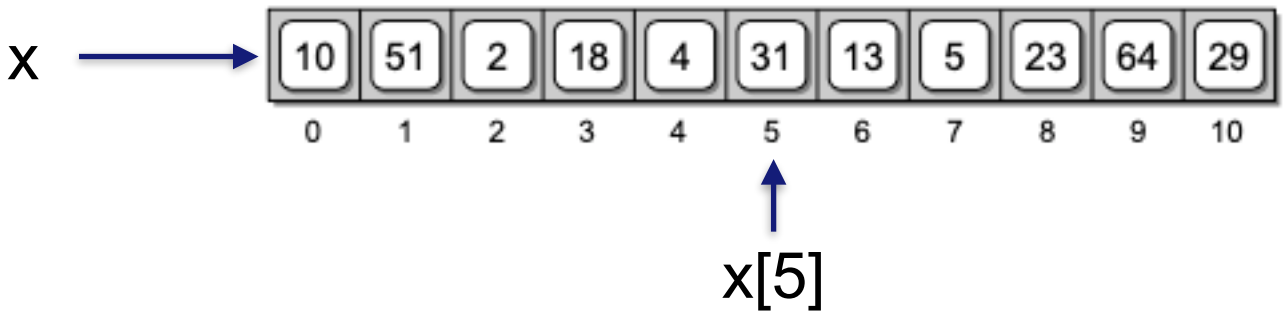
SUPSI

Arrays

Based on the book “Data structures and algorithms using Python” by Rance D. Necaise

The Array structure

- Arrays are available as one-dimensional structures at the hardware level
- A one-dimensional array:
  - is organised in contiguous elements of memory
  - all the elements are the same type (won't be the case here!)
  - the elements can be randomly accessed
  - the variable name refers to the whole memory containing the elements



Arrays and Python lists

- They are similar but:
  - in an array I can only set a value and read a value of a cell
  - the size of the array is usually fixed
  - in a Python list there are many operations allowed
  - the list can grow and shrink

- Arrays are useful when the maximum size is known in advance and it won't change
- Lists are useful when the number of elements in it is not known and it is subject to dynamically vary

- In the arrays we trade the flexibility for efficiency (memory and speed)

- values = [None] * 100'000 creates a list with 100'000 elements

The array data structure

- The Python language does not offer arrays! We implement a one-dimensional array as a collection of contiguous elements in which individual elements are identified by a unique integer subscript starting with zero. Once an array is created, its size cannot be changed.
  - `Array(size)`: Creates a one-dimensional array consisting of size elements with each element initially set to None. size must be greater than zero.
  - `length()`: Returns the length or number of elements in the array.
  - `getitem (index)`: Returns the value stored in the array at element position index. The index argument must be within the valid range. Accessed using the subscript operator.
  - `setitem (index, value)`: Modifies the contents of the array element at position index to contain value. The index must be within the valid range. Accessed using the subscript operator.
  - `clear(value)`: Clears the array by setting every element to value.
  - `iterator()`: Creates and returns an iterator that can be used to traverse the elements of the array.

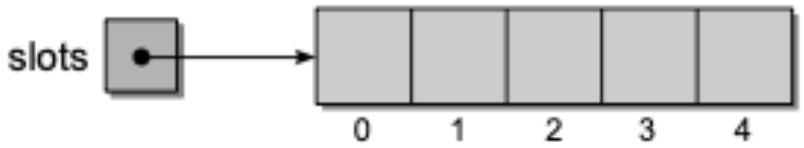
Example of use of the Array ADT

```
from array import Array
import random

# Fill a 1-D array with random values, then print them, one
per line.

# The constructor is called to create the array.
valueList = Array( 100 )

# Fill the array with random floating-point values.
for i in range(len(valueList)) :
    valueList[i] = random.random()

# Print the values, one per line.
for value in valueList :
    print( value )
```

Implementing the Array ADT (1)

- Most of of the data types available in Python are implemented in C!
- Python does not offer an array structure, but it offers the `ctypes` module
- We can use the types module to create a hardware supported array, powerful and efficient

```
import ctypes
ArrayType = ctypes.py_object * 5
    slots = ArrayType()
```



The slots are not yet initialised

Implementing the Array ADT (2)

- If we try to print the content of the slots we would get an exception

```
print(slots[0])
```

ValueError: PyObject is NULL

- We must initialise the values

```
for i in range(5):
    slots[i] = None
```

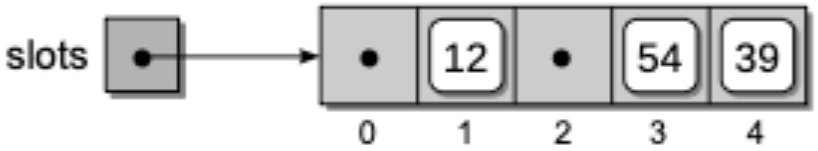
- Remember:
  - The hardware-supported array does not keep track of its size
  - The programmer must ensure not to access elements outside the range

