1 começando a codificar

Encontrando seu caminho

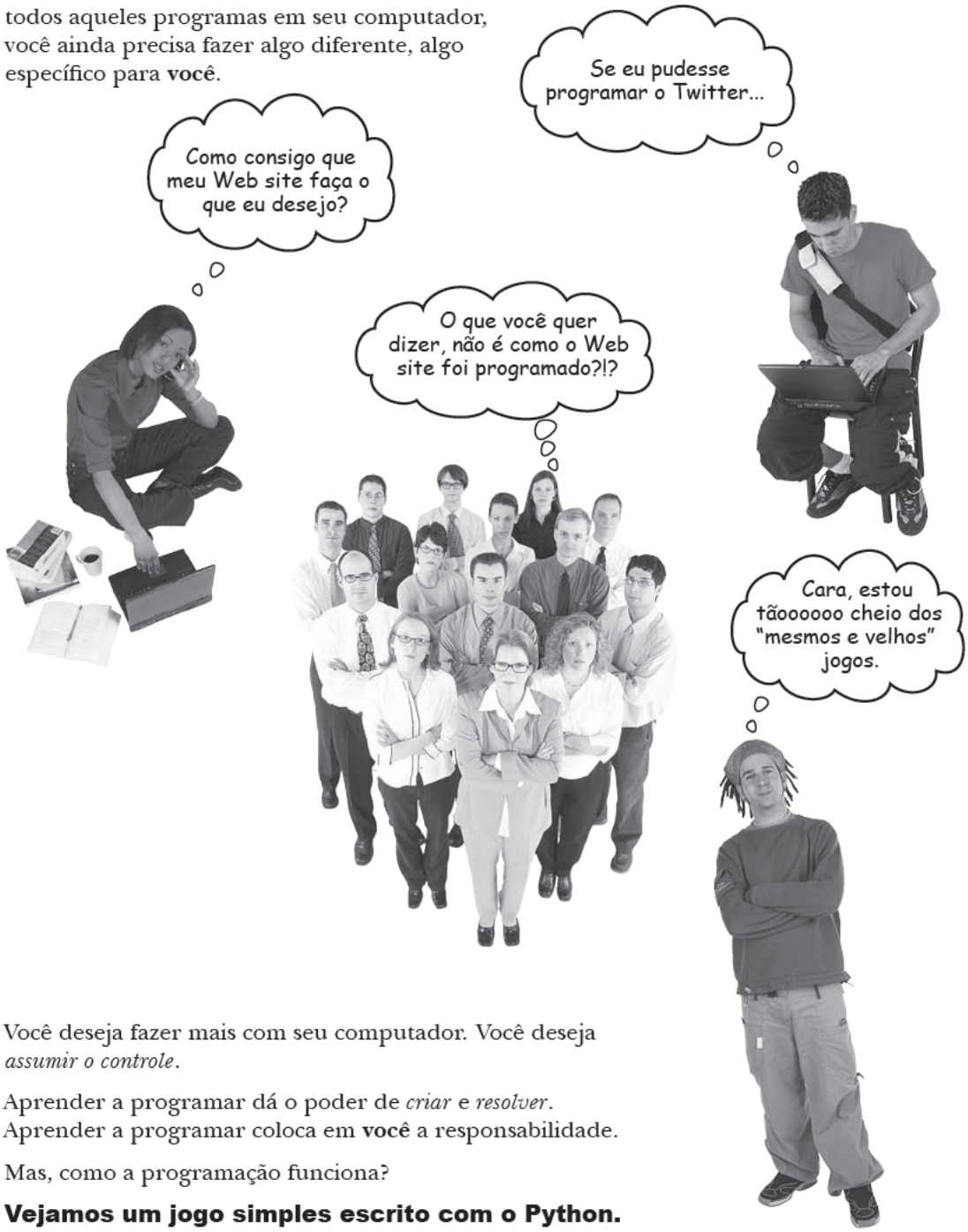
Escrever programas dá o poder de controlar seu PC.

Quase todos sabem como *usar* um computador, mas poucos realizam a próxima etapa e aprendem como *controlá-lo*. Se você usar o software de outras pessoas, sempre estará limitado àquilo que as outras pessoas acham que você deseja fazer. Escreva seus próprios programas e o único limite será sua própria imaginação. A programação irá torná-lo mais criativo, fará você pensar mais precisamente e irá ensiná-lo a analisar e resolver os problemas logicamente.

Você deseja ser programado ou ser o programador?

A programação permite que você faça mais.

Você tem problemas para resolver e trabalho a fazer, mas seu software existente não termina com isso. Mesmo com todos aqueles programas em seu computador, você ainda precisa fazer algo diferente, algo específico para **você**.



Você deseja fazer mais com seu computador. Você deseja *assumir o controle*.

Aprender a programar dá o poder de *criar* e *resolver*. Aprender a programar coloca em **você** a responsabilidade.

Mas, como a programação funciona?

Vejamos um jogo simples escrito com o Python.

Aponte seu lápis

Este código é um programa de adivinhação. Estude-o com cuidado, compare cada linha de código no programa e escreva o que você acha que o código faz. Se não estiver certo sobre o que uma determinada linha de código faz, *não se preocupe*, mas tente adivinhar de qualquer modo. Uma linha já foi fornecida para você começar:

```
print("Welcome!")
g = input("Guess the number: ")
guess = int(g)
if guess == 5:
    print("You win!")
else:
    print("You lose!")
print("Game over!")
```

.....

.....

Converte a entrada em um número.

.....

.....

.....

.....

.....

.....

Este código é escrito na versão 3 da linguagem de programação Python, que é usada neste livro.

Aponte seu lápis
Solução

Este código é um programa de adivinhação. Você deve escrever o que acha que o código faz.

Não se preocupe se suas respostas são diferentes das nossas. Se forem parecidas, então, tudo está certo.

`print("Welcome!")`	Exibe uma mensagem de boas vindas.
`g = input("Guess the number: ")`	Pede ao usuário para fornecer uma adivinhação.
`guess = int(g)`	Converte a entrada em um número.
`if guess == 5:`	O número adivinhado era igual a 5?
`    print("You win!")`	Informa ao usuário: "You win!" (Você venceu!)
`else:`	Do contrário,
`    print("You lose!")`	informe ao usuário: "You lose" (Você perdeu!)
`print("Game over!")`	Termina o programa

Mas o que são g e guess?

Você pode estar imaginando o que são `g` e `guess` no código. São as chamadas **variáveis** e são usadas para controlar os dados na memória do computador.

Uma variável

O valor fornecido será conhecido como "g".

```
g = input("Guess the number: ")
guess = int(g)
```

Uma variável

Isto cria uma versão numérica do valor g e chama-o de "guess".

Uma variável é realmente apenas um **rótulo** para os dados. Portanto, se o usuário digitar "3" no teclado, então `guess` será definida para o número 3 e sempre que o computador ler `guess`, ela será lida como o valor 3.

Veja bem!

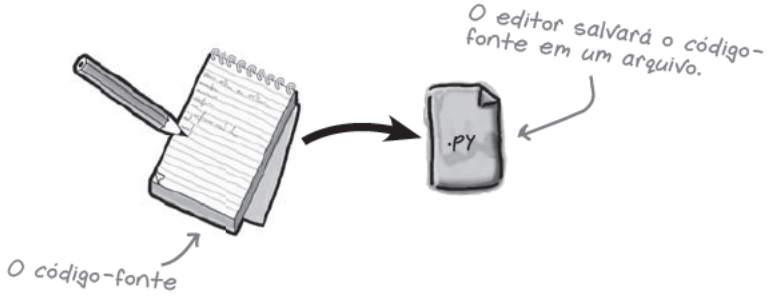
Tenha cuidado com os sinais de = no código.

As linguagens de programação usam os sinais de = para diferentes finalidades. Na maioria das linguagens (inclusive o Python), um sinal de igual duplo (= =) é um teste de igualdade. Significa: "estas duas coisas são iguais?". Em oposição, um sinal de igual simples (=) é uma instrução (conhecida como atribuição) que significa: "defina o valor para".

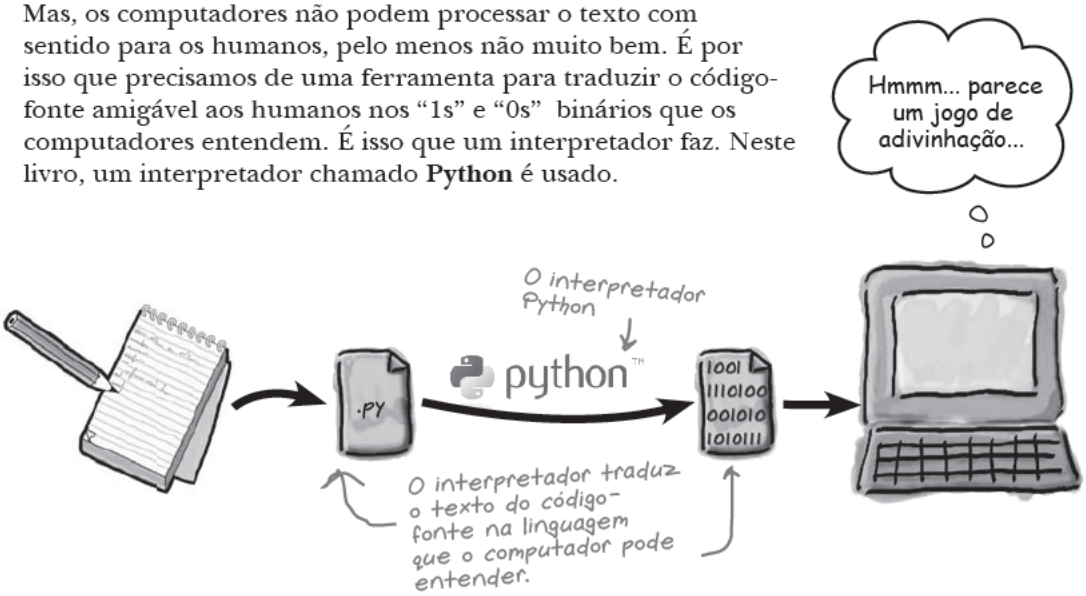
Então, como você executa seu código?

Há **duas** coisas que você precisará para executar o programa de adivinhação: um **editor** e um **interpretador**.

O editor salva o código escrito em um arquivo em seu disco rígido. O código (algumas vezes chamado de **código-fonte**) é apenas texto e pode ser escrito, e lido, pelos humanos.



Mas, os computadores não podem processar o texto com sentido para os humanos, pelo menos não muito bem. É por isso que precisamos de uma ferramenta para traduzir o código-fonte amigável aos humanos nos "1s" e "0s" binários que os computadores entendem. É isso que um interpretador faz. Neste livro, um interpretador chamado **Python** é usado.

