Access Data in Cell Array

This example shows how to read and write data to and from a cell array.

Open Live Script

Create a 2-by-3 cell array of text and numeric data.

```
C = {'one', 'two', 'three';
     1, 2, 3}
```

```
C=2×3 cell array
    {'one'}    {'two'}    {'three'}
    {[  1]}    {[  2]}    {[    3]}
```

There are two ways to refer to the elements of a cell array. Enclose indices in smooth parentheses, `()`, to refer to sets of cells--for example, to define a subset of the array. Enclose indices in curly braces, `{}`, to refer to the text, numbers, or other data within individual cells.

Cell Indexing with Smooth Parentheses, ()

Cell array indices in smooth parentheses refer to sets of cells. For example, to create a 2-by-2 cell array that is a subset of `C`, use smooth parentheses.

```
upperLeft = C(1:2,1:2)
```

```
upperLeft=2×2 cell array
    {'one'}    {'two'}
    {[  1]}    {[  2]}
```

Update sets of cells by replacing them with the same number of cells. For example, replace cells in the first row of `C` with an equivalent-sized (1-by-3) cell array.

```
C(1,1:3) = {'first','second','third'}
```

```
C=2×3 cell array
    {'first'}    {'second'}    {'third'}
    {[    1]}    {[     2]}    {[    3]}
```

If cells in your array contain numeric data, you can convert the cells to a numeric array using the `cell2mat` function.

```
numericCells = C(2,1:3)
```

```
numericCells=1×3 cell array
    {[1]}    {[2]}    {[3]}
```

```
numericVector = cell2mat(numericCells)
```

```
numericVector = 1×3

     1     2     3
```

`numericCells` is a 1-by-3 cell array, but `numericVector` is a 1-by-3 array of type `double`.

Content Indexing with Curly Braces, {}

Access the contents of cells--the numbers, text, or other data within the cells--by indexing with curly braces. For example, to access the contents of the last cell of `C`, use curly braces.

```
last = C{2,3}
```

```
last = 3
```

`last` is a numeric variable of type `double`, because the cell contains a `double` value.

Similarly, you can index with curly braces to replace the contents of a cell.

```
C{2,3} = 300
```

```
C=2×3 cell array
    {'first'}    {'second'}    {'third'}
    {[    1]}    {[     2]}    {[  300]}
```

You can access the contents of multiple cells by indexing with curly braces. MATLAB® returns the contents of the cells as a *comma-separated list*. Because each cell can contain a different type of data, you cannot assign this list to a single variable. However, you can assign the list to the same number of variables as cells. MATLAB® assigns to the variables in column order.

Assign contents of four cells of `C` to four variables.

```
[r1c1, r2c1, r1c2, r2c2] = C{1:2,1:2}
```

```
r1c1 =
'first'
r2c1 = 1
r1c2 =
'second'
r2c2 = 2
```

If each cell contains the same type of data, you can create a single variable by applying the array concatenation operator, `[]`, to the comma-separated list.

Concatenate the contents of the second row into a numeric array.

```
nums = [C{2,:}]
```

```
nums = 1×3

     1     2   300
```

See Also

`cell` | `cell2mat`

Related Topics

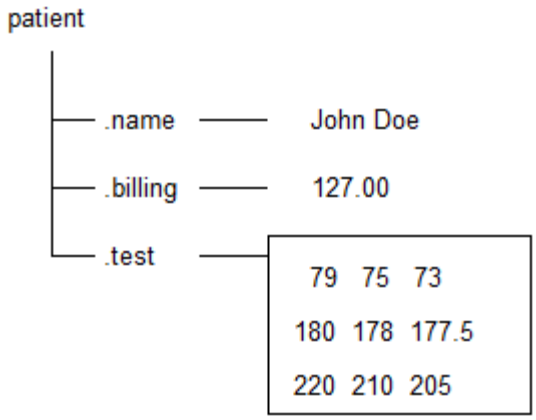
- Create Cell Array
- Multilevel Indexing to Access Parts of Cells
- Comma-Separated Lists

Structure Arrays

When you have data that you want to organize by name, you can use structures to store it. Structures store data in containers called *fields*, which you can then access by the names you specify. Use dot notation to create, assign, and access data in structure fields. If the value stored in a field is an array, then you can use array indexing to access elements of the array. When you store multiple structures as a structure array, you can use array indexing and dot notation to access individual structures and their fields.