Implementing the array ADT (3)

- If we assign the following values

```
slots[1] = 12
slots[3] = 54
slots[4] = 37
```

- We get this situation



- To remove an element from the array, we set its value to None

```
slots[3] = None
```

The class definition: the constructor

```
# Implements the Array ADT using array capabilities of the ctypes module.
import ctypes
class Array :
    # Creates an array with size elements.
    def __init__( self, size ):
        assert size > 0, "Array size must be > 0"
        self._size = size
        # Create the array structure using the ctypes module.
        PyArrayType = ctypes.py_object * size
        self._elements = PyArrayType()
        # Initialize each element.
        self.clear( None )
```

The class definition: setter and getter

```
# Puts the value in the array element at index position.
 def __setitem__( self, index, value ):
     assert index >= 0 and index < len(self), "Array subscript out of range"
     self._elements[ index ] = value

# Gets the contents of the index element.
 def __getitem__( self, index ):
     assert index >= 0 and index < len(self), "Array subscript out of range"
     return self._elements[ index ]
```

The class definition: utilities

```
# Returns the size of the array.
 def __len__( self ):
     return self._size

# Clears the array by setting each element to the given value.
 def clear( self, value ):
   for i in range( len(self) ):
     self._elements[i] = value

# Returns the array's iterator for traversing the elements.
 def __iter__( self ):
     return _ArrayIterator( self._elements )
```

The class definition: the iterator

```
  # An iterator for the Array ADT.
class _ArrayIterator :
    def __init__(self, theArray):
        self._arrayRef = theArray
        self._curNdx = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self._curNdx < len(self._arrayRef):
            entry = self._arrayRef[self._curNdx]
            self._curNdx += 1
            return entry
        else:
            raise StopIteration
```

Hands-on activity (20 min)

- Take a look at the code of the Array class presented in the textbook and implement it in two Python files
  - main.py - containing the main body that will use the implemented functions
  - array.py - containing the code provided in the slides and on the textbook
- Is important that you read the code that has been provided and you understand its content.
- BEWARE: there may be a few mistakes in the code of the book, fix them!

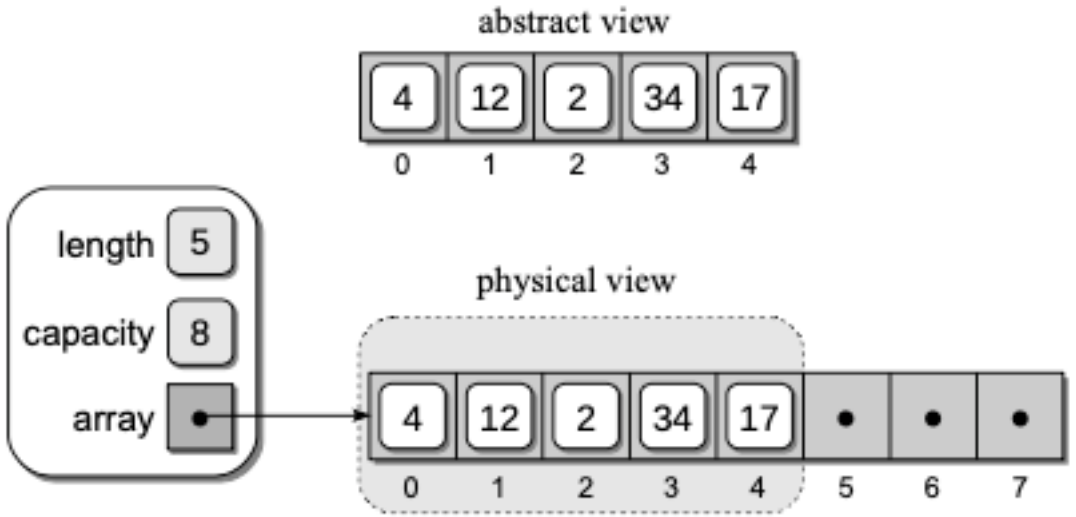
A list refresher

The Python list

- The Python list is a **mutable** sequence container
- Creating a list:

```
pyList = [ 4, 12, 2, 34, 17 ]
```

- The instruction above calls the `list()` constructor and fills the list with the above values.



