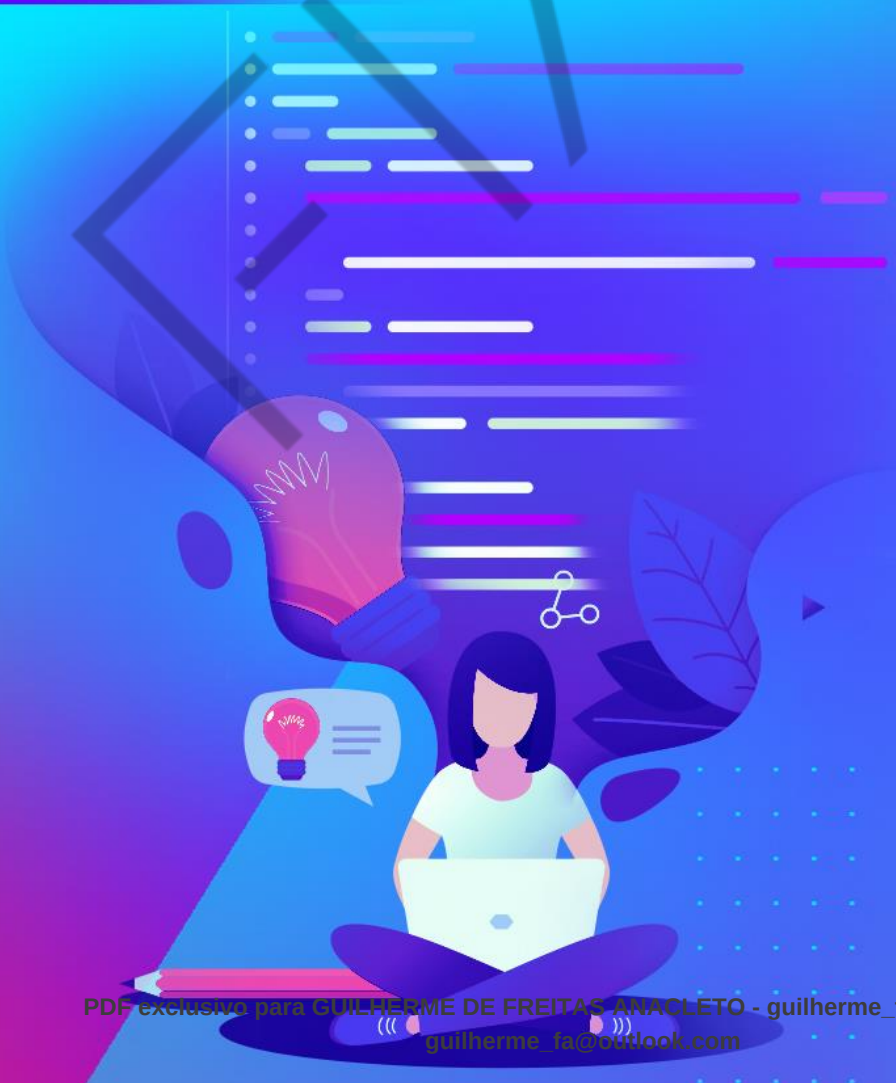
PYTHON

INTRODUÇÃO À LINGUAGEM PYTHON

HUMBERTO DELGADO DE SOUSA



1

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO À LINGUAGEM PYTHON

1.1 Por que linguagem de programação em nosso curso?

Sejam todos bem-vindos à nossa disciplina de *Optimize Technology*. Este conteúdo foi desenvolvido com a missão de demonstrar para você a utilidade de uma linguagem de programação em um ambiente em que encontramos várias “coisas” interconectadas. Essas “coisas” vão desde os nossos computadores comuns até os *wareables* que se fazem presentes dentro do nosso dia a dia, ou seja, tudo aquilo que utilizamos e que nos permite estar conectados de alguma forma.

Logo, você pode estar se perguntando, o que faremos com uma linguagem de programação dentro desse ambiente? Desenvolveremos sistemas comerciais? Armazenaremos dados em banco de dados, como normalmente é utilizada uma linguagem de programação em um curso focado para desenvolvimento de sistemas? A resposta é: NÃO!!! Vejamos alguns exemplos do enfoque que desejamos obter dentro desta disciplina:

- **Automatizar a geração de ativos de uma rede:** normalmente, essa tarefa é realizada nas grandes empresas por meio de alguma planilha em que um profissional será o responsável por documentar todos os ativos.
- **Alterar configuração dos ativos de uma rede:** imagine um profissional que deverá alterar uma faixa de *IP*, *porta*, *DNS* ou qualquer outra informação importante para um ativo de rede, alterações essas que podem ocorrer frequentemente de acordo com as normas de segurança. Agora imagine um cenário pior, em que esses ativos estão geograficamente distantes, até mesmo em países diferentes. A solução será enviar um profissional para todas as filiais? Contratar em cada filial um profissional com essas habilidades? Ou permitir que um script possa ser executado remotamente, e que seja responsável por alterar as configurações desejadas?
- Realizar geração e leitura de *logs* locais com o objetivo de gerenciar e obter informações importantes sobre o comportamento dos equipamentos

e principalmente dos usuários, por questões de desempenho ou até mesmo por segurança.

- Desenvolver programas capazes de testar portas e endereçamentos lógicos, de maneira automática e frequente, dentro de uma rede, a fim de identificar possíveis vulnerabilidades.
- Enviar e/ou receber informações para equipamentos de IoT por meio de uma porta serial.
- Direcionar *patch´s* de atualizações e/ou correções de acordo com o sistema operacional que está sendo executado na máquina cliente, sem depender da interatividade dele.

Essas são algumas das funções em que hoje aplicamos as linguagens de programações, dentro de uma rede de dispositivos, que, por sua vez, está suscetível a invasões, ataques, roubos e muitos outros prejuízos que já conhecemos ou de que ouvimos falar.

Nos dias atuais, existe uma lacuna profissional entre programadores (pensando nos profissionais que desenvolvem sistemas comerciais) e os responsáveis pela infraestrutura de uma rede de dispositivos, porque os programadores, na maioria das vezes, não se preocupam em saber como funciona tecnicamente uma comunicação por protocolos, topologias de redes, entre outros assuntos que os profissionais de infraestrutura dominam. Também não se interessam em aprender, por exemplo, quando ocorre um *deploy* de uma aplicação WEB, ou até mesmo as camadas que compõem uma aplicação comercial WEB ou não.

