More 2D Array and Loop Examples
Functions and Parameters

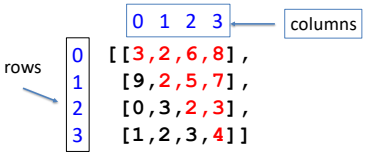
A few matrix and array problems

Given a matrix (2D array with equal dimension), how to compute the sum for the top-right half?

```
[[3,2,6,8],
 [9,2,5,7],
 [0,3,2,3],
 [1,2,3,4]]
```

The result should be 42

The key is to figure out the indices



When row is 0, column goes from 0 to 3
When row is 1, column goes from 1 to 3
When row is 2, column goes from 2 to 3
When row is 3, column goes from 3 to 3

```
for row in range( 4 ):
    for col in range( row, 4 ):
        # do work
```



```
def sum_upper_right( matrix ):
    ''' Sum up the upper-right corner of a matrix. Matrix is
    a 2D array with equal dimensions '''

    sum = 0
    for row in range( len( matrix ) ):          # row
        for col in range( row, len( matrix[0] ) ):  # column
            sum += matrix[row][col]
    return sum

matrix = [[3,2,6,8],
          [9,2,5,7],
          [0,3,2,3],
          [1,2,3,4]]
value = sum_upper_right( matrix )
print( 'the sum of right upper corner is ', value )
```

matrix_tophalf.py

Given a matrix (2D array with equal dimension), how to compute the maximum for each row and each column?

```
# compute row max for a given 'row'
row_max = matrix[row][0]
for i in range( len( matrix[row] ) ):
    if matrix[row][i] > row_max:
        row_max = matrix[row][i]
```

But how to go through a column to compute the maximum?

```
# compute column max for a given 'column'
col_max = matrix[0][col]
for i in range( len( matrix ) ):
    if matrix[i][col] > col_max:
        col_max = matrix[i][col]
```

In addition to the row and column maximum, find the maximum of the entire matrix?

```
def find_max( matrix, row_max, col_max ):
    ''' Given a matrix, find and return the global max, an
        array of row max and an array of column max '''

    max = matrix[0][0]                  # current max
    for i in range( len( matrix ) ):    # find each row max
        row_max[i] = find_row_max( matrix, i )
        if row_max[i] > max:
            max = row_max[i]

    for i in range( len( matrix[0] ) ):  # find each column max
        col_max[i] = find_col_max( matrix, i )
        if col_max[i] > max:
            max = col_max[i]

    return max
```

array_max.py

Functions and Parameters

- We've learned how to develop functions

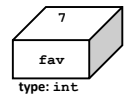
```
def find_max( a_list ):
    max = a_list[0]
    for i in range( len(a_list) ):
        if a_list[i] > max:
            max = a_list[i]
    return max
```

```
def sum_list( aList ):
    sum = 0
    for i in range( len(a_list) ):
        sum += a_list[i]
    return sum
```

- In both cases, 'a_list' is called a **parameter** for the function
- A function can have multiple parameters
- Two types of parameters, **mutable** and **immutable**
- Let's try out the examples (`param_passing.py`)

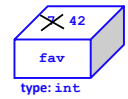
Pass By Value: parameters immutable

```
def main()
    """ calls conform """
    print(" Welcome to Conformity, Inc. ")

    fav = 7
    conform(fav)

    print(" My favorite number is", fav )
```



PASS BY VALUE

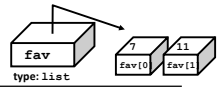
```
def conform(fav)
    """ sets input to 42 """
    fav = 42
    return fav
```



"Pass by value" means that the data's value is **copied** when sent to a function…

Passing by reference: parameters are mutable

```
def main()
    """ calls conform2 """
    print " Welcome to Conformity, Inc. "
    fav = [ 7, 11 ]
    conform2(fav)
    print(" My favorite numbers are", fav )
```

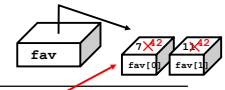



```
def conform2(fav)
    """ sets all of fav to 42 """
    fav[0] = 42
    fav[1] = 42
```

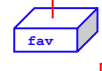
What gets passed by value here?