Appending items

- If there is enough residual capacity the instruction

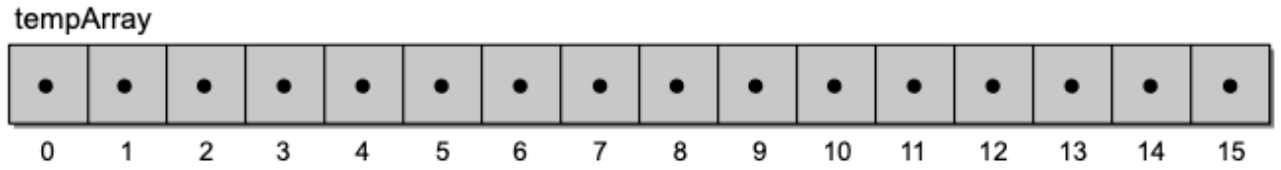
```
pyList.append(50)
```

- adds an element at the end of the list

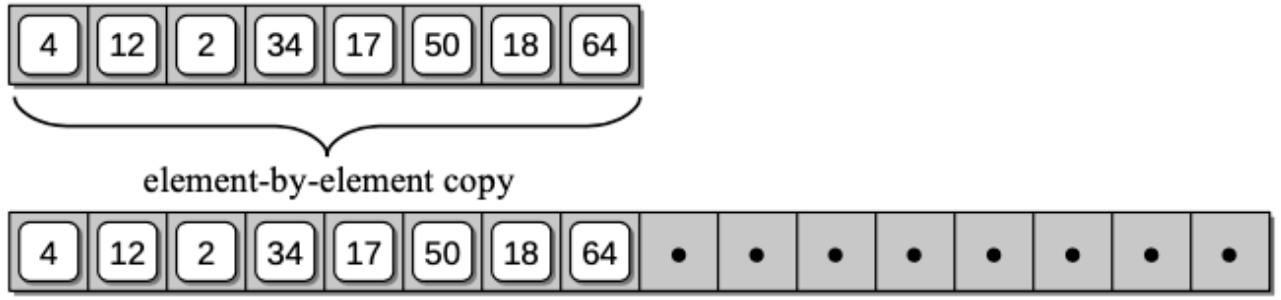
- else the list is expanded to create space for the extra values:
  - a new array is created with additional capacity
  - the items are copied from the old array to the new array
  - the new larger array is set as data storage for the list
  - the original array is destroyed

Appending items (2)

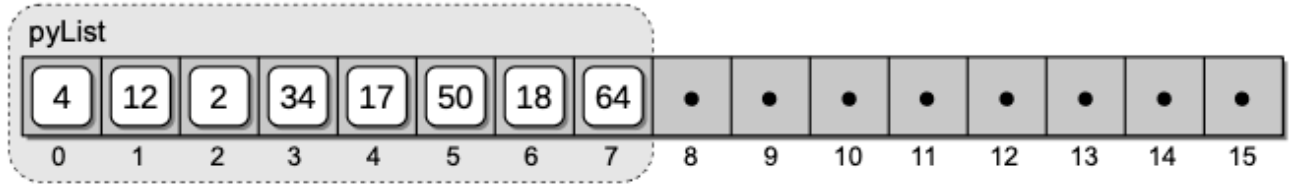
(1) A new array, double the size of the original, is created.



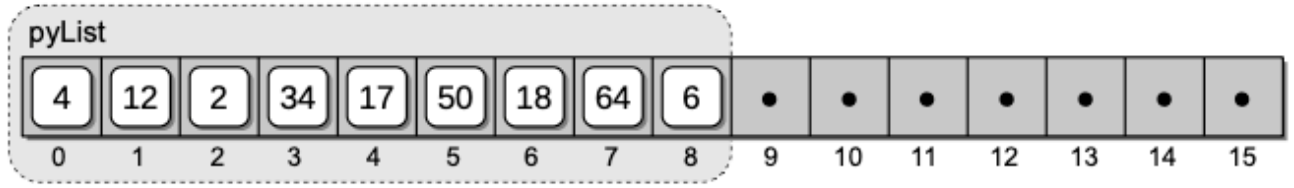
(2) The values from the original array are copied to the new larger array.



(3) The new array replaces the original in the list.



(4) Value 6 is appended to the end of the list.

