1. **awk ‘{print $1}’ AwkLab.data** is used to print only the first names of people in the AwkLab.data file.

**print $1** within the **‘{}’** which if you put them together would become this: **‘{print $1}’** is used to tell awk to print only the first names. This is because of the default field separator being a space in awk, for example, the first name is field 1, space and then everything after that is field 2 which would be either last names or middle names. As such, the screenshot below shows only the first names. A black screen with white text

Description automatically generated

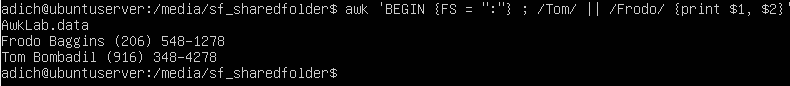
1. **awk ‘BEGIN {FS = “:”} ; /Tom/ || /Frodo/ {print $1, $2}’ AwkLab.data**

**Begin {FS = “:”}** is a beginning block which is executed before reading the input file. This is a method to change the default field separator from a space to a colon. The colon is the field separator because the colon comes after the last name and the phone number.

**/Tom/ || /Frodo/ {print $1, $2}** is the main block of AWK code that consists of a pattern and then an action.

**/Tom/ || /Frodo/** is the pattern that matches lines that either have tom or Frodo. Note that the 2 **||** are used to signify an and operator so that awk looks for 2 patterns at once.

**{print $1, $2}** is an action associated with the pattern. The first and second fields ($1 and $2) which are separated by a colon are printed into the output. **Awklab.data** is the input file that awk processes.



1. **awk ‘/Peregrin/ {print $1, $2}’ Awklab.data**

**/Peregrin/ {print $1, $2}** is the pattern-action statement within the single quotes that awk is executing.

**/Peregrin/** is the pattern specifying the condition for the performed action. In this case, it would be searching for the “Peregrin” string.

**{print $1, $2}** is the action part of the argument enclosed in the curly braces. What’s happening here is that awk will be told what to do when the pattern is matched. $1 and $2 refer to the first and second fields of the input line.

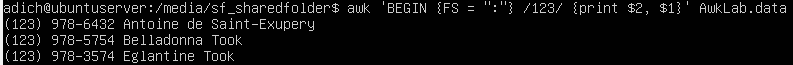


1. **awk ‘BEGIN {FS = “:”} /123/ {print $2, $1}’ AwkLab.data**

**BEGIN {FS = “:”}** initializes the awk script and works like this: The **BEGIN** block is executed before processing any input lines. **{FS = “:”}** is used to set the field separator to **“:”.** The fields in the input file are separated by colons.

**/123/** is a pattern match matching lines that contain 123 anywhere in the line.

**{print $2, $1}** is an action executed for lines that match the pattern **/123/ .** The second field which is **$2** followed by the first field which is **$1**. Fields in awk are numbered starting from 1 which means **$1** represents the first field of data and **$2** represents the second field of data. As such, the output prints the phone numbers in the 123 area code first and then their full names.



Here is the command I used: Also, note that the curly braces are dividers for different parts of the code. The curly braces in blue are for the whole script not including the if statement and the yellow curly braces only apply to the print statement directly below the if statement. As such, here is the screenshot

1. A black background with colorful text

   Description automatically generated

**#!/usr/bin/awk -f**  specifies the line path to the awk interpreter in your server and tells linux to execute the script file with awk. The file path is /usr/bin/awk by default. Run the command **which awk** in order to find out the file path so you know what to but for your shebang line fir your script. The **-f** is an awk option which tells awk that the next series of arguments after the script name is the name of the file containing the awk script to be executed. In summary, this tells linux that the script is to be executed using awk.

**FS=”:”** is for telling awk to change the delimiter from space to colon. What this means is that awk will split every input line into fields based on the colon delimiter.

**split($1, words, “ ”)** splits the first field of each line ($1) into words based on the space delimiter. The **split** part of the code is the name of the built-in awk function which splits strings into arrays. **$1** is the first field of the input line.

