BASH Strings

SCRIPTING ESSENTIALS

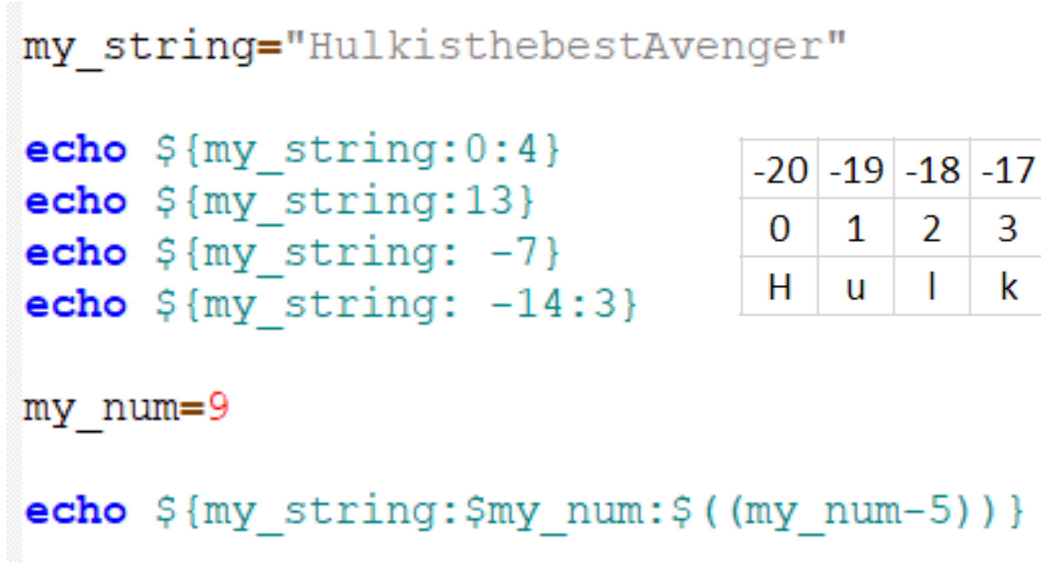
DR. BURKMAN

BASH Strings

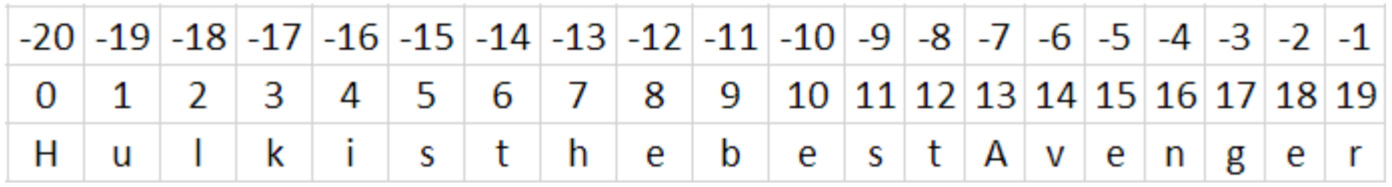
Escaping double quotes

- Just like we know. \”

Slicing Strings



```
my_string="HulkisthebestAvenger"

echo ${my_string:0:4}
echo ${my_string:13}
echo ${my_string: -7}
echo ${my_string: -14:3}

my_num=9

echo ${my_string:$my_num:$((my_num-5))}
```

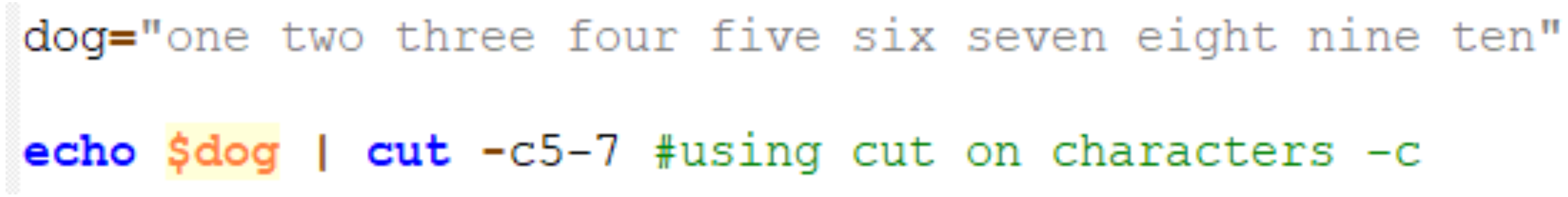


-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
H	u	l	k	i	s	t	h	e	b	e	s	t	A	v	e	n	g	e	r

