Transcrição do curso de Python 3 – parte 1

Olá, bem vindo à Alura, ao nosso primeiro curso sobre Python na versão 3. Nesse curso introdutório, vamos implementar um jogo para começar a programar com Python. É um jogo de adivinhação, onde o computador vai escolher um número e o usuário precisa adivinhá-lo.

Vamos aprender a trabalhar com variáveis de diversos tipos, como gerar um número aleatório, como tomar decisões através de **if else**, pois o computador dará uma dica para nós, se o número é menor ou maior. O jogo tem várias rodadas, então existe a necessidade de repetir uma parte do código. É preciso ler a entrada do usuário e mostrar o resultado para o jogador.

Enfim, nesse curso teremos vários desafios a resolver e eu convido você a me acompanhar nessas próximas aulas, para aprender essa linguagem fantástica e se divertir muito com Python 3.

Nesse vídeo veremos como instalar o Python no Windows. O primeiro passo é acessar o site do Python: <https://www.python.org/>. Na sessão de Downloads, automaticamente já será disponibilizado o instalador específico do Windows, portanto é só baixar o **Python 3**, na sua versão mais atual.

**Instalando o Python**

Após o download ser finalizado, abra-o e logo na primeira tela é importante marcar a *checkbox* ***Add Python 3.5 to PATH***. Essa opção é importante para conseguirmos executar o Python dentro do Prompt de Comando do Windows.

Vamos selecionar a instalação customizada somente para vermos a instalação com mais detalhes. Na segunda tela, podemos clicar em ***Next***. Já na terceira tela, também vamos deixar como está, mas vamos nos atentar ao **diretório de instalação do Python**, para caso queiramos procurar o executável, ou algo que envolva o seu diretório. Visto isso, podemos clicar ***Install*** e aguardar o término da instalação.

**Testando o Python**

Terminada a instalação, podemos testar se o Python foi instalado corretamente. Podemos abrir o Prompt de Comando e executar o seguinte comando:

python -V

Esse comando imprime a versão do Python instalada no Windows. Se a versão for impressa, significa que o Python foi instalado corretamente. Logo, se rodarmos o comando **python**, teremos acesso ao Prompt do próprio Python, conseguindo assim utilizá-lo.

**Outras formas de utilizar o Python**

Podemos rodar o Python diretamente do seu próprio Prompt. Podemos procurar pelo Python na caixa de pesquisa do Windows e abri-lo, assim o seu console próprio será aberto.

Uma outra forma é abrir a **IDLE do Python**, que se parece muito com o console, mas com um menu, que possui algumas opções extras

Os sistemas operacionais baseados no Debian já possuem o Python 3 pré-instalado, mas o comandos para instalá-lo pelo terminal é:

sudo apt-get update

sudo apt-get install python3

Assim como no Windows, você pode verificar se o Python 3 está instalado executando o seguinte comando:

python3 -V

E para executar o Python 3, basta rodar o comando **python3** no terminal.

## Instalando o Python no MacOS

Para instalar o Python 3 no MacOS, temos duas opções, através do [Homebrew](http://brew.sh/index_pt-br.html), fazemos:

brew update

brew install python3

Mas caso você não utiliza o Homebrew, podemos baixar o instalador do Python através do seu [site oficial](https://www.python.org/). Assim como no Windows, na sessão de Downloads, o site automaticamente já detectará o seu sistema operacional e disponibilizará o instalador específico para o seu MacOS, portanto é só baixar o Python 3, na sua versão mais atual.

E assim como nos sistemas baseados no Debian, verificamos a versão do Python com o seguinte comando:

python3 -V

E o executamos rodando o comando **python3** no terminal.

Também conseguimos utilizar o Python sem instalá-lo na nossa máquina, executando-o através de um serviço na web. Há vários sites que disponibilizam esse serviço, entre eles o [**repl.it**](https://repl.it/). Nele podemos programar em várias linguagens, para começar basta clicar em **Start coding now!** e escolher o **Python3**.

À esquerda escrevemos o código em Python e à direita é a saída. Podemos testar esse serviço, imprimindo uma mensagem:

print("ola aluno! bem vindo!!")

Ao clicar em **run**, a mensagem é impressa à direita.

É uma boa alternativa caso não tenhamos o Python instalado localmente, mas é bom ter em mente que em algum momento a instalação local será necessária :)

Antes de criarmos um projeto, veremos como trabalhar no console do Python.

**A função print**

Já vimos no vídeo anterior como imprimir uma mensagem:

>>> print("ola mundo")

ola mundo

Vamos analisar mais a fundo essa função. No console do Python temos uma função de ajuda, a função **help()**:

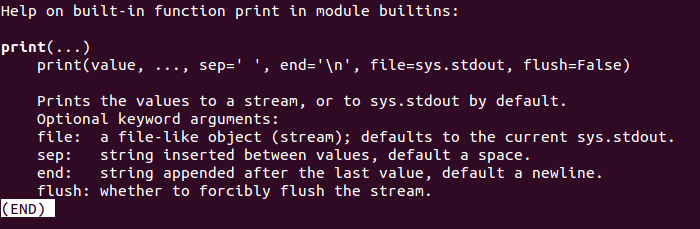
>>> help()

Podemos reparar que um novo console aparece:

help>

Agora, como queremos saber mais sobre a função **print**, vamos digitá-la:

help(print)



Inicialmente, o que nos importa são os três primeiros valores que a função **print** pode receber:

* **value** é o valor que queremos imprimir, as reticências indicam que a função pode receber mais de um valor, basta separá-los por vírgula.
* **sep** é o separador entre os valores, por padrão o separador é um espaço em branco.
* **end** é o que acontecerá ao final da função, por padrão há uma quebra de linha, uma nova linha (**\n**).

Podemos apertar a tecla **Q** para sair da documentação da função e **CTRL + C** ou **CTRL + D** para sair do console de ajuda do Python. De volta ao console do próprio Python, podemos testar a função **print** com os valores que vimos:

>>> print("Brasil", "ganhou", 5, "titulos mundiais", sep="-")

Brasil-ganhou-5-titulos mundiais

Como modificamos o separador, agora os valores são separador por um hífen. Vamos testar o **end** agora, não passando nada para ele:

>>> print("Brasil", "ganhou", 5, "titulos mundiais", end="")

Brasil ganhou 5 titulos mundiais>>>

Uma nova linha não é criada, ou seja, o que colocarmos dentro do **end** será impresso ao final da função.

**Variáveis**

Agora queremos flexibilizar a função, queremos poder imprimir outros países, como Itália, Alemanha, Argentina... Para isso teremos que mudar o nome do país e o número de títulos conquistados. Então vamos **definir o valor com o nome do país fora da função print**. Vamos definir uma **variável** para o nome do país, já que o seu valor pode variar:

>>> pais = "Italia"

Definimos uma variável e atribuímos a ela um valor. Assim como fizemos com o nome do país, vamos fazer também com a quantidade de títulos:

>>> pais = "Italia"

>>> quantidade = 4

Com as variáveis definidas, podemos refazer a função print, só que dessa vez passando as variáveis no lugar dos antigos valores:

>>> pais = "Italia"

>>> quantidade = 4

>>> print(pais, "ganhou", quantidade, "titulos mundiais")

Italia ganhou 4 titulos mundiais

Agora a mensagem é impressa no mesmo molde da anterior, só que dessa vez com variáveis! Mas qual é o tipo dessas variáveis? O tipo da variável depende do valor que passarmos para ela. Podemos "perguntar" para a variável qual é o seu tipo, passando-a para a função **type**:

>>> type(pais)

<class 'str'>

>>> type(quantidade)

<class 'int'>

O valor **str** significa que a variável é do tipo **string**, já que o seu valor está entra aspas duplas. E **int** significa que a variável é do tipo **inteiro**, já que passamos um valor inteiro para a variável.

Veremos mais sobre os tipos das variáveis no próximo vídeos, até lá!

Exercício proposto:

Para representar uma data, temos as variáveis **dia**, **mes** e **ano**:

>>> dia = 15

>>> mes = 10

>>> ano = 2015

Sem alterar as variáveis e sem passar nenhuma string adicional à função **print()**, como faríamos para ter como resultado a impressão, de uma data formatada:

15/10/2015

Para ver a resposta do instrutor, basta clicar em **Continuar**.