**Words** is the name of the array that will contain the split components of field 1. The double quotes which is the third argument enclosed in parenthesis is the delimiter used to actually split the first field into words.

The delimiter is a single space in this case which means that every ford in the first field is a separate element in the words array.

So, in summary when you put it all together, this part of the code takes the first field of each line, splits it into words using a space as the delimiter and then the resulting words are stored in the **words** array. It then holds each individual word of the first field which is accessed later in the script. Each word can be processed separately.

**‘if (substr(words[length(words)], 1, 1) = = “T” || substr(words[length(words)], 1, 1) == “D”** checks to see if the first character of the last word in the line starts with either a “T” or “D” .

**If** is a keyword in awk which is used to start a conditional statement which is something that’s present in a lot of programming languages such as python, C and so on.

**(substr(words[length(words)], 1, 1)** substr extracts a substring from a string. For example, this case signifies that the first character of the last word which is the last element of the **words** array.

**words[length(words)]** accesses the last element of the words array which represents the last word of the line.

The first 1 in **(substr(words[length(words)], 1, 1)** is the starting position for the substring which means it’s the first character.

The second 1 in **(substr(words[length(words)], 1, 1)** is the length of the substring which in this case means that only 1 character is to be extracted.

The **==** represent the equality comparison operator in awk which checks if the first character extracted is equal to “T” in this case. The “T” is a literal string.

The **||** represent an OR operator which allow for multiple conditions.

**substr(words[length(words)], 1, 1) == “D”** is exactly the same explanation. The only difference here is that the “D” is present.

When you put it all together, this part of my code is the pattern matching part of my code.

**print words[length(words)];** if the condition in the if statement is true, the last word of each line is printed.

How I did this step is that I broke this up into small pieces, I first tried to understand which field is which since a lot of what awk involves is field separators which have spaces as the default delimiter. I also tried to get lines starting with t or d by using the ^ character in my code. I basically slowly built pieces together which eventually lead to how I solved it. Make sure you give your file executable permissions. For simplicity’s sake, just do **chmod 777 filename.awk.** Make sure to replace filename with your actual script file. To actually run the script you do ./**filename.awk Awklab.data.**  This tells linux “hey, I would like you to execute this script file on this data file. ” As such, here is the screenshot showing my output.

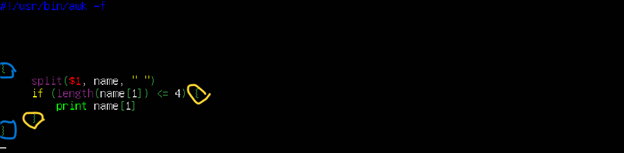
A screenshot of a computer

Description automatically generated

Works cited for number 5:

<https://opensource.com/article/18/7/cheat-sheet-awk>

Here is the code for question 6: The curly braces in blue are for the whole script not including the if statement and the yellow curly braces only apply to the print statement directly below the if statement. As such, here is the screenshot.

1. 

**#!/usr/bin/awk -f**  specifies the line path to the awk interpreter in your server and tells linux to execute the script file with awk.

**split($1, name, " ")** tells Linux to split field 1 using space as the delimiter and stores the parts in the array which in this case is name.

For context, the first line is: “Samuel Vimes:(510) 548-1278:250:100:175”.

After the name array is split, name[1] = “Samuel” and name[2] = “Vimes:510” is the basis of the code because the space is our new delimiter.

**if (length(name[1]) <= 4)** checks if the length of the first part of the splitting spring (which is the first name) to see if it’s less than 4 characters.

For example, let’s look at the line: Angua von Überwald:(206) 654-6279:250:60:50.

The name[1] part of the code would print “ponder” because it’s the first field.

Because we have the <= 4, the condition is false. When you really think about it, the field separator to change the default delimiter from space to colon isn’t really needed because there is no colon to worry about because we only want the first names.

As such, the conditional iterates over every line to see if the first name is 4 characters or less.

