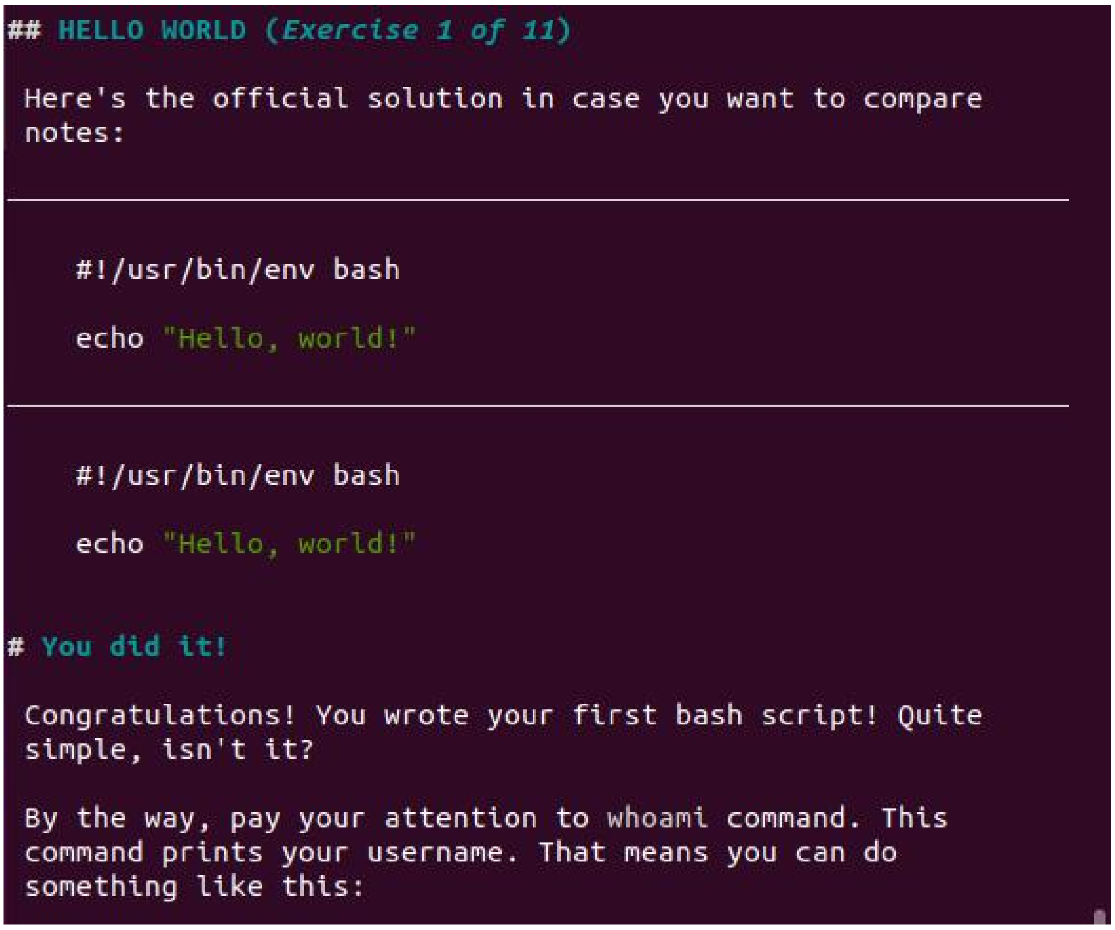
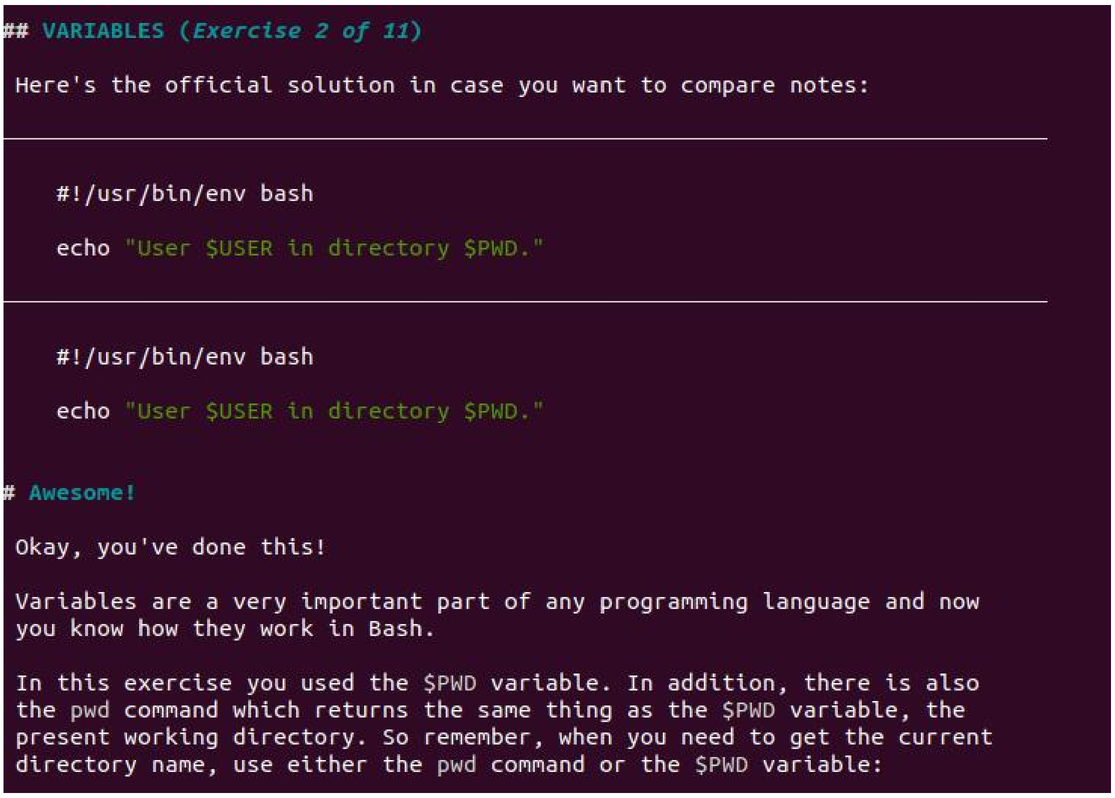
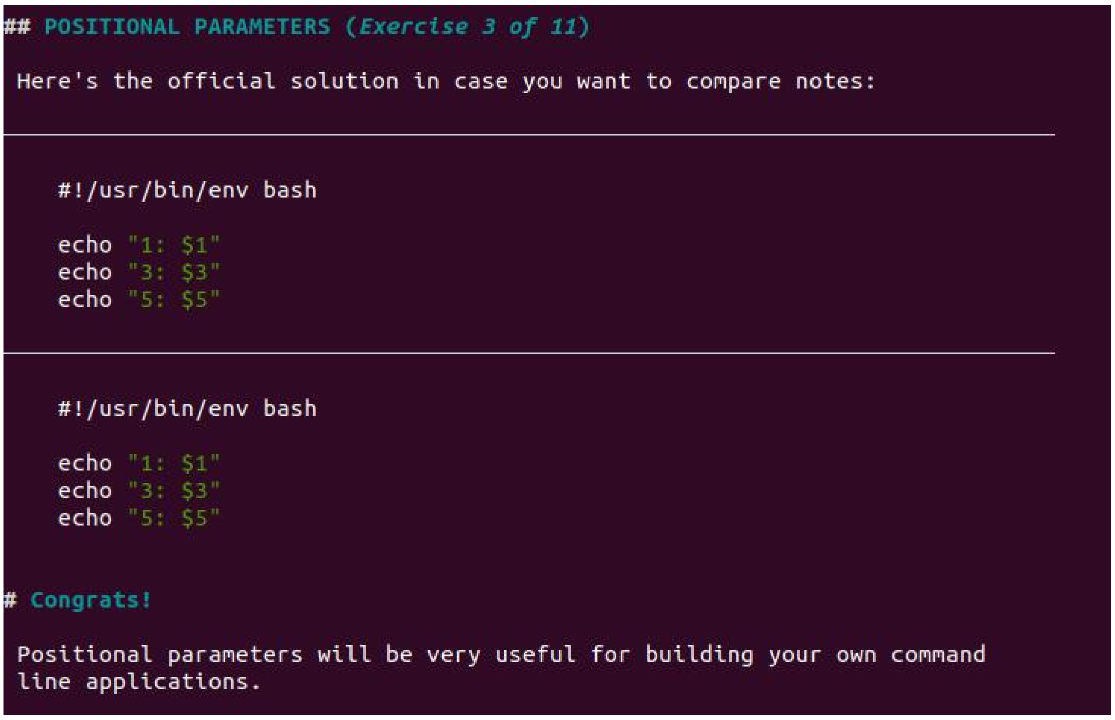


```
## HELLO WORLD (Exercise 1 of 11)

 Here's the official solution in case you want to compare
 notes:

────────────────────────────────────────────────────────────

    #!/usr/bin/env bash

    echo "Hello, world!"

────────────────────────────────────────────────────────────

    #!/usr/bin/env bash

    echo "Hello, world!"

# You did it!

 Congratulations! You wrote your first bash script! Quite
 simple, isn't it?

 By the way, pay your attention to whoami command. This
 command prints your username. That means you can do
 something like this:
```



```
## VARIABLES (Exercise 2 of 11)

 Here's the official solution in case you want to compare notes:

──────────────────────────────────────────────────────────────────

    #!/usr/bin/env bash

    echo "User $USER in directory $PWD."

──────────────────────────────────────────────────────────────────

    #!/usr/bin/env bash

    echo "User $USER in directory $PWD."

# Awesome!

 Okay, you've done this!

 Variables are a very important part of any programming language and now
 you know how they work in Bash.

 In this exercise you used the $PWD variable. In addition, there is also
 the pwd command which returns the same thing as the $PWD variable, the
 present working directory. So remember, when you need to get the current
 directory name, use either the pwd command or the $PWD variable:
```