Create Scalar Structure

First, create a structure named `patient` that has fields storing data about a patient. The diagram shows how the structure stores data. A structure like `patient` is also referred to as a *scalar structure* because the variable stores one structure.



Use dot notation to add the fields `name`, `billing`, and `test`, assigning data to each field. In this example, the syntax `patient.name` creates both the structure and its first field. The commands that follow add more fields.

```
patient.name = 'John Doe';
patient.billing = 127;
patient.test = [79 75 73; 180 178 177.5; 220 210 205]
```

```
patient = struct with fields:
       name: 'John Doe'
    billing: 127
       test: [3x3 double]
```

Access Values in Fields

After you create a field, you can keep using dot notation to access and change the value it stores.

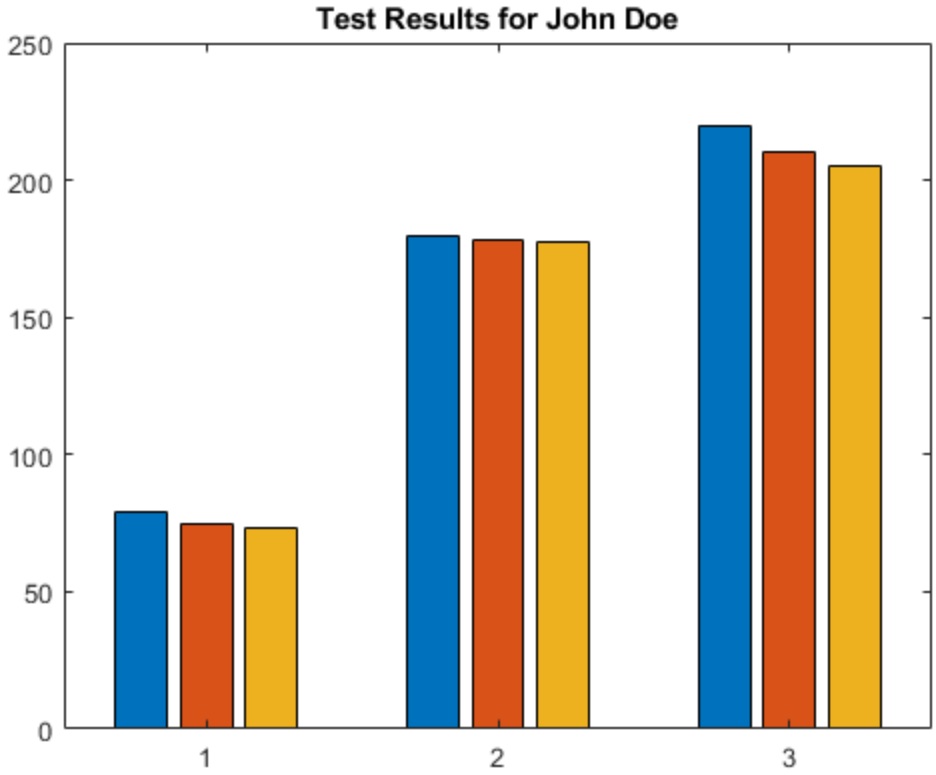
For example, change the value of the `billing` field.

```
patient.billing = 512.00
```

```
patient = struct with fields:
       name: 'John Doe'
    billing: 512
       test: [3x3 double]
```