**{print name[1]}** prints the first names according to the conditional statement, this is the action part of the code. It prints names in field 1 which is separated by spaces.



Works cited for number 6:

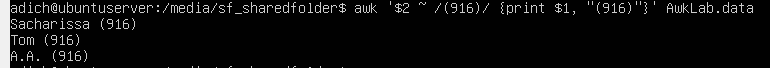
<https://www.digitalocean.com/community/tutorials/if-else-in-shell-scripts>

<https://www.educative.io/answers/what-are-unix-linux-basic-operators>

7.) **awk ‘$2 ~ /(916)/ {print $1, “(916)”}’ AwkLab.data**

**‘$2 ~ /(916)/** searches field 2 for any line matching 916. The ~ is used to search a field which in this case is field 2 or $2

**{print $1, “(916)”}** is the action part of the command. If there is a match, their first name along with the 916 area code in parentheses will be printed.



8.)Here is what I did for the script: Note that the blue circled braces correspond with the FS block and the print statements outside of the for loop. The yellow applies only to code enclose in the for loop.

A computer screen shot of text and symbols

Description automatically generated

**BEGIN {FS = “:”}** Sets the field separator to colon which means

**/Sacharissa Cripslock/** is a pattern match that matches lines containing Sacharissa Cripslock.

**printf "Sacharissa Cripslock”** is the action associated with the pattern.

**for(i=3; i<=NF; i++)** is a loop starting from the third field which is i=3 and iterates over all the fields the i <=NF is a conditional checking if the variable I is less than or equal to the number of fields NF in the current record. In this case, this loop iterates over each field starting from the third field which is i=3 up to the last field. The i++ is used within the for loop and is increased by 1 allowing the loop to run through each field all the way up to the last field.

**printf " $%d", $i** is for formatting and printing text. How it works is that the %d is a character telling awk to print a dollar sign followed by an interger and $i is the value of the ‘i’ th field in the current line. In this case, I is the third field in the line we want. Because we have i++ in the loop, the value of all the i’s are being printed by including $i in our statement.

**print "\n";** tells awk to print a newline character ensuring your output ends with a new line to make your output easier to read.

9.) **awk -F’:’ ‘/[A-Z]/ {split($1, name, “ “); gsub(/^[[:space:]]+|[[:space:]]+$/, “”, name[length(name)]); print name[length(name)]; print($2) }’ AwkLab.data**

**-F’:’** Sets the field separator from space to colon.

**/[A-Z]/** Searches for any capital letters and in this case, is used to target the last names.

**{split($1, name, “ “);** separates the first field into an array which is called name.

**gsub(/^[[:space:]]+|[[:space:]]+$/, “”, name[length(name)])** removes leading and trailing whitespace from the last element of the name array.

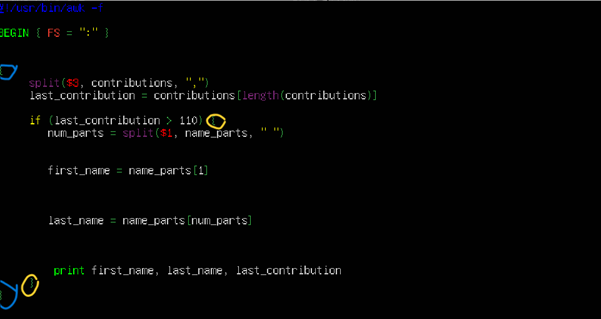
**name[length(name)]** corresponds to the last element of the name array.

**print name[length(name)]** is the action statement to print the last element of the name array which in this case is just the last names.

**print($2)** prints the phone numbers. Here are the screenshots: 

A screenshot of a computer screen

Description automatically generated

10.) Here is the script I used. The curly braces in blue are for the whole script not including the if statement and the yellow curly braces only apply to the print statement directly below the if statement. As such, here is the screenshot.

**BEGIN { FS = ":" }** Sets the field separator to colon which will split each line into fields whenever a colon is encountered.