POSITIONAL PARAMETERS (*Exercise 3 of 11*)

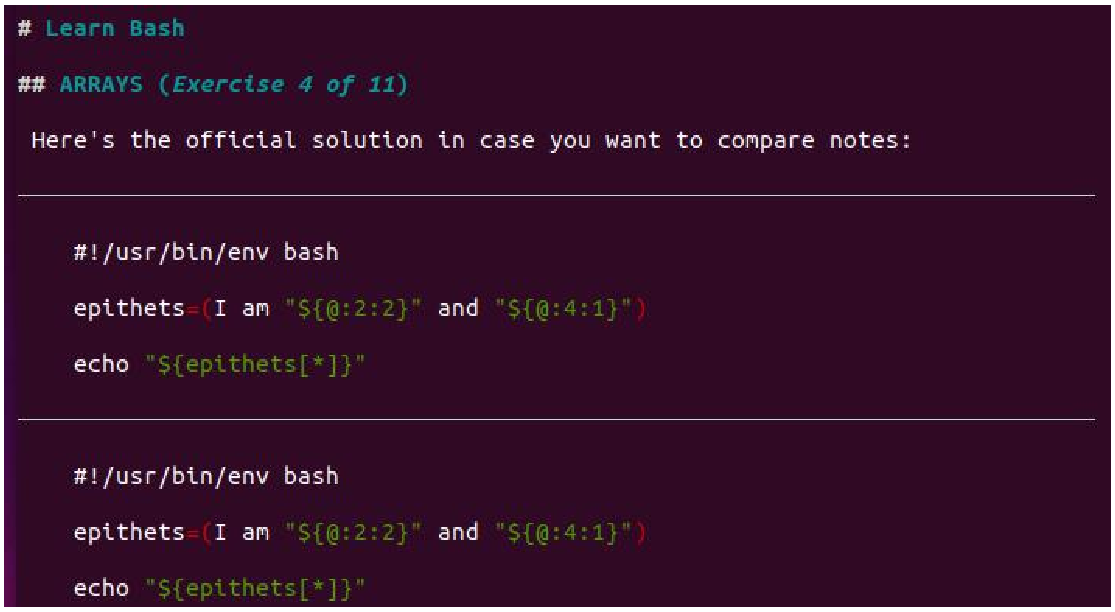
Here's the official solution in case you want to compare notes:

```bash
#!/usr/bin/env bash

echo "1: $1"
echo "3: $3"
echo "5: $5"
```

```bash
#!/usr/bin/env bash

echo "1: $1"
echo "3: $3"
echo "5: $5"
```

Congrats!

Positional parameters will be very useful for building your own command line applications.



