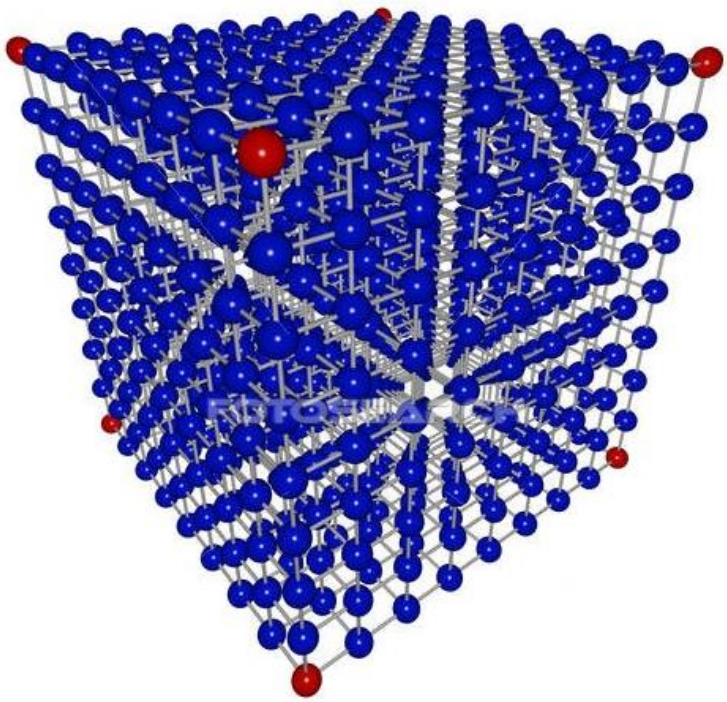
Capítulo

2 Array e Matriz



Array

O **array** é uma coleção de objetos armazenados de forma contígua e cuja ordem de inserção e remoção não é definida previamente. O tipo do objeto pode ser um número inteiro, um caractere, um string, ou um elemento qualquer de uma classe específica. Como exemplos podem-se destacar: o vetor com os números inteiros, o vetor com as letras do alfabeto e o vetor com nomes de pessoas de uma seleção de candidatos de um concurso.

Para exemplificar a importância de se utilizar o **array**, considere o seguinte Fragmento de Código 2.1. Este código imprime o mês por extenso de acordo com a referência numérica do mês (variável **m**). Por exemplo, representa-se **m**=1 para Janeiro, **m**=2 para Fevereiro, e assim sucessivamente.

```
if    (m ==  1):
      print("Janeiro")
elif  (m ==  2):
      print ("Fevereiro")
elif  (m ==  3):
      print ("Março")
elif  (m ==  4):
     print ("Abril")
elif  (m ==  5):
     print ("Maio")
elif  (m ==  6):
     print ("Junho")
elif  (m ==  7):
     print ("Julho")
elif  (m ==  8):
     print ("Agosto")
elif  (m ==  9):
    print ("Setenbro")
elif  (m == 10) :
    print ("Outubro")
elif  (m == 11):
    print ("Novembro")
else (m == 12):
    print ("Dezembro")
```

Fragmento de Código 2.1 Implementação em Python de uma rotina que imprime o mês por extenso.

O Python implementou a classe array com todo o formalismo que a classe necessita. O link para ter acesso à documentação dessa classe é

array— Vetores eficientes de valores numéricos — documentação Python 3.9.1

sintaxe: <variável> = array(‘<tipo>, <valores>)

Exemplos de criação de vetores

```
from array import *     # carrega a biblioteca array e importa tudo
a=array('i',[10,20,30]) # cria um array de inteiros inicializado com os valores 10, 20 e 30
b=array('u','string')   # cria um vetor de characteres inicializado com "string"
```

A tabela a seguir ilustra todos os tipos de código utilizado na classe array.

