ataccama

ONE Desktop Workshop

Data Transformations

Prepared for:	v15.4.x
Prepared by:	Ataccama
Dated:	October 2024

Ataccama Corporation | www.ataccama.com
Toronto, CA – Prague, CZ – London, UK – Paris, FR – Stamford, US – Munich, DE – Sofia, BG – Sydney, AU

Contents of the Document

1. Introduction

This workshop will introduce Transformation steps within the Ataccama ONE Desktop environment.

We will have a look at these steps in this workshop:

Column Assigner	Regex Matching
Transliterate	

We will also look at various debug options.

CAUTION *The following activities require you to have completed the previous exercise (**ONE DESKTOP Workshop – Flow Control**), so you have the plan created then available for use.*

2. Tasks

In this workshop, we will continue working with data that was used in previous workshops. This time the data will be modified on its way to the desired output. We will keep using the data from the **party_full_1.csv** source but create a new plan for our transformations by reusing the previous one.

We have received some more requirements for the Stream 1 output:

- The **fix_email** attribute needs to be upper case (but needs to remain lower case in **CONTACT** output).
- Create a new column **current_year** – it will contain the numeric value of the actual year.
- Create a new column **age** – it will calculate the difference between a person's year of birth and a current year.
- Create a new column **trans_name** - it takes values from **src_name** and replaces all occurrences of **"Smith"** with **"Smyth"**.
- Create a new column **trans_email** - it takes values from **src_email** and again replaces all occurrences of **"Smith"** with **"Smyth".**
- Create new columns **user** and **domain** – they will take the user and domain parts from **src_email**.

2.1. Column Assigner step

This step is a basic transformation step that allows you to use most of the expressions and apply them to your data. We will start with creating a new plan so we can use it:

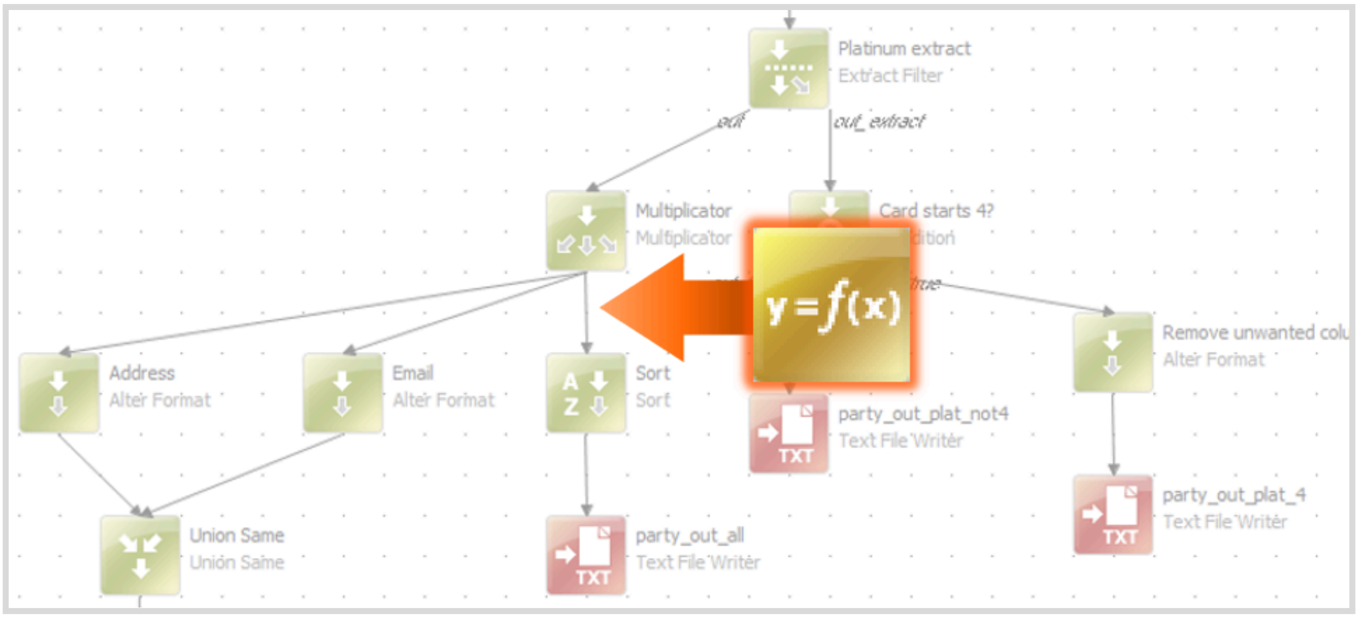
› In File ***Explorer***, make sure you are in your **Training** project and the ***plans*** folder.

