Parei na página { 133 } do how to automate brong stuff with python.

FLOW CONTROL:

LOOPS:

While:

O ‘while’ executa em quanto determinada condicao for verdade então para encerrar ele precisa criar uma condição a ser atingida para que seja falso.

nome = *None  
while* nome != 'seu nome':  
 print('por favor digite seu nome')  
 nome = input()  
print('Obrigado')

só vai deixar de ser verdade quando digitar ‘seu nome’ ai ele sai do bloco ‘while’

Levando isso em conta, o código seguinte é outra maneira de escrever a mesma ação

*while True*:  
 nome = str(input('Digite seu nome: '))  
 *if* nome == 'seu nome':  
 print('Obrigado')  
 *break*

nesse caso é criado um loop infinito e o programa checa no bloco ‘if’ se o texto digitado é o mesmo que ‘seu nome’, se for ele executa o ‘break’ que é usado para sair do loop

CASO O PROGRAMA ENTRE EM UM LOOP INFINITO O COMANDO CRTL+C É USADO PARA SAIR DELE

Aqui está outro exemplo do ‘while’, só que agora usando o continue:

*while True*:  
 nome = str(input('Quem é voce? '))  
 *if* nome != 'leo':  
 *continue* senha = str(input('Beleza, qual a senha? '))  
 *if* senha == 'senha':  
 *break*print('voce é o bichao mesmo hein doido')

O continue serve para executar de novo o bloco ao qual esta identado, ele diz “execute de novo se passar por aqui(continue)”, é como se fosse o contrario do ‘break’, que diz “saia do bloco que eu estava executando”.

VALORES FALSOS: O PYTHON CONSIDERA O ‘’ (str vazia), 0 e 0.0 COMO SENDO FALSOS

For:

O loop do tipo ‘for’ serve para executar o código identado apenas uma quantidade determinada de vezes, do contrario do ‘while’ que executa em quanto uma condição for verdade, como no exemplo a seguir:

print('Olá!')  
*for* i *in* range (1, 4):  
 print('meu nome é leo "tres vezes" - ({})'.format(i))  
 i += 1

Aqui ele executa o código 3 vezes conforme definido no método ‘range’ (esta de 1 a 4 no caso pois ele conta até o numero anterior do 4) e para aumentar em 1 o valor da variável ‘i’ cada vez que for executao foi adicionado ao bloco identado o i += 1 que diz para o código adicionar 1 ao valor da variável 1 cada vez que o código rodar.

No exemplo seguinte podemos usar o loop ‘for’ para fazer a soma gaussiana usando força bruta:

total = 0  
*for* num *in* range (101):  
 total = total + num  
print('Os numeros somados de 0 a 100 equivalem a: {}'.format(total))

A soma gaussiana é caracterizada pela soma de todos os números de 0 a 100. Gauss descobriu um truque para fazer essa soma de maneira mais simples mas ja que temos um computador a nossa disposição podemos fazer isso de maneira mais ‘forçada’.

Mexendo agora um pouco com a função ‘range()’, ela pode ser chamada usando 3 parametros (argumentos), o primeiro e o segundo sendo o valor inicial, o segundo o valor final, e o terceiro o valor que ela vai contar por vez que rodar, por exemplo:

*for* i *in* range(0,11,2):  
 print(i)

Nesse exemplo o loop vai ser executado de 0 ate 11 contando de 2 em 2, no caso ele dá ‘print’ nos valores pares, o mesmo pode ser feito para os valores impares alterando um pouco o programa, como segue:

*for* i *in* range(1,10,2):  
 print(i)

Obs: valores negativos no terceiro parâmetro fazem a função fazer uma contagem regressiva.