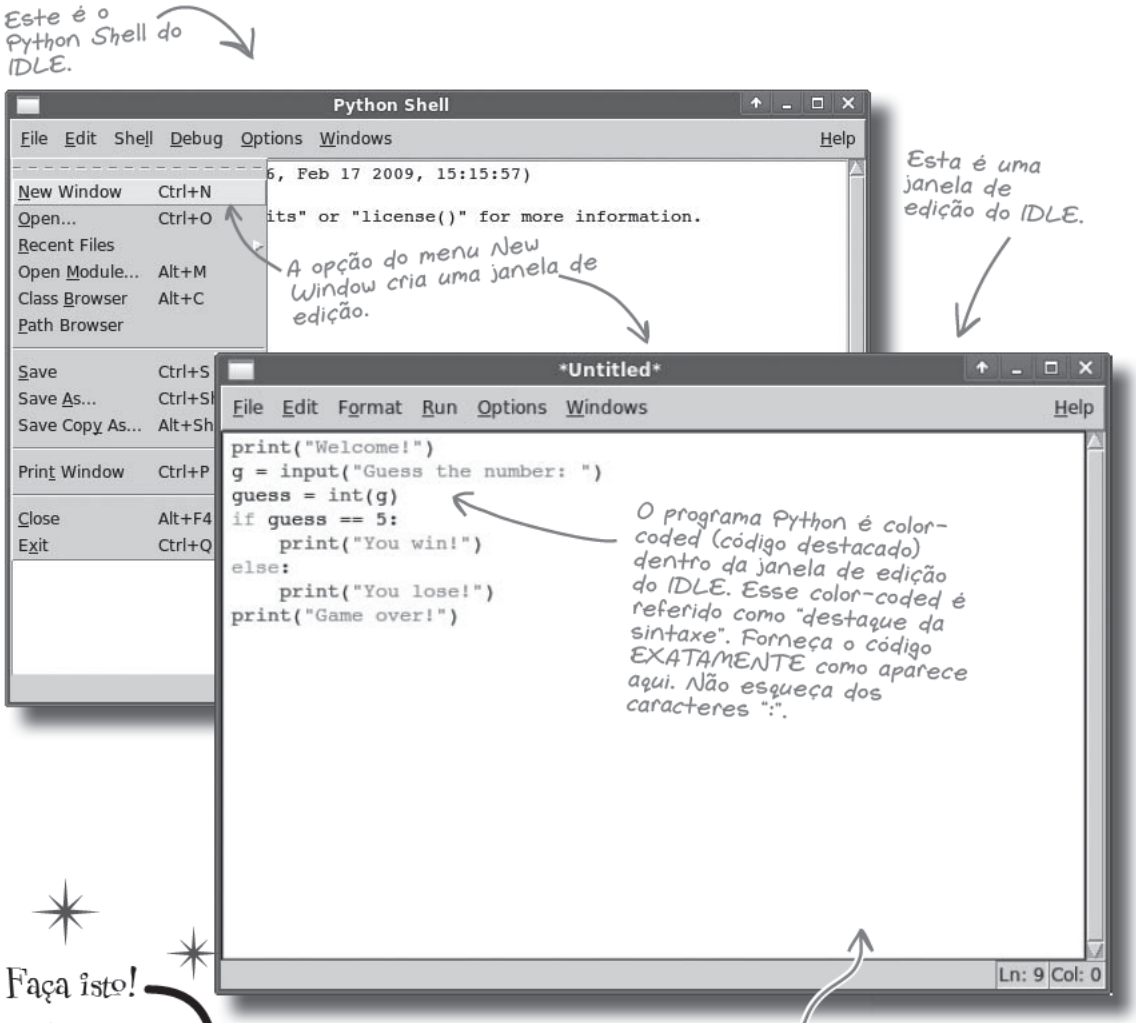


Então, precisamos de um editor e um interpretador Python. Felizmente, o Python 3 vem com um aplicativo predefinido chamado **IDLE**, que faz os dois trabalhos e mais. O IDLE permite que você escreva e edite o código Python, traduza esse código na forma binária e, finalmente, execute o programa Python 3. Por isso, o IDLE é conhecido como um Integrated Development Environment (**Ambiente de Desenvolvimento Integrado**).

Vejamos estas etapas em ação.

Crie um novo arquivo de programa

Quando você inicia o IDLE, ele exibe uma janela chamada **Python Shell**. Selecione a opção `New Window` (Nova Janela) no menu `File` (Arquivo) do Python Shell, que cria uma nova janela de edição para você. Forneça o código de seu programa como texto nessa janela de edição e estará no caminho certo.



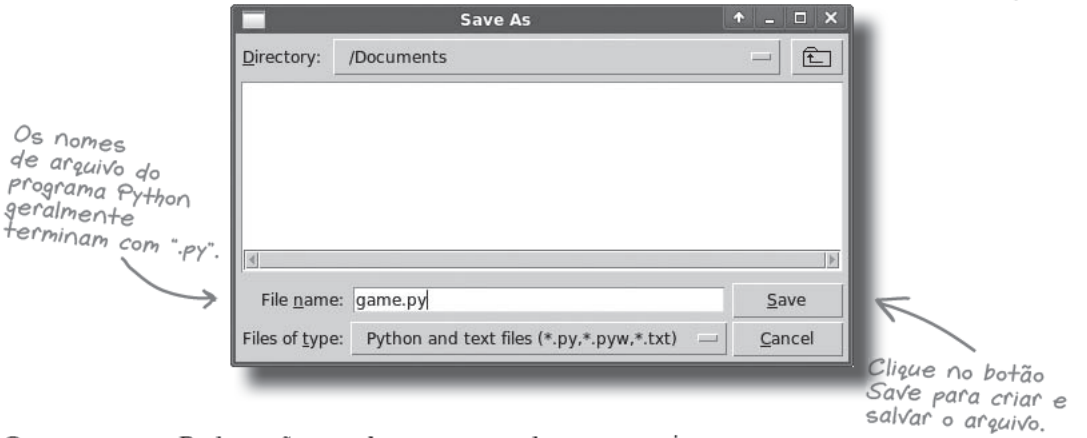
Faça isto!

Vá em frente, abra uma nova janela de edição IDLE e digite o código da página 3.

Prepare e execute seu código

A próxima etapa é *preparar* o código de seu programa para a execução. Para tanto, selecione `File` → `Save` (Salvar) no menu para salvar o código de seu programa em um arquivo. Escolha um nome apropriado para seu programa.

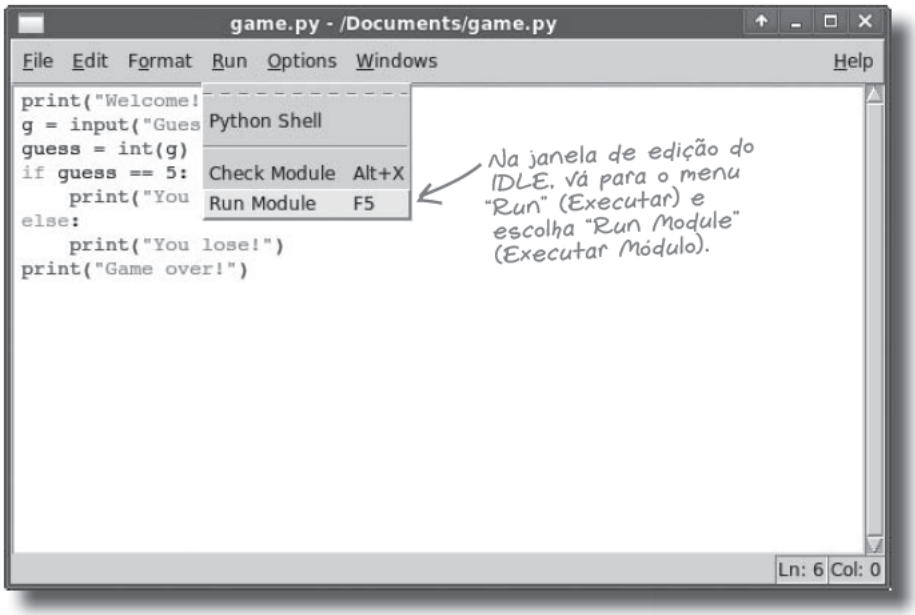
Se você escolher File → Save no menu, poderá salvar seu código em um arquivo.



Os programas Python são geralmente gravados em arquivos que terminam com `.py`, portanto iremos chamar este programa de `game.py`.

Realmente não importa para o IDLE em qual diretório você salva o arquivo. Alguns codificadores gostam de criar diretórios especiais para cada novo projeto de programação. Mas, agora, apenas salve o código em algum diretório que seja fácil de lembrar.

Agora, vejamos o que acontece quando executamos o programa.

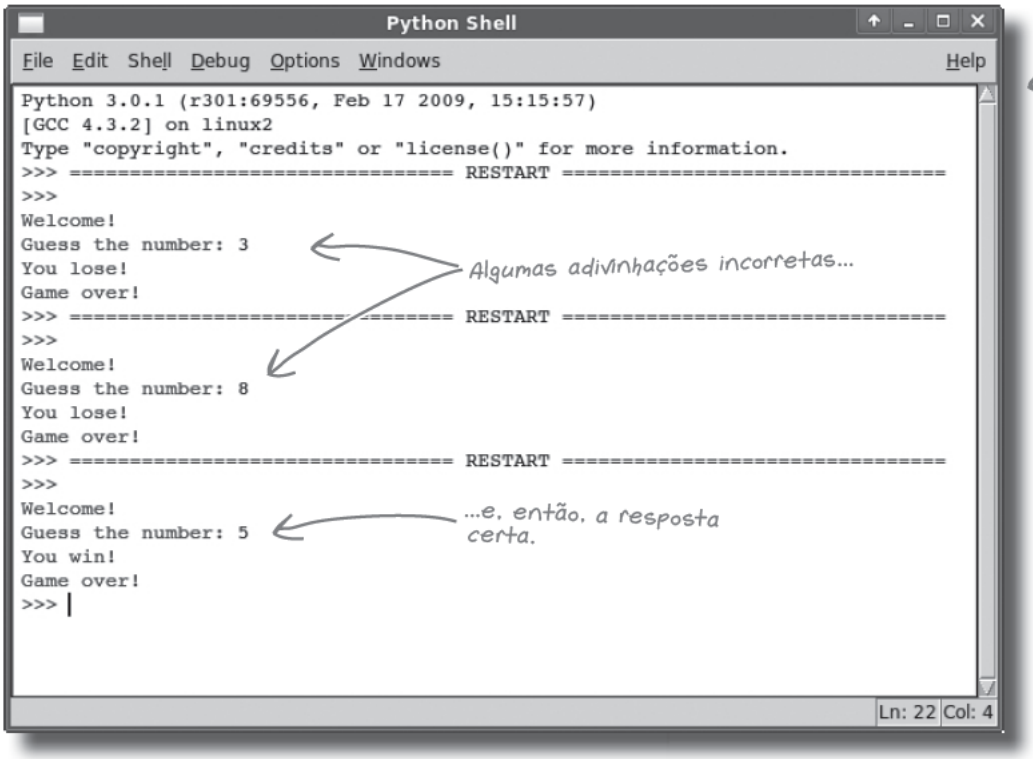


TEST DRIVE

Para executar o programa, você precisa assegurar que a janela de edição para o código do programa `game.py` esteja selecionada. Sempre que você executar (ou executar de novo) o programa, precisará clicar na janela de edição do IDLE e escolher a opção `Run Module` no menu `Run`. A palavra ***módulo*** (em inglês, *module*) é um nome que o IDLE usa para se referir ao código de seu programa.