Cut starts with 1, not 0

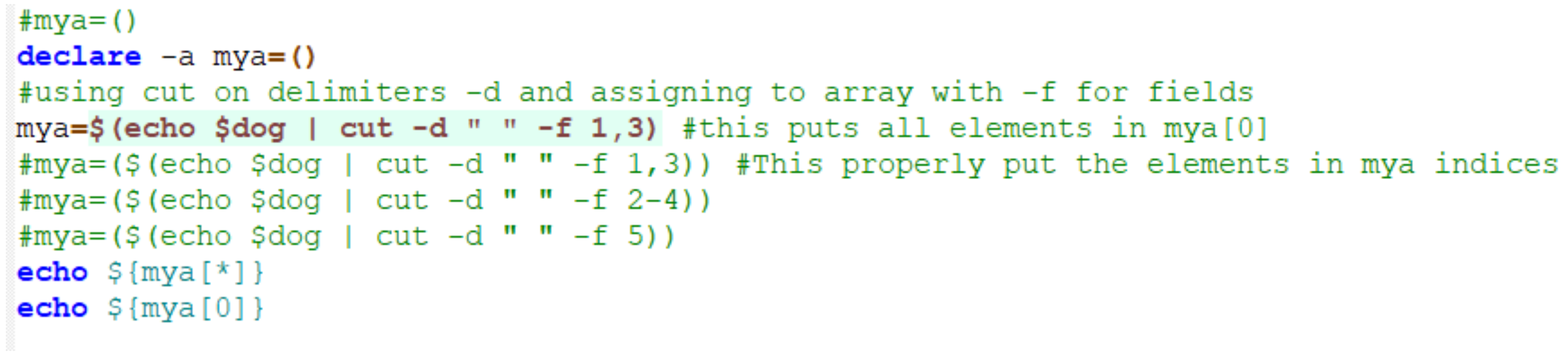
Adding items to array
my_array+=("$cat ")

Use cut to slice up a string with -c, or use cut with -d to split a string on a delimiter. If you use –d also use –f to indicate the field or fields that you want.



```
dog="one two three four five six seven eight nine ten"

echo $dog | cut -c5-7 #using cut on characters -c
```

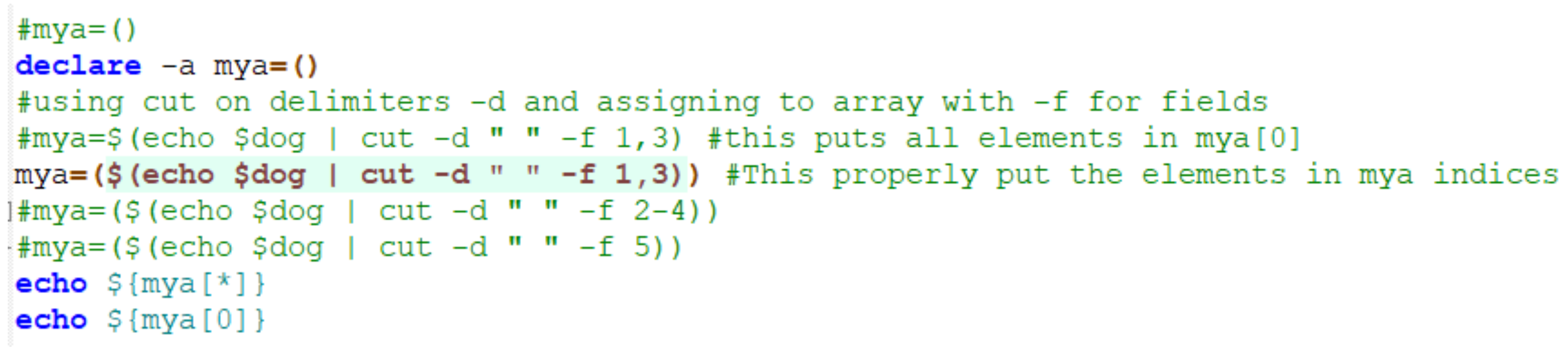
This makes $mya an array with everything at index 0. Declaring it an an array first will make no difference.



```
#mya=()
declare -a mya=()
#using cut on delimiters -d and assigning to array with -f for fields
mya=$(echo $dog | cut -d " " -f 1,3) #this puts all elements in mya[0]
#mya=($(echo $dog | cut -d " " -f 1,3)) #This properly put the elements in mya indices
#mya=($(echo $dog | cut -d " " -f 2-4))
#mya=($(echo $dog | cut -d " " -f 5))
echo ${mya[*]}
echo ${mya[0]}
```