Podemos alterar o separador entre os valores que a função **print()** recebe, utilizando o parâmetro **sep**, que por padrão é um espaço em branco. Basta utilizá-lo, dizendo que seu valor será uma barra (**/**):

>>> dia = 15

>>> mes = 10

>>> ano = 2015

>>> print(dia, mes, ano, sep="/")

15/10/2015

Ainda no console do Python, vimos no vídeo anterior que uma variável sempre terá um tipo associado:

>>> pais = "Brasil"

>>> type(pais)

<class 'str'>

Mas em nenhum local definimos explicitamente que a variável **pais** receberá valores do tipo string. Talvez você já tenha visto isso em outras linguagens, como C, C++, Java, em que definimos o tipo da variável na hora da sua declaração, algo como:

>>> str pais = "Brasil"

Mas isso em Python **não funciona**. Ou seja, no mundo Python não somos obrigados a definir explicitamente o tipo da variável. Podemos até passar outros tipos de valores para a variável:

>>> pais = "Brasil"

>>> type(pais)

<class 'str'>

>>> pais = 644

>>> type(pais)

<class 'int'>

Além de funcionar, o tipo da variável também muda! O tipo da variável mudou dinamicamente, de acordo com o valor que é atribuído a ela, logo, o tipo da variável é definido de acordo com o valor que ela guarda, isso faz parte da **tipagem dinâmica** do Python.

Agora temos tudo para começar o nosso projeto no próximo capítulo!

No capítulo anterior fizemos os primeiros passos com o Python, desde a sua instalação e até vimos um pouco da sua sintaxe no console. Mas para escrever uma aplicação completa, utilizando o console, não parece ser uma boa ideia. Podemos ter um editor de texto que nos auxilie nessa programação, nos permitindo trabalhar com vários arquivos, auxiliando a nossa vida.

Há várias opções de editores de texto no mercado, entre elas o [Atom](https://atom.io/) e o [Sublime Text](https://www.sublimetext.com/). Apesar de esses editores nos ajudarem a escrever o código, eles não são focados no Python, e sim em dar suporte a várias linguagens. Então, vamos utilizar uma ferramenta (**IDE**, do inglês ***I***ntegrated ***D***evelopment ***E***nvironment) só focada para o Python, assim como existe o Eclipse para o Java.

## Instalando o PyCharm

Uma IDE que é voltada exclusivamente para o Python é o [**PyCharm**](https://www.jetbrains.com/pycharm/), e é ela que iremos utilizar aqui no treinamento. A sua instalação é bem simples, basta acessar a sessão de [**Download**](https://www.jetbrains.com/pycharm/download/) do site oficial, baixar e instalar a versão **Community** da IDE, já que a versão **Professional** é paga.

## Conhecendo o PyCharm e criando o primeiro projeto

Após instalar o PyCharm, vamos abri-lo e criar o nosso primeiro projeto, clicando em **Create New Project**. Na tela que irá se abrir, nos é perguntado a localização do projeto e a versão do Python. Vamos criar o projeto **jogos**, dentro da pasta **PycharmProjects** mesmo, e nos atentar à versão do Python (caso você tenha mais de uma versão instalada) ela deve ser a **versão 3**.

O projeto ficará sendo exibido na esquerda, e para criar o primeiro arquivo Python dentro dele, basta clicar com o botão direito do mouse em cima dele e clicar em **New -> Python File**, vamos colocar o nome do arquivo de **adivinhacao.py**.

A fim de testes, vamos imprimir uma mensagem simples:

print("Bem vindo ao jogo de Adivinhação!")

Para executar o arquivo, clicamos em **Run -> Run...** no menu superior, e selecionamos o arquivo que acabamos de criar. O console do PyCharm é aberto e exibe a saída do nosso programa, que é a mensagem **Bem vindo ao jogo de Adivinhação!**.

Em uma só tela, conseguimos ver os arquivos do projeto, o seu código fonte e o console, que exibe a saída do programa que for executado. Há vários outros recursos que ainda veremos mais à frente, mas o primeiro passo foi realizado!

A ideia do nosso jogo é termos que acertar um número secreto. Quando o programa estiver rodando, teremos que digitar um número e o programa dirá se acertamos ou erramos o número, com várias tentativas e níveis.

Vamos começar definindo esse número secreto (mais à frente vamos ver como gerar um número aleatório):

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Bem vindo ao jogo de Adivinhação!")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

numero\_secreto = 42

## Capturando a entrada do usuário

Agora, para que o usuário possa digitar o número, vamos utilizar a função **input**, ela trava o programa até que o usuário digite algo e tecle ENTER. Ela recebe por parâmetro a mensagem que será exibida no console e nos retorna o que o usuário digitou, logo vamos guardar esse resultado em uma variável, que chamaremos de **chute**:

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Bem vindo ao jogo de Adivinhação!")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

numero\_secreto = 42

chute = input("Digite o seu número: ")

Para testar, vamos ao final do programa imprimir o conteúdo da variável chute, para mostrar realmente que o seu conteúdo será o que o usuário digitou:

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Bem vindo ao jogo de Adivinhação!")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

numero\_secreto = 42

chute = input("Digite o seu número: ")

print("Você digitou: ", chute)

Podemos rodar o programa e ver que realmente é impresso o valor que digitarmos.

## Comparando valores

Agora que conseguimos capturar o que o usuário digitou, precisamos comparar esse valor com o número secreto, para poder dizer ao usuário se ele digitou o número correto ou não. Bom, já sabemos o número secreto que o chute do usuário, então vamos comparar os dois, algo como:

se numero\_secreto igual chute

print("Você acertou!")

senão

print("Você errou!")

Só que as palavras **se**, **senão** e **igual** não funcionam no mundo Python, temos que respeitar a sua sintaxe. O **se** em Python é **if**, o **igual** é a comparação **==** e o **senão** é **else**. Então, resumindo a sintaxe do Python é:

if (condição):

executa código caso a condição seja verdadeira

else:

executa código caso a condição seja falsa

Mas precisamos prestar atenção a alguns detalhes. É uma recomendação que a condição fique dentro de parênteses (apesar de também funcionar sem); para marcar o fim da instrução e início de um bloco (o código que será executado caso a condição seja verdadeira ou falsa), é utilizado dois pontos (**:**), e esse bloco obrigatoriamente deve estar 4 espaços (ou um TAB) mais à direita. Então o código ficará assim:

if (numero\_secreto == chute):

print("Você acertou!")

else:

print("Você errou!")

Podemos rodar o programa e verificar que mesmo se digitarmos o número certo, recebemos a mensagem **Você errou**. Porque?

## Convertendo uma string para número inteiro

Isso acontece porque a função **input** nos retorna uma string, pois qualquer coisa pode ser digitada, não é garantido que o usuário irá digitar um número. Como não há essa garantia, o retorno é uma string.

Já a variável **numero\_secreto** é um número! Logo, do tipo inteiro. Então estamos testando a igualdade de um inteiro com uma string, logo essa comparação sempre será falsa, apesar da string representar um número inteiro. Para resolver isso precisamos **mudar o tipo da variável**, convertendo uma string em número inteiro.

Para isso, o Python possui a função **int**, que recebe um valor e o converte para inteiro, justamente o que queremos. Logo, vamos utilizá-la no nosso código:

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Bem vindo ao jogo de Adivinhação!")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

numero\_secreto = 42

chute\_str = input("Digite o seu número: ")

print("Você digitou: ", chute\_str)

chute = int(chute\_str)

if (numero\_secreto == chute):

print("Você acertou!")

else:

print("Você errou!")

Agora a comparação é feita corretamente! Para sair do bloco do **else**, basta escrevermos algo depois dele, sem a indentação de 4 espaços:

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Bem vindo ao jogo de Adivinhação!")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

numero\_secreto = 42

chute\_str = input("Digite o seu número: ")

print("Você digitou: ", chute\_str)

chute = int(chute\_str)

if (numero\_secreto == chute):

print("Você acertou!")

else:

print("Você errou!")

print("Fim do jogo")

Nesse vídeo veremos um pouco das diferenças entre o Python 3 e o Python 2, porque o Python 2 ainda é muito utilizado. Para esse vídeo, não é necessário instalar o Python 2, já que aqui só veremos as diferenças, a menos que você queira utilizá-lo.

## A função print

A primeira diferença que podemos ver é que no Python 2, não precisamos colocar os parênteses na função **print**, eles são opcionais:

>>> print "python2"

python2

>>> print("python2")

python2

Já no Python 3, os parênteses são obrigatórios. Ainda na função **print**, no Python 2 não há os parâmetros **sep** e **end**, ao contrário do Python 3, e quando a função recebe mais de um valor, sua saída é diferente:

>>> print("python", "2")

('python', '2')

## A função input

Outra diferença que podemos ver é na função **input**. Sabemos que no Python 3, essa função sempre retornará uma string. Já no Python 2, ela **automaticamente converte o tipo da variável**. Por exemplo:

>>> chute = input("Digite o seu número: ")

Digite o seu número: 42

>>> type(chute)

<type 'int'>

Isso foi considerado má prática, porque pode ou não ser a intenção do desenvolvedor converter o tipo da variável. Por isso é bem comum encontrar a função **raw\_input** sendo utilizada no Python 2:

>>> chute = raw\_input("Digite o seu número: ")

Digite o seu número: 42

>>> type(chute)

<type 'str'>

O retorno dessa função será sempre uma string, equivalente à função **input** do Python 3, mas ela **não existe** nessa versão.

