



# Lab #6

**More Loops, Functions, & GitHub**

BINF 2111, Fall 2023





# IMPORTANT!



**Your midterm is in 2 weeks!**

I will send out a **poll** to see where everyone's current comprehension level resides.

I will also post a **study guide** and hold a **review session** (maybe multiple) before the midterm.

# Terminology



## Increment

The process of increasing a numeric value by another value, usually by 1 with `num++`



## Function

Self contained modules of code that accomplish a specific task, usually taking in data, processing it, and returning a result



## Argument

A special kind of variable used in a function to refer to one of the pieces of data provided as input to the function

# Loops - while

- Do something **while** a condition is **true**

```
while [[ condition ]]; do
    commands
done
```
- Can use the same conditions as if statements
- Opposite of until loops
- Common Uses
  - Using a counter (increment/decrement)
  - Reading files line by line
  - Infinite loops
  - Writing information into a file



# Loops - while

- Using a counter (increment with ++ and decrement with --)
- While a is less than 10, print a and increment by 1

Initialize `a`  
to 0

```
a=0
while [[ $a -lt 10 ]]
do
    echo $a
    ((a++))
done
```

Use a conditional to  
check if `a` is less  
than 10


Print `a`

Increment `a`  
by 1 with ++



# Loops - while

- Reading files line by line
- Read each line in example2.fasta, find the character count of that line, add it to the sum, and print it all out.



```
while read line
do
    chars=$(echo $line | wc -c)
    sum=$((sum+chars))
    echo The sum of all the characters in the file is $sum
done < example2.fasta
```

Use `read` to read files in

`line` is the variable for each line in the file

Count the characters in each line with `wc -c`

Add the character amount to `sum` and set it equal to `sum`

Print it out

Use `example2.fasta` as input

# Loops - while

- **Infinite loops**
- Press Ctrl+C to get out of the loop/end the script

```
while :  
do  
    echo "An Infinite loop"  
done
```

Use `:` to make the loop infinite by not checking for a condition

Print a statement that will repeat forever



# Loops - while

- Writing information into a file
- Press Enter, Ctrl-D when you are done typing the file contents

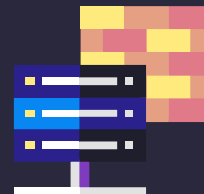
Print a prompt statement

Use `read` to read in the user's input and store it as `filename`

```
echo -n "Enter the filename to create: "  
read filename  
  
while read line  
do  
    echo $line >> $filename  
done
```

Use `read` to read in the user's input for the file's contents and store each line as `line`

Print the line into the file





# Loops - until

- Do something **until** a condition is **false**  

```
until [[ condition ]]; do  
    commands  
done
```
- Can use the same conditions as if statements
- Opposite of while loops
- Common Uses
  - Using a counter (increment/decrement)
  - Reading files line by line
  - Infinite loops
  - Writing information into a file



# Loops - until

- Using a counter (increment/decrement)
- Until a is NOT less than 10, print a and add 1

Initialize `a`  
to 0

```
a=0
until [[ ! $a -lt 10 ]]
do
    echo $a
    ((a++))
done
```

Use a conditional to  
check if `a` is NOT less  
than 10

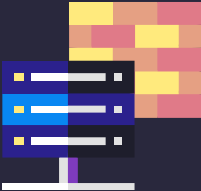
Print `a`

Increment `a`  
by 1 with `++`



# Loops - until

- Reading files line by line
- Read each line in example2.fasta, find the character count of that line, add it to the sum, and print it all out.
- Does the same thing as the while loop shown before



```
until ! read line
do
    chars=$(echo $line | wc -c)
    (($sum+=chars))
    echo The sum of all the characters in the file is $sum
done < example2.fasta
```

Use `! read` to read files in

`line` is the variable for each line in the file

Count the characters in each line with `wc -c`

Add the character amount to `sum` and set it equal to `sum`

Print it out

Use `example2.fasta` as input

# Loops - until

- **Infinite loops**
- Until condition is true, print the iteration number, increment the iteration number, and wait 1 second
- This loop will never end since the condition is hard coded to false. Press Ctrl-C to end the loop

Set `con` to `false`  
and `itnum` to `0`

```
con=false
itnum=0
until $con
do
    echo "Iteration no : $itnum"
    ((itnum++))
    sleep 1
done
```

Run `until` `con` is  
`false` (forever)

Print `itnum` and  
increment by 1

Wait 1 second  
before  
continuing



# Loops - until

- Writing information into a file
- Press Enter, Ctrl-D when you are done typing the file contents
- Does the same thing as the while loop shown before

Print a prompt statement

Use `read` to read in the user's input and store it as `filename`

```
echo -n "Enter the filename to create: "  
read filename  
  
until ! read line  
do  
    echo $line >> $filename  
done
```

Use `read` to read in the user's input for the file's contents and store each line as `line`

Print the line into the file



# Functions

- Self contained modules of code that accomplish a specific task, usually taking in data, processing it, and returning a result
- Two ways to set it up (you choose!):

```
function_name() {  
    commands  
}
```

Includes **parentheses** after the function's name

OR

```
function function_name {  
    commands  
}
```

No parentheses and the word "**function**" before the function's name

- Run functions by "**calling**" them:

```
function_name
```



# Functions - Examples

- Function names should be meaningful and relate to the functionality of the function!

```
method1() {  
    echo This is a function  
}
```

First method contains  
**parentheses** after the  
function's name

```
function method2 {  
    echo This is also a function  
}
```

Second method contains  
**"function"** before the  
function's name

```
# Calling both functions  
method1  
method2
```

Run the functions by  
**calling** them (use the  
function's name)



# Functions With Arguments

- Arguments are set similar to parameters
- Each argument is specified by a dollar sign and the argument number
  - \$1, \$2, etc

```
function_name() {  
    echo $1  
    echo $2  
}
```

- Call the function to set the arguments:

```
function_name arg1 arg2
```





# Functions With Arguments

- Similar to parameters, arguments can be set equal to variables

Create a function called `summation`

```
summation() {  
    echo Adding $1 to $2  
    sum=$(( $1 + $2 ))  
    echo $sum  
}
```

Take in the first parameter with `$1`

Take in the second parameter with `$2`

Add them together and set it equal to `sum`

Call the function, changing the arguments each time you call it

```
summation 2 6 # will add 2+6  
summation 3 0 # will add 3+0
```

\$1	\$2
2	6
3	0



# Example Scripts

- Example scripts contain everything we went over and then some!
- **loops.sh**
  - Arrays in for loops
  - While loops and various uses (infinite loop commented out)
  - Until loops and various uses (infinite loop commented out)
- **functions.sh**
  - Basic functions
  - Functions with arguments
  - Return values
  - Variable scope
  - Parameters in scripts vs arguments in functions



# GitHub

- A code hosting platform for version control and collaboration.
- Things you'll do today:
  - Make a new repository (sometimes called a repo)
  - Add directories within the repo
  - Make and edit readmes
  - Upload files
- An example repository:  
<https://github.com/madelinebellanger/BINF2111/>
- Tutorial time!