Aqui está o que acontece quando você executa o código:

Quando você executa seu código dentro do IDLE, qualquer mensagem aparece dentro do Python Shell, não dentro da janela de edição. O IDLE torna automaticamente o shell a janela selecionada, no instante em que seu programa é executado.



Parabéns! O programa funciona.

Sempre que você executa o código, ele exibe uma mensagem "Welcome!" (Bem-vindo), aceita a entrada do teclado e, então, informa se adivinhamos ou não a resposta certa. Isso significa que o programa está aceitando a **entrada**, está **processando** os dados e, então, gera a **saída**.

Não existem Perguntas Idiotas

P: Nunca ouvi falar sobre o Python. Ele é popular?

R: O Python é usado em muitos lugares legais. O Google, Industrial Light & Magic, YouTube e NASA (para citar alguns), usam o Python. Achamos que eles sabem o que estão fazendo.

P: Então, quando eu terminar este livro, jogarei fora o Python e usarei outra coisa, tal como C# ou Java?

R: Só se você quiser. O Python pode ser a única linguagem de programação da qual você precisará. Mas sim, se você quiser aprender outra linguagem de programação, poderá pegar tudo que aprender sobre programação neste livro e aplicar em qualquer outra linguagem, com um mínimo de esforço.

P: Mas um amigo me disse que eu devo aprender o Java ou o C#. Por que você não está usando uma dessas linguagens de programação neste livro?

R: O Java e o C# são ótimas tecnologias de programação, mas podem ser difíceis de aprender, especialmente quando você está apenas começando. Este não é o caso do Python. E, de qualquer modo, este é um livro designado a ensinar a programar e a usar o Python como sua primeira linguagem de programação nos ajudará a fazer isso.

P: Parece haver muitas versões diferentes do Python. Qual devo usar?

R: Há duas versões principais do Python: 2 e 3. Este livro baseia-se na versão 3 da linguagem. O Python 3 é o futuro desta linguagem; qualquer novo recurso tem a garantia de ser adicionado à versão 3 da linguagem, não à versão 2. Naturalmente, como todas as versões, o Python 3 continua sendo um download gratuito, o que torna óbvio decidir se você pode conseguir usá-lo.

P: O Python será executado em meu telefone, como o Java?

R: Isso realmente depende de seu telefone. O Python é designado para ser executado em muitas tecnologias diferentes e sistemas operacionais. Executar seu próprio código em seu próprio telefone é uma exigência muito específica e o Java tem feito isso muito bem no momento. Como uma tecnologia de programação, o Java foi designado inicialmente para ser executado em dispositivos muito pequenos, portanto não é nenhuma surpresa que seja uma escolha forte e popular para a telefonia.

P: Por que o IDE do Python é chamado de IDLE?

R: Em parte porque parece IDE, mas achamos que tem mais relação com Eric Idle, um dos membros fundadores do grupo de comédia Circo Voador do Monty Python.

P: Pode repetir?!? O que é voador do Monty Python?

R: Circo. Sim, sabemos que parece bobo, não é? E, acredite,é. É engraçado também. O criador do Python, Guido van Rossum, é um grande fã do Monty Python e supostamente assistia as reprises do show enquanto planejava o Python. Você encontrará muitas referências ao folclore do Monty Python na comunidade Python. Os papagaios mortos é um dos favoritos.

P: O que significa int(g)?

R: Informa ao Python para interpretar a entrada do usuário como um número, ao invés de uma letra. Dentro das linguagens de programação, o número 5 é diferente da letra '5'.

P: Então, e se eu o omitisse?

R: O computador teria tratado a entrada fornecida pelo usuário do programa como uma letra. Se você perguntar ao computador se uma letra é igual a um número, ele ficará confuso e dirá que não é.

P: Por quê?

R: Porque se o computador achar que duas partes de informação são "tipos" diferentes, irá supor que não há nenhum modo de serem iguais.

P: Então, e se eu não tivesse digitado um número quando fui solicitado a adivinhar? E se eu apenas tivesse fornecido meu nome ou algo assim?

R: O código teria paralisado com um erro. Na verdade, o Python irá reclamar que o programa paralisou com um "ValueError" (mais sobre estas mensagens de erro posteriormente no livro).

O programa precisa fazer mais.

No momento, o jogo de adivinhação informa ao usuário se sua adivinhação está certa ou errada, mas nada mais que isso. Poderia ser mais útil se o programa exibisse mais mensagens informativas, tais como, se a adivinhação é **mais alta** ou **mais baixa** que a resposta certa. Isso ajudaria o usuário a se concentrar na resposta certa na próxima vez em que o programa for executado.

Podemos fazer isso mudando o código. Mas, de que modo?

Precisamos que este programa exiba mensagens que sejam mais informativas.

game.py - /Documents/game.py

File Edit Format Run Options Windows Help

```
print("Welcome!")
g = input("Guess the number: ")
guess = int(g)
if guess == 5:
    print("You win!")
else:
    print("You lose!")
print("Game over!")
```

Ln: 9 Col: 0

Aponte seu lápis

Você precisa decidir quais mensagens devem ser exibidas para o usuário. Abaixo, está uma tabela mostrando alguns valores típicos que o usuário pode fornecer. O que você acha que a mensagem deve dizer?

Modificação do Programa

Número fornecido	Mensagem a exibir
3	
5	
7	
8	

PODER DO CÉREBRO

Pense no código original. Você precisará usar mais do que apenas os comandos `print( )` para fornecer mais retorno informativo. Do que mais você precisará?

Aponte seu lápis
Solução

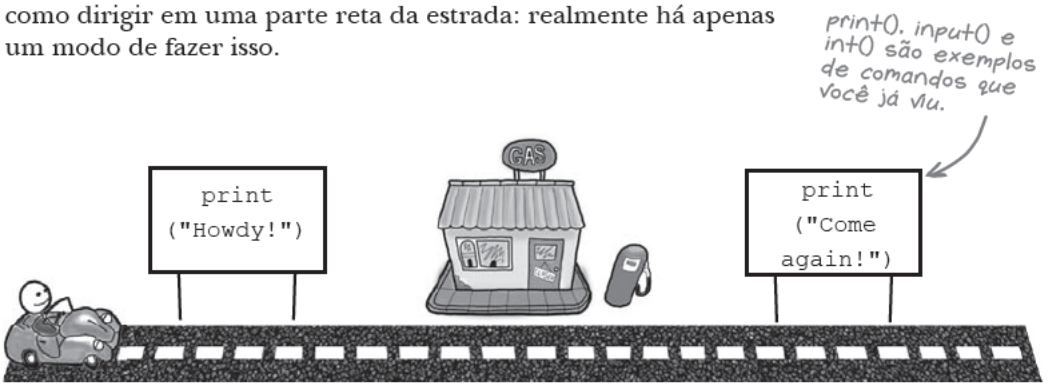
Você precisa decidir quais mensagens devem ser exibidas para o usuário. Abaixo, está uma tabela mostrando alguns valores típicos que o usuário pode fornecer. O que você acha que a mensagem deve dizer?

Modificação do Programa

Número fornecido	Mensagem a exibir
3	Too low (Baixo demais)
5	You win! (Você venceu!)
7	Too high (Alto demais)
8	Too high (Alto demais)

Um programa é mais do que uma lista de comandos

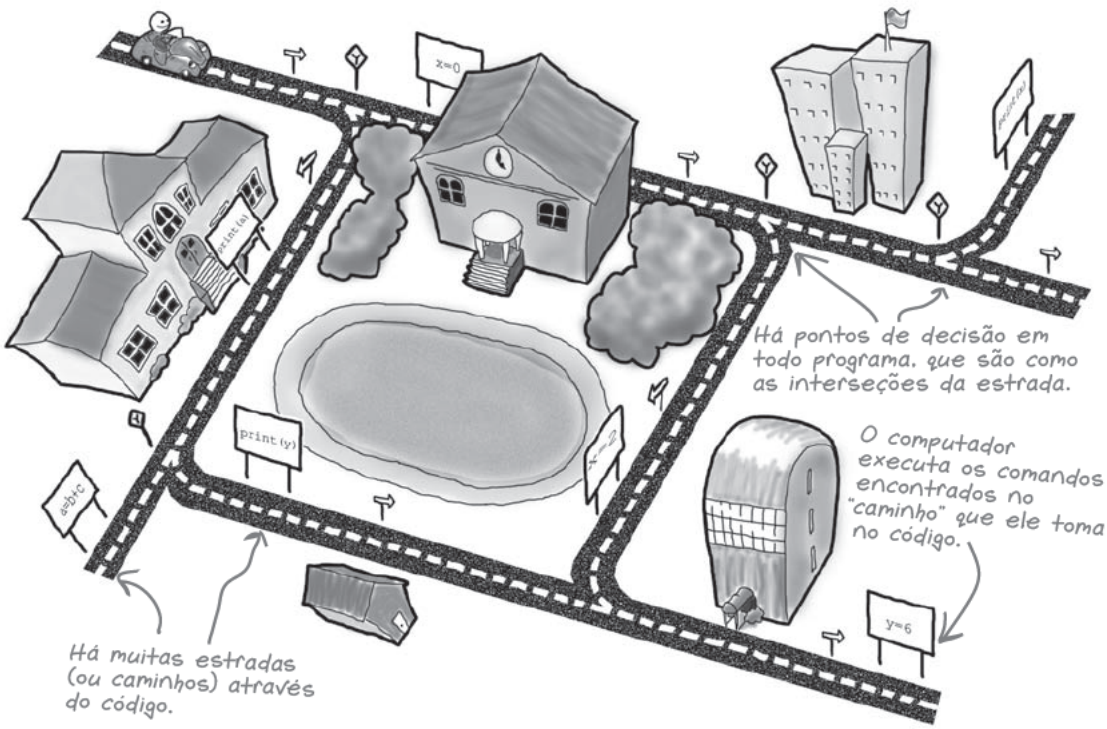
Você *poderia* criar um programa que fosse simplesmente uma **lista** de comandos. Mas, quase nunca criará. Isso porque uma simples lista de comandos pode ser executada apenas em uma direção. É como dirigir em uma parte reta da estrada: realmente há apenas um modo de fazer isso.



Mas, os programas precisam ser muito mais inteligentes que isso.