Ao longo do treinamento veremos mais diferenças entre as versões!

Muito pode se falar na comparação das duas linguagens, mas para esse exercício vamos focar nas operações de adição e multiplicação. Vimos que o Python apenas soma valores de tipos numéricos. Ou seja, o exemplo seguir **não funciona** por causa do tipo **str**:

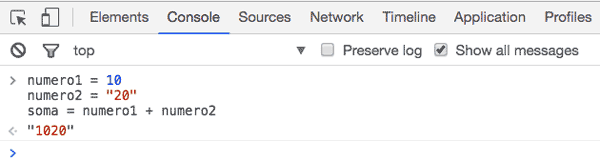
numero1 = 10

numero2 = "20"

soma = numero1 + numero2

TypeError: unsupported operand type(s) for +: 'int' and 'str'

Agora, o que acontece com o mesmo código no mundo JavaScript? Você pode testar isso facilmente dentro do seu navegador, apertando **F12** para abrir o seu console. Nele, digite o mesmo código:



Repare que o JavaScript concatena os valores, criando a string: "1020"

Você pode pensar que isso faz sentido, já que a variável **numero2** é do tipo string, no entanto o que o JavaScript faz é uma conversão implícita. O JavaScript converte a variável **numero1** automaticamente para string, e isso pode ser perigoso.

Vou tentar dar mais uma exemplo com JavaScript. Agora vamos multiplicar **numero1** com **numero2**:

numero1 = 10

numero2 = "20"

produto = numero1 \* numero2

Soma JavaScript

Repare que o JavaScript multiplicou e imprimiu 200! Agora também aconteceu uma conversão automática, mas dessa vez a variável **numero2** foi convertida para **int**. Novamente, o JavaScript é tolerante nesse aspecto e converte quando julga necessário. **O Python é mais rígido nesse sentido e não faz essas conversões implícitas.**

Ótimo, então vamos testar a multiplicação com Python:

numero1 = 10

numero2 = "20"

produto = numero1 \* numero2

print(produto)

O resultado nos surpreende:

20202020202020202020

Não deu erro e sim imprimiu 10 vezes 20! Mas eu não acabei de falar que o Python é rígido e não converte automaticamente?

Falei e na verdade não aconteceu uma conversão automática/implícita. Trata-se apenas de um **syntax sugar** do mundo Python. Um syntax sugar, docinho de sintaxe da linguagem, apenas simplifica algo que seria trabalhoso, mas não muda a característica da linguagem. Então, ao invés de escrever dez vezes o número 20, podemos simplificar e escrever **10 \* "20"**. Tudo bem?

No capítulo anterior começamos a implementar o jogo, vimos como capturar os dados digitados pelo usuário, como converter o valor e como fazer um **if** para saber se o usuário acertou ou não.

Nesse capítulo, vamos fazer com que o usuário possa dar vários chutes para tentar acertar o número, já que atualmente ele só tem uma tentativa. Mas antes disso, vamos implementar uma dica para o usuário, dizendo se o número que ele chutou é maior ou menor que o número secreto.

Para isso, precisamos mexer no bloco do **else**. Vamos ter que testar novamente, se o número for maior, imprimimos uma mensagem dizendo isso ao usuário, se for menos, diremos ao usuário que o número digitado é menor que o número secreto:

if (numero\_secreto == chute):

print("Você acertou!")

else:

if (chute > numero\_secreto):

print("Você errou! O seu chute foi maior que o número secreto.")

if (chute < numero\_secreto):

print("Você errou! O seu chute foi menor que o número secreto.")

Podemos testar e ver que tudo está funcionando perfeitamente.

## else com condição de entrada

Podemos notar que, se o chute não for igual, nem maior que o número secreto, obviamente ele será menor, então o último if não é necessário:

if (numero\_secreto == chute):

print("Você acertou!")

else:

if (chute > numero\_secreto):

print("Você errou! O seu chute foi maior que o número secreto.")

else:

print("Você errou! O seu chute foi menor que o número secreto.")

Mas para esses casos, podemos fazer um **else** com uma **condição de entrada**, o **elif**. Vamos utilizá-lo para deixar o código mais semântico, já que na prática não há diferença:

if (numero\_secreto == chute):

print("Você acertou!")

else:

if (chute > numero\_secreto):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (chute < numero\_secreto):

print("Você errou! O seu chute foi menor que o número secreto.")

## Melhorando a legibilidade do código

Podemos melhorar a legibilidade do nosso código, para que outros programadores que possam vir a desenvolver conosco o entendam melhor. Vamos deixar nossas condições mais claras, o que significa **chute == numero\_secreto**, por exemplo? Que o usuário acertou, logo vamos extrair essa condição para uma variável:

acertou = chute == numero\_secreto

if (acertou):

print("Você acertou!")

else:

if (chute > numero\_secreto):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (chute < numero\_secreto):

print("Você errou! O seu chute foi menor que o número secreto.")

Agora a condição **if** fica um pouco mais clara. Vamos fazer a mesma coisa para as outras duas condições:

acertou = chute == numero\_secreto

maior = chute > numero\_secreto

menor = chute < numero\_secreto

if (acertou):

print("Você acertou!")

else:

if (maior):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (menor):

print("Você errou! O seu chute foi menor que o número secreto.")

Podemos testar e ver que tudo continua funcionando como antes, mas agora com um código um pouco mais legível. No próximo vídeo implementaremos a chance do usuário poder dar vários chutes para tentar acertar o número secreto. Até lá!

Exercício proposto:

Henrique, mesmo dando os primeiros passos com a linguagem Python, decidiu criar um sistema de identificação de usuários. É claro que em uma aplicação real é necessário acessar banco de dados, entre outras coisas, mas usando o que ele já aprendeu, ele consegue algo parecido. Esse é o código do aluno:

usuario = input("Informe o usuário do sistema!")

if(usuario == "Flávio"):

print("Seja bem-vindo Flávio!")

else(usuario == "Douglas"):

print("Seja bem-vindo Douglas!")

else(usuario == "Nico"):

print("Seja bem-vindo Nico")

else:

print("Usuário não identificado!")

A ideia de Henrique é simples, porém não muito eficiente. Ele quer aceitar apenas os usuário Flávio, Douglas e Nico. No entanto, seu código não funciona!

Consegue identificar a razão? Quebre a cabeça um pouquinho.

Resposta instrutor:

O problema é que a instrução **else** não aceita receber uma condição. Nesse caso, para resolver o problema do código, precisamos trocar para a instrução **elif**:

usuario = input("Informe o usuário do sistema!")

if(usuario == "Flávio"):

print("Seja bem-vindo Flávio!")

elif(usuario == "Douglas"):

print("Seja bem-vindo Douglas!")

elif(usuario == "Nico"):

print("Seja bem-vindo Nico")

else:

print("Usuário não identificado!")

Veja que deixamos apenas um else que não recebe qualquer condição. Também tem que ser assim, porque se o usuário identificado não for nenhum dos que listamos, imprimimos na tela "Usuário não identificado".

Queremos dar mais de uma oportunidade para o usuário tentar acertar o número secreto, já que é um jogo de adivinhação. A primeira ideia é repetir o código, desde a função input até o bloco do elif. Ou seja, para cada nova tentativa que quisermos dar ao usuário, copiaríamos esse código novamente.

Só que copiar código sempre é uma má prática, queremos escrever o nosso código apenas uma vez, e **repeti-lo**. Se queremos repetir o código, faremos um **laço**, ou um **loop**. O laço que queremos fazer é:

enquanto ainda há tentativas:

chute\_str = input("Digite o seu número: ")

print("Você digitou: ", chute\_str)

chute = int(chute\_str)

acertou = numero\_secreto == chute

maior = chute > numero\_secreto

menor = chute < numero\_secreto

if (acertou):

print("Você acertou!")

else:

if (maior):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (menor):

print("Você errou! O seu chute foi menor que o número secreto.")

print("Fim do jogo")

Só que o Python não entende português, então vamos traduzi-lo. A palavra **tentativas** será uma variável, chamaremos-a de **total\_de\_tentativas**:

total\_de\_tentativas = 3

enquanto ainda há total\_de\_tentativas:

executa o código

A palavra **enquanto** no Python é o **while**, e assim como o **if**, ele recebe uma condição. A diferença é que o **if**, caso a condição seja verdadeira, executa apenas uma vez o código do seu bloco, já o **while** executa **enquanto** a condição for verdadeira:

total\_de\_tentativas = 3

while (ainda há total\_de\_tentativas):

executa o código

Resta agora a expressão **ainda há**. A ideia é que o usuário tenha 3 tentativas, representada no código pela variável **total\_de\_tentativas**. A cada rodada subtraímos **1** do valor dessa variável, até o valor chegar a **0**, que é quando devemos sair do **while**, logo vamos executá-lo enquanto a variável *total\_de\_tentativas* ***for maior que*** 0\*:

total\_de\_tentativas = 3

while (total\_de\_tentativas > 0):

chute\_str = input("Digite o seu número: ")

print("Você digitou: ", chute\_str)

chute = int(chute\_str)

acertou = numero\_secreto == chute

maior = chute > numero\_secreto

menor = chute < numero\_secreto

if (acertou):

print("Você acertou!")

