

# Cell and Structure Arrays

- A cell array is an array in which each element is a *bin*, or *cell*, which can contain an array. You can store different classes of arrays in a cell array, and you can group data sets that are related but have different dimensions. You access cell arrays using the same indexing operations used with ordinary arrays.
- You can create a cell array by using assignment statements or by using `cell` function. We will look at the latter first.

---

<b>Function</b>	<b>Description</b>
<code>C = cell(n)</code>	Creates an $n \times n$ cell array <code>C</code> of empty matrices
<code>C = cell(n,m)</code>	Creates an $n \times m$ cell array <code>C</code> of empty matrices
<code>celldisp(C)</code>	Displays the contents of cell array <code>C</code>
<code>cellplot(C)</code>	Displays the graphical representation of the cell array <code>C</code>
<code>C = num2cell(A)</code>	Converts a numeric array <code>A</code> into a cell array <code>C</code>
<code>[X,Y, ...] = deal(A,B, ...)</code>	Matches up the input and output lists. Equivalent to <code>X = A, Y = B, ...</code>
<code>[X,Y, ...] = deal(A)</code>	Matches up the input and output lists. Equivalent to <code>X = A, Y = A, ...</code>
<code>iscell(C)</code>	Returns a 1 if <code>C</code> is a cell array; otherwise, returns a 0.

## Using Cell Indexing

```
>> A(1,1)={'Walden Pond'};
>> A(1,2)={'June 13, 1997'};
>> A(2,1)={[60,72,65]};
>> A(2,2)={[55,57,56;54,56,55;52,55,53]};
>> A
```

A =

```
'Walden Pond'    'June 13, 1997'
 [1x3 double]    [3x3 double]
```

Type `celldisp(A)`

A{1,1} =

Walden Pond

A{2,1} =

60 72 65

A{1,2} =

June 13, 1997

A{2,2} =

55 57 56  
54 56 55  
52 55 53

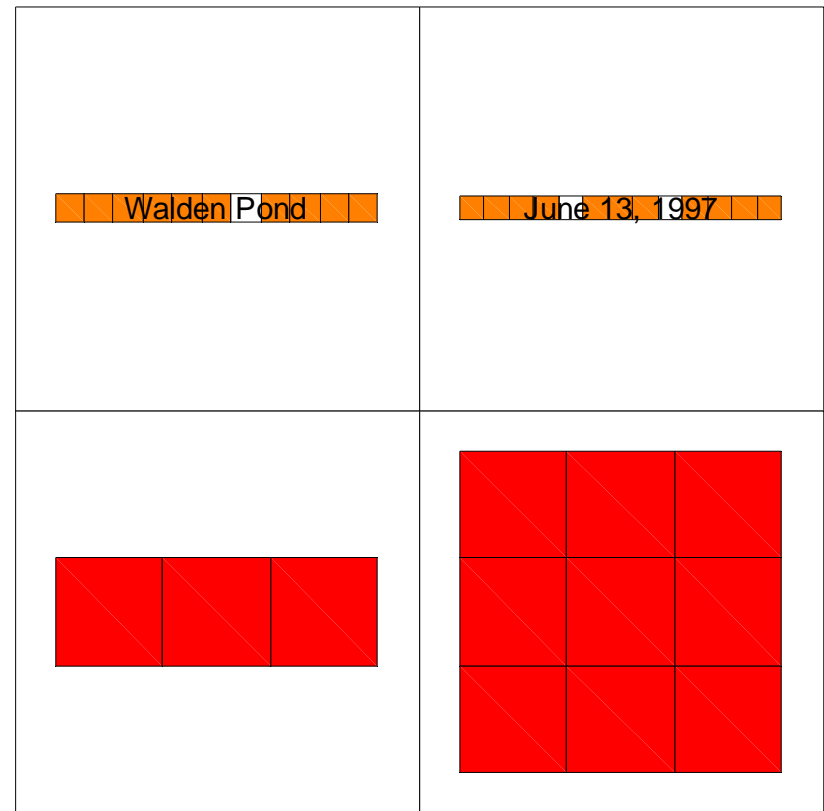
# Using Content Indexing and the `celldisp` function

```
>> A{1,1}='Walden Pond';  
>> A{1,2}='June 13, 1997';  
>> A{2,1}=[60,72,65];  
>> A{2,2}=[55,57,56;54,56,55;52,55,53];  
>> A
```