Codeville: Seu programa é como uma rede viária

Os programas precisam fazer coisas diferentes sob circunstâncias diferentes. No jogo, o código exibirá "You win!" se o usuário adivinhar o número corretamente e "You lose!", se não. Isso significa que todos os programas, até os realmente simples, em geral têm diversos **caminhos** através deles.



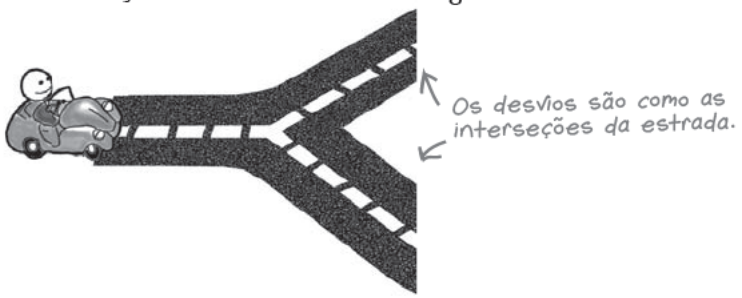
Um **caminho** (em inglês, *path*) se refere ao conjunto de instruções que o computador realmente seguirá (ou executará). Seu código é como uma rede de ruas, com muitas seções de código conectadas, como as ruas de uma cidade. Quando você dirige em uma cidade, toma decisões sobre quais ruas passar, virando à esquerda ou à direita, em diferentes interseções. É igual para um programa. Ele também precisa tomar decisões de vez em quando sobre qual caminho seguir, mas para seu código, não é como dirigir em uma estrada, *ele está executando um caminho em particular*.

Vejamos com mais detalhes como um programa decide qual caminho tomar.

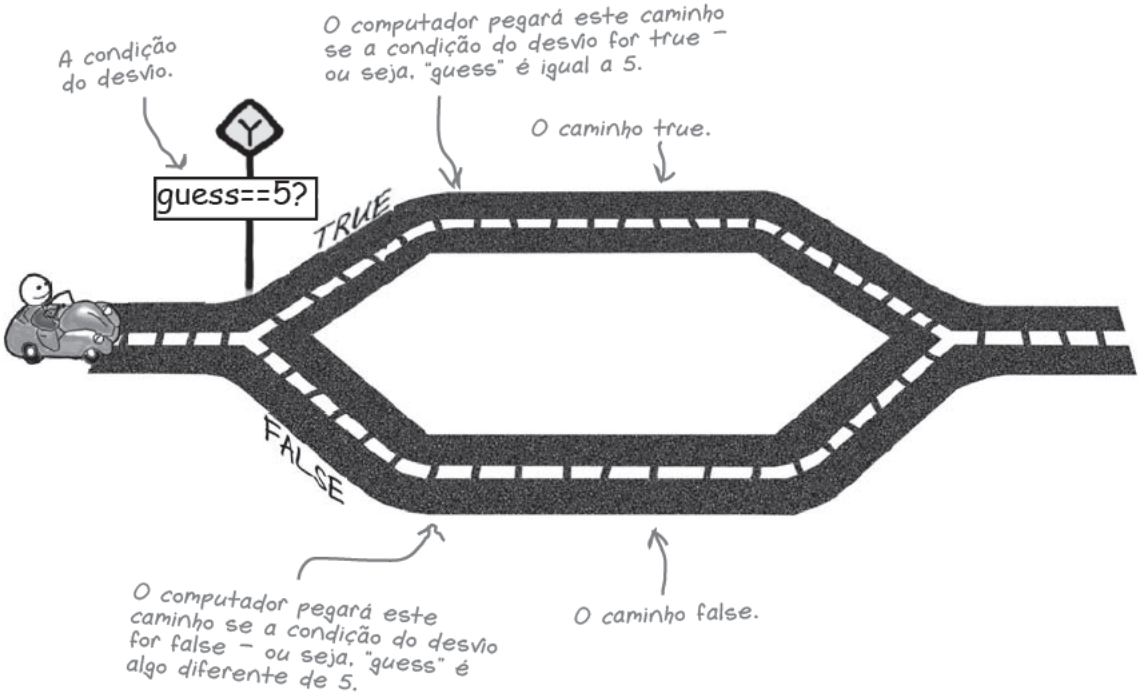
Os desvios são as interseções do código

Dirigir em uma rua é fácil. Você precisa tomar uma decisão apenas quando chega a uma interseção. É igual para seu programa. Quando um programa tem uma lista de comandos, ele pode executá-los cegamente, um após o outro. Mas, às vezes, seu programa precisa tomar uma decisão. Ele executa ***esta*** ou ***aquela*** parte do código?

Estes pontos de decisão são chamados de **desvios** (em inglês, *branches*) e são as interseções da estrada em seu código.



Seu programa toma uma decisão usando uma **condição de desvio** (em inglês, *branch condition*). Uma condição de desvio tem o valor **true** (verdadeiro) ou **false** (falso). Se a condição de desvio for verdadeira, ele executará o código no desvio true. E se a condição do desvio for falsa, executar o código no desvio false.



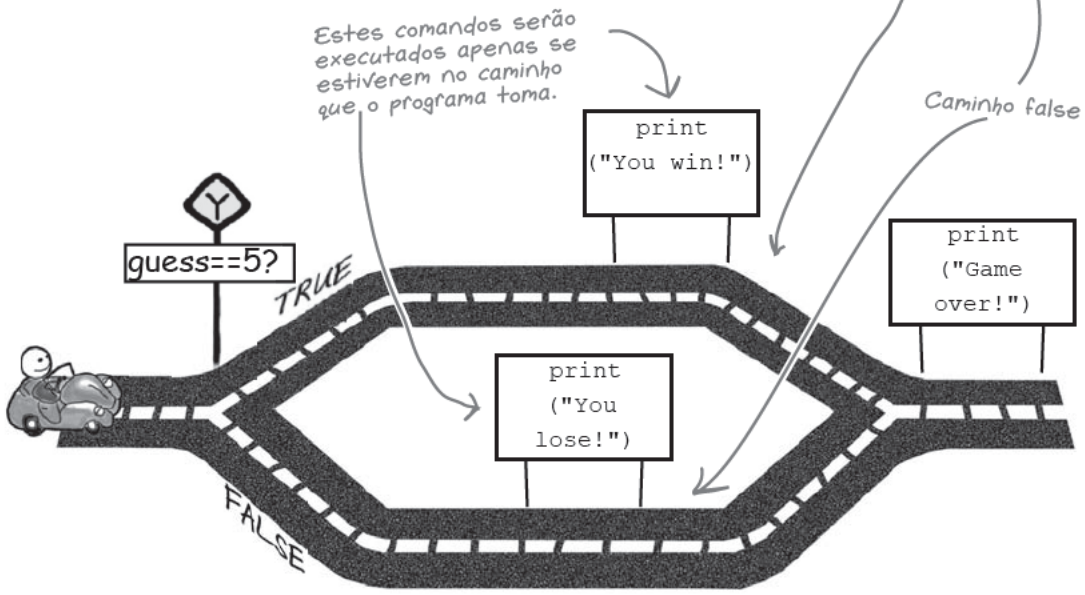
Desvios if/else

Já vimos um desvio no programa de jogo Python:

```
if guess == 5:
    print("You win!")
else:
    print("You lose!")
print("Game over!")
```

O Python, como muitas linguagens, tem desvios `if/else`. Em nosso exemplo, a condição do desvio é a parte do código `if guess == 5`. Este é um **teste de igualdade** e resultará no valor **true** ou **false**.

O código no caminho true é recuado e dado depois da linha **if**. O código no caminho false é recuado e dado depois da linha **else**:



Você precisa corrigir o programa de jogo para dar mais mensagens informativas para o usuário.

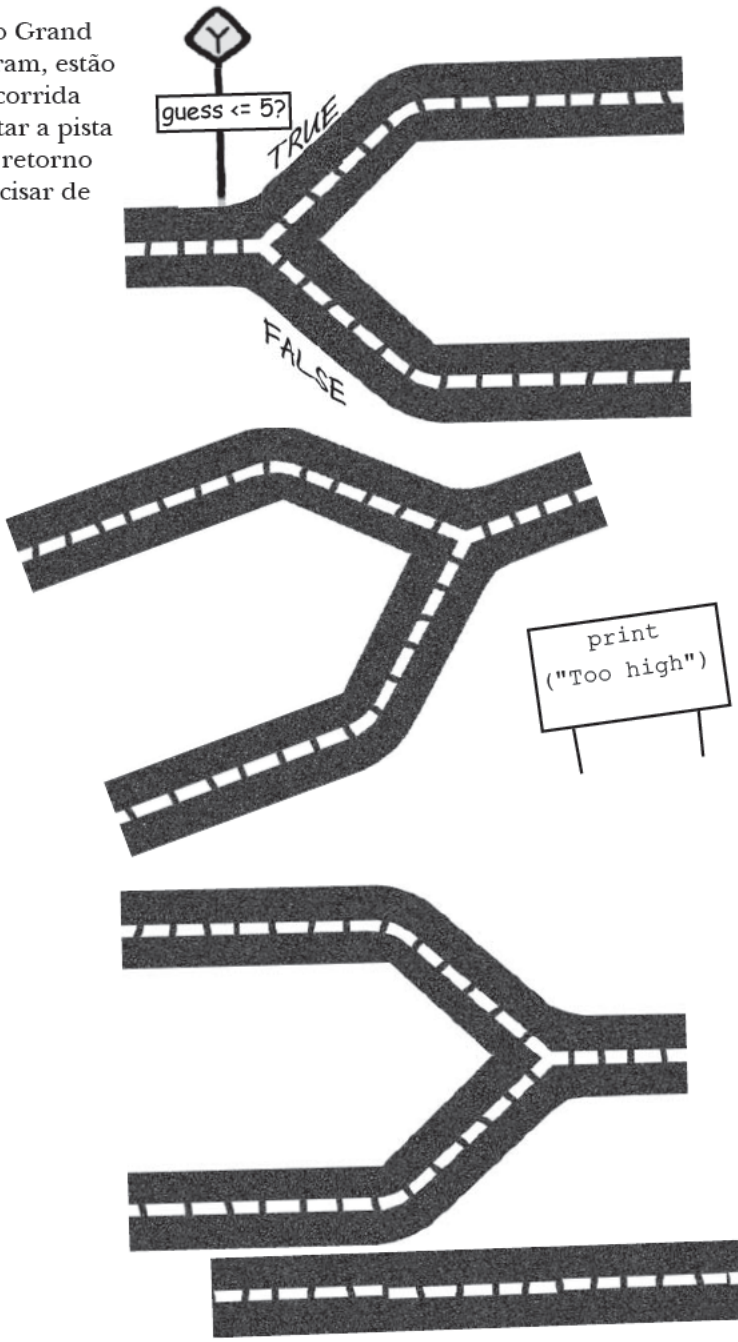
Mas, como serão os caminhos no programa?

