**FACULDADE DE COMPUTAÇÃO E INFORMÁTICA**

**CIÊNCIA DA COMPUTAÇÃO E SISTEMAS DE INFORMAÇÃO**

**Linguagem de Programação I**

**Aula 05: Sequências (Tuplas) em Python – Parte II**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | http://images.flatworldknowledge.com/ketchen/ketchen-fig05_x001.jpg | Na aula passada aprendemos o que são listas e como utilizá-las. Em Python, uma lista é um tipo de **sequência**.  Nossos objetivos nesta aula são:   * Entender o que são e como utilizar outras sequências em Python. * Aprender a percorrer por todos os elementos de uma sequência. | | Macintosh HD:Users:anacris:Desktop:Captura de Tela 2017-02-19 às 18.57.35.png | A referência para esta aula está no **Capítulo 4** do livro:  DIERBACH, C. *Introduction to Computer Science Using Python: A Computational Problem Solving Focus.* 1st Edition, New York: Wiley, 2012. | |
|  |

# O que é uma sequência?

Uma sequência em Python é um conjuntos de elementos que podem ser acessados por um índice numérico e que estão ordenados de forma linear.

Desta maneira, a **lista** que nós aprendemos na última aula é uma **sequência**. No entanto, há outras sequências: tuplas e strings.

Já fazemos uso de strings há algumas aulas. Uma string é uma sequência cujos elementos são os próprios caracteres que formam a string.

As tuplas estão sendo trazidas ao nosso conhecimento nesta aula.

# O que é uma tupla?

Uma **tupla** é uma estrutura de dados linear **imutável**. Ou seja, uma tupla não pode ser alterada após a sua definição. Por isso nós, que aprendemos as listas na última aula, podemos entender a tupla como uma lista que não pode ser alterada.

Em Python, uma tupla é representada em um conjunto de elementos dentro de **parênteses** e separados por vírgula.

(10, 20, 30)  
('Marcos', 1.68, 20)

Podemos atribuir uma tupla a uma variável, e a partir disto realizar operações na tupla por meio desta variável. O exemplo abaixo mostra a atribuição de uma tupla a uma variável, a impressão do conteúdo desta variável e a recuperação dos elementos através dos índices numéricos.

aluno = ('Marcos', 1.68, 20)  
print(aluno)  
nome = aluno[0]  
altura = aluno[1]  
idade = aluno[2]  
print('Nome:', nome)  
print('Altura:', altura)  
print('Idade:', idade)

Se formos definir uma **tupla com um único elemento**, devemos tomar o cuidado de adicionar uma vírgula logo após o elemento.

notas = (10.0,)  
print(notas)

Se não incluírmos a vírgula, o interpretador não entenderá isto como uma tupla, mas somente como o próprio elemento.

notas = (10.0)  
print(notas)

Uma tupla vazia é representada por um par de parênteses.

vazia = ()  
print(vazia)

Mais à frente no nosso curso vamos aprender possíveis usos para tuplas de um único elemento e tuplas vazias.

Das 5 operações que aprendemos a realizar em uma lista (recuperar, substituir, inserir, remover e acrescentar) a única que pode ser realizada em tuplas é a operação de **recuperar**.

gastos = (150.01, 200.23, 22.55)  
print('Segundo elemento:', gastos[1])

# Exercício 1

Quais serão os resultados apresentados se tentarmos executar as seguintes linhas em um interpretador Python?

>>> t = (10, 20, 30)  
>>> print(t[0])  
???  
  
>>> del t[1]  
???  
  
>>> t.insert(1,15)  
???  
  
>>> t.append(40)  
???

# Operações em sequências

Há algumas operações que podem ser executadas em todas as sequências, não importando se são listas, tuplas ou strings.

## *Length*

Sintaxe (para uma sequência acessada pela variável *s*):

len(s)

Descrição:

Obtém o número de elementos dentro de uma sequência.

Exemplos:

string = 'Tudo bem?'  
tupla = (40, 33, 22, 53, 98, 10)  
lista = [25, 44, 99]  
print(len(string), len(tupla), len(lista))

## *Select*

Sintaxe (para uma sequência acessada pela variável *s*, na posição *indice*):

s[indice]

Descrição:

Recupera o elemento na posição com o índice numérico informado.