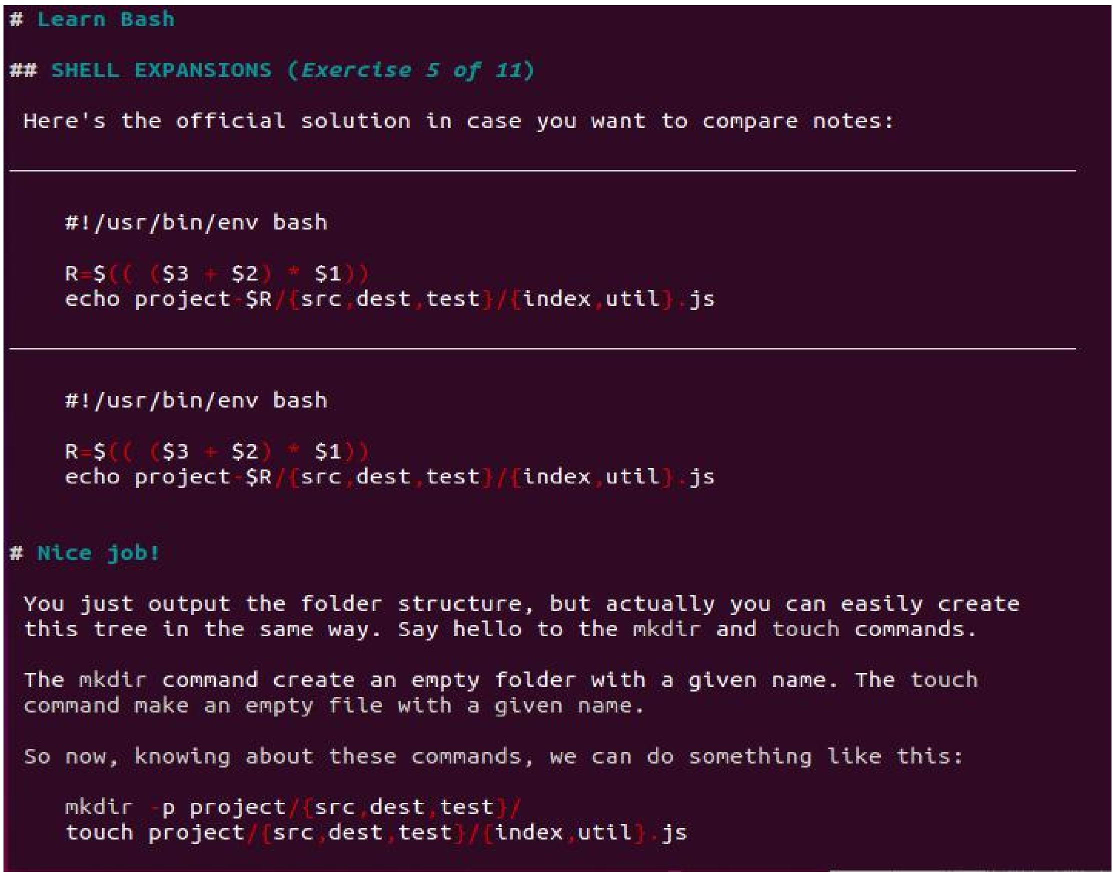
Learn Bash

ARRAYS (*Exercise 4 of 11*)

Here's the official solution in case you want to compare notes:

```bash
#!/usr/bin/env bash

epithets=(I am "${@:2:2}" and "${@:4:1}")

echo "${epithets[*]}"
```

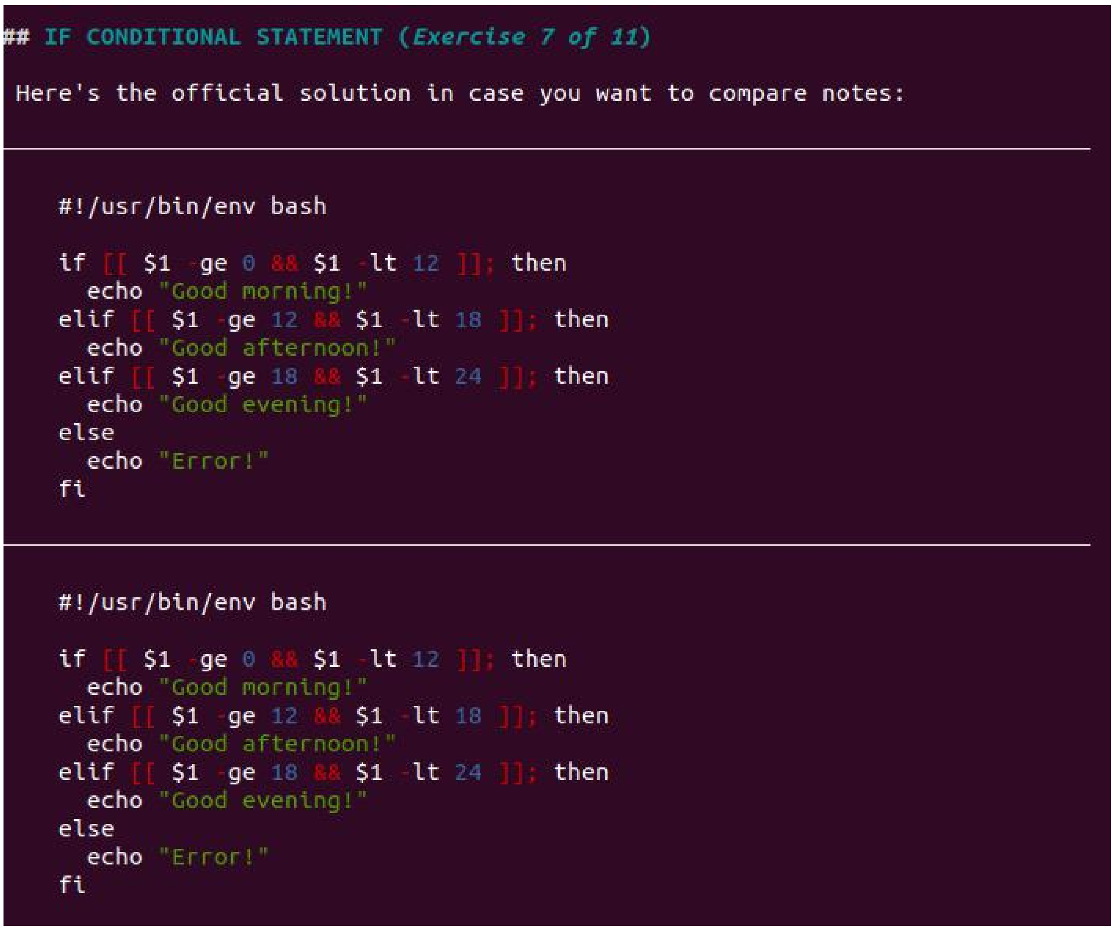
```bash
#!/usr/bin/env bash

epithets=(I am "${@:2:2}" and "${@:4:1}")

echo "${epithets[*]}"
```



```
# Learn Bash

## SHELL EXPANSIONS (Exercise 5 of 11)

 Here's the official solution in case you want to compare notes:

 ─────────────────────────────────────────────────────────────────

    #!/usr/bin/env bash

    R=$(( ($3 + $2) * $1))
    echo project-$R/{src,dest,test}/{index,util}.js

 ─────────────────────────────────────────────────────────────────
```