IMPORTANDO METODOS:

Para importar um método ele deve obedecer a sintaxe como no código abaixo:

*import* random  
vezes = int(input('Quantas vezes deseja rolar o d20? : '))  
*for* i *in* range(vezes):  
 d20 = random.randint(1, 21)  
 *if* d20 == 20:  
 print('Acerto crítico! Você rolou: {} =D'.format(d20))  
 *elif* d20 == 1:  
 print(print('Erro crítico! Você rolou: {} =['.format(d20)))  
 *else*:  
 print('Você rolou: {} '.format(d20))

Nesse exemplo importamos o modulo ‘random’ da biblioteca padrão do python e usamos sua função ‘randint()’. Para usar uma função você deve escrever o método de onde ele vem ‘.’ Sua função a ser usada.

Há duas maneiras de importar uma função especifica para ser usada no código, a primeira é simplesmente usando o ‘import funcaoX’ como visto acima e o segundo é ‘from Modulo import funcaoX’:

*from* random *import* randint  
vezes = int(input('Quantas vezes deseja rolar o d20? : '))  
*for* i *in* range(vezes):  
 d20 = randint(1, 21)  
 *if* d20 == 20:  
 print('Acerto crítico! Você rolou: {} =D'.format(d20))  
 *elif* d20 == 1:  
 print(print('Erro crítico! Você rolou: {} =['.format(d20)))  
 *else*:  
 print('Você rolou: {} '.format(d20))

Dessa maneira você não precisa indicar de qual modulo está importando a função porem o código tende a ficar menos legível então sempre que possível use a primeira maneira, ou faça como desejar.

Outro modulo interessante para se importar é o ‘sys’, ele tras funções do sistema, veja:

*import* sys  
*while True*:  
 sair = str(input('digite "sair" para sair: '))  
 *if* sair == 'sair':  
 sys.exit()

Aqui o programa entra em um loop infinito e em quanto o usuário não digitar ‘sair’ o programa continua executando, o ‘exit()’ sai do programa.

FUNÇÕES:

Funções são um jeito de unir um conjunto de instruções em código em um único bloco que pode ser utilizado ao longo do código; funciona da mesma maneira que funções já conhecidas como o ‘print()’.

Para criar uma função utilizamos a palavra ‘def’ e então declaramos o nome da função; o nome da função sempre é seguido pelo ‘()’ que pode ou não conter parâmetros para o funcionamento desta função, estes parâmetros nada mais são que variáveis que operam somente dentro da função, ao chamar a função (executa-la no código) os parâmetros são chamados de argumentos; então começa a indentação com as instruções de como essa função se comportará; o que vem depois de ‘return’ é o que a função dará de output. Vide código a seguir:

*import* random  
*def* bolaOito(*number*):  
 *if number* == 1:  
 *return* 'Com certeza!'  
 *elif number* == 2:  
 *return* 'Sim será.'  
 *elif number* == 3:  
 *return* 'Sim.'  
 *elif number* == 4:  
 *return* 'Está meio nublado, tente de novo.'  
 *elif number* == 5:  
 *return* 'Me pergunte depois.'  
 *elif number* == 6:  
 *return* 'Concentre-se e pergunte depois.'  
 *elif number* == 7:  
 *return* 'Talvez.'  
 *elif number* == 8:  
 *return* 'Não.'  
r = random.randint(1, 8)  
print(r)  
print(bolaOito(r))

Esse programa funciona como uma Bola8 que te dá respostas baseadas em um número predefinido, importamos e usamos o método ‘random.randint’ para gerar o número aleatório e criamos a função ’ bolaOito()’ e aplicamos esse número, delegado a uma variável chamada ‘r’, como argumenta desta função

Aqui é importante freezar que a chamada de uma função pode ser usada em uma expressão pois seu valor equivale ao valor de retorno dela.

None:

‘None’ é um tipo de valor em Python assim como Boolean ou string e ele representa a ausência de valor; por se tratar de um tipo de valor em si ele deve ser escrito com a letra inicial maiúscula, ou seja, None.

Uma interação curiosa do None é com o print. O print() é uma função que imprime na tela o valor dentro do argumento e justamente por fazer só isso ela não tem valor retornado, por isso no código a seguir o valor booleano que None == print(‘ola’) retorna é True.

spam = print ('Olá')  
*if* spam == *None*:  
 print(*True*)  
*else*:  
 print(*False*)

Argumentos:

Quando se tratando de variáveis é importante discernir dois escopos: local e global. De maneira resumida o local é dentro da indentação de funções e o global é fora dele, ou seja, dentro de todo o resto.

