functions

Defining functions enable you to execute the same code snippet in a shell script repeatedly without having to write the entire code again. You first define a function and then call (execute) the function in the main program as shown below.

```
$ cat funcs
function function1
{
echo "enter your name : "
read a
echo "Hello, $a... Good to see you"
}

func2()
{
echo "enter your name : "
read a
echo "Please visit us again $a !!!"
}

function1

func2
```

```
$ ksh funcs

enter your name :

one

Hello, one... Good to see you

enter your name :

two

Please visit us again two !!!
```

Exit status functions / Return value functions – are functions that return an output when called (executed) much similar to the “$?” shell variable
$ cat funcsion

```
func1()

{

number=$1

if [ "$number" -lt 5 ]

then

echo "$number is less than 5..." > /dev/null

echo 0

else

echo "$number is not less than 5..." > /dev/null

echo 1

fi

}


func1 4

func1 5

func1 6
```

```
$ ksh funcsion
0
1
1
```

Functions as comments

```
$ cat one
echo "this is 1"
echo "this is 2"
func()
{
echo "this is 3"
echo "this is 4"
echo "this is 5"
}
$ ksh one
this is 1
this is 2
```

Awk

Extracting fields using awk, awk is used to extract a field(column) from a given output in tabled format,awk can also extract using delimiters as shown below. By default awk takes the white space as the delimiter.

-F	: use a specified field seperator(delimiter) to extract field(s)
-v	: initialize a variable which can be used inside the BEGIN and END block

-f : read the awk program source from a program-file instead of the first command line argument

```
$ cat rows
one     two     three
four    five    six
seven   eight   nine
ten     eleven  twelve

$ awk '{print $1}' rows
one
four
seven
ten

$ awk '{print $3}' rows
three
six
nine
twelve
```

using awk with hyphen(-) as the delimiter.

```
$ cat rows

one-two-three

four-five-six

seven-eight-nine

ten-eleven-twelve


$ awk -F "-" '{print $2}' rows

two

five

eight

eleven
```

exercise : use awk to extract the filenames from the output of ls –ltr command, use the pipe (|) operator to pass the output of ls command to awk - the file name field in the ls –ltr command is the 9 field.

```
$ cat rows1

one two three

four five six

seven eight nine

ten eleven twelve

$ awk -v var=Iam '{print var" " $1}' rows1

Iam one

Iam four

Iam seven

Iam ten
```

```
$ awk -v var='this is' '{print var" " $1}' rows1
this is one
this is four
this is seven
this is ten

$ cat rows1
one two three
four five six
seven eight nine
ten eleven twelve

$ awk -v var="this is" '{print var" " $1}' rows1
this is one
this is four
this is seven
this is ten

$ awk –f comamnd_file input_file

$ cat cmd.awk
{print $1}

$ awk –f cmd.awk rows1
one
four
seven
ten
```

```
$ cat cmd.awk
{print var1" "$1}

$ awk -v var1="this is" -f cmd.awk rows1
this is one
this is four
this is seven
this is ten

$ cat cmd1.awk
{print var1" "$1" "var2" "$2}

$ awk -v var1="this is" -v var2="and" -f cmd1.awk -F "-" rows
this is one and two
this is four and five
this is seven and eight
this is ten and eleven
```

Cut

Cut is another unix binary/utility that lets you extract fields from a given output, cut does not have any default delimiting case behavior