Com isso, se faz necessário um profissional que consiga, por exemplo, gerar um *container* (dentro de um *docker*) para uma aplicação comercial, subir esse container para a *cloud* da empresa e liberar as portas necessárias dentro do *firewall*, sem comprometer as questões de segurança.

Percebe a forte ligação que há entre uma arquitetura de sistemas (na qual atuam os desenvolvedores) e a arquitetura de redes (na qual atuam os profissionais de infraestrutura)?

Recomendo fortemente que você leia o conteúdo das duas matérias a seguir, que poderão contribuir para a sua compreensão sobre o assunto:

- https://exame.com/carreira/os-7-profissionais-mais-disputados-na-area-de-ti/
- http://igti.com.br/blog/carreira-ti-infraestrutura-servico/

Você deve estar pensando, mas isso tudo deve ser demais para um profissional só, certo? Não, quando você aprende a automatizar ações e consegue entender como funciona basicamente o mundo dos desenvolvedores.

Não queremos torná-los programadores nesta disciplina, mas desejamos que você navegue por ela aproveitando para entender como é útil o conhecimento de programação, dentro do contexto do seu curso, e que possa aproveitar ao máximo para facilitar as tarefas do seu dia a dia profissional.

A pergunta agora é: qual linguagem de programação iremos utilizar? Vamos seguir para o próximo tópico, lá encontraremos a resposta para esta pergunta…

1.2 Conhecendo o Python

Bem-vindo ao mundo dos répteis… ops… bem-vindo ao mundo Python!!! Vamos começar explicando o porquê do Python.

Além do fato de que o Python hoje é a linguagem mais utilizada para desenvolvimento de scripts para integrações de ambientes em diferentes plataformas, também é uma linguagem simples para aprender a programação que contém em sua comunidade um forte pilar para a sua propagação e incentivo à sua utilização.

Um outro fator que será importante para você é que o Python é a linguagem de programação mais utilizada no mundo de segurança da informação. Não iremos nos estender sobre sua história, mas algumas informações são importantes:

- O Python foi lançado na Internet em 1991 (comparada com outras linguagens mais tradicionais, como a linguagem C, por exemplo, o Python é um bebê).
- O seu criador chama-se Guido van Rossum, um holandês que desenvolveu essa linguagem com o objetivo de criar algo que não se

parecesse com qualquer outro concorrente. Por isso, já na sua concepção, o Python tinha como objetivo atender os profissionais que trabalham com tecnologia, mas que são “resistentes” à programação, como: cientistas, engenheiros de diversas áreas, especialistas em dados, especialistas em segurança, entre outros. Por isso, duas palavras nortearam o trabalho de Guido van Rossum: legibilidade e produtividade.

- Trabalharemos utilizando a versão 3.6.1, cuidado… pois existem grandes diferenças entre as gerações 2 e 3 do Python, ou seja, entre o Python 2.x e o Python 3.x, há um abismo de diferenças que podem interferir diretamente no seu código.
- Alocação de memória e gerenciamento de recursos são conceitos com que não precisamos nos preocupar em Python, uma vez que a própria linguagem se encarrega de controlar, de maneira eficiente, esses serviços. Em relação ao desenvolvedor, isso é de grande valia, uma vez que o programador foca apenas no desenvolvimento da sua solução/ferramenta.
- O Python é uma linguagem livre, que depende bastante da sua comunidade, por isso, não deixe de participar “ativamente” da comunidade. Temos uma comunidade para brasileiros, disponível no site: http://python.org.br/. Você irá se surpreender em como poderá ser útil e como irá aprender de maneira mais rápida e eficiente participando de uma comunidade.

Um pouco mais sobre as características do Python, veremos no tópico a seguir, mãos à obra…

1.2.1 Python é interpretado?

Esta é uma característica muito polêmica no mundo dos desenvolvedores, por isso iremos abordar de uma maneira mais ampla e procuraremos ser o mais claro possível para alguém que não seja programador.

Vamos começar demonstrando as duas arquiteturas mais comuns, utilizadas pelas linguagens de programação:

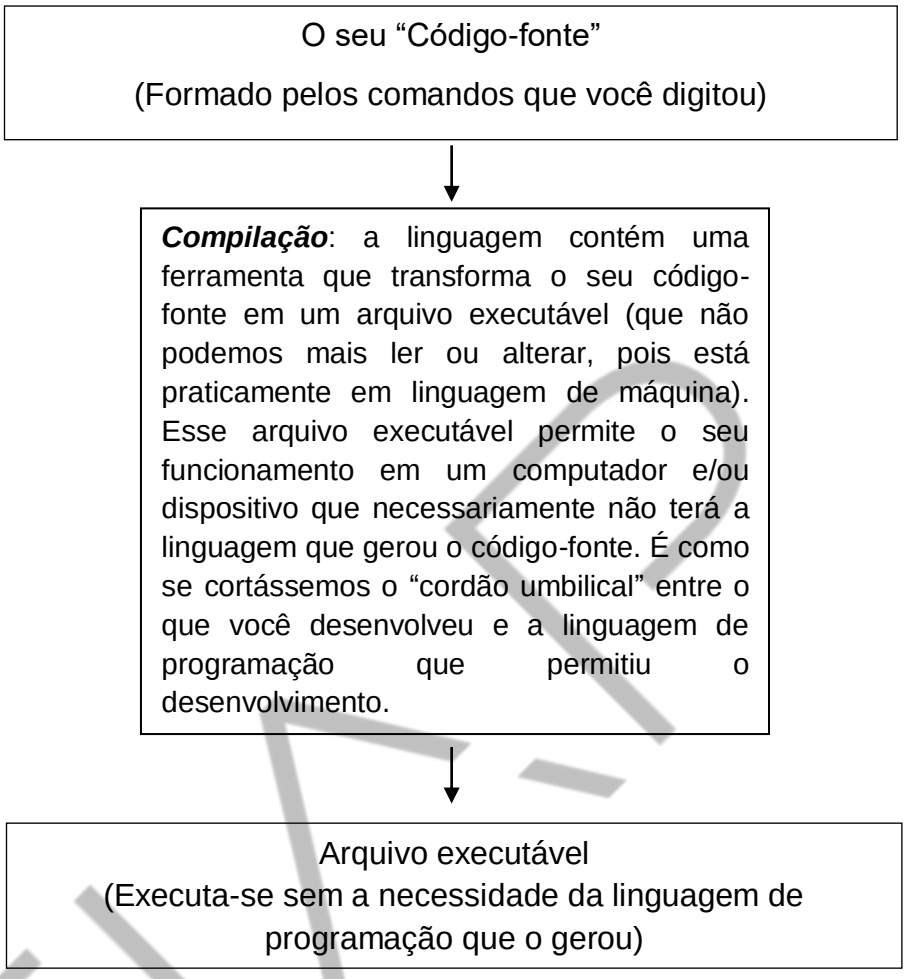


Figura 1.1 – Modelo para representação de uma linguagem compilada
Fonte: Elaborado pelo autor (2017)