O local lida diretamente com os parâmetros e variáveis criados dentro de uma função, são variáveis que só agem em quanto a função é executada, sendo assim não podem sair da indentação dela e por isso não são acessíveis fora dela, o único jeito de acessar alguma coisa que esteve dentro de uma função é através do return dela.

Todos as variáveis criadas fora do escopo local, ou seja, dentro de funções, são variáveis que agem no escopo global e podem ser acessadas no programa a qualquer momento.

Uma variável só pode ser uma das duas coisas, ou é global ou é local, nunca as duas.

Sendo assim, o escopo local pode acessar variáveis de escopo global mas o global não pode acessar as de locais e diferentes escopos locais também não tem acesso entre si; cada escopo local é um universo a parte em si. Só existe um escopo global, que é iniciado quando o programa roda.

NO ENTANTO há uma maneira de um escopo local alterar o escopo global que é através do uso do comando ‘global’, veja:

eggs = 'Hi Lorena'  
*def* spam (*word*) :  
 *global* eggs  
 eggs = *word*spam('Ja acabou, jessica?')  
print(eggs)

Perceba que o que é impresso no console é a frase ‘Já acabou, jessica?’ pois ao declarar que o programa deve tratar a variável ‘eggs’ como sendo global dentro do escopo da função ‘spam()’ ela alterou seu valor global para o valor dado como argumento da função.

EXCEÇÕES/ERROS:

Em python é possível controlar o que o programa deve fazer quando encontra um erro, isto é útil pois é interessante garantir o comportamento correto dele mesmo caso aconteçam erros.

É possível delimitar o comportamento de um erro especifico ou de uma arvore de erros da seguinte maneira:

#Esse erro ocorre ao tentar dividr por zero.  
*def* spam (*divideBy*):  
 *return* 42 / *divideBy  
try*:  
 print(spam(10))  
 print(spam(64))  
 print(spam(0))  
 print(spam(42))  
*except* ZeroDivisionError:  
 print("Argumento da funcao invalido. digite outro valor.")

O bloco ‘try’ tenta executar um código mesmo havendo possibilidade de ocorrer um erro e quando ele ocorre ele aplica a condição respectiva ao erro do bloco exception. No casso este programa imprime os valores da divisão pelos valores dados nos argumentos e quando ele tenta dividr por zero na terceira tentativa o que o programa imprime é o ‘print’ do bloco exception.

LISTAS E TUPLAS:

Listas, representadas pelo ‘[]’, e tuplas, representadas pelo ‘{}’, são objetos que guardam vários valores, são uteis pois dessa forma podemos condensar e hierarquizar a informação dentro do código de uma maneira mais organizada.

Para criar uma lista basta usar a sintaxe a seguir:

lista2 = ['baiacu','baleia']  
lista1 = ['cachorro','gato','rato',lista2]  
print(lista2[0])  
print(lista1[3])  
print(lista1[3][0])  
print(lista1[-2])

é bem simples, e veja que também é possível acessar listas dentro de listas e valores dessas listas dentro das listas como exemplificado no terceiro print, freezando mais uma vez a importância do uso de listas para organizar o código; também pode-se usar valores negativos para acessar valores dentro da lista, fazendo isso o python interpreta como lendo a lista ao contrario, por exemplo no quarto print ele imprime a palavra rato, pois é o segundo índice de tras para frente.

Slices:

Um conceito também existente quando trabalhando com listas é o de ‘slice’, funciona de maneira análoga ao de seleção de varias células em planilhas, você pega uma range de valores dentro de uma lista, observe:

# slices em listas  
lista = ['oculos','celular','computador','garrafa','sanduiche','cerveja','café']  
print('lista original: {}'.format(lista))  
print('mostrando o slice do 0 ao 3 (lista[:3]): {}'.format(lista[:3]))  
print('mostrnado o slice do 3 ao 6 (lista[3:]): {}'.format(lista[3:]))  
print('mostrando o slice do 2 ao 5 (lista[2:5]): {}'.format(lista[2:5]))

Aqui cabe a observação de que se omitir o primeiro argumento do slice ele leva em conta que vale zero e que se omitir o segundo ele leva em conta que vai ate o fim da lista; Tambem é possível pegar a quantidade de items que tem na lista com o len().

