Computer Programming with MATLAB

VANDERBILT UNIVERSITY

Lesson 7: Data Types

by

Akos Ledeczi and Mike Fitzpatrick

The Limitation of Computers

- Real numbers in mathematics:
  - Can be infinitely large
  - Have infinitely fine resolution
- Computers: Finite memory
  - Upper limit on the largest number that can be represented
  - Lower limit on the absolute value of any non-zero number
- The set of values that can be represented by a MATLAB variable is finite.

Data Types

- MATLAB: many different data types
- A data type is defined by:
  - Set of values
  - Set of operations that can be performed on those values
- MATLAB:
  - All elements of a given array must be of the same type
  - Elementary type
- Type of a MATLAB array is defined by
  - Number of dimensions
  - Size in each dimension
  - Elementary type

Numerical Types

- double
  - Default type in MATLAB
  - Floating point representation
    - Example: $12.34 = 1234 * 10^{-2}$
    - Mantissa and exponent
  - 64 bits (8 bytes)
- single
  - 32-bit floating point
- Integer types
  - Signed, unsigned
  - 8-, 16-, 32-, 64-bit long

Range of Values

DATA TYPE	RANGE OF VALUES
int8	-2^7 to 2^7-1
int16	-2^{15} to $2^{15}-1$
int32	-2^{31} to $2^{31}-1$
int64	-2^{63} to $2^{63}-1$
uint8	0 to 2^8-1
uint16	0 to $2^{16}-1$
uint32	0 to $2^{32}-1$
uint64	0 to $2^{64}-1$
single	-3.4×10^{38} to 3.4×10^{38}, Inf, NaN
double	-1.79×10^{308} to 1.79×10^{308}, Inf, NaN

Range of Values

DATA TYPE	RANGE OF VALUES
int8	-2^7 to 2^7-1
int16	-2^{15} to $2^{15}-1$
int32	-2^{31} to $2^{31}-1$
int64	-2^{63} to $2^{63}-1$
uint8	0 to 2^8-1
uint16	0 to $2^{16}-1$
uint32	0 to $2^{32}-1$
uint64	0 to $2^{64}-1$
single	-3.4×10^{38} to 3.4×10^{38}, Inf, NaN
double	-1.79×10^{308} to 1.79×10^{308}, Inf, NaN

Inf:
   "Infinity"
NaN:
   "Not-a-number"

Useful functions

- Type check:
  - ◦ `class`
  - ◦ `isa`
    `>> isa(x,'double')`
- Range check:
  - ◦ `intmax`, `intmin`
  - ◦ `realmax`, `realmin`
    `>> intmax('uint32')`
- Conversion:
  - ◦ Name of function = name of desired data type
  - ◦ `int8(x), uint32(x), double(x)`, etc.

Operators

- Arithmetic operators
  - Operands of the same data type: No problem
  - Different data types:
    - "mixed-mode arithmetic"
    - Many rules and restrictions
- Relational operators
  - Different data types are always allowed
  - Result is always of logical type

Strings

- Text: string
- We have used them already:
  - Argument to `fprintf` and other functions
- String: vector of char-s
- Numerical type
  - Uses an encoding scheme
  - Each character is represented by a number
  - ASCII scheme

ASCII code

- American Standard Code for Information Interchange
- Developed in the1960's
- 7 bits, 128 characters:
  - Latin alphabet
  - Digits
  - Punctuation
  - Special characters
- Newer schemes with far more characters
- ASCII is a subset of them