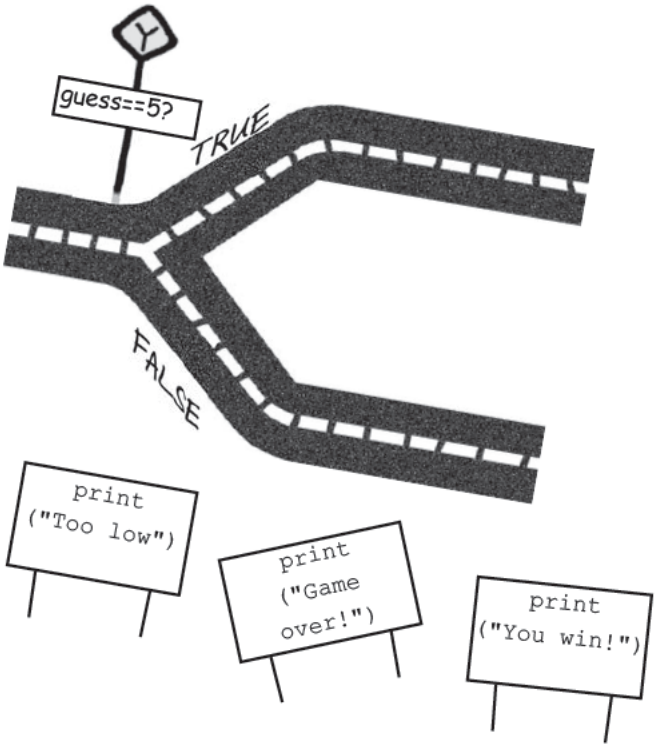
Autódromo
Kit de Construção

A contagem regressiva é iniciada no Grand Prix de Codeville. Os carros chegaram, estão aquecendo seus pneus no grid e a corrida está para começar. Você pode montar a pista para que ela exiba a mensagem de retorno certa? Note que você pode não precisar de todas as partes da pista.

Número fornecido	Mensagem de retorno
3	`Too low`
5	`You win!`
7	`Too high`
8	`Too high`

A linha de partida da corrida é fixada aqui.

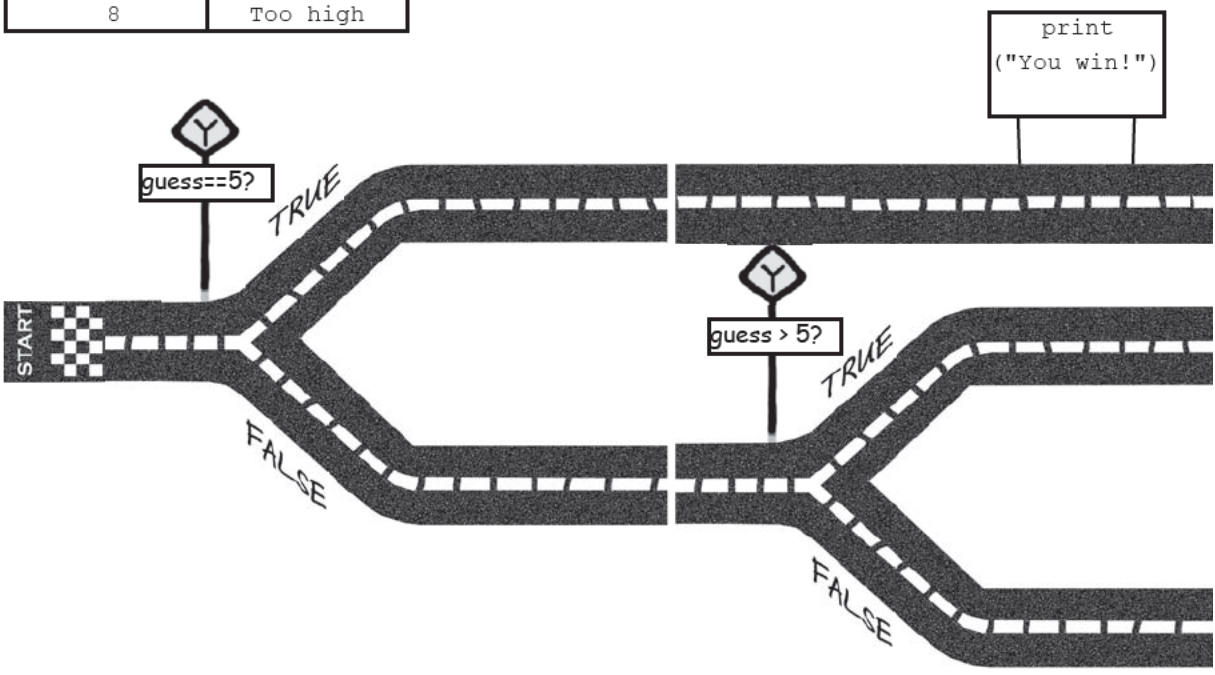


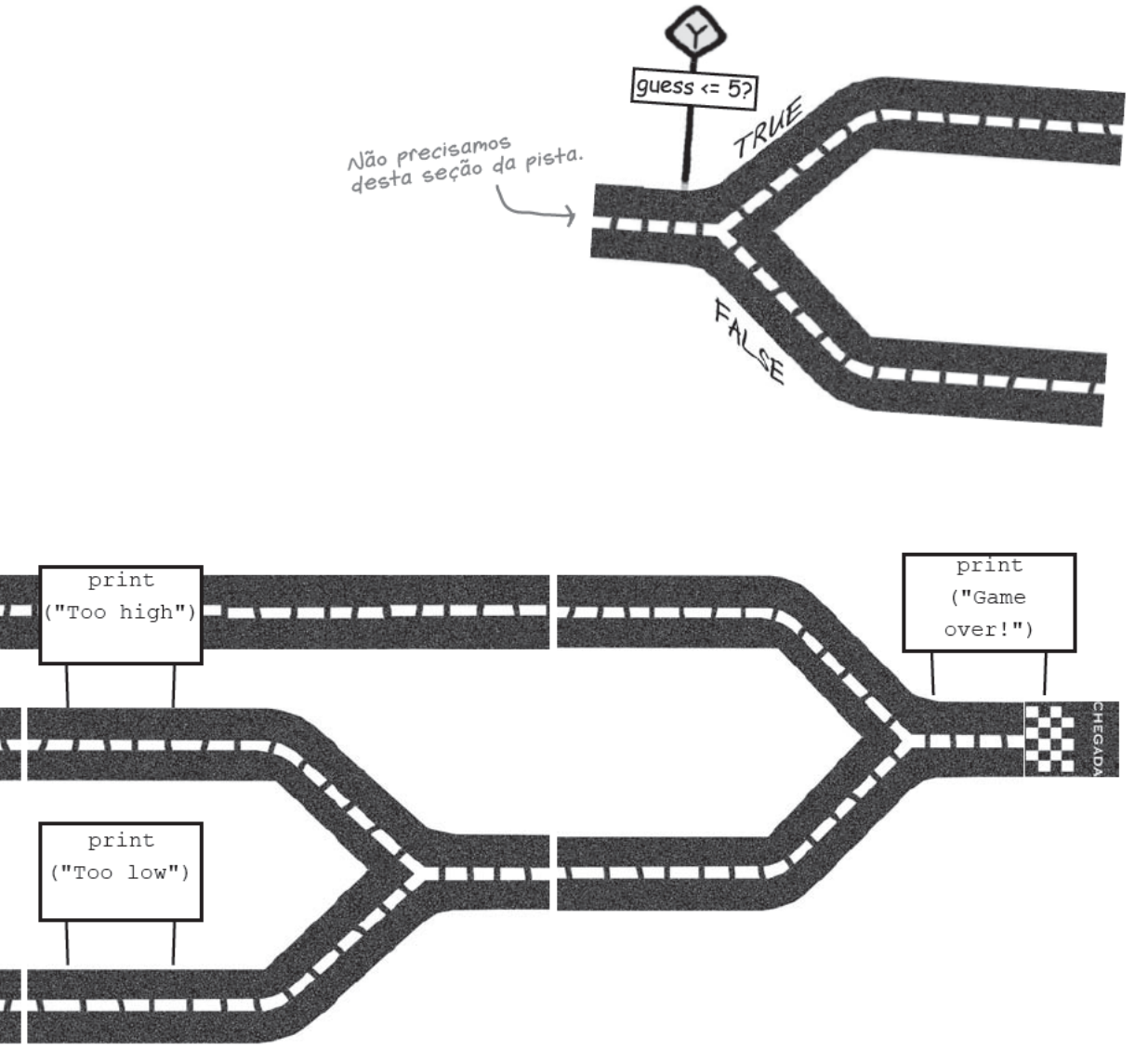

guess==5?
TRUE
FALSE
print ("Too low")
print ("Game over!")
print ("You win!")
CHEGADA
A linha de chegada da corrida é fixada aqui.
guess > 5?
TRUE
FALSE

Autódromo
Solução do kit de Construção

A contagem regressiva é iniciada no Grand Prix de Codeville. Os carros chegaram, estão aquecendo seus pneus no grid e a corrida está para começar. Você foi capaz de montar a pista para que ela exiba a mensagem de retorno certa?

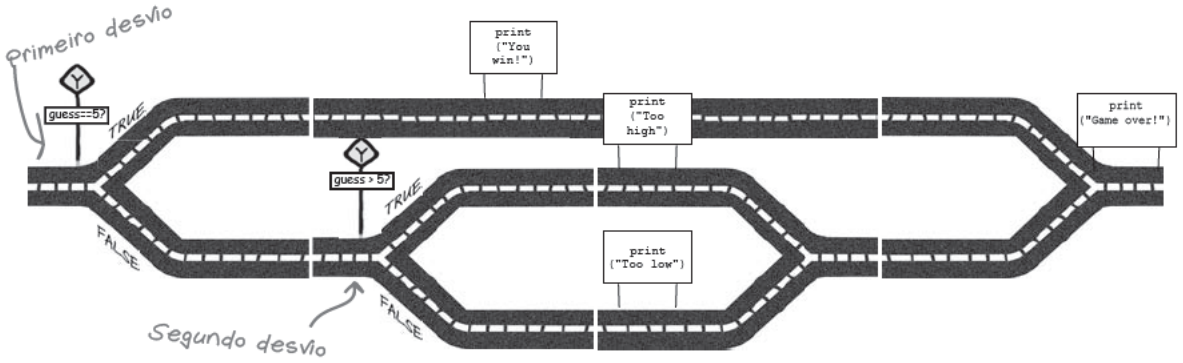
Número fornecido	Mensagem de retorno
3	Too low
5	You win!
7	Too high
8	Too high




guess <= 5?
TRUE
FALSE
Não precisamos desta seção da pista.
print ("Too high")
print ("Too low")
print ("Game over!")
CHEGADA