Type code	Tipo em C	Tipo em Python	Tamanho mínimo em bytes	Notas
'b'	signed char	int	1	
'B'	unsigned char	int	1	
'u'	Py_UNICODE	Caractere unicode	2	(1)
'h'	signed short	int	2	
'H'	unsigned short	int	2	
'i'	signed int	int	2	
'I'	unsigned int	int	2	
'l'	signed long	int	4	
'L'	unsigned long	int	4	
'q'	signed long long	int	8	
'Q'	unsigned long long	int	8	
'f'	float	float	4	
'd'	double	float	8	

Nesse capítulo, apresentaremos a implementação da estrutura **array** utilizando a estrutura list() do Python que será estudada com mais detalhes no capítulo Estruturas de Dados Básicas.

Uma forma mais elegante, está em definirmos um vetor de meses do tipo String, onde os índices do vetor representam o mês na forma inteira. Por exemplo no vetor months, para representarmos o mês de janeiro escreveríamos months[1], months[2] para representarmos o mês de fevereiro e assim sucessivamente.

months = [“”, "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]

Como o tipo list() do Python começa sempre do índice 0 (zero), pode-se observar anteriormente que, months[0] foi inicializado com “ ”.

Para imprimir um determinado mês de índice i, podemos escrever

```
print(months[i])
```

De um modo geral podemos definir o **array** da seguinte forma:

```
nome_do_array = []
```

A seguir temos alguns exemplos de definição de **arrays**

```
digitos10 = [0,1,2,3,4,5,6,7,8,9] # vetor dos dígitos na base decimal
dia = [‘segunda’,’terça’,’quarta’,quinta’,’sexta’,’sábado’, ‘domingo’]
#  vetor dos dias da semana
vogais = [‘a’,’e’,’i’,’o’,’u’]   # vetor de  vogais
```

Antes da utilização da variável, devemos inicializar os vetores. Algumas formas de inicialização podem ser exemplificadas

```
números = []  # inicialização do vetor sem elementos
alfabeto = [''] * 26 #  inicialização do vetor com 26 caracteres vazios
```

No caso o vetor é um conjunto enumerável de elementos, logo, pode-se inicializá-lo enumerando todos os seus elementos, como por exemplo:

```
vogais = [  'a','e','i','o','u']
digitos2 = [0,1]
```

Representação **na** memória

Quando criamos um novo **array**, o Python reserva espaço na memória (e inicializa os valores) para este vetor. Este processo é chamado de alocação de memória. Na realidade, quando se cria uma variável do tipo vetor, esta variável irá guardar um endereço de memória, variável de referência do vetor, e neste endereço serão armazenados os seus elementos.

Quando, por exemplo, estivermos manipulando um determinado elemento do vetor, como por exemplo months[5], estamos fazendo referência ao valor do elemento da célula de posição 5. Um vetor de N elementos, começa na posição 0, e termina na posição N-1. A figura 2.1 ilustra graficamente a variável months armazenada na memória.

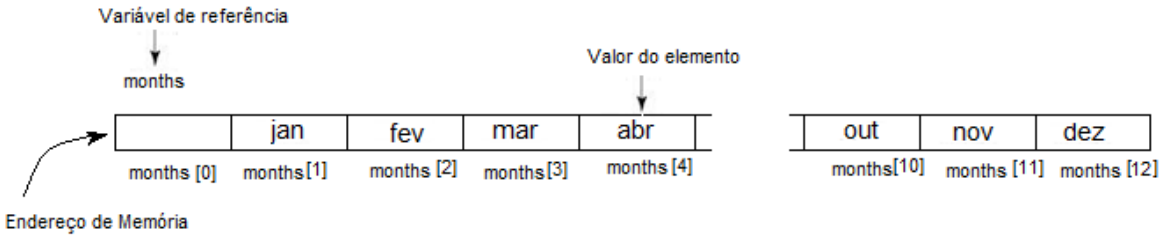
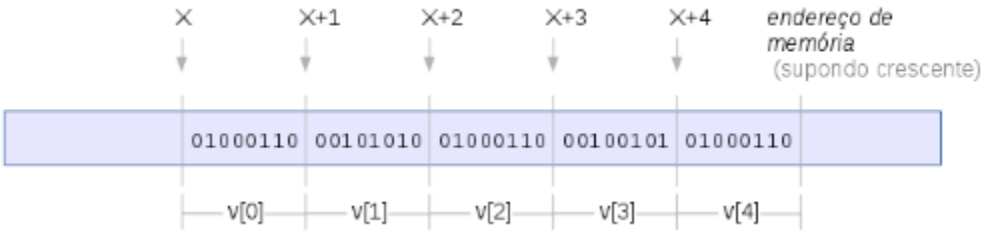


