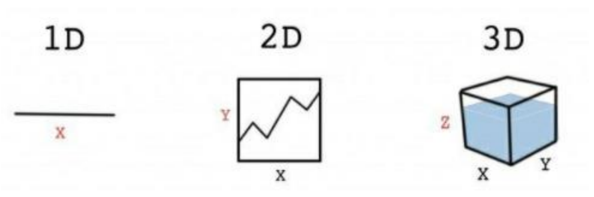
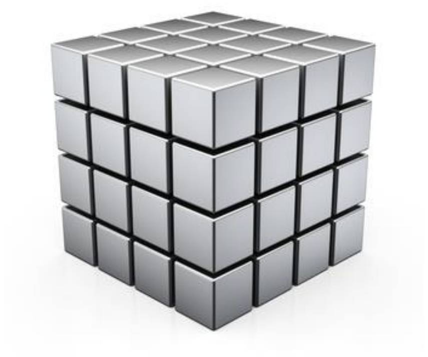
Intro to 2-Dimensional Lists in Python

As you may have noticed, the world we live in is complicated and difficult to represent with single a straight line. We live in a 3-Dimensional world and most of the things we model in mathematics and science have more than a single dimension.



In most programming languages you can create **multi-dimensional** data structures to help up represent anything 2D, 3D, or even more dimensions. These structures are necessary to:

- Model 2-dimentioanal and 3 dimensional *space*
- Create data tables.
- Perform various mathematical operations (*matrices*)
- And much much more…

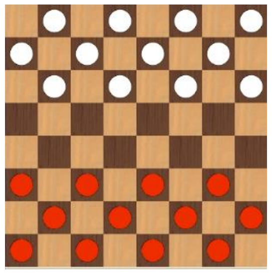


2-Dimensional Lists:

Look at the images to the right. 2D *lists* are simply a 2-Dimensional grid of data. Storing data this way can make **accessing data easier** and allows us to **model real 2-D data in a more accurate way**.

	Col1	Col2	Col3	Col4	····
Row1	Arr[0][0]	Arr[0][1]	Arr[0][2]	Arr[0][3]	
Row2	Arr[1][0]	Arr[1][1]	Arr[1][2]	Arr[1][3]	
Row3	Arr[2][0]	Arr[2][1]	Arr[2][2]	Arr[2][3]	
Row4	Arr[3][0]	Arr[3][1]	Arr[3][2]	Arr[3][3]	
⋮					

Weather we are working with a **table of information** or a **2-D game board**. Using a 2D *lists* is an easy tool to help us out.



In Python a 2D list is… "a list **of** lists":

```
2dlist = [[1, 2], [3, 4],[5, 6]]
```

As you can see above we have a list **of** 3 *sub*-lists:

- Each *sub*-list in the list represents a new **row**
- Each *element* in the each *sub*-list represents a new **column**

1	2
3	4
5	6

2d list addresses (in python)

2dlist[0][0]	2dlist[0][1]
2dlist[1][0]	2dlist[1][1]
2dlist[2][0]	2dlist[1][1]

Exercise#1

Enter the following into the IDE of your choice. Look carefully at the output and **make sure you understand the address system for 2-dimensional lists in Python**.

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

print a

print(a[0])
print(a[1])
print(a[2])
print(a[0][0])
print(a[0][2])
print(a[2][0])

a[0][0]=14
a[1][2]=27
print a
```

⇐ 2D list is… "a list **of** lists":

Create your own **4x4** 2D list of any integers you wish and **print** the following using the 2D list address system illustrated above:

1. first element in the first column of the 2D list
2. the element in the 3rd row and 3rd column.
3. the element in the 4th row and 2nd column.

Now **change** the following elements in your 4x4 list and print the entire 2D list at the end to see if you were able to make the changes correctly:

1. Change the element in the first row of the 1st column to 23
2. Change the element in the 2nd row of the 3rd column to 47
3. Change the element in the 4th row of the 4th column to 100

Save all your work from entire page above and *submit as* **Exercise#1**

Displaying 2D lists

It is often useful/necessary to **display** 2-D lists in a form that is easily readable by humans. Use the example code below to build a student timetable and then display it *neatly* so it can be used by students.

Exercise#2

Enter the code below to create and display a student timetable.

My Timetable

	P1	P2	P3	P4	P5
Monday	History	Maths	Computer Science	PE	Music
Tuesday	English	Spanish	Maths	Geography	Art
Wednesday	PE	English	Science	Art	PE
Thursday	Maths	English	Philosophy	Spanish	Music
Friday	Science	Drama	History	Geography	Science