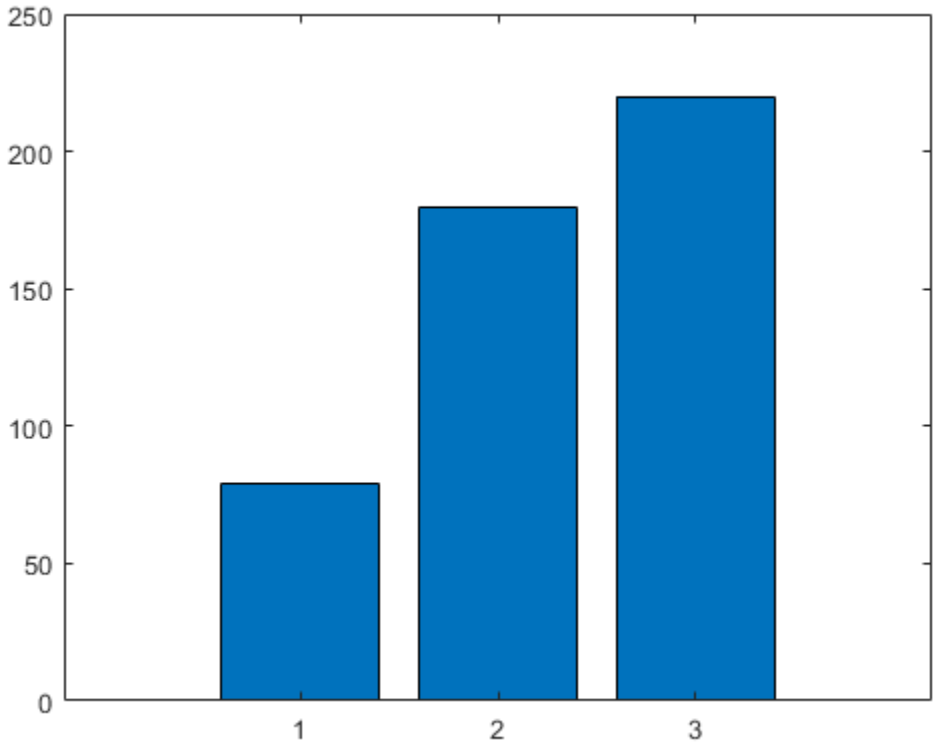
With dot notation, you also can access the value of any field. For example, make a bar chart of the values in `patient.test`. Add a title with the text in `patient.name`. If a field stores an array, then this syntax returns the whole array.

```
bar(patient.test)
title("Test Results for " + patient.name)
```



To access part of an array stored in a field, add indices that are appropriate for the size and type of the array. For example, create a bar chart of the data in one column of `patient.test`.

```
bar(patient.test(:,1))
```



Index into Nonscalar Structure Array

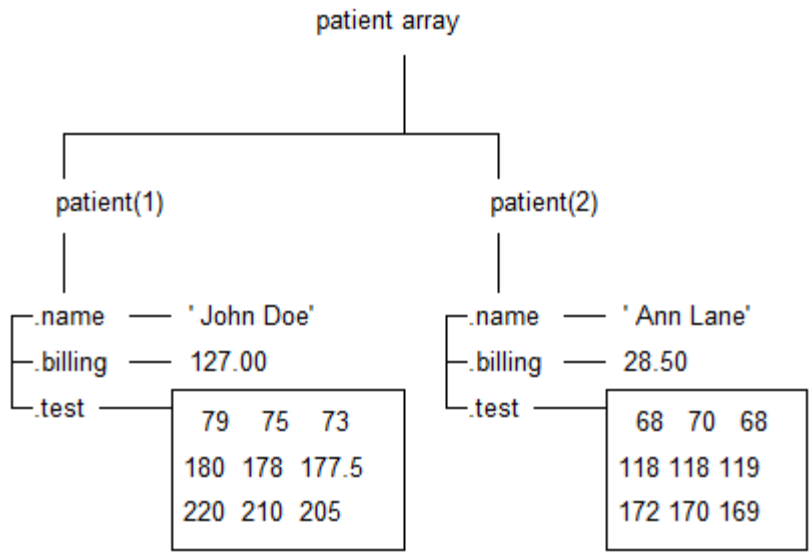
Structure arrays can be nonscalar. You can create a structure array having any size, as long as each structure in the array has the same fields.

