Conditional expressions are powerful because you can use them to control how a Bash program that you're writing is executed. One of the fundamental constructs in Bash programming is the **IF statement**. Code written inside of an IF statement is only executed *if* a certain condition is true, otherwise the code is skipped. Let's write a small program with an IF statement:

```
#!/usr/bin/env bash
# File: simpleif.sh

echo "Start program"

if [[ $1 -eq 4 ]]
then
  echo "You entered $1"
fi

echo "End program"

```

First this program will print "Start program", then the IF statement will check if the conditional expression [[$1 -eq 4]] is true. It will only be true if you provide 4 as the first argument to the script. If the conditional expression if true then it will execute the code in between then and fi, otherwise it will skip over that code. Finally the program will print "End program."

Let's try running this Bash program a few different ways. First we'll run this program with no arguments:

```
bash simpleif.sh

## Start program
## End program

```

Since we didn't provide any arguments to simpleif.sh the code within the IF statement was skipped! Now let's try providing an argument to this script:

```
bash simpleif.sh 77

## Start program
## End program

```

We provided the argument 77, however 77 is not equal to 4, therefore the code within the IF statement was once again skipped. Finally let's provide 4 as an argument:

```
bash simpleif.sh 4

## Start program
## You entered 4
## End program
```

It worked! Since the first argument to this script was 4, and 4 is equal to 4, the code within the IF statement was executed. You can pair IF statements with ELSE statements. An ELSE statement only runs if the conditional expression being evaluated by the IF statement is false. Let's create a simple program that uses an ELSE statement:

```
#!/usr/bin/env bash
# File: simpleifelse.sh

echo "Start program"

if [[ $1 -eq 4 ]]
then
  echo "Thanks for entering $1"
else
  echo "You entered: $1, not what I was looking for."
fi

echo "End program"
```

Now let's try running this program a few different ways:

```
bash simpleifelse.sh 4

## Start program
## Thanks for entering 4
## End program

```

The conditional expression [[$1 -eq 4]] was true so code inside of the IF statement was run and the code in the ELSE statement was not run. What do you think will happened when we make the conditional expression false?

```
bash simpleifelse.sh 3

## Start program
## You entered: 3, not what I was looking for.
## End program

```

The conditional expression [[$1 -eq 4]] was false so code inside of the ELSE statement was run and the code in the IF statement was not run.

Between IF and ELSE statements you can also have ELIF statements. These statements act like IF statements except they're only evaluated if preceding IF and ELIF statements have all evaluated false conditional expressions. Let's create a brief program using ELIF:

```
#!/usr/bin/env bash
# File: simpleelif.sh

if [[ $1 -eq 4 ]]
then
  echo "$1 is my favorite number"
elif [[ $1 -gt 3 ]]
then
  echo "$1 is a great number"
else
  echo "You entered: $1, not what I was looking for."
fi

```

First let's run the program with 4 as the first argument:

```
bash simpleelif.sh 4

## 4 is my favorite number

```

The condition in the IF statement was true, so only the first echo command was executed. Now let's run the program with 5 as the first argument:

```
bash simpleelif.sh 5

## 5 is a great number

```

If first condition is false since 5 is not equal to 4, but then the next condition in the ELIF statement is true since 5 is greater than 3, so that echo command is executed and the rest of the statement is skipped. Try to guess what will happen if we use 2 as an argument:

```
bash simpleelif.sh 2

## You entered: 2, not what I was looking for.
```

Since 2 is neither equal to 4 nor greater than 5, the code in the ELSE statement is executed.

You should also know that you can combine conditional execution, conditional expressions, and IF/ELIF/ELSE statements. The conditional execution operators AND (&&) and OR (||) can be used in an IF or ELIF statement. Let's look at an example using these operators in an IF statement:

```
#!/usr/bin/env bash
# File: condexif.sh

if [[ $1 -gt 3 ]] && [[ $1 -lt 7 ]]
then
  echo "$1 is between 3 and 7"
elif [[ $1 =~ "Jeff" ]] || [[ $1 =~ "Roger" ]] || [[ $1 =~ "Brian" ]]
then
  echo "$1 works in the Data Science Lab"
else
  echo "You entered: $1, not what I was looking for."
fi
```

Now let's test this script with a few different arguments:

```
bash condexif.sh 2
bash condexif.sh 4
bash condexif.sh 6
bash condexif.sh Jeff
bash condexif.sh Brian
bash condexif.sh Sean

## You entered: 2, not what I was looking for.
## 4 is between 3 and 7
## 6 is between 3 and 7
## Jeff works in the Data Science Lab
## Brian works in the Data Science Lab
## You entered: Sean, not what I was looking for.
```

The conditional execution operators work just like they would on the command line. If the entire conditional expression evaluates to the equivalent of truethen the code within the IF statement is executed, otherwise it is skipped.

Finally we should note that IF/ELIF/ELSE statements can be nested inside of other IF statements. Here's a small example of a program with nested statements:

```
#!/usr/bin/env bash
# File: nested.sh

if [[ $1 -gt 3 ]] && [[ $1 -lt 7 ]]
then
  if [[ $1 -eq 4 ]]
  then
    echo "four"
  elif [[ $1 -eq 5 ]]
  then
    echo "five"
  else
    echo "six"
  fi
else
  echo "You entered: $1, not what I was looking for."
fi
```

Now let's run it a few times:

```
bash nested.sh 2
bash nested.sh 4
bash nested.sh 6

## You entered: 2, not what I was looking for.
## four
## six
```

In order to get to the inner IF statement, the conditions for the outer IF statement must be met first (the first argument for the script must be between 3 and 7). As you can see combining variables, arguments, conditional expressions, and IF statements allow you to write more powerful Bash programs.

Summary

- All Bash programs have an exit status. true has an exit status of 0 and false has an exit status of 1.
- Conditional execution uses two operators: AND (&&) and OR (||) which you can use to control what command get executed based on their exit status.
- Conditional expressions are always in double brackets ([[ ]]). They have exit an exit status of 0 if they contain a true assertion or 1 if they contain a false assertion.
- IF statements evaluate conditional expressions. If an expression is true then the code within an IF statement is executed, otherwise it is skipped.
- ELIF and ELSE statements also help control the flow of a Bash program, and IF statements can be nested within other IF statements.

Mark as completed