```
timetable[3][0] = "Maths"
```

```
timetable[0][2] = "Computer Science"
```

```
#My weekly timetable
timetable = []
#Monday
timetable.append(["History","Maths","CompSci","PE","Music"])
#Tuesday
timetable.append(["English","Spanish","Maths","Geography","Art"])
#Wednesday
timetable.append(["PE","English","Science","Art","PE"])
#Thursday
timetable.append(["Maths","English","Philosohpy","Spanish","Music"])
#Friday
timetable.append(["Science","Drama","History","Geography","Science"])
```

```
for x in timetable:
  print (x)
```

```
for row in timetable:
    for val in row:
        print (f'{val:12}',end='')
    print ("")
```

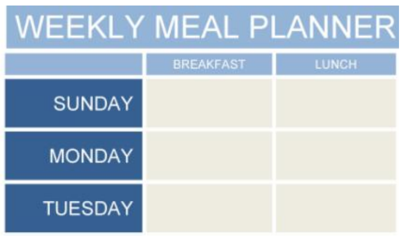
KNOW this great way to display 2D lists!

Exercise#2...continued.

Now *add* to the program on the previous page so it can do the following:

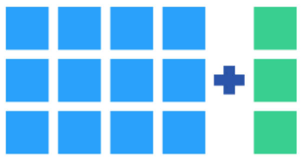
1. Asks the user to input a **day** of the week (e.g. *Tuesday*)
2. Asks the user to input a **period** during the school day (between *1 and 5*)
3. Retrieve and output the class on the **day** and **period** the user selected (e.g. *Spanish*)

Adding/Removing entire Rows or Columns to your 2D lists:



WEEKLY MEAL PLANNER

	BREAKFAST	LUNCH
SUNDAY		
MONDAY		
TUESDAY		



One thing you might want to do with a 2-Dimensional set of data is add an **entire** column or row.

For example let's say you wanted to create a weekly meal plan like the one shown above
As you can see, it's not complete. Maybe you would like to add *"dinners"* to the meal plan or add the rest of the days of the week. This is relatively easy to do in Python.

Exercise#3

Adding Rows:

$$\begin{bmatrix} 1 & 8 & 13 & 12 \\ 14 & 11 & 2 & 7 \\ 4 & 5 & 16 & 9 \\ 15 & 10 & 3 & 6 \end{bmatrix}$$

Enter the following code into an IDE to see how to **add a row**. Then *add a least one more additional row to the 4 rows that have been created.*

```
list=[['coffee','salmon','steak'],
['cereal','sandwich','soup'],
['eggs','sandwich','pasta']]

for x in list:
  print (x)
print('\n')

list.append(['waffles','soup','hambuger'])

 for x in list:
  print (x)
print('\n')
```

Adding rows is simple. Just **append** an extra list to the list.

Exercise#3 continued:

What if we want to **insert** a row somewhere else *besides* the **bottom** row?
We can use the `insert()` function.

Add the following to the code on the previous page and run the program to see what it does. Try changing the 1 to a 2, and run the code again. What do the numbers mean?
Now, *insert **another** row (at the very **top** the chart)*

```
print('\n')

list.insert(1,['toast','chili','chicken'])

for x in list:
  print (x)

print('\n')
```

Exercise#4

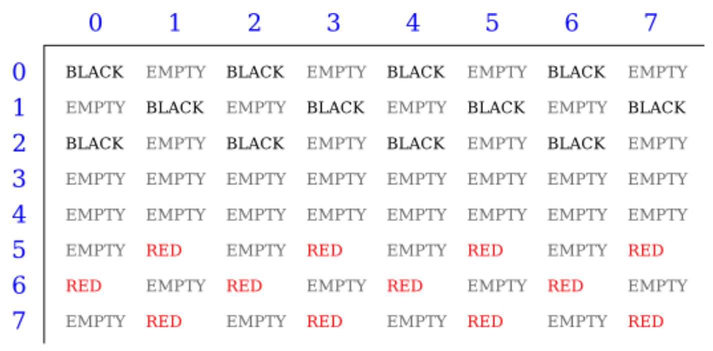
Adding Columns.

Adding, again, to the code from the previous exercise, try to add an additional column to your meal plan called **midnight_snacks**:

```
print('\n')

midnight_snack=['ice cream','cereal','ham sandwich','donut','Glass_o_Milk']

y=0
for x in range(len(list)):
    list[x].append(midnight_snack[y])
    y=y+1
    if y>(len(midnight_snack)):
      break

print('\n')

for x in list:
  print (x)
```

Building 2D list with For Loops

Let’s say you want to create a 2D game that involved a 7X7game board. *Initially* you wish each space on the board to be **EMPTY**. This means you will have to create a 7X7 2D list of “Empty” Values.



	0	1	2	3	4	5	6	7
0	BLACK	EMPTY	BLACK	EMPTY	BLACK	EMPTY	BLACK	EMPTY
1	EMPTY	BLACK	EMPTY	BLACK	EMPTY	BLACK	EMPTY	BLACK
2	BLACK	EMPTY	BLACK	EMPTY	BLACK	EMPTY	BLACK	EMPTY
3	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY
4	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY
5	EMPTY	RED	EMPTY	RED	EMPTY	RED	EMPTY	RED
6	RED	EMPTY	RED	EMPTY	RED	EMPTY	RED	EMPTY
7	EMPTY	RED	EMPTY	RED	EMPTY	RED	EMPTY	RED

Instead of *typing out the entire 2-D list* you could do the following:

Enter the following code into your IDE, run it and **make sure you understand how they both work.**

```
r = 7
c = 7
a = ["Empty"] * r
for i in range(r):
    a[i] = ["Empty"] * c



for row in a:
    for val in row:
        print (f'{val:9}',end='')
    print ("")
```

```
#alternatively you could do:
rows=7
cols=7
two_d_list=[]
for i in range(rows):
  row = []
  for j in range(cols):
     row.append(0)
  two_d_list.append(row)


for x in two_d_list:
  print (x)
```

When you want to represent **2D** data that is *regular* or *repeating* you can usually use a **for loop** like the ones shown on the previous page.

Exercise#5

Use the code on the previous page to:

- Create a 11x11 grid where all elements are the word *"matrix"*. Then:
- Create a 15 x 15 grid where all elements are the integer 0.
- Create a 9 x 9 grid of the word "grass"
- Create a 10 x 10 grid where each positions ***alternates*** between 0 and 1. *(solution below if you are having difficulty)*

Solution to creating a 10 x 10 grid where each positions alternates between 0 and 1.

```
rows = 10
cols = 10

two_d_list=[]
for i in range(rows):
  row = []
  for j in range(cols):
    if j%2==0:
      row.append(0)
    else:
      row.append(1)
  two_d_list.append(row)

for x in two_d_list:
  print x
```

Exercise#5...continued.

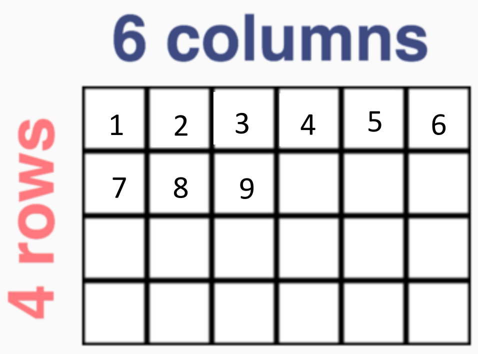
Create a 16 x 16 grid where the **first** element is the word "good" and the *second* elements is the word "bad"....alternate these elements in the array.

Exercise#6

Create a 20 x 20 grid where the first row is all 1's, the second row is all 2's, the third row is all 3's....continue this pattern for all 20 rows.

Exercise#7

Create a 6 x 4 grid where the first *element* is 1 and the *second* is 2 and each element increase until they reach 24.



User Created Input for a 2D array.

Enter the following the examples into an IDE of your choice and ***make sure you know how they work***. Save and submit the examples as Exercise#10. Then work on the exercises after the examples (exercises closely related to the examples).

Exercise#8

Examples:

Example#1 User Inputs each line of input as a row. *User enters entire row on a single line* with *each element separated by a space*. They hit enter to go on to the next row.