You *must* cut on a space to reliably create an array from a string

This makes $mya an array with elements in multiple indexes, regardless of whether you first initialize it or not.

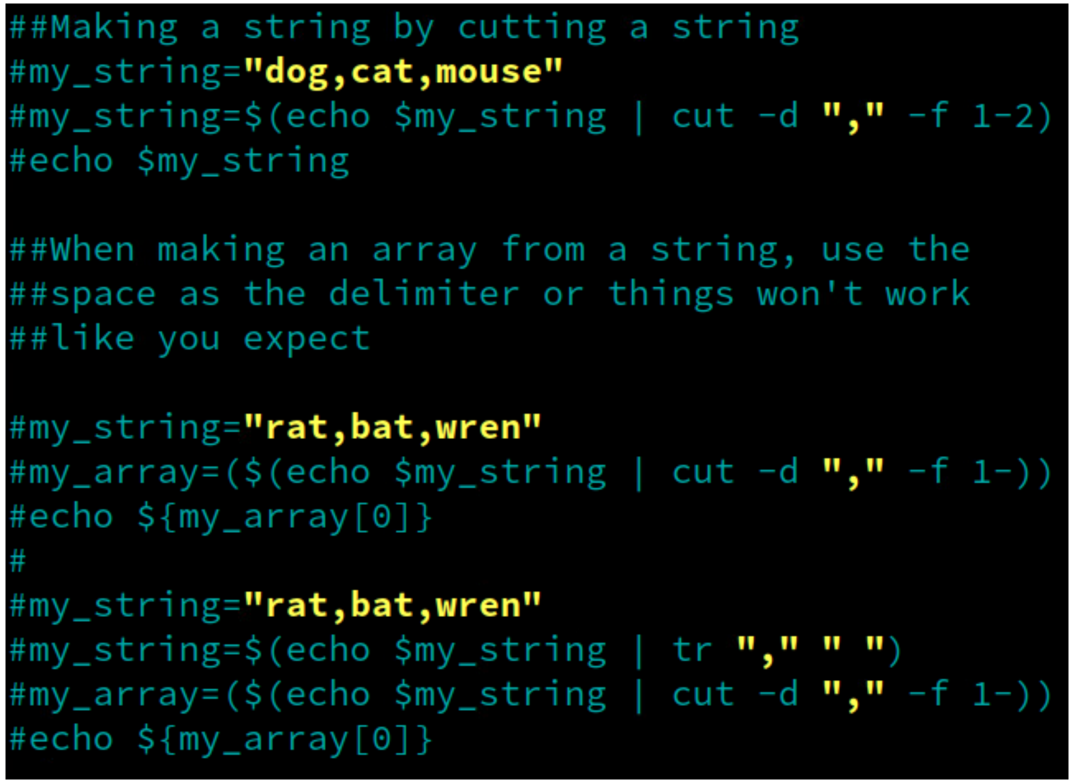


```
#mya=()
declare -a mya=()
#using cut on delimiters -d and assigning to array with -f for fields
#mya=$(echo $dog | cut -d " " -f 1,3) #this puts all elements in mya[0]
mya=($(echo $dog | cut -d " " -f 1,3)) #This properly put the elements in mya indices
#mya=($(echo $dog | cut -d " " -f 2-4))
#mya=($(echo $dog | cut -d " " -f 5))
echo ${mya[*]}
echo ${mya[0]}
```

You can cut strings to make new strings, even with different delimiters

You can cut strings on delimiters to make new strings.

