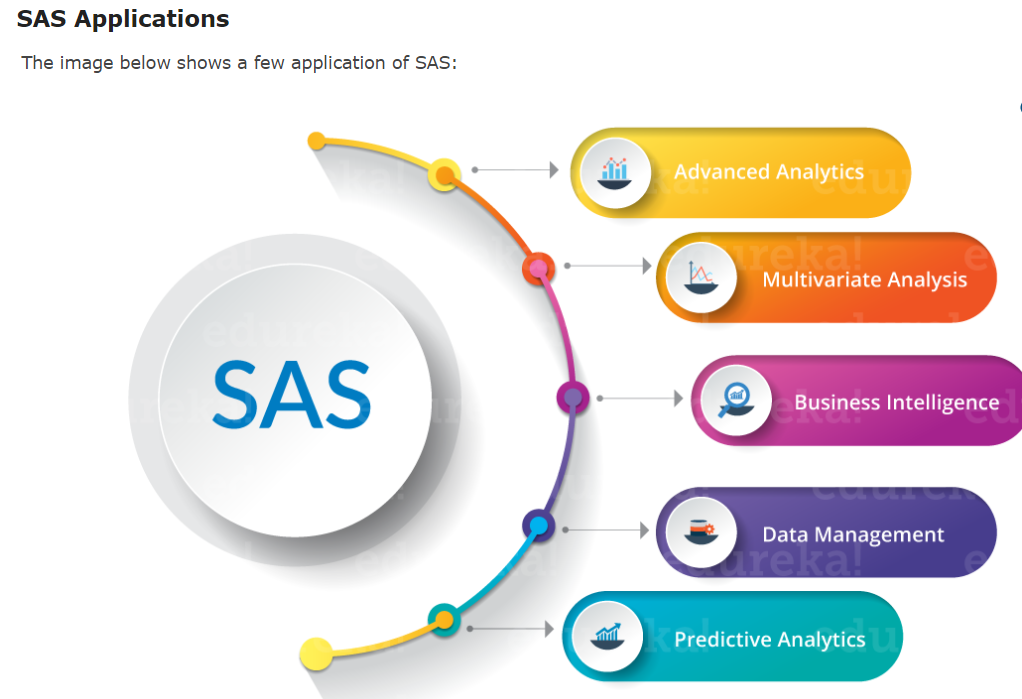
* What is SAS?
* Why SAS? And its applications
* Data, and Proc steps
* SAS Installation and Environment
* SAS first program
* Fundamentals Of SAS Programming
* SAS Code Structure
* Informats And Formats In SAS
* SAS Loops
* Basic Statistical Procedures Using SAS

SAS (Statistical analysis system) is one of the most popular software for data analysis. It is widely used for various purposes such as data management, data mining, report writing, statistical analysis, business modeling, applications development and data warehousing. Knowing SAS is an asset in many job markets. It is tagged 'leader' in Advanced Analytics Platforms as per Gartner 2015 and 2016 reports.



**Installation Steps**

You need to **download two files**. One is virtualization software package and the other one is free SAS version software -

1. Download Oracle VirtualBox / VMware Player 6.0 and later software. ***Download any one of the two***.
   * VMware Player 6.0 or later - Download link
   * Oracle VirtualBox 4.3.12 - Download link
2. Download the SAS University Edition vApp.

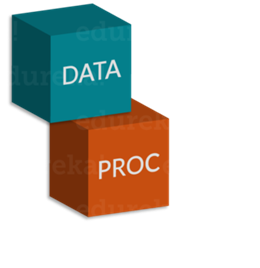
**Download link** - Download Free SAS Software

**SAS Components**

Let us move ahead with our SAS tutorial and take a look at few important SAS components:

* **Base SAS:**It is the most widely used component. It has data management facility. You can do data analysis using Base SAS.
* **SAS/GRAPH:**With the use SAS/Graph you can represent data as graphs. This makes data visualization easy.
* **SAS/STAT:**It lets you perform Statistical analysis, such as Variance, Regression, Multivariate, Survival and Psychometric analysis.
* **SAS/ETS:**It is suited for Time Series Analysis.
* **Data**
* Data is central to every data set. In SAS, data is available in tabular form where variables occupy the column space, and observations occupy the row space.
* **Data types:**
* SAS treats numbers as numeric data and everything else falls under character data. Hence SAS has two data types, numeric and character. Easy, isn’t it?
* DATA step and PROC step form the basic building blocks of a SAS program. What do these building blocks do is what we are going to discuss in this SAS tutorial.

