Control structures

```
# ---------- CONTROL STRUCTURE ----------

# ---------- CONDITIONALS ----------

# IF ELIF ELSE
    # predicate is surrounded by [] square brackets

peepee = 10
if [$peepee -eq 10]; then
    echo "peepee is equal to 10!"
elif [$peepee -ne 10]; then
    echo "peepee is not equal to 10!"
else
    echo "logically speaking this should not be possible"
fi

# CASE IN *
    # case in => declares a case block similar to switch case statements in other languages
    # ) => appends every case statement's specified value
    # ;; => append every case statement's logic, acting as the equivalent of a break statement to ensure logic breaks out after a given case is hit
    # * => specifies the default case should all other conditions fall through

car = 10
case $car in
    "BMW" )
        echo "your kar is a BMW";;
    "Toyota" )
        echo "aigh bet";;
    "Honda" )
        echo "your car is a Hoonda";;
    "Toyota" )
        echo "car ni na";;
    * )
        echo "this is the default case";;
esac

# ---------- LOOPS ----------
    # break => breaks out of the current loop
    # condition => skips to the next iteration of the given loop

# WHILE DO LOOPS
    # loop condition specified within [] square brackets

number = 1
while [ $number -lt 10 ]
do
    echo "$number"
    number = $((number+1))
done

# UNTIL LOOPS
    # loop condition specified within [] square brackets
    # equivalent of a while false loop in Bash

number = 1
until [ $number -ge 10 ]
do
    echo $number
    number $((number+1))
done

# FOR LOOPS
    # allows for rudimentary for loops with an explicit start, step and end

for (( i=0; i<5; i++ ))
do
    echo $i
done

# FOR IN LOOPS
    # allows for iteration over a specified range
    # .. => allows for creation of implicit ranges with the syntax => {START..END..STEP}

for i in 1 2 3 4 5
do
    echo $i
done # prints 1\n2\n3\n4\n5\n to the stdout

for i in {0..20..2}
do
    echo $i
done # prints 0\n2\n4\n6\n8\n10\n12\n14\n16\n18\n20\n
```

Data structures