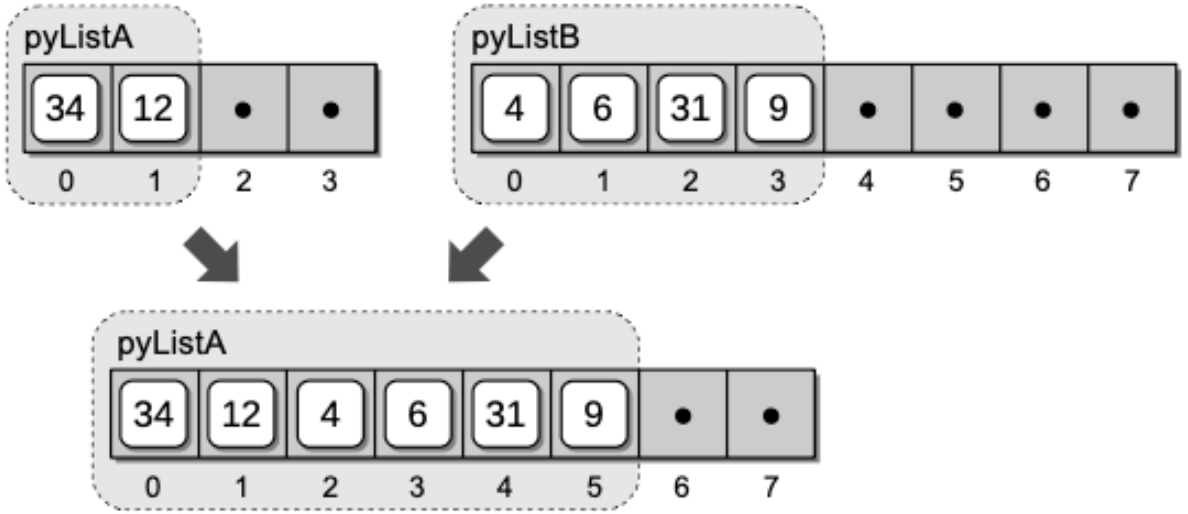


Extending a list

- A list can be appended to an existing list using the `extend()` method

```
pyListA = [ 34, 12 ]
pyListB = [ 4, 6, 31, 9 ]
pyListA.extend( pyListB )
```

- If there is no space in pyListA for all the elements of pyListB, then it is expanded as in the previous case

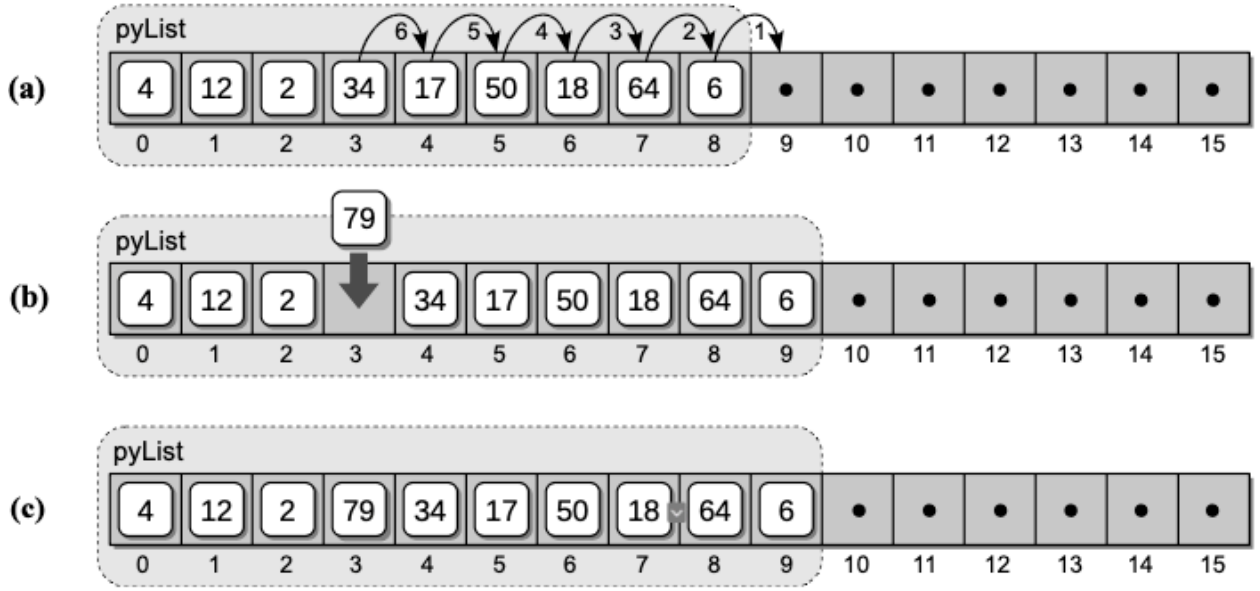


Inserting values in a list

- If pyList= [4, 12, 2, 34, 17, 50, 18, 64, 6]
- Inserting the value 79 in position 3 (which is actually the fourth)

```
pyList.insert(3, 79)
```

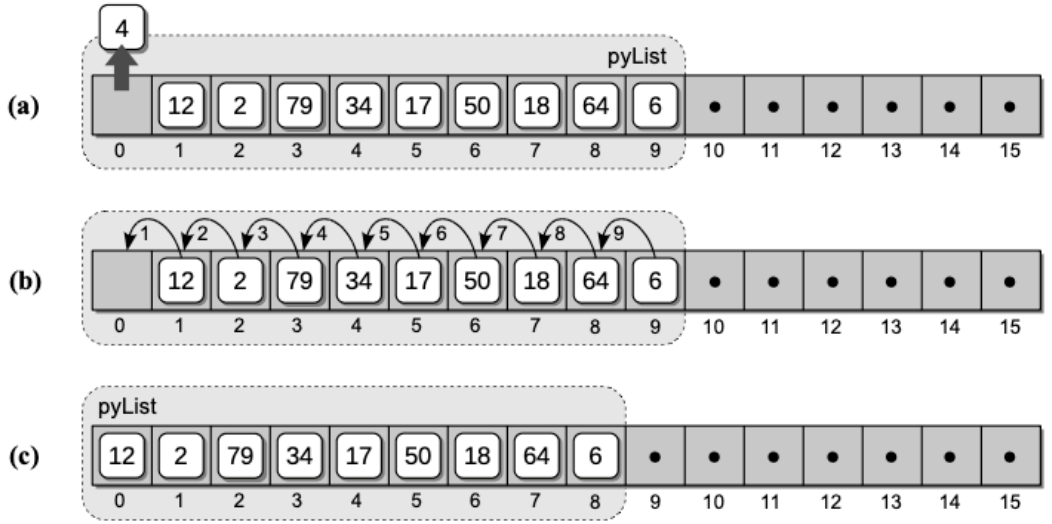
- Causes pyList to become:
- [4, 12, 2, 79, 34, 17, 50, 18, 64, 6]