## ****Building Blocks Of SAS****

We start a program with a DATA step to create a SAS data set and then pass the data onto a PROC step. The PROC step processes the data. In order to understand how DATA and PROC steps work, let us consider the below example.

Suppose I wanted to convert a number which is in inches to centimeters and store the result in a variable called ‘size’ and print it, then the DATA step would convert the number in inches to centimeters and PROC step would print the result.

 The image below shows a code snippet for the above mentioned problem:



The statements constitute DATA and PROC steps. The length of a step may vary from one, to more than hundred statements. It is important you remember that DATA steps are used to read and modify data, whereas PROC steps are used to analyse data, perform utility functions, or print reports.

DATA steps begin with the keyword DATA which is followed by a name that you choose for your SAS data set. It is evident that the above DATA step produces a data set named size. DATA steps read data from external data files and may also be used to include loops and case statements. It can be used to merge, sort, combine and concatenate data.

Similarly, procedures start with a PROC statement where the keyword PROC follows the name of the procedure used (for example the name of the procedure may be PRINT, SORT, or MEAN). SAS procedures mostly have a handful of possible statements.

Each time SAS comes across a new step (marked by a DATA or PROC statement), it terminates or ends the previous step and starts with a new one.

While a typical program starts with a DATA step to input or modify data, and then passes the data to a PROC step, it is certainly not the only pattern for mixing DATA and PROC steps. Just as you can stack building blocks in any order, you can arrange DATA and PROC steps in any order. A program could even contain only DATA steps or only PROC steps.

## ****Running A SAS Program****

Now that we have understood how to install SAS University Edition, next in our SAS Tutorial let us take at a sample SAS program.

The code below shows how to print a Fibonacci sequence. In case, if you don’t know what a Fibonacci sequence is, let me define it for you.

The Fibonacci sequence is a set of numbers that starts with a one or a zero, followed by a one, and proceeds based on the rule that, each number (called a Fibonacci number) is equal to the sum of the preceding two numbers. If the Fibonacci sequence is denoted F(n), where n is the first term in the sequence, the following equation shows Fibonacci sequence for n=0, where the first two terms are defined as 0 and 1 by convention:

F (0) = 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 …

In some context, it is customary to use n=1. In that case, the first two terms are defined as 1 and 1 by default, and therefore:

F (1) = 1, 1, 2, 3, 5, 8, 13, 21, 34 …

Let us take a look at this SAS code which generates a Fibonacci sequence that starts with one.

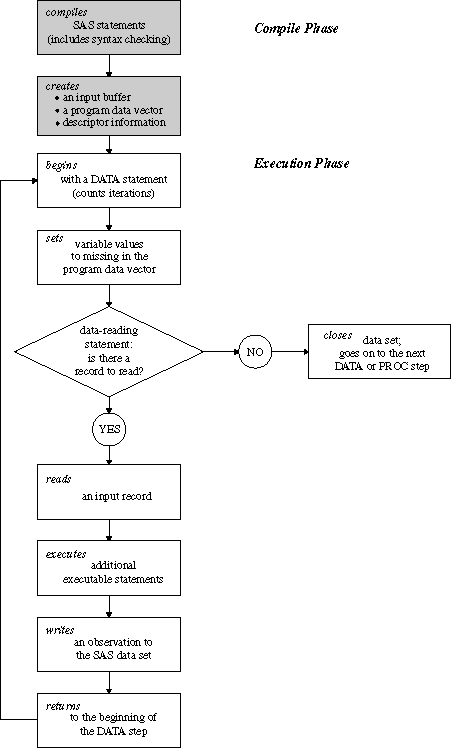
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | DATA Fibonacci;  Do i =1 to 10;  Fib = Sum(Fib, lag(Fib));  if i = 1 then Fib =1;  output;  end;  PROC PRINT Fibonacci;  Run; |

# Overview of DATA Step Processing

|  |
| --- |
|  |
| **Flow of Action** |

When you submit a DATA step for execution, it is first compiled and then executed. The following figure shows the flow of action for a typical SAS DATA step.