-d	: use a specified delimiter to extract fields
-f	: extract only these fields; also print any line that contains no delimiter character, unless the -s option is specified
-s	: do not print lines not containing delimiters
-c	: extract only these characters

```
$ cat rows|cut -d"-" -f1-2

one-two

four-five

seven-eight

ten-eleven


$ cat rows|cut -d"-" -f1,3

one-three

four-six

seven-nine

ten-twelve


$ cat rows|cut -d"-" -f1-3

one-two-three

four-five-six

seven-eight-nine

ten-eleven-twelve


$ cat rows

one-two-three

four-five-six

seven-eight-nine

ten-eleven-twelve
```

```
$ cut -c 1-4 rows
one-
four
seve
ten-

$ cut -c 1,4 rows
o-
fr
se
t-



$ cat rows
one-two-three
four-five-six
seven-eight-nine
ten-eleven-twelve

$ cut -c 5-12 rows
two-thre
-five-si
n-eight-
eleven-t
```

```
$ cat rows1
one two three
four five six
seven eight nine
ten eleven twelve

$ cut -f1 rows1
one two three
four five six
seven eight nine
ten eleven twelve

$ cut -d " " -f2 rows1
two
five
eight
Eleven


$ cat rows2
one-two-three
four-five-six
seven-eight-nine
ten-eleven-twelve
one two three
four five six
seven eight nine
ten eleven twelve
```

```
$ cut -s -d "-" -f2,3 rows2
two-three
five-six
eight-nine
eleven-twelve

$ cat rows2
one-two-three
four-five-six
seven-eight-nine
ten-eleven-twelve
one two three
four five six
seven eight nine
ten eleven twelve

$ cut -s -d " " -f1,3 rows2
one three
four six
seven nine
ten twelve
```

```
$ cat rows2
one-two-three
four-five-six
seven-eight-nine
ten-eleven-twelve
one two three
four five six
seven eight nine
ten eleven twelve
```

```
$ cut -d " " -f1,3 rows2
one-two-three
four-five-six
seven-eight-nine
ten-eleven-twelve
one three
four six
seven nine
ten twelve
```

Sed

sed like awk is another language in itself, sed stands for stream editor. sed has so many applications but the most predominantly used are search and replace

number	a number in the script ensures that the patterns are changed only on that line of the file
w	‘w’ flag in the script writes a new file with the resulting output but only for the matching lines
g	replaces the pattern globally in the entire line

-e	: with this option multiple scripts for search and replace can be included
-f	: takes a command file with search and replace scripts
: or |	: using these two characters will change the syntax of the sed command to include these characters instead of slash (/) as the delimiter

```
$ sed 's/search_string/replace_string/' file_name
```

```
$ cat streaming
this is line 1
this is line 2
this is line 3
.
.
this is line n
```

```
$ sed 's/this/that/' streaming
that is line 1
that is line 2
that is line 3
.
.
that is line n
```

```
$ cat streaming
this is line 1
this is line 2
this is line 3
.
.
this is line n

$ sed '3 s/this/that/' streaming
this is line 1
this is line 2
that is line 3
.
.
this is line n

$ cat streaming
this is line 1
this is line 2
this is line 3
.
.
this is line n
```

```
$ sed '1,3 s/this/that/' streaming

that is line 1

that is line 2

that is line 3

.

.

this is line n
```

```
$ cat streaming

this is line 1

this is line 2

this is line 3

.

.

this is line n
```

```
$ sed '3,6 s/this/that/' streaming

this is line 1

this is line 2

that is line 3

.

.

that is line n
```

```
$ cat streaming
this is line 1
this is line 2
this is line 3
.
.
this is line n

$ sed 's/this/that/w file1' streaming
that is line 1
that is line 2
that is line 3
.
.
that is line n

$ cat file1
that is line 1
that is line 2
that is line 3
that is line n
```

```
$ cat streaming
this is line 1
this is line 2
this is line 3
.
.
this is line n
```

```
$ sed -e 's/this/that/' -e 's/is/was/' streaming
that was line 1
that was line 2
that was line 3
.
.
that was line n
```

```
$ cat streaming
this is /line 1
this is /line 2
this is /line 3
.
.
this is /line n
```

```
$ sed 's/\/line/phrase/' streaming
this is phrase 1
this is phrase 2
this is phrase 3
.
.
this is phrase n
```

```
$ cat streaming
this is /line 1
this is /line 2
this is /line 3
.
.
this is /line n

$ sed 's:/line:phrase:' streaming
this is phrase 1
this is phrase 2
this is phrase 3
.
.
this is phrase n
```

```
$ cat streaming

this is /line 1

this is /line 2

this is /line 3
.

.

this is /line n



$ sed 's|/line|/phrase|' streaming

this is /phrase 1

this is /phrase 2

this is /phrase 3

.

.

this is /phrase n



$ cat streaming

this is /line 1

this is /line 2

this is /line 3

.

.

this is /line n
```

```
$ cat cmd.sed
s/this/that/
$ sed -f cmd.sed streaming
that is /line 1
that is /line 2
that is /line 3
.
.
that is /line n
```

```
$ cat streaming
this is /line 1
this is /line 2
this is /line 3
.
.
this is /line n
```

```
$ cat cmd.sed
s/this/that/
s/is/was/
s/1/11/
s/2/22/
```

```
$ sed -f cmd.sed streaming

that was /line 11

that was /line 22

that was /line 3

.

.

that was /line n



$ cat streaming

this is line 1 and this is 1st line

this is line 2 and this is 2nd line

this is line 3 and this is 3rd line

.

.

this is line n and this is nth line



$ sed 's/this/that/' streaming

that is line 1 and this is 1st line

that is line 2 and this is 2nd line

that is line 3 and this is 3rd line

.

.

that is line n and this is nth line
```

```
$ cat streaming
this is line 1 and this is 1st line
this is line 2 and this is 2nd line
this is line 3 and this is 3rd line
.
.
this is line n and this is nth line
```

```
$ sed 's/this/that/g' streaming
that is line 1 and that is 1st line
that is line 2 and that is 2nd line
that is line 3 and that is 3rd line
.
.
that is line n and that is nth line
```

```
$ cat streaming
this is line 1 and this is 1st line
this is line 2 and this is 2nd line
this is line 3 and this is 3rd line
.
.
this is line n and this is nth line
```

```
$ cat cmd.sed
s/this/that/g
s/is/was/
s/1/11/g
s/2/22/


$ sed -f cmd.sed streaming
that was line 11 and that is 11st line
that was line 22 and that is 2nd line
that was line 3 and that is 3rd line
.
.
that was line n and that is nth line
```