else:

if (maior):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (menor):

print("Você errou! O seu chute foi menor que o número secreto.")

print("Fim do jogo")

A condição está perfeita, falta, dentro do laço, subtrairmos 1 da variável **total\_de\_tentativas**:

total\_de\_tentativas = 3

while (total\_de\_tentativas > 0):

chute\_str = input("Digite o seu número: ")

print("Você digitou: ", chute\_str)

chute = int(chute\_str)

acertou = numero\_secreto == chute

maior = chute > numero\_secreto

menor = chute < numero\_secreto

if (acertou):

print("Você acertou!")

else:

if (maior):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (menor):

print("Você errou! O seu chute foi menor que o número secreto.")

total\_de\_tentativas = total\_de\_tentativas - 1

print("Fim do jogo")

Testamos o código e ótimo, ele funciona! Mas pode ficar ainda melhor.

## Representando a rodada

Vamos imprimir para o usuário qual o número da rodada que ele está jogando, para deixar claro quantas tentativas ele tem. Para isso vamos criar a variável **rodada**, que começa com o valor **1**:

total\_de\_tentativas = 3

rodada = 1

E vamos imprimi-la antes do usuário digitar o seu chute:

total\_de\_tentativas = 3

rodada = 1

while (total\_de\_tentativas > 0):

print("Tentativa", rodada, "de", total\_de\_tentativas)

chute\_str = input("Digite o seu número: ")

# restante do código comentado

E para a variável **total\_de\_tentativas** continuar com o valor **3**, não vamos mais subtrair 1 do seu valor, e sim adicionar 1 ao valor da variável **rodada**:

total\_de\_tentativas = 3

rodada = 1

while (total\_de\_tentativas > 0):

print("Tentativa", rodada, "de", total\_de\_tentativas)

chute\_str = input("Digite o seu número: ")

print("Você digitou: ", chute\_str)

chute = int(chute\_str)

acertou = numero\_secreto == chute

maior = chute > numero\_secreto

menor = chute < numero\_secreto

if (acertou):

print("Você acertou!")

else:

if (maior):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (menor):

print("Você errou! O seu chute foi menor que o número secreto.")

rodada = rodada + 1

print("Fim do jogo")

Por fim, precisamos modificar a condição, como o **total\_de\_tentativas** permanecerá com o valor 3, o código precisa ficar executando enquanto o valor da rodada for menor ou igual ao total de tentativas:

total\_de\_tentativas = 3

rodada = 1

while (rodada <= total\_de\_tentativas):

print("Tentativa", rodada, "de", total\_de\_tentativas)

chute\_str = input("Digite o seu número: ")

# restante do código comentado

Agora conseguimos imprimir para o usuário quantas tentativas restantes ele possui!

Com a lógica de tentativas implementada, vamos focar na impressão do número de tentativas para o usuário. Atualmente ela está assim:

print("Tentativa", rodada, "de", total\_de\_tentativas)

Desse jeito a frase é impressa do jeito que queremos, mas tem uma forma mais elegante de imprimir essa frase. Podemos deixar a string toda no código, dizendo onde que ela eventualmente pode mudar, no nosso caso é nos números. Onde a string pode mudar, colocamos **chaves** (**{}**):

print("Tentativa {} de {}")

As chaves significam que o Python deve substituí-las pelos valores das variáveis, então vamos passá-las:

print("Tentativa {} de {}", rodada, total\_de\_tentativas)

Voltando ao código do nosso jogo de adivinhação, implementamos o loop **while**, no qual temos uma variável **rodada** que começa com o valor 1, e é incrementada dentro do loop, que por sua vez tem uma condição de entrada, que é a **rodada** ser menor ou igual ao total de tentativas, que é 3.

Ou seja, a **rodada** tem um valor inicial, que é 1, e vai até 3. Fazemos um laço começando com um valor inicial, até um valor final, sempre incrementando esse valor a cada iteração.

Em casos como esse, existe um outro loop que simplifica essa ideia de começar com um valor, e incrementá-lo até chegar em um valor final, o loop **for**.

## Entendendo o for

Para entender o loop **for**, podemos ir até o console do Python para ver o seu funcionamento. A ideia é nós definirmos o valor inicial e o valor final, que o loop o incrementa automaticamente. Para definir o valor inicial e final, utilizamos a função **range**, passando-os por parâmetro, definindo assim a série de valores. A sintaxe é a seguinte

>>> para variável em série de valores:

... faça algo

Isso, em Python, pode ficar assim:

>>> for rodada in range(1,10):

...

Na primeira iteração, o valor da variável **rodada** será 1, depois 2 e até chegar ao **valor final da função range menos 1**, isto é, o segundo parâmetro da função não é inclusivo. No exemplo acima, a série de valores é de 1 a 9. Podemos confirmar isso imprimindo o valor da variável **rodada**:

>>> for rodada in range(1,10):

... print(rodada)

...

1

2

3

4

5

6

7

8

9

Com a função **range**, podemos definir um step, que é o intervalo entre os elementos, por padrão o step é 1. Definimos-o passando um terceiro parâmetro para a função:

>>> for rodada in range(1,10,2):

... print(rodada)

...

1

3

5

7

9

Mas não necessariamente precisamos usar a função **range** no **for**, podemos passar os valor manualmente:

>>> for rodada in [1,2,3,4,5]:

... print(rodada)

...

1

2

3

4

5

O resultado é o mesmo, mas o código fica mais verboso.

## Utilizando o for no jogo

Voltando ao nosso jogo, não vamos mais utilizar o **while, e sim o** for**, começando no 1 e indo até o total de tentativas. Para isso precisamos remover a declaração da variável** rodada\* e o seu incremento dentro do loop:

numero\_secreto = 42

total\_de\_tentativas = 3

for rodada in range(1, total\_de\_tentativas):

print("Tentativa {} de {}".format(rodada, total\_de\_tentativas))

chute\_str = input("Digite o seu número: ")

print("Você digitou: ", chute\_str)

chute = int(chute\_str)

acertou = numero\_secreto == chute

maior = chute > numero\_secreto

menor = chute < numero\_secreto

if (acertou):

print("Você acertou!")

else:

if (maior):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (menor):

print("Você errou! O seu chute foi menor que o número secreto.")

print("Fim do jogo")

É importante saber que **o for não deve ter parênteses**.

Podemos testar e ver que só fizemos 2 tentativas. Isso porque, como foi falado anteriormente, o segundo parâmetro da função **range não é inclusivo**, no caso do nosso jogo, **`range(1,3)** irá gerar a série 1 e 2 somente. Logo vamos somar 1 ao total de tentativas dentro da função **range**:

numero\_secreto = 42

total\_de\_tentativas = 3

for rodada in range(1, total\_de\_tentativas + 1):

print("Tentativa {} de {}".format(rodada, total\_de\_tentativas))

chute\_str = input("Digite o seu número: ")

print("Você digitou: ", chute\_str)

chute = int(chute\_str)

acertou = numero\_secreto == chute

maior = chute > numero\_secreto

menor = chute < numero\_secreto

if (acertou):

print("Você acertou!")

else:

if (maior):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (menor):

print("Você errou! O seu chute foi menor que o número secreto.")

print("Fim do jogo")

Agora podemos testar novamente o nosso jogo, e ver que tudo está funcionando perfeitamente!

Se executarmos o programa, a seguinte frase é impressa:

Tentativa {} de {} 1 3

Não é exatamente isso que queremos, as primeiras chaves devem receber o valor da rodada, e as segundas o total de tentativas. Para isso funcionar, devemos chamar uma função baseada nessa string, a função **format**, passando para ela as variáveis que devem ficar no lugar das chaves:

print("Tentativa {} de {}".format(rodada, total\_de\_tentativas))

Podemos testar e ver que agora está tudo funcionando como antes! O que acabamos de fazer se chama **interpolação de strings**, muito comum nas linguagens e que nos oferece recursos da string para fazermos essas substituições.

Assim o nosso código fica um pouco mais elegante, já que nele vemos a string inteira, sabendo exatamente onde ela será alterada.

No nosso jogo, sabemos que o número secreto é fixo e definido com o valor 42, por enquanto. Vamos jogar e digitar esse valor de primeira:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Bem vindo ao jogo de Adivinhação!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 3

Digite o seu número: 42

Você digitou: 42

Você acertou!

