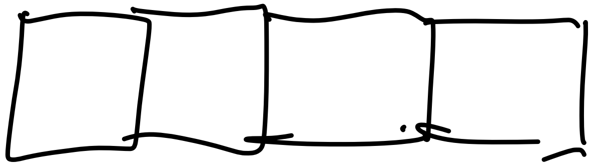
CPE112RC — Module1 — Linear Data Structures



# Lecture2: Array vs. Python List (Array/Array2D/Matrix)

Dr. Prapong Prechaprapranwong

Computer Engineering

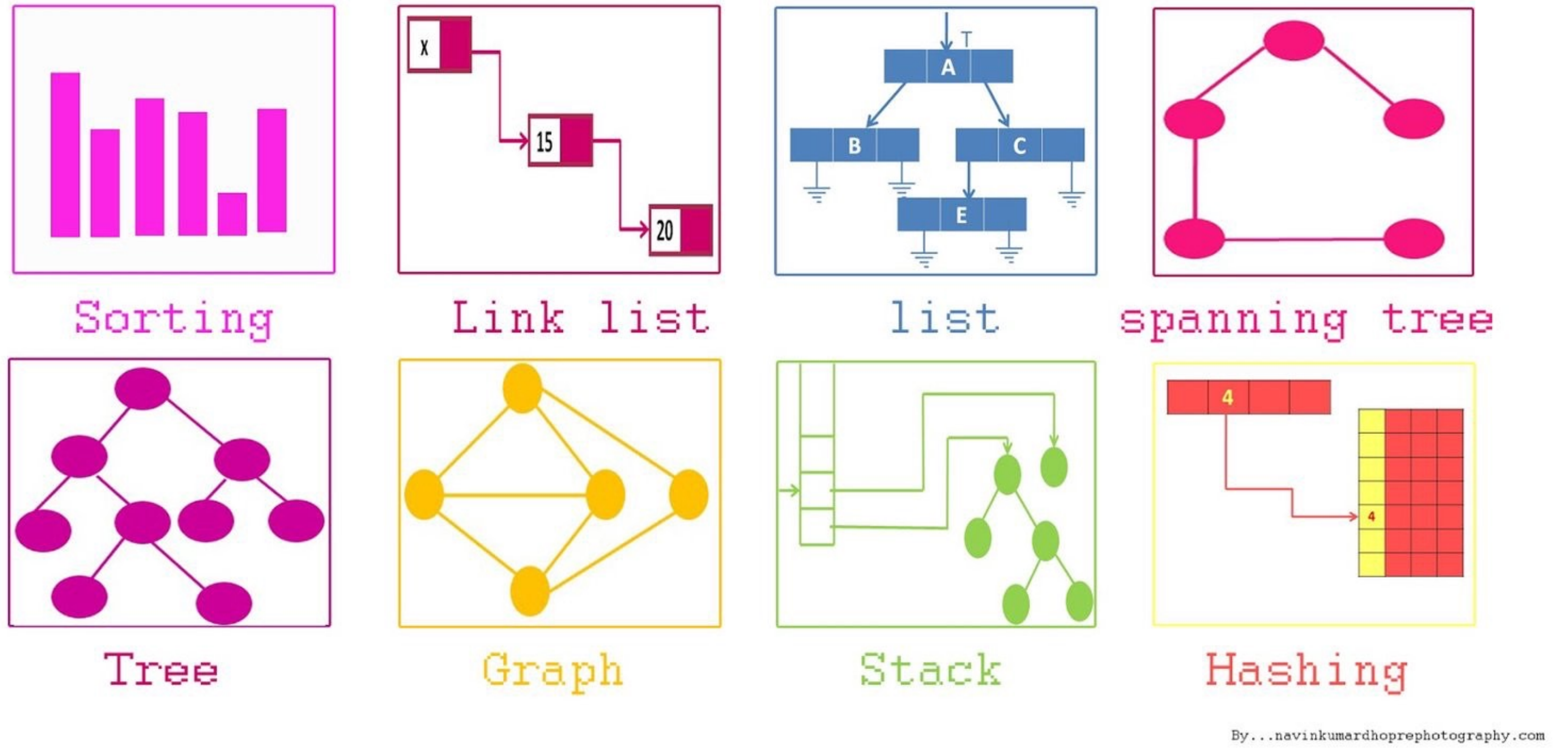
King Mongkut's University of Technology Thonburi

นามธรรม ประเภท

# Abstract Data Type (ADT) =

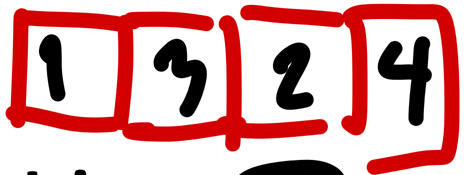
A definition for a data type

- a set of values
- a set of Operations allowed on data type



## Content

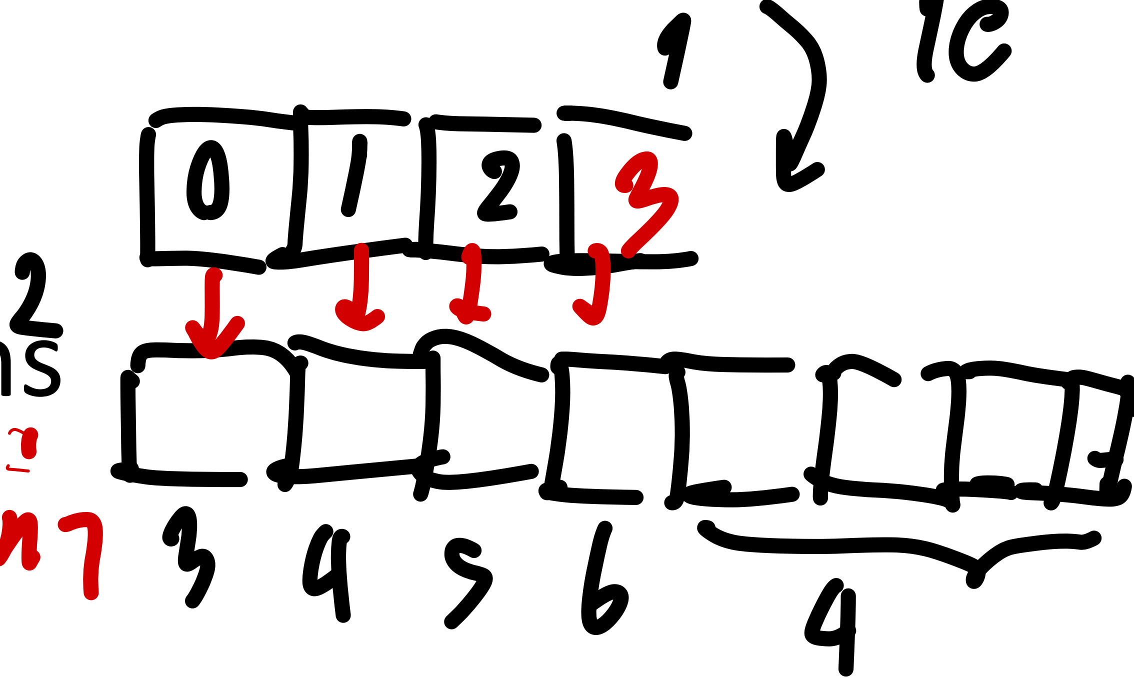
$\rightarrow$ C int a[4];