O exemplo acima demonstra uma *compilação*, o caso da linguagem C, por exemplo, em que você escreve um arquivo com comandos em C, gera um arquivo-fonte, fonte: aplicacao.c. Quando conclui o seu código, precisa executá-lo em outro computador, que não terá a linguagem C instalada, então, você compila, ou seja, gera um arquivo executável no seu computador, normalmente com *extensão* “exe”, o qual ficaria com o nome aplicacao.exe. Esse arquivo executável não depende mais da existência do C na máquina em que será executado.

Vejamos agora outra arquitetura utilizada pelas linguagens de programação. Um exemplo que utiliza essa arquitetura é a linguagem *Basic*, em que o código-fonte agora não será convertido, apenas será interpretado por alguma aplicação que a própria linguagem possui, ou seja, caso queira executar sua aplicação, deverá instalar o responsável da linguagem que interpretará o seu código-fonte.

Não será gerada outra forma de arquivo. É como ocorre com o HTML (que não é exatamente uma linguagem de programação, mas não iremos entrar no mérito da questão nesse momento) que depende de um browser para que seja executado. Sem um browser, um HTML é apenas um aglomerado de códigos.

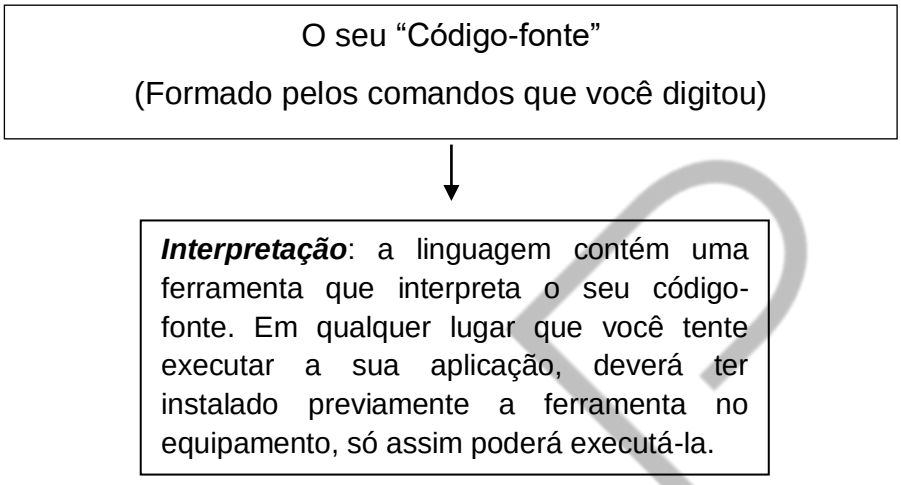


Figura 1.2 – Modelo para representação de uma linguagem interpretada
Fonte: Elaborado pelo autor (2017)

Bem, agora o que ocorre com o Python? O Python trabalha dentro da arquitetura interpretada, ou até mesmo com as duas arquiteturas (compilada e interpretada) juntas, formando uma arquitetura híbrida, o que termina causando as eternas discussões: o Python faz uso da arquitetura interpretada, compilada e/ou híbrida?

Quando você gera o seu código-fonte em Python (arquivo com extensão .py), você pode compilá-lo, o que irá gerar um arquivo com extensão .pyc (*Python Compile*), que não é independente, e precisará da ferramenta que irá interpretar o seu conteúdo, a qual chamamos de *PVM* (*Python Virtual Machine*). Logo, para executar um arquivo compilado em Python, você não precisa do Python como ferramenta, necessitará apenas da *PVM* instalada, ou seja, nesse processo ele compilou e interpretou, logo é híbrido.

Mas como já afirmamos anteriormente, o Python não obriga a compilação, ou seja, um arquivo .py (código fonte) pode ser diretamente interpretado pela *PVM* (essa é a situação mais comum de utilização do Python no mercado), logo ele é interpretado.

Vejam, que as duas situações podem existir. Aqui no curso, você utilizará o Python dentro da arquitetura interpretada, mas cabe a você agora discutir se o Python é uma linguagem que trabalha com a arquitetura interpretada ou híbrida. Boas discussões!!!!

Caso você queira saber mais sobre as vantagens e desvantagens de uma linguagem que utiliza uma arquitetura híbrida, compilada ou interpretada, recomendamos as leituras das referências bibliográficas para se aprofundar no assunto. Como a ideia não é torná-los programadores, os conceitos apresentados até agora são suficientes para continuarmos a nossa jornada.

1.2.2 Python, qual o seu paradigma?

Filosófico o título deste tópico, não é mesmo? Vamos entender primeiro o que é paradigma de linguagem de programação. Um paradigma de programação nada mais é do que uma forma de pensar e organizar os comandos dentro do desenvolvimento de programa. Cada paradigma possui as suas devidas vantagens e desvantagens e a ideia aqui novamente não é entrar em discussões sobre a melhor ou pior.

Aqui entra, sim, mais uma vantagem do Python em relação às outras linguagens que são mais engessadas quanto aos paradigmas. Isso porque o Python é multiparadigma, ou seja, atende alguém que já esteja adaptado a pensar e organizar os seus códigos por meio do paradigma *Orientado a Objetos*, aqueles que fazem uso do paradigma *Estruturado* e até mesmo aqueles que trabalham com o paradigma *funcional*, que é o paradigma mais distante do Python, mas que também pode ser utilizado.