› Right-click on your existing **02_party.plan** and select the ***Copy*** option.

› Right-click on the **plans** folder and click ***Paste*** to drop it here. You will be prompted to enter a new name – call it '**03_party.plan**'.

› Open your new **03_party.plan**

› Find the **Column Assigner** step and add it to this plan right between the **Multiplicator** and **Sort** steps for Stream 1:

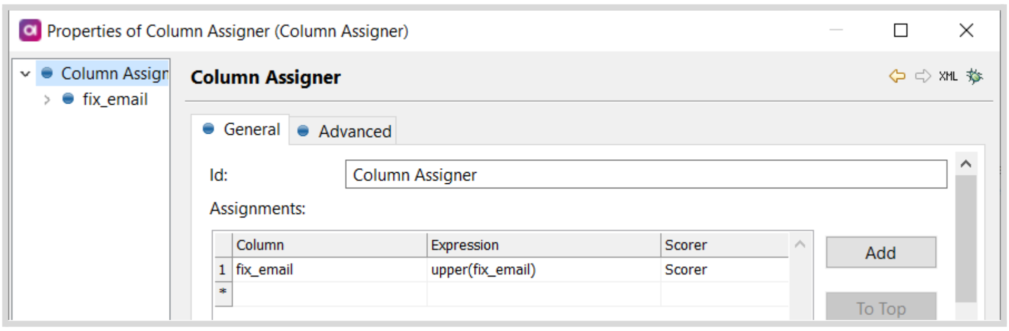


HINT

Dropping your step onto the connecting line will automatically include it in your flow

Once the **Column Assigner** step is in your plan you can start entering expressions into it:

- Double-click the **Column Assigner** to open it.
- In the ***Column*** field, set which attribute will be used for storing the modified values - **fix_email.**
- In the ***Expression*** field configure your functions or logic - ***upper*****(fix_email)**



Id: Column Assigner

Assignments:

	Column	Expression	Scorer
1	fix_email	upper(fix_email)	Scorer
*			

CAUTION

*Be careful **- Column Assigner** can overwrite existing values with new changes at once.*

The next task is to get the value of the current year. To do that, we will need to use a function that will be able to extract the value from a result of a complete current day's date (e.g., "12/26/2005"). The process will contain two basic procedures:

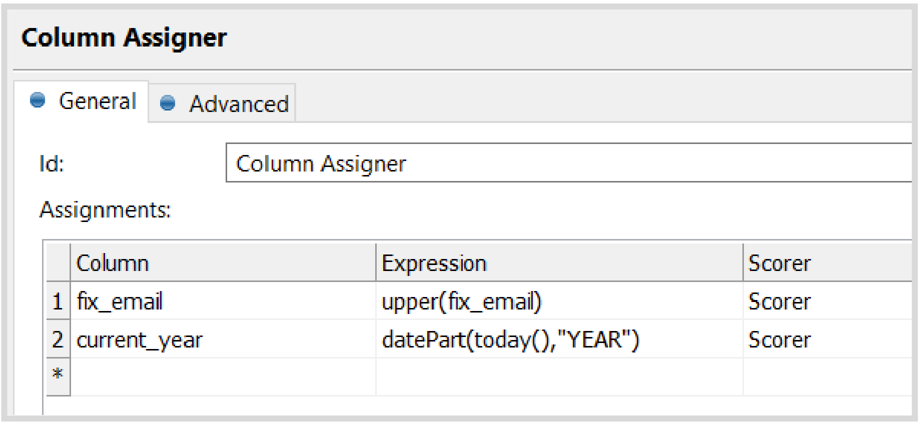
1. Get today's date – use the result of a function ***today*()**.
2. Extract only the year value so it's populated into the new column – use of a ***DatePart()*** function.

