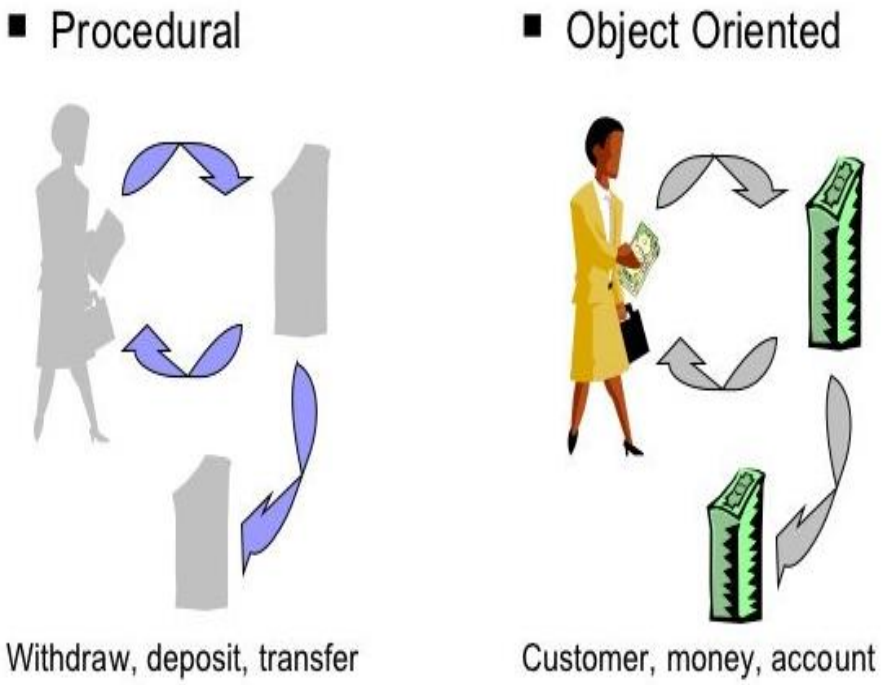
Módulo I – Fundamentos Computacionais
Sessão 6 - Aula 1 - Arrays

Prof. Dr. Luiz Alberto Vieira Dias
Prof. Dr. Lineu Mialaret

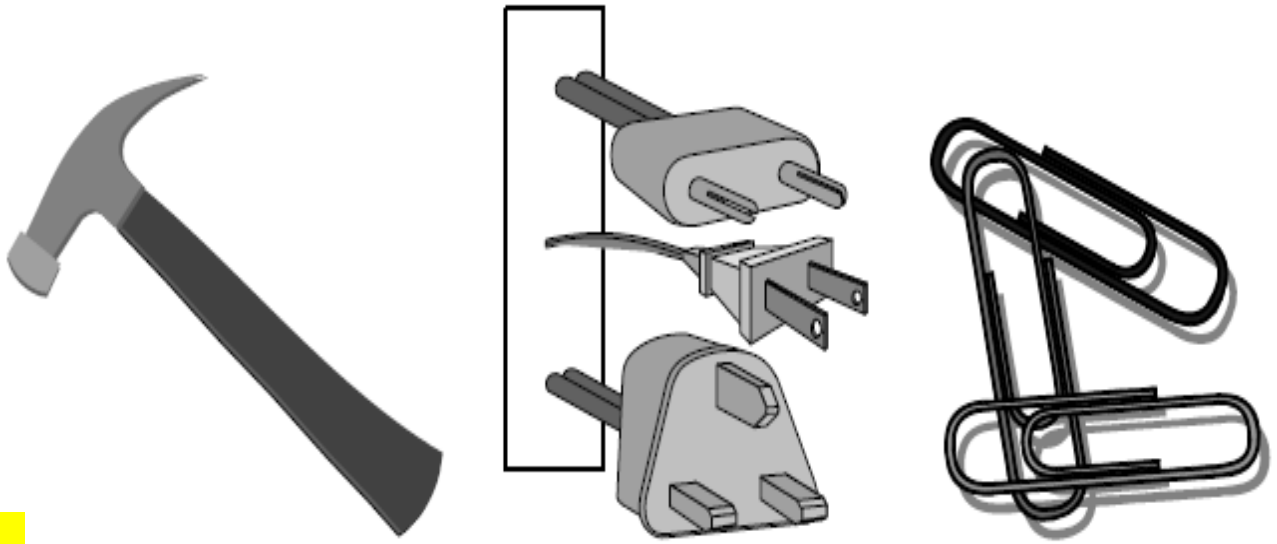
Orientação a Objetos
Características da Orientação a Objetos - OO

- Ability to simulate real-world event much more effectively
- Code is reusable thus less code may have to be written
- Better able to create GUI (graphical user interface) applications
- Programmers are able to produce faster, more accurate and better written applications
- ...



Orientação a Objetos
Metas de Projeto OO

- Software deve ter robustez, adaptabilidade e reusabilidade



Goals of object-oriented design.

Adaptabilidade:
O software precisa ser capaz de evoluir com o tempo, em resposta às mudanças de condições em seu ambiente. Deve ter portabilidade

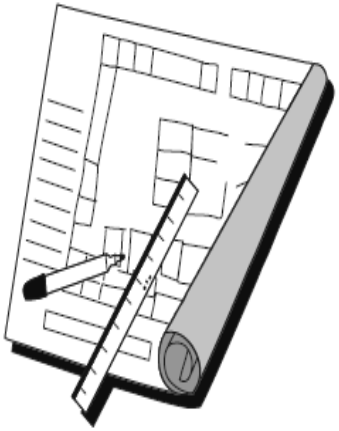
Robustez:
O software deve funcionar corretamente mesmo em condições de anormais

Reusabilidade:
O mesmo código deve ser usado como um componente de diferentes sistemas em várias aplicações

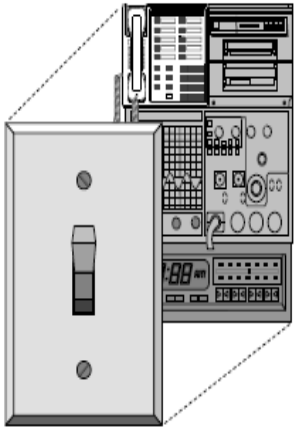
Orientação a Objetos
Princípios do Projeto OO

- Dentre os princípios do Paradigma da OO que se destinam a facilitar as metas descritas acima, destacam-se:
  - Modularidade
  - Abstração
  - Encapsulamento
  - Herança
  - Polimorfismo

Modularity



Abstraction



Encapsulation

Principles of object-oriented design.

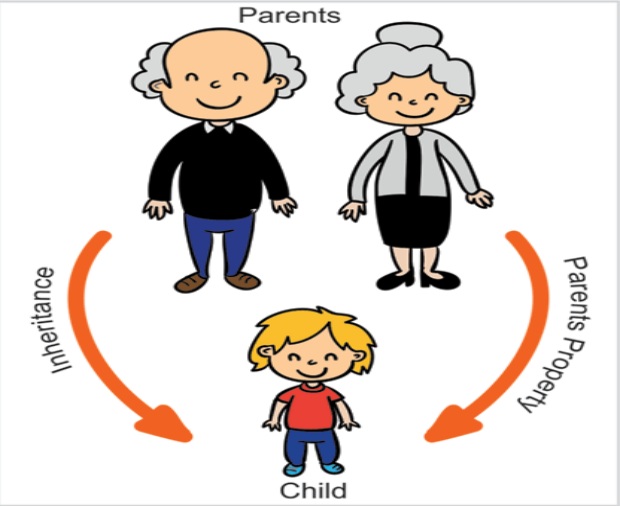
Modularidade refere-se a um princípio no qual diferentes componentes de um sistema de software são divididos em unidades funcionais separadas.

É o modo de se pensar concentrando-se em aspectos essenciais do problema e ignoramos todos os outros aspectos não essenciais

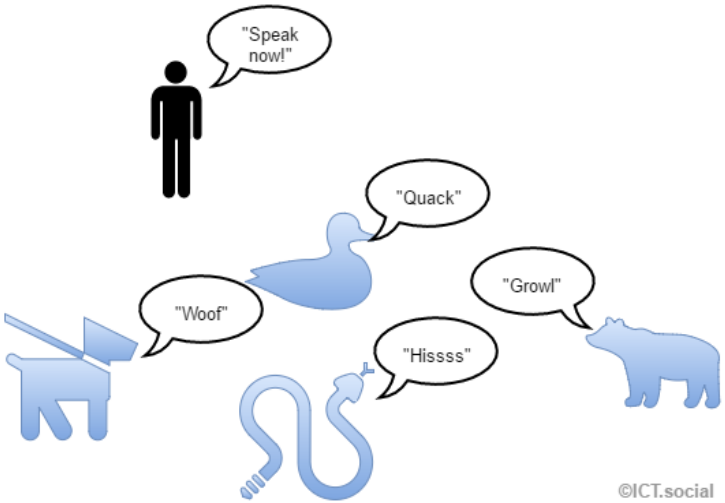
Encapsulamento é reunir em uma entidade estado e funcionalidade.
Ocultamento da informação é restringe à interface pública de um tipo para propósitos de inspeção ou modificação

Orientação a Objetos
Princípios do Projeto OO (cont.)