***Flow of Action in the DATA Step***



|  |
| --- |
|  |
| **The Compilation Phase** |

When you submit a DATA step for execution, SAS checks the syntax of the SAS statements and compiles them, that is, automatically translates the statements into machine code. In this phase, SAS identifies the type and length of each new variable, and determines whether a type conversion is necessary for each subsequent reference to a variable. During the compile phase, SAS creates the following three items:

|  |  |
| --- | --- |
| input buffer | is a logical area in memory into which SAS reads each record of raw data when SAS executes an INPUT statement. Note that this buffer is created only when the DATA step reads raw data. (When the DATA step reads a SAS data set, SAS reads the data directly into the program data vector.) |
| program data vector (PDV) | is a logical area in memory where SAS builds a data set, one observation at a time. When a program executes, SAS reads data values from the input buffer or creates them by executing SAS language statements. The data values are assigned to the appropriate variables in the program data vector. From here, SAS writes the values to a SAS data set as a single observation.  Along with data set variables and computed variables, the PDV contains two automatic variables, \_N\_ and \_ERROR\_. The \_N\_ variable counts the number of times the DATA step begins to iterate. The \_ERROR\_ variable signals the occurrence of an error caused by the data during execution. The value of \_ERROR\_ is either 0 (indicating no errors exist), or 1 (indicating that one or more errors have occurred). SAS does not write these variables to the output data set. |
| descriptor information | is information that SAS creates and maintains about each SAS data set, including data set attributes and variable attributes. It contains, for example, the name of the data set and its member type, the date and time that the data set was created, and the number, names and data types (character or numeric) of the variables. |

|  |
| --- |
|  |
| **The Execution Phase** |

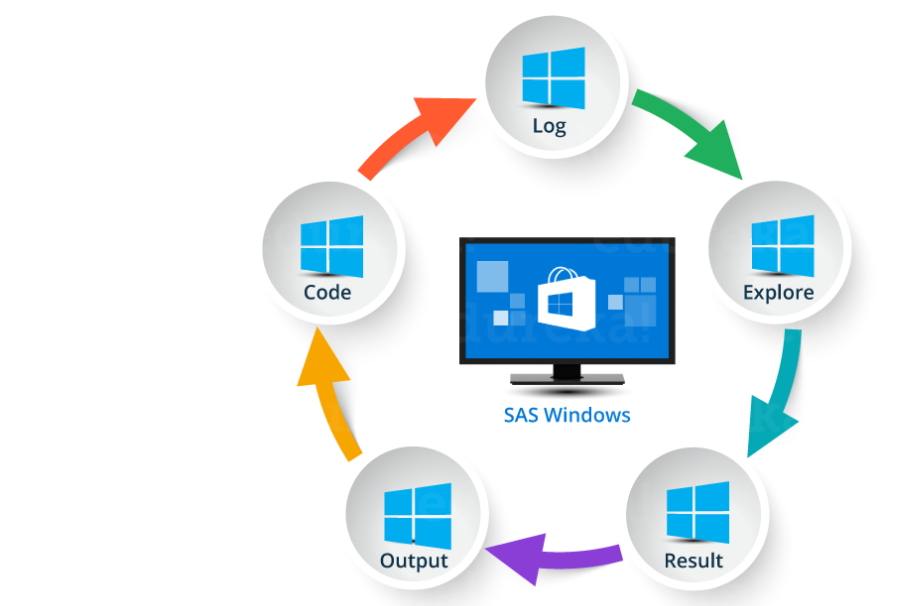
By default, a simple DATA step iterates once for each observation that is being created. The flow of action in the Execution Phase of a simple DATA step is described as follows:

1. The DATA step begins with a DATA statement. Each time the DATA statement executes, a new iteration of the DATA step begins, and the \_N\_ automatic variable is incremented by 1.
2. SAS sets the newly created program variables to missing in the program data vector (PDV).
3. SAS reads a data record from a raw data file into the input buffer, or it reads an observation from a SAS data set directly into the program data vector. You can use an INPUT, MERGE, SET, MODIFY, or UPDATE statement to read a record.
4. SAS executes any subsequent programming statements for the current record.
5. At the end of the statements, an output, return, and reset occur automatically. SAS writes an observation to the SAS data set, the system automatically returns to the top of the DATA step, and the values of variables created by INPUT and assignment statements are reset to missing in the program data vector. Note that variables that you read with a SET, MERGE, MODIFY, or UPDATE statement are not reset to missing here.
6. SAS counts another iteration, reads the next record or observation, and executes the subsequent programming statements for the current observation.
7. The DATA step terminates when SAS encounters the end-of-file in a SAS data set or a raw data file.

