Conhecendo o jogo

O jogo da forca também é um jogo de adivinhação, mas no caso o usuário deve adivinhar uma palavra secreta. No arquivo forca.py, já há a função jogar definida, onde ficará o código do nosso jogo:

```
def jogar():
    print("**********************************")
    print("***Bem vindo ao jogo da Forca!***")
    print("**********************************")


    print("Fim do jogo")
```

E caso executemos o programa como programa principal, executamos essa função jogar. Verificamos isso através de um if:

```
if(__name__ == "__main__"):
    jogar()
```

Definindo a palavra secreta

Como no jogo devemos adivinhar uma palavra secreta, nada mais justo do que defini-la em uma variável. Por enquanto a palavra será fixa, com o nome de banana:

```
def jogar():
    print("**********************************")
    print("***Bem vindo ao jogo da Forca!***")
    print("**********************************")


    palavra_secreta = "banana"


    print("Fim do jogo")
```

Mais à frente deixaremos essa palavra secreta mais dinâmica.

Primeiros passos do jogo

Agora, enquanto o usuário estiver jogando, devemos ficar verificando se o usuário acertou a palavra secreta ou se ele perdeu (se "enforcou"), mas devemos fazer isso em quantas iterações?

O usuário continua jogando enquanto ele não acertar a palavra e enquanto ele ainda tiver tentativas para acertar a palavra, ou seja, enquanto ele não "se enforcar". Para representar esse loop, já conhecemos o while:

```
palavra_secreta = "banana"
```

```
while(não acertou E não enforcou):
    print("Jogando...")
```

Mas o que colocamos como condição do while, está em português e não existe em Python, logo isso não funcionará.
No caso de se enforcar, podemos ter dois valores, ou o usuário se enforcou, ou não. Logo, podemos ter os valores verdadeiro ou falso.

O tipo bool
Nas linguagens de programação, e no Python não é diferente, há um tipo específico para representar esses valores, o tipo bool. Ele pode ter os valores True (Verdadeiro) e False (Falso). Então vamos definir uma variável enforcou, que começará com o valor False:

```
palavra_secreta = "banana"

enforcou = False

while(não acertou E não enforcou):
    print("Jogando...")
```

Criaremos também a variável acertou, que representa se o usuário acertou ou não a palavra. Ela também começará com o valor False, pois quando o jogo começa, o usuário ainda não acertou a palavra secreta:

```
palavra_secreta = "banana"

enforcou = False
acertou = False

while(não acertou E não enforcou):
    print("Jogando...")
```

Agora podemos modificar a condição do while, mas a condição é não acertar a palavra e não se enforcar. Para representar o não, no Python existe a palavra chave de negação, o not. E para representar o E, existe o operador lógico and:

```
palavra_secreta = "banana"

enforcou = False
acertou = False

while(not acertou and not enforcou):
    print("Jogando...")
```

Então, enquanto não acertamos a palavra e enquanto não nos enforcamos, continuamos jogando.

Daremos continuação à programação do nosso jogo no próximo capítulo :)

Chegou a hora de colocarmos em prática o que aprendemos neste capítulo. Em linhas gerais, precisamos de uma palavra secreta e vamos começar a criar o game loop, para verificar se o usuário já se enforcou ou acertou.

Para tal, abra o arquivo forca.py:
1) Dentro da função jogar, declare uma nova variável, que guarda a palavra secreta:

```
palavra_secreta = "banana"
```

2) Depois crie mais duas, para guardar o status enforcou e acertou, inicializando-as com False:

```
enforcou = False
acertou = False
```

3) Para o loop, use um laço com condição de entrada, verificando se o usuário não enforcou e nem acertou:

```
while (not acertou and not enforcou):
    print("Jogando...")
```

4) Rode o código no PyCharm e fique atento a possíveis erros. Ao rodar, o jogo fica em um loop infinito.

Na opinião do instrutor você encontrará o código completo.

Segue uma possível implementação, ainda bem simples:

```
def jogar():
    print("**********************************")
    print("***Bem vindo ao jogo da Forca!***")
    print("**********************************")

    palavra_secreta = "banana"

    enforcou = False
    acertou = False

    while(not acertou and not enforcou):
        print("Jogando...")

    print("Fim do jogo")



if(__name__ == "__main__"):
    jogar()
```

No próximo capítulo, vamos capturar e tratar a entrada do usuário. Pronto para continuar?

Já temos o laço do jogo pronto, enquanto o usuário não acertou a palavra secreta e não se enforcou, ele continua jogando. Agora falta implementar a lógica do jogo.

Capturando a entrada do usuário

Ao jogar, o usuário deve digitar uma letra, então devemos pedi-la para ele. Já fizemos isso no treinamento anterior, basta utilizar a função input para capturar a entrada do usuário:

```
while (not acertou and not enforcou):

    chute = input("Qual letra? ")

    print("Jogando...")
```

Com o chute do usuário em mãos, devemos procurar se essa letra existe dentro da palavra secreta. Mas como fazer isso?

Encontrando a posição de uma letra em uma string

A palavra secreta é uma string, tipo str, e nesse tipo há uma função em que podemos procurar algo dentro da string. Essa função é a find, ao passar o chute do usuário para ela, a mesma retorna a posição na palavra em que a letra se encontra, começando da posição 0 (que representa a primeira letra). Por exemplo:

```
>>> palavra = "banana"
>>> palavra.find("b")
0
>>> palavra.find("n")
2
```

Caso a letra não exista na string, nos é retornado o número -1:

```
>>> palavra = "banana"
>>> palavra.find("z")
-1
```

Mas podemos reparar em um problema da função, pois ao pesquisar pela letra n, foi retornado o número 2, mas a letra n também existe na posição 4 da palavra, porém a função não faz isso, ela só retorna uma única posição, a primeira que ela encontrar. Por isso não vamos utilizar essa função. Então o que faremos?

Iterando sobre a palavra

Resolveremos isso iterando sobre a palavra secreta. Faremos um laço em cima de cada letra, no nosso caso, na primeira iteração teremos a letra b, na segunda teremos a letra a, depois a n e assim até terminar a palavra. Com a letra em mãos, basta comparar com o chute do usuário.
Mas aí entramos em outra questão: como fazemos um laço em cima de uma palavra?

A palavra é uma sequência de letras, logo podemos utilizar o laço `for`! Por exemplo:

```
>>> palavra = "banana"
>>> for letra in palavra:
...     print(letra)
...
b
a
n
a
n
a
```

Faremos isso no nosso jogo. Para cada letra da palavra, comparamos com o chute do usuário, e se o chute for igual à letra, podemos imprimi-lo:

```
while (not acertou and not enforcou):

    chute = input("Qual letra? ")

    for letra in palavra_secreta:
        if (chute == letra):
            print(chute)

    print("Jogando...")
```

Agora, ao executar o programa e procurar pela letra a, vemos o seguinte resultado:

```
*********************************
***Bem vindo ao jogo da Forca!***
*********************************
Qual letra? a
a
a
a
Jogando...
```

Podemos melhorar ainda mais, exibindo a posição do chute na palavra, assim como a função `find` fazia.

Exibindo a posição da letra

Temos que fazer isso na mão, então, fora do `for`, vamos criar a variável `index`, com o valor 0. E dentro do `for`, vamos incrementar essa variável a cada iteração:

```
while (not acertou and not enforcou):

    chute = input("Qual letra? ")

    index = 0
    for letra in palavra_secreta:
        if (chute == letra):
            print(chute)
        index = index + 1

    print("Jogando...")
```

Com a letra e a posição em mãos, vamos imprimi-los, utilizando a interpolação de strings:

```
while (not acertou and not enforcou):

    chute = input("Qual letra? ")

    index = 0
    for letra in palavra_secreta:
        if (chute == letra):
            print("Encontrei a letra {} na posição {}".format(letra, index))
        index = index + 1

    print("Jogando...")
```

Ao executar novamente o nosso jogo:

```
**********************************
***Bem vindo ao jogo da Forca!***
**********************************
Qual letra? n
Encontrei a letra n na posição 2
```

```
Encontrei a letra n na posição 4
Jogando...
```

Mas se digitarmos uma letra em maiúsculo, por exemplo a letra A, a mesma não é encontrada na palavra. O ideal é não fazermos essa diferenciação entre letras minúsculas e maiúsculas. Vamos resolver esse problema no próximo capítulo :)

Ao procurar por uma letra maiúscula, a mesma não é encontrada na palavra, por exemplo:

```
**********************************
***Bem vindo ao jogo da Forca!***
**********************************
Qual letra? N
Jogando...
```

Não queremos que essa distinção entre letras minúsculas e maiúsculas seja feita.