- Dentre os princípios do Paradigma da OO que se destinam a facilitar as metas descritas acima, destacam-se:
  - Modularidade
  - Abstração
  - Encapsulamento
  - Herança
  - Polimorfismo



Herança é um princípio de orientação a objetos, que permite que classes compartilhem atributos e métodos, através de "heranças". Ela é usada na intenção de reaproveitar código ou comportamento generalizado ou especializar operações ou atributos



Na Orientação a Objetos, o polimorfismo permite um objeto responda a uma mensagem de forma específica a sua identidade

Orientação a Objetos
Princípios do Projeto OO (cont.)

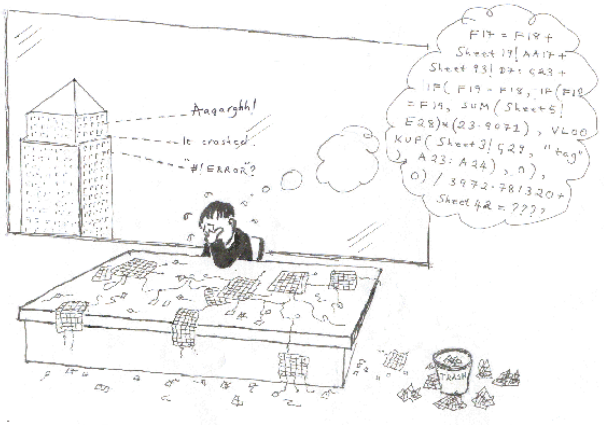
- Um padrão de projeto (*Design Pattern*) descreve uma solução para algum problema típico
  - O padrão disponibiliza um modelo (*template*) para uma solução que pode ser aplicada em muitas diferentes situações
  - Ele descreve os principais elementos da solução de um modo abstrato que pode ser especializado para um problema especifico

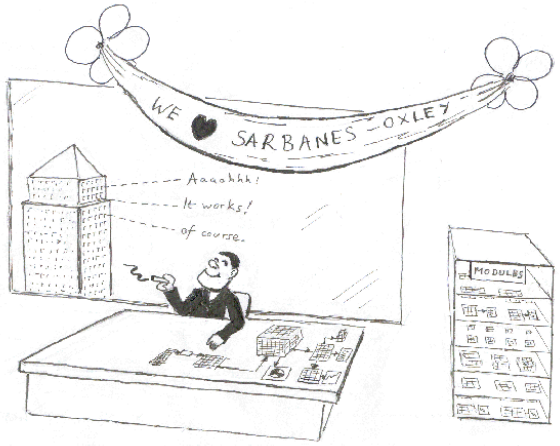
- **Algorithmic patterns:**
- Recursion
- Amortization
- Divide-and-conquer
- Prune-and-search
- Brute force
- Dynamic programming
- The greedy method

- **Software design patterns:**
- Iterator
- Adapter
- Position
- Composition
- Template method
- Locator
- Factory method

Orientação a Objetos
Princípios do Projeto OO - Modularidade

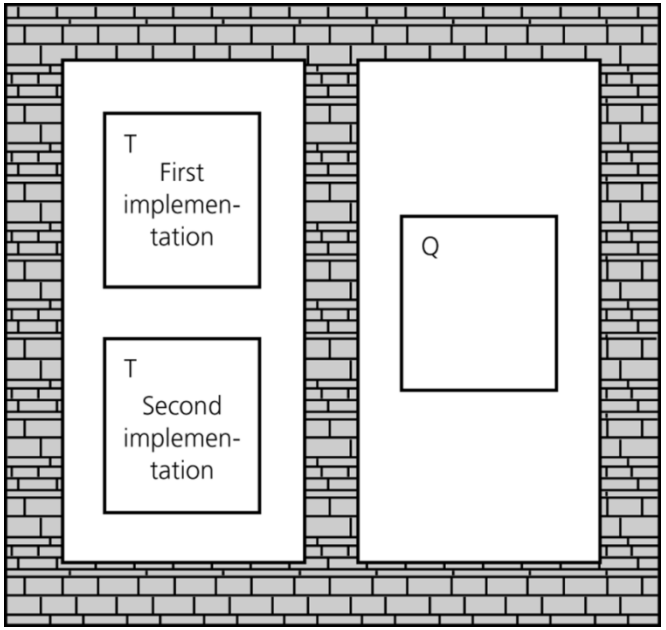
- Modularity (divide a program into manageable parts)
  - Keeps the complexity of a large program manageable
  - Isolates errors
  - Eliminates redundancies
  - Encourages reuse (write libraries)
  - A modular program is
    - Easier to write / Easier to read / Easier to modify





Orientação a Objetos
Princípios do Projeto OO – *Information Hiding*

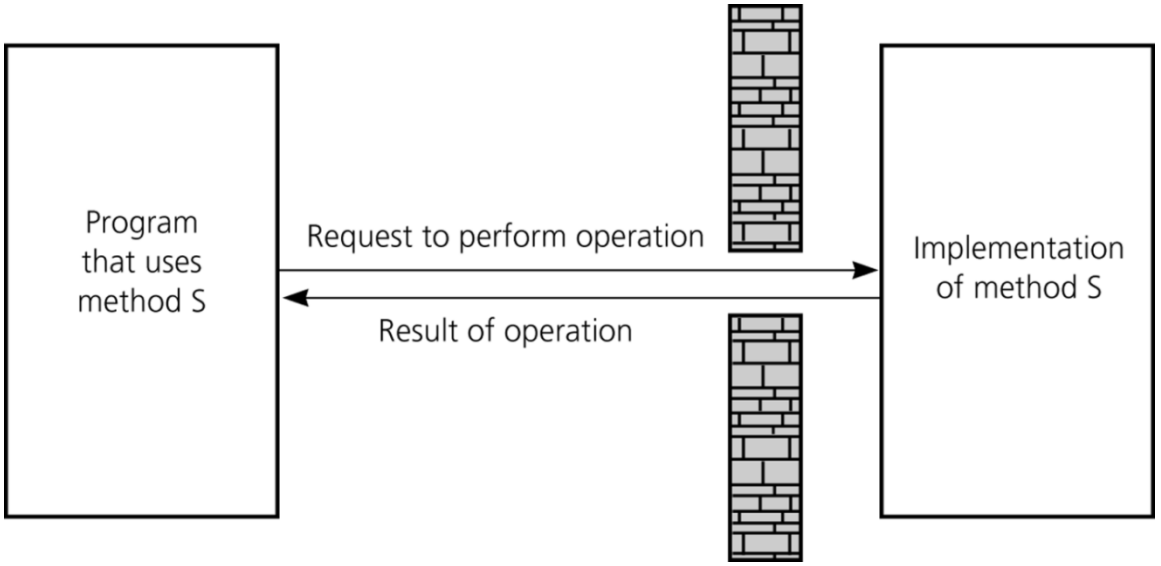
- Hides certain implementation details within a module
- Makes these details inaccessible from outside the module
- Isolate the implementation details of a module from other modules



Isolated tasks: the implementation of task *T* does not affect task *Q*

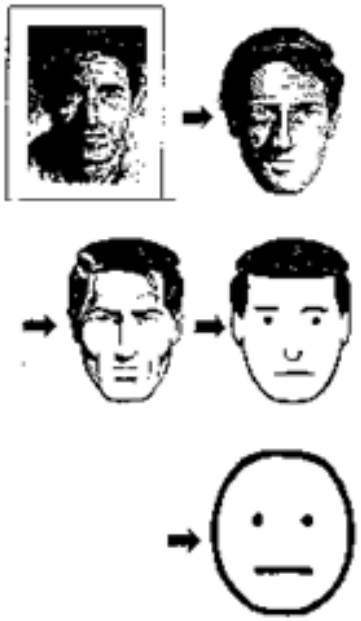
Orientação a Objetos
Princípios do Projeto OO – Isolamento de Módulos