## ****Fundamentals Of SAS Programming****

## ****SAS Windows****

Large organisations and training institutes prefer using SAS Windows. SAS Windows has a lot of utilities that help reduce the time required to write codes.



* **Log Window**: It is an execution window. Here, you can check the execution of your program. It also displays errors, warnings and notes.
* **Code Window**: This window is also known as editor window. Consider it as a blank paper or a notepad, where you can write your SAS code.
* **Output Window**: As the name suggests, this window displays the output of the program/ code which you write in the editor.
* **Result Window**: It is an index that list all the outputs of programs that are run in one session. Since it holds the results of a particular session, if you close the software and restart it, the result window will be empty.
* **Explore Window**: It holds the list of all the libraries in the system. You can also browse the system supported files here.

A few organisations use Linux, however, with no graphical user interface you have to write code for every query. Hence it is inconvenient to use.

## ****SAS Data Sets****

SAS data sets are called as data files. Data files constitute of rows and columns. Rows hold observations and columns holdVariable names.

## ****SAS Variables****

SAS has two types of variables:

* **Numeric variables**:This is the default variable type. These variables are used in mathematical expressions.
* **Character variables**:Character variables are used for values that are not used in mathematical expressions.  
  They are treated as text or strings. A variable becomes a character variable by adding a ‘$’ sign at the end of the variable name.

## ****SAS Libraries****

SAS library is a collection of SAS files that are stored in the same folder or directory on your computer.

* **Temporary Library**: In this library, the data set gets deleted when the SAS session ends.
* **Permanent Library**: Data sets are saved permanently. Hence, they are available across sessions.

## ****SAS Programming: SAS Code Structure****

SAS programming is based on two building blocks:

* **DATA Step**: The DATA step creates a SAS data set and then passes the data onto a PROC step
* **PROC Step**: The PROC step processes the data

 A SAS program should follow below mentioned rules:

* Almost every code will begin with either DATA or a PROC Step
* Every line of SAS code ends with a semi colon
* A SAS code ends with RUN or QUIT keyword
* SAS codes are not case sensitive
* You can write a code across different lines or you can write multiple statements in one line

Now that we have seen a few basic terminologies, let us get started with SAS programming with this basic code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | DATA Employee\_Info;  input Emp\_ID Emp\_Name$ Emp\_Vertical$;  datalines;  101 Mak SQL  102 Rama SAS  103 Priya Java  104 Karthik Excel  105 Mandeep SAS  ;  Run; |

In the above code, we created a data set called as Employee\_Info. It has three variables, one numeric variable as Emp\_Id and two character variables as Emp\_Name and Emp\_Verticals. The Run command displays the data set in the Output Window.

Let’s say we want to add employee’s Date of joining to the data set. So we create a variable called as DOJ, give it as input and print the result.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | DATA Employee\_Info;  input Emp\_ID Emp\_Name$ Emp\_Vertical$ DOJ;  datalines;  101 Mak SQL 18/08/2013  102 Rama SAS 25/06/2015  103 Priya Java 21/02/2010  104 Karthik Excel 19/05/2007  105 Mandeep SAS 11/09/2016  ;  Run;  PROC PRINT DATA=Employee\_Info;  Run; |

# SAS - Data Sets

## The data that is available to a SAS program for analysis is referred as a SAS Data Set. It is created using the DATA step. SAS can read a variety of files as its data sources like CSV, Excel, Access, SPSS and also raw data. It also has many in-built data sources available for use. The Data Sets are called temporary Data Set if they are used by the SAS program and then discarded after the session is run. But if it is stored permanently for future use then it is called a permanent Data set. All permanent Data Sets are stored under a specific library. The SAS Data set is stored in form of rows and columns and also referred as SAS Data table. Below we see the examples of permanent Data sets which are in-built as well as red from external sources.