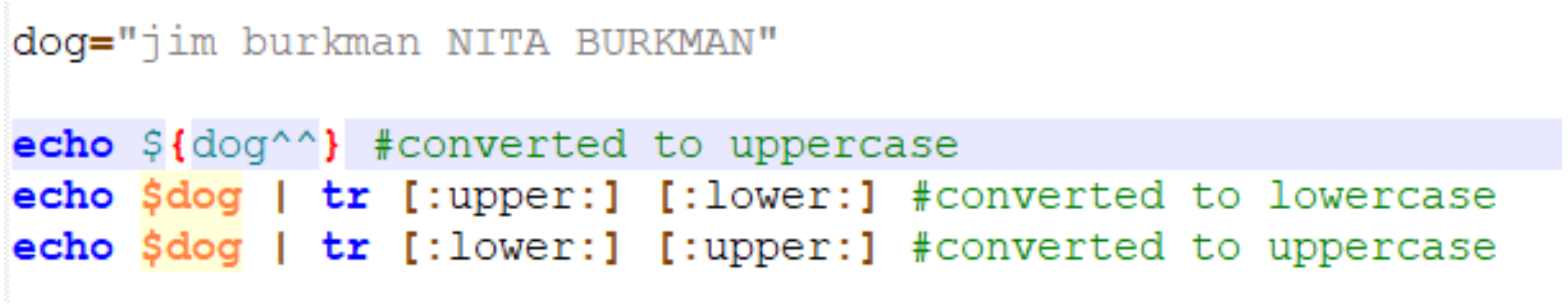
But when cutting a string to make an array be sure to only use the space " " delimiter. Otherwise your array won't be correct.



```
##Making a string by cutting a string
#my_string="dog,cat,mouse"
#my_string=$(echo $my_string | cut -d "," -f 1-2)
#echo $my_string

##When making an array from a string, use the
##space as the delimiter or things won't work
##like you expect

#my_string="rat,bat,wren"
#my_array=($(echo $my_string | cut -d "," -f 1-))
#echo ${my_array[0]}
#
#my_string="rat,bat,wren"
#my_string=$(echo $my_string | tr "," " ")
#my_array=($(echo $my_string | cut -d "," -f 1-))
#echo ${my_array[0]}
```