a :  t:4

- Array ADT
  - Array v.s. Python List
  - Behind dynamic array
- 2D-Array ADT
- Matrix ADT
  - Matrix Operations

ช้า

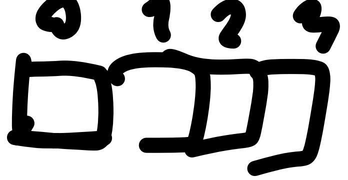
List



Python List 4~5 เท่า

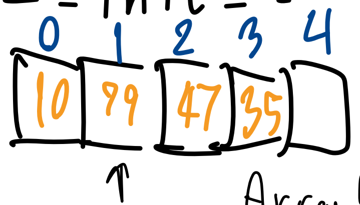
## Array ADT

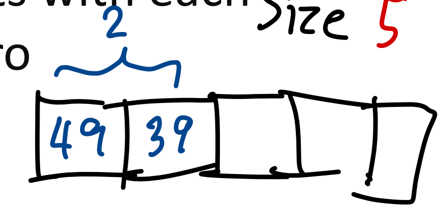
→ การสร้าง Array



A one-dimensional array is 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.

→ สร้าง Method





Length

Size 5

| | | |
|---|---|---|
| 1. | **Array( size ):** | __init__ 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 ):** | Array[0] Returns the value stored in the array at element position index. The Index argument must be within the valid range. Accessed using the subscript Array[0] operator : 20 |
| | **__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. |
| | → ทำให้เป็นทุกช่อง | |
| | **clear( value ):** | Clears the array by setting every element to value |
| * | **iterator ():** | # for i in array: Creates and returns an iterator that can be used to traverse the elements of the array |

for i in [1,3,2,4]

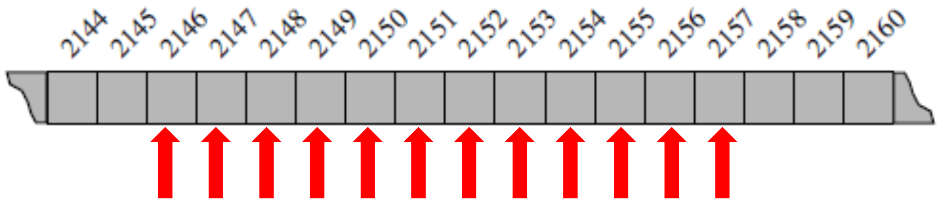
Iterable

↳ เมื่อใช้ Loop for จะได้ ทำงานได้

RAM คือ random access

ก็คือมันจะจิ้มช่องไหนก็ได้ แต่จะไหลต่อ 12 ช่อง

Data stored in Memory address



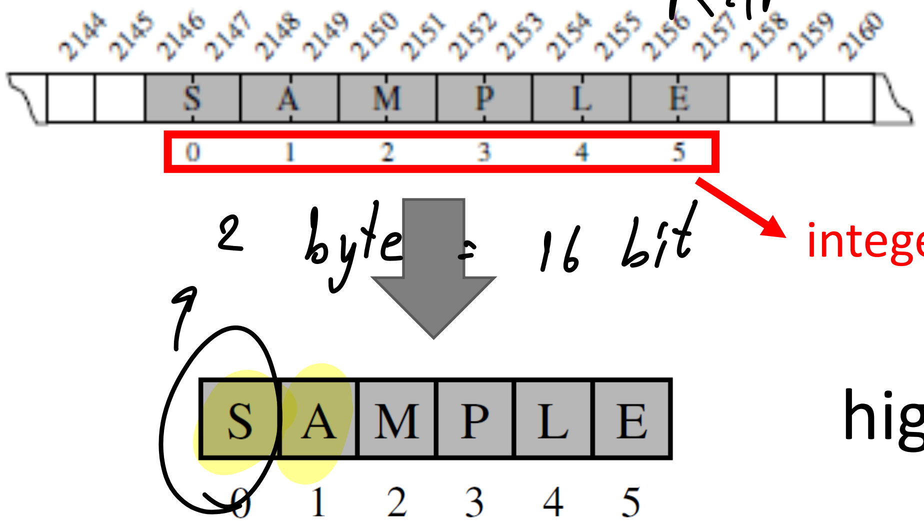
1char: 2 bytes

**a n a r r a y o f s i x c h a r a c t e r s**, requires 12 bytes of memory. We will refer to each location within an array as a cell, and will use an **integer index** to describe its location within the array

