www.estudar.com.vc

Mac em Python

Fuja do Nabo P2 2018.1

Resumo e Lista de Exercícios

Resumo Teórico

As seguintes informações foram tiradas da apostila elaborada pelo professor também usada em aula.

Listas

Listas são uma forma de guardar uma sequência de elementos numa única variável. Como a sequência de naturais de 1 a 10:

In [16]:
```
naturais = [1,2,3,4,5,6,7,8,9,10]
print(naturais)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Podemos criar essa mesma sequência é através do comando list() que recebe um iterável como parâmetro (por exemplo, o comando range()):

In [17]:
```
naturais_2 = list(range(1,11))
print(naturais_2)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Uma outra forma de criar essa lista é através de um loop, adicionando os elementos um a um a partir de uma lista vazia:

In [18]:
```
naturais_3 = []
for elemento in range(1,11):
    naturais_3.append(elemento)

print(naturais_3)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Podemos acessar um elemento único de uma lista através da forma "lista[índice]" sendo que o primeiro índice é o 0. Ou mesmo uma parte da lista na forma "lista[início:fim]" sendo que a lista completa seria "lista[:]"

In [21]:
```
letras = list("palavras")
# Sim, repare que 'palavras' é um iterável por ser dividido em letras.
primeira_letra = letras[0]
quarta_letra = letras[3]
última_letra = letras[-1] # Ao colocarmos o '-' indicamos que estamos indo de trás pra frente.
primeira_metade = letras[:4]
teste = letras[2:7:2] # Aqui estamos pegando a fatia do índice 2 ao 7, indo de 2 em 2 (ou seja, o passo é 2)

print(primeira_metade)
print(teste)
```

```
['p', 'a', 'l', 'a']
['l', 'v', 'a']
```

Em python e outras linguagens de programação, quando passamos para uma variável uma lista, nós estamos na verdade apenas apontando tanto a variável quanto a lista para um mesmo espaço de memória, por isso, se copiamos uma lista apenas com "lista1=lista2", não estaremos fazendo uma cópia de fato.

In [23]:
```
letras = naturais # Aqui a variável letras, que antes era uma lista, recebe o valor da variável naturais
naturais.append(11) # Vamos testar a hipótese acima adicionando um elemento na lista 'naturais'
print(letras)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Aqui seguem exemplos de fazer uma cópia corretamente de uma lista

In [36]:
```
letras = naturais[:]
## OU
letras = naturais.copy() # Esse .copy() é um método que podemos aplicar a listas, ele faz uma cópia de lista
## OU
letras =[]
for elementos in naturais:
   letras.append(elementos)

# Agora sim fizemos uma cópia, para provar:
teste = naturais
naturais.remove(11) # Esse comando remove a primeira incidência do que é dado como argumento, que aqui é o 11
print(letras)
print(teste)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Matrizes

Podemos pensar nas matrizes como uma aplicação de Listas, pois se numa lista tivermos como seus elementos outras listas de tamanhos iguais é possível abstrair que o número total de elementos seria o número de linhas e o tamanho de cada elemento (que é uma lista) seria o número de colunas

In [4]:
```
linha1 = list(range(10))
linha2 = list("hipopotamo")
linha3 = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

# Se quisermos, podemos entender essas listas acima como linhas de uma matriz.
# Repare que todas têm o mesmo tamanho

Matriz = []
Matriz.append(linha1)
Matriz.append(linha2)
Matriz.append(linha3)

print(Matriz)
```

```
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], ['h', 'i', 'p', 'o', 'p', 'o', 't', 'a', 'm', 'o'], [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]]
```

In [2]:
```
# Para provar a questão dos tamanhos que comentei
# vamos usar o comando len() que devolve o tamanho de um iterável

print(len(linha1)==len(linha2)==len(linha3)) # Se os tamanhos forem iguais, será printado o 'True'

num_linhas = len(Matriz) # Aqui vemos quantos elementos existem na Matriz, ou seja, quantas listas existem
num_colunas = len(Matriz[0]) # Aqui poderia pegar qualquer um dos 3 indices (0, 1 ou 2)
                             # já que seus tamanhos são iguais

elemento = Matriz[1][2]
```

```
True
```

Vamos agora fazer loops pela Matriz para printá-la de uma forma mais elegante

In [20]:
```
def printa_Matriz(Matrix):
    print("+ "+"-"*(2*len(Matriz[0])-1)+" +") # Aqui estamos printando entre dois sinais de "+" alguns traços
                                              # que variam em função do número de colunas da Matriz