**split($3, contributions, ",")**  tells awk to split the third part of every line which contain the different contribution numbers into smaller parts based on commas.

For example, the line Samuel Vimes:(510) 548-1278:250:100:175 would split 250:100:175 to 250, 100, and 175.

**last\_contribution = contributions[length(contributions)]** assigns the last contribution amount to the variable last\_contribution. That same line as above would be 175.

**if (last\_contribution > 110)** initiates a conditional statement. It tells awk that if the last contribution is greater than 110, then do something.

**num\_parts = split($1, name\_parts, " ")** splits part 1 of every line containing the name into smaller parts based on spaces. In the line: Samuel Vimes:(510) 548-1278:250:100:175, samuel vines would be split into Samuel and vines which is to say that they’re split into two different parts of one line.

**first\_name = name\_parts[1]** assigns the first part of the name to **first\_name.**

**last\_name = name\_parts[num\_parts]** assigns the last part of the name to the **last\_name** variable. The last part of the name is captured which could either be, the last name alone or the last part of someone’s full name if it contains middle names.

**print first\_name, last\_name, last\_contribution** prints out the first name, last name and the contribution amount for the last month but only for those that contributed more than $110!

As such, here is the output.

A screenshot of a computer

Description automatically generated

11.) Here is the script: The curly braces in blue are for the whole script not including the if statements and the yellow curly braces only apply to in this case, the phone number RSTART and RLENGTH statements. There is another set of yellow curly braces that apply to the if statement. As such, here is the screenshot.

**BEGIN { FS = ":" }** BEGIN block is executed before any processing of input lines occur. The FS is set to colon.

**split($1, name\_phone, "(")** Splits field 1 into the **name\_phone** array while using the ( as the delimiter because the first part of the phone number has parenthesis.

**split(name\_phone[1], name\_parts, " ")** Splits the first element of the array which is **name\_phone** this time using space as the delimiter.

**last\_name = name\_parts[length(name\_parts)]** pulls out the last element of of **name\_parts** which assumes the last name.

**phone\_number** initializes the phone number variable.

**if (match($0, /\([0-9]+\)[[:space:]][0-9]+-[0-9]+/)) {** Checks if the entire line which is $0 contains a phone number in the format of (000)-000-0000. Should a match be found, the **match()** returns the starting positions of the match in **RSTART** and the length of the match **RLENGTH.** Here’s how it works when broken down:

The escaped parentheses which would be in **if (match($0, /\([0-9]+\)[[:space:]][0-9]+-[0-9]+/))** match literal parentheses because parentheses are special characters in regex and are needed to be escaped to be literals.

**[0-9]+** matches one or more digits that specify any digit from 0 to 9 is allowed and the + indicates that there has to be one digit.

**[[:space:]]** is a character class matching any whitespace character such as spaces, tabs and line breaks.

**[0-9]+** matches one or more digits that specify any digit from 0 to 9 is allowed and the + indicates that there has to be one digit.

The hyphen character – is matched literally because it’s a special character in regex which separates the digits in the phone number.

**[0-9]+** matches one or more digits that specify any digit from 0 to 9 is allowed and the + indicates that there has to be one digit.

Speaking of RSTART and RLENGTH, **phone\_number = substr($0, RSTART, RLENGTH)** extracts the matched phone numbers from the line.

**split($3, contributions, ",")** Splits the $3 into contributions array using a comma as the delimiter.

**first\_contribution = contributions[1]** Extracts the first element of contributions which assumes that it’s the first month of the contribution.

**if (first\_contribution < 150)** Checks if month 1 of contributions is less than 150.

**print last\_name, phone\_number, first\_contribution** prints the last name, phone number as well as their month 1 contribution. Here is the output screenshot

12.) Here is the script for number 12: The curly braces in blue are for the whole script not including the if statement and the yellow curly braces only apply to the print statement directly below the if statement. As such, here is the screenshot. Note that This isn’t the full screenshot as **BEGIN {FS = “:”}** isn’t in the screenshot. I accidentally cut it out but make sure that this is at the top of your script before the main part of the code after you type out the whole shebang line for awk before you try this yourself.A screenshot of a computer program

Description automatically generated

**split($1, name\_phone, "(")** splits field 1 into the **name\_phone** array using the ( as a delimiter which separates the name from the phone number.