Removing items with pop

```
pyList.pop(0) # remove the first item
pyList.pop() # remove the last item
```

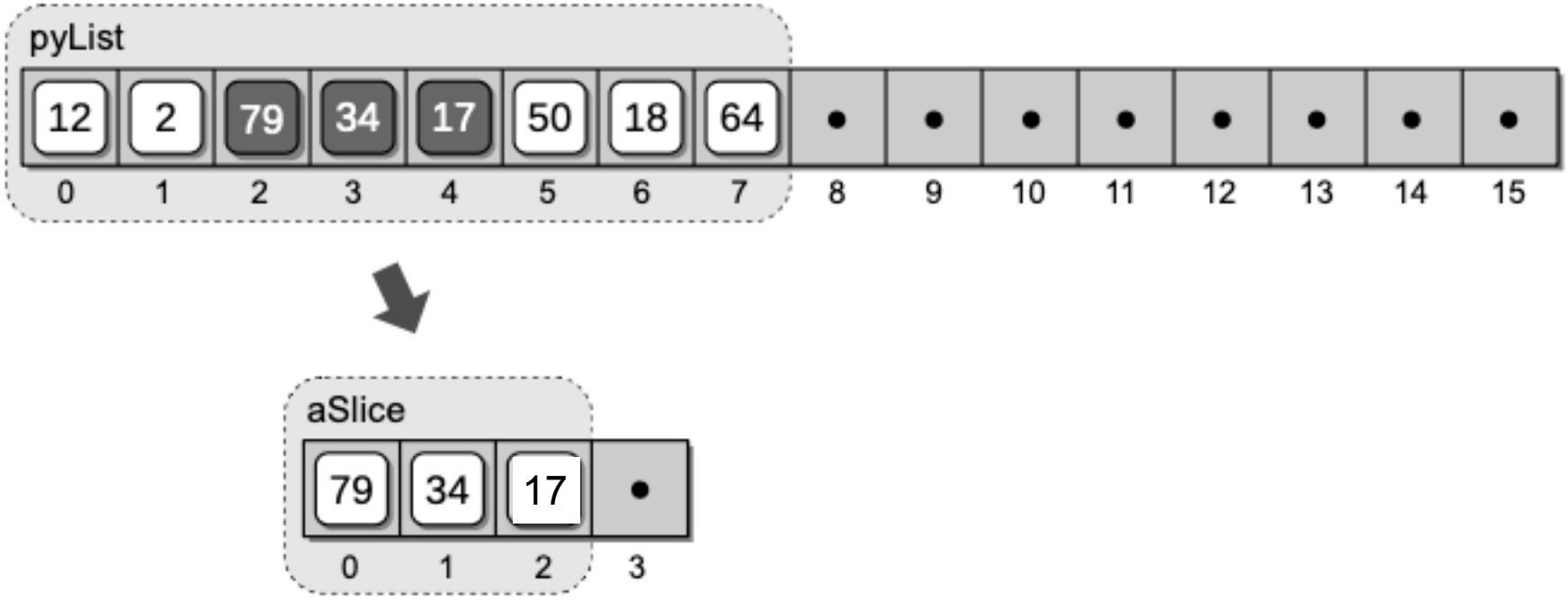
- After the item is removed, typically by setting the reference variable to None, the items following it within the array are shifted down, from left to right, to close the gap.
- Finally, the length of the list is decremented to reflect the smaller size



List slices

- Slicing a list does not modify the original list
  - Provided the elements are primitive types!
- The referenced elements are copied and put in a new list