Converting case

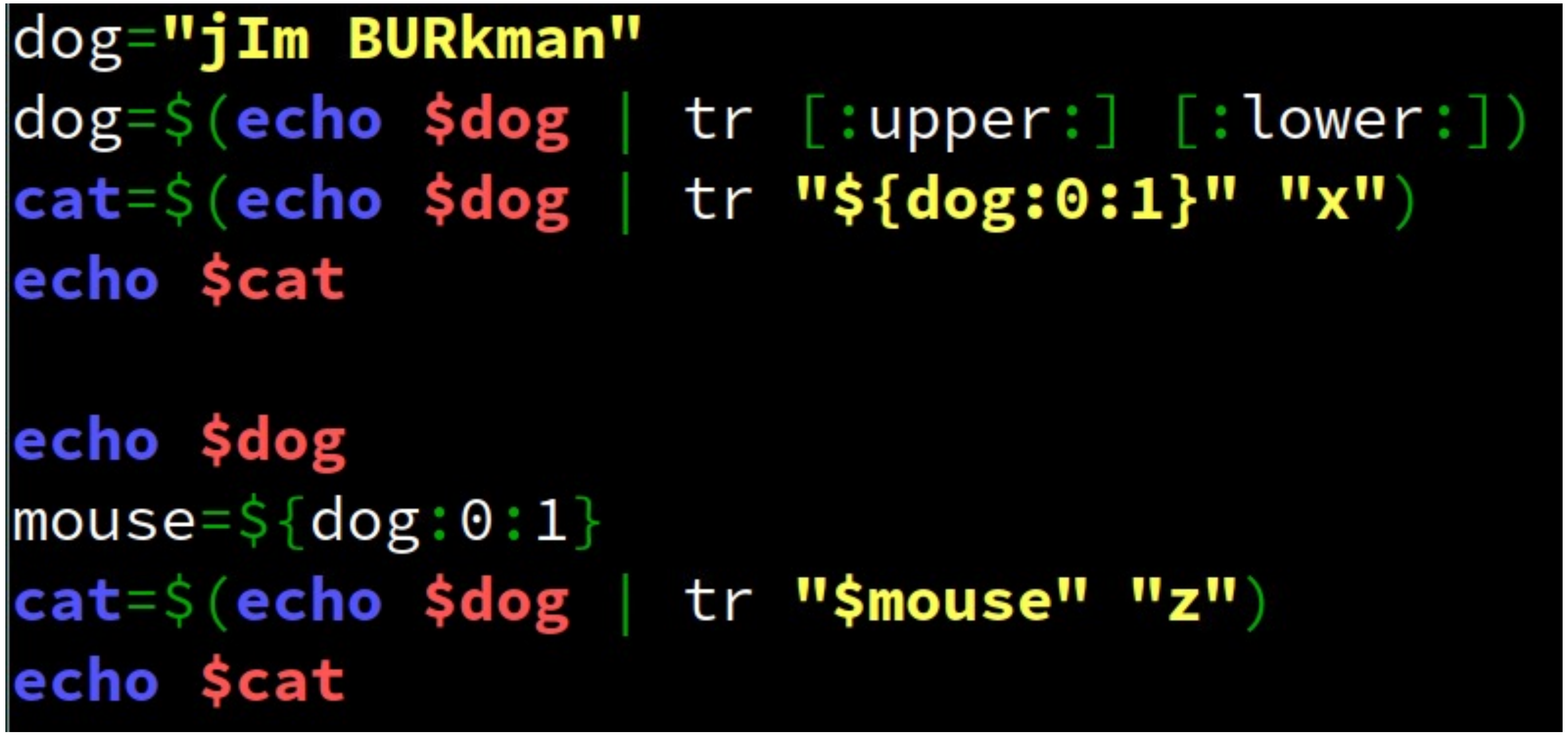
Upper to lower and lower to upper is easy with tr. There is no mechanism for title case.



```
dog="jim burkman NITA BURKMAN"

echo ${dog^^} #converted to uppercase
echo $dog | tr [:upper:] [:lower:] #converted to lowercase
echo $dog | tr [:lower:] [:upper:] #converted to uppercase
```