O código Python precisa interconectar os caminhos

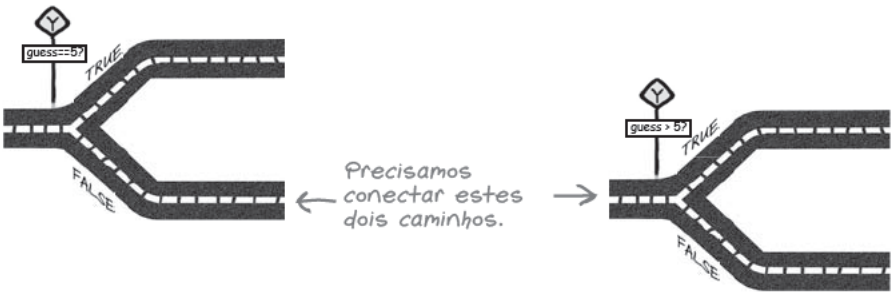
A solução está mapeada e, agora, sabemos que o código do programa precisará ter caminhos que coincidam com isto:



Mas não há um problema aqui? No projeto, há muitos caminhos de *interconexão*, mas até agora, escrevemos o código que contém apenas **um** desvio:

```
if guess == 5:
    print("You win!")
else:
    print("You lose!")
```

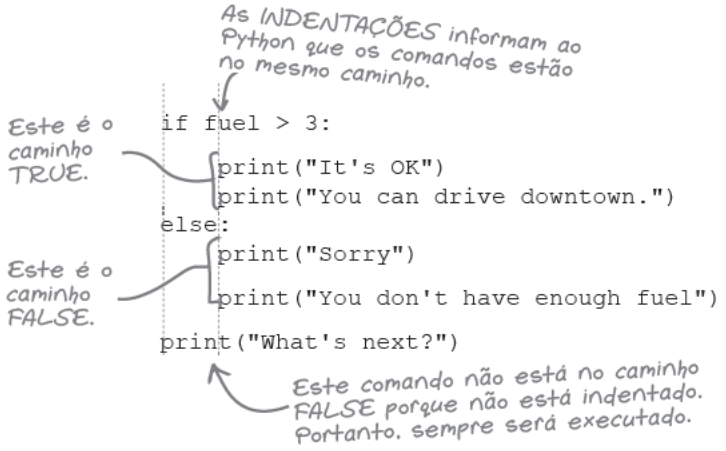
No novo código, precisaremos *conectar os dois desvios*. Precisamos que o segundo desvio apareça no **caminho false** do primeiro.



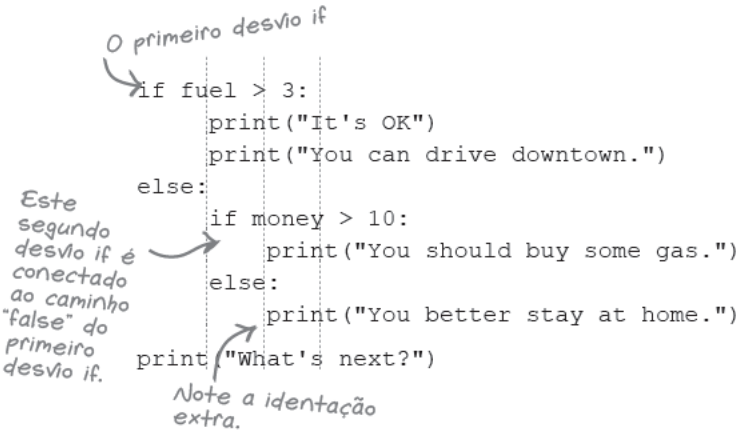
Portanto, como você conecta os desvios no Python?

O Python usa indentações para conectar os caminhos

O código dentro das instruções `if` e `else` é **indentado**. Não é apenas para tornar o código bonito. No Python, as **indentações importam**. Consideremos uma parte diferente do código de exemplo: algo que decidirá se você pode dirigir no centro da cidade. O Python usa identaçõess para conectar uma sequência de comandos para formar caminhos.



Então, como você conecta os desvios? Simplesmente, você indenta (ou recua) o segundo desvio em **mais um nível**.



Veja bem!

As indentações importam no Python.

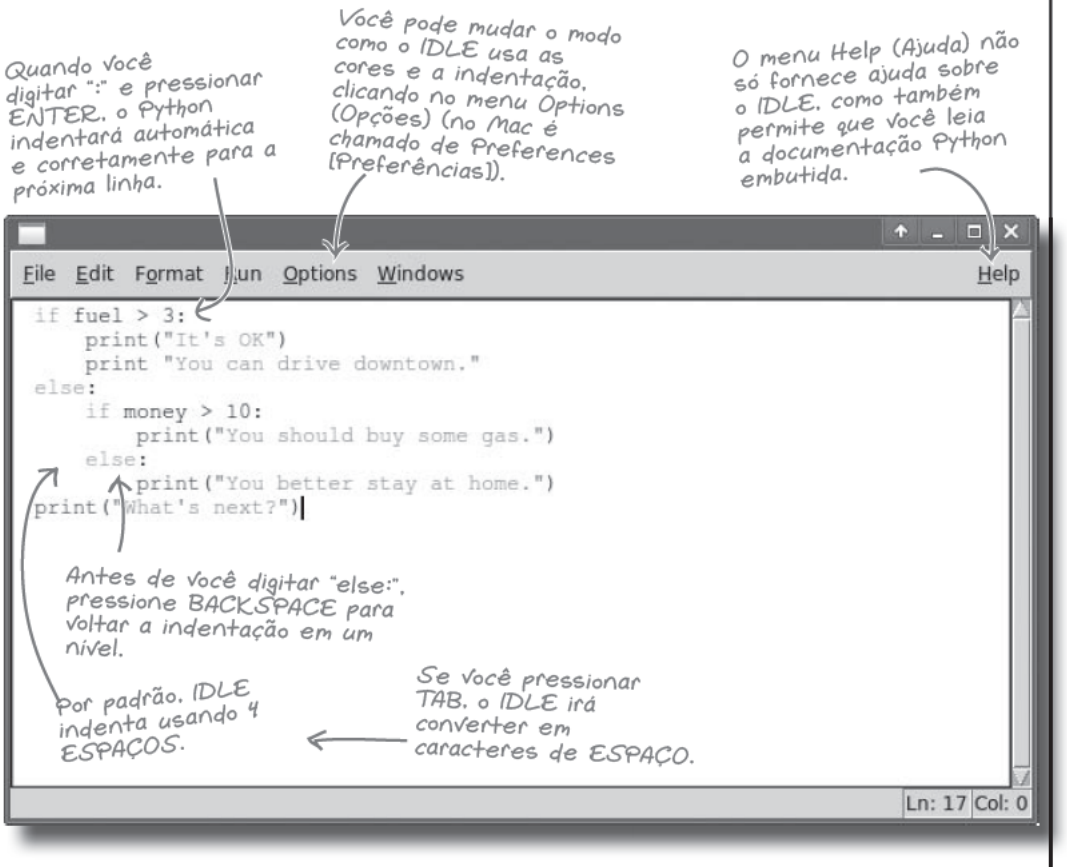
Tenha cuidado com o modo como você indenta o código no Python; se você não o fizer corretamente, seu código poderá fazer algo bem diferente do que você esperava.

Agora, você deve ter informações suficientes para corrigir o código, mas antes de fazermos isso, vejamos como o *IDLE* ajuda a **indentar o código**.

IDLE... de relance

Você estará usando o **IDLE** para fornecer todo o código Python neste livro; portanto, vale a pena passar um pouco de tempo familiarizando-se com alguns de seus recursos.

Mesmo que o IDLE pareça um editor simples, realmente está **cheio** de toques inteligentes que facilitam muito a programação Python e a agilizam para você. Vale a pena passar um tempo explorando os menus do IDLE e o sistema de ajuda; mas agora, eis algumas *sugestões úteis* para ajudá-lo a se sentir em casa.



Quebra-cabeças da Piscina

Sua **tarefa** é pegar os fragmentos de código do Python na piscina e colocá-los nas linhas em branco do jogo. Você **não** pode usar o mesmo fragmento de código mais de uma vez e não precisará usar todos os fragmentos de código. Seu **objetivo** é completar o programa de adivinhação.

Sugestão: Não esqueça de indentar.

```
print("Welcome!")
g = input("Guess the number: ")
guess = int(g)
```

.....

.....

.....

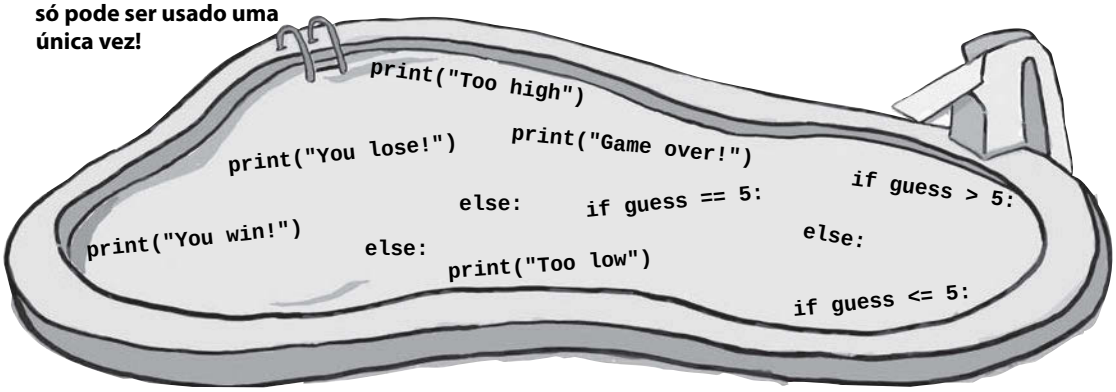
.....

.....

.....

.....

Nota: cada fragmento de código da piscina só pode ser usado uma única vez!



Solução do Quebra-cabeça da Piscina

Sua **tarefa** era pegar os fragmentos de código do Python na piscina e colocá-los nas linhas em branco no jogo. Você **não** pôde usar o mesmo fragmento de código mais de uma vez e não precisou usar todos os fragmentos. Seu **objetivo** era completar o programa de adivinhação.