**split(name\_phone[1], name\_parts, " ")** splits the first part of the array **name\_phone** into the **name\_parts** array using the space as a delimiter

**first\_name = name\_parts[1]** Assigns the first word of the name to **first\_name** representing the first name.

**split($3, contributions, ",")** breaks up the third field which is $3 into the **contributions** array with the comma enclosed by double quotes as the delimiter.

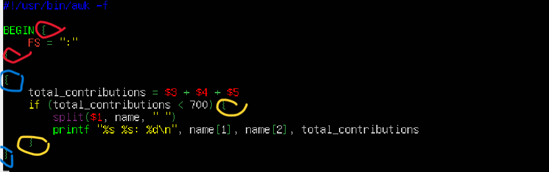
**first\_contribution = contributions[1]** Assigns element 1 of contributions array to **first\_contribution** representing month 1 of contributions.

**if (first\_contribution >= 10 && first\_contribution <= 200)** checks if the first contributions fall between 10 dollars and 200 dollars.

If the condition is true, then **print first\_name, first\_contribution** will print the first name and month’s contribution should it fall within the range.

A black screen with white text

Description automatically generated

13.) Here is my script. The curly braces circled in red are part of the FS = “:” bit. The blue curly braces act as separators for the main part of the code not including the if statement. The yellow curly braces apply to the lines below the if statement. 

**BEGIN { FS = ":" }** Sets the field separator to colon which will split each line into fields whenever a colon is encountered.

**total\_contributions = $3 + $4 + $5** Calculates the total contributions by summing up the values in fields 3, 4 and 5.

**if (total\_contributions < 700)** is an if statement that checks if the total contributions are less than 700. If it’s true, then the code will print.

**split($1, name, " ")** splits the contents of column 1 using a space as the delimiter storing the results in the name array.

**printf "%s %s: %d\n", name[1], name[2], total\_contributions** prints the first name, last name as well as the total contributions separated by spaces as well as being followed by new lines. Here’s how it works exactly:

printf is an awk command used for formatted printing.

%sis a placeholder for a string.

%sis another placeholder for a string

: represents a literal colon character.

%d represents a decimal integer placeholder.

\n is an escape sequence which signifies a newline character which tells awk to move to the next line after printing.

name[1] represents the first name extracted from column 1 of the input line

name[2] represents the last name extracted from column 1 of the input line

**total\_contributions** tells awk to print the value of the variable which sums up fields 3, 4 and 5. As such, here is the screenshot

A screenshot of a computer

Description automatically generated

14.) Here is the script I used: The blue curly braces are for the main part of my code. The yellow circled part of my code is for the lines below the if statement. A screenshot of a computer

Description automatically generated

**BEGIN { FS=":"; }** sets the field separator to colon.

The main block of the code starts with **total\_contributions=$3+$4+$5** which assigns fields $3, $4 and $5 of the data file to a variable called **total\_contributions.** All 3 fields are added up.

**average=total\_contributions/3** calculates the average contribution by dividing the total contributions by 3 and stores the result in the **average** variable.

**if (average > 300)** checks to see if the average contribution is greater than 3 after the division is done.

**split($1, name, " ")** splits field 1 which is the name into different parts using a space as the delimiter. This is then stored in the array called name.

**first\_name = name[1]** Pulls out the first name from the name array.

**last\_name\_initial = substr(name[length(name)], 1, 1)** Pulls out the first initial of the last name by taking the last element of the name array and taking the first character.