Trabalhando com listas:

Aqui está um exemplo de como trabalhar concatenando strings e listas:

#Listagem de nomes de gatos  
catList = []  
*while True*:  
 print('Digite o nome do gato no. {} ou aperte enter para encerrar o processo.'.format(len(catList)+1))  
 name = input()  
 *if* name == '':  
 *break* catList = catList + [name]  
print('Os nomes dos gatos sao:')  
*for* i *in* range (0, len(catList)): #Aqui criei a variavel 'i' pois quis mostrar o numero do gato dentro da lista no print a seguir.  
 print('O gato no. {} se chama: {}'.format(i+1, catList[i]))  
# caso nao quisesse mostrar poderia ter feito apenas o seguinte:  
*for* name *in* catList:  
 print('Um dos gatos se chama: {}'.format(name))

Quando estudamos loops aprendemos a usar o range(), o código ‘#for i in range(4):...’, por exemplo, só dá certo pois o range(4) é uma função que retorna ao python uma lista [0,1,2,3], por isso é possível fazer o ‘#for name in catList:...’, uma vez que o ‘for’ é um loop que é basicamente usado para contar objetos em uma lista.

In e Not:

Podemos usar o ‘In’ e o ‘Not’ para retornar um valor booleano que dirá se o valor checado está ou não numa lista. Veja:

#Testando In e Not:  
lista = ['cao','gato','periquito']  
spam = 'cao' *in* lista  
print(spam)  
spam = 'baleia' *not in* lista  
print(spam)  
spam = 'macaco' *in* lista  
print(spam)

Macete!

#Macete para adicionar valores contidas em listas a variaveis  
gato = ['gordo','cinza','velha']  
# inves de fazer isso:  
tamanho = gato[0]  
cor = gato[1]  
idade = gato[2]  
#Faça isso:  
tamanho, cor, idade = gato

Métodos:

Segundo o livro: “Método é a mesma coisa que uma função só que é chamada em um valor”, por exemplo:

# Esse programa demonstra um método (index()) aplicado a listas.  
lista = ['baleia','cachorro','gato']  
spam = lista.index('cachorro')  
print(spam)

A variável spam guarda o valor retornado pelo método ‘spam’ e em seguida é impresso.

# exemplo de append(), insert() e remove()  
lista = ['baleia','cachorro','gato']  
lista.append('macaco')  
print(lista)  
lista.insert(0,'periquito')  
lista.append('gato')  
print(lista)  
lista.remove('gato')  
print(lista)

O ‘append()’ insere um item ao final da lista, o ‘insert()’ em uma posição especifica e o ‘remove()’ remove um objeto da lista mas caso existam objetos repetidos é removido apenas o primeiro valor.

Veja o próximo exemplo que utiliza uma lista para armazenar os valores das respostas da bola 8 e então imprime um valor baseado no índice gerado aleatoriamente:

#Bola 8 usando listas  
*import* random  
i = random.randint(0,7)  
bola8 = ["Com certeza sim","Sim","Talvez","Nem pense nisso","Volte mais tarde","Se concentre mais e pergunte novamente","Impossível","Nao"]  
# print(i)  
print(bola8[i])

Tuplas:

Tuplas são praticamente a mesma coisa que listas exceto por duas diferença: 1. São escritas usando ‘()’ invés de ‘[]’ e os valores contidos dentro delas são imutáveis.

Aqui estão as soluções para os projetos do capitulo 4:

#Projeto do capitulo 4  
lista = ['bacon','ovos','salada','feijao']  
print(' e '.join(lista))

Dicionários:

Dicionários são um tipo de estrutura que assim como listas e tuplas servem para armazenar vario dados, porem invés de índices os valores ficam armazenados atrelados a palavras chave, conhecidas como ‘keys’.