Grep

grep stands for global regular expression and print, is used to search for patterns in a given output/file. grep extracts the horizontal lines (rows) unlike awk which prints the vertical columns (fields)

-c	: print a count of matching lines for the input file/text
-i	: ignore case, prints all the lower/upper matching patterns from a given input file/text
-n	: prefix each line of output with the line number
-v	: inverts the sense of matching, to select non-matching lines

```
$ cat regexp

There was a fisherman named Fisher

who fished for some Fish in a fissure.

Till a fish with a grin,

pulled the FISHerman in.

Now they're fishing the FIssure for Fisher.

Is FISHer now done with fISHing ?

because now the FISH need to be sold on the fiSH market



$ grep "fish" regexp

There was a fisherman named Fisher

who fished for some Fish in a fissure.

Till a fish with a grin,

Now they're fishing the FIssure for Fisher.



$ grep -c "fish" regexp

4

$ grep -n "fish" regexp

1:There was a fisherman named Fisher

2:who fished for some Fish in a fissure.

3:Till a fish with a grin,

5:Now they're fishing the FIssure for Fisher.
```

```
$ grep -i "fish" regexp
There was a fisherman named Fisher
who fished for some Fish in a fissure.
Till a fish with a grin,
pulled the FISHerman in.
Now they're fishing the FIssure for Fisher.
Is FISHer now done with fISHing ?
because now the FISH need to be sold on the fiSH market
```

```
$ grep -v "fish" regexp
pulled the FISHerman in.
Is FISHer now done with fISHing ?
because now the FISH need to be sold on the fiSH market
```

```
$ cat regexp
There was a fisherman named Fisher
who fished for some Fish in a fissure.
Till a fish with a grin,
pulled the FISHerman in.
Now they're fishing the FIssure for Fisher.
Is FISHer now done with fISHing ?
because now the FISH need to be sold on the fiSH market
```

```
$ grep -nv "fish" regexp
4:pulled the FISHerman in.
6:Is FISHer now done with fISHing ?
7:because now the FISH need to be sold on the fiSH market
```

```
$ grep -cn "fish" regexp

4
```

```
$ grep -ci "fish" regexp

7
```

Streaming

Writing to files in a loop can be achieved with “>” and “>>”, but how do you read from a file and manipulate the line for further processing ?

Use the “while construct” by taking the output from “cat” command

```
$ cat streaming

this is line 1

this is line 2

this is line 3

.

.

this is line n
```

```
$ cat streaming|while read line
do
     echo "$line"
done
this is line 1
this is line 2
this is line 3
.
.
this is line n
```