    for linha in Matriz:                      # Agora, percorremos as linhas da matriz
        print("|", end=" ")                   # Em cada linha printamos uma barra lateral no início e fim
        for elemento in linha:                # Aí percorremos os elementos da linha com espaço entre eles
            print(elemento, end=" ")
        print("|")
    print("+ "+"-"*(2*len(Matriz[0])-1)+" +")

printa_Matriz(Matriz)
```

```
+ ------------------ +
| 0 1 2 3 4 5 6 7 8 9 |
| h i p o p o t a m o |
| 9 8 7 6 5 4 3 2 1 0 |
+ ------------------ +
```

Strings

String (type str()) são uma sequência de caracteres e são representadas entre aspas "" ou entre apóstrofes ''

In [2]:
```
frase = "Isso é uma string"

outra_frase = "Podemos marcar uma palavra dentro de uma string, 'assim', viram?"
```

As strings em python tem muitas funções semelhantes as listas. Podemos usar len(), podemos iterar com loops, podemos acessar um caracter da string através do string[índice] e até mesmo concatenar strings

In [36]:
```
print(len(frase))
print(list(frase))
print(frase[11:]==outra_frase[41:47]) # Aqui estamos pegando os trechos em que ocorre a palavra "string"

for letra in outra_frase:
    if letra < "a":                   # Estamos pegando apenas os caracteres que são menores do que a,
        print(letra, end=" ")         # ou seja, letras maiúsculas e símbolos.

nova_frase = frase[:7]+"também"+outra_frase[14:18]+outra_frase[40:47]

print()                               # Como usamos o parâmetro end= no úlimo print() que impede a quebra de
print(nova_frase)                     # linha, esse print() serve para realizar a quebra de linha apenas
```

```
17
['I', 's', 's', 'o', ' ', 'é', ' ', 'u', 'm', 'a', ' ', 's', 't', 'r', 'i', 'n', 'g']
True
P               ,    ' ' ,    ?
Isso é também uma string
```

Mas como nem tudo são flores, strings também não são listas, então nao permitem, por exemplo, assinalar valores

In [37]:
```
frase[11:] = "não string"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-37-2a0ddc66b28a> in <module>()
----> 1 frase[11:] = "não string"

TypeError: 'str' object does not support item assignment
```

In [3]:
```
#Então precisamos fazer criar uma nova string para modificar a anterior

frase2 = frase[:11]+"coisa estranha"

print(frase2)
```

```
Isso é uma coisa estranha
```

E já que falamos em caracteres, existe um índice relacionado com a tabela ASC II para cada caractere sendo do 32-64 os índices para os caracteres especiais, começando pelo espaço e, a partir do 65 o alfabeto com todas as maiúsculas primeiro. O comando chr() recebe o índice e devolve o caractere, enquanto que o comando ord() recebe um caractere e devolve seu índice.

In [63]:
```
for indice in range(65,91):
    print(chr(indice), end=" ")

print("\n","Z" < "a" and "f" > "a")

letra = chr(ord("a")+10)
print(letra)
```

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 True
k
```

Exercícios

1. Listas

Adaptado da P2 2017

Simule o código abaixo. Qual a impressão do programa?

```
def main():
    L0 = [ ]
    v = 32
    while (v > 0):
        v = (v//2) - 2
        L0.append([v, 2*v])
        L1=L0
        L1[0][1] = (L1[0][0] + L1[len(L1)-1][0])//2
    print(L0)
main()
```

2. Listas

P2 2017

Considere os 4 trechos de código de um mesmo programa (*T1* até *T4*) e depois selecione as afirmações verdadeiras sobre cada um deles. Consideração: as opções sobre cada trecho podem conter de nenhuma afirmação correta até todas.

```
#Trecho T1:

def funcA(L):
    v = 0
    i = 0
    while i < len(L):
        if v < L[i]:
            v = L[i]
        i+=1
    return v
```

```
#Trecho T2:

def funcB(L):
    v = L[0]
    i = 1
    while i < len(L):
        if v > L[i]:
            v = L[i]
        i+=1
    return v
```

```
#Trecho T3:

def funcC(v,L):
    i = 0
    while i < len(L) and v != L[i]:
        i+=1
    if i < len(L):
        R = i
    else:
        R = -1
    return R
```

```
#Trecho T4:

def main():
    N = [0,1,3,2,4,7,5]
    A = funcA(N)
    B = funcB(N)
    iA = funcC(A, N)
    iB = funcC(B,N)
    print(A, B, iA, iB)
```

T1

A. Retorna o maior valor em L se ela contém apenas naturais.

B. Não retorna o maior valor em L se ela contém apenas negativos.

C. Dá erro de execução se L for vazia.

D. Dá erro de execução se L tiver repetição de valores.

T2

A. Retorna o menor valor em *L* se ela contém apenas naturais.
B. Dá erro de execução se *L* for vazia.
C. Não retorna o menor valor em *L* se ela contém apenas negativos.
D. Dá erro de execução se *L* contém valores *float*.