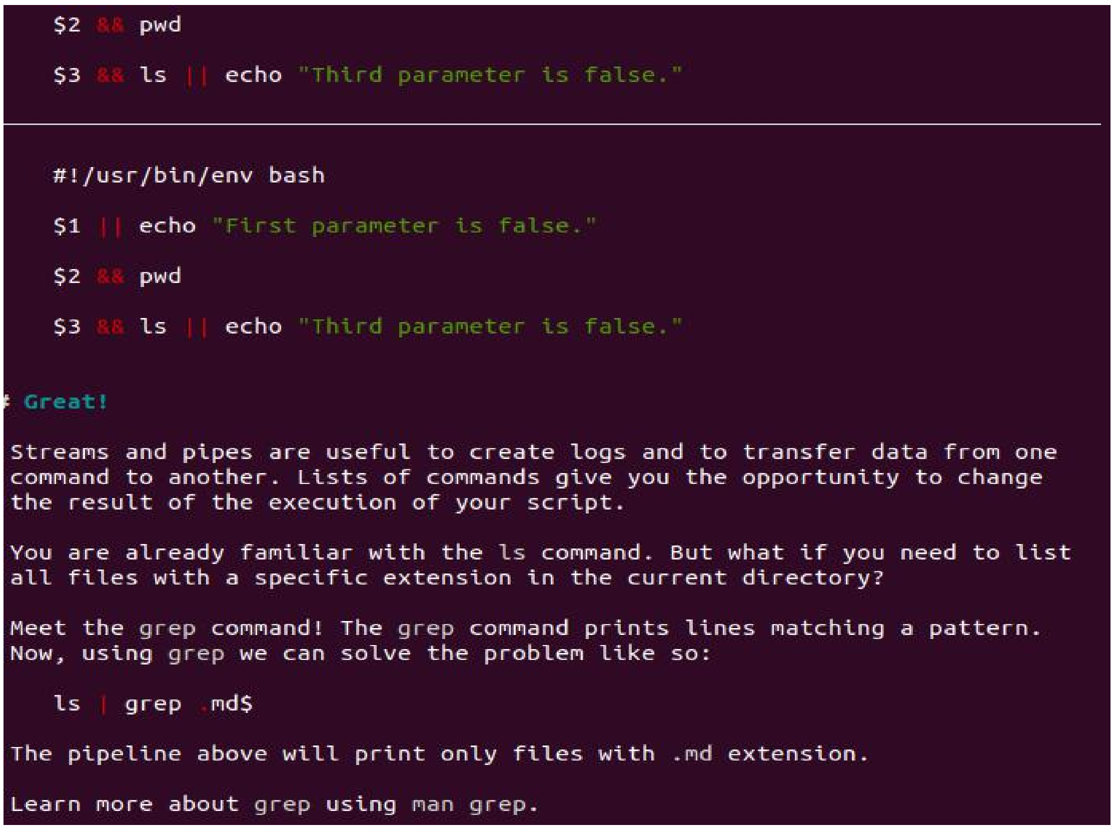
```
    #!/usr/bin/env bash

    R=$(( ($3 + $2) * $1))
    echo project-$R/{src,dest,test}/{index,util}.js
```

```
# Nice job!

 You just output the folder structure, but actually you can easily create
 this tree in the same way. Say hello to the mkdir and touch commands.

 The mkdir command create an empty folder with a given name. The touch
 command make an empty file with a given name.

 So now, knowing about these commands, we can do something like this:

    mkdir -p project/{src,dest,test}/
    touch project/{src,dest,test}/{index,util}.js
```



```
## IF CONDITIONAL STATEMENT (Exercise 7 of 11)

 Here's the official solution in case you want to compare notes:

 ─────────────────────────────────────────────────────────────────

    #!/usr/bin/env bash

    if [[ $1 -ge 0 && $1 -lt 12 ]]; then
      echo "Good morning!"
    elif [[ $1 -ge 12 && $1 -lt 18 ]]; then
      echo "Good afternoon!"
    elif [[ $1 -ge 18 && $1 -lt 24 ]]; then
      echo "Good evening!"
    else
      echo "Error!"
    fi

 ─────────────────────────────────────────────────────────────────
```

```
    #!/usr/bin/env bash

    if [[ $1 -ge 0 && $1 -lt 12 ]]; then
      echo "Good morning!"
    elif [[ $1 -ge 12 && $1 -lt 18 ]]; then
      echo "Good afternoon!"
    elif [[ $1 -ge 18 && $1 -lt 24 ]]; then
      echo "Good evening!"
    else
      echo "Error!"
    fi
```



```
    $2 && pwd

    $3 && ls || echo "Third parameter is false."
```

```
    #!/usr/bin/env bash

    $1 || echo "First parameter is false."

    $2 && pwd

    $3 && ls || echo "Third parameter is false."
```