Alterando a caixa da string

Já vimos algumas funções que a string possui, como a `format` e `find`, mas há diversas outras, como podemos ver na documentação do Python 3. Por exemplo, existe a função `capitalize`, que retorna a string com a primeira letra em maiúsculo e o restante em minúsculo.

Existe também a função `lower`, que retorna a string com todas as letras em minúsculo:

```
>>> palavra = "BANANA"
>>> palavra.lower()
'banana'
```

Antagonicamente, existe a função `upper`, que retorna a string com todas as letras em maiúsculo:

```
>>> palavra = "banana"
>>> palavra.upper()
'BANANA'
```

Então, ao compararmos o chute do usuário com a letra da palavra secreta, podemos comparar as duas strings em letras maiúsculas, assim não haverá distinção na hora da comparação:

```
while (not acertou and not enforcou):

    chute = input("Qual letra? ")

    index = 0
    for letra in palavra_secreta:
        if (chute.upper() == letra.upper()):
            print("Encontrei a letra {} na posição {}".format(letra, index))
```

```
        index = index + 1


    print("Jogando...")
```

Problema resolvido! Mas o que acontece se, ao digitar o chute, o usuário digitar espaços em branco no início ou no fim da letra?

Removendo espaços em branco no início e no fim de uma string

Ao digitar a letra com espaços em branco no seu início, vejamos o que acontece:

```
**********************************
***Bem vindo ao jogo da Forca!***
**********************************
Qual letra?         n
Jogando...
```

A letra não é reconhecida! Então precisamos remover todos os espaços no início e no fim do chute do usuário. E para isso existe a função `strip`, que faz exatamente isso:

```
>>> palavra = "  banana   "
>>> palavra.strip()
'banana'
```

Logo, após capturar o chute do usuário, vamos chamar essa função, atribuindo o seu retorno à variável `chute`:

```
while (not acertou and not enforcou):


    chute = input("Qual letra? ")
    chute = chute.strip()


    index = 0
    for letra in palavra_secreta:
        if (chute.upper() == letra.upper()):
            print("Encontrei a letra {} na posição {}".format(letra, index))
        index = index + 1


    print("Jogando...")
```

Podemos executar o programa novamente, e digitar uma letra com alguns espaços em branco no seu início:

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

```
**********
Qual letra?  N
Encontrei a letra n na posição 2
Encontrei a letra n na posição 4
Jogando...
```

Agora a letra é encontrada na palavra, mesmo com os espaços a mais!

Chegou a hora de colocarmos em prática o que aprendemos neste capítulo. Em linhas gerais, dentro do loop, devemos perguntar ao jogador o seu chute da palavra secreta. Por enquanto, exibiremos apenas se o chute foi encontrado ou não, inclusive sua posição na string.

Todas as alterações acontecem dentro do arquivo forca.py.
1) Dentro de um loop while, nosso famoso game loop, vamos perguntar qual a letra do chute. Só não esqueça de usar a função .strip() para remover espaços em branco à direita e à esquerda da letra digitada:

```
while (not acertou and not enforcou):
    chute = input("Qual letra? ")
    chute = chute.strip()
```

2) Agora que já temos o chute do jogador, podemos compará-lo com cada letra de palavra_secreta. Lembre-se que uma string pode ser iterada através de for. Para evitarmos diferenças entre maiúsculas e minúsculas, faça com que a comparação considere tanto o chute quanto a letra iterada em maiúscula. Por fim, crie a variável index para guardar a posição da letra encontrada, pois no final queremos exibir essa informação para o usuário:

```
while (not acertou and not enforcou):


    chute = input("Qual letra? ")
    chute = chute.strip()


    index = 0
    for letra in palavra_secreta:
        if (chute.upper() == letra.upper()):
            print("Encontrei a letra {} na posição {}".format(letra, index))
        index = index + 1
    print("Jogando...")
```

3) Rode e teste o seu código. Ainda estamos executando um loop infinito e por isso é preciso terminar cada execução do jogo explicitamente pelo PyCharm.
Na opinião do instrutor você encontrará o código completo.

O código final do nosso programa fica assim:

```
def jogar():
    print('*****************************************')
```

```
    print('****** Bem-vindo ao jogo da Forca *******')
    print('****************************************')

    palavra_secreta = "banana"
    enforcou = False
    acertou = False

    while (not acertou and not enforcou):

        chute = input("Qual letra? ")
        chute = chute.strip()

        index = 0
        for letra in palavra_secreta:
            if (chute.upper() == letra.upper()):
                print("Encontrei a letra {} na posição {}".format(letra, index))
            index = index + 1

        print("Jogando...")

    print("Fim do jogo")

if(__name__ == '__main__'):
    jogar()
```

Atualmente, já dizemos ao usuário em que posição a letra que ele chutou está na palavra secreta, caso a letra exista na palavra. Mas em um jogo real de forca, o jogador vê quantas letras há na palavra secreta. Algo como:

```
Qual letra? _ _ _ _ _ _
```

E se ele encontrar alguma letra, a mesma tem a sua lacuna preenchida. Ao digitar a letra "a", ficaria:

```
_ a _ a _ a
```

Muito mais intuitivo, não? Vamos implementar essa funcionalidade.

Conhecendo uma nova estrutura de dados: lista
Para exibir as letras dessa forma, precisamos guardar os chutes certos do usuário, mas como fazer isso?

Para tal, o Python nos oferece uma estrutura de dados que nos permite guardar valores. Essa estrutura é a lista. Para criar uma lista, utilizamos colchetes ([]):

```
>>> valores = []
>>> type(valores)
<class 'list'>
```

Assim como a string, list também é uma sequência de dados. Então podemos ver a sua documentação. Nela, podemos ver o que podemos fazer com uma lista, podemos verificar o seu valor mínimo com o min e o seu valor máximo com o max. Mas a nossa lista ainda tá vazia, certo? Podemos, já no momento de sua inicialização, passar valores para guardar nessa lista:

```
>>> valores = [0,1,2,3,4]
```

Agora podemos verificar seu menor e seu maior valor:

```
>>> valores = [0,1,2,3,4]
>>> min(valores)
0
>>> max(valores)
4
```

Para acessar um valor específico, podemos acessá-lo através do seu índice. O primeiro elemento da lista possui o índice 0, o segundo possui o índice 1 e assim por diante:

```
>>> valores = [0,1,2,3,4]
>>> valores[2]
2
```

Podemos também saber o tamanho da lista com o len e verificar se determinado valor está guardado nela:

```
>>> valores = [0,1,2,3,4]
>>> 0 in valores
True
>>> 8 in valores
False
```

Além disso, existem funções específicas da lista, que podem ser vistas aqui. Podemos adicionar elementos ao final da lista com o append, exibir e remover um elemento de determinada posição com o pop, entre diversas outras funcionalidades.
Agora que sabemos como guardar valores em uma lista, podemos voltar ao nosso jogo e guardar os acertos do usuário. Faremos isso no próximo vídeo :)

Agora que já conhecemos um pouco como a lista funciona, chegou a hora de utilizá-la no nosso jogo. Existem diversas formas de implementar essa funcionalidade, mas aqui faremos da seguinte

forma: como queremos exibir os espaços vazios primeiro, criaremos uma lista com eles, na mesma quantidade de letras da palavra secreta:

```
palavra_secreta = "banana"
letras_acertadas = ["_", "_", "_", "_", "_", "_"]
```

Atualmente isso tudo está fixo, caso queiramos mudar a palavra secreta, teremos que mudar os espaços vazios. Mas não se preocupe, tornaremos isso dinâmico mais à frente.