T3

A. Para todo *v* e toda lista *L*, retorna a posição do maior valor em *L.*
B. Retorna -1 apenas quando *L* é vazia.
C. Retorna a posição da primeira ocorrência de *v* em *L.*
D. Dá erro de execução se *L* for vazia.

T4

A. Mostra a posição do maior natural em *N.*
B. Mostra o maior natural em *N.*
C. Determina se *N* é crescente.
D. Não mostra a posição do menor natural em *N.*

3. Matrizes

Exemplo usado em aula na disciplina MAC2166

Dizemos que uma matriz quadrada inteira é um quadrado mágico se a soma dos elementos de cada linha, a soma dos elementos de cada coluna e a soma dos elementos das diagonais principal e secundária são todas iguais. Exemplo de matriz quadrado-mágico:

8	0	7
4	5	6
3	10	2

Faça um programa que, dados um inteiro positivo n $(n \geq 1)$ e uma matriz quadrada $A_{n\,x\,n}$, verificar se A é quadrado mágico.

4. *Strings*

Exemplo usado em aula na disciplina MAC2166

Faça um programa que, dada uma frase (uma *string* contendo variados tipos de caracteres), imprima o comprimento da palavra mais longa nela.

Gabarito

1. $[[14, 7], [5, 10], [0, 0]]$

***2. T1*:** Alternativas A. e B.
***T2*:** Alternativas A. e B.
***T3*:** Alternativa C.
***T4*:** Alternativas A. e B.

3.

In [1]:
```python
def quadrado_magico(matriz):
    quadrado_magico = True

    n = len(matriz)  # Como a matriz é suposta quadrada, não é necessário pegar o numero de colunas

    # A soma de referência será a soma dos números da linha 1 da matriz
    soma_referencia = 0
    for numero in matriz[0]:
        soma_referencia += numero

    # Verifica se a soma de cada uma das demais linhas é igual à soma de referência
    i = 1
    while quadrado_magico and i < n:    # Percorre as linhas da matriz a partir da segunda linha
        soma_linha = 0
        for numero in matriz[i]:   # Para cada número na linha matriz[i], faça
            soma_linha += numero   # acrescente o número na soma da linha matriz[i]

        if soma_linha != soma_referencia:
            quadrado_magico = False

        i += 1

    # Verifica se a soma de cada uma das colunas dá igual à soma de referência
    j = 0
    while j < n and quadrado_magico:   # Percorre as colunas da matriz
        soma_coluna = 0
        for i in range(n):             # Percorre as linhas de uma dada coluna
            soma_coluna += matriz[i][j]

        if soma_coluna != soma_referencia:
            quadrado_magico = False

        j += 1

    # obtém  a soma de cada uma das diagonais
    soma_diagonal_principal = 0
    soma_diagonal_secundaria = 0

    k = 0
    while quadrado_magico and k < n:
        soma_diagonal_principal += matriz[k][k]
        soma_diagonal_secundaria += matriz[k][n-k-1]
        k += 1

    if soma_diagonal_principal != soma_referencia or soma_diagonal_secundaria != soma_referencia:
            quadrado_magico = False

    if quadrado_magico:
        print("A matriz é um quadrado magico! :) ")
    else:
        print("A matriz não é um quadrado magico! :( ")
```

4.

In [10]:
```
def main():
    frase = input("Digite uma frase: ")

    n = len(frase)
    tam = 0
    max = 0
    for ch in frase:
        if 'a' <= ch <= 'z' or 'A' <= ch <= 'Z':
            tam = tam + 1
        else:
            tam = 0

        if tam > max:
            max = tam

    print("\nMaior comprimento de uma palavra = %d\n" %(max)

# início da execução do programa
main()
```

```
Digite uma frase: Batatinha quando nasce, espalha a rama pelo chão

Maior comprimento de uma palavra = 9
```