- The isolation of modules is not total (otherwise module would be useless)
  - Methods' specifications, or contracts, govern how they interact with each other
  - Similar to having a wall hiding details, but being able to access through "hole in the wall"



Orientação a Objetos
Princípios do Projeto OO - Abstração

- Separates the purpose and use of a module from its implementation
- A module's specifications should
  - Detail how the module behaves
  - Identify details that can be hidden within the module
-
- Advantages
  - Hides details (easier to use)
  - Allows us to think about the general framework (overall solution) & postpone details for later
  - Can easily replace implementations by better ones



Orientação a Objetos
Princípios do Projeto OO – Abstração de Dados

- Asks you to think ***what*** you can do to a collection of data **independently** of ***how*** you do it
- Allows you to develop each data structure in relative isolation from the rest of the solution
- A natural extension of procedural abstraction

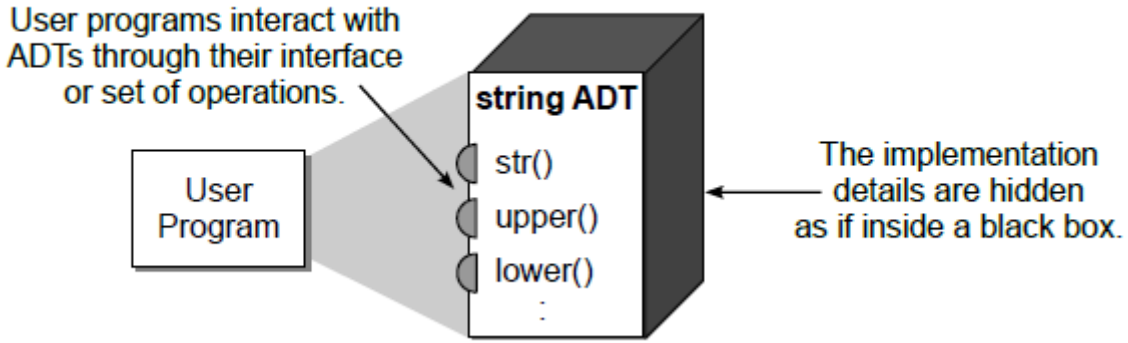
CAR:
Start()
Stop()
TurnLeft()
TurnRight()

NOTE: Implementation can be different for different cars, e.g. automatic transmission

CAR:
Start(){
Select first gear
Release parking brake
Bring clutch up to the friction point
Press gas pedal
Release clutch
}

Orientação a Objetos
Princípios do Projeto OO – Abstração de Dados (cont.)

- Um Tipo Abstrato de Dados (Abstract Data Type – ADT) é uma maneira sistemática de organizar e acessar dados
  - A organização dos dados depende da abstração utilizada
  - O acesso aos dados é feito pela interface por métodos construtores; de acesso; modificadores e iteradores



Constructors: Create and initialize new instances of the ADT.

Accessors: Return data contained in an instance without modifying it.

Mutators: Modify the contents of an ADT instance.

Iterators: Process individual data components sequentially.

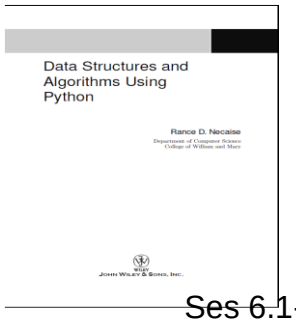
Orientação a Objetos
Princípios do Projeto OO – Abstração de Dados (cont.)

- An ADT is composed of
  - A collection of **data**
  - A set of **operations** on that data
- Specifications of an ADT indicate
  - What the ADT operations do, not how to implement them
- Implementation of an ADT
  - Includes choosing a particular data structure

- ADT is a logical description of how we view the data and the operations that are allowed without regard to how they will be implemented

Orientação a Objetos
Princípios do Projeto OO – Abstração de Dados (cont.)

- A simple ADT is composed of a single or several individually named data fields such as those used to represent a date or rational number
- The complex ADTs are composed of a collection of data values such as the Python list or dictionary
- Complex abstract data types are implemented using a particular data structure, which is the physical representation of how data is organized and manipulated
- Data structures can be characterized by how they store and organize the individual data elements and what operations are available for accessing and manipulating the data



Orientação a Objetos
Princípios do Projeto OO – Abstração de Dados (cont.)

- Linear structures are sequential data collections whose items are **ordered** depending on how they are **added** or **removed** from the structure
- Once an item is added, it stays in that **position** relative to the other elements that came before and came after it
- Linear structures can be thought of as having two **ends**, **top** and **bottom**, (or front and end or front and back)
- What distinguishes one linear structure from another is the way in which **items are added and removed**, in particular the **location** where these **additions** and **removals occur**, e.g., add only to one end, add to both, etc.

Arrays
Definição

- A linguagem Python disponibiliza de diversas classes de sequências
  - Classes *list*, *tuple* e *str*
- Elas têm características comuns
  - Cada uma delas suporta indexação para acesso a um elemento da sequência com uma sintaxe do tipo *seq*[*k*]
  - Elas usam uma estrutura de dados denominada de *array* para representar a sequência

```
>>> lista = ['vermelho']
>>> type (lista)
<class 'list'>
>>> tupla = ('vermelho',)
>>> type (tupla)
<class 'tuple'>
>>> texto = 'vermelho'
>>> type (texto)
<class 'str'>
```

Arrays
Definição (cont.)

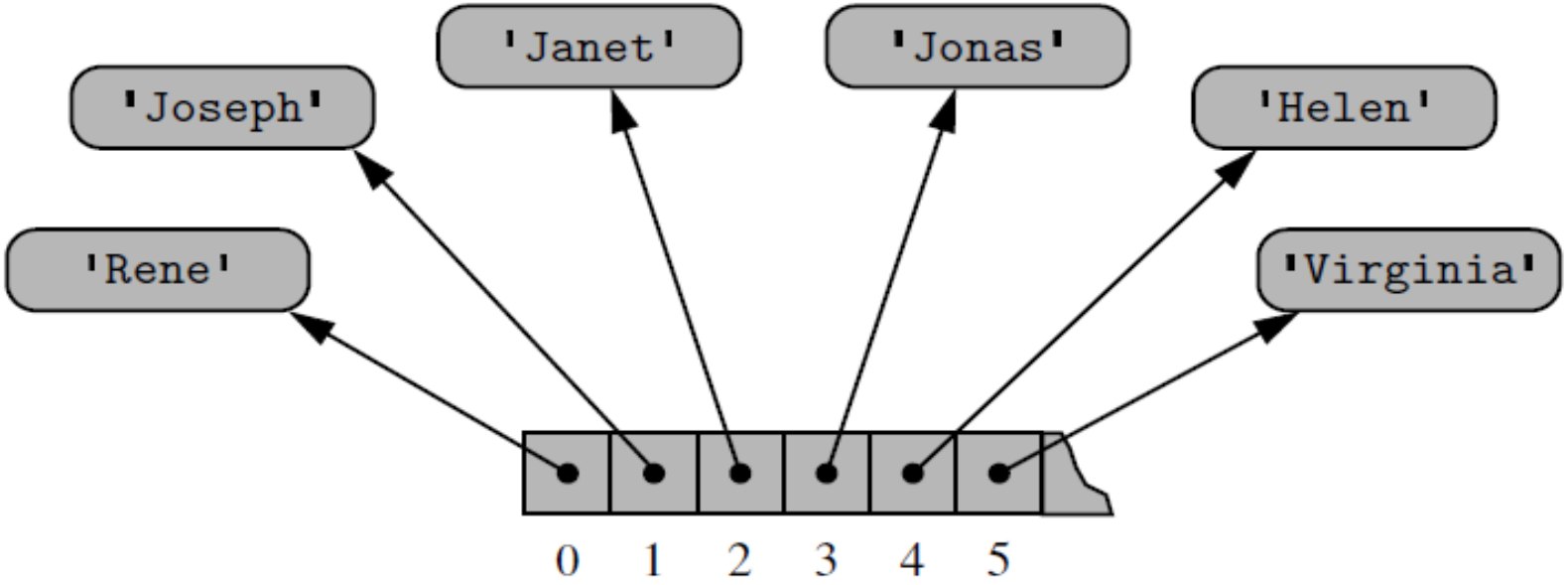
- A memória de um computador é um conjunto de bytes endereçáveis com acesso aleatório e com tempo ***O***(1) para inserção e recuperação de dados
- Um conjunto de posições de memória contíguas se constitui em um *array*

A representation of a portion of a computer's memory, with individual bytes labeled with consecutive memory addresses.