(nul)	0	(sp)	32	@	64	`	96
(soh)	1	!	33	A	65	a	97
(stx)	2	"	34	B	66	b	98
(etx)	3	#	35	C	67	c	99
(eot)	4	$	36	D	68	d	100
(enq)	5	%	37	E	69	e	101
(ack)	6	&	38	F	70	f	102
(bel)	7	'	39	G	71	g	103
(bs)	8	(	40	H	72	h	104
(ht)	9	)	41	I	73	i	105
(nl)	10	*	42	J	74	j	106
(vt)	11	+	43	K	75	k	107
(np)	12	,	44	L	76	l	108
(cr)	13	-	45	M	77	m	109
(so)	14	.	46	N	78	n	110
(si)	15	/	47	O	79	o	111
(dle)	16	0	48	P	80	p	112
(dc1)	17	1	49	Q	81	q	113
(dc2)	18	2	50	R	82	r	114
(dc3)	19	3	51	S	83	s	115
(dc4)	20	4	52	T	84	t	116
(nak)	21	5	53	U	85	u	117
(syn)	22	6	54	V	86	v	118
(etb)	23	7	55	W	87	w	119
(can)	24	8	56	X	88	x	120
(em)	25	9	57	Y	89	y	121
(sub)	26	:	58	Z	90	z	122
(esc)	27	;	59	[	91	{	123
(fs)	28	<	60	\	92	|	124
(gs)	29	=	61	]	93	}	125
(rs)	30	>	62	^	94	~	126
(us)	31	?	63	_	95	(del)	127

Exercise

▸ Print out the visible characters of the ASCII table:

```matlab
function char_codes
for ii = 33:126
    fprintf('%s',char(ii));
end
fprintf('\n');
```

String functions

FUNCTION	DESCRIPTION
char	converts type to char
findstr	finds the positions of a substring in a string
ischar	returns 1 if argument is a character array and 0 otherwise
isletter	finds letters in string
isspace	finds spaces, newlines, and tabs in string
isstrprop	finds characters of specified type in string
num2str	converts number to string
length	determines the number of letters in string
lower	converts string to lower case
sprintf	writes formatted data to string (compare with fprintf)
strcmp	compares strings
strcmpi	like strcmp but independent of case
strmatch	search array for rows that begin with specified string
strncmp	like strcmp but compares only first n characters
strncmpi	like strncmp but independent of case
str2num	converts string to number
upper	converts string to upper case

Structs

- An array must be homogeneous:
  - It cannot contain elements of multiple types.
- A struct can be heterogeneous:
  - It <u>can</u> contain multiple types.
- A struct is different from an array:
  - fields, not elements
  - field names, not indices
  - Fields in the same struct can have different types.
- Versatility inside:
  - A field of a struct can contain another struct.

Structs in action

```
>> r.ssn = 12345678
r =
        ssn: 12345678
>> class(r)
ans =
        struct
>> class(r.ssn)
ans =
        double
>> r.name = 'Homer Simpson'
r =
        ssn: 12345678
        name: 'Homer Simpson'
>> r.address.street = '742 Evergreen Terrace'
r =
        ssn: 12345678
        name: 'Homer Simpson'
        address: [1x1 struct]
```

Structs

▸ An array must be homogeneous:
  ◦ It cannot contain elements of multiple types.
▸ A struct can be heterogeneous:
  ◦ It can contain multiple types.
▸ A struct is different from an array:
  ◦ fields, not elements
  ◦ field names, not indices
  ◦ Fields in the same struct can have different types.
▸ Versatility inside:
  ◦ A field of a struct can contain another struct.
  ◦ **Structs can hold arrays, and arrays can hold structs.**

Struct functions

FUNCTION	DESCRIPTION
getfield	returns the value of a field whose name is specified by a string
isfield	true if a struct has a field whose name is specified by a string
isstruct	returns true if argument is of type struct
orderfields	changes the order of the fields in a struct
rmfield	removes from a struct a field whose name is specified by a string
setfield	assigns a value to a field whose name is specified by a string -or- if the field is not present, adds it and assigns it the value
struct	create a struct with fields whose name are specified by strings

Pointers

- How to store a page of text?
  - Each line should be a separate string
  - Cannot use an array of chars:
    - Each line would have to have the same length
  - A vector of objects with each referring to one line
- Pointer
  - Each variable (scalar, vector, array, etc.) is stored in the computer memory.
  - Each memory location has a unique address.
  - A pointer is a variable that stores an address.
  - MATLAB calls a pointer a "cell".