```
n = int(input())
a = []
for i in range(n):
    row = input().split()
    for i in range(len(row)):
        row[i] = int(row[i])
    a.append(row)

for x in a:
    print x
```

Example#2 User inputs every element on a separate line. *User enters each element then pressed enter.*

```
grid = []
# taking 3x3 matrix from the user
for i in range(3):
   row = []
   for j in range(3):
      element = int(input())
      row.append(element)
   grid.append(row)

for x in grid:
  print x
```

Exercise#9

Create a program that will allow the user to input a list of grades for a group of students. The first column of each row will be the **student's name** the next 5 elements in each row will be the student's grades separated by a space. Print out the user's input in a neat format.

Exercise#10

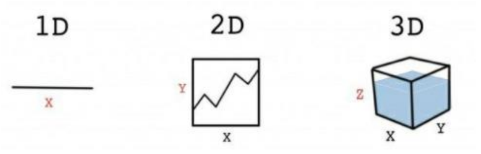
Create a program that allows a student to enter a list of temperature data by just entering the temp for that day and pressing enter. The program should be such that every 8th input will be put into a new row of data (7 days within a week). After the data is entered, print out the data an organised table.

Exercise#11

Create a 2D game text game (using a 2D list) that has the following features:

- A 10 x 10 game board (initially all X's)
- A single player 'O' that can move forward left right back with keystrokes (ex. arrow keys)
- 5 gold coins "G" who's position of the grid are *randomly* generated at the start of each game.
- The player's goal is to move and collect all the coins as fast as they can.
- The game should be **timed**.
- The game ends when the player has gotten all 5 coins.
- The game should show the player's time after each game.
- The game should have clear instructions at the beginning on how to move and how to play.

3D and Other Multi-dimensional Lists.



Just like 2D lists, a 3D list is just a list *of* lists. But in this case we will have sub lists *within* our sub lists:

Exercise#12

Try the following code in Replit. See if you can **predict** the printed output.
Then add print statements that will print out the 3,6,8,8,12.

```
three_d_list=[[[2, 0, 0], [3, 0, 0], [4, 0, 0]],
              [ [0, 4, 0], [0, 6, 0], [0, 8, 0]],
              [ [0, 0, 8], [0, 0, 8], [0, 0, 12]]]

print(three_d_list[0][1])
print(three_d_list[0][1][2])
```

Exercise#13

Creating a 3D lists with nested for loops:

Look carefully at the nested for loops below and the comments to see how you can create a 3D list using for loops. See if you can predict what the print statement will produce. Enter the code in to Replit to see what it does.

```
list3d=[]                                   # create a list

for i in range(3):
    list3d.append([])                       # add 3 empty lists within that list

    for j in range(3):
        list3d[i].append([])                # add 3 more list with each of the previous

        for k in range(3):
            list3d[i][j].append(0)          # put a 0 in for each element in the 3D list

print(list3d)                               # print out the 3D list


for x in list3d:                            # print out the list as 3 (3x3) 2D lists
    print (x)
```

Exercise#14

Creating a 3D lists with list comprehensions:

Look carefully at the **list comprehension** below to see how you can create a 3D list using list comprehensions. See if you can predict what all three print statements will produce. Enter the code in to Replit to see what it does.

```
lst = [[['3D' for x in range(3)] for y in range(3)] for z in range(3)]

print(lst)

for i in lst:
  print (i)

for x in lst:
  for y in x:
    for z in y:
      print (z)
```

Exercise#15

Use a 3-dimensional list to create a virtual "house". The house will be a 3x3x3 list where each element is a different *string* that represent rooms. For example: "Entrance", "kitchen", "Dining Room", "Washroom".

Create a program that will allow a user to go forward, back, left, right, up, and down to explore the different rooms of the house using different keystrokes. Assume that every room has access to all the rooms adjacent to it and above and below it (no stairs needed). Have fun!

Exercise#16

Imagine you are the owner of a 4x4x4 multistory parking garage (each **floor has 16 spaces arranged in a 4x4 square area**). Use a 3-dimensional list to create a model of your garage where an **empty space is represented by the number zero** and an occupied space is occupied by the number one.

Create a program that will allow the user (parking garage attendant) to:

a) repeatedly enter coordinates of spots where cars have come in and have occupied.
b) Your program should give an alert to the user **when a floor has filled up**.
c) Your program should **allow the user to print out a neat diagram of the 3D list** to show what spot are currently occupied.

Example:

```
Car has entered.  What floor has it gone to 1-4?   2
What spot has occupied (1-4 forward, 1-4 right)?  2 3
#the spot that is 2 forward 2 to the right
Thank you!

Car has entered.  What floor has it gone to 1-4?   1
What spot has occupied (1-4 forward, 1-4 right)?  4 3
#the spot that is 4 forward 4 to the right
Thank you!
```