Adicionando as letras acertadas à lista

Já temos a posição da letra, que chamamos de `index`. Logo, caso o chute seja correto, basta guardar e letra dentro da lista, na sua posição correta:

```
while (not acertou and not enforcou):

    chute = input("Qual letra? ")
    chute = chute.strip()

    index = 0
    for letra in palavra_secreta:
        if (chute.upper() == letra.upper()):
            letras_acertadas[index] = letra
        index = index + 1

    print("Jogando...")
```

E após adicionar a letra, após o `for`, imprimimos a nossa lista:

```
while (not acertou and not enforcou):

    chute = input("Qual letra? ")
    chute = chute.strip()

    index = 0
    for letra in palavra_secreta:
        if (chute.upper() == letra.upper()):
            letras_acertadas[index] = letra
        index = index + 1

    print(letras_acertadas)
```

Podemos executar o jogo, ao acertar uma letra, a palavra é exibida com a mesma preenchida:

```
**********************************
***Bem vindo ao jogo da Forca!***
**********************************
Qual letra? b
['b', '_', '_', '_', '_', '_']
Qual letra? a
['b', 'a', '_', 'a', '_', 'a']
```

A saída ainda não está visualmente agradável, mas melhoraremos isso. Para ficar ainda melhor, vamos exibir a lista no início do jogo também:

```
print(letras_acertadas)


while (not acertou and not enforcou):


    chute = input("Qual letra? ")
    chute = chute.strip()


    index = 0
    for letra in palavra_secreta:
        if (chute.upper() == letra.upper()):
            letras_acertadas[index] = letra
        index = index + 1


    print(letras_acertadas)
```

Com isso, avançamos mais ainda na implementação do nosso jogo!

Vamos aproveitar o nosso novo conhecimento em listas para fazer com que a nossa forca se lembre das letras acertadas pelo nosso jogador.

1- De início crie uma nova lista chamada `letras_acertadas` abaixo da variável `palavra_secreta`. Aproveite e inicie esta lista com 6 elementos do caractere "_", para representar uma letra faltando. Por enquanto estamos fazendo com um número fixo de letras, mas em breve melhoraremos isto também.

```
letras_acertadas = ["_", "_", "_", "_", "_", "_"]
```

2- Já sabemos quando um usuário acerta uma letra, afinal fazemos isto no if que já temos:

```
if(chute.upper() == letra.upper()):
    print("Encontrei a letra {} na posição {}".format(chute, index))
```

Mas agora, em vez de apenas imprimirmos uma mensagem ao acertar, vamos substituir no nosso array de letras faltando. Como já temos o índice da letra, basta substituir naquela posição do array pela letra que acertamos:

```
if (chute.upper() == letra.upper()):
    letras_acertadas[index] = letra
```

3- Para que o jogador acompanhe o resultado a cada chute que ele der, após o laço for imprima também a lista de letras_acertadas para que ele veja como ele está indo no jogo:

```
for letra in palavra_secreta:
    ...
    ...
print(letras_acertadas)
```

4- E claro, para dar uma dica ao nosso jogador de quantas letras a palavra tem, vamos colocar acima do while um print inicial para que ele veja de início qual o tamanho da palavra:

```
print(letras_acertadas)


while (not acertou and not enforcou):
...
```

Faça o teste e veja na resposta se seu código está batendo com o do instrutor.

O seu código final deve estar parecido com este:

```
def jogar():
    print("*********************************")
    print("***Bem vindo ao jogo da Forca!***")
    print("*********************************")

    palavra_secreta = "banana"
    letras_acertadas = ["_", "_", "_", "_", "_", "_"]

    enforcou = False
    acertou = False

    print(letras_acertadas)

    while(not enforcou and not acertou):

        chute = input("Qual letra? ")
        chute = chute.strip()

        index = 0
        for letra in palavra_secreta:
```

```
            if(chute.upper() == letra.upper()):
                letras_acertadas[index] = letra
            index = index + 1

        print(letras_acertadas)

    print("Fim do jogo")

if(__name__ == "__main__"):
    jogar()
```

Além das funções min(), max() e len() que vimos neste capítulo, as listas do Python tem outros recursos que facilitam nossa vida. Vamos conhecer alguns deles:

A função .count()

Um jeito fácil de contar o número de ocorrências de um determinado elemento em uma lista é a função .count() das listas, veja:

```
valores = [ 0, 0, 0, 1, 2, 3, 4]
print(valores.count(0))
```

O código acima nos retorna 3, pois em nossa lista encontramos 3 vezes o número zero nela.
Utilizando a função .count() podemos por exemplo, detectar quantas letras ainda faltam para o nosso usuário preencher:

```
letras_acertadas = ['_','_','_','_','_','_','_']
letras_faltando = str(letras_acertadas.count('_'))
print( 'Ainda faltam acertar {} letras'.format(letras_faltando))
```

A função .index()

Uma outra função que pode ser bastante útil é a função .index(), que nos retorna o índice da primeira ocorrência de um determinado elemento em uma lista, veja:

```
frutas = ['Banana', 'Morango', 'Maçã', 'Uva', 'Maçã', 'Uva']
print(frutas.index('Uva'))
```

O código acima nos retorna 3, pois é o índice da primeira ocorrência do elemento 'Uva' na lista frutas (lembre-se nas listas começamos a contar do índice 0).
Só tome cuidado quando utilizar a função .index(), pois a mesma pode retornar um erro caso você tente buscar pelo índice de um elemento que não existe. Veja o caso abaixo:

```
frutas = ['Banana', 'Morango', 'Maçã', 'Uva']
print(frutas.index('Melancia'))
```

Ao tentar buscar pela fruta 'Melancia', obteremos o erro "ValueError: 'Melancia' is not in list" no console, já que é impossível buscar o índice de algo que não está na lista. Por isto, é sempre uma boa prática verificar se o elemento está na lista com o operador in antes de obter o seu índice. Um código ideal que faz uso da função index() é demonstrado abaixo:

```
frutas = ['Banana', 'Morango', 'Maçã', 'Uva']


fruta_buscada = 'Melancia'
if fruta_buscada in frutas:
    print(frutas.index(fruta_buscada))
else:
    print('Desculpe, a {} não está na lista frutas'.format( fruta_buscada))
```

Assim temos certeza que a fruta_buscada está dentro da lista antes de perguntarmos o seu índice, evitando assim de receber um erro no console.

No capítulo anterior, vimos que strings e listas são tipos de sequências. Além dessas sequências, também há o range, que vimos no curso anterior, e a tupla, que veremos agora.

Conhecendo mais uma sequência, a tupla

Nós gostaríamos de definir os dias da semana, então podemos defini-los em uma lista:

```
>>> dias = ["Domingo", "Segunda", "Terça", "Quarta", "Quinta", "Sexta", "Sábado"]
```

Faz sentido apagar um dia da semana? Provavelmente não, já que a semana sempre possui sete dias. Assim como também não faz sentido criarmos um novo dia da semana, mas nada nos impede de criarmos um:

```
>>> dias = ["Domingo", "Segunda", "Terça", "Quarta", "Quinta", "Sexta", "Sábado"]
>>> dias.append("Sábado2")
>>> dias
['Domingo', 'Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta', 'Sábado', 'Sábado2']
```

Portanto, essa lista deveria ser fixa, inalterável. Para isso existe o tuple, que é uma lista imutável.