Cells

- MATLAB has a restrictive pointer model
  - Strict rules on what can be done with cells
  - Harder to make mistakes
- But it is a powerful way to store heterogeneous data
  - Cell arrays
  - Used more frequently than structs
- New syntax:
  - To access the data a cell points to, use: { }

Cell array example
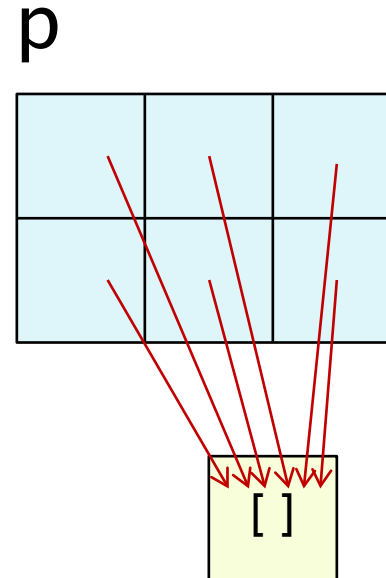
```
>> p = cell(2,3)
p =
   [ ]        [ ]        [ ]
   [ ]        [ ]        [ ]
>>
```
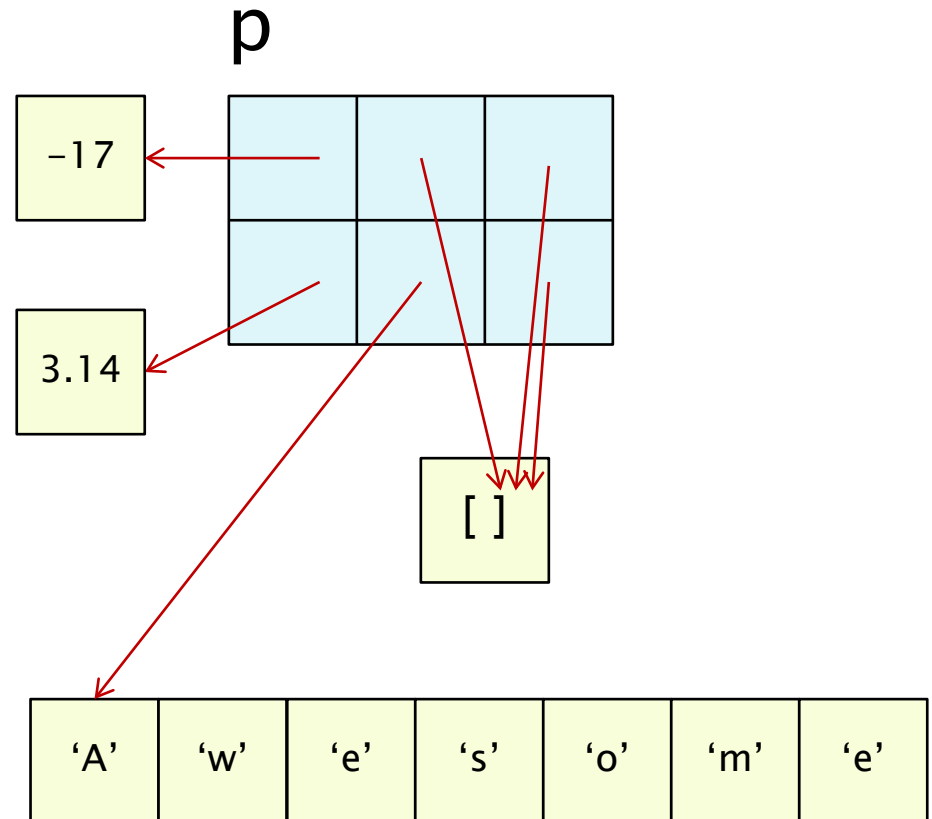
p

Cell array example

```
>> p = cell(2,3)
p =
    []         []         []
    []         []         []
>> p{2,1} = pi
p =
    []         []         []
    [3.14]    []         []
>>
```