Arrays
Arrays Referenciais

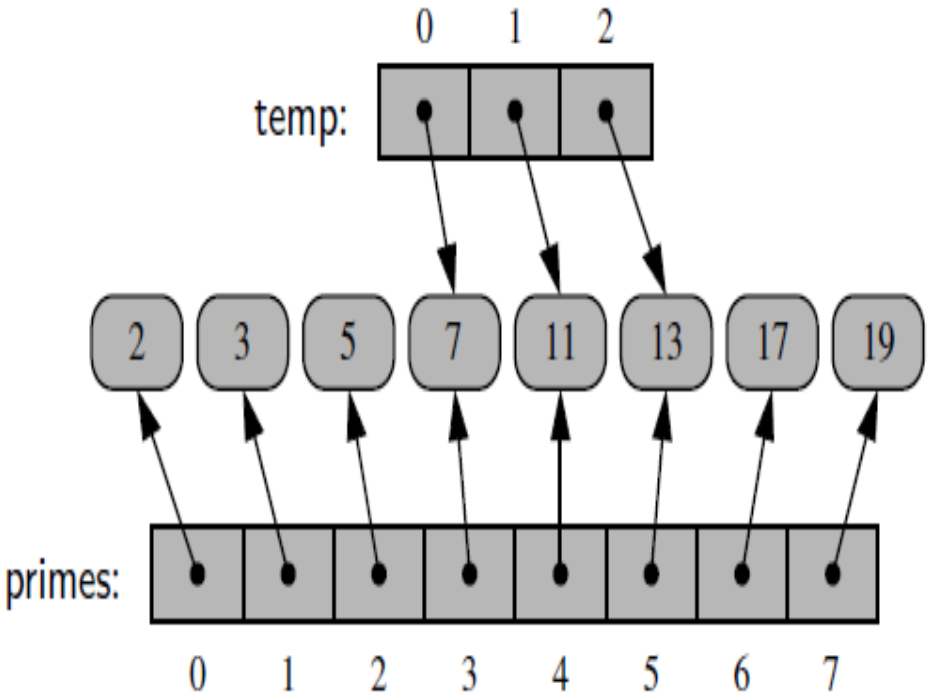
- A linguagem Python representa uma lista ou uma tupla usando um *array* de referências de objetos

```
>>> L = [ 'Rene' , 'Joseph' , 'Janet' , 'Jonas' , 'Helen' , 'Virginia']
>>> L[0]
'Rene'
```



Arrays
Arrays Referenciais (cont.)

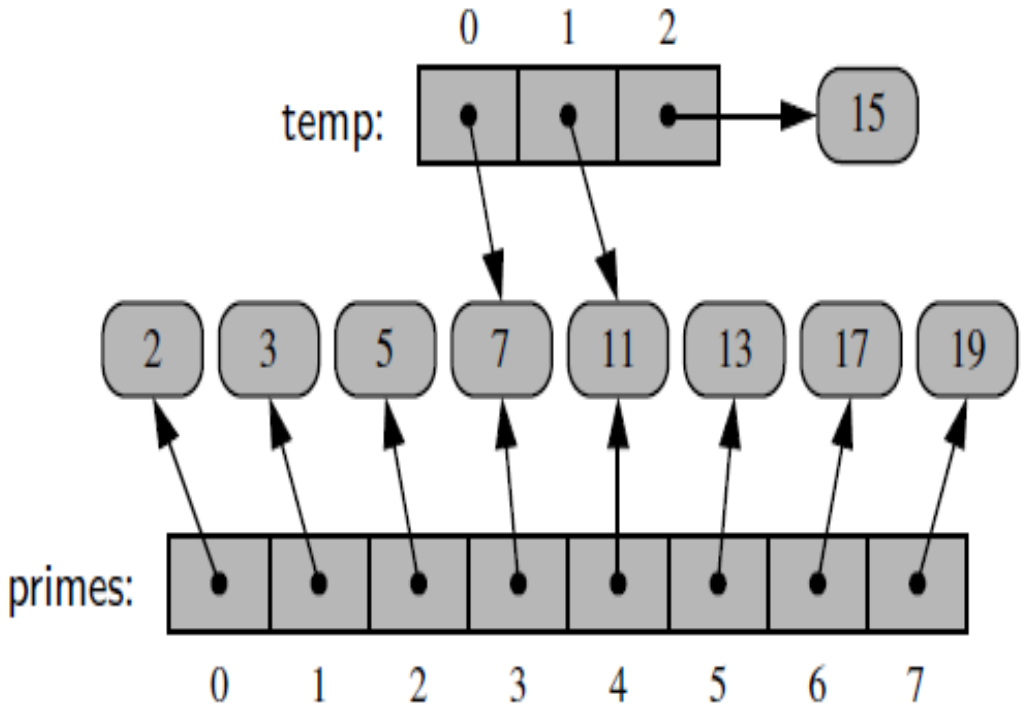
- Quando se computa uma fatia de uma lista, o resultado é uma nova instância de lista, com a nova lista referenciando os mesmos elementos que estão na lista original




```
>>> primes = [2,3,5,7,11,13,17,19]
>>> temp = primes[3:6]
>>> temp
[7, 11, 131]
```

Arrays
Arrays Referenciais (cont.)

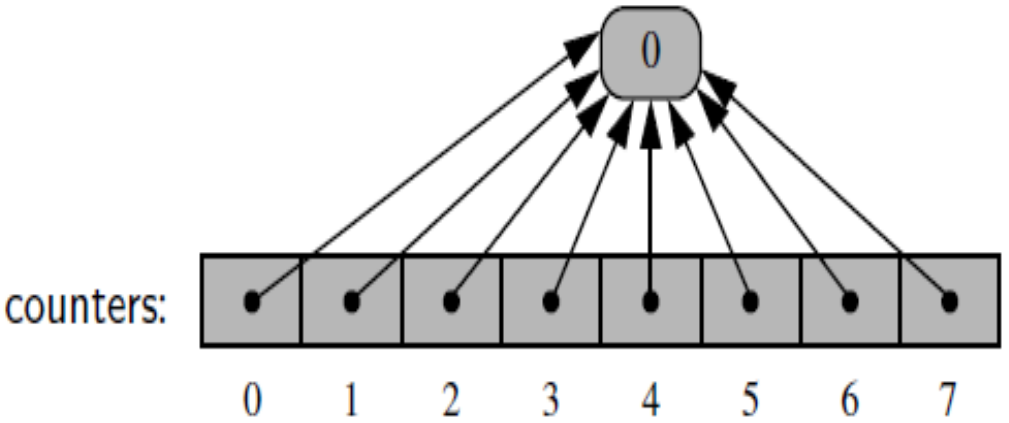
- Quando os elementos da lista são imutáveis, o fato das duas listas compartilharem elementos não causa problema



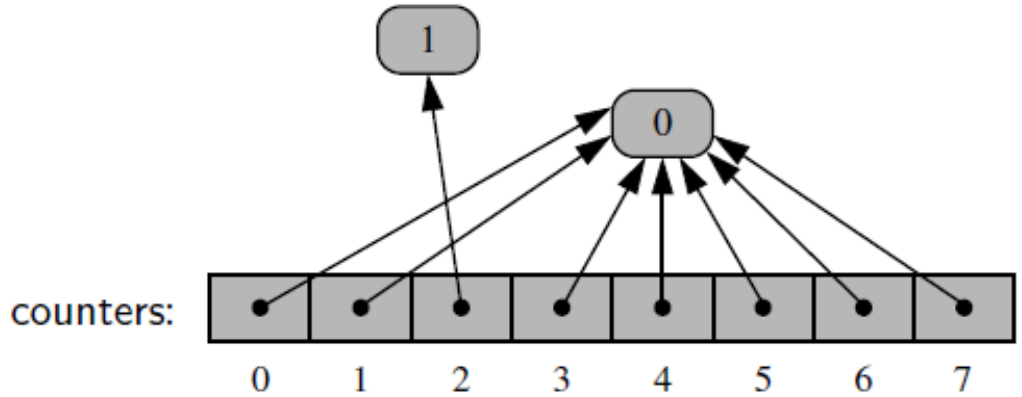

```
>>> temp[2]=15
>>> temp
[7, 11, 15]
```

Arrays
Arrays Referenciais (cont.)