Existem três métodos quando se tratando de dicionários que retornar valores que se assimilam a listas, não se tratam de listas, mas o python interpreta que podem ser usados para loop ‘for’, por exemplo, por serem similares, veja um exemplo de cada:

animais = {'felino': 'gato', 'canino':'cachorro'} #exemplo de dicionario  
  
*for* i *in* animais.values(): #'.value()' retorna os valores dentro de cada key  
 print(i)  
  
*for* j *in* animais.keys(): #'.keys()' retorna os valores das keys dentro do dicionario  
 print(j)  
  
*for* k *in* animais.items(): #'.items()' retorna cada item (keys e respectivos valores) do dict  
 print(k)

OBS: Como dito, esses valores não se tratam de listas mas podem ser transformados em uma usando o ‘lista(\*método do dict\*)’.

Método get():

Este método serve para verificar a existência de um valor key dentro de um dict (dicionário), ele recebe dois argumentos, o primeiro é o valor a ser procurado e o segundo é o valor a ser retornado caso a key pesquisada não seja encontrada. Caso você queira só checar se uma key exista em um dict basta utilizar algo como “print(‘felino’ in animais)” e ele retornará True ou Flase; veja aseguir um exemplo do get():

mochila = {'caderno':2,'caneta':2,'celular':1} #exemplo de dicionario  
print('Estou levando {caneta} canetas e {maca} maçãs na minha mochila.'.format(caneta = mochila.get('caneta', 0),maca = mochila.get('macas', 0)))

O programa deverá printar:

Estou levando 2 canetas e 0 maçãs na minha mochila.

setdefault():

O setdefault() é um método que checa se já existe uma key(primeiro argumento) com este valor, caso não exista ele cria uma key com este nome e com o valor dado pelo segundo argumento.

Aqui está um exemplo de um programa que conta quantas letras de cada tipo existem em uma string passada:

#Esse programa mostra a quantidade de cada letra em uma dada frase  
text = 'Eu gosto de sorvete de banana'  
count = {}  
*for* character *in* text:  
 count.setdefault(character,0)  
 count[character] += 1  
print(count)

Aqui está o mesmo exemplo só que utilizando o modulo pprint.pprint() que faz com o dicionário seja impresso de uma maneira mais legível:

#Esse programa mostra a quantidade de cada letra em uma dada frase e usa o modulo pprint  
*import* pprint  
text = 'Eu gosto de sorvete de banana'  
count = {}  
*for* character *in* text:  
 count.setdefault(character,0)  
 count[character] += 1  
pprint.pprint(count)

Segue um exemplo de um jogo da velha feito utilizando um exemplo de estruturação de dados para modelar um tabuleiro de jogo da velha usando um dicionário; nesse exemplo cada casa do tabuleiro é atribuída a uma key no dicionário criado como tabuleiro e o usuário pode modificar esse valor entrando com a casa que ele deseja colocar a pedra:

#Jogo da velha / tic-tac-toe  
theBoard = {  
 'top-L':' ','top-M':' ','top-R':' ',  
 'mid-L':' ','mid-M':' ','mid-R':' ',  
 'low-L':' ','low-M':' ','low-R':' '  
}  
*def* printBoard(*board*):  
 print('{}|{}|{}'.format(*board*['top-L'],*board*['top-M'],*board*['top-R']))  
 print('-+-+-')  
 print('{}|{}|{}'.format(*board*['mid-L'],*board*['mid-M'],*board*['mid-R']))  
 print('-+-+-')  
 print('{}|{}|{}'.format(*board*['low-L'],*board*['low-M'],*board*['low-R']))  
  
turn = 'X'  
*for* i *in* range(9):  
 printBoard(theBoard)  
 print('Vez de: ' + turn + '. Qual espaço deseja colocar sua peça? ')  
 move = input()  
 theBoard[move] = turn  
 *if* turn == 'X':  
 turn = 'O'  
 *else*:  
 turn = 'X'