Criando uma tupla

Para criar uma tupla, é bem simples. É do mesmo jeito que criamos uma lista, mas ao invés de usar colchetes, usamos parênteses:

```
>>> dias = ("Domingo", "Segunda", "Terça", "Quarta", "Quinta", "Sexta", "Sábado")
>>> type(dias)
<class 'tuple'>
```

Agora não conseguimos adicionar, nem remover elementos. Por exemplo:

```
>>> dias = ("Domingo", "Segunda", "Terça", "Quarta", "Quinta", "Sexta", "Sábado")
>>> dias.append("Sábado2")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
```

Tuplas dentro de uma lista

Agora queremos armazenar instrutores, cada um com o seu nome e sua idade:

```
>>> instrutor1 = ("Nico", 39)
>>> instrutor2 = ("Flávio", 37)
```

Agora podemos criar uma lista de instrutores:

```
>>> instrutor1 = ("Nico", 39)
>>> instrutor2 = ("Flávio", 37)
>>> instrutores = [instrutor1, instrutor2]
```

Isso mesmo! A lista pode armazenar tuplas dentro dela. Ao imprimi-la, vemos:

```
>>> instrutores
[('Nico', 39), ('Flávio', 37)]
```

Podemos acessar somente um tupla pelo seu índice na lista:

```
>>> instrutores[0]
('Nico', 39)
```

E podemos também acessar, através da lista, somente um elemento da tupla. Por exemplo, para acessar a idade do Nico, acessamos a sua tupla, e acessamos a sua idade passando mais um índice:

```
>>> instrutores[0][1]
39
```

E podemos fazer o contrário também, podemos colocar listas dentro de tuplas :)

Convertendo listas em tuplas

Agora temos a seguinte situação: precisamos ler de um arquivo, mas não sabemos quantas linhas esse arquivo tem. Então, vamos lendo linha por linha, e adicionando-as em uma lista:

```
>>> linhas = []
>>> linhas.append("linha 1")
>>> linhas.append("linha 2")
>>> linhas.append("linha 3")
```

Em algum momento o arquivo irá acabar, e quando esse momento chegar, não queremos mais adicionar linhas à lista. Então, devemos transformar a lista em uma lista imutável, no caso uma

tupla. E o Python possui uma função específica para isso, a tuple(), que recebe por parâmetro a lista a ser convertida:

```
>>> linhas_tuple = tuple(linhas)
>>> type(linhas_tuple)
<class 'tuple'>
>>> linhas_tuple
('linha 1', 'linha 2', 'linha 3')
```

Convertendo tuplas em listas

Com uma simples função, transformamos uma lista em uma tupla! E o contrário também pode ser feito, com a função list():

```
>>> linhas_list = list(linhas_tuple)
>>> type(linhas_list)
<class 'list'>
>>> linhas_list
['linha 1', 'linha 2', 'linha 3']
```

Quando uma sequência não é suficiente

Já aprendemos algumas características das sequências. Elas guardam valores de qualquer tipo de dado e possuem uma determinada ordem, pois possuem um índice. As sequências também são chamadas de coleções e são o arroz e feijão para qualquer desenvolvedor Python!
Agora dá uma olhada no exemplo abaixo onde criamos uma lista de CPFs:

```
lista = [11122233344, 22233344455, 33344455566]
```

Nada errado aqui, mas imagine agora que você precisa evitar que exista algum CPF duplicado nesta lista. Isso é algo muito comum no dia-a-dia! Em muitas circunstâncias precisamos de uma coleção que não permita valores duplicados, mas nada nos impede adicionar um mesmo CPF nessa lista, por exemplo:

```
lista.append(11122233344) #funciona!
```

Isso também funciona se fosse uma tuple! Uma tuple também permite elementos duplicados!

Conhecendo o set

E agora? Será que o Python não oferece alguma coleção onde não podem existir elementos duplicados? Claro que existe uma coleção com esse propósito e ela se chama set.
Veja o mesmo exemplo, mas agora inicializando um set:

```
colecao = {11122233344, 22233344455, 33344455566}
```

Repare que usamos {} chaves para declarar os elementos iniciais. Pouca diferença comparando com as sequências, não?

Adicionando elementos no set
Para adicionar um elemento a um set devemos chamar a função add (invés da append):

```
colecao.add(44455566677) #vai adicionar pois não existe ainda
```

E se usarmos um CPF que já existe? Não deve funcionar, certo? Vamos testar,e ver que o set vai ignorá-lo:

```
colecao.add(11122233344) #nao vai adicionar pois este CPF já existe!
```

set não possui um índice
É importante notar que o set não é uma sequência, pois não tem um índice. O código abaixo não funciona:

```
colecao[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support indexing
```

Isso mesmo, não tem índice. E como não temos um índice não sabemos em qual ordem vem os valores quando imprimimos um set de dentro de um laço for:

```
for cpf in colecao:
    print(cpf)
```

Imprime:

```
11122233344
44455566677
33344455566
22233344455
```

Repare que os elementos foram listados fora da ordem de inserção. Isso acontece porque o set não é ordenado.

Resumindo
Um set é uma coleção não ordenada de elementos. Cada elemento é único, isso significa que não existem elementos duplicados dentro do set.
Respire fundo e fique tranquilo pois o set será abordado ainda com mais detalhe em outros cursos.
Vamos continuar?

Você aguenta aprender mais um pouco sobre coleções? Então vamos lá :)

Vamos lembrar rapidamente do último exemplo no vídeo quando misturamos as listas e tuples.
Primeiro criamos pessoas com nome e idade usando tuples:

```
pessoa1 = ("Nico", 39)
pessoa2 = ("Flavio", 37)
```

```
pessoa3 = ("Marcos", 30)
```

Depois guardamos as tuples dentro de uma lista:

```
instrutores = [pessoa1, pessoa2, pessoa3]
```

Se imprimimos instrutores recebemos:

```
[('Nico', 39), ('Flavio', 37), ('Marcos', 30)]
```

Para saber a idade do Flavio, devemos usar os índices (primeiro da lista, depois da tuple). Lembrando que acessamos o índice com os []:

```
instrutores[1][1]
```

```
37
```

Agora vem o problema: E se não sabemos a posição do Flavio? Em geral, como podemos descobrir a idade de um instrutor sem saber a posição dele?

Instrutor pelo nome

Repare que temos, na nossa coleção, sempre um nome associado com a idade. Sempre temos um par de valores, aqui é:

```
nome : idade
```

Invés de usar a posição gostaria de descobrir a idade através do nome, algo assim:

```
instrutores['Flavio']
```

Só que isso não vai funcionar com lista, nem com tuple, nem combinando os dois :(É preciso usar uma nova estrutura de dados, o Dictionary!

Conhecendo o dictionary

Para criar um dicionário devemos inicializar os instrutores de um modo um pouco diferente. Veja o código:

```
instrutores = {'Nico' : 39, 'Flavio': 37, 'Marcos' : 30}
```

Repare que usamos as chaves {} (como se fosse um set), mas sempre tem em pares. O primeiro par é 'Nico' : 39, o segundo é 'Flavio': 37, etc.

Nesse par, temos no lado esquerdo a chave e no lado direito o valor. Isso é importante pois usamos a chave para recuperar um valor e assim resolvemos nosso problema:

```
instrutores['Flavio']
```

Imprime:

```
37
```

Isso foi apenas uma introdução, mas não se preocupe pois veremos ainda mais sobre dicionários dentro da carreira Python 3.

Agora vamos dar continuidade ao nosso jogo, pois ele ainda não está totalmente funcional, pois ao acertar a palavra secreta, o jogo ainda fica pedindo que o usuário digite uma letra:

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

```
**********
['_', '_', '_', '_', '_', '_']
Qual letra? b
['b', '_', '_', '_', '_', '_']
Qual letra? a
['b', 'a', '_', 'a', '_', 'a']
Qual letra? n
['b', 'a', 'n', 'a', 'n', 'a']
Qual letra?
```

O contrário também acontece, podemos errar várias vezes, até digitar quase o alfabeto inteiro, se quisermos. Ou seja, o usuário possui infinitas tentativas para descobrir a palavra, já que não há um limite de tentativas.

Encerrando o jogo quando o usuário errar seis vezes

Como os valores das variáveis `enforcou` e `acertou` nunca mudam, o loop é infinito. Logo, precisamos mudar os seus valores de acordo com situações do jogo.