Let's start with the implementation of this task:

- Go back to the **Alter Format** step and add two new columns – **current_year** (INTEGER) and **age** (INTEGER).
- Return to the **Column Assigner** step and create a new assignment - set the **current_year** attribute as a target in the ***Columns***.
- Populate the ***Expression*** with the following definition as a combination of the two functions mentioned above:

  datePart(today(),'YEAR'**)**

Your result should now look like this:



Column Assigner

General | Advanced

Id: Column Assigner

Assignments:

	Column	Expression	Scorer
1	fix_email	upper(fix_email)	Scorer
2	current_year	datePart(today(),"YEAR")	Scorer
*			

Now we move to the more complicated scenario – calculating the person's age. To do that, we need to start thinking about how to set the logic:

1. We have a value of a birth date available (**src_birth_date**) but in a **STRING** format.
2. We need to populate the current year's value.
3. We need to combine these two values and get their difference so we can populate the **age** column.

We can now separate these phases into a definition applied by expressions:

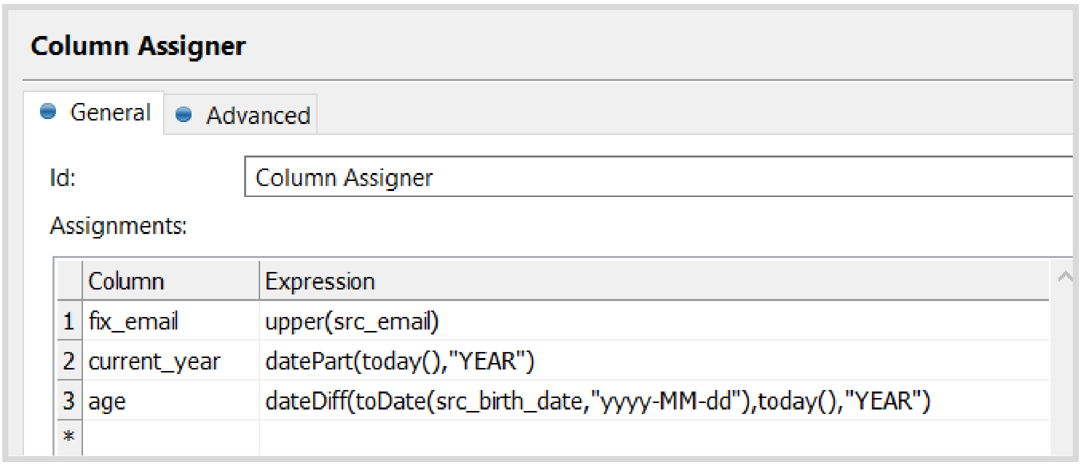
1. Convert the **src_birth_date** to a **DATE** format:
   ***toDate*(src_birth_date,"yyyy-MM-dd")**
2. Get today's date including the year: ***today*()**

The two values will be calculating the difference and populating the column via the ***dateDiff*** function. This function can work with individual parts of a date (minutes, days, years…) but it cannot process the **STRING** values.

- Within the Column Assigner, create a new assignment line to populate the **age** attribute.
- Fill in the assignment with the following expression:

  dateDiff(**toDate**(**src_birth_date**,"yyyy-MM-dd"),**today**(),"YEAR")

Here's how the result combining all the logic together will look like:



Column Assigner

General | Advanced

Id: Column Assigner

Assignments:

	Column	Expression
1	fix_email	upper(src_email)
2	current_year	datePart(today(),"YEAR")
3	age	dateDiff(toDate(src_birth_date,"yyyy-MM-dd"),today(),"YEAR")
*		

- All set – your **Column Assigner** step is now complete. Click **OK** to return to your plan.

