Chapter 7

Arrays

Overview

- Arrays
- Processing Array Elements
- Multidimensional arrays
- Command Line Parameter Passing

Lesson: Arrays

- Simple Arrays
- Associative Arrays

Simple Arrays

- Array is multi-valued variables
- Every value is accessed by using index
- Index is string

```
arr[1]= "this"
arr[2]= "is"
arr[3]= "just"
arr[4]= "like"
arr[5]= "POSIX"
```

	1	2	3	4	5
arr	this	is	just	like	POSIX

```
    {my_array[NR] = $0}
END {for (i=NR; i>0; i--) print my_array[i]}
```

Associative Arrays

- As index is string, array can be used like hash table
  - Slower to process (no simple way to iterate over array)

```
sales_000["nov"] = 1100.2
sales_000["dec"] = 1225.6
sales_000["jan"] = 1254.8
sales_000["feb"] = 1305.4
sales_000["mar"] = 1428.2
```

	nov	dec	jan	feb	mar
sales_000	1100.2	1225.6	1254.8	1305.4	1428.2

```
awk 'BEGIN {tot_sales = 0; }
    { sales_000[$1] = $2 ; tot_sales += $2}
END { printf "%s%6.2f\n", "Total Sales:", tot_sales
  print "\n  Monthly sales figures"
  for (month in sales_000)
    print month,"\t",sales_000[month]}' sales_data.txt
```

Lesson: Processing Array Elements

- `for` Statement
- The Array `in` Operator
- Deleting Array Elements
- The `split` Function

Processing Arrays — `for` Loop

- Use numeric for loop

```
for (i=1; i<=NR; i++) print my_array[i]
```

- Loop over the elements of the array
- Syntax:

```
for (variable in array)
  action
```

- Example:

```
for (month in sales_000)
    print month, sales_000[month]
```

The Array `in` Operator

- You can test, if specified index exist in array.
- Syntax:

```
var in array
```

- Example:

```
if ("312" in area_code_count)
  print "312 area code found"
```

- Can't be used for testing indexed value.

```
for (varx in area_code_count)
  if (area_code_count[varx] >= 100)
    print varx, area_code_count[varx]
```

Deleting Array Elements

- Syntax:

```
delete array[index]
```

- Example:

```
BEGIN { array[1]="one";   array[2]="two"
        array[3]="three"; array[4]="four"

for (var in array) printf("%s # ",array[var])
print ""

array[3]=""
for (var in array) printf("%s # ",array[var])
print ""

delete array[3]
for (var in array) printf("%s # ",array[var])
print "" }
```

The `split` Function

- You can use split built-in function to convert any string to array.
- Syntax:

```
split(string, array[, separator])
```

- If *separator* is not specified, then `FS` is used

```
#!/bin/awk -f
# will split input lines based on the ":"
  { split($0, fields, ":")
    for (var in fields) printf("%s # ", fields[var])
    print ""
  }
```

Lesson: Multidimensional arrays

- There is no support for multidimensional arrays
- But this can be simulated by constructing index string

```
arr[1,1]= "this" ;  arr[2,1]= "row2"
arr[1,2]= "is" ;    arr[2,2]= "data"
arr[1,3]= "not" ;   arr[2,3]= "is"
arr[1,4]= "like" ;  arr[2,4]= "now"
arr[1,5]= "POSIX" ; arr[2,5]= "set"
```

arr	1	2	3	4	5
1	this	is	not	like	POSIX
2	row2	data	is	now	set

```
multi[147, 37, 275, 18, 43, 2]
```

Lesson: Command Line Parameter Passing

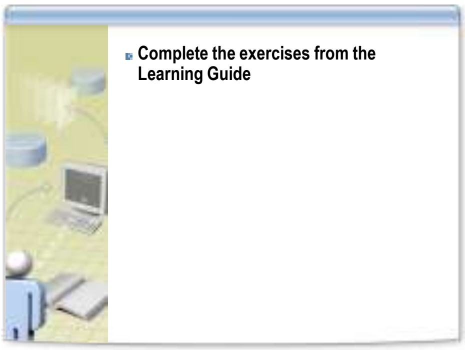
- Built-in array `ARGV` will contain command line arguments (after processing options)
- Variable `ARGC` has count of `ARGV` elements

```
$ awk -f filex abc NR=1 xyz
```

Variable	Value
ARGC	4
ARGV[3]	xyz
ARGV[2]	NR=1
ARGV[1]	abc
ARGV[0]	awk

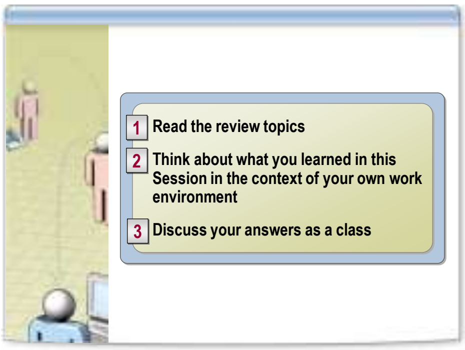
```
BEGIN { printf "A=%d, B=%d\n", A, B
  for (i = 0; i < ARGC; i++)
    printf "\tARGV[%d] = %s\n", i, ARGV[i] }
END { printf "A=%d, B=%d\n", A, B }
```

Review Exercises



- Complete the exercises from the Learning Guide

Topics for Review



1. Read the review topics
2. Think about what you learned in this Session in the context of your own work environment
3. Discuss your answers as a class