```
   /* Section 1 - Accessing Data Using SQL */
proc sql noprint;
   create table base_1 as
   select name, team, natbat
   from sashelp.baseball;quit;
/* • Create new calculated columns. */
proc sql noprint;
   create table base_2 as
   select *, 100*(crhome/cratbat)
   from sashelp.baseball;
quit;
/* • Assign an alias with the AS keyword. */
proc sql noprint;
   create table base_3 as
   select *, 100*(crhome/cratbat) as crHomeRunPerc
   from sashelp.baseball;
quit;
/* • Use case logic to select values for a column. */
proc sql noprint;
   create table base_4 as
   select *, 100*(crhome/cratbat) as crHomeRunPerc,
   case
     when calculated crHomeRunPerc > 5 then 'high'
     when calculated crHomeRunPerc < 5 and calculated crHomeRunPerc > 1 then 'medium'
     else 'low' end as crHomeRunClass
   from sashelp.baseball;
quit;
/* • Retrieve rows that satisfy a condition with a WHERE clause. */
proc sql noprint;
   create table cars_0 as
   select *
   from sashelp.cars
   where Type = 'Sports';
quit;

/* • Subset data by calculated columns. */
proc sql noprint;
   create table cars_1 as
   select *, msrp*0.78 as gbPrice
   from sashelp.cars
   where calculated gbPrice lt 40000;quit;
/* • Join tables - inner joins, full joins (coalesce function), right joins, left joins. */
proc sql noprint;
   create table join_1 as
   select a.*, b.density_2010
   from sashelp.prdsal2 a, sashelp.us_data b
   where a.state = b.statename;quit;
proc sql noprint;
   create table join_2 as
   select a.*, b.density_2010
   from sashelp.prdsal2 a inner join sashelp.us_data b
   on a.state = b.statename;quit;
proc sql noprint;
   create table coalesce_example as
   select statename as state,
          coalesce(reps_1910, reps_1920, reps_1940, reps_1950, reps_1960) as reps
   from sashelp.us_data;quit;
proc sql noprint;
   create table full_join as
   select a.*, b.*, coalesce(a.mapidname, b.statename) as new_state_name
   from sashelp.gcstate a full join sashelp.us_data b
   on a.mapidname = b.statename
   order by calculated new_state_name;quit;
proc sql noprint;
   create table right_join as
   select a.*, b.*
   from sashelp.gcstate a right join sashelp.us_data b
   on a.mapidname = b.statename;quit;
```

```
proc sql noprint;
   create table left_join as
   select a.*, b.*
   from sashelp.gcstate a left join sashelp.us_data b
   on a.mapidname = b.statename;quit;
/* • Combine tables using set operators - union, outer union, except, intersect. */
proc sql noprint;
   create table union_example as
   select *
   from sashelp.prdsal2(where=(state='California'))
   union
   select *
   from sashelp.prdsal2(where=(state='Quebec'));quit;
proc sql noprint;
   create table outer_union_example as
   select *
   from sashelp.prdsal2(where=(state='California'))
   outer union
   select *
   from sashelp.prdsal3;
quit;
proc sql noprint;
   create table except1 as
   select *
   from sashelp.prdsal2
   except
   select *
   from sashelp.prdsal2(where=(state='California'));
quit;
proc sql noprint;
   create table except2 as
   select *
   from sashelp.prdsal2
   except
   select *
   from sashelp.prdsal2(where=(state ne 'California'));
quit;
proc sql noprint;
   create table intersect_example as
   select *
   from sashelp.prdsal2(where=(state = 'Saskatchewan'))
   intersect
   select *
   from sashelp.prdsal2(where=(state in ('California','Oregon','Saskatchewan')));
quit;
/* • Sort data with an ORDER BY clause. */
proc sql noprint;
   create table cars_3 as
   select *, msrp*0.78 as gbPrice
   from sashelp.cars
   where calculated gbPrice lt 40000
   order by calculated gbPrice desc;
quit;

/* • Assign labels and formats to columns. */

proc sql noprint;
   create table cars_4 as
   select *, msrp*0.78 as gbPrice "MSRP (in Pound Sterling)" format nlmnlgbp32.2
   from sashelp.cars
   where calculated gbPrice lt 40000
   order by calculated gbPrice desc;
quit;

/* Generate summary reports by working with a single table, joining tables, or using
Summarize data across and down columns using summary functions (AVG, COUNT,MAX, MIN, SUM). */

proc sql noprint;
   create table cars_5 as
```

```
   select max(msrp) as max_msrp, *
   from sashelp.cars;
quit;

proc sql noprint;
   create table cars_6 as
   select max(msrp) as max_msrp, make, model, msrp, max(msrp)-msrp as max_msrp_diff
   from sashelp.cars;
quit;

/* • Group data using GROUP BY clause. */
proc sql noprint;