Vamos começar com a variável `enforcou`. Quando é que o usuário se enforca? Vamos definir que é quando o jogador errar 6 vezes, então precisamos contar esses erros. Logo, vamos declarar uma variável `erros` e inicializá-la com o valor `0`:

```
enforcou = False
acertou = False
erros = 0
```

Agora, se o usuário errou, nós incrementamos a variável. Para tal, precisamos realizar um teste. Vamos testar se o chute está na palavra secreta, se estiver nós executamos o código que posiciona a letra na lista, etc. Mas se o chute não estiver na palavra secreta, incrementamos a variável `erros`:

```
while (not acertou and not enforcou):

    chute = input("Qual letra? ")
    chute = chute.strip()

    if (chute in palavra_secreta):
        index = 0
        for letra in palavra_secreta:
            if (chute.upper() == letra.upper()):
                letras_acertadas[index] = letra
            index = index + 1
    else:
```

```
        erros = erros + 1


    print(letras_acertadas)
```

Ao final do while, vamos verificar se a variável erros é igual a 6** e atribuir esse booleano (true** ou false) à variável enforcou:

```
while (not acertou and not enforcou):


    chute = input("Qual letra? ")
    chute = chute.strip()

    if (chute in palavra_secreta):
        index = 0
        for letra in palavra_secreta:
            if (chute.upper() == letra.upper()):
                letras_acertadas[index] = letra
            index = index + 1
    else:
        erros = erros + 1

    enforcou = erros == 6
    print(letras_acertadas)
```

Agora, ao executarmos o jogo e errar 6 vezes, o jogo é encerrado! Para melhorar ainda mais o nosso código, podemos seguir a dica do PyCharm e realizar o incremento das variáveis index e erros da seguinte forma:

```
while (not acertou and not enforcou):


    chute = input("Qual letra? ")
    chute = chute.strip()

    if (chute in palavra_secreta):
        index = 0
        for letra in palavra_secreta:
            if (chute.upper() == letra.upper()):
                letras_acertadas[index] = letra
            index += 1
    else:
```

```
        erros += 1


    enforcou = erros == 6
    print(letras_acertadas)
```

O resultado é o mesmo.

Palavra secreta e chute sempre em caixa alta

Mas agora, se testarmos um chute com letra maiúscula?

```
**********************************
***Bem vindo ao jogo da Forca!***
**********************************
['_', '_', '_', '_', '_', '_']
Qual letra? B
['_', '_', '_', '_', '_', '_']
Qual letra?
```

Ué, já não tínhamos resolvido isso? Acontece que no `if` que acabamos de escrever, nós não fazemos a comparação do chute com as letras em caixa alta. Para não termos que sempre nos lembrar disso, vamos já definir a palavra secreta e o chute em letra maiúsculas. Assim, nas comparações não precisamos mais deixar todas as letras em caixa alta. O código ficará assim:

```
def jogar():

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

    palavra_secreta = "banana".upper()
    letras_acertadas = ["_", "_", "_", "_", "_", "_"]

    enforcou = False
    acertou = False
    erros = 0

    print(letras_acertadas)

    while (not acertou and not enforcou):
```

```
    chute = input("Qual letra? ")
    chute = chute.strip().upper()

    if (chute in palavra_secreta):
        index = 0
        for letra in palavra_secreta:
            if (chute == letra):
                letras_acertadas[index] = letra
            index += 1
    else:
        erros += 1

    enforcou = erros == 6
    print(letras_acertadas)

print("Fim do jogo")
```

Por fim, falta encerrarmos o jogo quando o jogador acertar a palavra.

Encerrando o jogo quando o usuário acertar a palavra

Uma das maneiras de implementar essa funcionalidade é aproveitar a nossa lista de letras acertadas, já que enquanto o jogador não acertar a palavra, o caractere `_` vai existir na lista.

Logo, enquanto a lista possui o `_`, significa que a palavra ainda não foi totalmente preenchida, já que os `_` são substituídos a cada letra acertada. Então vamos adicionar essa verificação abaixo da verificação da variável `enforcou`:

```
enforcou = erros == 6
acertou = "_" not in letras_acertadas
```

Ao testar, e acertar a palavra, temos o seguinte resultado:

```
**********************************
***Bem vindo ao jogo da Forca!***
**********************************
['_', '_', '_', '_', '_', '_']
Qual letra? b
['B', '_', '_', '_', '_', '_']
Qual letra? a
['B', 'A', '_', 'A', '_', 'A']
```

```
Qual letra? n
['B', 'A', 'N', 'A', 'N', 'A']
Fim do jogo
```

O jogo é encerrado ao acertarmos a palavra! Ótimo, funcionalidade concluída. Falta apenas exibir uma mensagem ao usuário, dizendo se ele acertou ou não a palavra. Após o `while`, adicionamos:

```
if (acertou):
    print("Você ganhou!")
else:
    print("Você perdeu!")
```

No próximo vídeo iremos melhorar a declaração da quantidade de `_` na lista de palavras acertadas, já que por enquanto a mesma está fixa.

Atualmente, a nossa palavra secreta é banana, que possui seis letras. Por isso a lista de palavras acertadas possui seis `_`. Mas se trocarmos a palavra secreta? Precisamos alterar a quantidade de `_` na lista, de acordo com o número de letras da nova palavra.

Meio trabalhoso, né? Então vamos tornar isso mais dinâmico.

Inicializando a lista de acordo com o número de letras da palavra

Queremos que a inicialização da lista de palavras acertadas seja dinâmica, baseada na quantidade de letras que houver na palavra secreta. Já sabemos implementar isso, podemos criar a lista vazia, e para cada letra na palavra secreta, adicionamos um `_` à ela:

```
palavra_secreta = "banana".upper()
letras_acertadas = []

for letra in palavra_secreta:
    letras_acertadas.append("_")
```

Perfeito, está funcionando! Mas há uma forma mais interessante de fazer isso.

Realizando um laço dentro da inicialização da lista

Quando inicializamos a lista, queremos inicializá-la com um caractere `_` para cada letra na palavra secreta. Só que com Python, podemos fazer isso diretamente, dentro da inicialização da lista:

```
palavra_secreta = "banana".upper()
letras_acertadas = ["_" for letra in palavra_secreta]
```

Essa funcionalidade do Python se chama List Comprehension, em português, Compreensão de lista.

No jogo da forca, implemente a lógica para que o jogo dê apenas 6 chances para o jogador tentar acertar a palavra. Primeiro, crie uma variável chamada `erro` fora do loop e inicialize-a com zero.

```
erros = 0
```

1.Em seguida, coloque uma condição dentro do loop para verificar se o jogador acertou a letra. Ela deve englobar a inicialização da variável index e o loop for. Crie um else para incrementar a variável erros.

```
if(chute in palavra_secreta):
    index = 0
    for letra in palavra_secreta:
        if(chute == letra):
            letras_acertadas[index] = letra
        index = index + 1
else:
    erros = erros + 1
```

2.No final do loop, atualize a variável enforcou para que ela se torne verdadeira quando a quantidade de erros for 6. Teste o jogo colocando 6 letras erradas e verifique que o jogo termina!

```
enforcou = erros == 6
```

3.Agora implemente a lógica para que o jogo termine quando o jogador acertar a palavra. Crie mais uma linha no final do loop para atualizar a variável acertou com a verificação se o caracter underscore não existe em letras_acertadas. Teste a aplicação e coloque as letras da palavra secreta. Confirme que o jogo termina.

```
acertou = "_" not in letras_acertadas
```

4.Apesar do jogo encerrar, o jogador ainda não sabe se acertou ou foi enforcado. Após o loop e antes da impressão da mensagem Fim do jogo, coloque um if para imprimir Você ganhou!! caso ele tenha vencido a forca e Você perdeu! caso contrário. Teste novamente sua aplicação.

```
if(acertou):
    print("Você ganhou!!")
else:
    print("Você perdeu!!")
```

5.Inicialize a variável letras_acertadas dinamicamente a partir de qualquer palavra secreta. Utilize o recurso List Comprehension para criar um laço dentro da inicialização da lista.

```
letras_acertadas = ["_" for letra in palavra_secreta]
```

Seu código deverá ficar parecido com o abaixo:

```
def jogar():
    print("*********************************")
    print("***Bem vindo ao jogo da Forca!***")
    print("*********************************")


    palavra_secreta = "maça".upper()
```

```
    letras_acertadas = ["_" for letra in palavra_secreta]

    enforcou = False
    acertou = False
    erros = 0

    print(letras_acertadas)

    while(not enforcou and not acertou):

        chute = input("Qual letra? ")
        chute = chute.strip().upper()

        if(chute in palavra_secreta):
            index = 0
            for letra in palavra_secreta:
                if(chute == letra):
                    letras_acertadas[index] = letra
                index += 1
        else:
            erros += 1

        enforcou = erros == 6
        acertou = "_" not in letras_acertadas
        print(letras_acertadas)


    if(acertou):
        print("Você ganhou!!")
    else:
        print("Você perdeu!!")
    print("Fim do jogo")

if(__name__ == "__main__"):
    jogar()
```

Uma funcionalidade que ainda nos atrapalha no nosso jogo é a palavra secreta, que atualmente está fixa. Se queremos que a palavra seja diferente, devemos modificá-la no código.

