Functions

- Clearest if defined with function keyword, note that parameters do not get listed, accessed as $1, $2, etc
- Local variables defined with keword local, e.g.

```
function foo () {
    local x=$1 # store params in local vars
    local y=$2
    echo “you passed $x, $y”
}

foo hello 23  # sample call
```

Return values

- Bash does have a return statement (e.g. return $x), but it is actually meant to return status code, i.e. a single byte representing integer value in 0-255
- If you return a larger value, only last byte actually gets returned (and will be treated as an int 0-255)
- To “return” text, or larger values, or more than one value, the technique used is to have the function print out the value(s) and then have the calling routine capture the function output

Capturing function output

- function prints square of its parameter, the caller captures output using $( ) then displays the captured text

```
function square () {
    local tmp=$1
    (( tmp = tmp * tmp ))
    echo “$tmp”
}
result=$(square 205)
echo “square(205) is $result”
```

Special variables

- As mentioned, $1, $2, etc give the function params
- $0 gives the name of the function (like argv[0] in C)
- $# gives you the number of parameters passed
- $@ gives you all the arguments as an array (of strings)
- $* gives you all the arguments but as a single big string (as they would appear if a user had typed the call)
- $? gives you the exit code of the most recent command

Example: iterating through params

- Iterating through parameters as an array

```
function f() {
    local i=0
    for arg in “$@”; do
        echo “arg $i is $arg”
        (( i++ ))
    done
}

f 10 “20 30” 40
# note here arg 0 is “10”, arg 1 is “20 30”, arg 2 is “40”
```

Error checking parameters

- Functions should check they were passed the correct number and correct types of parameters before using them
- Checking the correct number is easy (use $#)
- Checking the type is trickier since they’re all getting passed as text
- Often we see if the string that was passed for a parameter fits the right pattern for what we want (e.g. if we’re expecting an integer, is the text made up of all digits?): we’ll revisit this with regular expressions and pattern matching soon

Calling functions from command line

- Suppose we define a useful function, f, in a file, mystuff.sh, and want to call it from the command line
- We can use the “source” command to apply the definitions in the file (i.e. “source mystuff.sh”) then anytime after that we can call function f from the command line
- Need to run source again if we edit function f
- If we put the “source mystuff.sh” in our .bashrc file in our home directory then it will automatically get source’d every time we login (though we want to make very sure it’s not buggy first!)