela enfatiza a aplicação de funções, em contraste da programação imperativa, que enfatiza mudanças no estado do programa.

Resolvemos o problema utilizando apenas funções, sem “loops” explícitos, portanto, simplificando-o.

O que vimos até agora?

1. Python é uma linguagem de programação interpretada, ou pseudo-compilada em uma só passagem;
2. Os tipos de dados das variáveis são inferidos a partir dos valores atribuídos;

3. Em Python usamos muitos recursos de programação funcional, em vez de programação imperativa;
4. Em Python, não precisamos criar classes, ao contrário de Java;

Wikipedia: Na Ciência da Computação, programação imperativa é um paradigma de programação que descreve a computação como ações, enunciados ou comandos que mudam o estado (variáveis) de um programa. Muito parecido com o comportamento imperativo das linguagens naturais que expressam ordens, programas imperativos são uma sequência de comandos para o computador executar. O nome do paradigma, Imperativo, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo... Este paradigma de programação se destaca pela simplicidade, uma vez que todo ser humano, ao se programar, o faz imperativamente, baseado na ideia de ações e estados, quase como um programa de computador.

Python no seu computador

Embora eu aconselhe fortemente a usar o MyCompiler enquanto ainda está aprendendo, pode ser que você queira começar com o interpretador instalado localmente, então vou mostrar como fazer isso no Linux e no Windows.

Windows

Baixe o instalador para windows diretamente do site:

https://www.python.org/downloads/windows/

Você precisa saber a arquitetura do seu computador e sistema operacional. Se for 64 bits, então selecione a opção: “Download Windows installer (64-bit)”. Agora, se for 32 bits, use esta outra opção: “Download Windows installer (32-bit)”.

Execute o instalador e pronto! Você já tem o Python. A instalação é simples, mas eu recomendo você olhar as instruções:

https://docs.python.org/3.9/using/windows.html

O Python vem com uma IDE simples chamada “idle” além do PIP, um gerenciador de pacotes e dependências para o python.

Linux

Os sistemas Linux já vêm com Python instalado. Se for um Linux antigo, virá com Python 2.7. O 20.04.2.0 LTS já vem com Python 3.8 instalado. Utilizando o comando “python3” podemos executar scripts.

Antes de instalar o Python 3 no seu Linux, saiba qual é a versão instalada com o comando:

```
python --version
```

Se a versão for menor do que 3.x, então você deve instalar o Python 3:

- **Ubuntu**:

```
$ sudo apt-get update
$ sudo apt-get install build-essential libpq-dev libssl-dev openssl
libffi-dev zlib1g-dev
$ sudo apt-get install python3-pip python3-dev
```

Para outras versões de Linux, consulte a documentação:

https://wiki.python.org/moin/BeginnersGuide/Download

Se você tem um Linux muito desatualizado, melhor atualizá-lo antes de instalar o Python 3.

Luz de emergência

Para que não se perca neste início de curso, aqui vão algumas dicas básicas sobre python.

Estrutura do programa

Um programa python é um arquivo com extensão “.py”, contendo texto. Python trabalha com UTF-8.

A estrutura de um programa é aproximadamente assim:

```
<importações>
<declaração de objetos globais>
<declaração de funções ou classes>
<código de inicialização quando importado>
<código principal quando executado>
```

Importações

Módulos externos que podem ser importados para uso dentro do seu código-fonte. Temos a instrução “import” em seus vários formatos:

```
from model.celula import Celula
import sys
```

Declaração de objetos globais

Você pode necessitar declarar objetos que são globais para o seu programa (módulo), e, neste caso, basta declará-los fora de qualquer declaração de função ou classe:

```
from model.labirinto import Labirinto
from solver import Solver
import sys
linhas = int(sys.argv[1]) if len(sys.argv)>1 else 10
colunas = int(sys.argv[2]) if len(sys.argv)>2 else 10
```

Vemos dois objetos (linhas e colunas) declarados globalmente.

Declaração de funções ou classes

```
class Celula():
     def __init__(self):
     self.paredes = [True, True, True, True];
     self.visitada = False;
     self.inicio = False;
     self.fim = False;
     self.x = 0;
     self.y = 0;
     self.ocupada = False;
     self.objeto = 0;

def funcao():
     local='Variável local'
     while True:
     var_bloco='Var bloco'
     break
     print(glob,local,var_bloco)
```

Temos uma classe “Celula” e uma função “funcao”, que podem ser instanciadas ou invocadas em qualquer ponto do seu programa (ou módulo - comece a se acostumar com esta denominação).