Figura 2.1. Representação esquemática do vetor months em memória.

Ao declarar uma *lista*, significa que associamos um nome de variável à uma sequência de posições de memória, sendo que cada item pode ter um tamanho distinto dos demais, como por exemplo [1,[2],"tres"];



Acessando e atribuindo valores aos elementos de um array

O acesso aos elementos do **array** é feito colocando o índice da célula entre colchetes, da seguinte forma:

```
números[0]    #  obtém o primeiro elemento do vetor números
v = vogais[3] #  obtém o quarto elemento do vetor vogais
```

A atribuição é feita da seguinte forma:

```
numeros[0]  = 10  # atribui-se o inteiro 10 à posição 0 do vetor números
alfabeto[1] = 'a' # atribui-se o caractere 'a'  à posição 1 do vetor alfabeto
nome[2] = "Luiz Thadeu" # atribui-se o String  "Luiz Thadeu"  à posição 2
                        # do vetor  nome
```

Obtendo o índice de um determinado elemento em um vetor

```
alfabeto = ['a','e','i','o','u']
print(alfabeto.index('e'))
1
```

Cuidado, se o elemento não existir a rotina é terminada com erro, da seguinte forma

```
    print(alfabeto.index('f'))
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    print(alfabeto.index('f'))
ValueError: 'f' is not in list
```

Por isso, antes de cada execução da verificação do índice de um determinado elemento no vetor, deve-se verificar se está no vetor, da seguinte forma

```
'e' in alfabeto
```

retornará True

Observações

- É importante frisar que o vetor declarado com N posições é endereçado de 0 até N-1, isto é, o vetor números[10] pode ser acessado pelos endereços 0, 1, 2, ..., 9.

- O tamanho do vetor é determinado pelo método len da classe list(). A seguir pode-se destacar alguns exemplos de utilização do método len aplicado à alguns vetores.

```
len(números)   # retorna o tamanho do vetor números
len(letras)    # retorna o tamanho do vetor letras
```

Verificação de limites. Ao programar com vetores, deve-se ser sempre cuidadoso. É de responsabilidade do programador usar índices permitidos ao acessar um determinado elemento da matriz.

Exemplo: Podemos usar o seguinte código em um programa que processa um jogo de cartas.

```
suit_names = ["Clubs", "Diamonds", "Hearts", "Spades"]
rank_names = [None, "Ace", "2", "3", "4", "5", "6", "7",
          "8", "9", "10", "Jack", "Queen", "King"]
```

Após a criação destes dois vetores, podemos imprimir uma determinada carta de um baralho virtual da seguinte forma:

```
print((random.choice(suit_names),random.choice(rank_names)))
```

Após a execução do comando anterior teríamos a impressão de um possível par (tuple), (naipe, valor) como a seguir:

```
('Hearts', '10')
```

Percorrendo um array

Existem casos em que se necessita percorrer todas as células ou grande partes das células de um **array**, como por exemplo: imprimir todos os elementos, ou pesquisar se um determinado elemento está ou não no **array**.

A primeira forma de se percorrer o vetor é utilizar o processo de iteração da classe list(). Este processo está descrito a seguir

```
for s in suit_names:
    print(s)
```

A segunda forma de se percorrer o vetor é obter o valor de cada célula do vetor de forma exaustiva da seguinte maneira

```
for i in range(len(suit_names)):
    print(suit names[i])
```

A estrutura list() do Python, aceita qualquer tipo de objeto como sendo o elemento, ou seja, podemos inserir diversos tipos de elementos numa mesma lista.