Great!

Streams and pipes are useful to create logs and to transfer data from one command to another. Lists of commands give you the opportunity to change the result of the execution of your script.

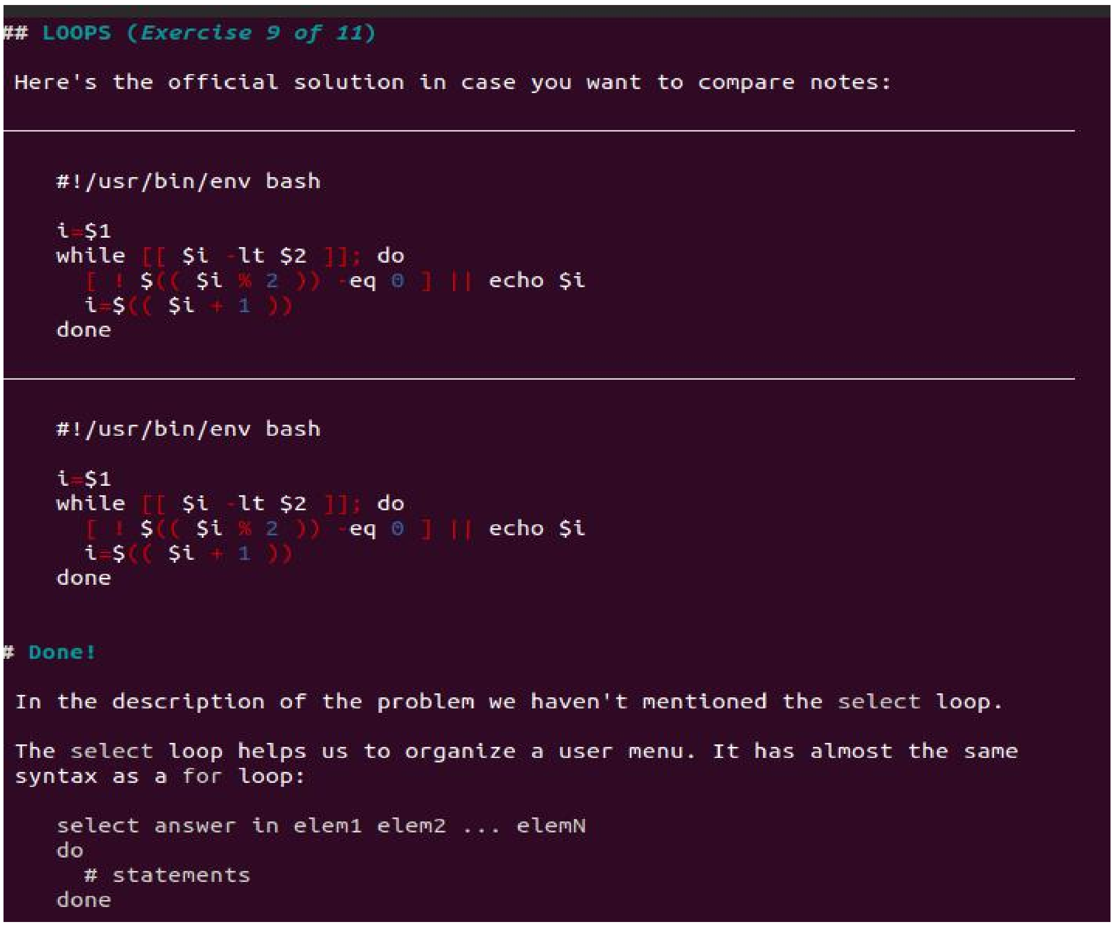
You are already familiar with the ls command. But what if you need to list all files with a specific extension in the current directory?