p
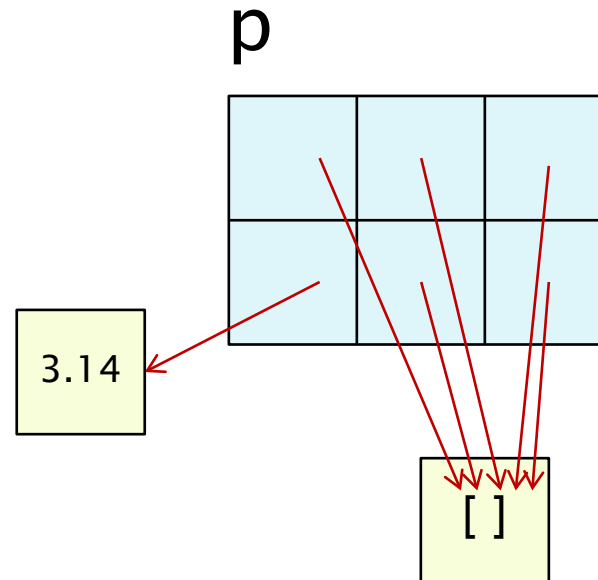
Cell array example

```
>> p = cell(2,3)
p =
     []        []          []
     []        []          []
>> p{2,1} = pi
p =
     []        []          []
    [3.14]    []          []
>> p{1,1} = int8(-17)
p =
     [-17]    []          []
    [3.14]    []          []
>>
```

p
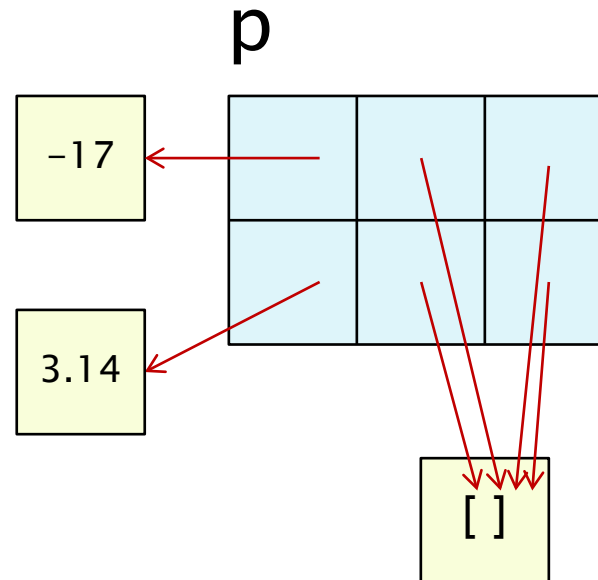
Cell array example

```
>> p = cell(2,3)
p =
    []        []        []
    []        []        []
>> p{2,1} = pi
p =
    []        []        []
    [3.14]    []        []
>> p{1,1} = -17
p =
    [-17]     []        []
    [3.14]    []        []
>> p{2,2} = 'Awesome'
P =
    [-17]     []        []
    [3.14]  'Awesome'[]
```
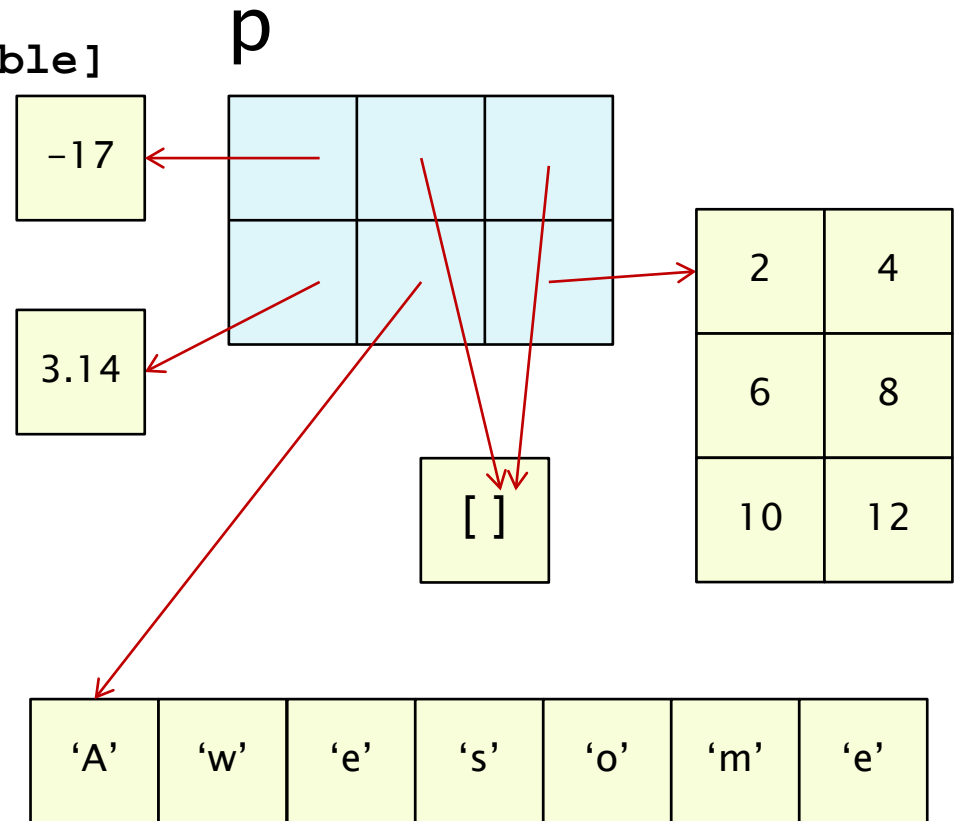
p