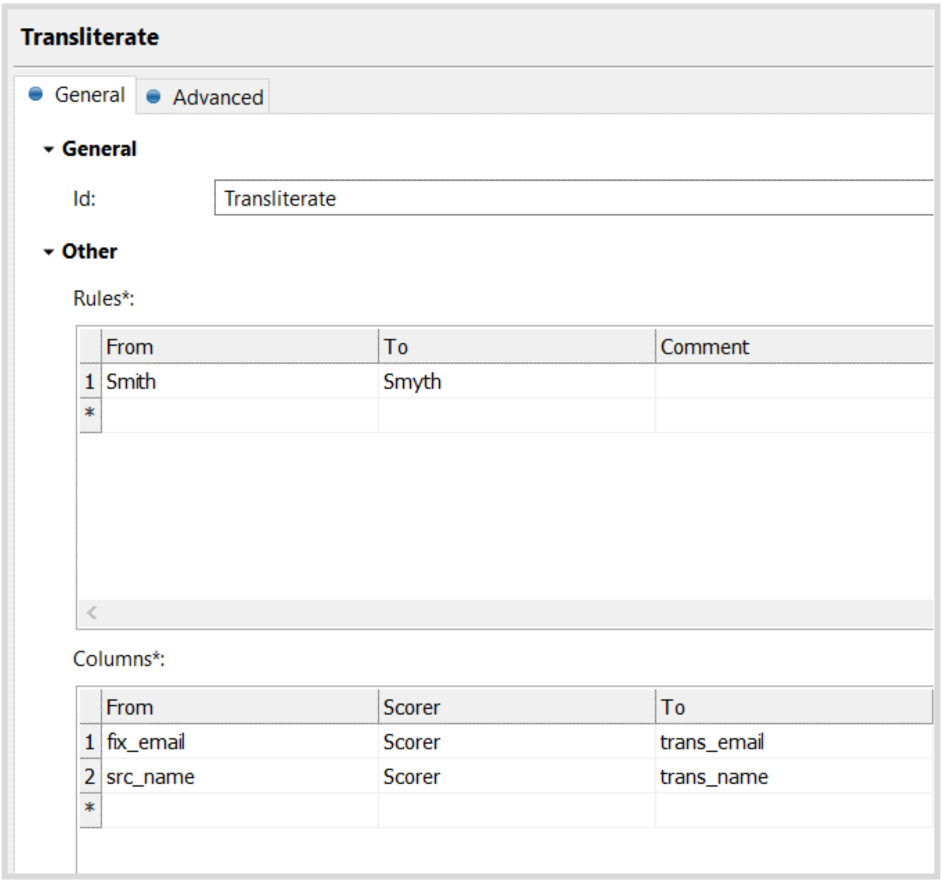
CAUTION

Notice that the birth date values are inconsistent – you don't see the AGE values for all records due to their difference in the date format after converting to DATE.

2.2. Transliterate step

The **Transliterate** step could sometimes be used to replace strings instead of the **replace**() function. This is especially useful if you need to do the same replacement across multiple columns.

- Go back to the **Alter Format** step and add two new columns - **trans_name** (STRING) and **trans_email** (STRING).
- Add the **Transliterate** step into the flow after the **Column Assigner** step.
- Configure the replacement ***Rules*** and applicable columns as follows:



General

Id: Transliterate

Other

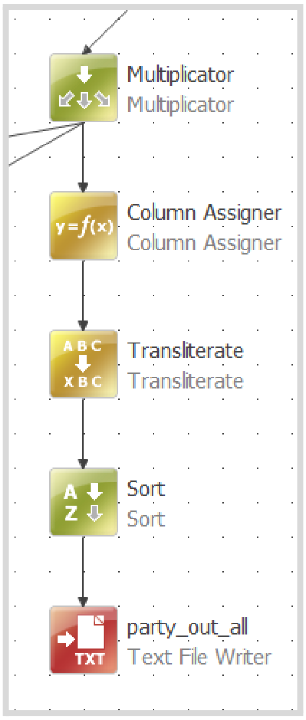
Rules*:

	From	To	Comment
1	Smith	Smyth	
*			

Columns*:

	From	Scorer	To
1	fix_email	Scorer	trans_email
2	src_name	Scorer	trans_name
*			

Zooming into the branch for Stream 1, this is what the plan looks like now:



- **Run** the plan and check the output file's contents to see the changes.

TASK *The **trans_email** attribute does not seem to have the string transliterated! Can you figure out why?*