8 bit 8bit = 1 byte

16 bit

RAM

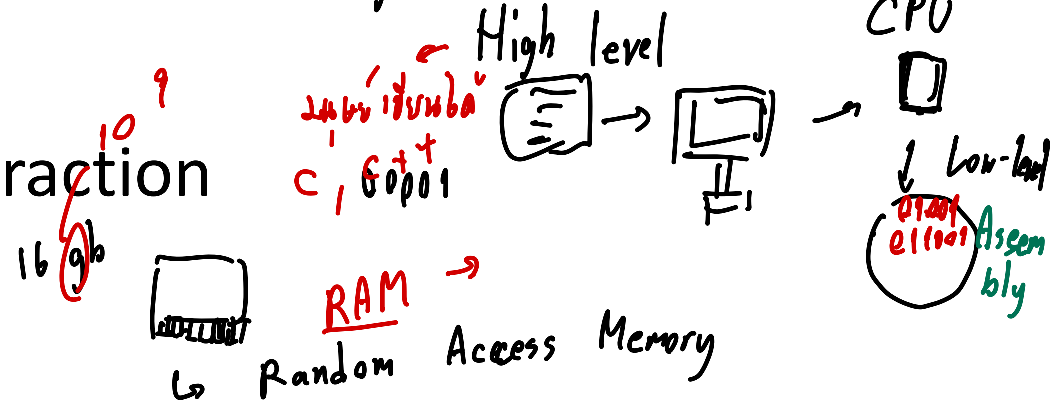


integer index

sequential data type

higher-level abstraction

Signal 0 1

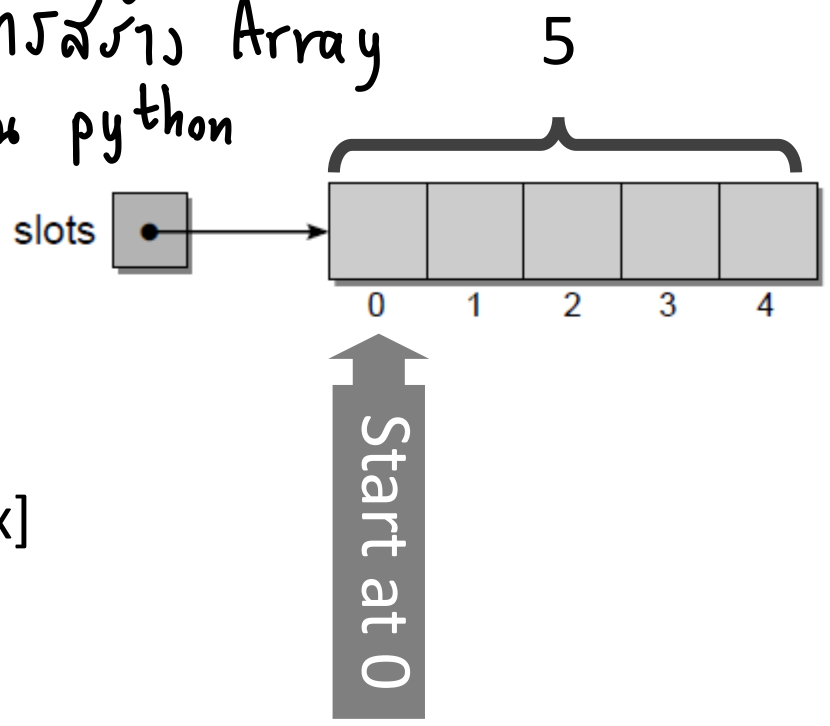


array(size) - Create array with size elements $\rightarrow$ การสร้าง Array ใน python

$\rightarrow$ ชนิดของ C

```
import ctypes

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



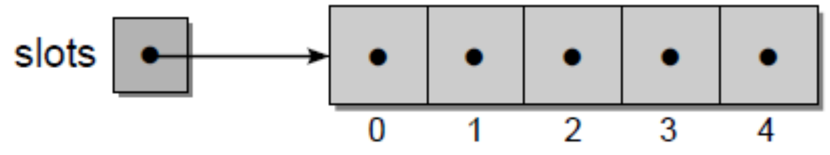
__getitem__(index) - recall the value at slots[index]

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

$\longrightarrow$ Error

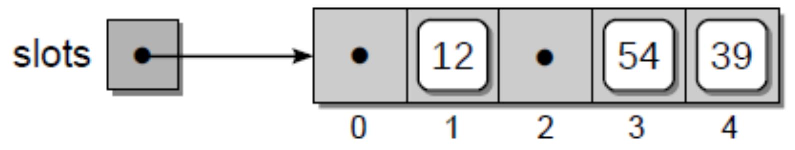
clear(None) - assign all value to **N o n e**

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



__setitem__(index,value) - assign other values

```
slots[1] = 12
slots[3] = 54
slots[4] = 39
```



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.

```
# 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 )

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

check status, “Error message”

Constructor

The __len__method, which returns the number of elements in the array, simply returns the value of size that was saved in the constructor.

the __getitem__ operator method takes the array index as an argument and returns the value of the corresponding element.

```
# 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 ]

# 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

# 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
```

check status, “Error message”

the __setitem__operator method is used to set or change the contents of a specific element of the array

## Python List →Dynamic Array

```
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
```

## Array vs Python List

- an array has a limited number of operations → มี method น้อย
- The array is suited when the maximum no. of elements are known
- The list is the better choice when the size may change after it created
  ↳ มันเปลี่ยนได้
- Python list will use four to five time as much memory