Tentativa 2 de 3

Digite o seu número:

Acertamos o número, mas ainda temos uma segunda e terceira tentativas! Não faz muito sentido isso né? Se nós ganhamos, temos que parar as rodadas, não devemos continuar.

## Parando o laço

Dentro do **if**, se acertarmos, devemos parar e sair do laço. Para isso existe um comando do Python, assim como outras linguagens, o **break**, que faz com que saiamos do laço:

if (acertou):

print("Você acertou!")

break

else:

if (maior):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (menor):

print("Você errou! O seu chute foi menor que o número secreto.")

Podemos agora jogar novamente e...:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Bem vindo ao jogo de Adivinhação!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 3

Digite o seu número: 42

Você digitou: 42

Você acertou!

Fim do jogo

Ótimo! Acertamos o número e o jogo foi encerrado, sem mais rodadas.

## Limitando o número a ser digitado

Vamos limitar o número que o usuário deve digitar, de 1 a 100. Vamos deixar isso claro para ele alterando a mensagem do **input**:

for rodada in range(1, total\_de\_tentativas + 1):

print("Tentativa {} de {}".format(rodada, total\_de\_tentativas))

chute\_str = input("Digite um número entre 1 e 100: ")

print("Você digitou: ", chute\_str)

chute = int(chute\_str)

## resto do código comentado

Só que agora não devemos aceitar valores fora desse limite, logo vamos verificar o número digitado, e se ele for menor que 1 **OU** (em Python, a palavra chave **or**) maior que 100, vamos exibir uma mensagem para o usuário:

for rodada in range(1, total\_de\_tentativas + 1):

print("Tentativa {} de {}".format(rodada, total\_de\_tentativas))

chute\_str = input("Digite um número entre 1 e 100: ")

print("Você digitou: ", chute\_str)

chute = int(chute\_str)

if (chute < 1 or chute > 100):

print("Você deve digitar um número entre 1 e 100!")

## resto do código comentado

Mas não faz sentido continuarmos executando o código do loop se o valor não estiver no intervalo exigido. O que queremos não é sair do laço, e sim continuar para a próxima rodada, **acabando com a iteração**. Para isso existe a palavra chave **continue**:

for rodada in range(1, total\_de\_tentativas + 1):

print("Tentativa {} de {}".format(rodada, total\_de\_tentativas))

chute\_str = input("Digite um número entre 1 e 100: ")

print("Você digitou: ", chute\_str)

chute = int(chute\_str)

if (chute < 1 or chute > 100):

print("Você deve digitar um número entre 1 e 100!")

continue

## resto do código comentado

Esse comando faz com que a iteração do laço acabe, e comece a próxima. Vamos testar:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Bem vindo ao jogo de Adivinhação!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 3

Digite um número entre 1 e 100: 0

Você digitou: 0

Você deve digitar um número entre 1 e 100!

Tentativa 2 de 3

Digite um número entre 1 e 100:

Perfeito! O número digitado era incorreto, então fomos para a próxima tentativa.

Então vimos aqui o **break**, que acaba, encerra o laço; e o **continue**, que acaba, encerra a iteração, continuando para a próxima.

Como apresentamos no vídeo, substitua o laço while com o for. A função range deve gerar uma sequência de 1 até total\_de\_tentativas + 1:

total\_de\_tentativas = 3

for rodada in range(1, total\_de\_tentativas + 1):

No final do código, **não esqueça de remover a linha que incrementa a variável rodada**. Lembrando, o incremento é automaticamente feito pelo laço for.

## Encerrando o jogo

Dentro do **if**, se acertarmos, devemos parar e sair do laço para parar o jogo.

Para tal, use o comando break:

if (acertou):

print("Você acertou!")

break

## Limitando a entrada

Para limitar o número que o usuário deve digitar, altere a mensagem da função input:

chute\_str = input("Digite um número entre 1 e 100: ")

Agora verifique o número digitado através de um novo if, logo após a declaração da variável chute. Use o comando continue para continuar com a próxima iteração:

for rodada in range(1, total\_de\_tentativas + 1):

print("Tentativa {} de {}".format(rodada, total\_de\_tentativas))

chute\_str = input("Digite um número entre 1 e 100: ")

print("Você digitou: ", chute\_str)

chute = int(chute\_str)

if (chute < 1 or chute > 100):

print("Você deve digitar um número entre 1 e 100!")

continue

## Rodando o jogo

Rode o programa e teste uma entrada inválida (-1, por exemplo).

O for é uma alternativa interessante ao while, muito utilizado para iterar por uma sequência de valores. Lembrando que break e continue também podem ser utilizados com while.

De qualquer forma, segue o código escrito até o momento:

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Bem vindo ao jogo de Adivinhação!")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

numero\_secreto = 42

total\_de\_tentativas = 3

for rodada in range(1, total\_de\_tentativas + 1):

print("Tentativa {} de {}".format(rodada, total\_de\_tentativas))

chute\_str = input("Digite um número entre 1 e 100: ")

print("Você digitou " , chute\_str)

chute = int(chute\_str)

if(chute < 1 or chute > 100):

print("Você deve digitar um número entre 1 e 100!")

continue

acertou = chute == numero\_secreto

maior = chute > numero\_secreto

menor = chute < numero\_secreto

if(acertou):

print("Você acertou!")

break

else:

if(maior):

print("Você errou! O seu chute foi maior do que o número secreto.")

elif(menor):

print("Você errou! O seu chute foi menor do que o número secreto.")

print("Fim do jogo")

Segue o link da documentação que mencionei no video, nele tem vários exemplos de formatação:

<https://docs.python.org/3/library/string.html#formatexamples>

Nesse vídeo veremos um pouco mais sobre interpolação de strings. Para isso, vamos utilizar o console do Python 3.

No capítulo anterior, fizemos uma interpolação semelhante a essa:

>>> print("Tentativa {} de {}".format(1, 3))

Essa interpolação é útil para formatação de strings, quando temos um texto muito grande e precisamos inserir valores no meio dele, ao invés de ficarmos concatenando, trabalhando com várias strings separadas.

Mas a função **format** tem outras utilidades, então veremos mais alguns detalhes sobre essa função. O primeiro detalhe que veremos é que os parâmetros podem ser invertidos na string. Podemos dizer que queremos nas primeiras chaves o segundo parâmetro da função, e o primeiro parâmetro nas segundas chaves.

Fazemos isso passando o **índice do parâmetro** dentro das chaves. O primeiro parâmetro tem índice **0**, o segundo **1**, e daí por diante. Logo, basta passar o índice 1 nas primeiras chaves e o 0 nas segundas chaves:

>>> print("Tentativa {1} de {0}".format(1, 3))

Tentativa 3 de 1

## Formatação de floats

Agora vamos trocar o exemplo, e formatar um valor em reais, por exemplo:

>>> print("R$ {}".format(1.59))

R$ 1.59

Só que um valor pode ter vários tamanhos e até duas casas decimais, por exemplo:

1.59

45.9

1234.97

O ideal é que esses valores sempre tenham a mesma formatação:

1.59

45.9

1234.97

Então precisamos preencher as lacunas, os espaços em branco. E a função **format** faz isso para nós. Primeiro precisamos dizer para ela que estamos recebendo um valor do tipo **float**, passando **:f** dentro das chaves da string:

>>> print("R$ {:f}".format(1.59))

R$ 1.590000

Podemos reparar que só de dizer que estamos passando um float, a formatação já muda, mas podemos manipulá-la, modificá-la, dizendo quantos números devem vir antes e depois do ponto. Queremos que após o ponto tenha apenas 2 números, logo:

>>> print("R$ {:.2f}".format(1.59))

R$ 1.59

Podemos testar passando um número de apenas uma casa decimal:

>>> print("R$ {:.2f}".format(1.5))

R$ 1.50

Ótimo, agora vamos testar com um número maior:

>>> print("R$ {:.2f}".format(1.5))

R$ 1.50

>>> print("R$ {:.2f}".format(1234.50))

R$ 1234.50

Mas queremos que o ponto fique sempre no mesmo local, ou seja, ele deve ser o quinto caractere. Para essa formatação, precisamos dizer quantos caracteres o número terá no máximo, no nosso caso são 7 (4 números, mais o ponto, mais as duas casas decimais). Então vamos passar o valor 7 dentro das chaves também:

>>> print("R$ {:7.2f}".format(1234.50))

R$ 1234.50

>>> print("R$ {:7.2f}".format(1.5))

R$ 1.50

Ou seja, dos 7 caracteres, os três últimos serão o ponto mais dois números das casas decimais.