Exemplos:

string = 'Tudo bem?'  
tupla = (40, 33, 22, 53, 98, 10)  
lista = [25, 44, 99]  
print(string[2], tupla[2], lista[2])

## *Slice*

Sintaxe (para uma sequência acessada pela variável *s*, posições *indice1* até *indice2*):

s[indice1:indice2]

Descrição:

Recupera uma "subsequência" formada pelos elementos que estão entre as posições de índices informados, incluindo o elemento da posição *indice1* mas sem incluir o elemento que está na posição *indice2*. Se o *indice2* não for informado, isto quer dizer que deverá retornar os elementos até o final da sequência original.

Exemplos:

string = 'Tudo bem?'  
tupla = (40, 33, 22, 53, 98, 10)  
lista = [25, 44, 99]  
print(string[1:3])  
print(string[1:])  
print(tupla[1:3])  
print(tupla[1:])  
print(lista[1:3])  
print(lista[1:])

## *Count*

Sintaxe (para uma sequência acessada pela variável *s*, deseja-se contar as ocorrências de *elemento*):

s.count(elemento)

Descrição:

Obtém o número de ocorrências de um elemento dentro da sequência.

Exemplos:

string = 'Eu estou bem'  
tupla = (40, 33, 22, 53, 98, 10)  
lista = [25, 44, 99]  
print('ocorrências da letra e na string:', string.count('e'))  
print('ocorrências do número 22 na tupla:', tupla.count(22))  
print('ocorrências do número 100 na lista:', lista.count(100))

## *Index*

Sintaxe (para um determinado *elemento* em uma sequência acessada pela variável *s*):

s.index(elemento)

Descrição:

Obtém o índice da primeira ocorrência do elemento dentro da sequência.

Exemplos:

string = 'Eu estou bem'  
tupla = (40, 33, 22, 53, 98, 10)  
lista = [25, 44, 99]  
print("índice da primeira ocorrência da letra 'e' na string:", string.index('e'))  
print('índice da primeira ocorrência do número 22 na tupla:', tupla.index(22))  
print('índice da primeira ocorrência do número 44 na lista:', lista.index(44))

Se procurarmos por um elemento que não está na lista, isto irá gerar um erro (**ValueError**) que interromperá a execução.

lista = [25, 44, 99]  
print('índice da primeira ocorrência do número 100 na lista:', lista.index(100))  
print('fim')

Note que este erro causa a interrupção da execução (a string 'fim' não chega a ser apresentada). Aprenderemos a tratar este tipo de erro no curso mais à frente.

## *Membership*

Sintaxe (para um determinado *elemento* em uma sequência acessada pela variável *s*):

elemento in s

Descrição:

Verifica se o elemento está ou não está presente na sequência.

Exemplos:

string = 'Eu estou bem'  
tupla = (40, 33, 22, 53, 98, 10)  
lista = [25, 44, 99]  
print("letra 'e' está na string:", 'e' in string)  
print('número 22 está na tupla:', 22 in tupla)  
print('número 100 está na lista:', 100 in lista)

## Concatenação

Sintaxe (para duas sequências, acessadas pelas variáveis *s1* e *s2*):

s1 + s2

Descrição:

Cria uma nova sequência que é composta pelos elementos da sequência *s1* seguidos pelos elementos da sequência *s2*.

Conhecemos o operador + como aquele que realiza a adição de 2 números. No entanto, quando ele é utilizado com sequências passa a realizar a concatenação das sequências. Por ter comportamentos diferentes de acordo com o tipo dos operandos, dizemos que o operador + é um operador **sobrecarregado**.

Exemplos:

string1 = 'Eu estou bem'  
string2 = ', e você?'  
tupla1 = (40, 33, 22, 53, 98, 10)  
tupla2 = (1,)  
lista1 = [25, 44, 99]  
lista2 = [55, 100]  
print(string1 + string2)  
print(tupla1 + tupla2)  
print(lista1 + lista2)

## Mínimo, máximo e soma

Sintaxe (para uma sequência acessada pela variável *s*):

min(s)  
max(s)  
sum(s)

Descrição:

Retornam, respectivamente, o elemento com o menor valor, o elemento com o maior valor e a soma dos elementos da sequência.