For example, add a second structure to `patients` having data about a second patient. Also, assign the original value of `127` to the `billing` field of the first structure. Since the array now has two structures, you must access the first structure by indexing, as in `patient(1).billing = 127`.

```
patient(2).name = 'Ann Lane';
patient(2).billing = 28.50;
patient(2).test = [68 70 68; 118 118 119; 172 170 169];
patient(1).billing = 127
```

```
patient=1×2 struct array with fields:
    name
    billing
    test
```

As a result, `patient` is a 1-by-2 structure array with contents shown in the diagram.



Each patient record in the array is a structure of class `struct`. An array of structures is sometimes referred to as a *struct array*. However, the terms *struct array* and *structure array* mean the same thing. Like other MATLAB® arrays, a structure array can have any dimensions.

A structure array has the following properties:

- All structures in the array have the same number of fields.
- All structures have the same field names.
- Fields of the same name in different structures can contain different types or sizes of data.

If you add a new structure to the array without specifying all of its fields, then the unspecified fields contain empty arrays.

```
patient(3).name = 'New Name';
patient(3)
```

```
ans = struct with fields:
       name: 'New Name'
    billing: []
       test: []
```

To index into a structure array, use array indexing. For example, `patient(2)` returns the second structure.

```
patient(2)
```

```
ans = struct with fields:
       name: 'Ann Lane'
```

```
    billing: 28.5000
       test: [3x3 double]
```

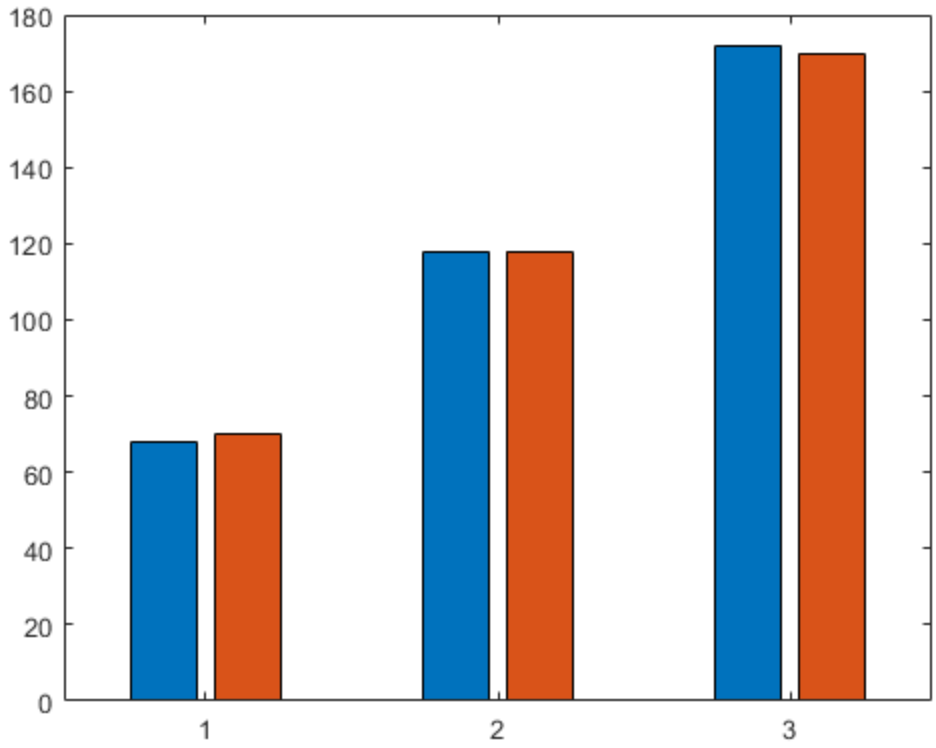
To access a field, use array indexing and dot notation. For example, return the value of the `billing` field for the second patient.

```
patient(2).billing
```

```
ans = 28.5000
```