- É uma prática comum no Python a inicialização de *arrays* de inteiros



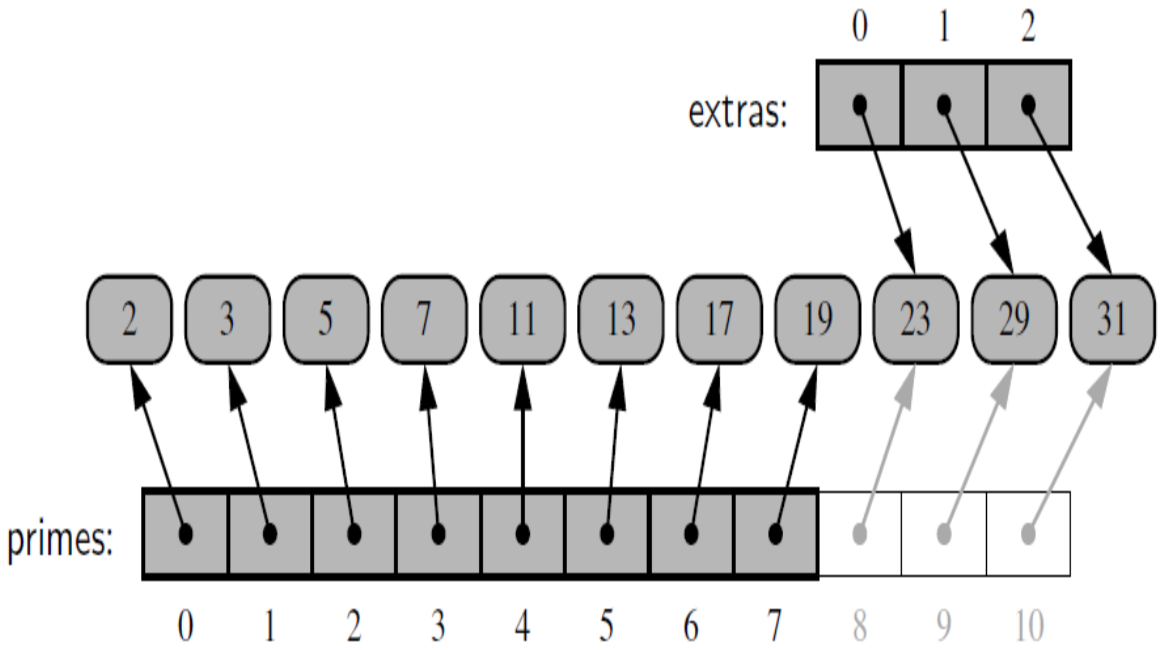

```
>>> counters = [0]*8
>>> counters
[0, 0, 0, 0, 0, 0, 0, 0]
```




```
>>> counters[2] +=1
>>> counters
[0, 0, 1, 0, 0, 0, 0, 0]
```

Arrays
Arrays Referenciais (cont.)

- O método *extend* é utilizado para adicionar todos os elementos de uma lista para o final de outra lista. A lista estendida recebe referências dos elementos acrescentados




```
>>> extras = [23,29,31]
>>> primes
[2, 3, 5, 7, 11, 131, 7, 19]
>>> primes.extend(extras)
>>> primes
[2, 3, 5, 7, 11, 131, 7, 19, 23,
29, 31]
```

Arrays
Arrays Compactos

- De forma diferente de *arrays* de referências, *arrays* compactos armazenam o próprio objeto
  - Isto gera economia espaço de armazenamento
  - Há melhoria de performance, pois os dados são armazenados de forma contígua na memória
- O suporte para *arrays* compactos é dado pelo módulo *array*
  - Ele define uma classe, também denominada de *array* que fornece armazenagem compacta para *arrays* de tipos de dados primitivos

Integers stored compactly as elements of a Python array.

Arrays
Arrays Compactos (cont.)

- A interface pública da classe *array* é similar a da classe *list*
  - Entretanto o construtor da classe requer o tipo de dado como parâmetro inicial

```
>>> from array import array
>>> primess = array('i',[2,3,5,7,11,17,19])
>>> primess
array('i', [2, 3, 5, 7, 11, 17, 19])
```

Integers stored compactly as elements of a Python array.

Arrays
Arrays Compactos (cont.)

- Pode-se criar Os seguintes tipos de códigos são utilizados pra a criação de *arrays* compactos

Code	C Data Type	Typical Number of Bytes
'b'	signed char	1
'B'	unsigned char	1
'u'	Unicode char	2 or 4
'h'	signed short int	2
'H'	unsigned short int	2
'i'	signed int	2 or 4
'I'	unsigned int	2 or 4
'l'	signed long int	4
'L'	unsigned long int	4
'f'	float	4
'd'	float	8

Arrays
Arrays Referenciais sem a classe List

The `ctypes` Module

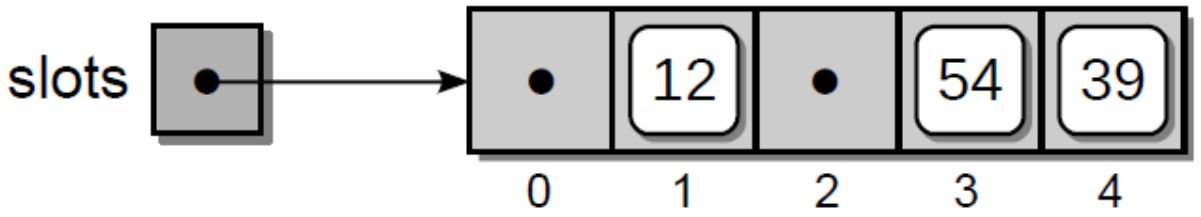
Many of the data types and classes available in Python are actually implemented using appropriate types from the C language. While Python does not provide the array structure as part of the language itself, it now includes the `ctypes` module as part of the Python Standard Library. This module provides access to the diverse set of data types available in the C language and the complete functionality provided by a wide range of C libraries.

The `ctypes` module provides the capability to create hardware-supported arrays just like the ones used to implement Python's string, list, tuple, and dictionary collection types. But the `ctypes` module is not meant for everyday use in Python

Arrays
Arrays Referenciais sem a classe List (cont.)

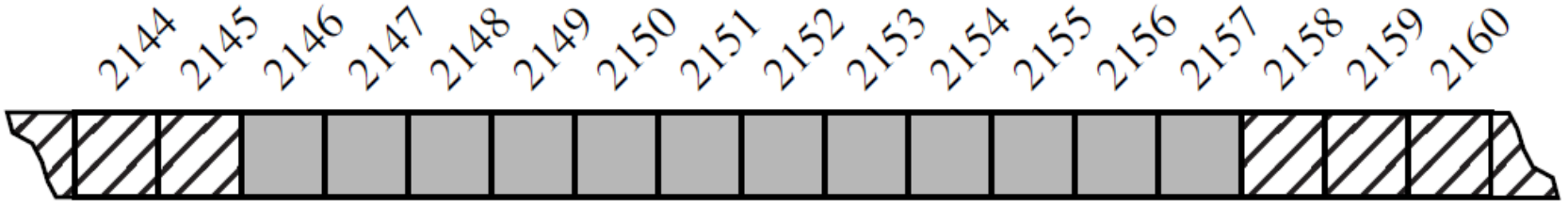
- Pode-se criar *arrays* referenciais com o módulo *ctypes*

```
>>> import ctypes
>>> ArrayType = ctypes.py_object * 5
>>> slots = ArrayType()
>>> for i in range( 5 ) : slots[i]=None
...
>>> slots[1] = 12
>>> slots[3] = 54
>>> slots[4] = 37
>>> type (slots)
<class '__main__.py_object_Array_5'>
```



Arrays
Arrays Dinâmicos

- Ao se criar um *array*, seu tamanho exato deve ser explicitamente declarado para que o sistema aloque corretamente um trecho de memória contígua para o seu armazenamento



An array of 12 bytes allocated in memory locations 2146 through 2157.

- Uma vez que o sistema operacional dedica as posições de memória vizinhas para alocar outros dados, a capacidade de armazenamento do *array* não aumenta
  - Para os tipos *tuple* e *str* isto não representa problema

Arrays
Arrays Dinâmicos (cont.)

- A classe list, implementada em *array*, ainda que tenha um determinado tamanho quando é criada, permite a adição de elementos na lista sem nenhum limite sobre a capacidade geral da lista
  - Ela é implementada no que se chama de *array* dinâmico
- O *array* dinâmico
  - Mantém uma capacidade adicional de espaço quando é criado, ou seja, se o usuário cria uma lista com 5 elementos, o sistema reserva espaço adicional na lista para 8 elementos
  - Quando a capacidade de inserção da lista se esgota, o sistema cria uma nova lista com maior capacidade, contendo a lista antiga

Arrays
Arrays Dinâmicos (cont.)