Cell array example

```
>> p{2,3} = [2 4; 6 8; 10 12]
P =
     [-17]   []          []
    [3.14] 'Awesome'[3x2 double]

>>
```

Cell array example

```
>> p{2,3} = [2 4; 6 8; 10 12]
P =
      [-17]    []          []
    [3.14] 'Awesome'[3x2 double]

>> p{1,3} = sum(p{2,3});
>>
```
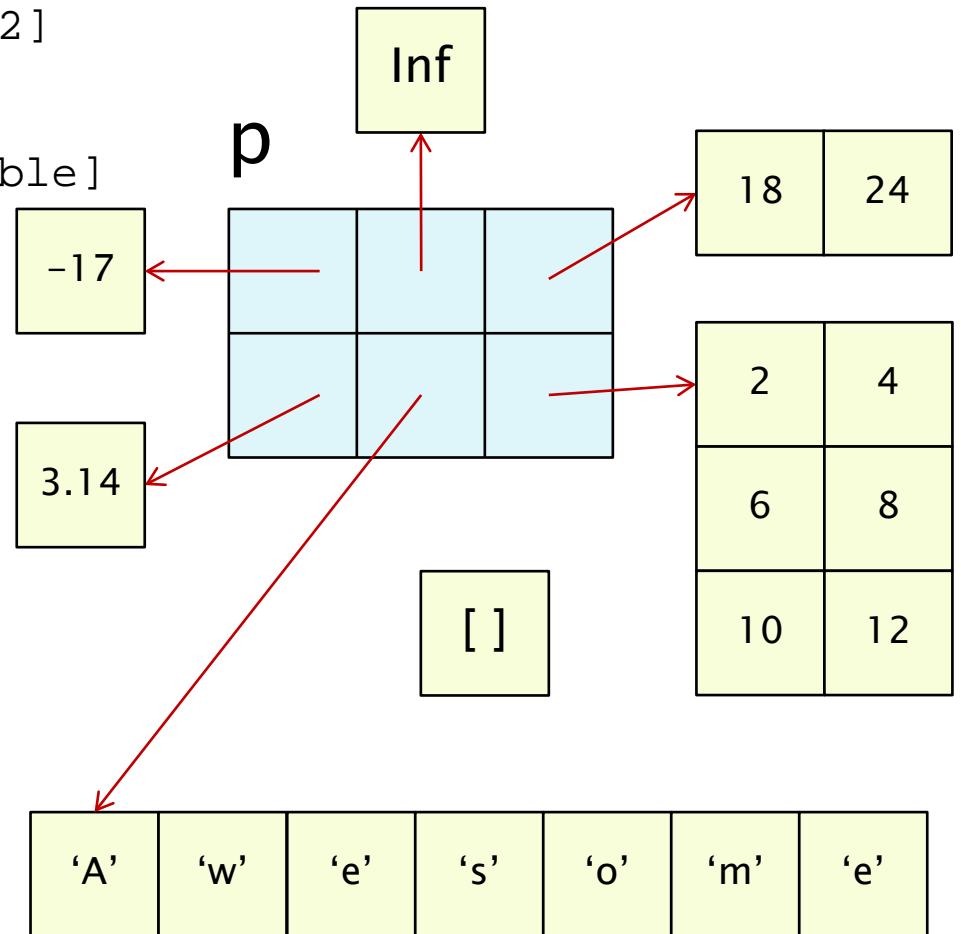
p

