```
*SAS MACROS;
/*
SAS has a powerful programming feature called Macros which allows us to avoid
repetitive sections of code and to use them again and again when needed.
It also helps create dynamic variables within the code that can take different values
for different run instances of the same code.
Macros can also be declared for blocks of code which will be reused multiple times
in a similar manner to macro variables.
We will see both of these in the below examples.
*/

*Macro variables;
/*
These are the variables which hold a value to be used again and again by a
SAS program. They are declared at the beginning of a SAS program and called out
later in the body of the program.
They can be Global or Local in scope.
*/

*Global Macro variable;
/*
They are called global macro variables because they can accessed by any SAS program
available in the SAS environment. In general they are the system assigned
variables which are accessed by multiple programs.
A general example is the system date.
*/

/*
Below is a example of the SAS variable called SYSDATE which represents the
system date. Consider a scenario to print the system date in the title of the
SAS report every day the report is generated. The title will show the current date
 and day without we coding any values for them.
We use the in-built SAS data set called CARS available in the SASHELP library.
*/

proc print data = sashelp.cars;
where make = 'Audi' and type = 'Sports' ;
   TITLE "Sales as of &SYSDAY &SYSDATE";
run;

*Local Macro variable;
/*
These variables can be accessed by SAS programs in which they are declared as
part of the program. They are typically used to supply different variables to the
same SAS statements sl that they can process different observations of a data set.
*/

*Syntax: % LET (Macro Variable Name) = Value;

/*
The variables are used by the SAS statements using the & character appended at the
beginning of the variable name. Below program gets us all the observation of the
make 'Audi' and type 'Sports'. In case we want the result of different make,
we need to change the value of the variable make_name without changing any other
part of the program. In case of bring programs this variable can be referred again
and again in any SAS statements.
*/

%LET make_name = 'Audi';
%LET make_type = 'Sports';
proc print data = sashelp.cars;
where make = &make_name and type = &make_type;
TITLE "Sales as of &SYSDay &SYSDATE &SYSTIME";
run;

*Macro Programs;
/*
Macro is a group of SAS statements that is referred by a name and to use it in
```

```
program anywhere, using that name.
It starts with a %MACRO statement and ends with %MEND statement.
Syntax:
# Creating a Macro program.
%MACRO <macro name>(Param1, Param2,….Paramn);

Macro Statements;

%MEND;

# Calling a Macro program.
%MacroName (Value1, Value2,…..Valuen);

*/
```

```
%MACRO show_result(make_ , type_);
proc print data = sashelp.cars;
where make = "&make_" and type = "&type_" ;
*remember the quotes and & in the above code;
   TITLE "Sales as of &SYSDAY &SYSDATE";
run;
%MEND;

%show_result(Audi,Sports);

*Commonly Used Macros;
/*
SAS has many MACRO statements which are in-built in the SAS programming language.
They are used by other SAS programs without explicitly declaring them.
*/

*Macro %PUT - This macro statement writes text or macro variable information to
the SAS log.;
```

```
data _null_;
call symput('today',
TRIM(PUT("&sysdate"d,worddate22.)));
run;
%put &today;

*Macro %RETURN - Execution of this macro causes normal termination of the currently
executing macro when certain condition evaluates to be true.;

/*
In the below examplewhen the value of the variable "val" becomes 10,
the macro terminates else it contnues.
*/
```

```
%macro check_condition(val);
   %if &val = 10 %then %return;

   data p;
      x = 34.2;
   run;

%mend check_condition;

%check_condition(11);

*Macro %END;

/*
This macro definition contains a %DO %WHILE loop that ends, as required,
with a %END statement. In the below example the macro named test takes a
user input and runs the DO loop using this input value. The end of DO loop
is achieved through the
%end statement while the end of macro is achieved through %mend statement.
*/
```

```
%macro test(finish);
```

```
    %let i = 1;
    %do %while (&i <&finish);
       %put the value of i is &i;
       %let i=%eval(&i+1);
    %end;
 %mend test;
 %test(5)
```