## SAS Built-In Data Sets

These Data Sets are already available in the installed SAS software. They can be explored and used in formulating sample expressions for data analysis. To explore these data sets go to **Libraries -> My Libraries -> SASHELP**. On expanding it we see the list of names of all the built-in Data Sets available.

# SAS - Variables

## SAS Variable Types

SAS has three types of variables as below:

### Numeric Variables

This is the default variable type. These variables are used in mathematical expressions.

### Syntax

INPUT VAR1 VAR2 VAR3; #Define numeric variables in the data set.

In the above syntax, the INPUT statement shows the declaration of numeric variables.

### Example

INPUT ID SALARY COMM\_PERCENT;

### Character Variables

Character variables are used for values that are not used in Mathematical expressions. They are treated as text or strings. A variable becomes a character variable by adding a $ sing with a space at the end of the variable name.

### Syntax

INPUT VAR1 $ VAR2 $ VAR3 $; #Define character variables in the data set.

In the above syntax, the INPUT statement shows the declaration of character variables.

### Example

INPUT FNAME $ LNAME $ ADDRESS $;

### Date Variables

These variables are treated only as dates and they need to be in valid date formats. A variable becomes a date variable by adding a date format with a space at the end of the variable name.

### Syntax

INPUT VAR1 DATE11. VAR2 MMDDYY10. ; #Define date variables in the data set.

In the above syntax, the INPUT statement shows the declaration of date variables.

### Example

INPUT DOB DATE11. START\_DATE MMDDYY10. ;

## Use of Variables in SAS Program

The above variables are used in SAS program as shown in below examples.

### Example

The below code shows how the three types of variables are declared and used in a SAS Program

DATA TEMP;

INPUT ID NAME $ SALARY DEPT $ DOJ DATE9. ;

FORMAT DOJ DATE9. ;

DATALINES;

1 Rick 623.3 IT 02APR2001

2 Dan 515.2 OPS 11JUL2012

3 Michelle 611 IT 21OCT2000

4 Ryan 729 HR 30JUL2012

5 Gary 843.25 FIN 06AUG2000

6 Tusar 578 IT 01MAR2009

7 Pranab 632.8 OPS 16AUG1998

8 Rasmi 722.5 FIN 13SEP2014

;

PROC PRINT DATA=TEMP;

RUN;

# SAS - Strings

## Declaring String Variables

We can declare the string variables and their values as shown below. In the code below we declare two character variables of lengths 6 and 5. The LENGTH keyword is used for declaring variables without creating multiple observations.

data string\_examples;

LENGTH string1 $ 6 String2 $ 5;

/\*String variables of length 6 and 5 \*/

String1 = 'Hello';

String2 = 'World';

Joined\_strings = String1 ||String2 ;

run;

proc print data = string\_examples noobs;

run;

# SAS - Arrays

## Syntax

In SAS an array is declared by using the following syntax:

ARRAY ARRAY-NAME(SUBSCRIPT) ($) VARIABLE-LIST ARRAY-VALUES

In the above syntax:

* **ARRAY** is the SAS keyword to declare an array.
* **ARRAY-NAME** is the name of the array which follows the same rule as variable names.
* **SUBSCRIPT** is the number of values the array is going to store.
* **($)** is an optional parameter to be used only if the array is going to store character values.
* **VARIABLE-LIST** is the optional list of variables which are the place holders for array values.
* **ARRAY-VALUES** are the actual values that are stored in the array. They can be declared here or can be read from a file or dataline.

### Examples of Array Declaration

Arrays can be declared in many ways using the above syntax. Below are the examples.

# Declare an array of length 5 named AGE with values.

ARRAY AGE[5] (12 18 5 62 44);

# Declare an array of length 5 named COUNTRIES with values starting at index 0.

ARRAY COUNTRIES(0:8) A B C D E F G H I;

# Declare an array of length 5 named QUESTS which contain character values.

ARRAY QUESTS(1:5) $ Q1-Q5;

# Declare an array of required length as per the number of values supplied.

ARRAY ANSWER(\*) A1-A100;

## Accessing Array Values

The values stored in an array can be accessed by using the **print** procedure as shown below. After it is declared using one of the above methods, the data is supplied using DATALINES statement.