Agora espaços ficam na frente quando um número for menor! Deixando o ponto sempre como quinto caractere. Se quisermos preencher os espaços em branco com zeros, é só passar um 0 antes do 7:

>>> print("R$ {:07.2f}".format(1.5))

R$ 0001.50

## Formatação de inteiros

Conseguimos formatar números inteiros também, não só números flutuantes. Para números inteiros, passamos a letra **d**:

>>> print("R$ {:07d}".format(4))

R$ 0000004

Podemos usar isso para formatar uma data:

>>> print("Data {:02d}/{:02d}".format(9, 4))

Data 09/04

>>> print("Data {:02d}/{:02d}".format(19, 11))

Data 19/11

Não se preocupe em decorar a sintaxe, o importante é saber que no Python existe a funcionalidade de interpolação de strings, e quando vocês realmente precisarem usar isso, olhem na [documentação](https://docs.python.org/3/library/string.html#formatexamples).

## No Python 3.6+

A partir da [versão 3.6 do Python](https://docs.python.org/3/whatsnew/3.6.html#whatsnew36-pep498), foi adicionado um novo recurso para realizar a interpolação de strings. Esse recurso é chamado de f-strings ou formatted string literals.

Esse recurso funciona da seguinte forma. Vamos imaginar que temos uma variável nome:

>>> nome = 'Matheus'

>>> print(f'Meu nome é {nome}')

Meu nome é Matheus

Quando colocamos a letra f antes das aspas, informamos ao Python que estamos utilizando uma f-string. Dessa forma o Python consegue, em tempo de execução, captar a expressão que está entre chaves ({ }) e avaliá-la.

Além de variáveis, podemos passar também de funções e métodos:

>>> nome = 'Matheus'

>>> print(f'Meu nome é {nome.lower()}')

Meu nome é matheus

A lógica principal do nosso jogo já está funcionando, mas ainda há um detalhe, o número secreto não é tão secreto assim, pois ele está fixo! Então vamos alterar isso, para que ele passe a ser um número aleatório, coisa que veremos nesse capítulo.

## Gerando um número aleatório

A ideia é que o próprio jogo, toda vez que for executado, gere esse número, ele que decide isso, não nós. E para gerar um número aleatório, o Python 3 possui a função **random()**, que gera um número no intervalo entre 0.0 e 1.0. Mas ao contrário das [funções built-in](https://docs.python.org/3/library/functions.html) do Python, como as funções **input()**, **int()**, **print()** e **range()**, que são **funções embutidas** do Python (que já vem com o mesmo), a função **random** não vem, pois está em um módulo separado, e esse módulo precisa ser importado.

Podemos ir ao console do Python e testar isso. Primeiro importando o módulo:

>>> import random

E a partir desse módulo, chamamos a função **random()**:

>>> import random

>>> random.random()

0.6022965518496559

## Arredondando um número

Só que, como podemos perceber, o número gerado tem muitas casas decimais e está no intervalo entre 0.0 e 1.0, mas no nosso jogo precisamos de um número entre 1 e 100. O que podemos fazer é multiplicar o número gerado por 100:

>>> import random

>>> random.random() \* 100

58.30742817094118

Já conseguimos chegar a um número mais próximo do ideal, falta agora removermos as casas decimais. Podemos utilizar a já conhecida função **int**, que irá converter o número aleatório, que é um float, em um número inteiro:

>>> int(random.random() \* 100)

91

Mas reparem no exemplo abaixo:

>>> numero\_random = random.random() \* 100

>>> numero\_random

18.895629671768187

>>> int(numero\_random)

18

A função **int** nada mais faz do que **remover** as casas decimais do número flutuante. Mas o número gerado acima está mais próximo de 19 do que de 18, correto? Será que temos uma função que **arredonda** esse número para nós? Sim! Temos mais uma função built-in, a **round**:

>>> numero\_random = random.random() \* 100

>>> numero\_random

18.895629671768187

>>> int(numero\_random)

18

>>> round(numero\_random)

19

Conhecendo isso, podemos aplicar ao nosso jogo. Faremos isso no próximo vídeo, até lá!

Para gerar um número aleatório no nosso jogo, a primeira coisa que devemos fazer é importar o seu módulo, no início do programa:

import random

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Bem vindo ao jogo de Adivinhação!")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

numero\_secreto = 42

total\_de\_tentativas = 3

# restante do código comentado

Com o módulo importado, vamos remover o valor fixo da variável **numero\_secreto** e substituir por um valor aleatório que será gerado pela função **random()**:

import random

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Bem vindo ao jogo de Adivinhação!")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

numero\_secreto = random.random()

total\_de\_tentativas = 3

# restante do código comentado

Mas não podemos nos esquecer de multiplicar esse número por 100 e arredondá-lo:

import random

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Bem vindo ao jogo de Adivinhação!")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

numero\_secreto = round(random.random() \* 100)

total\_de\_tentativas = 3

# restante do código comentado

Perfeito, conseguimos aplicar a mudança ao nosso código. Agora fica mais difícil de acertar o número secreto, até para nós, os desenvolvedores do jogo :)

## Gerando um número aleatório dentro de um intervalo

A ideia de multiplicar o número por 100 parece funcionar, mas podemos lembrar que o número gerado é entre 0.0 e 1.0, que quando multiplicado por 100 fica entre 0 e 100. Só que o nosso jogo não aceita o 0!

O ideal seria que pudéssemos definir um intervalo, dizer que queremos que o número gerado esteja entre 1 e 100. Como o **random** é um módulo, ele possui mais de uma função e a função **randrange()** serve exatamente para esse nosso problema. Se passarmos um parâmetro para ela, ela irá gerar um número inteiro de 0 até o valor desse parâmetro menos 1. Se passarmos dois parâmetros para ela, ela irá gerar um número inteiro do valor do primeiro parâmetro até o valor do segundo parâmetro menos 1, exatamente o que queremos!

Vamos, passando o intervalo que queremos para a função **randrange()**, lembrando que como queremos que o número gerado esteja entre 1 e 100 (inclusive), precisamos passar o número 101 como segundo parâmetro para a função:

import random

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Bem vindo ao jogo de Adivinhação!")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

numero\_secreto = random.randrange(1, 101)

total\_de\_tentativas = 3

# restante do código comentado

Ótimo, problema resolvido!

No próximo capítulo vamos focar na dificuldade do jogo. Um jogo mais fácil terá um número de tentativas maior, e um mais difícil terá um número de tentativas menor. Até lá!

## Pseudo-Random?

Aparentemente a geração de números aleatórios funcionou muito bem. Cada vez que chamamos o random.random() ou random.randrange(..), foi gerado um outro número.

No entanto, computadores têm os seus problemas com aleatoriedade, pois são sistemas determinísticos. Em outras palavras, o nosso Python é previsível e na verdade não sabe criar números verdadeiramente aleatórios. Por isso se chama Pseudo-Random!

## Por que funcionou então?

random é um função que, dada a mesma entrada, gerará o mesmo número. O truque é oferecer sempre uma entrada diferente para ter números diferentes e exatamente isso que está acontecendo por baixo dos panos.

Essa entrada também é chamada de seed (semente, em português). Entre as chamadas da função random, sempre é utilizado um novo seed. Por padrão o Python usa a hora (os milissegundos) como semente, mas nada nos impede de definir o mesmo seed antecipadamente. Para isso, existe a função seed!

## Usando seed

Por exemplo, no jogo usamos a função randrange para gerar um número aleatório entre 1 e 100. Antes do randrange podemos chamar o seed para definir a entrada:

>>> random.seed(1, 101)

>>> random.randrange(100)

19

Repare que foi gerado 19 e se usarmos o mesmo seed será gerado o mesmo número:

>>> random.seed(1, 101)

>>> random.randrange(100)

19

Repare que a biblioteca random é bem previsível e por isso se chama pseudo-random!

Vamos adicionar níveis ao nosso jogo, e conforme o nível vai ficando mais difícil, menos tentativas o usuário terá.

Começaremos perguntando ao usuário qual nível de dificuldade ele quer:

import random

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Bem vindo ao jogo de Adivinhação!")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

numero\_secreto = random.randrange(1, 100)

total\_de\_tentativas = 3

print("Qual o nível de dificuldade?")

print("(1) Fácil (2) Médio (3) Difícil")

# resto do código comentado

E capturaremos o que ele digitar, já convertendo o valor para inteiro e guardando em uma variável:

import random

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Bem vindo ao jogo de Adivinhação!")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

numero\_secreto = random.randrange(1, 100)

total\_de\_tentativas = 3

print("Qual o nível de dificuldade?")

print("(1) Fácil (2) Médio (3) Difícil")

nivel = int(input("Defina o nível: "))

# resto do código comentado

Agora falta mudar o total de tentativas baseado no nível que o usuário escolher. A variável será inicializada com 0, e faremos um **if** para verificar o nível escolhido, se o ele for fácil, o usuário terá 20 tentativas, se for médio terá 10, e se for difícil terá 5 tentativas:

import random

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Bem vindo ao jogo de Adivinhação!")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

numero\_secreto = random.randrange(1, 100)

total\_de\_tentativas = 0

print("Qual o nível de dificuldade?")

print("(1) Fácil (2) Médio (3) Difícil")

nivel = int(input("Defina o nível: "))

if (nivel == 1):

total\_de\_tentativas = 20

elif (nivel == 2):

total\_de\_tentativas = 10

else:

total\_de\_tentativas = 5

# resto do código comentado

Com isso conseguimos definir os níveis de dificuldade no nosso jogo. No próximo vídeos definiremos pontuação!