- Capacidade adicional da classe *list*

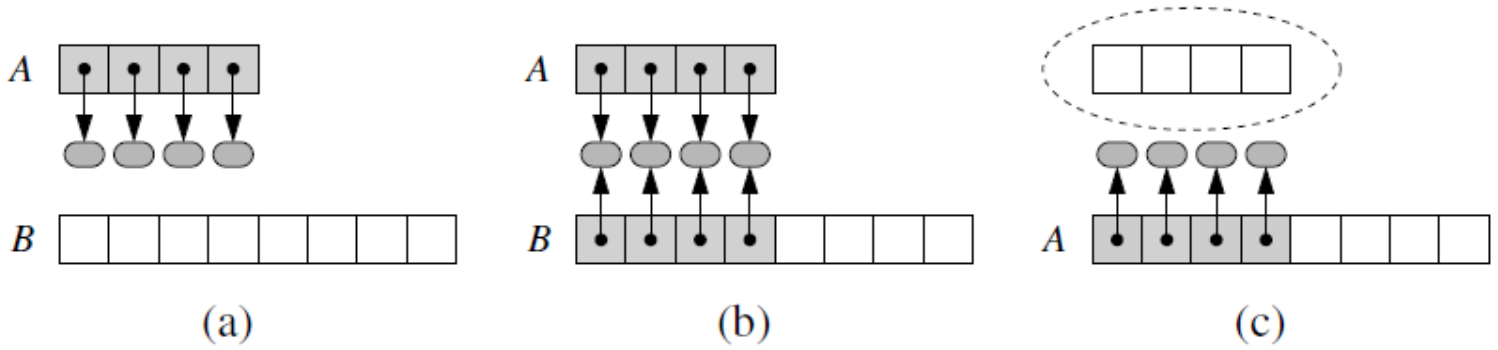
```
import sys                          # provides getsizeof function
data = []
for k in range(n):                  # NOTE: must fix choice of n
  a = len(data)                     # number of elements
  b = sys.getsizeof(data)           # actual size in bytes
  print('Length: {0:3d}; Size in bytes: {1:4d}'.format(a, b))
  data.append(None)                 # increase length by one
```

```
Length:   0; Size in bytes:   72
Length:   1; Size in bytes:  104
Length:   2; Size in bytes:  104
Length:   3; Size in bytes:  104
Length:   4; Size in bytes:  104
Length:   5; Size in bytes:  136
Length:   6; Size in bytes:  136
Length:   7; Size in bytes:  136
Length:   8; Size in bytes:  136
Length:   9; Size in bytes:  200
Length:  10; Size in bytes:  200
Length:  11; Size in bytes:  200
Length:  12; Size in bytes:  200
Length:  13; Size in bytes:  200
Length:  14; Size in bytes:  200
Length:  15; Size in bytes:  200
Length:  16; Size in bytes:  200
Length:  17; Size in bytes:  272
Length:  18; Size in bytes:  272
Length:  19; Size in bytes:  272
Length:  20; Size in bytes:  272
Length:  21; Size in bytes:  272
Length:  22; Size in bytes:  272
Length:  23; Size in bytes:  272
Length:  24; Size in bytes:  272
Length:  25; Size in bytes:  272
Length:  26; Size in bytes:  352
```

Arrays
Arrays Dinâmicos (cont.)

- If an element is appended to a list at a time when the underlying array is full, we perform the following steps

1. Allocate a new array B with larger capacity.
2. Set $B[i] = A[i]$, for $i = 0, \ldots, n-1$, where n denotes current number of items.
3. Set $A = B$, that is, we henceforth use B as the array supporting the list.
4. Insert the new element in the new array.

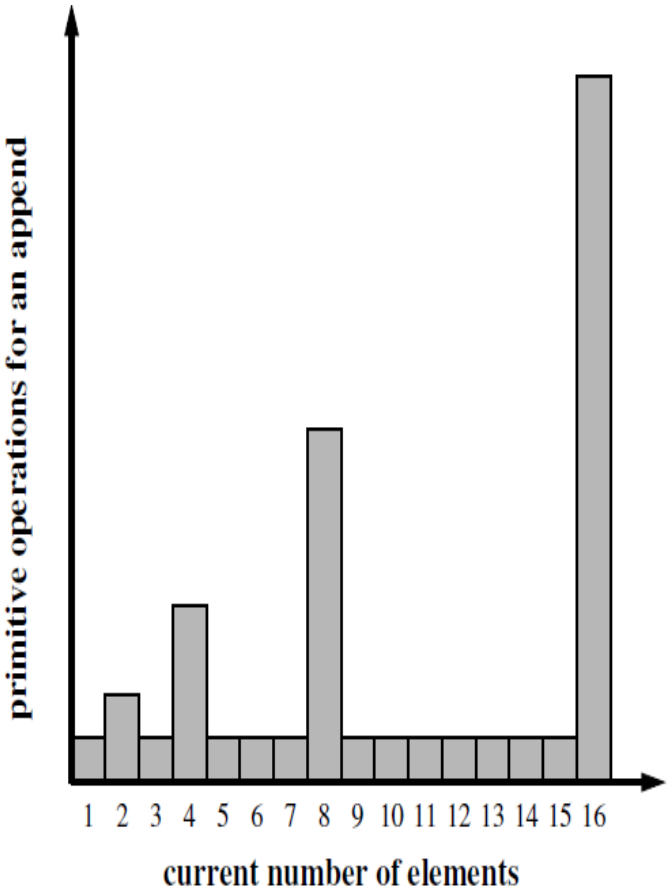


An illustration of the three steps for "growing" a dynamic array: (a) create new array B; (b) store elements of A in B; (c) reassign reference A to the new array. Not shown is the future garbage collection of the old array, or the insertion of the new element.

Arrays
Arrays Dinâmicos (cont.)

- O array dinâmico ainda tem desempenho bom, mesmo dobrando de tamanho

Proposition 5.1: *Let S be a sequence implemented by means of a dynamic array with initial capacity one, using the strategy of doubling the array size when full. The total time to perform a series of n append operations in S, starting from S being empty, is $O(n)$.*



Running times of a series of append operations on a dynamic array.

Arrays
Eficiência dos Tipos Sequenciais

Operation	Running Time
len(data)	$O(1)$
data[j]	$O(1)$
data.count(value)	$O(n)$
data.index(value)	$O(k+1)$
value **in** data	$O(k+1)$
data1 == data2 (similarly !=, <, <=, >, >=)	$O(k+1)$
data[j:k]	$O(k-j+1)$
data1 + data2	$O(n_1+n_2)$
c * data	$O(cn)$

Asymptotic performance of the nonmutating behaviors of the list and tuple classes. Identifiers data, data1, and data2 designate instances of the list or tuple class, and n, n_1, and n_2 their respective lengths. For the containment check and index method, k represents the index of the leftmost occurrence (with $k = n$ if there is no occurrence). For comparisons between two sequences, we let k denote the leftmost index at which they disagree or else $k = \min(n_1, n_2)$.

Arrays
Eficiência dos Tipos Sequenciais (cont.)

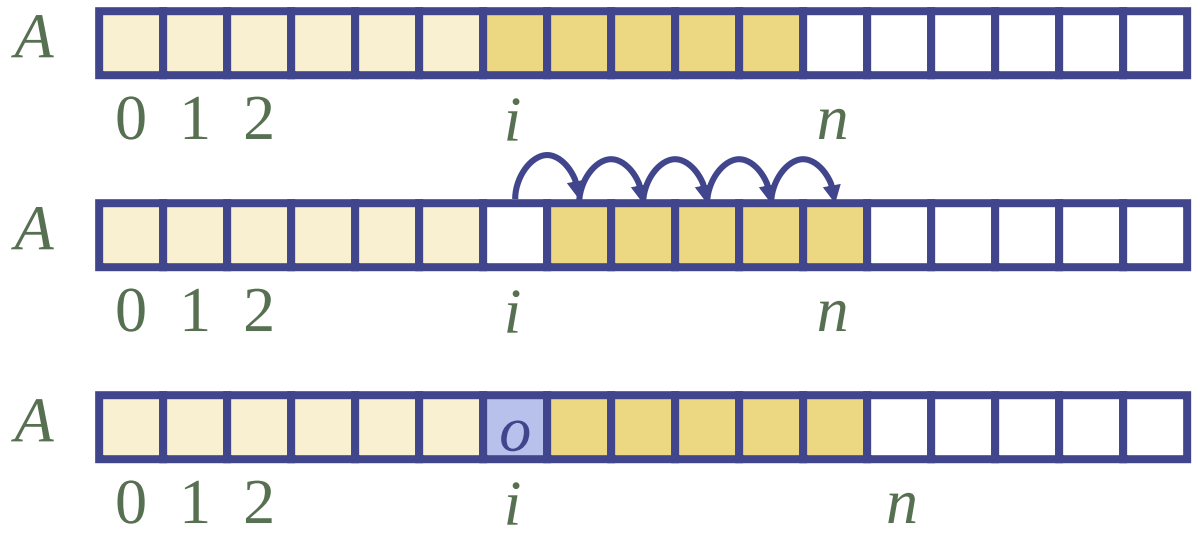
Operation	Running Time
data[j] = val	$O(1)$
data.append(value)	$O(1)^*$
data.insert(k, value)	$O(n-k+1)^*$
data.pop( )	$O(1)^*$
data.pop(k) **del** data[k]	$O(n-k)^*$
data.remove(value)	$O(n)^*$
data1.extend(data2) data1 += data2	$O(n_2)^*$
data.reverse( )	$O(n)$
data.sort( )	$O(n \log n)$