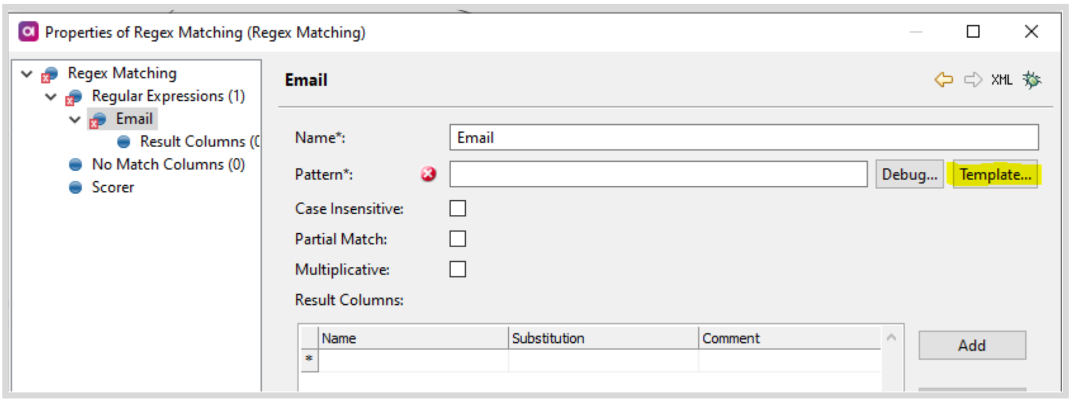
2.3. Regex Matching step

Regex (Regular Expression) is a string of characters that defines a search pattern. The syntax is industry standard and not Ataccama specific. It can be used in parsing values in various formats and extracting values from fields. We are going to extract the username and domain from the values present in **src_email**.

- Add a **Regex Matching** step into your plan between the **Transliterate** and **Sort** steps.
- Add new columns **user** and **domain** into the flow using the **Alter Format** step.
- Open the **Regex Matching** step.
- Set the **Input** as **fix_email**.

Now it is time to add the Regular Expression definition:

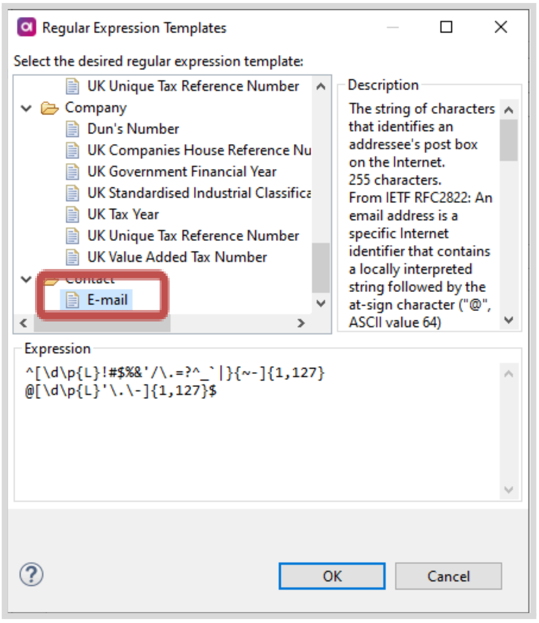
- Enter a ***Name*** for your regex: **"Email."**
- Move one level down in the hierarchy to the details of your **Email** expression. Then click on the **"Template…"** button:



Name*: Email
Pattern*:
Case Insensitive: ☐
Partial Match: ☐
Multiplicative: ☐
Result Columns:

Name	Substitution	Comment

- Scroll down and choose the **Email** option:


Regular Expression Templates

Select the desired regular expression template:

- UK Unique Tax Reference Number
- Company
  - Dun's Number
  - UK Companies House Reference Nu
  - UK Government Financial Year
  - UK Standardised Industrial Classifica
  - UK Tax Year
  - UK Unique Tax Reference Number
  - UK Value Added Tax Number
- Contact
  - E-mail

Description

The string of characters that identifies an addressee's post box on the Internet. 255 characters. From IETF RFC2822: An email address is a specific Internet identifier that contains a locally interpreted string followed by the at-sign character ("@", ASCII value 64)