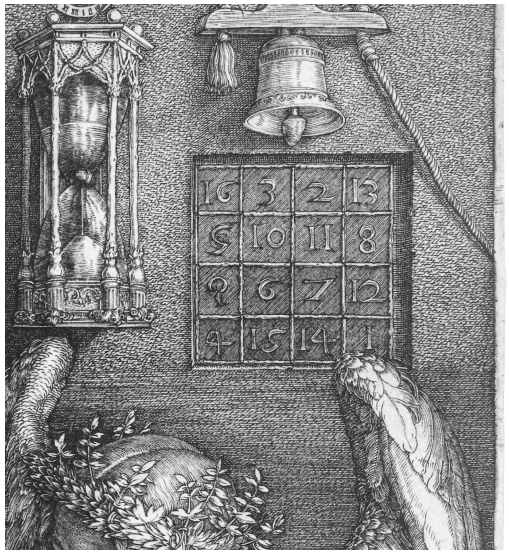
```
aSlice = pyList[2:5]
```

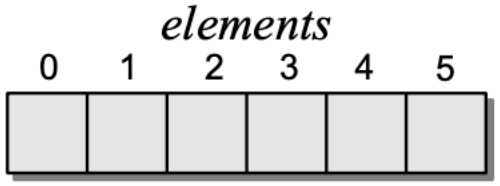


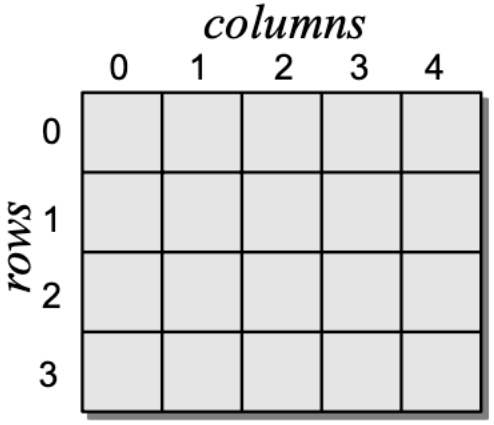
Two-dimensional arrays

Multi-dimensional arrays are useful

- Matrices, tensors, voxels, hypercubes and so on are all structures which can be represented with multi-dimensional arrays
- In the case of **2-D arrays** individual elements are accessed by specifying two indices, one for the row and one for the column, [i,j]
- Multi-D arrays usually are not hardware supported







The Array2D Abstract Data Type

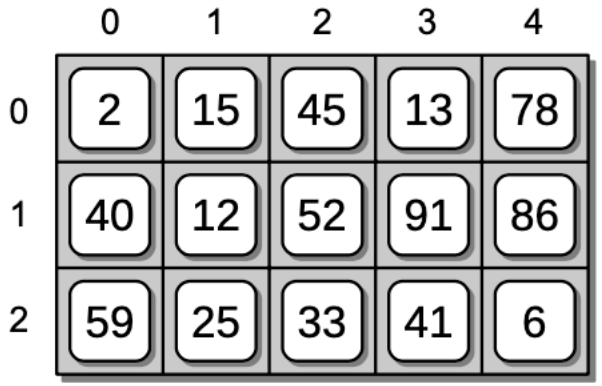
- A two-dimensional array consists of a collection of elements organized into rows and columns. Individual elements are referenced by specifying the specific row and column indices (r, c), both of which start at 0.
  - `Array2D( nrows, ncols )`: Creates a two-dimensional array organized into rows and columns. The nrows and ncols arguments indicate the size of the table. The individual elements of the table are initialized to None.
  - `numRows()`: Returns the number of rows in the 2-D array.
  - `numCols()`: Returns the number of columns in the 2-D array.
  - `clear( value )`: Clears the array by setting each element to the given value.
  - `getitem( i1, i2 )`: Returns the value stored in the 2-D array element at the position indicated by the 2-tuple (i1,i2), both of which must be within the valid range. Accessed using the subscript operator: y = x[1,2].
  - `setitem(i1, i2, value)`: Modifies the contents of the 2-D array element indicated by the 2-tuple (i1,i2) with the new value. Both indices must be within the valid range. Accessed using the subscript operator: x[0,3] = y.

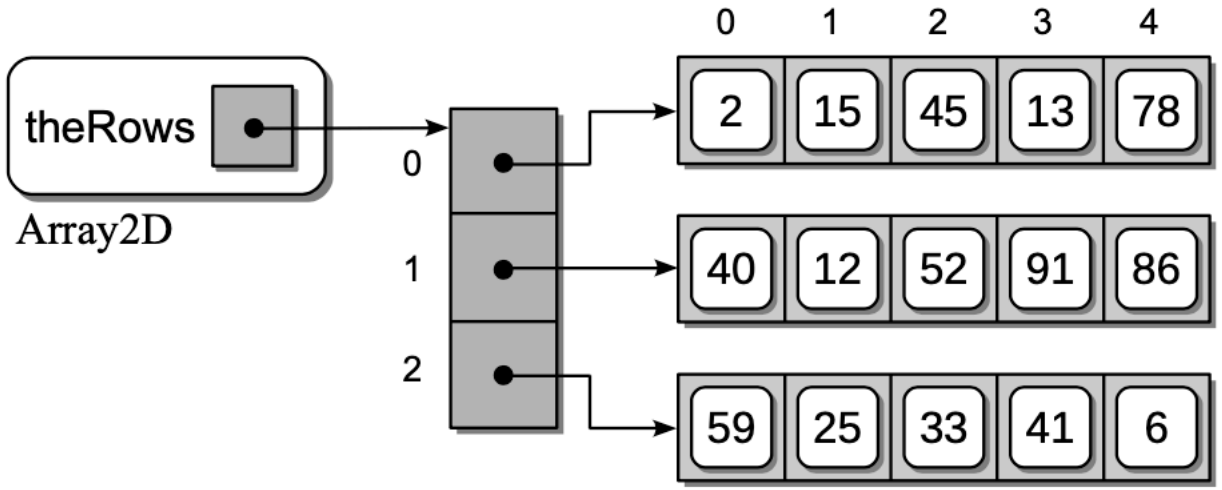
Example of use of the Array2D ADT

```
from array import Array2D
# Open the text file for reading.
gradeFile = open( filename, "r" )
# Extract the first two values which indicate the size of the array.
numExams = int( gradeFile.readline() )
numStudents = int( gradeFile.readline() )
# Create the 2-D array to store the grades.
examGrades = Array2D( numStudents, numExams )
# Extract the grades from the remaining lines.
i=0
for student in gradeFile :
    grades = student.split()
    for j in range( numExams ):
        examGrades[i,j] = int( grades[j] )
        i += 1
# Close the text file.
gradeFile.close()
```