A nossa ideia é ler palavras de um arquivo de texto, e dentre elas escolhemos uma palavra aleatoriamente, que será a palavra secreta do jogo

Escrita de um arquivo

Para abrir um arquivo, o Python possui a função built-in `open`. Ela recebe dois parâmetros: o primeiro é o nome do arquivo a ser aberto, e o segundo parâmetro é o modo que queremos trabalhar com esse arquivo, se queremos ler ou escrever. O modo é passado através de uma string: `"w"` para escrita e `"r"` para leitura.

No nosso jogo, faremos a leitura de um arquivo, mas vamos ver antes no terminal do Python 3 como funciona a escrita:

```
>>> arquivo = open("palavras.txt", "w")
```

Vale lembrar que o `w` sobrescreve o arquivo, se o mesmo existir. Se só quisermos adicionar conteúdo ao arquivo, utilizamos o `a`.

Agora que temos o arquivo, como escrevemos nele? Basta chamar no arquivo a função `write`, passando para ela o que queremos escrever no arquivo:

```
>>> arquivo.write("banana")
6
>>> arquivo.write("melancia")
8
```

O retorno dessa função é o número de caracteres que adicionamos no arquivo.

Fechando um arquivo

Quando estamos trabalhando com arquivos, devemos nos preocupar em fechá-lo. Assim como abrimos um arquivo, devemos fechá-los, chamando a função `close`:

```
>>> arquivo.close()
```

Após isso, podemos verificar o conteúdo do arquivo, ele foi criado na mesma pasta em que o comando para a abrir o console do Python 3 foi executado. Ao verificar o seu conteúdo, vemos:

```
bananamelancia
```

As palavras foram escritas em uma mesma linha. Mas como escrever em uma nova linha?

Escrevendo palavras em novas linhas

A primeira coisa que devemos fazer é abrir o arquivo novamente, dessa vez utilizando o `a`, de append:

```
>>> arquivo = open("palavras.txt", "a")
```

Podemos escrever novamente no arquivo, mas dessa vez com a preocupação de criar uma nova linha após o que vamos escrever. Para representar uma nova linha em código, adicionamos o \n ao final do que queremos escrever:

```
>>> arquivo.write("morango\n")
8
>>> arquivo.write("manga\n")
6
```

Ao fechar o arquivo e verificar novamente o seu conteúdo, vemos:

```
bananamelanciamorango
manga
```

A palavra morango ainda ficou na mesma linha, mas como especificamos na sua adição que após a palavra deverá ter uma quebra de linha, a palavra manga foi adicionada abaixo, em uma nova linha. Por fim, vamos mover esse arquivo para dentro do nosso projeto, e ajeitar as suas palavras, quebrando as linhas. Ele ficará assim:

```
banana
melancia
morango
manga
```

Ainda no terminal do Python 3, vamos ver o funcionamento da leitura de um arquivo. Como agora o arquivo palavras.txt está na pasta do projeto jogos, devemos executar o comando que abre o terminal do Python 3 na pasta do projeto.

Leitura de um arquivo

Vamos então abrir o arquivo no modo de leitura, basta passar o nome do arquivo e a letra r para a função open, como já vimos no vídeo anterior:

```
>>> arquivo = open("palavras.txt", "r")
```

Como abrimos o arquivo no modo de leitura, a função write não funciona. Para ler o arquivo inteiro, utilizamos a função read:

```
>>> arquivo.read()
'banana\nmelancia\nmorango\nmanga\n'
```

Mas se executarmos a função novamente:

```
>>> arquivo.read()
''
```

Nos é retornado uma string vazia. Por quê?

O arquivo é como um fluxo de linhas, que começa no início do arquivo, como se fosse o ponteiro. Ele vai descendo e lendo arquivo, após ler tudo, ele fica posicionado no final do arquivo, então quando chamamos a função read() novamente, não há mais conteúdo, pois ele todo já foi lido.
Ou seja, para ler o arquivo novamente, devemos fechá-lo e abri-lo novamente.

Lendo linha por linha do arquivo

Mas não queremos ler todo o conteúdo do arquivo, e sim ler linha por linha. Como já foi visto, um arquivo é um fluxo de linhas, uma sequência de linhas, então como é uma sequência, podemos fazer um `for` nela:

```
>>> arquivo = open("palavras.txt", "r")
>>> for linha in arquivo:
...     print(linha)
...
banana


melancia


morango


manga
```

Mas podemos reparar que existe uma linha entre cada fruta. Por que isso acontece? Para ver melhor, vamos ler somente uma linha do arquivo, com a função `readLine()`:

```
>>> arquivo = open("palavras.txt", "r")
>>> linha = arquivo.readline()
>>> linha
'banana\n'
```

Há um `\n` ao final da linha, porque a linha sabe que ao seu final deve ser ser feita uma nova linha. Mas anteriormente havíamos feito um `print`, que também quebra uma linha ao final da impressão, colocando também um `\n`! Assim, são criadas duas novas linhas, por isso havia uma linha em branco entre as frutas.

Limpando a linha

Como vimos, há um `\n` ao final de cada linha, de cada palavra, mas queremos somente a palavra. Já vimos como tirar espaços em branco no início e no fim da string, basta utilizar a função `strip()`, que também remove caracteres especiais, como o `\n`.

Sabendo disso tudo, podemos implementar a funcionalidade de leitura de arquivo no nosso jogo. Faremos isso no próximo vídeo.

Agora que já sabemos ler de um arquivo, podemos implementar a funcionalidade de escolher aleatoriamente a palavra secreta de um arquivo.

Lendo e guardando as linhas do arquivo

A primeira coisa que devemos fazer é abrir o arquivo, e como já sabemos, vamos fechá-lo:

```
def jogar():

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

    arquivo = open("palavras.txt", "r")




    arquivo.close()
```

Agora, vamos criar uma lista e fazer um laço, acessando cada linha e guardando-as nessa lista:

```
def jogar():

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

    arquivo = open("palavras.txt", "r")
    palavras = []

    for linha in arquivo:
        palavras.append(linha)

    arquivo.close()
```

Mas precisamos remover o **\n** ao final da linha, fazendo um **strip** nela:

```
def jogar():

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

    arquivo = open("palavras.txt", "r")
```

```
palavras = []

for linha in arquivo:
    linha = linha.strip()
    palavras.append(linha)

arquivo.close()
```

Agora já temos todas as palavras na lista, mas como selecionar uma delas aleatoriamente?

Gerando um número aleatório

Sabemos que cada elemento da lista possui uma posição, e vimos no treinamento anterior como gerar um número aleatório, vamos relembrar?

A biblioteca que sabe gerar um número aleatório é a **random**. Vamos testá-la no terminal do Python 3, primeiro importando-a:

```
>>> import random
```

Para gerar o número aleatório, utilizamos a biblioteca e chamamos a função **randrange**, que recebe o intervalo de valores que o número aleatório deve estar. Então vamos passar o valor **0** (equivalente à primeira posição da nossa lista) e **4** (lembrando que o número é exclusivo, ou seja, o número aleatório será entre 0 e 3, equivalente à última posição da nossa lista):

```
>>> import random
>>> random.randrange(0, 4)
0
>>> random.randrange(0, 4)
1
>>> random.randrange(0, 4)
3
>>> random.randrange(0, 4)
1
>>> random.randrange(0, 4)
3
```

Sabendo disso, vamos implementar esse código no nosso jogo.

Selecionando a palavra

Primeiramente, devemos importar a biblioteca. Vamos gerar um número de **0** até a quantidade de palavras da nossa lista, ou seja, vamos utilizar a função **len**, para saber o tamanho da lista:

```
import random


def jogar():

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

    arquivo = open("palavras.txt", "r")
    palavras = []

    for linha in arquivo:
        linha = linha.strip()
        palavras.append(linha)

    arquivo.close()

    numero = random.randrange(0, len(palavras))
```

Agora que temos o número aleatório, vamos utilizá-lo como índice para acessar a lista e atribuir essa palavra à variável **palavra_secreta**:

```
import random


def jogar():

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

    arquivo = open("palavras.txt", "r")
    palavras = []
```

```
    for linha in arquivo:
        linha = linha.strip()
        palavras.append(linha)

    arquivo.close()

    numero = random.randrange(0, len(palavras))

    palavra_secreta = palavras[numero].upper()
    letras_acertadas = ["_" for letra in palavra_secreta]
```

Podemos executar o jogo agora, e perceber que a palavra é selecionada aleatoriamente!