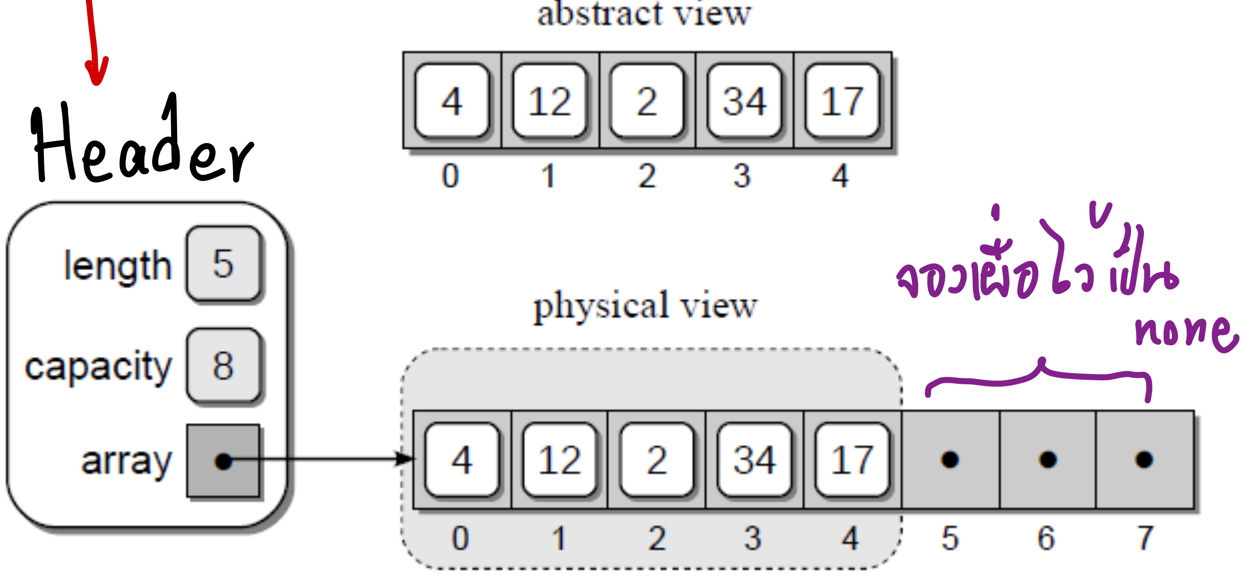
↳ ใช้ Memory มากกว่า 4-5 เท่า

## Behind the scene - Python List

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

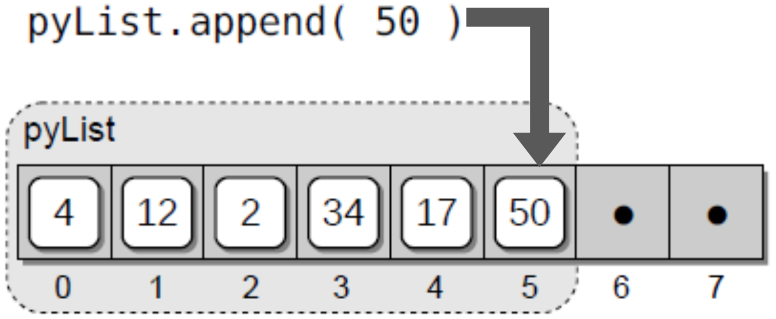
ทำให้เปลี่ยนค่า

Header

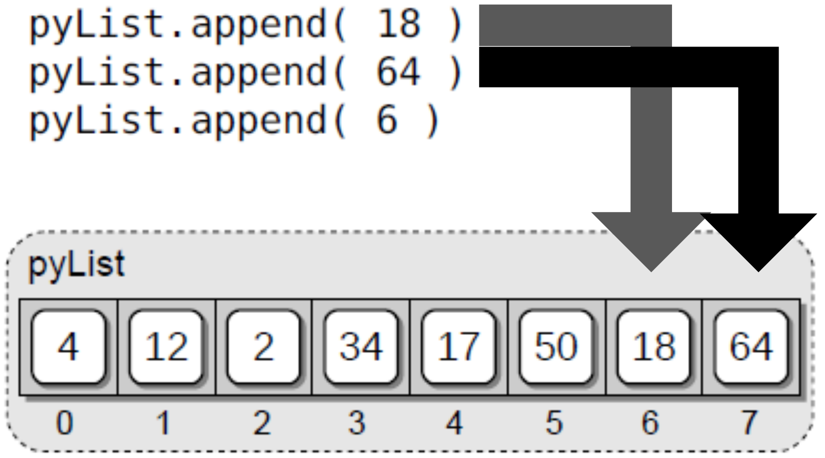


## Append the list

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



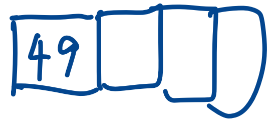
```
pyList.append( 18 )
pyList.append( 64 )
pyList.append( 6 )
```



จอง memory เผื่อไว้ ให้เติมค่าได้

Myarray.clear( )

4

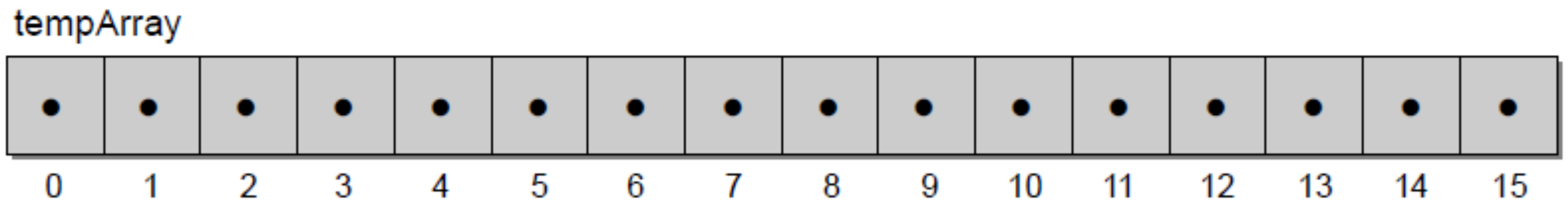


Myarray[0] =

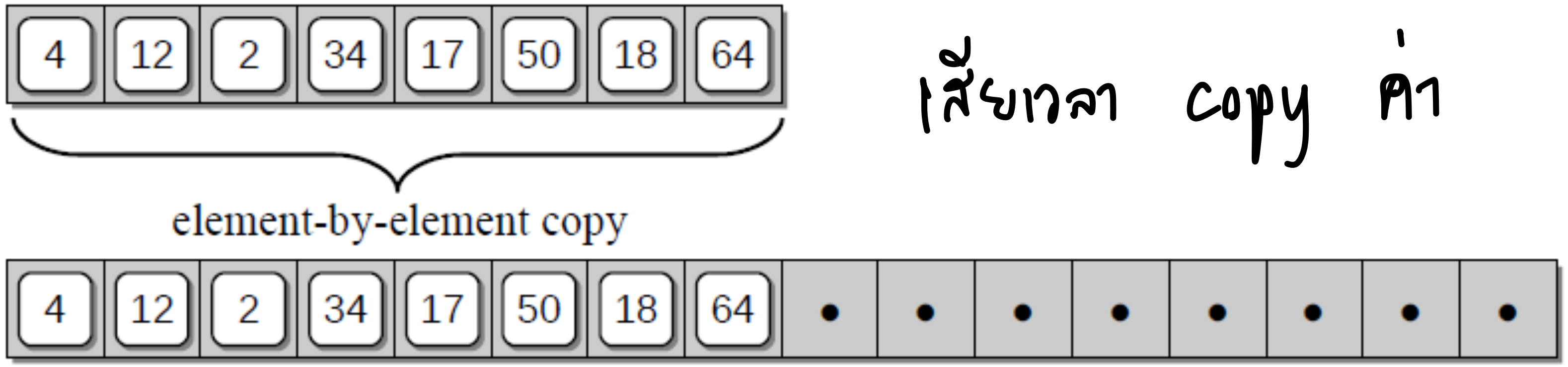
↑ MyArray[3]

ถ้าเรา append เกินกว่า capacity

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



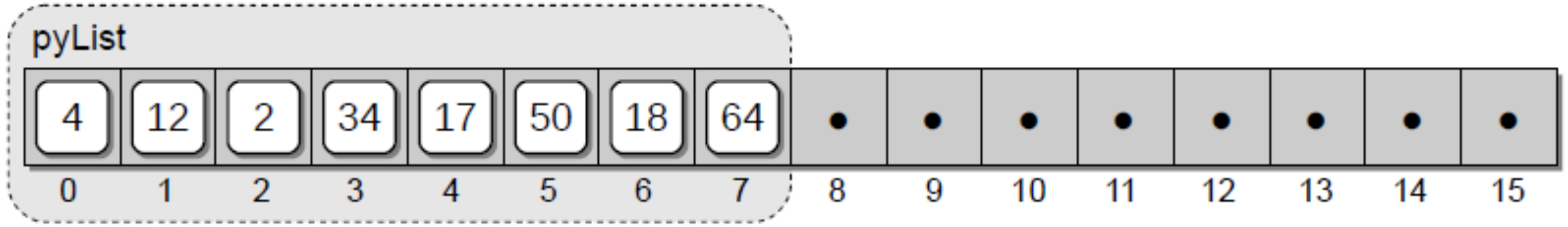
**(2)** The values from the original array are copied to the new larger array.