Essa característica de ser multiparadigmas termina tirando o peso do conhecimento prévio sobre os paradigmas e a preocupação em adotar mais esses conceitos que são mais importantes para aqueles que atuam ou desejam atuar como desenvolvedores. Dentro da nossa disciplina, iremos utilizar o paradigma *estruturado*, por isso, irei abordá-lo com um pouco mais de detalhes.

O paradigma *estruturado* faz uso basicamente de três pilares: sequência, decisão e iteração. Esses três pilares suportam e induzem a criação de rotinas

simples que possuam poucas linhas de códigos, ou simplesmente só o suficiente para que possam funcionar.

Imagine o seguinte: você precisa desenvolver para sua empresa diversas ferramentas: gerador de backup, criador de arquivos de log, leitor para arquivos de log, identificação de sistemas operacionais das máquinas cliente, testador de ips, varredor de portas lógicas abertas, enfim, essas são algumas rotinas que você deverá desenvolver. Se você criar tudo dentro de um único arquivo de código-fonte, ele ficará muito grande, com muitas linhas de código, o que irá dificultar manutenção, leitura, reaproveitamento do código, entre outras situações corriqueiras e do dia a dia.

Pense. Se você precisar enviar para um companheiro da sua área somente as linhas responsáveis pela geração de um backup ou ainda enviar as linhas responsáveis por varrer portas lógicas para outros departamentos que contenham uma Intranet, deverá ficar retalhando e separando algumas linhas de um amontado total, isso não é prático.

Dentro do paradigma estruturado, você será induzido a criar pequenas funções/sub-rotinas que irão conter apenas as linhas suficientes para a execução de uma ação específica. Então, ficará muito mais fácil realizar uma manutenção ou compartilhar funções que já estão prontas, tanto para exportar para outros projetos como importar para o seu projeto. Além disso, também permite-se dividir as tarefas de criação e gerenciamento de funções dentro do time de profissionais da sua empresa.

Muito mais prático, não? Perceba como isso está no seu dia a dia. Um exemplo bem prático: o controle remoto da sua televisão. Se ele ainda não existisse, você seria obrigado a ligar, trocar os canais e controlar o volume diretamente na sua TV, certo? E se essas funções dessem qualquer tipo de problema? Você deveria levar até um técnico para realizar a manutenção. Pois bem, veja como foi melhor após a aplicação da estruturação/modularização.

Com todas essas funções (ligar/desligar, aumentar/diminuir volume, trocar de canais) em um controle remoto, caso você tenha algum problema nesses módulos, basta levar o controle para a manutenção ou trocar por outro, o que torna o processo muito mais simples e rápido.

E ainda pode ser reaproveitado, pois, caso você tenha outra TV igual, poderá aproveitar o controle remoto de uma em outra. Percebeu como modularizar é importante para as manutenções/atualizações e também para o reaproveitamento? Assim funcionarão os códigos que iremos desenvolver dentro do Python.

1.2.3 Python é alto nível

Vamos entender agora por que o Python é considerado alto nível? Seria porque ele é utilizado pela alta elite da sociedade? Ok, sabemos que a piadinha foi horrível, mas, então, vamos ao que interessa. Imagine uma longa estrada, onde, em uma das suas extremidades, encontraremos a CPU e, na outra, o desenvolvedor.

Pois bem, quando você trabalha com linguagem de alto nível, o seu programa/ferramenta parte de um ponto dessa estrada, mais próximo ao desenvolvedor. Isso significa que ele está mais distante da CPU e, para chegar lá, vai demorar um pouco mais, quando comparamos com uma linguagem de baixo nível, que está mais próxima da CPU.

Por isso, existe uma diferença de performance entre uma linguagem de alto nível e uma linguagem de baixo nível. A quantidade de processos que existem entre a ferramenta do desenvolvedor e a CPU será maior quando a linguagem utilizada é de alto nível, e haverá menos processos quando a linguagem for de baixo nível.

Por outro lado, pelo fato de a linguagem de alto nível estar mais próxima do desenvolvedor, é mais legível para ele. Quanto mais se avança na estrada em direção à CPU, mais difícil vai ficando a leitura e a interpretação do código pelo programador. Por exemplo, observe as linhas abaixo:

```
se nota<6 .e. nota>=4 então
```

```
if nota<6 and nota>=4:
```

```
if (nota<6 && nota>=4) {
```

Na primeira linha, a interpretação é mais clara, pois está em altíssimo nível. Na verdade, não é nenhuma linguagem de programação de mercado, mas seria a linha de comando de uma ferramenta chamada VisualG, que auxilia no processo de aprendizagem de programação.

Já na segunda linha, a interpretação começa a ficar um pouco mais obscura, ela está escrita em Python; e, na terceira linha, o nível é ainda mais baixo, tornando o código mais ilegível, está escrita em Java. Perceba que, quanto mais alto o nível, mais fácil para ser compreendida, ou seja, mais fácil para que você possa aprender.

Falando em aprender, está na hora de colocarmos a mão na massa. Lembre-se de que até agora a maior preocupação foi situar você em relação ao Python, esclarecer a importância dessa linguagem para sua formação, apresentar os conceitos mais comuns com que você poderá se deparar no mercado de trabalho, entender como funciona e qual a principal proposta da linguagem de programação Python.

Seguimos em frente.

1.3 Instalando o Python

Iremos verificar agora como instalar e configurar o Python na sua máquina.