Com os níveis definidos, vamos agora calcular uma pontuação. Ela funcionará da seguinte maneira, o usuário começa o jogo com 1000 pontos, e a cada rodada que ele não acerta o número secreto, ele perderá pontos. Quanto mais distante for o chute, mais pontos o usuário irá perder. Por exemplo, se o número secreto for 40, e o usuário chutar 20, ele irá perder 20 pontos, que corresponde à distância entre os valores.

Vamos começar definindo a variável com 1000 pontos:

pontos = 1000

Após isso, o usuário irá perder pontos caso ele erre o chute, logo temos que implementar isso dentro da condição **else**:

if (acertou):

print("Você acertou!")

break

else:

if (maior):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (menor):

print("Você errou! O seu chute foi menor que o número secreto.")

Vamos definir a variável **pontos\_perdidos**, que subtrai o chute do número secreto:

if (acertou):

print("Você acertou!")

break

else:

if (maior):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (menor):

print("Você errou! O seu chute foi menor que o número secreto.")

pontos\_perdidos = numero\_secreto - chute

Depois, vamos subtrair os pontos perdidos da pontuação total:

if (acertou):

print("Você acertou!")

break

else:

if (maior):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (menor):

print("Você errou! O seu chute foi menor que o número secreto.")

pontos\_perdidos = numero\_secreto - chute

pontos = pontos - pontos\_perdidos

Isso funciona caso o usuário chute um número menor que o número secreto, mas e se for maior? Por exemplo, se o número secreto for 40 e o usuário chutar 60, de acordo com o cálculo do nosso código os pontos perdidos serão **-20**, e ao subtrair esse valor da pontuação total, ela irá aumentar!

Então queremos fazer a subtração dos pontos perdidos, mas caso essa subtração tenha como resultado um número negativo, queremos que "esquecer" esse sinal, queremos sempre o **número absoluto**.

E para extrair o número absoluto, existe mais uma função built-in, a **abs()**:

if (acertou):

print("Você acertou!")

break

else:

if (maior):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (menor):

print("Você errou! O seu chute foi menor que o número secreto.")

pontos\_perdidos = abs(numero\_secreto - chute)

pontos = pontos - pontos\_perdidos

Por fim, falta exibirmos a pontuação final ao usuário. Vamos alterar a mensagem de acerto do usuário, acrescentando a pontuação total. Faremos uso novamente da interpolação de strings:

if (acertou):

print("Você acertou e fez {} pontos!".format(pontos))

break

else:

if (maior):

print("Você errou! O seu chute foi maior que o número secreto.")

elif (menor):

print("Você errou! O seu chute foi menor que o número secreto.")

pontos\_perdidos = abs(numero\_secreto - chute)

pontos = pontos - pontos\_perdidos

Com isso chegamos ao final do nosso jogo! No próximo capítulo veremos um pouco sobre linguagens compiladas e interpretadas, entre outros assuntos. Até lá!

Se conseguimos executar o jogo dentro do PyCharm, também conseguimos rodar o jogo na linha de comando, no terminal. Dentro do diretório do projeto **jogos**, basta executar:

python3 adivinhacao.py

No próximo treinamento, criaremos mais um jogo, a **forca**. Então já vamos deixar o seu arquivo preparado, criando o **forca.py**, também dentro do projeto **jogos**. Dentro desse arquivo, vamos deixar as mensagens de início e fim de jogo, semelhante ao jogo de adivinhação:

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\*Bem vindo ao jogo da Forca!\*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Fim do jogo")

## Oferecendo todos os jogos ao usuário

Vamos oferecer os dois jogos ao usuário, ou seja, devemos perguntar ao usuário qual jogo ele quer executar, jogar. Mas onde vamos colocar essa funcionalidade? A ideia é não misturar os jogos, deixar cada um em um arquivo separado. Então vamos criar um novo arquivo com essa funcionalidade, o arquivo **jogos.py**, perguntando qual jogo ele quer escolher jogar:

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*Escolha o seu jogo!\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("(1) Forca (2) Adivinhação")

Agora vamos capturar a opção do usuário e verificar qual jogo ele escolheu:

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*Escolha o seu jogo!\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("(1) Forca (2) Adivinhação")

jogo = int(input("Qual jogo? "))

if (jogo == 1):

print("Jogando forca")

elif (jogo == 2):

print("Jogando adivinhação")

## Importando arquivos

Ótimo, mas se queremos chamar um arquivo dentro de outro, precisamos importá-lo, algo parecido com o que fizemos com o módulo **random**:

import forca

import adivinhacao

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*Escolha o seu jogo!\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("(1) Forca (2) Adivinhação")

jogo = int(input("Qual jogo? "))

if (jogo == 1):

print("Jogando forca")

elif (jogo == 2):

print("Jogando adivinhação")

## O problema do import

Podemos executar para ver como está ficando:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*Bem vindo ao jogo da Forca!\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Fim do jogo

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Bem vindo ao jogo de Adivinhação!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual o nível de dificuldade?

(1) Fácil (2) Médio (3) Difícil

Defina o nível:

Ué, o que aconteceu? Quando importamos um arquivo no Python, ele o executa! Podemos reparar que ele executou o arquivo **forca.py** e logo depois o **adivinhacao.py**. Mas obviamente não queremos isso, só queremos executar o arquivo quando nós quisermos! E é isso que faremos no próximo vídeo.

Pouco vimos sobre funções, mas não se preocupe. Na medida em que você avança nos cursos sobre Python 3, vamos introduzir mais recursos.

Para declarar uma função, devemos usar a palavra chave def do mundo Python, seguido pelo nome da função. Lembrando que é consenso usar a nomenclatura snake\_case:

def nome\_da\_funcao():

# todo o código identado faz parte da função

print("aprendendo funções")

Repare que uma função pode chamar uma outra função. print também é uma função e usamos ela dentro da nossa própria função.

## Executando funções

Para chamar a nossa própria função, usamos o nome dela seguido pelos parênteses, por exemplo:

nome\_da\_funcao()

Podemos chamar uma função quantas vezes quisermos:

nome\_da\_funcao()

nome\_da\_funcao()

nome\_da\_funcao()

Isso é a principal vantagem de funções, reaproveitar o código escrito nela!

## Parâmetros e retorno

Uma função também pode receber parâmetros e retornar algum valor, por exemplo:

def soma(a, b):

return a + b

A função soma recebe dois parâmetros (a e b) e retorna a soma. Ao chamar a função, podemos capturar o retorno:

s = soma(3, 4)

Isso foi apenas uma pequena introdução, mas novamente, ainda vamos utilizar muito as funções e praticar para fixar o conteúdo.

Não conseguimos mais jogar diretamente cada jogo porque o seu próprio arquivo não chama a sua função **jogar()**. Então, depois da função, vamos chamá-la:

# adivinhacao.py

import random

def jogar():

# código omitido

jogar()

Isso resolve o problema de jogar o jogo diretamente mas voltamos ao problema do vídeo anterior! Ao executarmos o arquivo **jogos.py**, como o próprio arquivo **adivinhacao.py** chama a função **jogar()**, ela será executada sem que queiramos isso.

Precisamos dar um jeito para que, quando executarmos o jogo de adivinhação diretamente, a função **jogar()** deve ser chamada, mas quando só o importamos, não queremos que a função seja chamada.

## Programa principal vs Programa importado

Quando rodamos diretamente um arquivo no Python, ele internamente cria uma variável e a preenche. E através dessa variável podemos fazer uma consulta, pois se ela estiver preenchida, significa que o arquivo foi executado diretamente, mas se a variável não estiver preenchida, então significa que o arquivo só foi importado.