Passing ***list*** content by reference…

```
def main()
    """ calls conform2 """
    print " Welcome to Conformity, Inc. "
    fav = [ 7, 11 ]
    conform2(fav)
    print(" My favorite numbers are", fav )
```




```
def conform2(fav)
    """ sets all of fav to 42 """
    fav[0] = 42
    fav[1] = 42
```



The **reference** is copied, i.e., passed by value, but the contents arn't !

and it can change data elsewhere!

Watch out!

You can change **the contents of lists** in functions that take those lists as input.

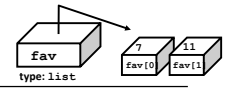
(actually, lists or any mutable objects)

Those changes will be visible **everywhere**.

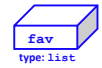
(immutable objects are safe, however)

But ***lists*** are passing by value!!!

```
def main()
    """ calls conform3 """
    print " Welcome to Conformity, Inc. "
    fav = [ 7, 11 ]
    conform3(fav)
    print(" My favorite numbers are", fav )
```

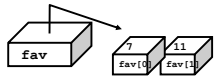



```
def conform3(fav)
    """ creates a new fav!!! """
    fav = [ 42, 42 ]
```

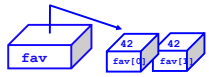


But ***lists*** are passing by value!!!

```
def main()
    """ calls conform3 """
    print " Welcome to Conformity, Inc. "
    fav = [ 7, 11 ]
    conform3(fav)
    print(" My favorite numbers are", fav )
```




```
def conform3(fav)
    """ creates a new fav!!! """
    fav = [ 42, 42 ]
```



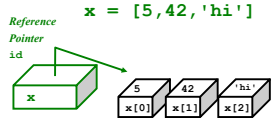
Mutable vs. Immutable

Pass by REFERENCE

`dictionary`

`list`

```
x = [5,42,'hi']
```



Lists and dictionaries are handled by reference (the variables really hold a ***memory address***)

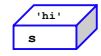
Pass by VALUE

`tuple` `float`

`string` `bool`

`int`

```
s = 'hi'
```

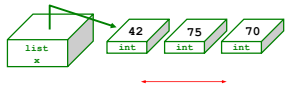


Other types of data, incl. strings, are handled by value: they ***hold*** the actual data

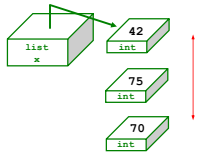
Lists' flexibility

Lists can hold **ANY** type of data

```
x = [ 42, 75, 70 ]
```

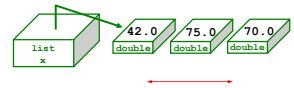


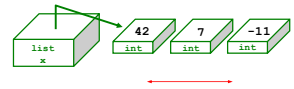
We can equally well imagine them as **vertical** structures.

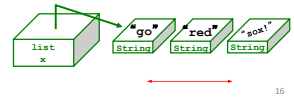


Lists' flexibility

Lists can hold **ANY** type of data







2d lists or ***arrays***

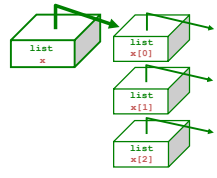
Lists can hold **ANY** type of data -- including lists !

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

2d arrays

Lists can hold **ANY** type of data -- including lists !

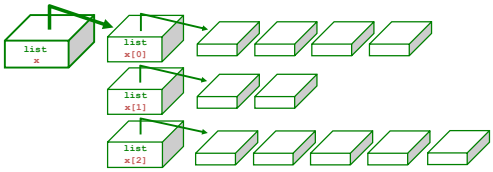
```
x = [ [1,2,3,4], [5,6], [7,8,9,10,11] ]
```



Jagged arrays

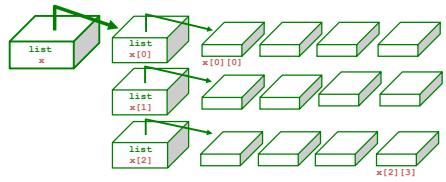
Lists can hold **ANY** type of data -- including lists !

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



Rows within 2d arrays need not be the same length

Rectangular arrays



What does `x[1]` refer to?

What value is changed with `x[1][2]=42` ?

How many rows does `x` have, in general ?

How many columns does `x` have, in general ?