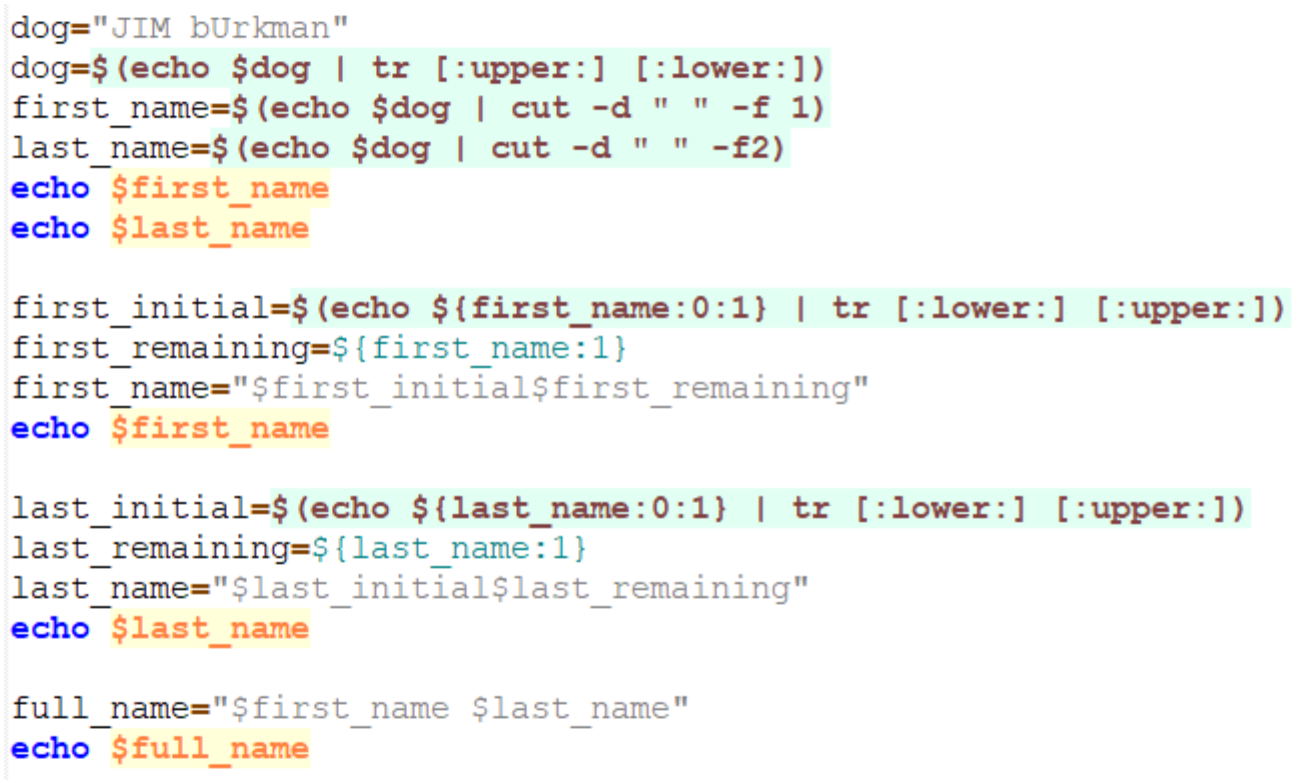
Converting case

Pay attention to the formatting if you transform with a variable:



```
dog="jIm BURkman"
dog=$(echo $dog | tr [:upper:] [:lower:])
cat=$(echo $dog | tr "${dog:0:1}" "x")
echo $cat

echo $dog
mouse=${dog:0:1}
cat=$(echo $dog | tr "$mouse" "z")
echo $cat
```

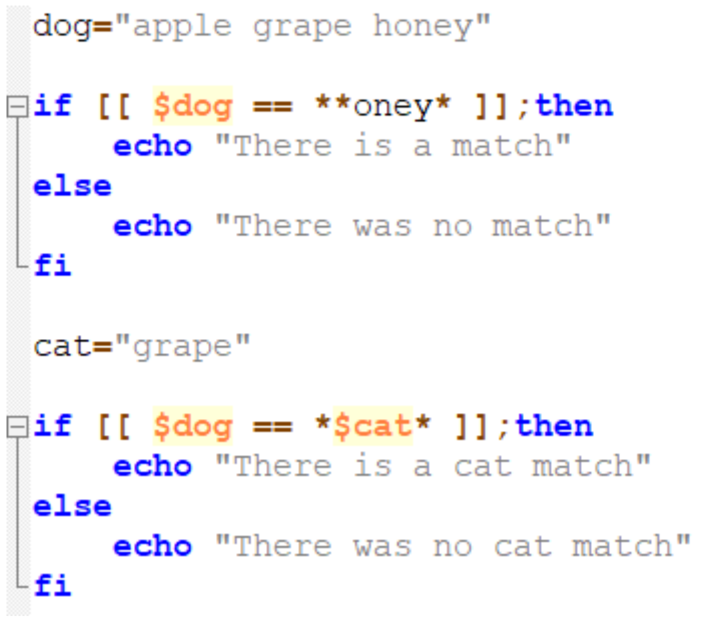
Title case a name



```
dog="JIM bUrkman"
dog=$(echo $dog | tr [:upper:] [:lower:])
first_name=$(echo $dog | cut -d " " -f 1)
last_name=$(echo $dog | cut -d " " -f2)
echo $first_name
echo $last_name

first_initial=$(echo ${first_name:0:1} | tr [:lower:] [:upper:])
first_remaining=${first_name:1}
first_name="$first_initial$first_remaining"
echo $first_name

last_initial=$(echo ${last_name:0:1} | tr [:lower:] [:upper:])
last_remaining=${last_name:1}
last_name="$last_initial$last_remaining"
echo $last_name

full_name="$first_name $last_name"
echo $full_name
```