A =

```
'Walden Pond'    'June 13, 1997'  
[1x3 double]    [3x3 double]
```

Type `cellplot(A)`



## Creating a 2x2 cell array

```
>> B={ [2,4], [6,-9;3,5]; [7;2],  
10};  
  
>> disp(B)  
  
[1x2 double] [2x2 double]  
[2x1 double] [          10]  
  
>> B  
  
B =  
  
[1x2 double] [2x2 double]  
[2x1 double] [          10]
```

**Do not** name a cell array with the same name as a previously used numeric array without first using the `clear` command! **Use** the `iscell` function to determine if an array is a cell array. And, **use** the `num2cell` function to convert a numeric array to a cell array,

## Preallocating Empty Cell Arrays

You can preallocate cell arrays of a specified size by using the `cell` function.

```
>> C=cell(3,5)  
  
C =  
  
[] [] [] [] []  
[] [] [] [] []  
[] [] [] [] []
```

Now you can use assignment statements to enter the contents of the cells:

`C(2,4)={ [6,-3,7] }` puts a 1x3 array in cell (2,4)

`C(1,5)={ 1:10 }` puts the numbers from 1 to 10 in cell (1,5)

`C(3,4)={ '30 mph' }` puts the string in cell (3,4)

# Accessing Cell Arrays

- You can access the contents of a cell array by using either cell indexing or content indexing:
  - `Speed = C(3,4)` places the contents of cell (3,4) of the array `C` in the new variable `Speed`
  - `D = C(1:3, 2:5)` This places the contents of the cells in rows 1 to 3, columns 2 to 5 in the new cell array `D`. The new cell array will have three rows, four columns, and 12 arrays.
  - `Speed = C{3,4}` To access some or all of the contents in a *single cell*, enclose the cell index expression in braces to indicate that you are assigning the contents, not the cells themselves, to a new variable. In this example, the content of cell (3,4) or '30 mph' is assigned to the variable `Speed`.
  - You cannot use content indexing to retrieve the contents of more than one cell at a time

# Structure Arrays

Structure arrays are composed of *structures*. This class of arrays allows you to store dissimilar arrays together. The elements in the structure are accessed using *named fields*.

Think about a student database (e.g. name, social security number, email address, test scores). There are four fields (4 field names): 3 string and 1 vector containing numerical elements. *A structure consists of all this information for a single student and a structure array is an array of such structures for different students.*

You create a structure array either by using assignment statements or by using the `struct` function. Structure arrays use the dot notation (`.`) to specify and to access the fields.

```

>> student.name='John Smith';
>> student.ssn='123-45-6789';
>> student.email='smithj@myschool.edu';
>> student.tests=[67,75,84];
>> student
student =
    name: 'John Smith'
    ssn: '123-45-6789'
    email: 'smithj@myschool.edu'
    tests: [67 75 84]
>> size(student)
ans =
     1     1
>> student(2).name='Mary Jones';
>> student(2).ssn='987-65-4321';
>> student(2).email='jonesm@myschool.edu';
>> student(2).tests=[84,78,93];
>> student
student =
1x2 struct array with fields:
    name
    ssn
    email
    tests

```

In the first part of this Matlab session, we are building information about 1 student (or a 1x1 structure array). In the second part, we are adding a second student. We now have a 1x2 structure array consisting of two structures arranged in one row and two columns.

Or, you can use the `struct` function to build structures (and *preallocate* a structure array).

```
Sa_1=struct('field1','values1',
'field2','values2',...)