Mas agora o nosso arquivo, a nossa função cresceu bastante, com várias funcionalidades e responsabilidades. Então, no próximo capítulo, organizaremos melhor o nosso código, separando-o em funções e deixando-o mais fácil de entender.

Agora que já sabemos ler de um arquivo, podemos implementar a funcionalidade de escolher aleatoriamente a palavra secreta de um arquivo.

1 - De início devemos abrir o arquivo, e como já sabemos é uma boa prática fechá-lo ao final:

```
def jogar():

    arquivo = open("palavras.txt", "r")

    arquivo.close()
```

2 - Depois temos que criar uma lista e percorrer o arquivo. Cada linha do arquivo deve ser guardada nessa lista:

```
def jogar():

    arquivo = open("palavras.txt", "r")
    palavras = []

    for linha in arquivo:
        palavras.append(linha)

    arquivo.close()
```

3 - Precisamos remover o `\n` ao final da linha, fazendo um `strip` nela:

```
def jogar():

    arquivo = open("palavras.txt", "r")
    palavras = []

    for linha in arquivo:
        linha = linha.strip()
        palavras.append(linha)

    arquivo.close()
```

4 - Agora que já temos todas as palavras na lista devemos acessá-las aleatoriamente. Para isso, vamos importar a biblioteca random?

```
import random
```

5 - Com a biblioteca já disponível temos que acessar uma das palavras incluídas na nossa lista. Para isso será necessário gerar um número com a posição aleatória. O número gerado deveria ser apenas de índices válidos na lista: 0 até o tamanho da lista:

```
    numero = random.randrange(0, len(palavras))
```

6 - Com o número gerado basta agora pegarmos a palavra secreta correspondente a essa posição:

```
    numero = random.randrange(0, len(palavras))

    palavra_secreta = palavras[numero].upper()
```

O seu código final deveria estar como o seguinte:

```
import random

def jogar():
    print("*********************************")
    print("***Bem vindo ao jogo da Forca!***")
    print("*********************************")



    arquivo = open("palavras.txt", "r")
    palavras = []

    for linha in arquivo:
        linha = linha.strip()
```

```
    palavras.append(linha)

arquivo.close()

numero = random.randrange(0,len(palavras))
palavra_secreta = palavras[numero].upper()

letras_acertadas = ["_" for letra in palavra_secreta]

enforcou = False
acertou = False
erros = 0

print(letras_acertadas)

while(not enforcou and not acertou):

    chute = input("Qual letra? ")
    chute = chute.strip().upper()

    if(chute in palavra_secreta):
        index = 0
        for letra in palavra_secreta:
            if(chute == letra):
                letras_acertadas[index] = letra
            index += 1
    else:
        erros += 1

    enforcou = erros == 6
    acertou = "_" not in letras_acertadas
    print(letras_acertadas)


if(acertou):
```

```
        print("Você ganhou!!")
    else:
        print("Você perdeu!!")
    print("Fim do jogo")

if(__name__ == "__main__"):
    jogar()
```

A função jogar() possui um código muito complexo, com muitas funcionalidades e responsabilidades.

Entre as funcionalidades que o código possui, está a apresentação do jogo, leitura do arquivo e inicialização da palavra secreta, entre outras. Vamos então separar as responsabilidades do código em funções, melhorando a sua legibilidade e organização.

Função que imprime a mensagem de apresentação do jogo

Vamos começar com a mensagem de apresentação do nosso jogo, vamos exportar o código para a função imprime_mensagem_abertura(). Não podemos nos esquecer de chamar essa função no início da função jogar():

```
import random


def imprime_mensagem_abertura():
    print("**********************************")
    print("***Bem vindo ao jogo da Forca!***")
    print("**********************************")


def jogar():


    imprime_mensagem_abertura()


    # restante do código omitido
```

Não importa o local da função, ela pode ser declarada antes ou depois da função jogar().

Função de leitura do arquivo e inicialização da palavra secreta

Agora, vamos separar o código que realiza a leitura do arquivo e inicializa a palavra secreta na função carrega_palavra_secreta():

```
import random
```

```
def jogar():

    imprime_mensagem_abertura()

    carrega_palavra_secreta()

    letras_acertadas = ["_" for letra in palavra_secreta]

    # restante do código omitido

def carrega_palavra_secreta():
    arquivo = open("palavras.txt", "r")
    palavras = []

    for linha in arquivo:
        linha = linha.strip()
        palavras.append(linha)

    arquivo.close()

    numero = random.randrange(0, len(palavras))
    palavra_secreta = palavras[numero].upper()
```

Só que a função jogar() irá reclamar que a palavra_secreta não existe. O que queremos é que, ao executar a função carrega_palavra_secreta(), que ela retorne a palavra secreta para nós, assim poderemos guardá-la em uma variável:

```
import random

def jogar():

    imprime_mensagem_abertura()

    palavra_secreta = carrega_palavra_secreta()

    letras_acertadas = ["_" for letra in palavra_secreta]

    # restante do código omitido
```

Retornando um valor em uma função

Só que como faremos a função `carrega_palavra_secreta()` retornar um valor, no caso a `palavra_secreta`?

A `palavra_secreta` já existe, mas só dentro da função `carrega_palavra_secreta()`. Para que ela seja retornada, utilizamos a palavra-chave `return`:

```
def carrega_palavra_secreta():
    arquivo = open("palavras.txt", "r")
    palavras = []

    for linha in arquivo:
        linha = linha.strip()
        palavras.append(linha)

    arquivo.close()

    numero = random.randrange(0, len(palavras))
    palavra_secreta = palavras[numero].upper()

    return palavra_secreta
```

Passando valores por parâmetro para a função

Agora vamos criar uma função que inicializa a lista de letras acertadas com o caractere _. Criaremos a função `inicializa_letras_acertadas()`:

```
import random

def jogar():

    imprime_mensagem_abertura()

    palavra_secreta = carrega_palavra_secreta()

    letras_acertadas = inicializa_letras_acertadas()

    # restante do código omitido

def inicializa_letras_acertadas():
```

```
    return ["_" for letra in palavra_secreta]
```

Mas a função inicializa_letras_acertadas() precisa ter acesso à palavra_secreta, pois ela não existe dentro da função, já que uma função define um escopo, e as variáveis declaradas dentro de uma função só estão disponíveis dentro dela.
Então, ao chamar a função inicializa_letras_acertadas(), vamos passar palavra_secreta para ela por parâmetro:

```
import random

def jogar():

    imprime_mensagem_abertura()

    palavra_secreta = carrega_palavra_secreta()

    letras_acertadas = inicializa_letras_acertadas(palavra_secreta)

    # restante do código omitido

def inicializa_letras_acertadas(palavra):
    return ["_" for letra in palavra]
```

Vocês podem estar se perguntando por que encapsulamos uma simples linha de código em uma função. Fizemos isso somente para deixar claro o que estamos fazendo, melhorando a legibilidade do código, mas precisamos tomar cuidado com a criação de funções, pois criar funções desnecessariamente pode aumentar a complexidade do código.

Por fim, podemos executar o nosso código, para verificar que o mesmo continua funcionando normalmente. Lembrando que o código que verifica se o programa é o principal, deve ficar no final do arquivo:

```
import random

def jogar():

    imprime_mensagem_abertura()
    palavra_secreta = carrega_palavra_secreta()
    letras_acertadas = inicializa_letras_acertadas(palavra_secreta)

    # restante do código omitido
```

```
def imprime_mensagem_abertura():
    print("**********************************")
    print("***Bem vindo ao jogo da Forca!***")
    print("**********************************")

def carrega_palavra_secreta():
    arquivo = open("palavras.txt", "r")
    palavras = []

    for linha in arquivo:
        linha = linha.strip()
        palavras.append(linha)

    arquivo.close()

    numero = random.randrange(0, len(palavras))

    palavra_secreta = palavras[numero].upper()

    return palavra_secreta

def inicializa_letras_acertadas(palavra):
    return ["_" for letra in palavra]

if (__name__ == "__main__"):
    jogar()
```

Tudo continua funcionando normalmente, mas agora com o nosso código um pouco mais legível e organizado. No próximo vídeo extrairemos mais código para mais funções.