1º-) Baixe o Python, diretamente do site dele, e evite os sites genéricos que "auxiliam" nos downloads, segue o link:

https://www.python.org/downloads/release/python-361/

Dentro da página, vá até a seção *Files* e faça o download de acordo com a sua plataforma do sistema operacional. Conforme a imagem abaixo:

Files

Version	Operating System	Description
Gzipped source tarball	Source release	
XZ compressed source tarball	Source release	
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later
Windows help file	Windows	
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64
Windows x86 embeddable zip file	Windows	
Windows x86 executable installer	Windows	
Windows x86 web-based installer	Windows	

Quadro 1.1 – Plataformas disponíveis para instalação do Python
Fonte: Python.org (2017)

2º-) Após a conclusão do download, execute o instalador e muita atenção à primeira tela da instalação:

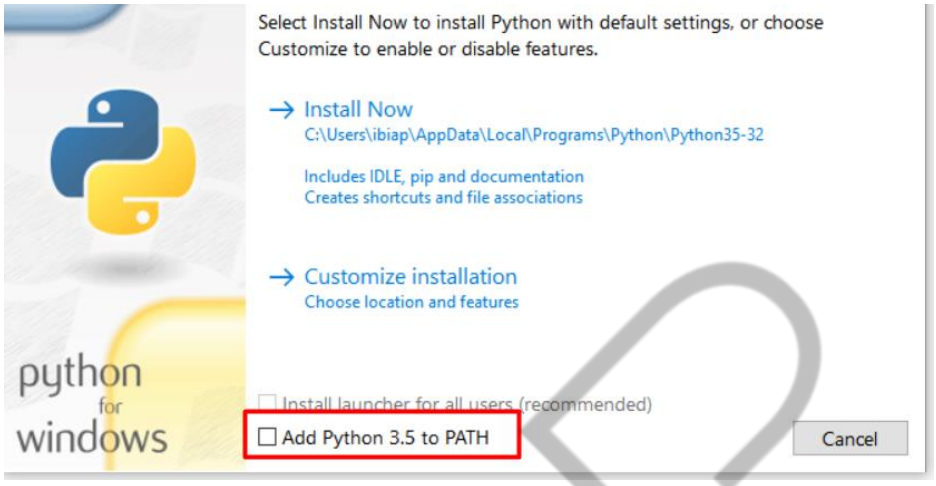


Figura 1.3 – Janela de instalação do Python
Fonte: Python.org (2017)

A parte destacada, na figura acima, deve ser selecionada, porque irá adicionar as referências para a execução do Python, dentro das variáveis de ambiente do sistema operacional, por isso, é fundamental selecioná-la. Caso não o faça, poderá executar o Python somente a partir da pasta em que você o instalou.

Em seguida, clique sobre a opção "***Install Now***", caso queira aceitar todas as sugestões-padrão, como ícones que estarão na sua área de trabalho, local de instalação etc. Se, por outro lado, preferir alterar alguma dessas configurações, utilize a opção "***Customize Installation***".

3º-) Após a conclusão da instalação, abra o *prompt de comando*:

- Se estiver no Windows, utilize, <Windows> + <R>, digite cmd e pressione <enter>.
- No *prompt de comando*, digite Python e, então, estará dentro do IDLE, que é o ambiente puro do Python para o desenvolvimento. Ou seja, você já está dentro do Python para programar.
- Digite **exit()** e pressione <*enter*> para sair do IDLE.
- Digite **exit** e pressione <*enter*> para sair do CMD.

4º-) Entretanto, não iremos programar dentro do IDLE, devido à sua falta de flexibilidade e praticidade em montar os códigos. Para isso, iremos utilizar o

PyCharm, uma das ferramentas mais utilizadas para programação Python, que se enquadra na categoria IDE. O IDE permite programar em determinada linguagem e possui recursos que viabilizam a produtividade e a praticidade durante o desenvolvimento e a execução do seu programa. Baixe o PyCharm por meio do link:

https://www.jetbrains.com/pycharm/download/

A página que será aberta possui duas opções de download, conforme podemos observar na imagem abaixo:

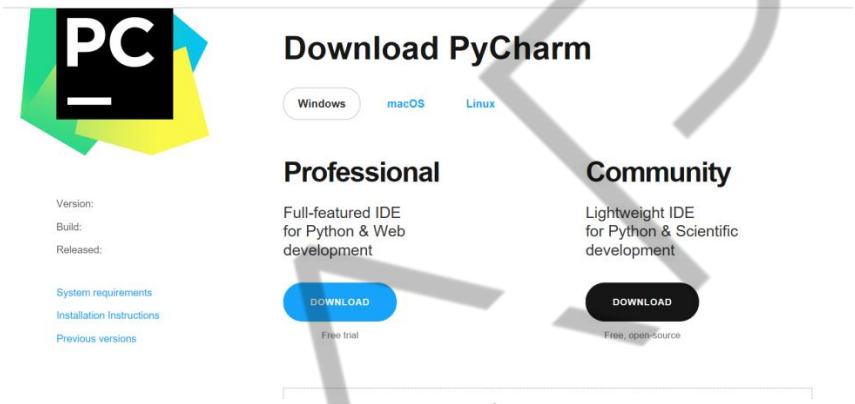


Figura 1.4 – Versões disponíveis do PyCharm
Fonte: Jetbrains.com (2017)

A opção *Professional* é uma versão gratuita por um determinado tempo, se quiser utilizá-la após o prazo, deverá pagar pela ferramenta. A versão *Community* é a versão gratuita, que você poderá utilizar livremente. Para a nossa disciplina, basta a versão *Community*, mas fique à vontade para escolher a que melhor lhe convier.

Após realizar o download, poderá instalar o PyCharm, não há qualquer observação relevante dentro do seu processo de instalação.

5º-) Ao executar o PyCharm pela primeira vez, irá surgir a tela abaixo:

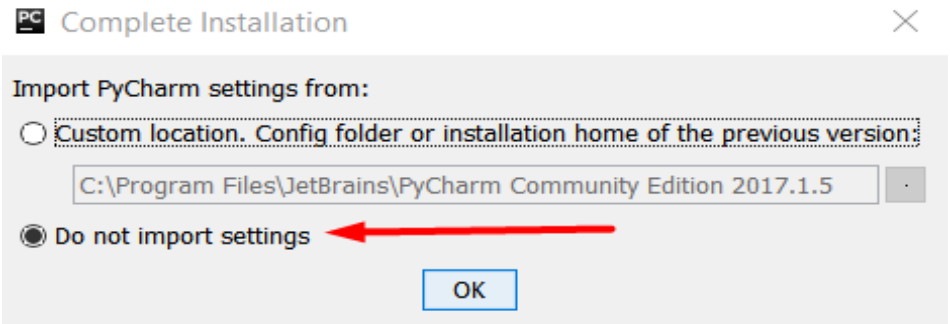


Figura 1.5 – Iniciando o PyCharm
Fonte: PyCharm – JetBrains (2017)

Marque a opção que está apontada pela seta vermelha e clique em ok. Logo depois, irá surgir a tela abaixo:

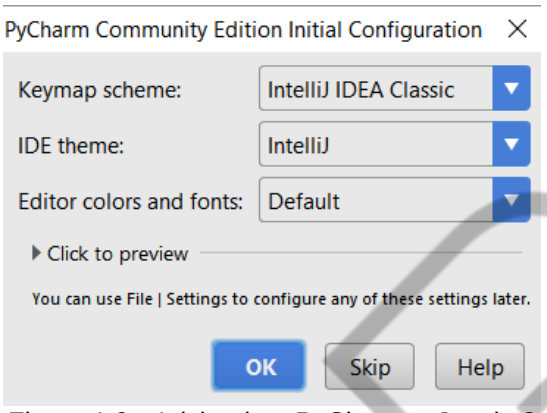


Figura 1.6 – Iniciando o PyCharm – Janela 2
Fonte: PyCharm – JetBrains (2017)

Não há necessidade de qualquer alteração na tela acima, todos os itens são modificáveis posteriormente e se referem à apresentação do ambiente, cores, temas etc. Pode clicar no OK e dar prosseguimento.

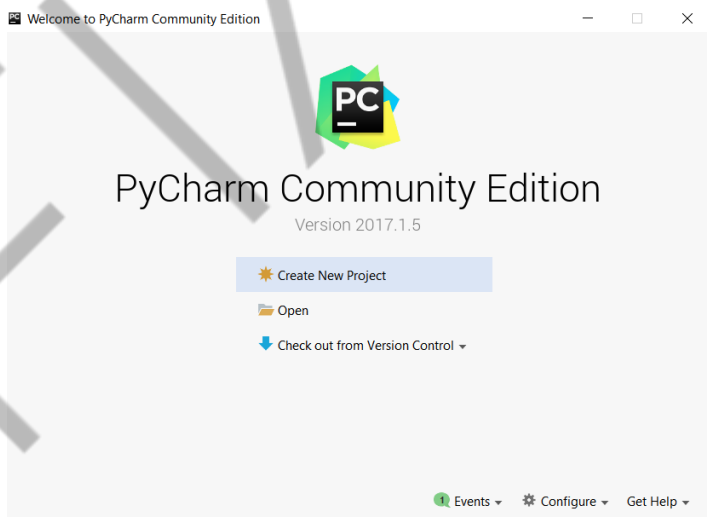


Figura 1.7 – Iniciando o PyCharm – Janela 3
Fonte: PyCharm – JetBrains (2017)

Na tela acima, poderá escolher se deseja iniciar um novo projeto ou abrir um projeto já existente, no nosso caso, selecione a opção “**Create New Project**”.

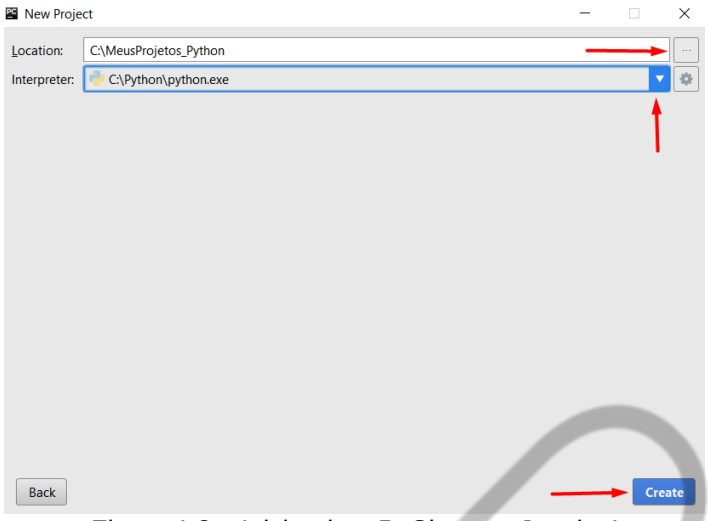


Figura 1.8 – Iniciando o PyCharm – Janela 4
Fonte: PyCharm – JetBrains (2017)

Na tela acima, deveremos selecionar o local para gravarmos o nosso projeto e, **principalmente,** selecionar no *DropBox* o local em que está instalado o Python na sua máquina. Com isso feito, clique em *Create*, caso apareça uma janela de dicas, e pronto o seu PyCharm estará aberto, conforme poderemos observar na figura abaixo:

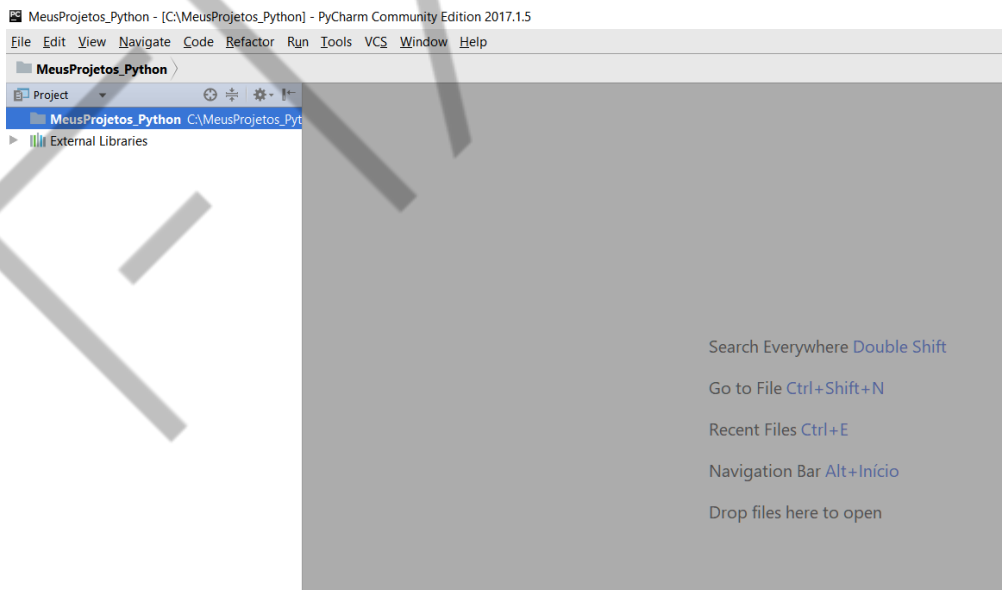


Figura 1.9 – IDE PyCharm
Fonte: PyCharm – JetBrains (2017)