Meet the grep command! The grep command prints lines matching a pattern. Now, using grep we can solve the problem like so:

```
    ls | grep .md$
```

The pipeline above will print only files with .md extension.

Learn more about grep using man grep.



LOOPS *(Exercise 9 of 11)*

Here's the official solution in case you want to compare notes:

```
    #!/usr/bin/env bash

    i=$1
    while [[ $i -lt $2 ]]; do
      [ ! $(( $i % 2 )) -eq 0 ] || echo $i
      i=$(( $i + 1 ))
    done
```

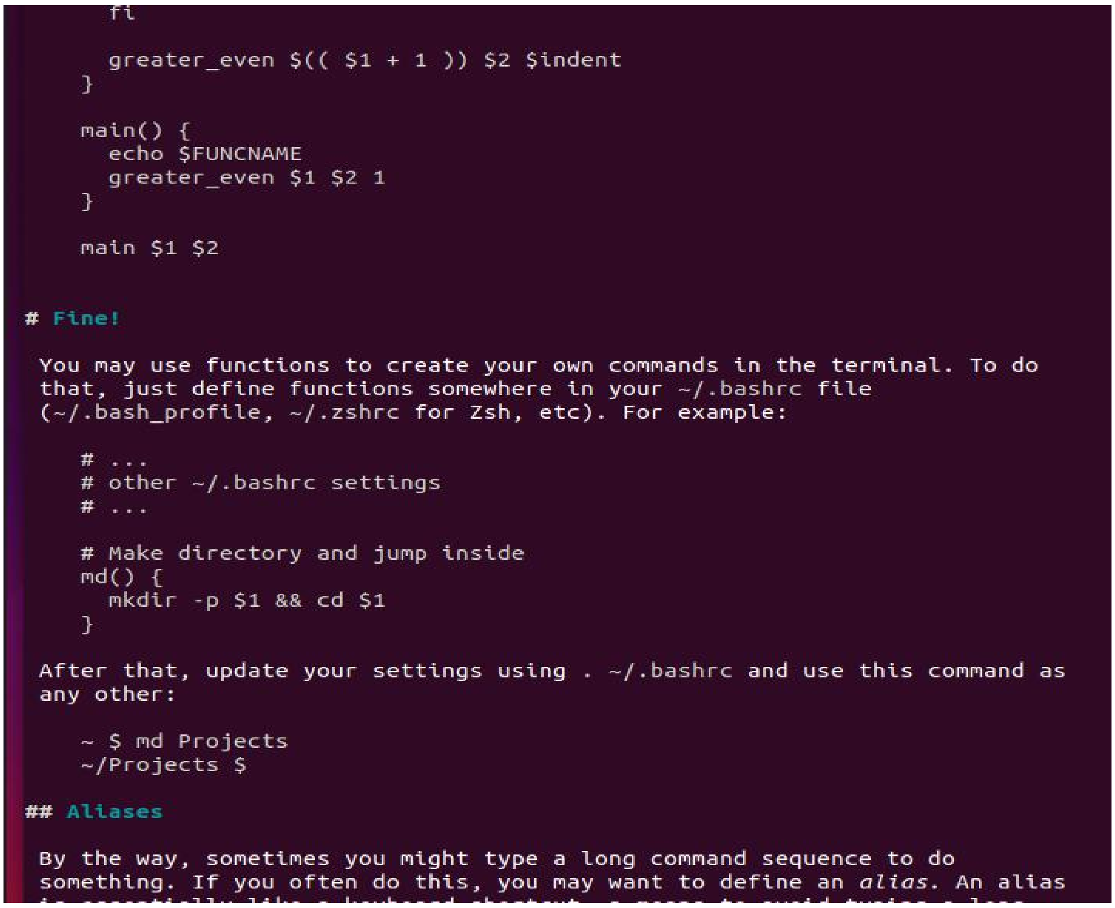
```
    #!/usr/bin/env bash

    i=$1
    while [[ $i -lt $2 ]]; do
      [ ! $(( $i % 2 )) -eq 0 ] || echo $i
      i=$(( $i + 1 ))
    done
```

Done!