18	24

-17

3.14

2	4
6	8
10	12

[ ]

'A'	'w'	'e'	's'	'o'	'm'	'e'

Cell array example

```
>> p{2,3} = [2 4; 6 8; 10 12]
P =
     [-17]   []          []
    [3.14] 'Awesome'[3x2 double]

>> p{1,3} = sum(p{2,3});
>> P{1,2} = 1/0;
>>
```

p

Inf

-17

3.14

[ ]

18	24

2	4
6	8
10	12

'A'	'w'	'e'	's'	'o'	'm'	'e'

Cell array example

```
>> p{2,3} = [2 4; 6 8; 10 12]
P =
     [-17]    []           []
    [3.14] 'Awesome'[3x2 double]

>> p{1,3} = sum(p{2,3});
>> P{1,2} = 1/0;
>> class(p)
ans =
       cell
>>
```

Cell array example

```
>> p{2,3} = [2 4; 6 8; 10 12]
P =
    [-17]   []          []
    [3.14] 'Awesome'[3x2 double]

>> p{1,3} = sum(p{2,3});
>> P{1,2} = 1/0;
>> class(p)
ans =
        cell
>> class(p{1,2})
ans =
        double
>>
```

Cell array example

```
>> p{2,3} = [2 4; 6 8; 10 12]
P =
     [-17]   []          []
     [3.14] 'Awesome'[3x2 double]

>> p{1,3} = sum(p{2,3});
>> P{1,2} = 1/0;
>> class(p)
ans =
        cell
>> class(p{1,2})
ans =
        double
>> class(p(1,2))
ans =
        cell
>>
```

Cell array example

```
>> p{2,3}(3,2)
ans =
      12
>>
```

p

Inf

18 | 24

−17

2 | 4
6 | 8
10 | 12

3.14

[ ]