Deixe o seu projeto selecionado, clique sobre ele com o botão direito do mouse e escolha a opção *New* e *Directory*, conforme é demonstrado na imagem abaixo:

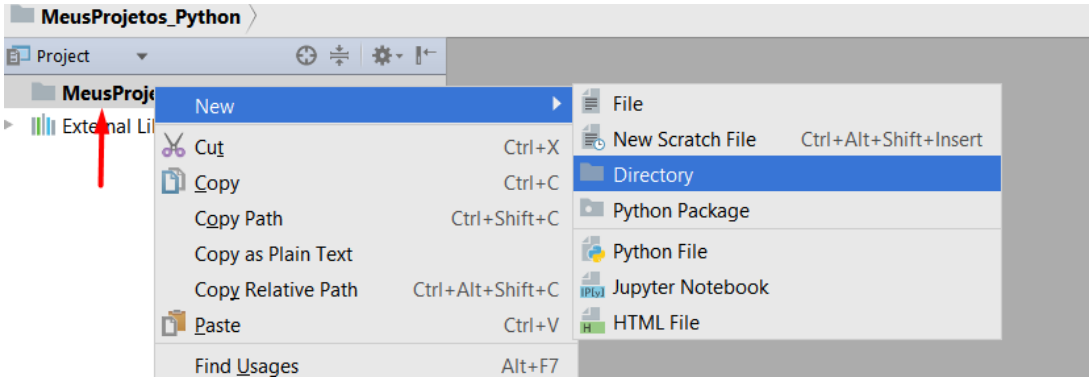


Figura 1.10 – Primeiro Projeto – Etapa 1
Fonte: PyCharm – JetBrains (2017)

Digite “PrimeiroProjeto” e clique em “OK”. Deverá aparecer uma pasta dentro do seu projeto. Clique com o botão direito sobre a sua pasta “PrimeiroProjeto” e escolha as opções: *New* e *Python File*, conforme podemos observar na imagem a seguir:

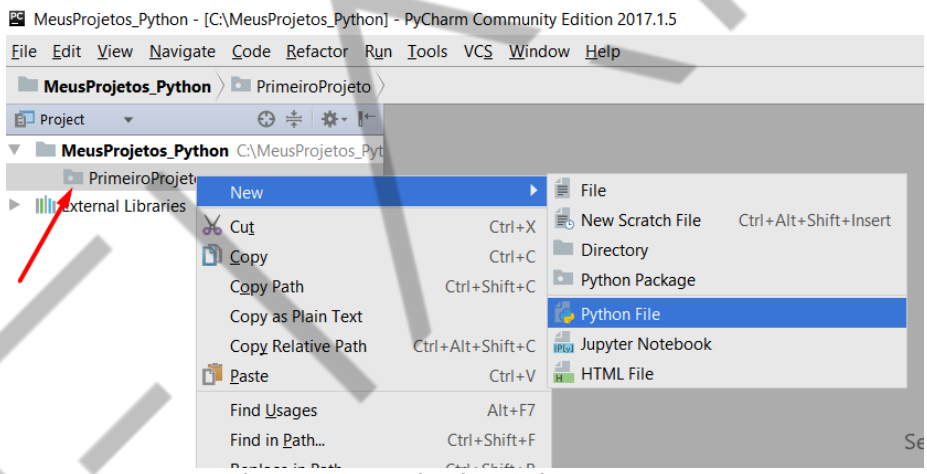


Figura 1.11 – Primeiro Projeto – Etapa 2
Fonte: PyCharm – JetBrains (2017)

Na caixa que será aberta, digite “HelloWorld” e clique em “OK”.

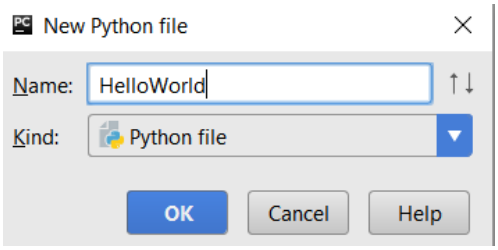


Figura 1.12 – Primeiro Projeto – Etapa 3
Fonte: PyCharm – JetBrains (2017)

Sua tela, então, ficará conforme a imagem abaixo:

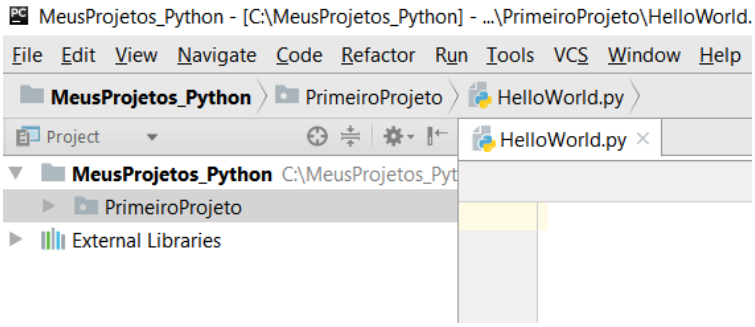


Figura 1.13 – Primeiro Projeto – Etapa 4
Fonte: PyCharm – JetBrains (2017)

Do seu lado direito, você terá o espaço dentro do arquivo HelloWorld.py, para poder digitar o seu primeiro programa.

Segundo os mais supersticiosos, você somente aprenderá a linguagem se começar por um HelloWorld. Não que sejamos supersticiosos, mas também não tem por que duvidar, não é mesmo?