DATA array\_example;

INPUT a1 $ a2 $ a3 $ a4 $ a5 $;

ARRAY colours(5) $ a1-a5;

mix = a1||'+'||a2;

DATALINES;

yello pink orange green blue

;

RUN;

PROC PRINT DATA=array\_example;

RUN;

# SAS - Operators

* Arithmetic Operators
* Logical Operators
* Comparison Operators
* Minimum/Maximum Operators
* Concatenation Operator

We will look at each of the one by one. The operators are always used with variables that are part of the data that is being analyzed by the SAS program.

## Arithmetic Operators

The below table describes the details of the arithmetic operators. Let’s assume two data variables **V1** and **V2**with values **8** and **4** respectively.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Addition | V1+V2=12 |
| - | Subtraction | V1-V2=4 |
| \* | Multiplication | V1\*V2=32 |
| / | Division | V1/V2=2 |
| \*\* | Exponentiation | V1\*\*V2=4096 |

### Example

DATA MYDATA1;

input @1 COL1 4.2 @7 COL2 3.1;

Add\_result = COL1+COL2;

Sub\_result = COL1-COL2;

Mult\_result = COL1\*COL2;

Div\_result = COL1/COL2;

Expo\_result = COL1\*\*COL2;

datalines;

11.21 5.3

3.11 11

;

PROC PRINT DATA=MYDATA1;

RUN;

## Logical Operators

The below table describes the details of the logical operators. These operators evaluate the Truth value of an expression. So the result of logical operators is always a 1 or a 0. Let’s assume two data variables **V1** and **V2**with values **8**and **4** respectively.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| & | The AND Operator. If both data values evaluate to true then the result is 1 else it is 0. | (V1>2 & V2 > 3) gives 0. |
| | | The OR Operator. If any one of the data values evaluate to true then the result is 1 else it is 0. | (V1>9 & V2 > 3) is 1. |
| ~ | The NOT Operator. The result of NOT operator in form of an expression whose value is FALSE or a missing value is 1 else it is 0. | NOT(V1 > 3) is 1. |

### Example

DATA MYDATA1;

input @1 COL1 5.2 @7 COL2 4.1;

and\_=(COL1 > 10 & COL2 > 5 );

or\_ = (COL1 > 12 | COL2 > 15 );

not\_ = ~( COL2 > 7 );

datalines;

11.21 5.3

3.11 11.4

;

PROC PRINT DATA=MYDATA1;

RUN;

## Comparison Operators

The below table describes the details of the comparison operators. These operators compare the values of the variables and the result is a truth value presented by 1 for TRUE and 0 for False. Let’s assume two data variables **V1**and **V2**with values **8** and **4** respectively.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | The EQUAL Operator. If both data values are equal then the result is 1 else it is 0. | (V1 = 8) gives 1. |
| ^= | The NOT EQUAL Operator. If both data values are unequal then the result is 1 else it is 0. | (V1 ^= V2) gives 1. |
| < | The LESS THAN Operator. | (V2 < V2) gives 1. |
| <= | The LESS THAN or EQUAL TO Operator. | (V2 <= 4) gives 1. |
| > | The GREATER THAN Operator. | (V2 > V1) gives 1. |
| >= | The GREATER THAN or EQUAL TO Operator. | (V2 >= V1) gives 0. |
| IN | The IN Operator. If the value of the variable is equal to any one of the values in a given list of values, then it returns 1 else it returns 0. | V1 in (5,7,9,8) gives 1. |

### Example

DATA MYDATA1;

input @1 COL1 5.2 @7 COL2 4.1;

EQ\_ = (COL1 = 11.21);

NEQ\_= (COL1 ^= 11.21);

GT\_ = (COL2 => 8);

LT\_ = (COL2 <= 12);

IN\_ = COL2 in( 6.2,5.3,12 );

datalines;

11.21 5.3

3.11 11.4

;

PROC PRINT DATA=MYDATA1;

RUN;

## Minimum/Maximum Operators

The below table describes the details of the Minimum/Maximum operators. These operators compare the values of the variables across a row and the minimum or maximum value from the list of values in the rows is returned.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| MIN | The MIN Operator. It returns the minimum value form the list of values in the row. | MIN(45.2,11.6,15.41) gives 11.6 |
| MAX | The MAX Operator. It returns the maximum value form the list of values in the row. | MAX(45.2,11.6,15.41) gives 45.2 |