*amortized

Asymptotic performance of the mutating behaviors of the list class. Identifiers data, data1, and data2 designate instances of the list class, and n, n_1, and n_2 their respective lengths.

Arrays
Eficiência dos Tipos Sequenciais (cont.)

- A classe *list* suporta um método *insert*(*k*, *value*) que insere um dado valor na lista no índice $0 \leq k \leq n$, enquanto realiza o deslocamento de todos subsequentes elementos por uma célula
- No pior caso ($k = 0$) isto tem um tempo ***O*(*n*)**



Arrays
Eficiência dos Tipos Sequenciais (cont.)

- Resultados experimentais comprovam o pior caso na inserção

	N				
	100	1,000	10,000	100,000	1,000,000
$k = 0$	0.482	0.765	4.014	36.643	351.590
$k = n // 2$	0.451	0.577	2.191	17.873	175.383
$k = n$	0.420	0.422	0.395	0.389	0.397

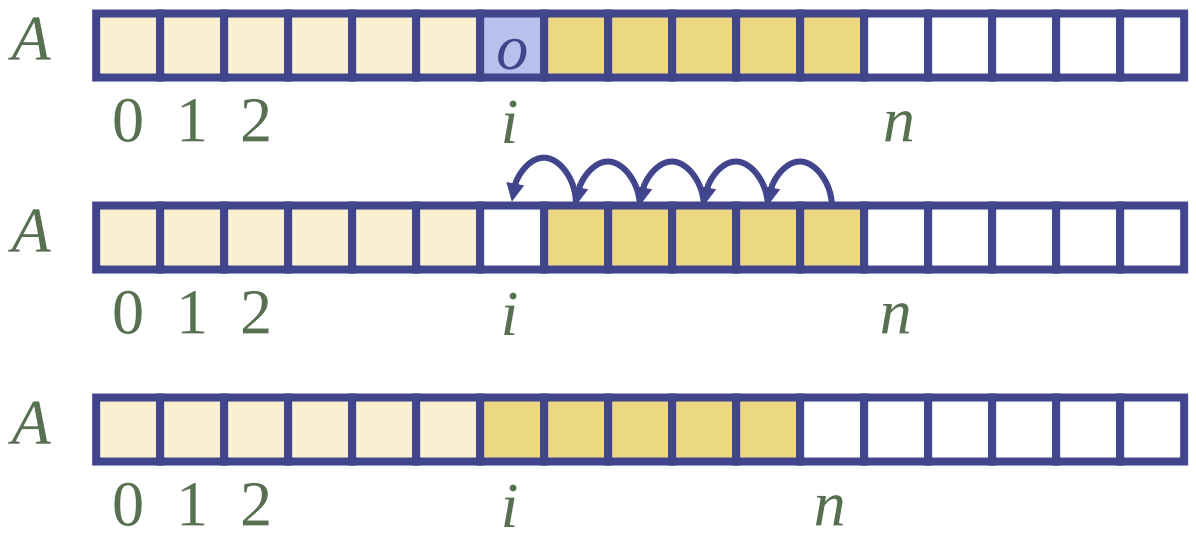
O(*n*)

O(1)

Average running time of insert(k, val), measured in microseconds, as observed over a sequence of N calls, starting with an empty list. We let n denote the size of the current list (as opposed to the final list).

Arrays
Eficiência dos Tipos Sequenciais (cont.)

- A classe *list* suporta o método *pop*()
  - *pop*() remove o último elemento da lista → ***O***(1)
  - *pop*(*k*) remove o elemento no índice *k* → ***O*(*n* – *k*)**



Arrays
Arrays Multidimensionais

- As classes *list*, *tuple* e *str* utilizam apenas um único índice para acessar cada elemento da sequência (*one-dimensional*)
- Diversas aplicações envolvem dados multidimensionais
  - Um *array* de duas dimensões é chamado de matriz
  - Pode-se utilizar dois índices, *i* e *j*, para se referenciar as células da matriz
  - O primeiro índice faz referência a um número de linha e o segundo índice faz referência a um número de coluna
  - Usa-se o operador de referência na forma [][] para acesso às células da matriz

Arrays
Arrays Multidimensionais (cont.)

	0	1	2	3	4	5	6	7	8	9
0	22	18	709	5	33	10	4	56	82	440
1	45	32	830	120	750	660	13	77	20	105
2	4	880	45	66	61	28	650	7	510	67
3	940	12	36	3	20	100	306	590	0	500
4	50	65	42	49	88	25	70	126	83	288
5	398	233	5	83	59	232	49	8	365	90
6	33	58	632	87	94	5	59	204	120	829
7	62	394	3	4	102	140	183	390	16	26

Illustration of a two-dimensional integer data set, which has 8 rows and 10 columns. The rows and columns are zero-indexed. If this data set were named stores, the value of stores[3][5] is 100 and the value of stores[6][2] is 632.

Arrays
Arrays Multidimensionais (cont.)

- No Python se representa uma matriz como uma lista de listas
  - Representa-se um *array* de duas dimensões (matriz) como uma lista de linhas, com cada linha sendo uma lista de valores
- Seja a seguinte matriz abaixo

22	18	709	5	33
45	32	830	120	750
4	880	45	66	61

- No Python é criada da seguinte forma

```
>>> data = [ [22, 18, 709, 5, 33], [45, 32, 830, 120, 750], [4, 880, 45, 66, 61] ]
>>> type (data)
<class 'list'>
```

Arrays
Arrays Multidimensionais (cont.)

- Uma vantagem desta representação é que pode-se utilizar a sintaxe
  - *data*[1][3] para representar o valor que está na linha índice 1 e coluna índice 3
  - *data*[1] para representar a segunda entrada da lista

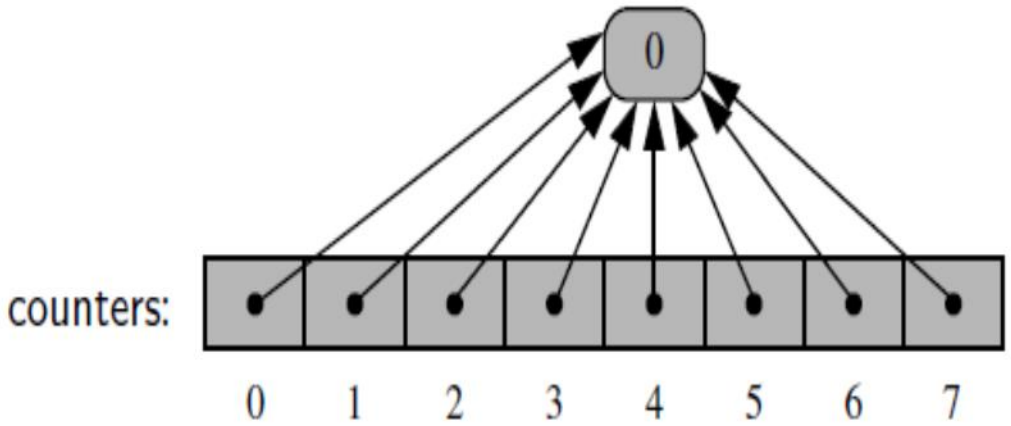
22	18	709	5	33
45	32	830	120	750
4	880	45	66	61

```
>>> data[1][3]
120
>>> data[1]
[45, 32, 830, 120, 750]
```

Arrays
Arrays Multidimensionais (cont.)