Implementing the Array2D ADT

- The matrix on the left is implemented as an “array of arrays”





The Array2D class definition: constructor

```
# Creates a 2-D array of size numRows x numCols.
def __init__( self, numRows, numCols ):
    # Create a 1-D array to store an array reference for each row.
    self._theRows = Array( numRows )
    # Create the 1-D arrays for each row of the 2-D array.
    for i in range( numRows ) :
        self._theRows[i] = Array( numCols )
```

The Array2D class definition: getter

```
# Gets the contents of the element at position [i, j]
def __getitem__( self, ndxTuple ):
    assert len(ndxTuple) == 2, "Invalid number of array subscripts."
    row = ndxTuple[0]
    col = ndxTuple[1]
    assert row >= 0 and row < self.numRows() \
        and col >= 0 and col < self.numCols(), \
            "Array subscript out of range."
    theldArray = self._theRows[row]
    return theldArray[col]
```

The Array2D class definition: setter

```
# Sets the contents of the element at position [i,j] to value.
def __setitem__(self, ndxTuple, value):
    assert len(ndxTuple) == 2, "Invalid number of array subscripts."
    row = ndxTuple[0]
    col = ndxTuple[1]
    assert row >= 0 and row < self.numRows() \
        and col >= 0 and col < self.numCols(), \
            "Array subscript out of range."
    the1dArray = self._theRows[row]
    the1dArray[col] = value
```

The Array2D class definition: utilities

```
# Returns the number of rows in the 2-D array.
def numRows( self ):
    return len( self._theRows )

# Returns the number of columns in the 2-D array.
def numCols( self ):
    return len( self._theRows[0] )

# Clears the array by setting every element to the given value.
def clear( self, value ):
    for row in self.numRows():
        row.clear( value )
```

Magic methods in Python

- __setitem__ and __getitem__ are “magic” methods (or also “dunder” methods)
- They are not supposed to be invoked directly
- The invocation happens automatically on a certain action

- Example:
  - __add__ is a magic method of the int class (try `dir(int)`)

```
>>> num=10
>>> num + 5
15
>>> num.__add__(5)
15
```

Hands-on activity (20 min)

- Take a look at the code of the 2D Array class presented in the textbook and implement it in two Python files
  - main.py - containing the main body that will test the implemented functions
  - Add the Array2D to the file array.py which you have already created for 1D arrays
- Is important that you read the code that has been provided and you understand its content.
- BEWARE: there may be a few mistakes in the code of the book, fix them!

The Matrix abstract data type

The Matrix ADT specification

- A matrix is a bidimensional array, but on which we define a set of operations

  - `Matrix( rows, ncols )`: Creates a new matrix containing nrows and ncols with each element initialized to 0.
  - `numRows()`: Returns the number of rows in the matrix.
  - `numCols()`: Returns the number of columns in the matrix.
  - `getitem ( row, col )`: Returns the value stored in the given matrix element. Both row and col must be within the valid range.
  - `setitem ( row, col, scalar )`: Sets the matrix element at the given row and col to scalar. The element indices must be within the valid range.
  - `scaleBy( scalar )`: Multiplies each element of the matrix by the given scalar value. The matrix is modified by this operation.
  - `transpose()`: Returns a new matrix that is the transpose of this matrix.
  - `add ( rhsMatrix )`: Creates and returns a new matrix that is the result of adding this matrix to the given rhsMatrix. The size of the two matrices must be the same.
  - `subtract ( rhsMatrix )`: The same as the add() operation but subtracts the two matrices.
  - `multiply ( rhsMatrix )`: Creates and returns a new matrix that is the result of multiplying this matrix to the given rhsMatrix. The two matrices must be of appropriate sizes as defined for matrix multiplication.