As atribuições são feitas de acordo com o tipo que se deseja, por exemplo

```
a[0]=10       # Armazenando valores inteiros
a[0]= 'Ana'   # Armazenando valores do tipo string
```

Deve-se tomar cuidado, considerando o vetor do tipo object, pois as atribuições podem ser feitas de tipos diferentes de elementos. Neste caso, deve-se fazer uma gerência do tipo que se está trabalhando para que o vetor fique sempre homogêneo. No caso do vetor não homogêneo o programador deverá descobrir o tipo de cada célula antes de manipulá-lo.

Matriz

O conceito inicial de matriz remete-se à matemática onde pode-se definir uma matriz como sendo uma coleção de objetos dispostos em linhas e colunas. Por exemplo a matriz A de ordem m x n (m linhas e n colunas) pode ser representada da seguinte forma.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}$$

Como em Python não se tem uma representação do tipo matriz, o que se faz é uma coleção de estruturas do tipo list() para fazer esta representação.

No caso anterior, supondo os elementos da matriz do tipo None, e m e n, variáveis do tipo inteiro, inicializadas previamente, pode-se representar a matriz A, da seguinte forma

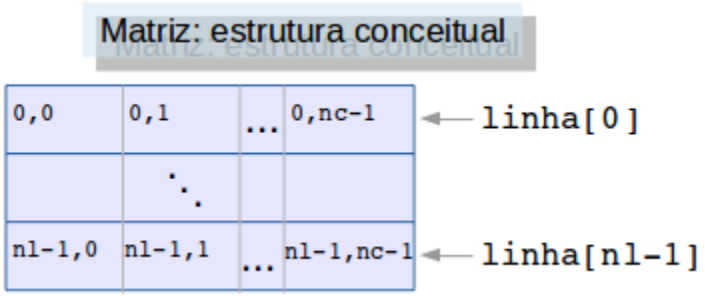
```
A = [[None] * n ] * m
```

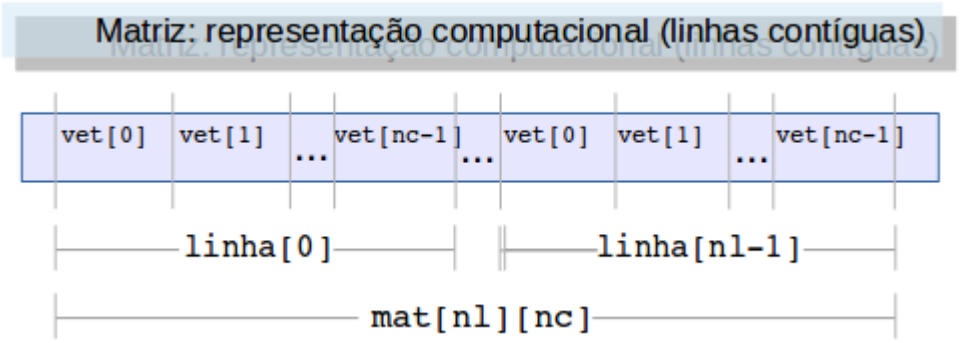
Uma outra forma de inicialização da matriz, está em se enumerar os elementos desta matriz, como por exemplo:

```
# inicialização de uma matriz 3 x 2 de números inteiros
A = [ [1,2],[3,4], [5,6]]
# inicialização de uma matriz 5 x 1 do tipo char
A = [ ['a], ['e'], ['i'], ['o'], ['u']]
```

Uma vez que um vetor/lista é armazenado consecutivamente na memória, então podemos armazenar vários vetores também consecutivamente para obter algo que represente (e seja tratado como) *matriz*. Como na imagem a seguir, em que cada alocamos *nl* vetores consecutivos na memória, cada vetor com *nc* posições de memória, desde vet[0] até vet[nc-1].