Então, vamos lá, digite os seguintes comandos:

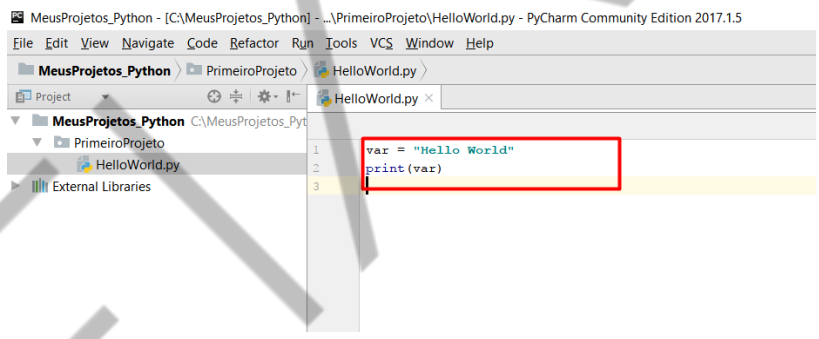


Figura 1.14 – Primeiro Projeto – Etapa 5
Fonte: PyCharm – JetBrains (2017)

Agora, precisamos interpretar esse nosso código-fonte, para isso, será necessário: clicar com o botão direito sobre o seu arquivo “HelloWorld.py” e selecionar a opção “Run HelloWorld”. Conforme está demonstrado na imagem abaixo:

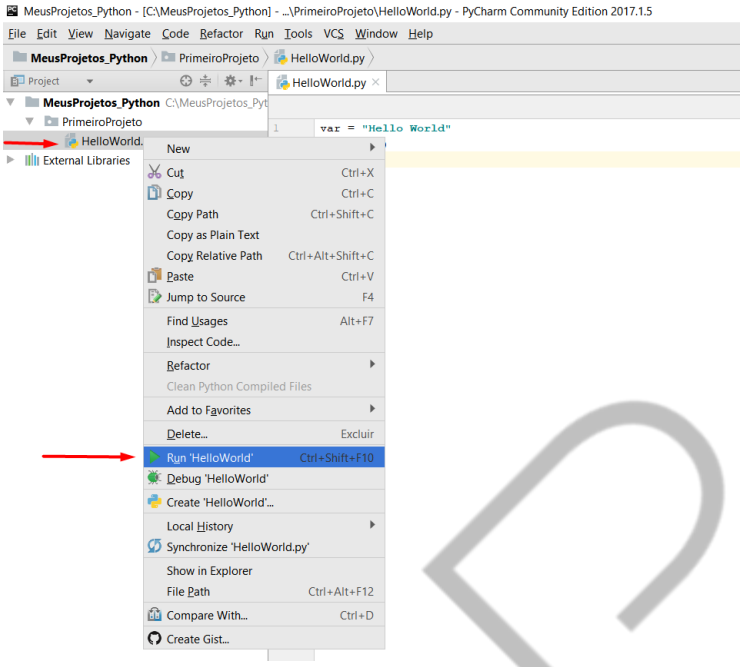


Figura 1.15 – Primeiro Projeto – Etapa 6
Fonte: PyCharm – JetBrains (2017)

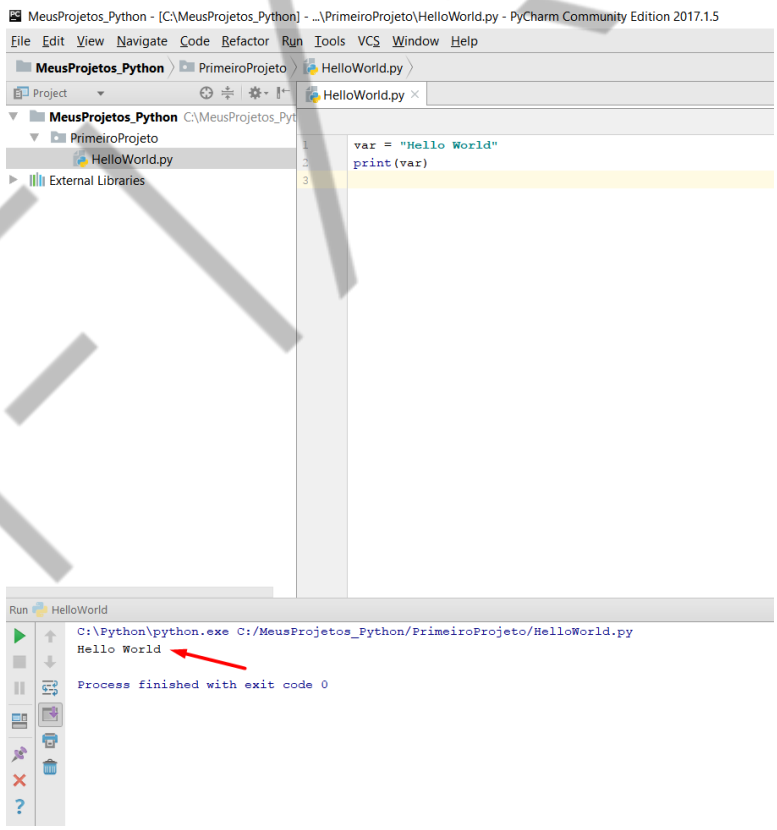


Figura 1.16 – Primeiro Projeto – Etapa 7
Fonte: PyCharm – JetBrains (2017)

Na parte inferior da sua tela, você verá o resultado da execução do seu primeiro programa, que foi a exibição da mensagem “Hello World”, conforme você

pode verificar na figura acima. Ufa... Quanto trabalho neste capítulo, hein? Bom, encerraremos por aqui. No próximo capítulo, explicaremos não só o que foi feito dentro do nosso código do “HelloWorld”, mas outros conceitos mais práticos, como variáveis, tipos de dados e muito mais. Reviva os principais conceitos desta etapa e incorpore-os no seu dia a dia. Até o próximo capítulo!

REFERÊNCIAS

ELMASRI, Ramez; NAVATHE, Shamkant, B. **Sistemas de banco de dados**. 6. ed. São Paulo: Pearson Addison-Wesley, 2011.

FORBELLONE, Andre Luiz Villar. **Lógica de Programação**. 3. ed. São Paulo: Prentice Hall, 2005.

KUROSE, James F. **Redes de computadores e a Internet:** uma abordagem top-down. 6. ed. São Paulo: Pearson Education do Brasil, 2013.

MAXIMIANO, Antonio Cesar Amaru. **Empreendedorismo**. São Paulo: Pearson Prentice Hall Brasil, 2012.

PIVA, Dilermando Júnior. **Algoritmos e programação de computadores**. São Paulo: Elsevier, 2012.

PUGA, Sandra. **Lógica de programação e estruturas de dados**. São Paulo: Prentice Hall, 2003.

RHODES, Brandon. **Programação de redes com Python**. São Paulo: Novatec, 2015.

STALLINGS, W. **Arquitetura e organização de computadores**. 8. ed. São Paulo: Pearson Prentice Hall Brasil, 2010.