```

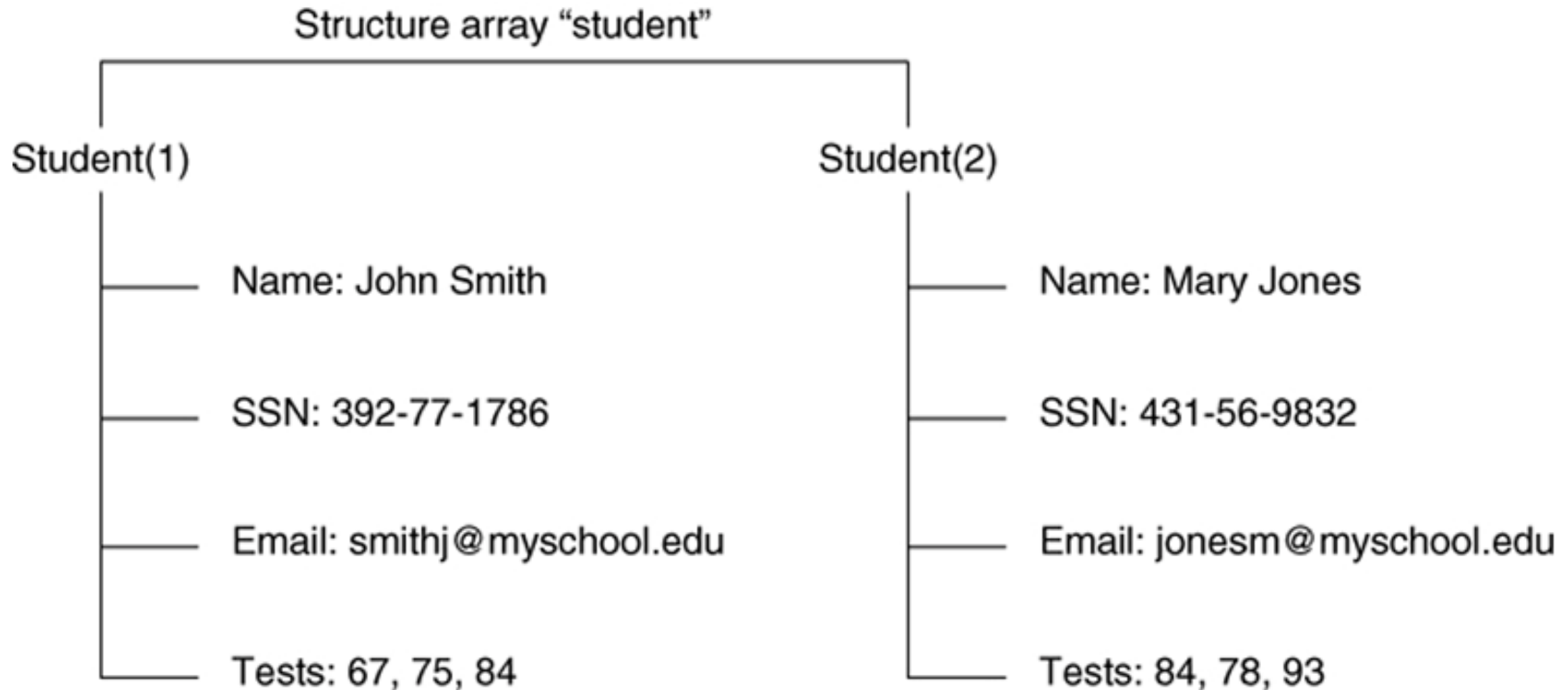
The arguments are the field names and their values. The values arrays `values1`, etc. must all be arrays of the same size, scalar cells, or single values. The elements of the values arrays are inserted into the corresponding elements of the structure array. The resulting structure array has the same size as the values arrays, or is 1x1 if none of the values arrays is a cell.

```

student=struct('name','John Smith',
'SSN','123-45-6789','email',
'smithj@myschool.edu','tests',[67,75,84])

```

# Arrangement of data in the structure array `student`.



# Structure functions

## Function

```
names = fieldnames(S)
```

```
F = getfield(S, 'field')
```

```
isfield(S, 'field')
```

## Description

Returns the field names associated with the structure array `S` as `names`, a cell array of strings.

Returns the contents of the field `'field'` in the structure array `S`. Equivalent to `F = S.field`.

Returns 1 if `'field'` is the name of a field in the structure array `S`, and 0 otherwise.

# Structure functions

```
S =  
rmfield(S, 'field')
```

Removes the field 'field' from the structure array S.

```
S =  
setfield(S, 'field',  
V)
```

Sets the contents of the field 'field' to the value V in the structure array S.

```
S =  
struct('f1', 'v1', 'f  
2', 'v2', ...)
```

Creates a structure array with the fields 'f1', 'f2', . . . having the values 'v1', 'v2', . . . .

# Accessing Structure Arrays and Modifying Structures

Type a period after the structure array name, followed by the field name to access the contents of a particular field:

`student(2).name` returns the value 'Mary Jones'

And, you can assign the result to a variable in the usual way:

`name2=student(2).name` assigns the value 'Mary Jones' to the variable `name2`.

To add more information to an existing database, do the following:

`student(1).phone='555=1653'` This adds a phone number field to the first student's; All other structures in the array will also have the phone number field, but will contain an empty array until you give them values.

To delete a field from every structure in the array, use the `rmfield` function. `new_struct=rmfield(array,'field');` where `array` is the structure array to be modified, `'field'` is the field to be removed, and `new_struct` is the name of the new structure array so created by the removal of the field.