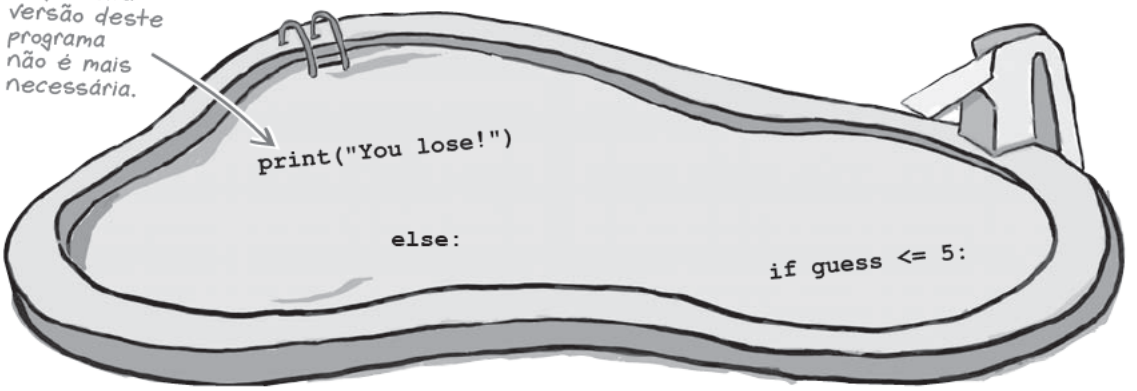
Sugestão: Não esqueça de indentar.

```
print("Welcome!")
g = input("Guess the number: ")
guess = int(g)
if guess == 5:
    print("You win!")
else:
    if guess > 5:
        print("Too high")
    else:
        print("Too low")
print("Game over!")
```

Você lembrou de indentar seu código o bastante?

Todo este código é indentado sob a parte else da parte if original.

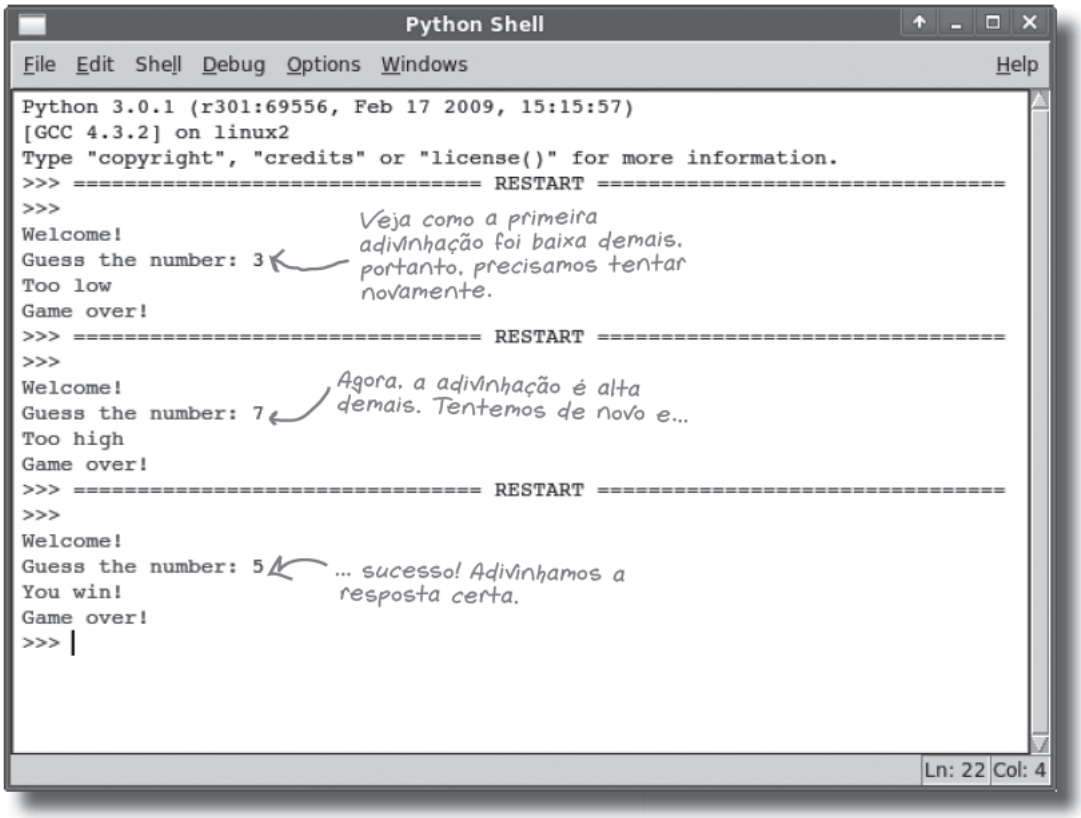
Esta parte do código da primeira versão deste programa não é mais necessária.



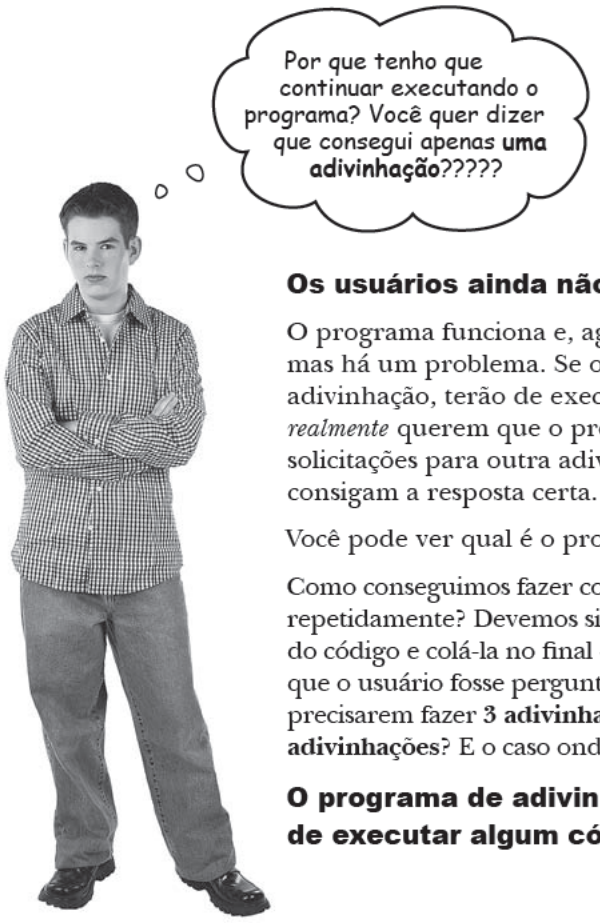
Test Drive

Então, o que acontecerá se você executar a nova versão do programa?

Façamos alguns testes. Lembre-se, você precisará mudar para a janela do programa para cada execução e escolher **`Run module`** (Executar módulo) no menu.



O programa funciona! Mas, os usuários estão mais contentes?



Os usuários ainda não gostam.

O programa funciona e, agora, gera um feedback extra, mas há um problema. Se os usuários quiserem fazer outra adivinhação, terão de executar o programa de novo. Eles *realmente* querem que o programa continue fazendo solicitações para outra adivinhação, até que finalmente consigam a resposta certa.

Você pode ver qual é o problema?

Como conseguimos fazer com que o computador faça algo repetidamente? Devemos simplesmente fazer uma cópia do código e colá-la no final do arquivo? Isso asseguraria que o usuário fosse perguntado duas vezes. Mas, e se eles precisarem fazer **3 adivinhações**? **4 adivinhações**? ou **10.000 adivinhações**? E o caso onde a adivinhação está correta?

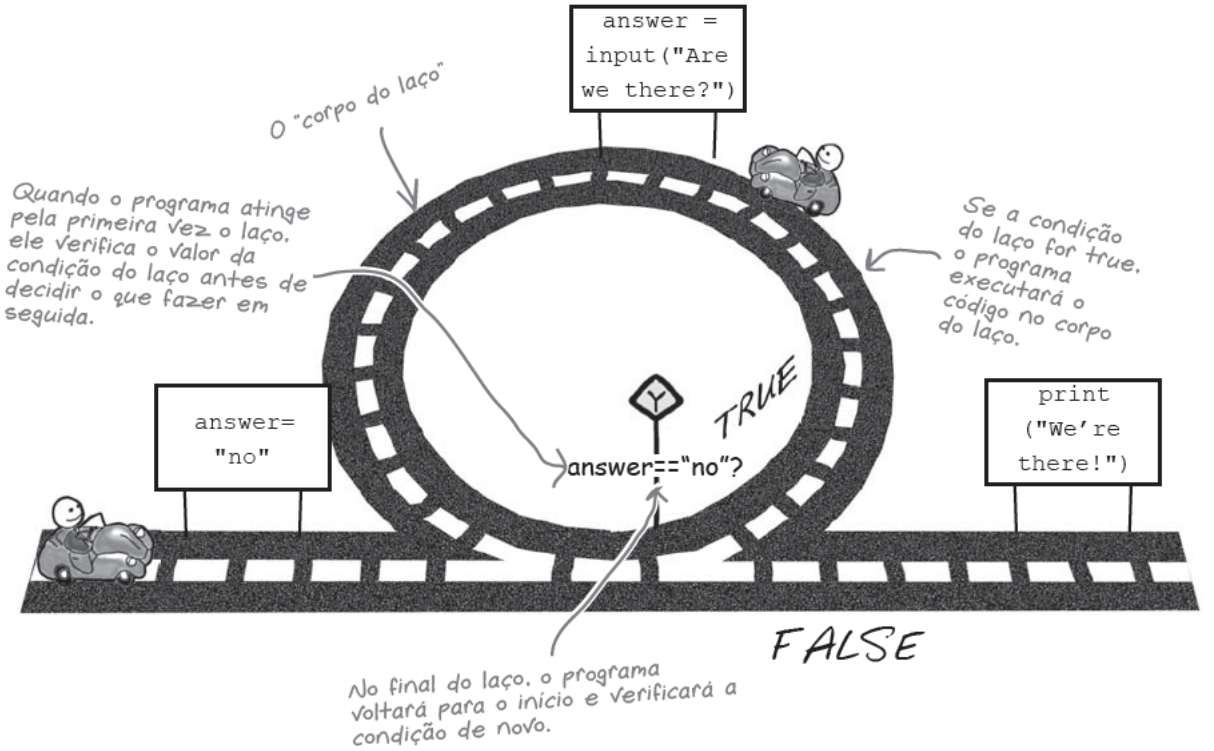
O programa de adivinhação precisa ser capaz de executar algum código repetidamente.


Não seria um sonho se houvesse um modo de fazer uma parte do código ser executada várias vezes? Mas, acho que é apenas uma fantasia...

Os laços (em inglês, loop) permitem que você execute a mesma parte do código sempre

Os programas geralmente precisam continuar a execução de alguma parte do código muitas vezes. Além dos desvios, as linguagens de programação também fornecem **laços**.

Os laços são um pouco parecidos com os desvios. Como os desvios, os laços têm uma condição (a **condição do laço**) que é verdadeiro ou falso. E mais, como a parte `if` dos desvios, se a condição do laço for true, então um laço executará uma dada parte do código. Para um desvio, esse código é chamado de **corpo**. Para um loop, é chamado de **corpo do laço**.



A grande diferença entre um laço e um desvio é **quantas vezes** ele executa o código associado. Um desvio executará seu código apenas uma vez. Mas um laço, executará o corpo do laço, então, verificará a condição do laço de novo e, se ainda for true, executará novamente o corpo do laço. Mais uma vez. Mais uma vez. Na verdade, continuará a executar o corpo do laço até que a condição do laço seja **false**.