Função para pedir o chute do jogador

Criaremos a função `pede_chute()`, que ficará com o código que pede o chute do usuário, remove os espaços antes e depois, e o coloca em caixa alta. Não podemos nos esquecer de retornar o `chute`:

```
import random

def jogar():
```

```
    imprime_mensagem_abertura()
    palavra_secreta = carrega_palavra_secreta()

    letras_acertadas = inicializa_letras_acertadas(palavra_secreta)
    print(letras_acertadas)

    enforcou = False
    acertou = False
    erros = 0

    while (not acertou and not enforcou):

        chute = pede_chute()

    # restante do código omitido

def pede_chute():
    chute = input("Qual letra? ")
    chute = chute.strip().upper()

    return chute
```

Função para colocar o chute do usuário na posição correta da lista

Ainda temos o código que colocar o chute na posição correta, dentro da lista. Vamos colocá-lo dentro da função `marca_chute_correto()`:

```
import random

def jogar():

    imprime_mensagem_abertura()
    palavra_secreta = carrega_palavra_secreta()

    letras_acertadas = inicializa_letras_acertadas(palavra_secreta)
    print(letras_acertadas)
```

```
    enforcou = False
    acertou = False
    erros = 0

    while (not acertou and not enforcou):

        chute = pede_chute()

        if (chute in palavra_secreta):
            marca_chute_correto()
        else:
            erros += 1

        enforcou = erros == 6
        acertou = "_" not in letras_acertadas
        print(letras_acertadas)

    if (acertou):
        print("Você ganhou!")
    else:
        print("Você perdeu!")

    print("Fim do jogo")

def marca_chute_correto():
    index = 0
    for letra in palavra_secreta:
        if (chute == letra):
            letras_acertadas[index] = letra
        index += 1
```

Mas a função `marca_chute_correto()` precisa ter acesso a três valores: `palavra_secreta`, `chute` e `letras_acertadas`. Então vamos passar esses valores por parâmetro:

```
import random


def jogar():
```

```
    imprime_mensagem_abertura()
    palavra_secreta = carrega_palavra_secreta()

    letras_acertadas = inicializa_letras_acertadas(palavra_secreta)
    print(letras_acertadas)

    enforcou = False
    acertou = False
    erros = 0

    while (not acertou and not enforcou):

        chute = pede_chute()

        if (chute in palavra_secreta):
            marca_chute_correto(chute, letras_acertadas, palavra_secreta)
        else:
            erros += 1

        enforcou = erros == 6
        acertou = "_" not in letras_acertadas
        print(letras_acertadas)

    if (acertou):
        print("Você ganhou!")
    else:
        print("Você perdeu!")

    print("Fim do jogo")

def marca_chute_correto(chute, letras_acertadas, palavra_secreta):
    index = 0
    for letra in palavra_secreta:
        if (chute == letra):
```

```
            letras_acertadas[index] = letra
        index += 1
```

Função para imprimir as mensagens de vencedor e perdedor do jogo
Por fim, vamos remover a mensagem de fim de jogo e exportar os códigos que imprimem as mensagens de vencedor e perdedor do jogo:

```
import random


def jogar():

    imprime_mensagem_abertura()
    palavra_secreta = carrega_palavra_secreta()

    letras_acertadas = inicializa_letras_acertadas(palavra_secreta)
    print(letras_acertadas)

    enforcou = False
    acertou = False
    erros = 0

    while (not acertou and not enforcou):

        chute = pede_chute()

        if (chute in palavra_secreta):
            marca_chute_correto(chute, letras_acertadas, palavra_secreta)
        else:
            erros += 1

        enforcou = erros == 6
        acertou = "_" not in letras_acertadas
        print(letras_acertadas)

    if (acertou):
        imprime_mensagem_vencedor()
```

```
    else:
        imprime_mensagem_perdedor()


    print("Fim do jogo")


def imprime_mensagem_vencedor():
    print("Você ganhou!")


def imprime_mensagem_perdedor():
    print("Você perdeu!")
```

Agora o nosso código está muito mais organizado e legível.

Novas mensagens de vencedor e perdedor

Vamos começar com a mensagem de perdedor, alterando a função `imprime_mensagem_perdedor`. Ela ficará assim:

```
def imprime_mensagem_perdedor(palavra_secreta):
    print("Puxa, você foi enforcado!")
    print("A palavra era {}".format(palavra_secreta))
    print("    _______________         ")
    print("   /               \       ")
    print("  /                 \      ")
    print("//                   \/\  ")
    print("\|   XXXX     XXXX   | /   ")
    print(" |   XXXX     XXXX   |/    ")
    print(" |   XXX       XXX   |     ")
    print(" |                   |     ")
    print(" \__      XXX      __/     ")
    print("   |\     XXX     /|       ")
    print("   | |           | |       ")
    print("   | I I I I I I I |       ")
    print("   |  I I I I I I  |       ")
    print("   \_             _/       ")
    print("     \_         _/         ")
    print("       \_______/           ")
```

Agora ela recebe a `palavra_secreta` por parâmetro, então não podemos esquecer de passá-la no momento que chamarmos a função:

```
def jogar():

    # restante do código omitido

    if(acertou):
        imprime_mensagem_vencedor()
    else:
        imprime_mensagem_perdedor(palavra_secreta)
```

Do mesmo jeito, vamos refazer a mensagem de vencedor, na função imprime_mensagem_vencedor:

```
def imprime_mensagem_vencedor():
    print("Parabéns, você ganhou!")
    print("       ___________      ")
    print("     '._==_==_=_.'     ")
    print("     .-\\:      /-.    ")
    print("    | (|:.     |) |    ")
    print("     '-|:.     |-'     ")
    print("       \\::.    /      ")
    print("        '::. .'        ")
    print("          ) (          ")
    print("        _.' '._        ")
    print("       '-------'       ")
```

Desenhando a forca

Por fim, o jogo da forca não seria o jogo da forca se não mostrássemos a forca, juntamente com o seu personagem. Vamos criar a função desenha_forca, que recebe os erros por parâmetro. Para cada valor de erros, a função imprime um desenho diferente:

```
def desenha_forca(erros):
    print("  _______     ")
    print(" |/      |    ")

    if(erros == 1):
        print(" |      (_)   ")
        print(" |            ")
        print(" |            ")
        print(" |            ")
```

```
if(erros == 2):
    print(" |      (_)   ")
    print(" |      \     ")
    print(" |            ")
    print(" |            ")

if(erros == 3):
    print(" |      (_)   ")
    print(" |      \|    ")
    print(" |            ")
    print(" |            ")

if(erros == 4):
    print(" |      (_)   ")
    print(" |      \|/   ")
    print(" |            ")
    print(" |            ")

if(erros == 5):
    print(" |      (_)   ")
    print(" |      \|/   ")
    print(" |       |    ")
    print(" |            ")

if(erros == 6):
    print(" |      (_)   ")
    print(" |      \|/   ")
    print(" |       |    ")
    print(" |      /     ")

if (erros == 7):
    print(" |      (_)   ")
    print(" |      \|/   ")
    print(" |       |    ")
    print(" |      / \   ")
```

```
    print(" |            ")
    print("_|___         ")
    print()
```

Para finalizar, devemos chamar essa função quando o jogador erra, dentro do `else` e aumentar o limite de erros para 7:

```
while (not acertou and not enforcou):

    chute = pede_chute()

    if (chute in palavra_secreta):
        marca_chute_correto(chute, letras_acertadas, palavra_secreta)
    else:
        erros += 1
        desenha_forca(erros)

    enforcou = erros == 7
    acertou = "_" not in letras_acertadas
    print(letras_acertadas)

# restante do código omitido
```

Com isso, chegamos ao final da implementação do nosso jogo da forca!