Matching a value to a string

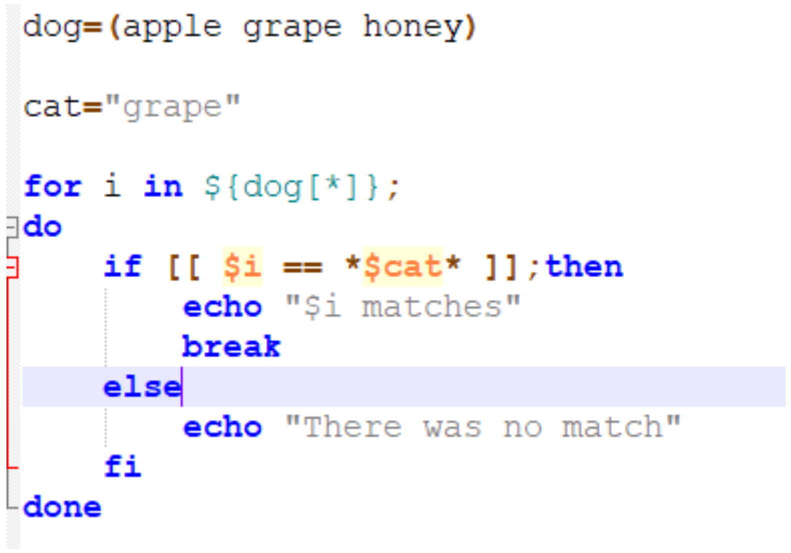
There are no methods like we've had. So, this:



```
dog="apple grape honey"

if [[ $dog == **oney* ]];then
    echo "There is a match"
else
    echo "There was no match"
fi

cat="grape"

if [[ $dog == *$cat* ]];then
    echo "There is a cat match"
else
    echo "There was no cat match"
fi
```

Matching a value to an array

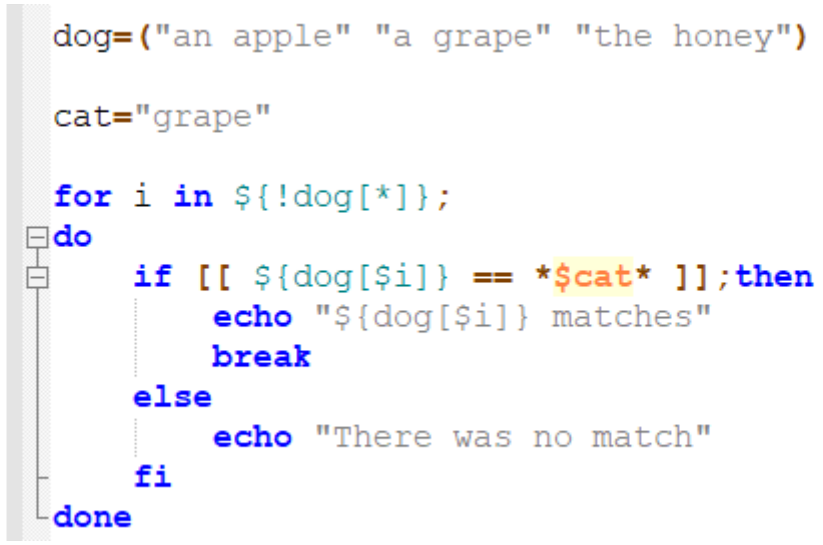
If your elements don’t have spaces you can do this:



```
dog=(apple grape honey)

cat="grape"

for i in ${dog[*]};
do
    if [[ $i == *$cat* ]];then
        echo "$i matches"
        break
    else
        echo "There was no match"
    fi
done
```

Matching a value to an array

If your element has spaces you must do this:

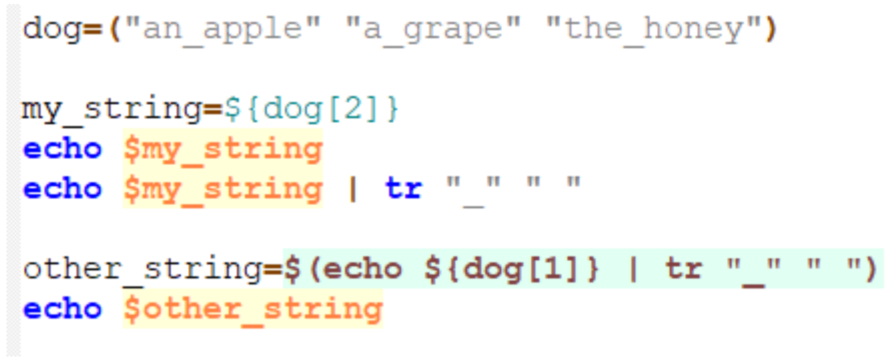


```
dog=("an apple" "a grape" "the honey")

cat="grape"

for i in ${!dog[*]};
do
    if [[ ${dog[$i]} == *$cat* ]];then
        echo "${dog[$i]} matches"
        break
    else
        echo "There was no match"
    fi
done
```