O laço while do Python

As linguagens de programação têm modos diferentes de criar laços, mas um dos modos mais simples no Python é usar um **laço while**. Eis um exemplo:

Queremos assegurar que o laço seja executado pela primeira vez.

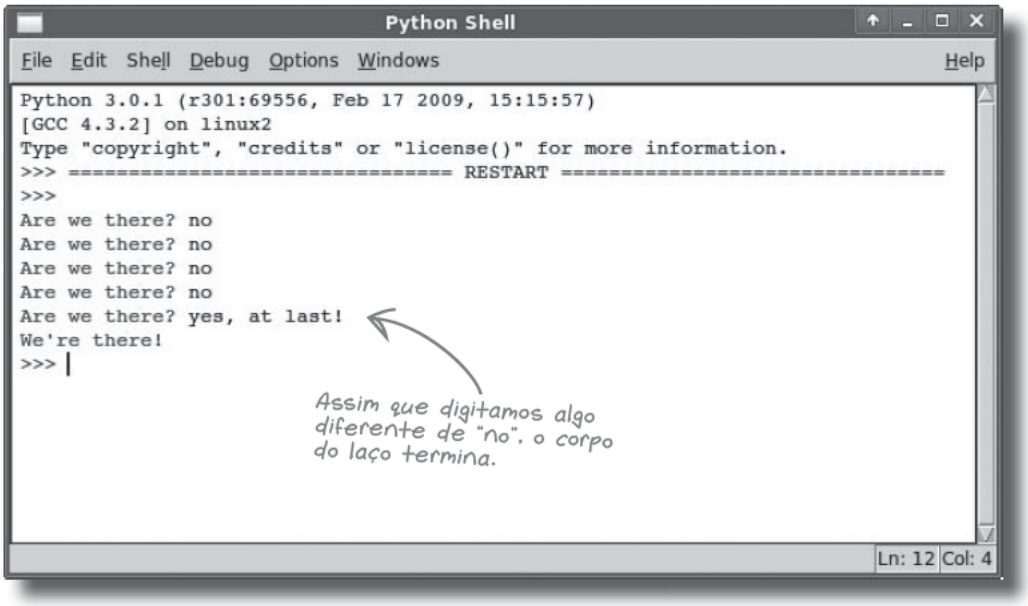
A condição do laço

O corpo do laço é o código indentado depois da linha "while".

```
answer = "no"
while answer == "no":
    answer = input("Are we there? ")
print("We're there!")
```

O corpo do laço é apenas uma linha de código neste exemplo, mas o corpo do laço pode ter muitas linhas de código. Pode até incluir desvios e outros laços.

O laço é assim quando você o escreve como um laço while do Python. O código continua fazendo a pergunta: "Nós já chegamos?", até que o usuário digite algo diferente de `no`. Fica assim quando executado:



Você notou que teve que definir o valor da variável `answer` para algo lógico antes de iniciar o laço? Isso é importante, pois se a variável `answer` não tivesse o valor `no`, a condição do laço teria sido false e o código no corpo do laço **nunca** teria sido executado.

Lembre-se disso. Pode ser útil no próximo exercício...

Exercício Longo

Agora, é hora de aplicar seu amuleto de programação. Tenha cuidado: este exercício é um pouco capcioso. Você precisa reescrever seu programa de jogo para que ele continue a execução até o usuário adivinhar a resposta certa. Você precisa usar todas as coisas que aprendeu neste capítulo. Precisa, também, desenvolver condições para cada um dos desvios e laços requeridos.

Lembre-se: o programa precisa continuar perguntando ao usuário uma resposta enquanto a adivinhação atual está errada.

Sugestão: *Se você precisar testar se duas coisas têm valores diferentes, use o operador* **!=**.

Este é o operador "diferente de".

Escreva seu código aqui.

Código Pronto

Se você adicionar estas duas linhas de código ao início de seu programa:

```
from random import randint
secret = randint(1, 10)
```

A variável `secret` *será definida para um número aleatório entre 1 e 10. Modifique seu programa em relação à página de abertura para que, ao invés da resposta sempre ser 5, ele use um número aleatório de 1 a 10 como a resposta.*

Escreva a próxima versão de seu programa aqui. Essa versão de seu programa usa o valor da variável "secret" como a resposta certa.

Exercício Longo Solução

Você precisava reescrever seu programa de jogo para que ele continuasse a execução até que o usuário adivinhasse a resposta certa. Você precisava usar todas as coisas que aprendeu neste capítulo. Precisava desenvolver condições para cada um dos desvios e laços requeridos.

Sugestão: *Se você precisar testar se duas coisas têm valores diferentes, use o operador* **!=**.

Não se preocupe se seu código não parecer exatamente com este. O importante é que ele funcione do mesmo modo quando executado.

Você lembrou de definir a adivinhação para algum valor padrão lógico, para assegurar que o laço fosse executado pela primeira vez?

```
print("Welcome!")
guess = 0
while guess != 5:
    g = input("Guess the number: ")
    guess = int(g)
    if guess == 5:
        print("You win!")
    else:
        if guess > 5:
            print("Too high")
        else:
            print("Too low")
print("Game over!")
```

Precisamos continuar executando enquanto a adivinhação está errada.

Todo este código é recuado, significando que tudo está dentro do corpo do laço.

Esta parte do programa é muito parecida com o que você tinha antes.

Código Pronto

Se você adicionar estas duas linhas de código ao n de seu programa:

```
from random import randint
secret = randint(1, 10)
```

A variável `secret` *será definida para um número aleatório entre 1 e 10. Você deveria modificar seu programa em relação à página de abertura para que, ao invés da resposta sempre ser 5, ele use um número aleatório de 1 a 10 como a resposta.*

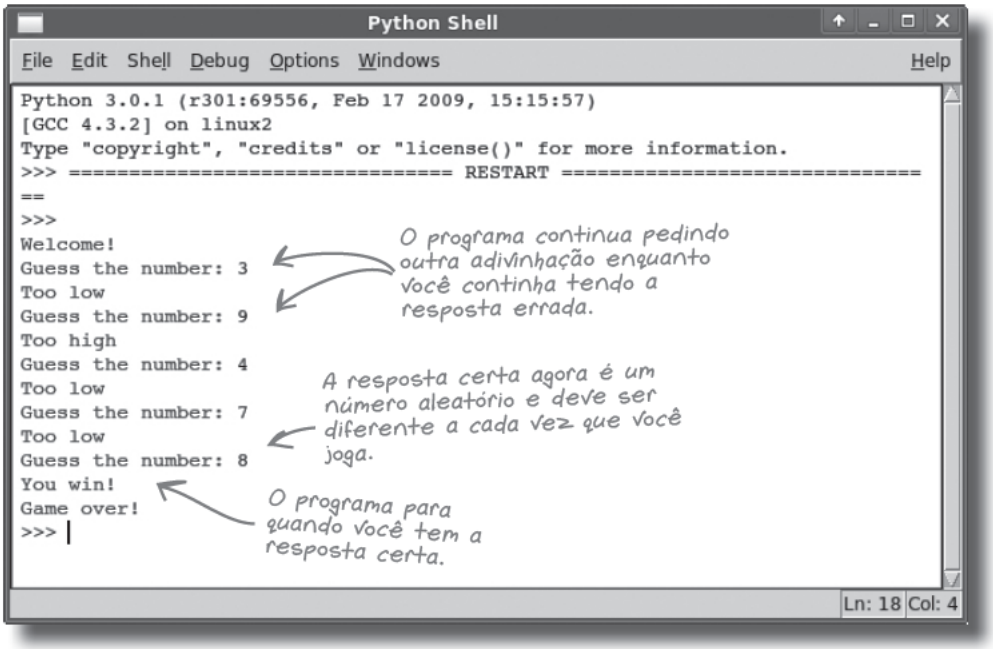
Aqui estão as duas linhas que criam o número aleatório.

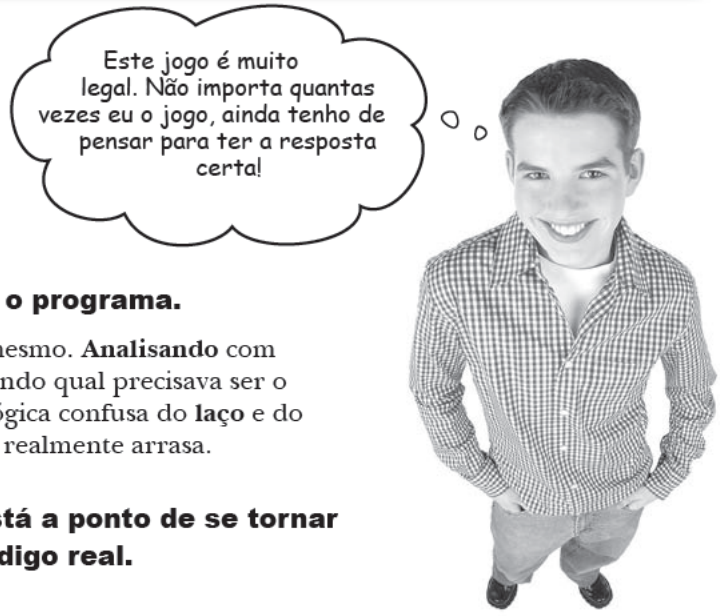
Agora, ao invés de verificarmos se a resposta é 5, verificamos novamente com o número aleatório, que é mantido na variável "secret".

```
from random import randint
secret = randint(1, 10)
print("Welcome!")
guess = 0
while guess != secret:
    g = input("Guess the number: ")
    guess = int(g)
    if guess == secret:
        print("You win!")
    else:
        if guess > secret:
            print("Too high")
        else:
            print("Too low")
print("Game over!")
```

TEST DRIVE

Então, o que acontece quando você executa a nova versão de seu programa?





Seus usuários adoram o programa.

E você pode criá-lo por si mesmo. **Analisando** com cuidado o problema, decidindo qual precisava ser o feddback e trabalhando a lógica confusa do **laço** e do **desvio**, você criou algo que realmente arrasa.

Bom trabalho. Você está a ponto de se tornar um manipulador de código real.

Sua Caixa de Ferramentas de Programação

Você colocou o Capítulo 1 em seu currículo. Vejamos de novo o que você aprendeu até então.

Ferramentas de Programação

* Os programas são criados a partir de códigos de instruções:
  - os comandos fazem coisas.
  - os desvios decidem as coisas.
  - os laços repetem as coisas.
* As condições ajudam você a decidir se algo é True ou False.
* A atribuição define um nome para um valor.
* Um valor nomeado é armazenado em uma "variável".

Ferramentas do Python

* desvios if/else
* laços while
* operador de atribuição =
* operador de igualdade ==
* operador diferente de !=
* operador maior que >
* print() exibe uma mensagem na tela
* input() obtém e retorna a entrada do usuário
* int() converte caracteres em números
* randint() produz um número aleatório