Quando aplicados a uma string, as funções *min* e *max* determinam o menor e o maior elemento de acordo com a posição do elemento dentro da **ordem alfabética**.

A função *sum* não se aplica a strings.

Exemplos:

string = 'elemento'  
tupla = (40, 33, 22, 53, 98, 10)  
lista = [25, 44, 99]  
print("menor elemento da string:", min(string))  
print("maior elemento da string:", max(string))  
print("menor elemento da tupla:", min(tupla))  
print("maior elemento da tupla:", max(tupla))  
print('soma dos elementos da tupla:', sum(tupla))  
print("menor elemento da lista:", min(lista))  
print("maior elemento da lista:", max(lista))  
print('soma dos elementos da lista:', sum(lista))

## Comparação

Sintaxe (para duas sequências, acessadas pelas variáveis *s1* e *s2*):

s1 == s2

Descrição:

Retorna True se as duas sequências forem do mesmo tamanho e se os seus elementos correspondentes forem iguais.

Exemplos:

string1 = 'Eu estou bem'  
string2 = ', e você?'  
tupla1 = (40, 33, 22, 53, 98, 10)  
tupla2 = (40, 33, 22, 53, 98, 10)  
lista1 = [25, 44, 99]  
lista2 = [44, 25, 99]  
print('strings são iguais:', string1 == string2)  
print('tuplas são iguais:', tupla1 == tupla2)  
print('listas são iguais:', lista1 == lista2)

# Exercício 2

Quais serão os resultados apresentados se tentarmos executar as seguintes linhas em um interpretador Python?

>>> s = "amendoim"  
>>> s[4:7]  
???  
  
>>> s.count('m')  
???  
  
>>> s.index('m')  
???  
  
>>> s + ' refrigerante'  
???  
  
>>> t = (10, 20, 30, 10)  
>>> t[1:3]  
???  
  
>>> t.count(10)  
???  
  
>>> t.index(10)  
???  
  
>>> t + (40, 50)  
???  
  
>>> lst = [10, 20, 30, 10]  
>>> lst[1:3]  
???  
  
>>> lst.count(10)  
???  
  
>>> lst.index(10)  
???  
  
>>> [40, 50] + lst  
???  
  
>>> lst + (40, 50)  
???

# Percurso pelos elementos de uma sequência

Percorrer os elementos de uma sequência (ou realizar a iteração sobre uma sequência) é acessar os elementos um a um, começando pelo primeiro e terminando no último elemento da sequência.

O percurso pelos elementos de uma sequência pode ser feita através de um laço de repetição.

Exemplo de percurso utilizando um laço **while**:

lista = [10, 20, 30, 5]  
indice = 0  
tamanho = len(lista)  
while indice < tamanho:  
 print(lista[indice])  
 indice = indice + 1  
print('fim')

Python oferece uma maneira conveniente de percorrer os elementos de uma sequência utilizando o laço **for**, com a seguinte sintaxe:

for k in sequencia:  
 instruções

A execução deste bloco irá fazer com que a variável **k** (que é a **variável do laço**) assuma, a cada execução das **instruções**, o valor de um elemento da sequência, começando pelo primeiro elemento e terminando pelo último.

Desta forma, o laço que demos como primeiro exemplo nesta seção pode ser implementado da seguinte maneira utilizando o "**for iterativo**":

lista = [10, 20, 30, 5]  
for x in lista:  
 print(x)  
print('fim')

## A função *range*

A função *range* (que já foi mencionada em aulas anteriores) gera uma sequência de inteiros que pode ser percorrida utilizando o **for iterativo**.

for i in range(5):  
 print(i)

for i in range(2,15):  
 print(i, end=' ')

Desta maneira, também é possível percorrer os elementos de uma sequência utilizando a função range para gerar os índices e, dentro do laço, recuperar o elemento da sequência pelo índice.

lista = [10, 20, 30, 5]  
for indice in range(len(lista)):  
 print(lista[indice])  
print('fim')

## Qual forma utilizar para percorrer os elementos de uma sequência

Pelo que foi exposto, conhecemos então 3 maneiras de percorrer os elementos de uma sequência:

1. Usando o **for iterativo**
2. Usando o **for iterativo** combinado com a função *range* e acessando o elemento pelo índice
3. Usando um laço while.