You also can index into an array stored by a field. Create a bar chart displaying only the first two columns of `patient(2).test`.

```
bar(patient(2).test(:,[1 2]))
```



Note

You can index into part of a field only when you refer to a single element of a structure array. MATLAB® does not support statements such as `patient(1:2).test(1:2,2:3)`, which attempt to index into a field for multiple elements of the structure array. Instead, use the `arrayfun` function.

See Also

`fieldnames` | `isfield` | `struct`

Related Topics

- Access Elements of a Nonscalar Structure Array
- Generate Field Names from Variables
- Create Cell Array
- Cell vs. Structure Arrays
- Create and Work with Tables
- Advantages of Using Tables

Cell vs. Structure Arrays

This example compares cell and structure arrays, and shows how to store data in each type of array. Both cell and structure arrays allow you to store data of different types and sizes.

Open Live Script

Structure Arrays

Structure arrays contain data in fields that you access by name.

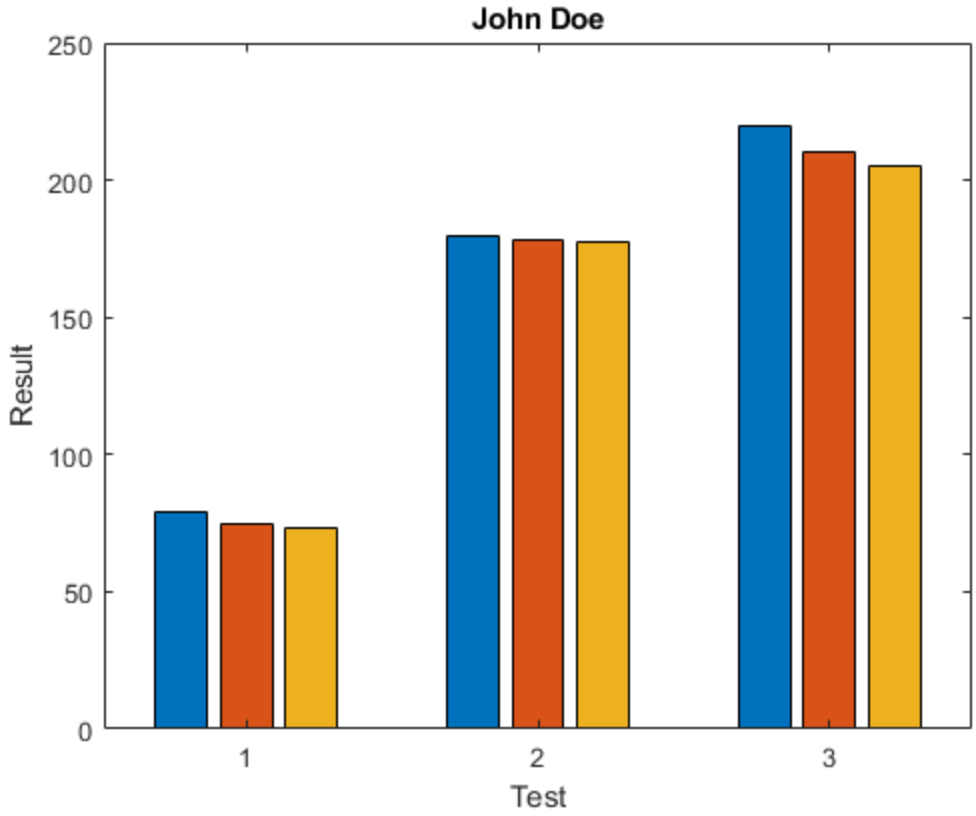
For example, store patient records in a structure array.