### Example

DATA MYDATA1;

input @1 COL1 5.2 @7 COL2 4.1 @12 COL3 6.3;

min\_ = MIN(COL1 , COL2 , COL3);

max\_ = MAX( COL1, COl2 , COL3);

datalines;

11.21 5.3 29.012

3.11 11.4 18.512

;

PROC PRINT DATA=MYDATA1;

RUN;

## Concatenation Operator

The below table describes the details of the Concatenation operator. This operator concatenates two or more string values. A single character value is returned.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| || | The concatenate Operator. It returns the concatenation of two or more values. | 'Hello'||' World' gives Hello World |

### Example

DATA MYDATA1;

input COL1 $ COL2 $ COL3 $;

concat\_ = (COL1 || COL2 || COL3);

datalines;

Tutorial s point

simple easy learning

;

PROC PRINT DATA=MYDATA1;

RUN;

Operators Precedence

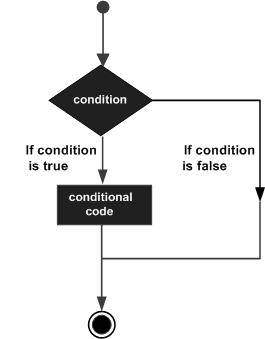
The operator precedence indicates the order of evaluation of the multiple operators present in complex expression. The below table describes the order of precedence with in a group of operators.

|  |  |  |
| --- | --- | --- |
| **Group** | **Order** | **Symbols** |
| Group I | Right to Left | \*\* + - NOT MIN MAX |
| Group II | Left to Right | \* / |
| Group III | Left to Right | + - |
| Group IV | Left to Right | || |
| Group V | Left to Right | < <= = >= > |

# SAS - Decision Making

Decision making structures require the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be **true**, and optionally, other statements to be executed if the condition is determined to be **false**.

Following is the general form of a typical decision making structure found in most of the programming languages:



SAS provides following types of decision making statements. Click the following links to check their detail.

## ****Informats And Formats In SAS****

It is important that you understand this topic well if you want to be good at SAS programming. If you can recall, I mentioned earlier that SAS has two standard variable types:

* Numeric
* Character

When SAS comes across non standard variables, SAS will throw an error or you won’t get the desired output. To overcome this problem, SAS uses Informats and Formats.

## ****Informat****

Informats are typically used to read or input data from external files or flat files (like text files or sequential files). The informat instructs SAS on how to read data into SAS variables. SAS  has three types of Informats: character, numeric, and date/ time. Informats are named according to the following syntax structure:

* Character Informat: $INFORMATw.
* Numeric Informat: INFORMATw.d
* Date/ Time Informat: INFORMATw.

The ‘$’ indicates a character informat. INFORMAT refers to the sometimes optional SAS informat name. The ‘w’ indicates the width (bytes or number of columns) of the variable. The ‘d’ is used for numeric data to specify the number of digits to the right of the decimal place. All informats must contain a decimal point(.) so that SAS can  
differentiate an informat from a SAS variable.

Let us go back to our previous code and see if Date/ Time Informat helps us. So let’s change the code accordingly and add a Date Informat to it as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | DATA Employee\_Info;  input Emp\_ID Emp\_Name$ Emp\_Vertical$ DOJ;  INFORMAT DOJ ddmmyy10.;  datalines;  101 Mak SQL 18/08/2013  102 Rama SAS 25/06/2015  103 Priya Java 21/02/2010  104 Karthik Excel 19/05/2007  105 Mandeep SAS 11/09/2016  ;  Run;  PROC PRINT DATA=Employee\_Info;  Run; |

Line number 3 in the code instructs SAS to read in the variable ‘date of joining’ (DOJ) using the date  
informat MMDDYYw. For each date field occupies 10 spaces, the ‘w.’ qualifier is set to 10.

## ****Format****

Informats are the instructions for reading data, whereas formats are the instructions used to display or output data. Defining a format for a variable is how you tell SAS to display the values in the variable. Formats are grouped into the same three classes as informats (character, numeric, and date-time) and also always contain a dot.