Quando precisamos, a cada iteração, apenas consultar o valor armazenados na sequência, sem alterá-los, o mais simples é utilizar o **for iterativo**.

lista = [25, 10, 20, 30, 5]  
for x in lista:  
 print(x)  
print('fim')

Quando precisamos, a cada iteração, **alterar** o valor dos elementos, será necessário acessar os elementos da sequência utilizando o índice. Suponha, por exemplo, que desejamos incrementar de 1 unidade cada elemento armazenado em uma lista:

lista = [25, 10, 20, 30, 5]  
for indice in range(len(lista)):  
 lista[indice] += 1  
print(lista)  
print('fim')

Quando precisamos percorrer uma sequência e este percurso puder ser interrompido se uma determinada condição for satisfeita, o uso do laço com **while** pode ser a opção mais adequada. Suponha que desejamos achar o índice do primeiro número par que aparece em uma tupla:

# teste este trecho de código para outros elementos nesta tupla  
tupla = [25, 15, 20, 30, 5, 100]  
  
indice = 0  
tamanho = len(tupla)  
par = False  
indicePrimeiroPar = -1  
while indice < tamanho and not par:  
 if tupla[indice] % 2 == 0:  
 par = True  
 indicePrimeiroPar = indice  
 indice = indice + 1

if par:  
 print('O primeiro par aparece na posição de índice', indicePrimeiroPar)  
else:  
 print('Não encontrei um número par na sequência')  
print('fim')

# Exercício 3

Qual será o resultado da execução do seguinte trecho de código Python?

k = 0  
sum = 0  
nums = range(100)  
while k < len(nums) and sum < 100:  
 sum = sum + nums[k]  
 k = k + 1   
print('Os primeiros', k, 'inteiros somam 100 ou mais')

# Exercícios extras

1. Escreva uma função **não recursiva** que receba como parâmetros uma letra e uma palavra, e retorne quantas vezes a letra aparece dentro da palavra.

2) Escreva uma função **recursiva** que receba como parâmetros uma letra e uma palavra, e retorne quantas vezes a letra aparece dentro da palavra.

3) Desenvolva um programa para cadastrar e mostrar os nomes de todos os seus amigos. Ao ser executado, seu programa deverá apresentar um menu com as seguintes opções:

Escolha uma opção:  
(1) Cadastrar um amigo (no final da lista)  
(2) Mostrar os nomes de todos os amigos  
(9) Sair do programa  
  
Opção selecionada:

Se o usuário escolher a opção 1, o programa deverá solicitar o novo nome e acrescentá-lo no final da lista:

Opção selecionada: 1  
Nome do amigo a ser cadastrado: João

Se o usuário escolher a opção 2, o programa deverá apresentar a lista com todos os nomes:

Opção selecionada: 2  
Amigos cadastrados:  
['Marcos', 'Alexandre', 'João']

O programa deverá executar a operação escolhida e retornar ao menu (a não ser que a opção 9 seja escolhida).

4) Altere a aplicação do exercício-extra 3 de forma a ter o menu com as seguintes opções:

Escolha uma opção:  
(1) Cadastrar um amigo (no final da lista)  
(2) Mostrar os nomes de todos os amigos  
(3) Cadastrar um amigo (no início da lista)  
(4) Remover um nome  
(5) Substituir um nome  
(6) Mostrar o número total de amigos cadastrados  
(7) Colocar os nomes dos amigos em ordem alfabética  
(9) Sair do programa  
  
Opção selecionada:

A opção 3 também deve adicionar um nome à lista, só que fazendo a inserção na primeira posição ao invés de acrescentar no final da lista.

Na opção 4, o programa deverá solicitar o índice do item a ser removido e executar a operação re remoção na lista.

Na opção 5, o programa deverá solicitar o índice do item a ser substituído e o novo nome; a seguir, deverá executar a substituição do nome na lista.