Essa variável é a **\_\_name\_\_**, e ela é preenchida com o valor **\_\_main\_\_** caso o arquivo seja executado diretamente. Vamos então fazer **if** para verificar se ela está preenchida ou não:

# adivinhacao.py

import random

def jogar():

# código omitido

if (\_\_name\_\_ == "\_\_main\_\_"):

jogar()

Podemos agora testar os dois casos, executar o arquivo diretamente e executar o arquivo **jogos.py**. Os dois estão funcionando, exatamente como queríamos. Falta fazermos a mesma coisa com o jogo da forca:

# forca.py

def jogar():

# código omitido

if (\_\_name\_\_ == "\_\_main\_\_"):

jogar()

E com o arquivo **jogos.py**, colocando o seu código dentro da função **escolhe\_jogo()** e chamando-a caso o programa seja o programa principal:

import forca

import adivinhacao

def escolhe\_jogo():

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*Escolha o seu jogo!\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("(1) Forca (2) Adivinhação")

jogo = int(input("Qual jogo? "))

if (jogo == 1):

print("Jogando forca")

forca.jogar()

elif (jogo == 2):

print("Jogando adivinhação")

adivinhacao.jogar()

if (\_\_name\_\_ == "\_\_main\_\_"):

escolhe\_jogo()

Com isso vimos como diferenciar se o programa é o principal ou não, se ele está sendo executado diretamente ou só sendo importado. Na hora de importar um arquivo, ele lê o código da função, mas não o executa, pois ele não é o arquivo principal

No curso [**C I: Introdução à Linguagem das Linguagens**](https://cursos.alura.com.br/course/introducao-a-programacao-com-c-parte-1) também é implementado um jogo de adivinhação, você pode baixá-lo [aqui](https://s3.amazonaws.com/caelum-online-public/python3/files/adivinhacao.c) e deixe-o na mesma pasta que os arquivos Python, ou seja, na pasta **jogos**. O que faremos nesse vídeo é comparar o Python com C, mas sem explicá-lo, a ideia é só enfatizar algumas diferenças entre eles.

A primeira diferença que vemos é que no C precisamos importar mais bibliotecas, isso porque algumas funções no C não são built-in, como a de imprimir (**printf**) e a de capturar entrada do usuário (**scanf**), por isso para utilizá-las é necessário importar algumas bibliotecas.

Outra diferença é que no C somos obrigados a definir a função **main**, que é considerada o início de qualquer programa, sem ela nada vai funcionar. Ao contrário do Python, já que nós só criamos a função **jogar()** quando precisamos importar o arquivo em outro, mas antes nós conseguíamos executar diretamente o jogo sem problemas.

No capítulo 1 falamos sobre tipagem de variáveis, e o C é uma linguagem que possui a **tipagem estática**, ou seja, quando declaramos uma variável, precisamos dizer qual será o tipo dela e esse tipo nunca mudará. O Python, como já sabemos, é uma linguagem que possui a **tipagem dinâmica**, já que o tipo da variável varia de acordo com o valor que ela recebe, por isso que em Python não podemos declarar variáveis vazias, só definindo o seu nome, porque se não atribuirmos um valor a uma variável, o Python não saberá o seu tipo.

O resto do código é bem parecido, com algumas diferenças de sintaxe e nomenclatura, como por exemplo a sintaxe dos blocos, para definir um bloco no C devemos colocá-lo entre chaves e a indentação não é obrigatório, apesar de todos os desenvolvedores utilizarem por conta da formatação do código, já no Python só precisamos colocar os dois pontos e indentar o código do bloco; o C também te obriga a colocar ponto e vírgula ao final das instruções.

Essas foram algumas das diferenças que podemos citar comparando os dois códigos. Agora veremos na hora da execução, mas no próximo vídeo :)

Para executar um arquivo Python, por exemplo o **adivinhacao.py**, fazemos:

python3 adivinhacao.py

Foi isso que fizemos o treinamento inteiro, seja pela linha de comando ou pelo PyCharm. E com o C? Será que existe um comando, algo como um **cexecuter**, para executarmos um arquivo C? Não, não existe.

O ambiente do C exige que primeiramente devemos passar o código fonte (o arquivo .c) para um **compilador**. O compilador lê o código fonte e faz uma análise da sintaxe, se esquecemos algum ponto e vírgula, ou de tipar alguma variável, etc. E feita essa análise, o compilador cria um outro arquivo, e é esse arquivo que podemos executar. Então primeiro vamos compilar o arquivo, vamos utilizar o compilador **gcc** (novamente, não é necessária a sua instalação, estamos usando-o somente para mostrar a diferença entre ambientes que usam o conceito de compilação e ambientes que usam o conceito de interpretação):

gcc adivinhacao.c o adivinhacao

Ou:

gcc -std=c99 adivinhacao.c -o adivinhacao

Esse comando compila o arquivo **adivinhacao.c** e se tudo estiver correto, criará o arquivo executável **adivinhacao**. Agora é só executar o arquivo gerado. Em UNIX, fazemos:

./adivinhacao

Essa é a diferença de um ambiente que usa o conceito de compilação, no qual o código fonte, que não é executável, deve ser compilado para criar um arquivo executável; e um ambiente que usa o conceito de interpretação, no qual o código fonte é executado diretamente.

## Transferindo código

Em Python, conseguimos executar um arquivo em qualquer sistema operacional, inclusive os códigos aqui feitos são disponibilizados para vocês, alunos, e ele poderá ser executado seja qual for o seu sistema operacional, basta ter o Python 3 instalado. Já o arquivo executável do C, gerado pelo compilador, não é executável em um sistema operacional diferente. É preciso compilar novamente o código fonte no sistema operacional desejado, para ter um executável funcional. E muitas vezes o código fonte (não é o nosso caso) utiliza algo específico do sistema operacional, passando a depender dele, então nem adiantaria compilá-lo em um SO diferente.

Logo, o Python tem uma portabilidade maior que o C.

## O Python é realmente uma linguagem estritamente interpretada?

Para finalizar, falamos que o Python utiliza o conceito de interpretação, ou seja, passamos o código fonte e ele é interpretado, mas não é bem assim. Podemos executar o arquivo **jogos.py** e reparar na pasta que é criada dentro do diretório, a **pycache**. Se formos ver o que tem dentro da sua pasta, vemos que há dois arquivos referentes aos módulos importados no **jogos.py**, ou seja, um arquivo referente à **adivinhacao** e outro à **forca**. Mas o que são esses arquivos?

O que o Python faz ao vivo é ler os módulos importados e os **compila para bytecode**. Esse código foi criado ao mesmo tempo em que executamos o arquivo **jogos.py**. Apesar do Python ter um ambiente de interpretação, ele compila os módulos importados para melhorar o desempenho, a execução do ambiente, apesar de não ter esse processo de compilação explícito.

Do ponto de vista do Python, ele considera que esses módulos não serão modificados, então na próxima execução, para melhorar o desempenho, esse código compilado é que será utilizado.

Com isso, terminamos aqui o nosso curso. No próximo implementaremos o jogo da forca, aprendendo mais sobre funções, coleções, outras funções built-in, e muito mais! Muito obrigado por assistirem esse curso e nos encontramos no próximo treinamento!

O senso comum é que o Python é uma **linguagem interpretada**. **Interpretado** significa que não há um processo de compilação que traduz o código fonte em algum código nativo, que o seu computador entende. A [documentação do Python](https://docs.python.org/3/glossary.html) confirma isso, no entanto também menciona a presença de um compilador:

**"Python is an interpreted language, as opposed to a compiled one, though the distinction can be blurry because of the presence of the bytecode compiler."**

Traduzido livremente: **\*"Python é uma linguagem interpretada, em oposição às compiladas, embora a distinção possa ficar desfocada devido à presença do compilador de bytecode."**

Temos um compilador, porém de bytecode. Bytecode é um código intermediário, normalmente independente do sistema operacional. Então, Python é uma linguagem compilada também? Em 2003, Fredrik Lundh, em seu artigo [Compiling Python Code](http://effbot.org/zone/python-compile.htm), título que perverte o senso comum, começa:

**"Python source code is automatically compiled into Python byte code by the CPython interpreter. Compiled code is usually stored in PYC (or PYO) files, and is regenerated when the source is updated, or when otherwise necessary."**

Novamente traduzindo livremente: **"O código fonte é automaticamente compilado para bytecode do Python pelo interpretador CPython. O código compilado é comumente armazenado nos arquivos no PYC (ou PY0), sendo regerado quando o arquivo fonte é atualizado ou quando é necessário."**

E aí? Python é uma linguagem interpretada ou compilada? As duas coisas? Há discussões acaloradas entre desenvolvedores, cada um com sua opinião. Então, [uma resposta interessante](http://stackoverflow.com/questions/6889747/is-python-interpreted-or-compiled-or-both) está no StackOverFlow, aliás, a resposta mais bem avaliada:

**"First off, interpreted/compiled is not a property of the language but a property of the implementation (...) Python is compiled. Not compiled to machine code ahead of time (i.e. "compiled" by the restricted and wrong, but also common definition), "only" compiled to to bytecode"**

Traduzindo: **"O fato de uma linguagem ser interpretada ou compilada não é uma questão da linguagem, mas da sua implementação. (...) Python é compilada. Não compilada para o código de máquina antes da execução, apenas para o bytecode.**

Isso significa que alguém pode implementar o Python totalmente compilado, totalmente interpretado ou ambos, a linguagem continua a mesma. Ser compilada/interpretada é mais propriedade da implementação do Python do que da linguagem.

E você, o que pensa dessa definição?