**Substr()** is a built in function for extracting substrings from strings.

**name[length(name)]** retrieves the last element of the array **name**.

**length(name)** returns the number of elements in the **name** array.

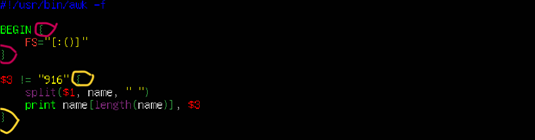
The first 1 in **last\_name\_initial = substr(name[length(name)], 1, 1)** is the starting position of the substring which tells awk to extract from the first character of the string obtained from the **name[length(name)]** bit.

The next 1 in **last\_name\_initial = substr(name[length(name)], 1, 1)** is the actual length of the substring to be extracted. This bit right here tells awk that a substring of the length is to be extracted. We only want the first character of that string which would be the first letter of last name.

**print first\_name, last\_name\_initial, average** prints the first name, last name initial and average contribution that is over 300 even after the division by 3.

Here is the output: A screenshot of a computer screen

Description automatically generated

15.) Here is the script used: The red curly braces apply to the BEGIN FS part of my code. The yellow curly brackets apply to what’s below the **$3 != "916"** 

**BEGIN FS=”[: ()]”** Sets the field separator to colon which will split each line into fields whenever a colon is encountered.

**$3 != "916"** represents a condition that checks whether the 3rd field of the input line which is the area code in this case is not equal(!=) to 916.

**split($1, name, “ “)** split the first field of the input line which is **$1** into pieces using the space as a delimiter which is **“ “** which is then stored in the **name**  array. For context, the first line and field of the data file is Samuel Vines. With the **name** array, the **name[1]** will contain Samuel and **name[2]** will contain Vimes.

**print name[length(name)], $3** prints the last element of the **name** array which corresponds to the last name which is then followed by the field $3. This will print the last name, and area code that isn’t 916. Here is the output

A screenshot of a computer

Description automatically generated

16.) Here is the command used as well as the output for the command. As you can see, the circled part in blue represents the actual command. A screenshot of a computer

Description automatically generated

**{print NR “:” $0}**  is the pattern and action statement of the command.

**NR** is the built-in awk variable representing the current record number being processed.

**:** is a string literal which contains a colon and space for formatting purposes.

**$0** is another built in variable in awk representing the whole current record which is the line in this case being processed.

17.) Here is what I did to print the name and total contribution of every person: The blue circled part represents the command used.

A screenshot of a computer

Description automatically generated

**-F:** tells awk to change the field separator from a space to a colon.

**total = $3 + $4 + $5** calculates the total contribution by adding up the values in the 3rd, 4th and 5th fields of every line assuming that the field separator is still colon and not space.

**print $1, “:”, total** prints the first field which is the full name including the middle names followed by a literal colon character as well as the total contribution which should have already been calculated by this point.

You can remove the literal colon character if you wanted to, the command will still work. Here is the screenshot showing what exactly this would look like. What you will notice is that between the **print$1** and the **total** is that there is nothing in between them. Which explains why the output is what you see in my screenshot below

A screen shot of a computer

Description automatically generated

18.) Here is the script that I did to do this step: The blue curly braces divide the FS part of my code and the yellow braces also divide the lines below tiffany aching which is meant for readability purposes.

A screen shot of a computer

Description automatically generated

**BEGIN FS= “:”** initiates the field separator to change from space to colon.

**$1 == "Tiffany Aching"** tells awk to check if the first field on any line has Tiffany Aching.

**$3 += 10;** Adds 10 to the third field which in this case would be tiffany’s first contribution amount.

**split($1, name, " ");** splits field $1 which is her full name into an array called **name** using a space as the delimiter which you can see by looking at the double quotes.

**fullname = name[1] " " name[2];** constructs the name by concatenating element 1 and 2 of the **name** array with a space in between. To be clear, the space is the double quotes in between name[1] and name[2].

**print fullname, $3** prints the value of the fullname variable as well as the updated first contribution. As such, here is the actual output. 

19.) Here is what I did for this step: circle blue part represents the command itself.A screenshot of a computer

Description automatically generated

**-F:** changes the field separator from space to colon.

**{sub()}** tells awk to substitute a thing or things. As such, the substitution arguments that take places would be enclosed within the parentheses.