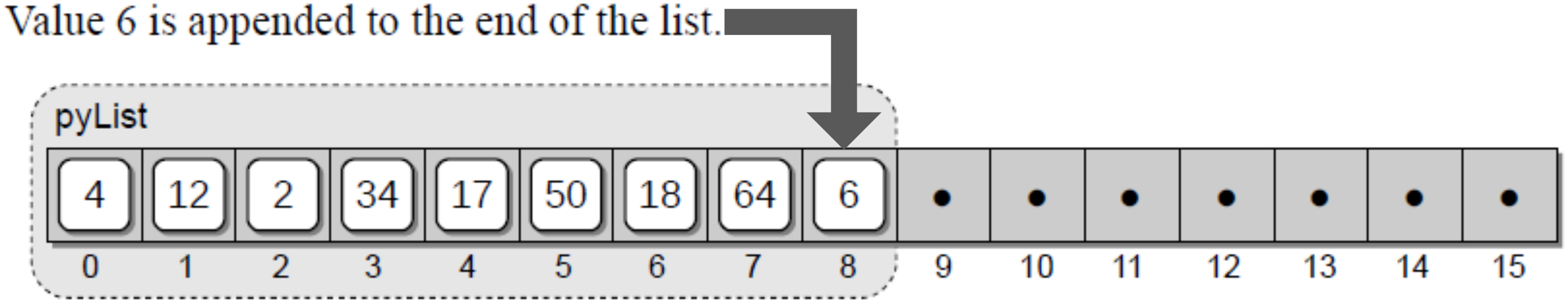
เสียเวลา copy ค่า

เอา Pointer มาวาง

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



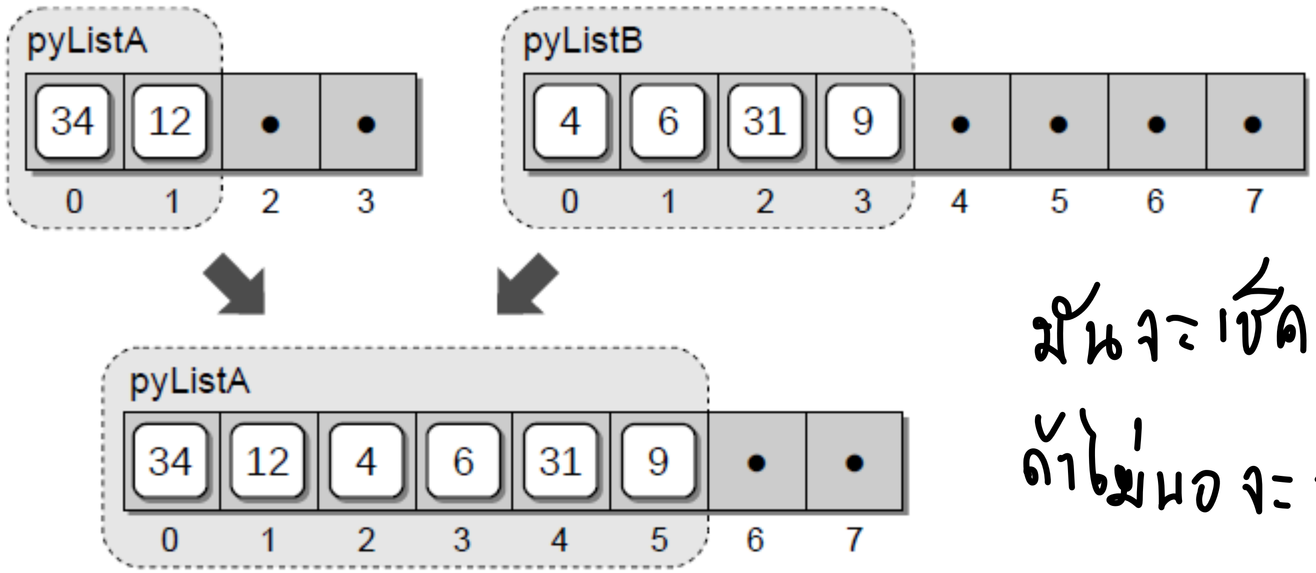
**(4)** Value 6 is appended to the end of the list.