Assim, podemos ver este agregado de dados como uma matriz mat[][], cuja primeira linha é o primeiro vetor e assim por diante. Isso está ilustrado na imagem a seguir.





Acessando e atribuindo valores aos elementos de uma matriz

Considere as matrizes A e B a seguir

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad \text{e} \quad B = \begin{bmatrix} a \\ e \\ i \\ o \\ u \end{bmatrix}$$

Em Python a criação destas matrizes pode ser da seguinte forma

```
# inicialização de uma matriz 3 x 2 de números inteiros
A = [ [1,2],[3,4], [5,6]]
# inicialização de uma matriz 5 x 1 do tipo char
A = [ ['a], ['e'], ['i'], ['o'], ['u']]
```

O acesso aos elementos da matriz é feito da seguinte forma:

```
# Obter o elemento da primeira linha e
# segunda  coluna da matriz A
a = A[0][1]
# Obter o elemento da segunda linha  e
# primeira coluna da matriz B
b = B[1][0]
```

A atribuição é feita da seguinte forma:

```
# atribui-se o inteiro 10 ao elemento da primeira linha
# e segunda  coluna da matriz A
A[0][1] = 10
# atribui-se o caractere 'm'  ao elemento da segunda linha
# e primeira coluna da matriz B
B[1][0] = 'm'
```

Quando criamos uma matriz por exemplo (4x2) do tipo inteiro, podemos escrever inicializados com zero

```
>>> # matriz de 4 linhas e 2 colunas com elementos nulos
>>> a = [ [0] * 2] * 4
>>> a
[[0, 0], [0, 0], [0, 0], [0, 0]]
```

A alocação das células desta matriz pode ser entendida da seguinte forma

zeroes[0][0]	zeroes[0][1]	zeroes[0][2]	zeroes[0][3]
zeroes[1][0]	zeroes[1][1]	zeroes[1][2]	zeroes[1][3]

O número de linhas e de colunas de uma matriz é determinado pelo método len da classe list(), utilizando-o da seguinte forma:

```
len(A)     # retorna o número de linhas da matriz A
len(A[i]) # retorna o número de colunas da linha i,
          # isto é, retorna o número de colunas
          # (considerando que todas as linhas tenham
          # o mesmo número de colunas
```

Percorrendo uma matriz

Existem casos em que se necessita percorrer todas as células de uma matriz, como por exemplo: imprimir todos os seus elementos. O Fragmento de código a seguir ilustra este procedimento, considerando a matriz como sendo do tipo inteiro.

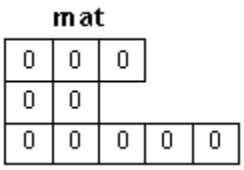
```
def display(mat):
    s = ''
    for i in range(len(mat)):
        for j in range(len(A[i])):
            s += str(mat[i][j]) + ', '
        s +='\n'
    return s
```

Como em Python, matrizes são na verdade lista de listas, podemos ter números diferentes de colunas para cada linha.

Exemplo:

```
mat = [[0] * 3, [0] * 2, [0] * 5 ]
#  alocamos 3 colunas para a linha 0
#  alocamos 2 colunas para a linha 1
#  alocamos 5 colunas para a linha 2
```

A representação esquemática da criação da matriz mat em memória pode ser ilustrada da seguinte forma.



O campo len está disponível para descobrirmos o tamanho dos vetores quando necessário. Para a matriz **mat** anterior, temos:

```
len(mat)    #  retorna 3 ( 3 linhas)
len(mat[0]) #  retorna 3 ( 3 colunas para a linha 0 )
len(mat[1]) #  retorna 2 ( 2 colunas para a linha 1 )
len(mat[2]) #  retorna 5 ( 5 colunas para a linha 2 )
```

Imprimindo uma matriz

O Fragmento de código a seguir ilustra este procedimento.

```
def show(matrix):
    m = len(matrix)
    n = len(matrix[0])
    print("\t")
    for i in range(m):
        linha = ''
        for j in range(n):
            linha += str(matrix[i][j]) + '\t'
        print(linha)
    print("\t")
```