Também é possível, inclusive muito útil em algumas situações, utilizar dicionários dentro de outros dicionários; veja no exemplo a seguir como saber o total de itens levados em uma festa:

#exemplo de dicionarios dentro de dicionarios  
convidados = {'Alice' : {'Bolos' : 1, 'Refrigerantes' : 2, 'Baloes' : 3},  
 'Antonia' : {'Salgadinhos' : 1, 'Sucos' : 2, 'Bolos' : 3},  
 'Elis' : {'Bolos' : 1, 'Salgadinhos' : 2, 'Sucos' : 3}}  
  
*def* totalItens(*convidados*,*item*): #Retorna o total do item discriminado  
 trazidos = 0  
 *for* i, k *in convidados*.items():  
 trazidos = trazidos + k.get(*item*, 0)  
 *return* print('- Numero de {} para a festa: {}'.format(*item*, trazidos))  
  
totalItens(convidados,'Bolos')  
totalItens(convidados,'Refrigerantes')  
totalItens(convidados,'Baloes')  
totalItens(convidados,'Sucos')

Perceba que neste exemplo, dentro do loop for, o valor de ‘i’ é associado ao nome da pessoa e o valor de ‘k’ é associado ao dicionário que contem os itens da festa (no caso o segundo valor dentro de cada item do dicionário mais externo); então se usa o ‘get()’ para pegar, caso exista, o valor associado ao item escolhido no segundo argumento da função.

Esta notação com duas variáveis no ‘for’ é muito útil quando se tratando de dicionários, veja no exemplo a seguir mais uma demonstração dessa notação, agora utilizando o projeto proposto no final do capitulo pelo livro:

#Itens de um inventario de RPG  
stuff = {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12} #Itens do inventario  
  
*def* displayInventory(*inventory*):#Funcao que retorna todos os itens do inventario e suas quantidades  
 print("Inventory:")  
 item\_total = 0  
 *for* itemName, numberOfItens *in inventory*.items():  
 print('- {} {}'.format(numberOfItens, itemName))  
 item\_total += v  
 print("Total number of items: " + str(item\_total))  
  
displayInventory(stuff)#chamada da funcao

Aqui o ‘for’ funciona como no exemplo anterior mas de uma maneira mais direta pois não contem dicionários dentro do dicionário; perceba que o primeiro valor a ser iterado(itemName) no loop ‘for’ se trata do ‘key’ dentro do dicionário, e o segundo (numberOfItens) é o ‘value’ associada àquela ‘key’.

Aqui está a continuação do exemplo agora com uma função de adicionar itens que vem a partir de uma lista:

#Itens de um inventario de RPG  
stuff = {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12} #Itens do inventario  
dragonLoot = ['gold coin', 'dagger', 'gold coin', 'gold coin', 'ruby']#Itens para ser adicionados  
*def* displayInventory(*inventory*):#Funcao que retorna todos os itens do inventario e suas quantidades  
 print("Inventory:")  
 item\_total = 0  
 *for* itemName, numberOfItens *in inventory*.items():  
 print('- {} {}'.format(numberOfItens, itemName))  
 item\_total += numberOfItens  
 print("Total number of items: " + str(item\_total))  
  
*def* addToinventory(*inventory*, *addedItems*):#Funcao que adiciona os itens de uma lista no dicionario utilizado como inventario  
 *for* item *in addedItems*:  
 *if* item *in inventory*:  
 *inventory*[item] += 1  
 *inventory*.setdefault(item ,1)  
 print('New itens added to inventory!'.format(*addedItems*))  
  
addToinventory(stuff, dragonLoot)  
displayInventory(stuff)

E aqui está a minha primeira tentavia no programa sem a ajuda do livro:

#Meu codigo  
bigBag = {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12} #inventario  
dragonLoot = ['gold coin', 'dagger', 'gold coin', 'gold coin', 'ruby'] # Itens para ser adicionados  
  
*def* showInventory(*inventory*):  
 totalOfItems = 0  
 *for* i *in inventory*.items():  
 print('{} {}'.format(i[1], i[0]))  
 totalOfItems += i[1]  
 print('Total number of items: {}'.format(totalOfItems))  
  
*def* addItems(*inventory*, *newItems*):  
 *for* item *in newItems*:  
 *if* item *in inventory*:  
 *inventory*[item] += 1  
 *inventory*.setdefault(item, 1)  
 print('New items added to inventory!' )  
  
addItems(bigBag,dragonLoot)  
showInventory(bigBag)

MANIPULANDO STRINGS:

Como já estudado, strings são um dataType em python que consiste numa sequencia determinada de caracteres para formar frases ou coisas do gênero.