Expression

```
^[\d\p{L}!#$%&'/\.=?^_`|}{~-]{1,127}
@[\d\p{L}'\.\-]{1,127}$
```

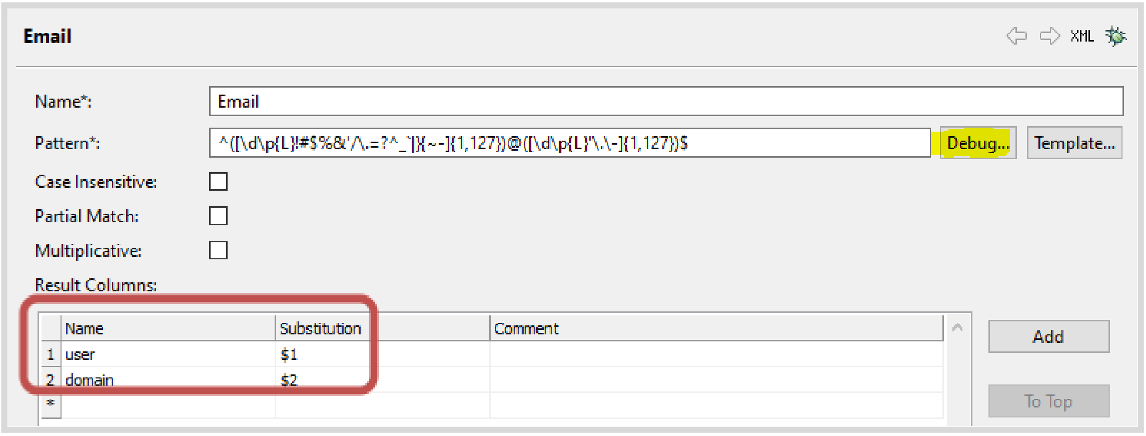
NOTE

You can write your own regular expression of course. In the interest of time, we will use a template today.

- Add extra brackets so the attributes can be grouped ***before*** and ***after*** the '@' symbol. Position them like this:

  ```
  ^([\d\p{L}!#$%&'/\.=?^_`|}{~-]{1,127})@([\d\p{L}'\.\-]{1,127})$
  ```

- Configure the ***Result columns*** on the output like this:


Email

Name*: Email

Pattern*: ^([\d\p{L}!#$%&'/\.=?^_`|}{~-]{1,127})@([\d\p{L}'\.\-]{1,127})$

Case Insensitive: ☐

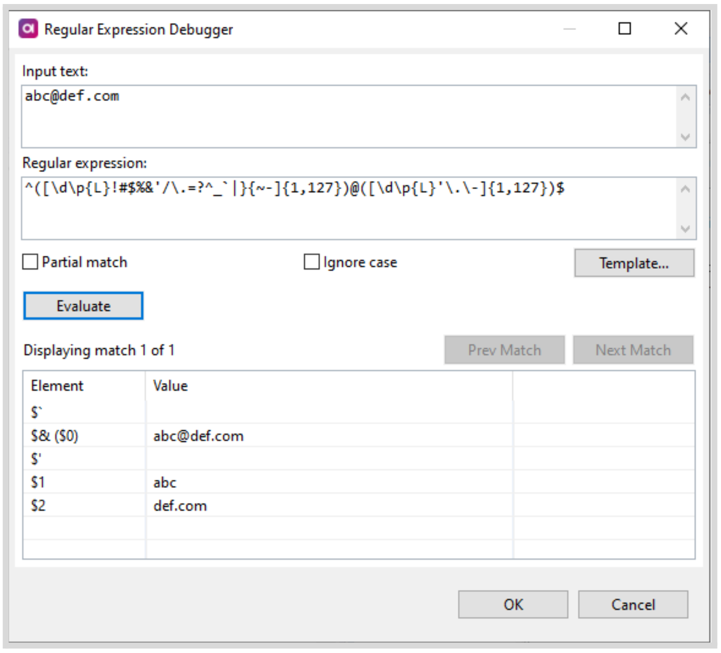
Partial Match: ☐

Multiplicative: ☐

Result Columns:

	Name	Substitution	Comment
1	user	$1	
2	domain	$2	
*			

- You can use the **Debug** mode to enter a few sample values in the ***Input text***. Test what is going to be the output from this step before applying the definition:



Input text:

```
abc@def.com
```

Regular expression:

```
^([\d\p{L}!#$%&'/\.=?^_`|}{~-]{1,127})@([\d\p{L}'\.\-]{1,127})$
```

☐ Partial match ☐ Ignore case

Displaying match 1 of 1

Element	Value
$`	
$& ($0)	abc@def.com
$'	
$1	abc
$2	def.com

NOTE

There are multiple debugging options available – they will be covered in the next exercise.

3. Conclusion

We have come to the end of the Data Transformation workshop. We have covered a few ways to transform or cleanse input data. There are many ways in which data can be transformed, changed, enriched, or cleansed, but in most cases, these modifications are applied via the mechanics we've demonstrated in this exercise.

In the next workshop, we will apply what we have learned about expressions and focus on testing and debugging it so we can be assured the result is achieved as expected.