The general form of a format statement is:

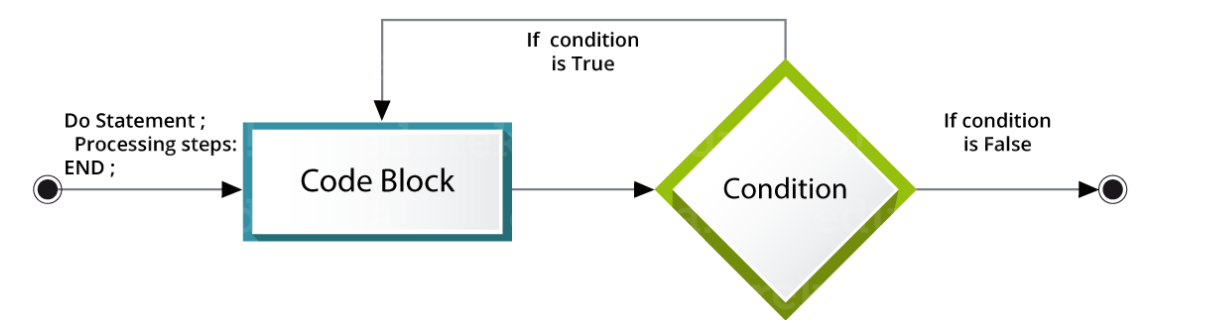
* FORMAT variable-name FORMAT-NAME.;

Let us go back to our code having dataset Employee\_Info to see if we can display the date correctly using FORMAT command.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | DATA Employee\_Info;  input Emp\_ID Emp\_Name$ Emp\_Vertical$ DOJ;  INFORMAT DOJ ddmmyy10.;  FORMAT DOJ ddmmyy10.;  datalines;  101 Mak SQL 18/08/2013  102 Rama SAS 25/06/2015  103 Priya Java 21/02/2010  104 Karthik Excel 19/05/2007  105 Mandeep SAS 11/09/2016  ;  Run;  PROC PRINT DATA=Employee\_Info;  Run; |

## ****SAS Loops****

While doing SAS programming, we may encounter situations where we repeatedly need to execute a block of code several number of times. It is inconvenient to write the same set of statements again and again. This is where loops come into picture. In SAS, the Do statement is used to implement loops. It is also known as the Do Loop. The image below shows the general form of the Do loop statements in SAS.



Following are the  types of DO loops in SAS:

* **Index**: The loop continues from the start value till the stop value of the index variable.
* **While**: The loop continues as long as the **While** condition becomes false.
* **Until**: The loop continues till the **Until** condition becomes True.

## ****Do Index loop****

We use an index variable as a start and stop value for **Do Index loop**. The SAS statements get executed repeatedly till the index variable reaches its final value.  
**Syntax:**

|  |  |
| --- | --- |
| 1  2  3 | Do indexvariable = initialvalue to finalvalue;  SAS statements;  End; |

Let us take a look at sample code to understand Do Index Loop. In the below code, VAR is the index variable.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | DATA SampleLoop;  SUM=0;  Do VAR = 1 to 10;  SUM = SUM + VAR;  END;  PROC PRINT DATA = SampleLoop;  Run; |

**Do While Loop**

The **Do While** loop uses a WHILE condition. This Loop executes the block of code when the condition is true and keeps executing it, till the condition becomes false. Once the condition becomes false, the loop is terminated.

**Syntax:**

|  |  |
| --- | --- |
| 1  2  3 | Do While (condition);  SAS statements;  End; |

Following sample code will help you understand DO WHILE loop.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | DATA SampleLoop;  SUM=0;  VAR=1;  Do While(VAR<15);  SUM = SUM + VAR;  VAR+1;  END;  PROC PRINT DATA = SampleLoop;  Run; |

**Do Until Loop**

The Do Until loop uses an **Until** condition.This Loop executes the block of code when the condition is false and keeps executing it, till the condition becomes true. Once the condition becomes true, the loop is terminated.

**Syntax:**

|  |  |
| --- | --- |
| 1  2  3 | Do Until (condition);  SAS statements;  END; |

Let us take a look at sample program.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | DATA SampleLoop;  SUM=0;  VAR=1;  Do Until(VAR>15);  SUM=SUM+VAR;  VAR+1;  END;  PROC PRINT;  Run; |