Matrix operations: summation and subtraction

- The sum of two matrices is obtained by summing the elements one by one

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix} + \begin{bmatrix} 6 & 7 \\ 8 & 9 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0+6 & 1+7 \\ 2+8 & 3+9 \\ 4+1 & 5+0 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \\ 5 & 5 \end{bmatrix}$$

- The elements of matrix C = A + B are given by:
  - $\left[c_{ij}\right] = \left[a_{ij} + b_{ij}\right]$

Matrix operations: scaling

- The scaling operation means to multiply a matrix by a constant coefficient

$$3\begin{bmatrix} 6 & 7 \\ 8 & 9 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 3*6 & 3*7 \\ 3*8 & 3*9 \\ 3*1 & 3*0 \end{bmatrix} = \begin{bmatrix} 18 & 21 \\ 24 & 27 \\ 3 & 0 \end{bmatrix}$$

- If the matrix is a vector this means to increase its length

- In general, each element of matrix $C = \lambda \cdot A$ is
  - $\left[c_{ij}\right] = \left[\lambda a_{ij}\right]$

Matrix operations: multiplication

- Matrix multiplication implies that the number of columns of the multiplicand is equal to the number of rows of the multiplier

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix} * \begin{bmatrix} 6 & 7 & 8 \\ 9 & 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} (0*6+1*9) & (0*7+1*1) & (0*8+1*0) \\ (2*6+3*9) & (2*7+3*1) & (2*8+3*0) \\ (4*6+5*9) & (4*7+5*1) & (4*8+5*0) \end{bmatrix}$$

$$= \begin{bmatrix} 9 & 1 & 0 \\ 39 & 17 & 16 \\ 69 & 33 & 32 \end{bmatrix}$$

- The elements of matrix $C_{(mxn)} = A_{(mxq)} \cdot B_{(qxn)}$ are given by:

$$\left[c_{ij}\right] = \left[\sum_{k=1..q} a_{ik} \cdot b_{kj}\right]$$

Matrix operations: transposition

- To transpose a matrix means to exchange the rows with the columns.

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}^T = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

- This can be expressed as $C = A^T$ and therefore $\left[c_{ij}\right] = \left[a_{ji}\right]$

- Other important operations are the calculation of the determinant, the inverse, the calculation of the eigenvalues and the eigenvectors, the singular value decomposition and many others

Matrix ADT implementation: the constructor and the utilities

```
class Matrix :
    # Creates a matrix of size numRows x numCols initialized to 0.
    def __init__( self, numRows, numCols ):
        self._theGrid = Array2D( numRows, numCols )
        self._theGrid.clear( 0 )

    # Returns the number of rows in the matrix.
    def numRows( self ):
        return self._theGrid.numRows()

     # Returns the number of columns in the matrix.
    def numCols( self ):
        return self._theGrid.numCols()
```

Matrix ADT implementation: setters and getters

```
# Returns the value of element (i, j): x[i,j]
def __getitem__( self, ndxTuple ):
    return self._theGrid[ ndxTuple[0], ndxTuple[1]]

# Sets the value of element (i,j) to the value s: x[i,j] = s
def __setitem__( self, ndxTuple, scalar ):
        self._theGrid[ ndxTuple[0], ndxTuple[1] ] = scalar
```

Matrix ADT implementation: scaling

```
# Scales the matrix by the given scalar.
def scaleBy( self, scalar ):
    for r in range( self.numRows() ) :
        for c in range( self.numCols() ) :
            self[ r, c ] *= scalar
```

Matrix ADT implementation: addition

```
def __add__( self, rhsMatrix ):
    assert rhsMatrix.numRows()  == self.numRows() and \
           rhsMatrix.numCols() == self.numCols(), \
           "Matrix sizes not compatible for the add operation."
    # Create the new matrix.
    newMatrix = Matrix( self.numRows(), self.numCols() )
    # Add the corresponding elements in the two matrices.
    for r in range( self.numRows() ) :
        for c in range( self.numCols() ) :
            newMatrix[ r, c ] = self[ r, c ] + rhsMatrix[ r, c ]
    return newMatrix
```

```
A=Matrix(2,3)
B=Matrix(2,3)
C=Matrix(2,3)
```

C

	0	1	2
0	1	3	5
1	5	2	1

R

0	1	2
3	1	2
4	5	6

```
for r in range( self.numRows() ) :
        for c in range( self.numCols() ) :
            newMatrix[ r, c ] = self[ r, c ] + rhsMatrix[ r, c ]



C=A.__add__(B)
C=A+B
```

Matrix ADT: examples of use

```
from matrix import Matrix
```

```
A=Matrix(2,2)
for i in range(2):
    for j in
range(2):
        A[i,j]=i*j
```

```
B=Matrix(2,2)
for i in range(2):
    for j in
range(2):
        A[i,j]=i-j
```

```
C=Matrix(2,2)
```

```
C= A+B
C=A.transpose()
A = A*B
```

Hands-on activity (20 min)

- Take a look at the code of the Matrix class presented in the textbook and implement it in two Python files
  - testmatrix.py - containing the main body that will use the implemented functions
  - matrix.py - containing the Matrix class
- Is important that you read the code that has been provided and you understand its content.
- BEWARE: there may be a few mistakes in the code of the book, fix them!