'A' | 'w' | 'e' | 's' | 'o' | 'm' | 'e'
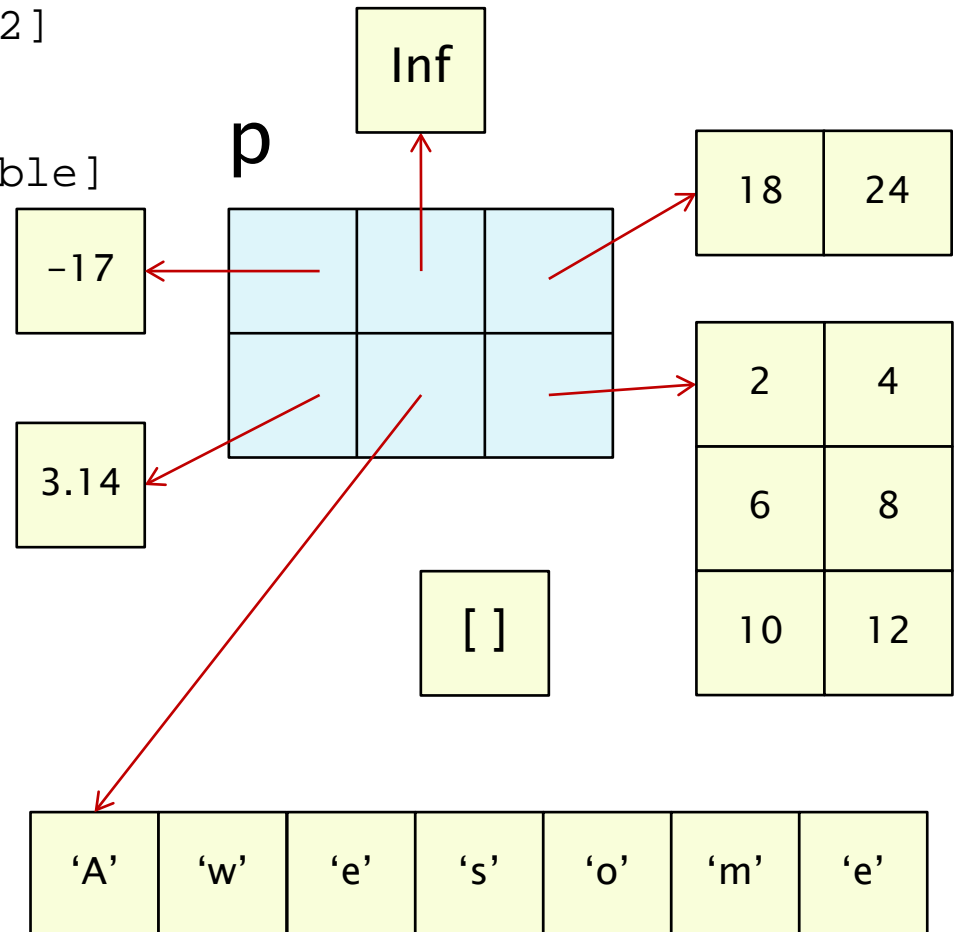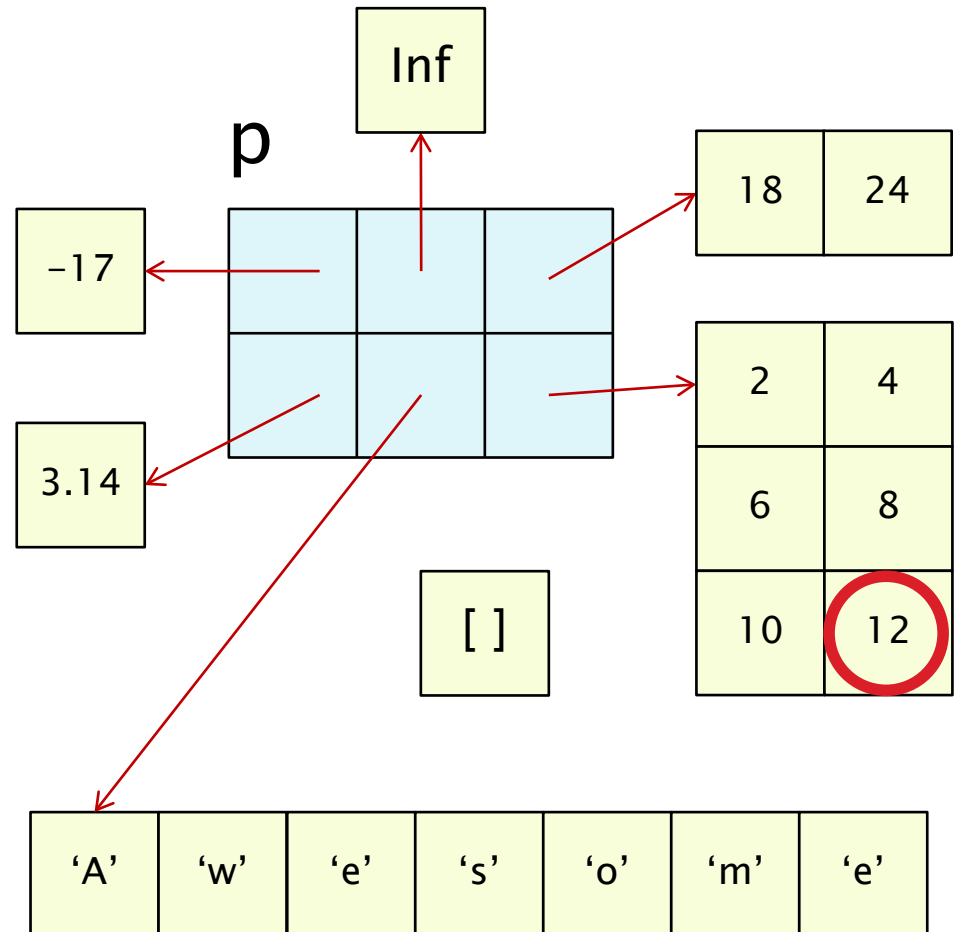
Cell functions

FUNCTION	DESCRIPTION
cell	create an array of type cell
celldisp	show all the objects pointed at by a cell array
cellfun	apply a function to all the objects pointed at by a cell array
cellplot	show a graphical depiction of the contents of a cell array
cell2struct	convert a cell array into a struct array
deal	copy a value into output arguments
iscell	returns true if argument is of type cell
num2cell	convert a numeric array into a cell array