## Extend the list

→ List 2 List มาต่อกัน

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



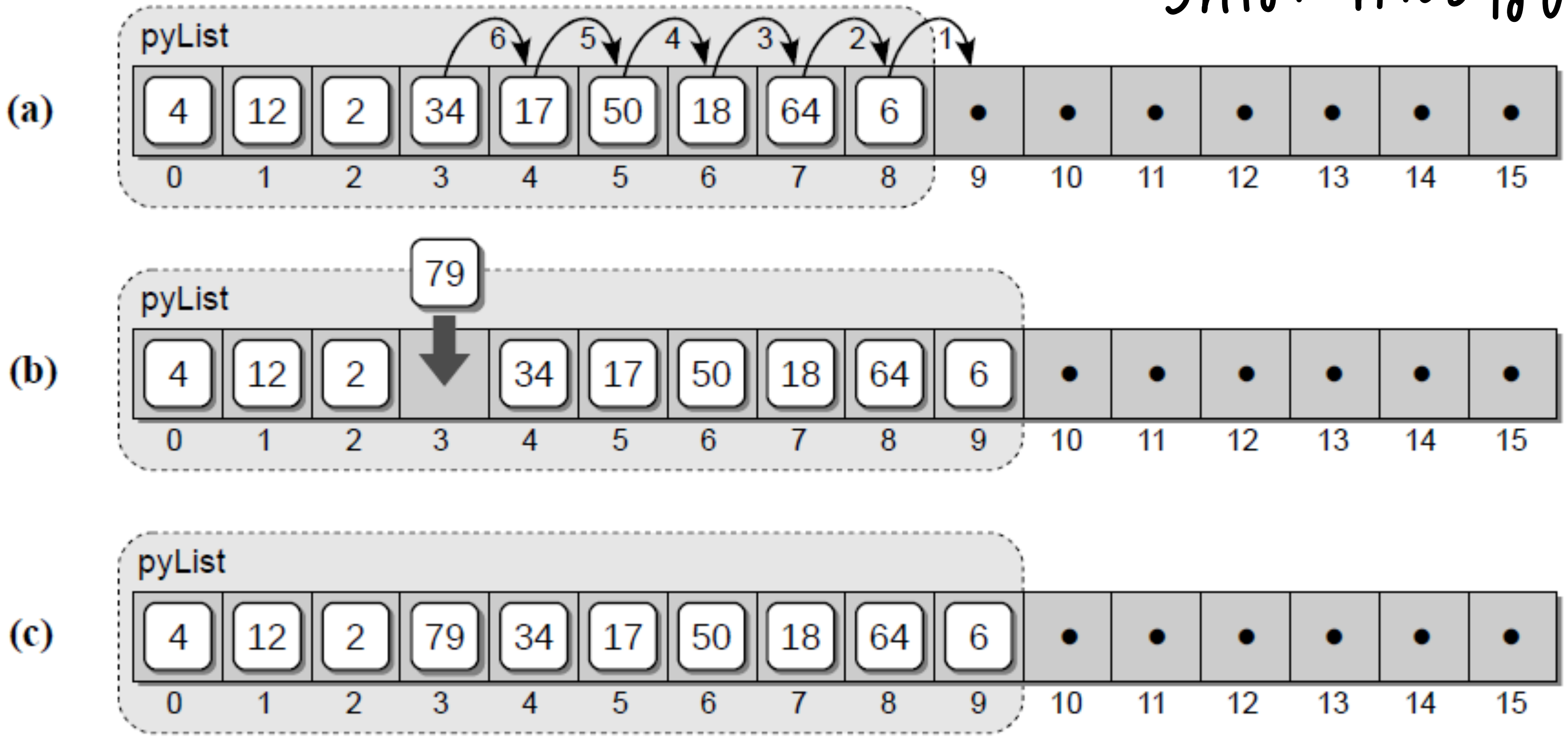
มันจะเช็คว่า pyList A จุพอไหม
ถ้าไม่พอจะจุ 2 เท่า หรือ 4 เท่า จนกว่าจะ
ยัด B มาได้

## Insert an element to the list

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

at position:3 with value:79

shift ค่าไปเรื่อยๆ ก่อให้เกิดค่า



ในวิชา Discrete Math เราจะใช้ Complexity

↳ แบบ Linear Priority O(n) Time Complexity

## pop an element from the list

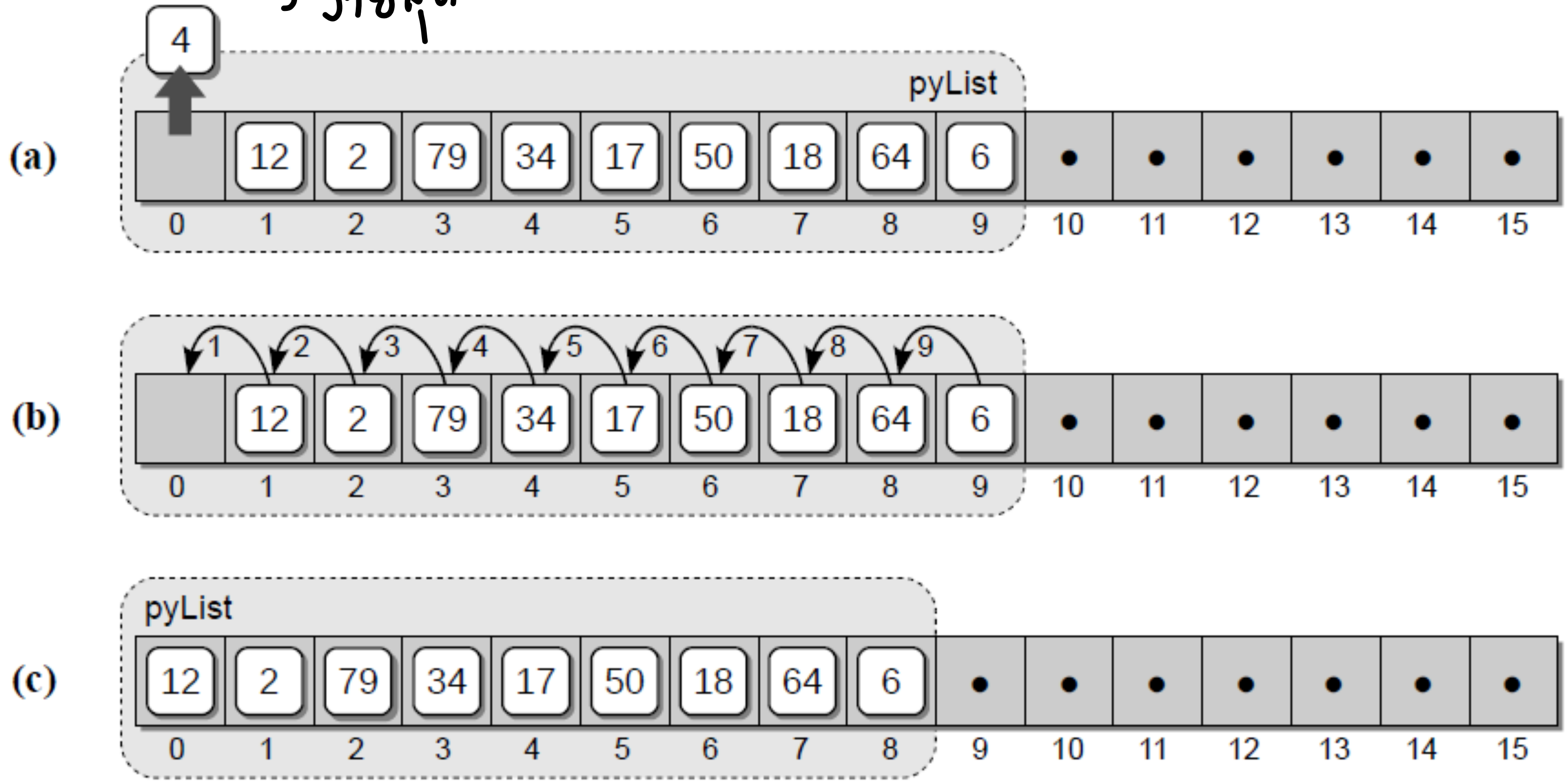
Implement Data ที่เรียกว่า stack , queue

↱ Linear Priority

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

→ เอาค่าสุดท้ายออก

↳ ง่ายสุด



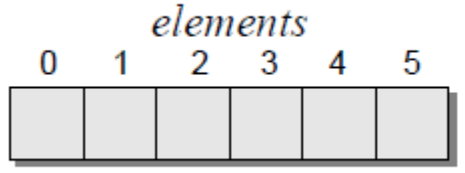
## Array2D ADT

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.

