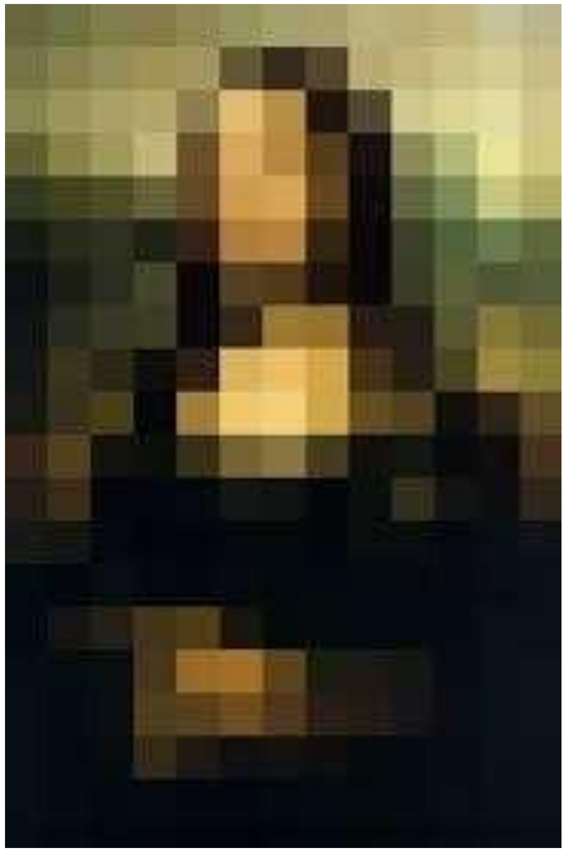
7. Two-Dimensional Arrays

Topics

- Where Do They Come From?
- Visualizing
- 2d Arrays in Python
- Accessing, Printing, Inserting, Updating and Deleting values

Where Do They Come From



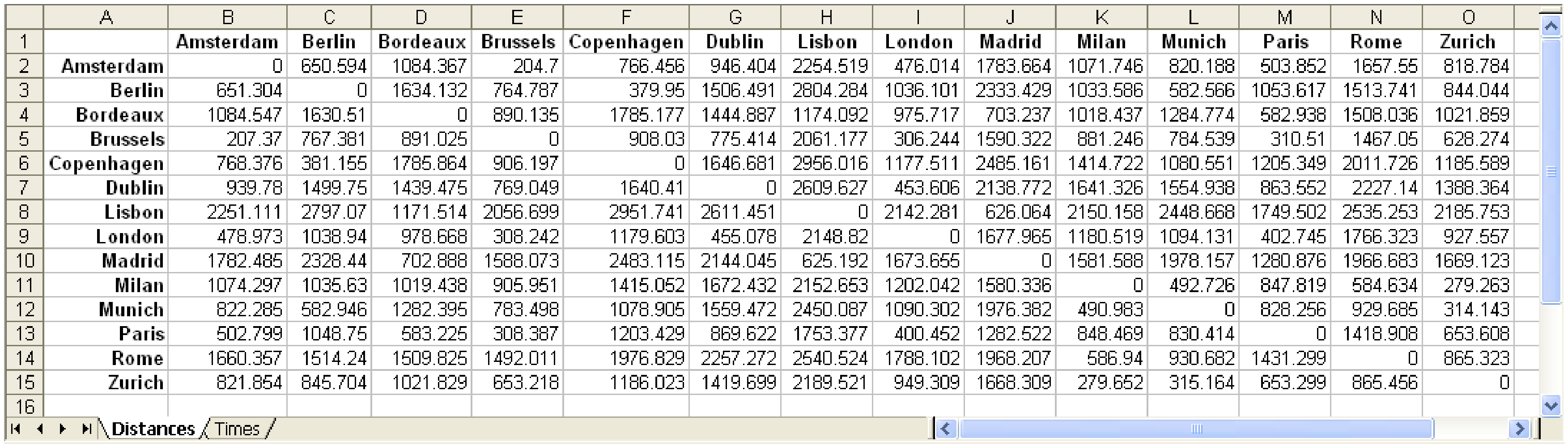
An m-by-n array
of pixels.

Each pixel encodes
3 numbers: a red value,
a green value, a blue
value

So all the information
can be encoded in three
2D arrays

Where Do They Come From?

Entry (i,j) is the distance from city i to city j



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		Amsterdam	Berlin	Bordeaux	Brussels	Copenhagen	Dublin	Lisbon	London	Madrid	Milan	Munich	Paris	Rome	Zurich
2	Amsterdam	0	650.594	1084.367	204.7	766.456	946.404	2254.519	476.014	1783.664	1071.746	820.188	503.852	1657.55	818.784
3	Berlin	651.304	0	1634.132	764.787	379.95	1506.491	2804.284	1036.101	2333.429	1033.586	582.566	1053.617	1513.741	844.044
4	Bordeaux	1084.547	1630.51	0	890.135	1785.177	1444.887	1174.092	975.717	703.237	1018.437	1284.774	582.938	1508.036	1021.859
5	Brussels	207.37	767.381	891.025	0	908.03	775.414	2061.177	306.244	1590.322	881.246	784.539	310.51	1467.05	628.274
6	Copenhagen	768.376	381.155	1785.864	906.197	0	1646.681	2956.016	1177.511	2485.161	1414.722	1080.551	1205.349	2011.726	1185.589
7	Dublin	939.78	1499.75	1439.475	769.049	1640.41	0	2609.627	453.606	2138.772	1641.326	1554.938	863.552	2227.14	1388.364
8	Lisbon	2251.111	2797.07	1171.514	2056.699	2951.741	2611.451	0	2142.281	626.064	2150.158	2448.668	1749.502	2535.253	2185.753
9	London	478.973	1038.94	978.668	308.242	1179.603	455.078	2148.82	0	1677.965	1180.519	1094.131	402.745	1766.323	927.557
10	Madrid	1782.485	2328.44	702.888	1588.073	2483.115	2144.045	625.192	1673.655	0	1581.588	1978.157	1280.876	1966.683	1669.123
11	Milan	1074.297	1035.63	1019.438	905.951	1415.052	1672.432	2152.653	1202.042	1580.336	0	492.726	847.819	584.634	279.263
12	Munich	822.285	582.946	1282.395	783.498	1078.905	1559.472	2450.087	1090.302	1976.382	490.983	0	828.256	929.685	314.143
13	Paris	502.799	1048.75	583.225	308.387	1203.429	869.622	1753.377	400.452	1282.522	848.469	830.414	0	1418.908	653.608
14	Rome	1660.357	1514.24	1509.825	1492.011	1976.829	2257.272	2540.524	1788.102	1968.207	586.94	930.682	1431.299	0	865.323
15	Zurich	821.854	845.704	1021.829	653.218	1186.023	1419.699	2189.521	949.309	1668.309	279.652	315.164	653.299	865.456	0
16															