- Para se inicializar rapidamente uma lista unidimensional, pode-se fazer da seguinte forma

```
>>> data = [0]*10
>>> data
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```



Arrays
Arrays Multidimensionais (cont.)

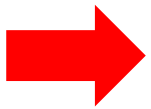
- Para se inicializar rapidamente uma lista multidimensional, com *r* linhas e *c* colunas, poderia se tentar fazer da seguinte forma

```
>>> # data = ([0]*c)*r
...
>>> data = ([0]*3)*3
>>> data
[0, 0, 0, 0, 0, 0, 0, 0, 0]
```

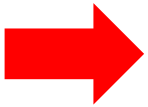
Arrays
Arrays Multidimensionais (cont.)

- Para se inicializar rapidamente uma lista multidimensional, com *r* linhas e *c* colunas, poderia se tentar fazer da seguinte forma

```
>>> # data = ([0]*c)*r
...
>>> data = ([0]*3)*3
>>> data
[0, 0, 0, 0, 0, 0, 0, 0, 0]
```



```
>>> data = ([0]*3)
>>> data
[0, 0, 0]
>>> data = data*3
>>> data
[0, 0, 0, 0, 0, 0, 0, 0, 0]
```



Arrays
Arrays Multidimensionais (cont.)

- Para se inicializar rapidamente uma lista multidimensional, com *r* linhas e *c* colunas, poderia se tentar fazer da seguinte nova forma

```
>>> # data = [[0]*c]*r
...
>>> data = [[0]*6]*3
>>> data
[[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]]
```

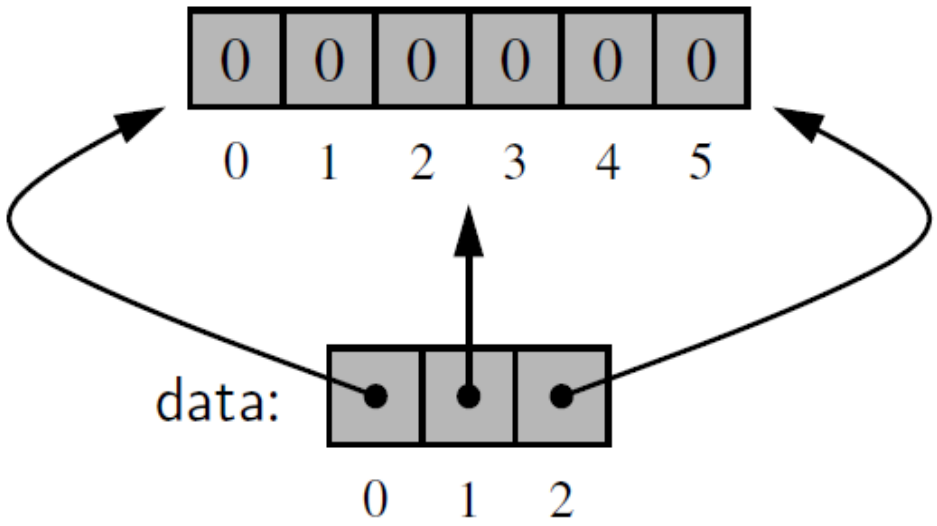
Arrays
Arrays Multidimensionais (cont.)

- Para se inicializar rapidamente uma lista multidimensional, com *r* linhas e *c* colunas, poderia se tentar fazer da seguinte nova forma

```
>>> # data = [[0]*c]*r
...
>>> data = [[0]*6]*3
>>> data
[[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]]
```

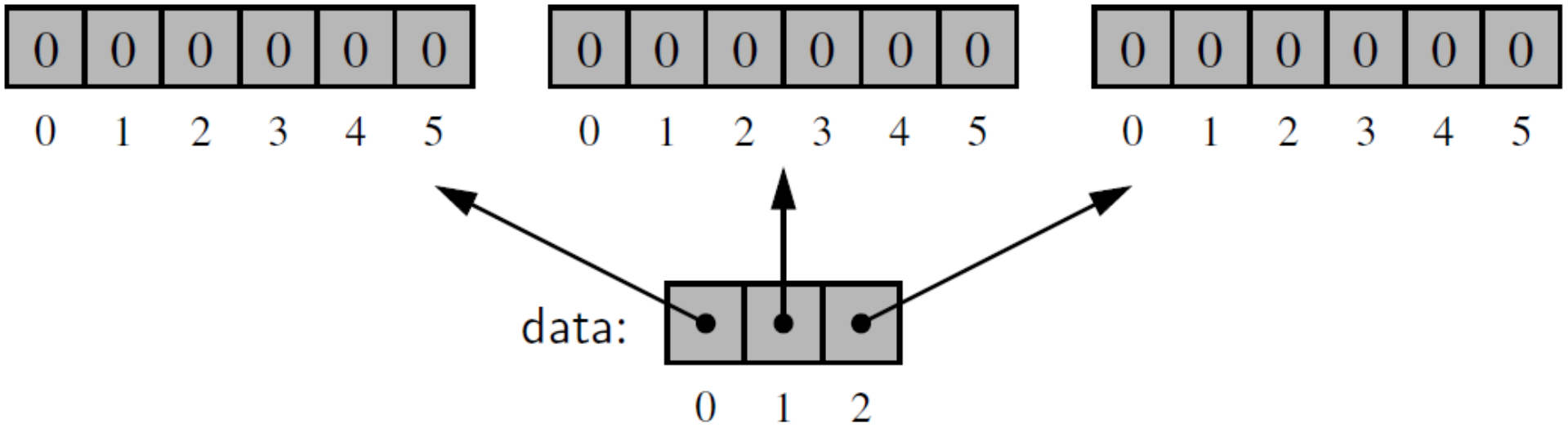
```
>>> data
[[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]]
>>> data [2][0] = 100
>>> data
[[100, 0, 0, 0, 0, 0], [100, 0, 0, 0, 0, 0], [100, 0, 0, 0, 0, 0]]
```

Arrays
Arrays Multidimensionais (cont.)



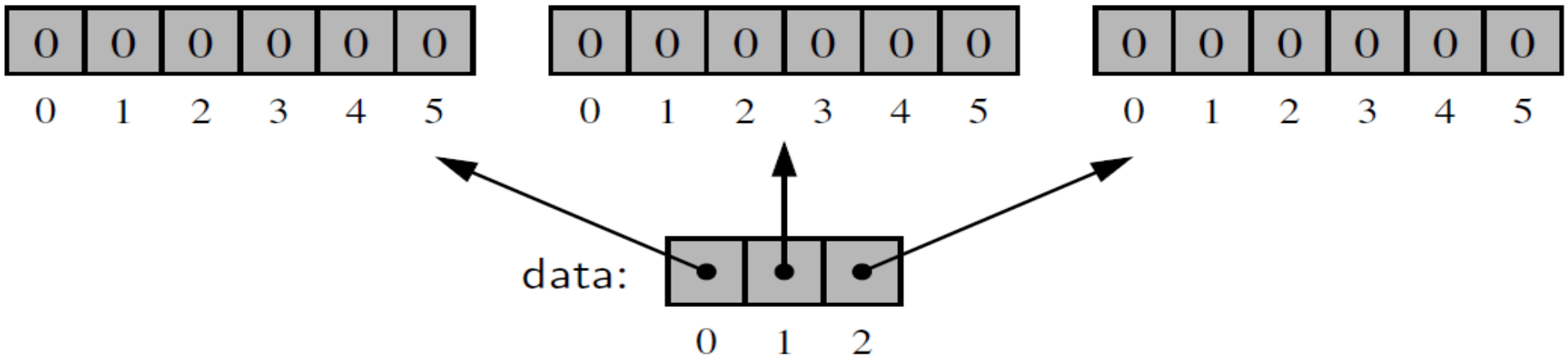
A flawed representation of a 3×6 data set as a list of lists, created with the command `data = [ [0] * 6 ] * 3`. (For simplicity, we overlook the fact that the values in the secondary list are referential.)

Arrays
Arrays Multidimensionais (cont.)



A valid representation of a 3×6 data set as a list of lists. (For simplicity, we overlook the fact that the values in the secondary lists are referential.)

Arrays
Arrays Multidimensionais (cont.)



A valid representation of a 3×6 data set as a list of lists. (For simplicity, we overlook the fact that the values in the secondary lists are referential.)

```
n = 3
m = 6
data = [0] * n
for i in range(n):
    data[i] = [0] * m
data [2][0] = 100
print(data)
```

```
========== RESTART: C:/Python-36/Codigo-Python-PEDS/matrix1.py ==========
[[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [100, 0, 0, 0, 0, 0]]
>>>
```