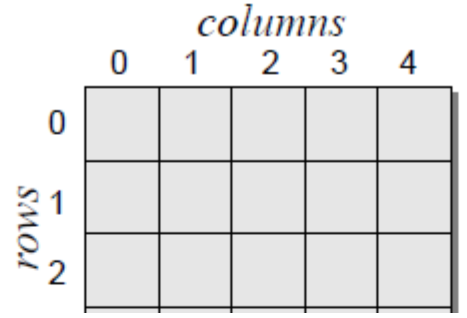
| | |
|---|---|
| **Array2D( nrows, ncols ):** | Creates a two-dimensional array organized into rows and columns. The **nrows** and **ncols** indicate the size of the table. The individual elements of the table are initialized to None |
| **numRows():** | Returns the number of rows in the 2-D array |
| **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 |

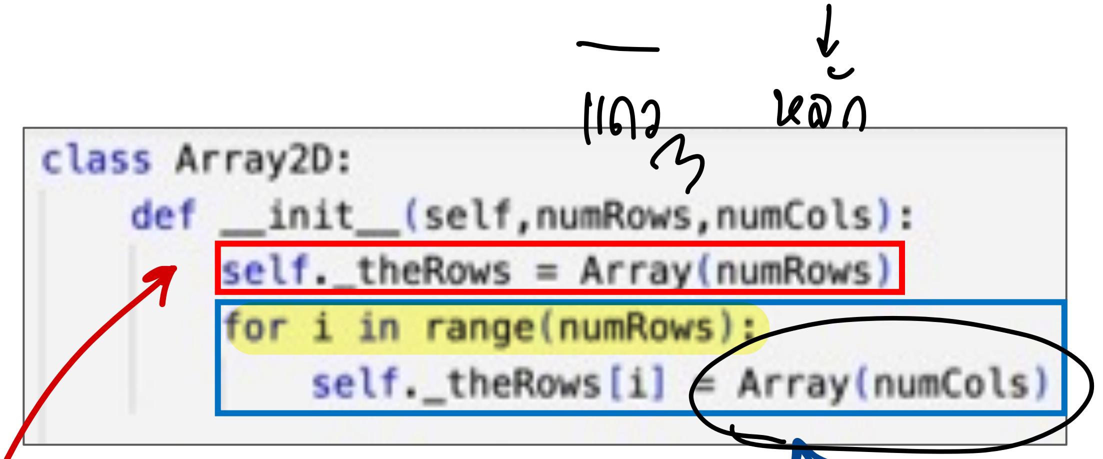
## The 2D-Array

A = Array(6)



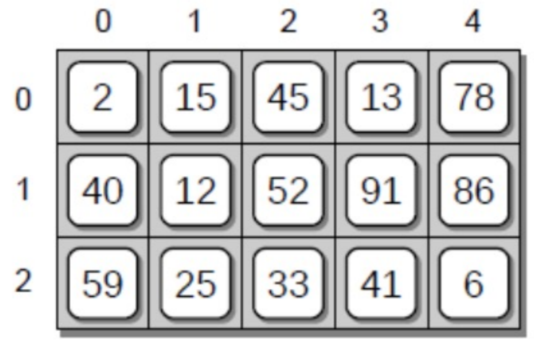
A2D = Array2D(3,5)



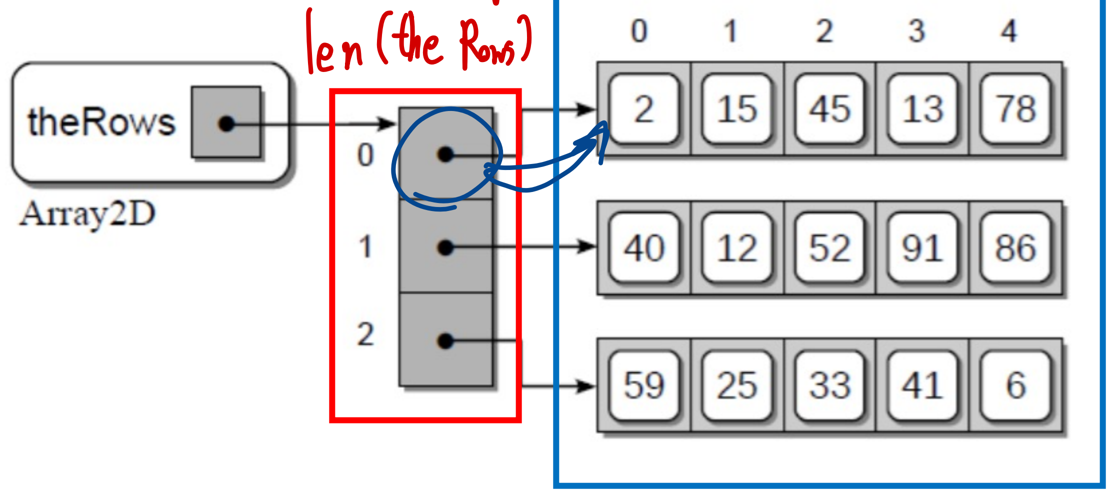


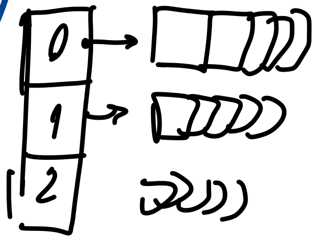
```
class Array2D:
    def __init__(self,numRows,numCols):
        self._theRows = Array(numRows)
        for i in range(numRows):
            self._theRows[i] = Array(numCols)
```