Visualizing

12	17	49	61
38	18	82	77
83	53	12	10

Can have a 2d array of strings or objects.

But we will just deal with 2d arrays of numbers.

A 2D array has rows and columns.

This one has 3 rows and 4 columns.

We say it is a "3-by-4" array (a.k.a matrix)

Rows and Columns

12	17	49	61
38	18	82	77
83	53	12	10

This is row 1.

Rows and Columns

12	17	49	61
38	18	82	77
83	53	12	10

This is column 2.

Entries

12	17	49	61
38	18	82	77
83	53	12	10

This is the (1,2) entry.

2d Arrays in Python

12	17	49	61
38	18	82	77
83	53	12	10

```
A = [[12,17,49,61],[38,18,82,77],[83,53,12,10]]
```

A list of lists.

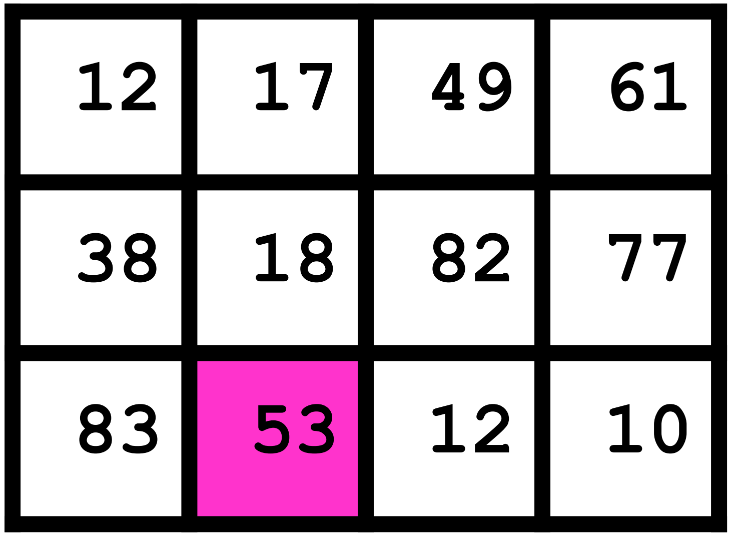
Accessing Entries

12	17	49	61
38	18	82	77
83	53	12	10

`A[1][2]`

```
A = [[12,17,49,61],[38,18,82,77],[83,53,12,10]]
```

Accessing Entries



12	17	49	61
38	18	82	77
83	53	12	10

```
A[2][1]
```

```
A = [[12,17,49,61],[38,18,82,77],[83,53,12,10]]
```

Accessing Entries

The example below illustrates how it works.

```
T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5],
[12,15,8,6]]
print(T[0])
print(T[1][2])
```

When the above code is executed, it produces the following result −

```
[11, 12, 5, 2]
10
```

Printing Arrays

To print out the entire two dimensional array we can use python for loop as shown below. We use end of line to print out the values in different rows.

```
T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5],
[12,15,8,6]]
for r in T:
   for c in r:
      print(c,end = " ")
   print()
```

When the above code is executed, it produces the following result –

```
11 12  5 2
15  6 10
10  8 12 5
12 15  8 6
```

Inserting Values

We can insert new data elements at specific position by using the insert() method and specifying the index.

In the below example a new data element is inserted at index position 2.

```
T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5],
[12,15,8,6]]

T.insert(2, [0,5,11,13,6])
```

```
for r in T:
 for c in r:
    print(c,end = " ")
 print()
```

When the above code is executed, it produces the following result −

```
11 12  5  2
15  6 10
 0  5 11 13 6
10  8 12  5
12 15  8  6
```

Updating Values

We can update the entire inner array or some specific data elements of the inner array by reassigning the values using the array index.

```
T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5],
[12,15,8,6]]

T[2] = [11,9]
T[0][3] = 7
for r in T:
   for c in r:
      print(c,end = " ")
   print()
```

When the above code is executed, it produces the following result −

```
11 12 5  7
15  6 10
11  9
12 15 8  6
```

Deleting the Values

We can delete the entire inner array or some specific data elements of the inner array by reassigning the values using the del() method with index. But in case you need to remove specific data elements in one of the inner arrays, then use the update process described above.

```
T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5],
[12,15,8,6]]
del T[3]
for r in T:
   for c in r:
      print(c,end = " ")
   print()
```

When the above code is executed, it produces the following result −

```
11 12 5 2
15 6 10
10 8 12 5
```