\ (barra invertida) e aspas triplas:

A barra é utilizada para que o python possa exibir caracteres que de outra forma não seria exibida, veja os exemplo na tabela abaixo:

Símbolo Print:

\' Aspas simples

\" Aspas duplas

\t Tab

\n Nova linha (line break)

\\ Barra invertida

# Exemplo de simbolos e aspas triplas  
print(''''Para: Leo  
Bom dia, Leo.  
Hoje o dia será chuvoso,   
Um certo \'Alguém\' Baterá na sua porta, NÃO ATENDA!  
Assinado: Leo do futuro.  
''')

Note aqui que foi usado ‘\’’ apenas para demonstrar como funciona pois quando se usa a aspas triplas não é necessário.

Índices em strings:

É possível usar índices também, veja:

# Indices em Strings  
spam = 'Hello World!'  
  
print('spam[1]: {}'.format(spam[1]))  
print('spam[3]: {}'.format(spam[3]))  
print('spam[-6]: {}'.format(spam[-6]))  
  
print('spam[1:6]: {}'.format(spam[1:6]))

In e not in:

spam = 'Meu nome é Bleblio'  
  
print('Bleblio' *in* spam)  
print('é' *not in* spam)

Métodos Strings úteis:

Aqui temos os exemplos de ‘upper()’ e ‘lower()’ que funcionam como o nome sugere:

#upper() e lower()  
spam = 'Walk like an Egyption....'  
  
print(spam)  
spam = spam.lower()  
print(spam)  
spam = spam.upper()  
print(spam)

Também tem-se os métodos isupper() e o islower() que checa se dado texto/variável está capitializado ou não e retorna um valor booleano. Há ainda vários outros métodos que retornam um booleano e checam para várias outras coisas, veja:

#Métodos isX()  
  
spam = 'Testando...'  
print(spam.isalpha()) #returns True if the string consists only of letters and is not blank.  
  
spam = '963852741 Tenho numeros também!'  
print(spam.isalnum()) #returns True if the string consists only of letters and numbers and is not blank.  
  
spam = '998855442211'  
print(spam.isdecimal()) #returns True if the string consists only of numeric characters and is not blank.  
  
spam = ' \n'  
print(spam.isspace()) #returns True if the string consists only of spaces, tabs, and newlines and is not blank.  
  
spam = 'Título'  
print(spam.istitle()) #returns True if the string consists only of words that begin with an uppercase letter followed by only lowercase letters.

Estes métodos a princípio parecem bobos, mas são bem uteis quando necessitamos validar inputs de usuário:

# Exemplo de uso de método isX()  
*while True*:  
 idade = input('Digite sua idade: ')  
 *if* idade.isdecimal():  
 *break* print('Digite um número para sua idade!')  
*while True*:  
 nome = input('Digite seu nome: ')  
 *if* nome.isalpha():  
 *break* print('Digite um nome somente com letras!')  
print('Bem vindo {}'.format(nome))

Veja mais dois exemplos de outros dois métodos:

# starswith() e endswith()  
spam = 'Hello World!'  
print(spam.startswith('Bola'))  
print(spam.endswith('ld!'))

Split() e join():

São dois métodos que são utilizado para obter listas a partir de uma string (split()) e para obter um string a partir de uma lista (join()), veja:

# split() e join()  
spam = 'Estou cansado de sempre ouvir Tarkus, porque demora tanto pra acabar?!'  
  
spam = spam.split()  
print(spam)  
  
spam = ' '.join(spam)  
print(spam)