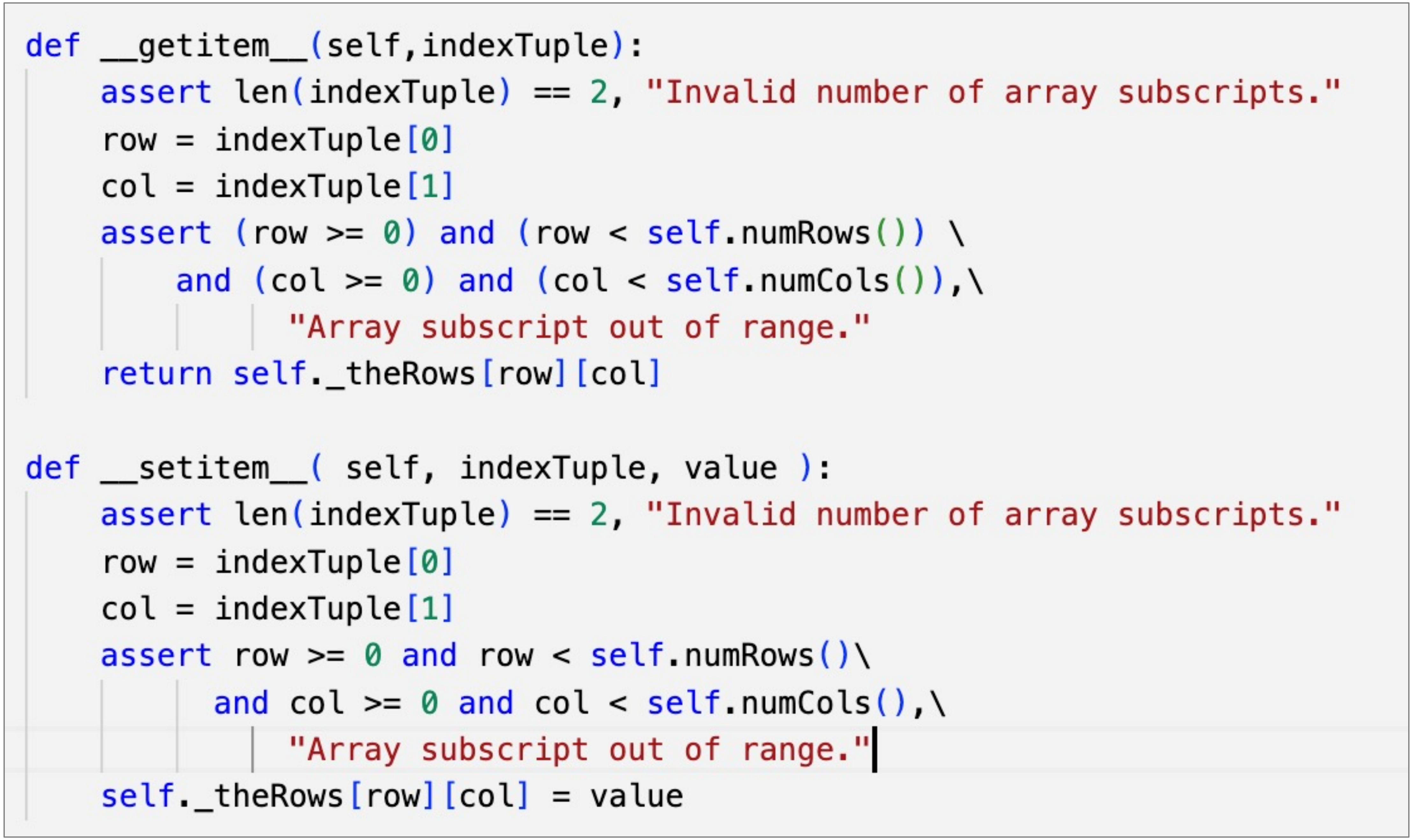
2-dimensional array creates from a number of 1-dimensional arrays (theRows)



|   | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 2 | 15 | 45 | 13 | 78 |
| 1 | 40 | 12 | 52 | 91 | 86 |
| 2 | 59 | 25 | 33 | 41 | 6 |







```python
def __getitem__(self,indexTuple):
    assert len(indexTuple) == 2, "Invalid number of array subscripts."
    row = indexTuple[0]
    col = indexTuple[1]
    assert (row >= 0) and (row < self.numRows()) \
        and (col >= 0) and (col < self.numCols()),\
            "Array subscript out of range."
    return self._theRows[row][col]

def __setitem__( self, indexTuple, value ):
    assert len(indexTuple) == 2, "Invalid number of array subscripts."
    row = indexTuple[0]
    col = indexTuple[1]
    assert row >= 0 and row < self.numRows()\
           and col >= 0 and col < self.numCols(),\
            "Array subscript out of range."
    self._theRows[row][col] = value
```

## Matrix ADT

A matrix is a collection of scalar values arranged in rows and columns as a rectangular grid of a fixed size. The elements of the matrix can be accessed by specifying a given row and column index with indices starting at 0.

| | |
|---|---|
| **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 |

## Additional operations

| | |
|---|---|
| **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 |

## Scaling in Matrix

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

```
 # 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 Addition

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

2×3 2×3

matrix + matrix2

```
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.          แถว          หลัก
  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[ 0, 0 ] = self[ 0, q ] + rhsMatrix[ 0, q ]
  return newMatrix
```

[ 1, 0 ] : ซ้าย 1, 0   ขวา 1, 0

## Matrix Subtraction

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

## Matrix transposition

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

$[i, j] : [j, i]$

for 3 in

for 2 in

[2][3]

ไล่เป็นก่อน

## Matrix Multiplication

[3][2] 3x2 2x3

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

∴ for 3 in
for 2

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

∴ for 3 in
for 2 in

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

total

$\ell$

$$A = \begin{bmatrix} A_{0,0} & A_{0,1} \\ A_{1,0} & A_{1,1} \\ A_{2,0} & A_{2,1} \end{bmatrix} \qquad B = \begin{bmatrix} B_{0,0} & B_{0,1} & B_{0,2} \\ B_{1,0} & B_{1,1} & B_{1,2} \end{bmatrix}$$

$$\begin{aligned}
C_{0,0} &= A_{0,0} * B_{0,0} + A_{0,1} * B_{1,0} \\
C_{0,1} &= A_{0,0} * B_{0,1} + A_{0,1} * B_{1,1} \\
C_{0,2} &= A_{0,0} * B_{0,2} + A_{0,1} * B_{1,2} \\
C_{1,0} &= A_{1,0} * B_{0,0} + A_{1,1} * B_{1,0} \\
C_{1,1} &= A_{1,0} * B_{0,1} + A_{1,1} * B_{1,1} \\
C_{1,2} &= A_{1,0} * B_{0,2} + A_{1,1} * B_{1,2} \\
C_{2,0} &= A_{2,0} * B_{0,0} + A_{2,1} * B_{1,0} \\
C_{2,1} &= A_{2,0} * B_{0,1} + A_{2,1} * B_{1,1} \\
C_{2,2} &= A_{2,0} * B_{0,2} + A_{2,1} * B_{1,2}
\end{aligned}$$

## What should you know

- Differentiate between Array and Dynamic array
- Choose appropriately to use Array or Python’s List
- How to implement Array class
- How to implement 2D-Array class
- How to implement Matrix class