The Samwise gamgee and “Sean Astin” in **{sub(“Samwise Gamgee”, “Sean Astin”)}** tells awk to substitute Samwise Gamgee with Sean Astin.

The 1 in in **{sub(“Samwise Gamgee”, “Sean Astin”)} 1** is considered a true condition in this case since there’s no action that’s specified. Awk performs the default action because of this which in this case is to print the current line.

20.)Here is the script: The blue curly braces correspond to different parts of the script instructions not including the conditional statements, for example, the first set of blue braces correspond to the part of the code that prints out all the first names of the tooks followed by total money contribtuions. As such, that’s how the script was organized. The yellow circled curly braces correspond to the conditonal statement. The line starting with **/took/** represent part 1 of my script with the total contribution for the tooks. The line starting with **last\_contribution** represents part 2 of my script with the last month contribution. The line with **sum = $3 + $4 + $5** represent the third part of the script with the average contributions

**A screenshot of a computer program

Description automatically generated**

**BEGIN { FS = ":" }** tells awk to changed the field separator from space to colon.

**Part 1 of the script explanation:**

**/Took:/** serves as search criteria for a pattern that should match any lines containing the string “Took”.

**first\_name = $1** assigns the first field to a variable which will extract the first field from the line.

**split(first\_name, name\_parts, " ")** splits the first name into parts while using the space as a delimiter which is then stored in the **name\_parts** array.

**printf("%s: Total contributions: $%d\n", name\_parts[1], $3 + $4 + $5)** prints the first name along with total contribution Here’s how it works exactly:

**printf** is an awk function for printing out formatted output.

**%s**  is a placeholder for a string and is replaced by the first name stored in **name\_parts[1]** .

**: Total contributions: $** is a literal strong that’s printed as is.

**%d** is a placeholder for integers and is what’s replaced by total contributions that were already calculated from the 3rd, 4th and 5th fields of the input lines.

**\n** tells awk to add a new line after printing lines.

**name\_parts[1]** is the first element of the array **name\_parts** which contains the first name pulled from the input line.

**$3 + $4 + $5** calculates total contributions by adding up all the values of the 3rd 4th and 5th fields of every line which in this case is just all the money contributions.

**Part 2 of the script:**

**last\_contribution = $NF** assigns the value of the last field of a line to the **last\_contribution** variable.

**if (last\_contribution >= 10 && last\_contribution <= 200)** checks to see if the value of the **last\_contribution** variable falls within the $10 to $200

**printf("Full Name: %s, Last Contribution: $%.2f\n", $1, last\_contribution)** prints everything including the **Full Name** which is a literal string that prints as is, the string placeholder which is **%s** followed by another literal string **Last Contribution** as well as a literal **$** sign. The value of the **last\_contribution** variable which is formatted with 2 decimal places**(%.2f)** as well as a newline character telling awk to start new lines **(\n).**

**Part 3 of the script**

**sum = $3 + $4 + $5** calculates the value of the 3rd 4th and 5th fields of the data file by adding them all together and storing the results in the **sum** variable.

**avg = sum / 3** calculates the average contribution by dividing the value of the **sum** variable by the number of contributions which is 3.

**if (avg < 300)** checks to see if the already calculated average contribution is less than $300.

**printf("Full Name: %s, Average Contribution: $%.2f\n", $1, avg)** prints the full name **($1)** followed by the average contribution **($)** as well as the value of **avg** after the initial calculations. The amount of average contribution is formatted with two decimal places (**%.2f)** as well as another newline **(\n)**

Here is the output. The yellow circled part in the below screenshot signify the first occurrence of the last contribution amount output which corresponds to part 2. The red circle part signifies the first occurrence of the average contribution amount output which corresponds to part 3 and the blue circled part signifies the total contribution amount output which corresponds to part 1.

A screenshot of a computer

Description automatically generated