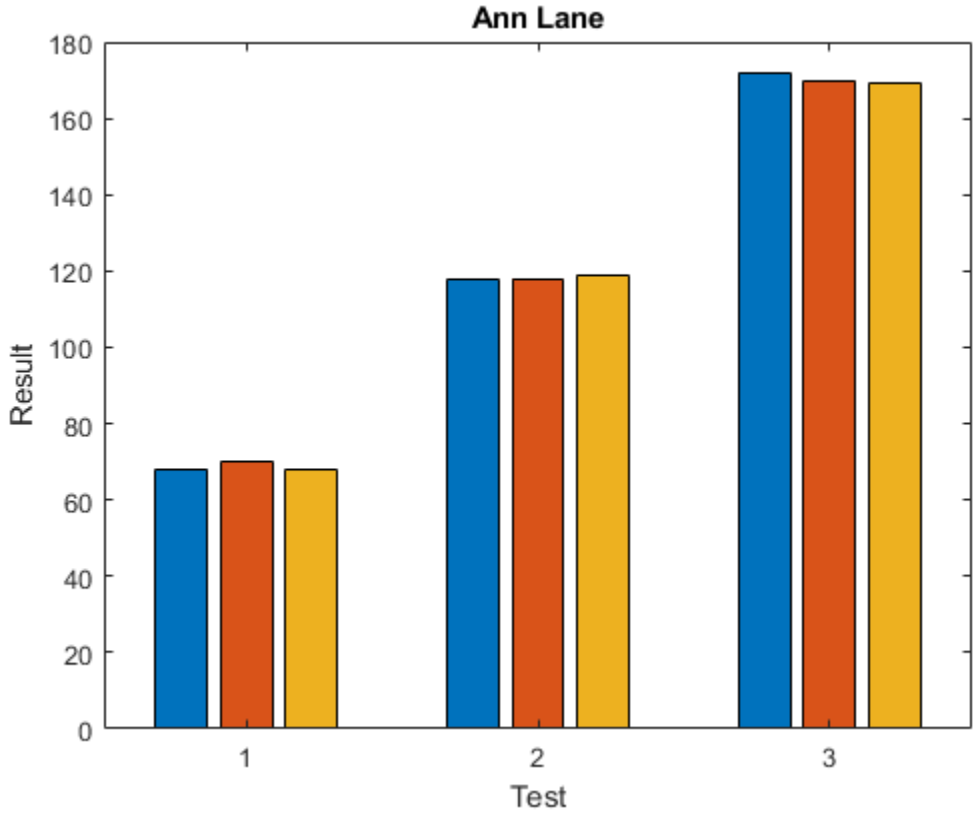
```
patient(1).name = 'John Doe';
patient(1).billing = 127.00;
patient(1).test = [79, 75, 73; 180, 178, 177.5; 220, 210, 205];

patient(2).name = 'Ann Lane';
patient(2).billing = 28.50;
patient(2).test = [68, 70, 68; 118, 118, 119; 172, 170, 169];

patient
```

```
patient=1×2 struct array with fields:
    name
    billing
    test
```

Create a bar graph of the test results for each patient.

```
numPatients = numel(patient);
for p = 1:numPatients
   figure
   bar(patient(p).test)
   title(patient(p).name)
   xlabel('Test')
   ylabel('Result')
end
```





Cell Arrays

Cell arrays contain data in cells that you access by numeric indexing. Common applications of cell arrays include storing separate pieces of text and storing heterogeneous data from spreadsheets.

For example, store temperature data for three cities over time in a cell array.