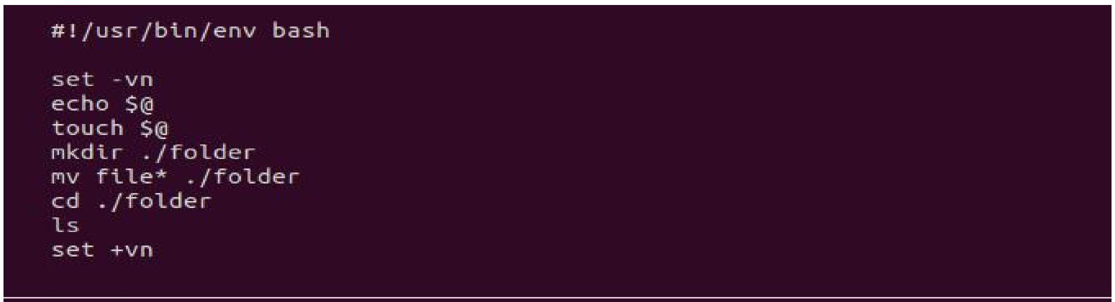
In the description of the problem we haven't mentioned the select loop.

The select loop helps us to organize a user menu. It has almost the same syntax as a for loop:

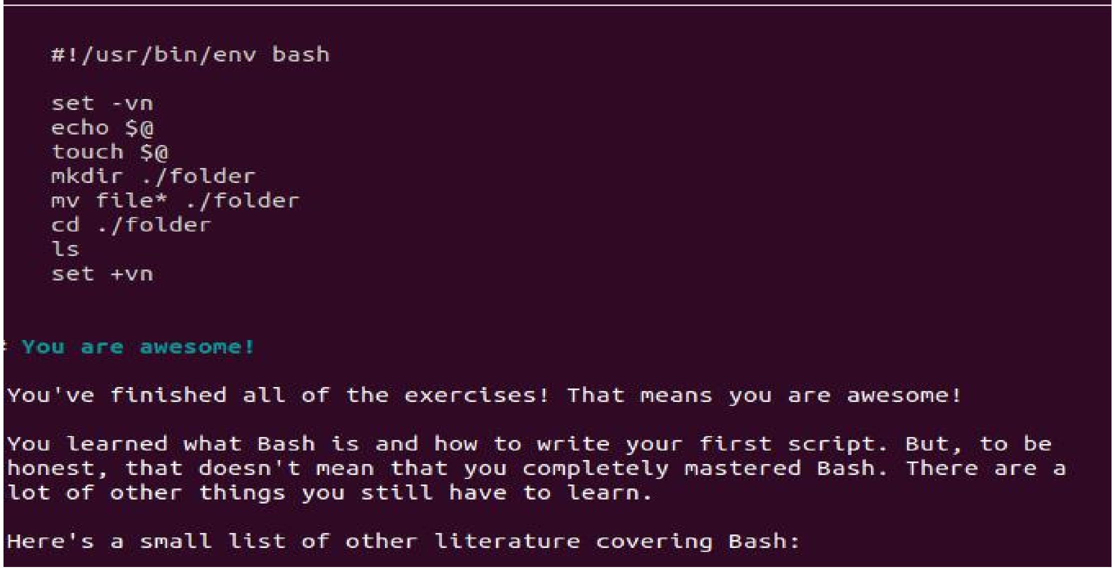
```
    select answer in elem1 elem2 ... elemN
    do
      # statements
    done
```



```
      fi

      greater_even $(( $1 + 1 )) $2 $indent
    }

    main() {
      echo $FUNCNAME
      greater_even $1 $2 1
    }

    main $1 $2


# Fine!

 You may use functions to create your own commands in the terminal. To do
 that, just define functions somewhere in your ~/.bashrc file
 (~/.bash_profile, ~/.zshrc for Zsh, etc). For example:

    # ...
    # other ~/.bashrc settings
    # ...

    # Make directory and jump inside
    md() {
      mkdir -p $1 && cd $1
    }

 After that, update your settings using . ~/.bashrc and use this command as
 any other:

    ~ $ md Projects
    ~/Projects $

## Aliases

 By the way, sometimes you might type a long command sequence to do
 something. If you often do this, you may want to define an alias. An alias
```



```
   #!/usr/bin/env bash

   set -vn
   echo $@
   touch $@
   mkdir ./folder
   mv file* ./folder
   cd ./folder
   ls
   set +vn
```



```
   #!/usr/bin/env bash

   set -vn
   echo $@
   touch $@
   mkdir ./folder
   mv file* ./folder
   cd ./folder
   ls
   set +vn


 You are awesome!

You've finished all of the exercises! That means you are awesome!

You learned what Bash is and how to write your first script. But, to be
honest, that doesn't mean that you completely mastered Bash. There are a
lot of other things you still have to learn.

Here's a small list of other literature covering Bash:
```