Considerando a matriz como sendo do tipo inteiro, podemos criar um exemplo de ativação da função show

```
# Ativação exemplo
A = [[1,2,3],[-1,1,2]]
show(A)
```

Será impresso

```
1       2       3
-1      1       2
```

Exercícios

1. **Reverso de um vetor**. Escreva uma função que retorna o vetor informado como parâmetro em ordem inversa.

```
def reverse(a):
```

2. **Maior elemento do vetor**. Dados um vetor de inteiros, fazer uma rotina para retornar o maior elemento deste vetor.

```
def  bigger(a):
```

3. **Menor elemento do vetor**. Dados um vetor de inteiros, fazer uma rotina para retornar o menor elemento deste vetor.

```
def smaller(a):
```

4. **Verificação de palímdromo**. Verificar se uma palavra informada como parâmetro é palíndromo. Uma palavra é considerada palíndromo, quando escrita de forma reversa é igual à palavra original. Por exemplo: ROMA e AMOR são palíndromos. Fazer uma função que retorna se duas palavras informadas como parâmetros são ou não palíndromos.

```
def palindromo(p1,p2):
```

   DICA: Utilize a função reverses definida no exemplo 1, sendo que p1 será igual ao reverses(p2)

5. **Produto de Matrizes**. Escreva um programa para efetuar o produto de duas matrizes informadas como parâmetros.

```
def product(m1,  m2):
```

6. **Soma de Matrizes**. Escreva um programa para somar duas matrizes informadas como parâmetros.

```
def add (m1,  m2):
```

7. O que o código a seguir vai imprimir

```
N = 10
a = [0] * N
a[0] = 0
a[1] = 1
for i   in range( N):
     a[i] = a[i-1] + a[i-2]
     print (a[i])
```

8. **Matriz de Hadamard**. Um Matriz é conhecida como matriz de Hadamard de ordem N, denominada de H(N), quando esta matriz é composta somente de elementos booleanos. A propriedade desta matriz está em que duas linhas diferentes quaisquer possuem exatamente N/2 elementos diferentes. Esta matriz é utilizada em tratamento de erros de códigos. H(1) é uma matriz composta por um elemento igual a True. Para N>1 as matrizes H(2N) são obtidas da seguinte forma: primeiro copia-se a matriz H(N) ao longo do quadrado de lado 2N, e finalmente a parte mais inferior a direita que também é uma matriz H(N) deverá ser invertida todos os seus elementos, isto é, se o elemento é True colocar false e caso contrário. A seguir exemplifica-se a matriz H(1), H(2) e H4).

def Hadamard(N):

```
H(1)  H(2)    H(4)      H(8)
----------------------------------------
 T    T T   T T T T    T T T T T T T T
      T F   T F T F    T F T F T F T F
            T T F F    T T T T T T T T
            T F F T    T F T F T F T F
                       T T T T T T T T
                       T F T F T F T F
                       T T T T T T T T
                        T F T F T F T F
```

Jacques Salomon Hadamard (Versalhes, 8 de dezembro de 1865 — Paris, 17 de outubro de 1963) foi um matemático francês.

Exercícios Criativos

1. **Validação de DNA**. Escreva uma função que toma como entrada uma string e retorna true se o string consiste inteiramente de A, C, G e T, e false caso contrário.

   def DNAtest (string):

2. **DNA para RNA**. Os nucleotídeos de RNA (chamados ribonucleotídeos) contêm as bases adenina (A), guanina (G), citosina (C) e uracila (U), mas esta última pirimidina, está presente em lugar de timina que está no DNA. Escreva uma função que recebe uma string de DNA (A, C, G, T) e retorna a seqüência de RNA (A, C, G, U).

   def DNAtoRNA (string):

Material de apoio : http://introcs.cs.princeton.edu/Python/14array/