Código de inicialização quando importado

Assim como você pode importar módulos externos, alguém pode importar o seu programa como se fosse uma biblioteca, tendo acesso às funções e classes. Quando seu programa é importado, qualquer código imediato (aquele que não esteja dentro de uma função ou classe) é executado. Serve para inicializar sua biblioteca de objetos:

```
glob='Variável global'

def funcao():
     local='Variável local'
     while True:
     var_bloco='Var bloco'
     break
     print(glob,local,var_bloco)

funcao()
```

Neste exemplo, se alguém importar o seu programa, a variável “glob”, a função “funcao” serão importadas, mas qualquer código imediato será executado neste momento, como a chamada da função na última linha:

```
funcao()
```

Se não for este o comportamento esperado, você pode proteger o código imediato, como veremos posteriormente.

Código principal quando executado

Se você tem um código imediato a ser executado, somente quando o programa for executado diretamente (e não quando for importado por outros programas), proteja-o com um “if” especial:

```
from model.labirinto import Labirinto
from solver import Solver
import sys

if __name__== '__main__' :
      linhas = int(sys.argv[1]) if len(sys.argv)>1 else 10
      colunas = int(sys.argv[2]) if len(sys.argv)>2 else 10

      labirinto = Labirinto(linhas,colunas)

      print(labirinto)

      solver = Solver()
      solver.solve(labirinto)
      print(labirinto)
```

O python tem muitas variáveis “embutidas” que você pode utilizar. A variável “__name__” (dois sublinhados + “name” + dois sublinhados) é o nome do módulo que está sendo executado. Quando você manda executar seu programa diretamente, ela contém: “__main__”. Neste caso, todo o código será executado apenas se o programa estiver sendo executado como principal, e não se ele estiver sendo importado por outro programa.

Idiossincrasias do python

Uma das coisas importantes são os comentários. Para marcar uma linha como comentário (será ignorada pelo interpretador), basta iniciar com um caracter: “#” (jogo da velha). Deste caracter em diante, até o final da linha, tudo é considerado comentário:

```
# Esta linha inteira é comentário
xpto = abcd # Só daqui para a frente é comentário
```

E se quiser comentar várias linhas? Python não tem uma sintaxe para comentar várias linhas, mas você pode usar um texto multilinha sem associá-lo:

```
"""
Este é um teste de comentários
de múltiplas linhas.
"""
print('Hello world!')
```

Três aspas duplas inicial ou terminam um texto de múltiplas linhas, como neste caso:

```
texto = """
Este é um texto
de múltiplas linhas.
"""
print(texto)
```

Porém, se você não atribuir o texto a nenhuma variável, então ele é ignorado, como um comentário.

Em python não utilizamos caractere de finalização de linha, como o “;” do Java. Embora seja possível usá-lo. Serve para separar comandos. Por exemplo:

```
texto = """
Este é um texto
de múltiplas linhas.
""";print(texto)
```

Não faça isso! Não junte vários comandos em uma única linha.

Ah, sim! Outra coisa meio irritante é a **indentação**! Python leva isso muito a sério e você pode se irritar se fizer errado. Blocos de comandos são separados por níveis de indentação (é verdade, não é piada):

```
if xpto >= 5:
     print(“Valor alto”)
else:
     for i in range(self.linhas):
               for j in range(self.colunas):
                    matriz = self._get_celula(self.celulas[i][j])
                    self._insert(linhas,matriz,i,j)
     flattened = [y for x in linhas for y in x]
```

Ok, preste bastante atenção a este trecho de código. Quantos blocos temos? Talvez isso esclareça:

```
if xpto >= 5:
      print("Valor alto")
else:
      for i in range(self.linhas):
            for j in range(self.colunas):
                  matriz = self._get_celula(self.celulas[i][j])
                  self._insert(linhas,matriz,i,j)
      flattened = [y for x in linhas for y in x]
```

Cada cor denota um bloco de comandos, separados por indentações. O sinal de dois pontos (":") demarca o início de um bloco (deve vir na próxima linha) e todos os comandos que fazem parte do mesmo bloco devem estar com a mesma indentação. Em Java isso não existe:

```
if (xpto >= 5) {
      print("Valor alto")
} else {
      for (int i=0; i<this.linhas; i++) {
            for (int j=0; j<this.colunas; j++){
                  matriz = this._get_celula(self.celulas[i][j])
                  this._insert(linhas,matriz,i,j)
            }
      }
      // flattened = [y for x in linhas for y in x] não tem em Java
}
```

A indentação não importa em Java, já que os blocos são separados por "{" e "}".

Exercício

Bom, você já sabe como usar Python, seja na nuvem ou no seu computador. Então, pegue o programa que mostrei no tópico “O “jeitão” do Python”, digite e execute. Faça-o de preferência no PythonAnywhere, que é o ambiente utilizado neste curso, mas pode usar no seu desktop também.