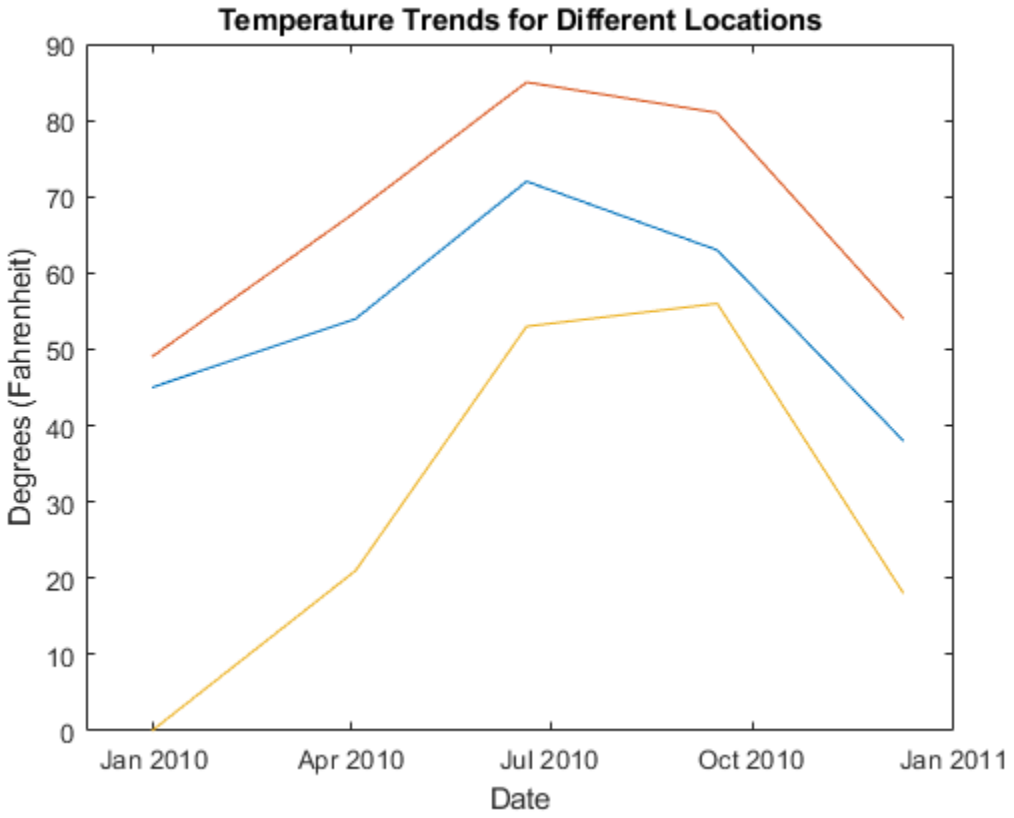
```
temperature(1,:) = {'2009-12-31', [45, 49, 0]};
temperature(2,:) = {'2010-04-03', [54, 68, 21]};
temperature(3,:) = {'2010-06-20', [72, 85, 53]};
temperature(4,:) = {'2010-09-15', [63, 81, 56]};
```

```
temperature(5,:) = {'2010-12-09', [38, 54, 18]};

temperature
```

```
temperature=5×2 cell array
    {'2009-12-31'}    {[ 45 49 0]}
    {'2010-04-03'}    {[54 68 21]}
    {'2010-06-20'}    {[72 85 53]}
    {'2010-09-15'}    {[63 81 56]}
    {'2010-12-09'}    {[38 54 18]}
```

Plot the temperatures for each city by date.

```
allTemps = cell2mat(temperature(:,2));
dates = datetime(temperature(:,1));

plot(dates,allTemps)
title('Temperature Trends for Different Locations')
xlabel('Date')
ylabel('Degrees (Fahrenheit)')
```



Other Container Arrays

Struct and cell arrays are the most commonly used containers for storing heterogeneous data. Tables are convenient for storing heterogeneous column-oriented or tabular data. Alternatively, use map containers, or create your own class.

See Also

`cell` | `cell2mat` | `containers.Map` | `datetime` | `plot` | `struct` | `table`

Related Examples

- Access Data in Cell Array
- Structure Arrays
- Access Data in Tables