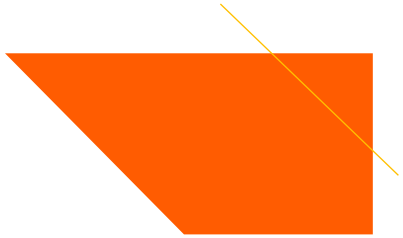
Introduction to Problem Solving in Python

COSI 10A



Class objectives

- Tallying
- 2D Lists (Section 7.4)

Tallying

A multi-counter problem

- Problem: Write a function `most_frequent_digit` that returns the digit value that occurs most frequently in a number.
  - Example: The number 669260267 contains:
    one 0, two 2s, four 6es, one 7, and one 9.
  - `most_frequent_digit(669260267)` returns 6.
  - If there is a tie, return the digit with the lower value.
  - `most_frequent_digit(57135203)` returns 3.

A multi-counter problem

- We could declare 10 counter variables ...

```
counter0, counter1, counter2, counter3, counter4,
counter5, counter6, counter7, counter8, counter9
```

- But a better solution is to use a list of size 10.
  - The element at index *i* will store the counter for digit value *i*.
  - Example for 669260267:

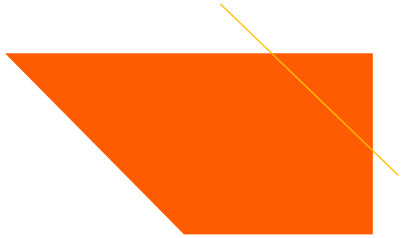
index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
value	1	0	2	0	0	0	4	1	0	0

  - How do we build such a list? And how does it help?

Creating a list of tallies

```
# assume n = 669260267
counts = [0] * 10
while n > 0:
    # pluck off a digit and add to proper counter
    digit = n % 10
    counts[digit] += 1
    n = n // 10
```

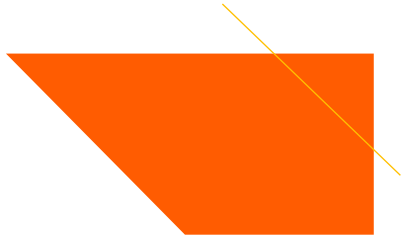
index	0	1	2	3	4	5	6	7	8	9
value	1	0	2	0	0	0	4	1	0	0



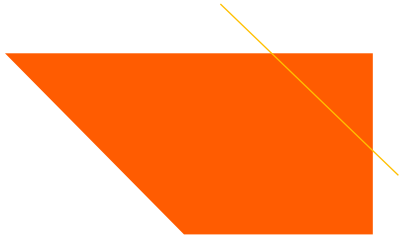
Tally solution

```
# Returns the digit value that occurs most frequently in n.
# Breaks ties by choosing the smaller value.
def most_frequent_digit(n):
    counts = [0] * 10
    while n > 0:
        digit = n % 10    # pluck off a digit and tally it
        counts[digit] += 1
        n = n // 10

    # find the most frequently occurring digit
    best_index = 0
    for i in range(1, len(counts)):
        if counts[i] > counts[best_index]:
            best_index = i
    return best_index
```



Lists of Lists



Multidimensional Lists

- One-dimensional list (a single row or a single column of data)
- A multidimensional list is a list of lists

- The most common multidimensional list is a two-dimensional list
  - The way we indicate/create a list of lists is by using two nested sets of `[]` brackets

Defining a two-dimensional list

Syntax:

```
name = [[value, value, … value],
        [value, value, … value],
        …,
        [value, value, … value]]
```

Syntax:

```
name = [[value] * length,
        [value] * length,
        …,
        [value] * length]
```

Example:

```
tmp = [[0,0,0,0],
       [0,0,0,0],
       [0,0,0,0]]
```

Example:

```
tmp = [[0]* 4,[0] *4,[0]*4, [0]*4]
```

Multidimensional Lists

- The indexes start with 0 for both rows and columns
- Once you have created the list, you can refer to individual elements by providing specific row and column number

```
tmp[0][0]          # 0
tmp[0][3] = 87     # set fourth value of first row
tmp[2][0] = 99     # set first value of third row
```

Row indices	0	1	2	3
0	0	0	0	87
1	0	0	0	0
2	99	0	0	0

Column indices

Multidimensional Lists

- The indexes start with 0 for both rows and columns
- Once you have created the list, you can refer to individual elements by providing specific row and column number

```
tmp          # entire grid
tmp[2]       # entire third row
tmp[2][0]    # first element of the third row
```

Multidimensional Lists

- The indexes start with 0 for both rows and columns
- Once you have created the list, you can refer to individual elements by providing specific row and column number

```
tmp = [[0]* 4,[0]* 4, [0]* 4]
len(tmp)     # number of rows
len(tmp[0]) # length of first row (i.e., number of columns)
```

Traverse a multidimensional lists

```
def print_grid(grid):
    for i in range(len(grid)):
        for j in range(len(grid[i]):
            print(grid[i][j], end=" ")
        print()
```

grid

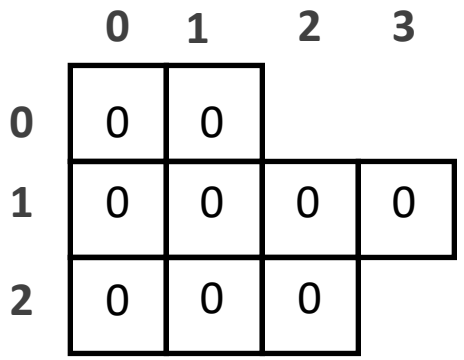
	0	1	2	3
0	0	2	0	87
1	0	0	6	0
2	99	0	9	3

Jagged Lists

- So far, we have seen rectangular grids (fixed number of rows and columns)
- It is also possible to create a jagged list in which the number of columns varies from row to row

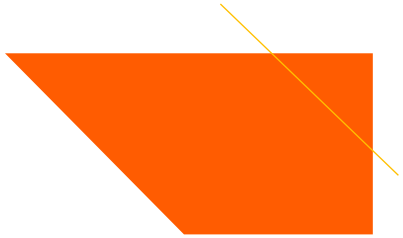
Syntax:

```
jagged = [[0]* 2,[0]* 4, [0]* 3]
```



Syntax:

```
jagged = []
jagged.append([0] * 2)
jagged.append([0] * 4)
jagged.append([0] * 3)
```



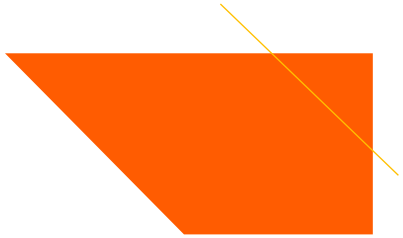
Exercise

- Consider the following function:

```
def mystery2d(a)
    for r in range(len(a)):
        for c in range(len(a[0]) - 1):
            if a[r][c + 1] > a[r][c]:
                a[r][c] = a[r][c+1]
```

If a 2-dimensional list is initialized to store the following integers, what are its contents after the call shown?

```
[[3, 4, 5, 6], [4, 5, 6, 7], [5, 6, 7, 8]]
```



Exercise

- Write a function draw_negative(image), that accepts a 2-D List that represents a simplified black and white image. The image is a 2D List that for every location/pixel (row, and column) has the value 1 to represent the white color, and 0 to represent black. The function will accept an image and convert the black pixels to white and vice-versa. As a last step your function will print the result which will be the negative image.