Note: Putting elements with spaces into an array is a huge hassle. Use _ instead of a space and just tr back and forth as needed for assigning variables and for printing

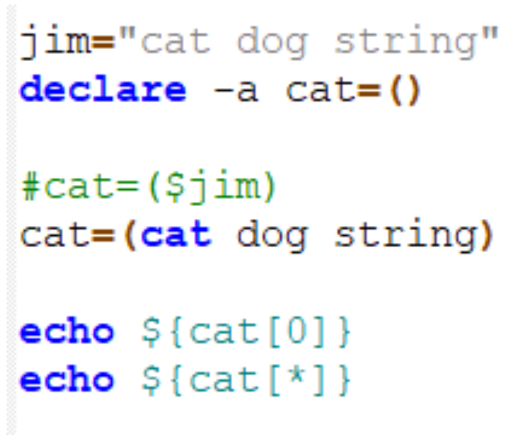
Array elements with spaces

Note: Putting elements with spaces into an array is a huge hassle. Use _ instead of a space and just tr back and forth as needed for assigning variables and for printing

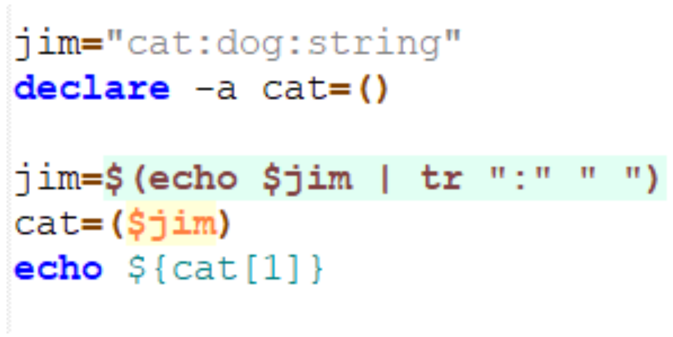


```
dog=("an_apple" "a_grape" "the_honey")

my_string=${dog[2]}
echo $my_string
echo $my_string | tr "_" " "

other_string=$(echo ${dog[1]} | tr "_" " ")
echo $other_string
```

Add() around anything being added to array



```
jim="cat dog string"
declare -a cat=()

#cat=($jim)
cat=(cat dog string)

echo ${cat[0]}
echo ${cat[*]}
```

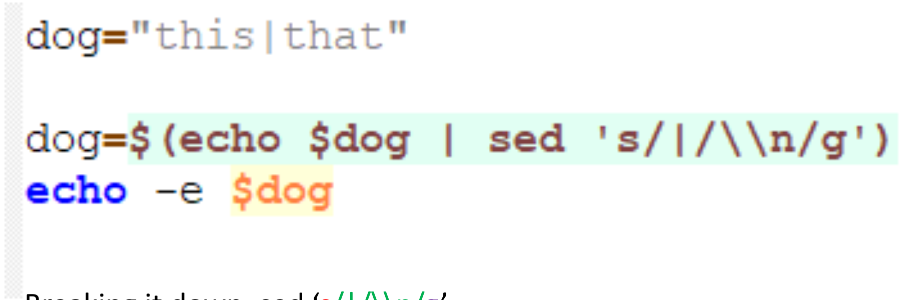


```
jim="cat:dog:string"
declare -a cat=()

jim=$(echo $jim | tr ":" " ")
cat=($jim)
echo ${cat[1]}
```

Sed for newlines

Echo –e interprets backslash escapes. If you want a new line in a string put \n in there, but you'll have to use echo -e when printing to get it to work.

tr is byte substitution so it cannot substitute new lines (that's two bytes). We have to use sed.



```
dog="this|that"

dog=$(echo $dog | sed 's/|/\\n/g')
echo -e $dog
```

Breaking it down: sed 's/|/\\n/g'

s for substitute
/x/y/ replace x with y (\\n have to escape the backspace for sed)
g global - do this substitution everywhere in the line