   create table cars_7 as
   select max(msrp) as max_msrp, make, model, msrp, origin
   from sashelp.cars
   group by origin;
quit;
/* • Filter grouped data using HAVING clause. */
proc sql noprint;
   create table cars_8 as
   select max(msrp) as max_msrp, make, model, msrp
   from sashelp.cars
   having msrp = max(msrp);
quit;

/* • Eliminate duplicate values with the DISTINCT keyword. */
proc sql noprint;
   create table cars_9 as
   select distinct make, origin
   from sashelp.cars;
quit;

/* Construct sub-queries and in-line views within an SQL procedure step. */
proc sql;
   create table cars_10 as
   select *
   from sashelp.cars
   where msrp gt
     (select avg(msrp)
     from sashelp.cars) + 1000;
quit;

/* • Subset data by using non-correlated subqueries. */
proc sql noprint;
   create table non_corr_subq as
   select a.*, b.density_2010, c.avg_density_2010
   from sashelp.prdsal2 a, sashelp.us_data b,
     (select statename, avg(density_2010) as avg_density_2010 from sashelp.us_data) c
   where a.state = b.statename and c.statename = b.statename
   order by b.density_2010;
quit;

/* • Reference an in-line view with other views or tables (multiple tables). */
proc sql;
   create table cars_11 as
   select *
   from sashelp.cars
   where msrp between
     (select avg(msrp)
     from sashelp.cars) - 1000
   and
     (select avg(msrp)
     from sashelp.cars) + 1000;
quit;

/* Use SAS SQL procedure enhancements.
 Use SAS data set options with PROC SQL (KEEP=, DROP=, RENAME=, OBS=). */
proc sql noprint;
   create table class_1 as
```

```
   select *
   from sashelp.class(keep=name sex age where=(age_renamed ge 15) rename=(age=age_renamed));
quit;

/* • Use PROC SQL invocation options (INOBS=, OUTOBS=. NOPRINT, NUMBER) */
proc sql noprint inobs=5;
   create table class_2 as
   select *
   from sashelp.class;
quit;

proc sql outobs=5 number;
   create table class_3 as
   select *
   from sashelp.class
   order by name descending;
quit;

/* • Use SAS functions (SCAN, SUBSTR, LENGTH). */
proc sql noprint;
   create table baseball_5 as
   select scan(name, 1) as lastName, scan(name, 2) as firstName,
          cat(substr(scan(name,2),1,1), substr(name,1,1)) as initials length=2,
          length(scan(name,1)) as lastNameLength, *
   from sashelp.baseball;
quit;

/* • Access SAS system information by using DICTIONARY tables (members, tables,columns) */

proc sql noprint;
   create table dict as
   select *
   from dictionary.columns
   where libname = 'SASHELP' and memname = 'CARS';
quit;

/* • Use the CALCULATED keyword. */
proc sql noprint;
   create table cars_12 as
   select *, msrp*0.78 as gbPrice
   from sashelp.cars
   where calculated gbPrice lt 40000;
quit;

/*  2 - Macro Processing
 Create and use user-defined and automatic macro variables within the SAS Macro
 Language. */

%let name = Pat;
%put name = &name;
%put date = &sysdate;

/* • Define and use macro variables. */
%let carMake = Acura;
proc sql noprint;
   create table cars_13 as
   select *
   from sashelp.cars
   having make eq "&carMake";
quit;

/* • Use macro variable name delimiter. (.) */
%let name = Pat;
%put name = &name.&name.;

/*  Use INTO clause of the SELECT statement in SQL to create a single variable or a list of
 variables. */
proc sql noprint;
   select make, model, msrp
```

```
   into :most_expensive_make, :most_expensive_model, :most_expensive_msrp
   from sashelp.cars
   having msrp = max(msrp);
quit;

%put most_expensive_make = &most_expensive_make.;
%put most_expensive_model = &most_expensive_model.;
%put most_expensive_msrp = &most_expensive_msrp.;
```

```
proc sql noprint;
   select distinct make
   into :make_list separated by ', '
   from sashelp.cars;
quit;

%put make_list = &make_list.;

/* Use the SYMPUTX routine in a DATA Step to create a single variable or a list of
 variables. */
```

```
data _null_;
   set sashelp.class(where=(sex='M'));
   call symputx(Name,Age);
run;

%put "Henry's age is &Henry..";

/* • Control variable scope with:  %GLOBAL statement , %LOCAL statement */
```

```
%macro demo1;
   %global x;
/*    %local x; */
   %let x = %sysfunc(rand(Integer,1,10));
   %put x =&x.;
%mend demo1;
%demo1;
%put x =&x.;
```

```
data _null_;
   set sashelp.class(where=(sex='M'));
   call symputx(Name,Age,'g');
run;
%put "Henry's age is &Henry..";


/* Automate programs by defining and calling macros using the SAS Macro Language.
 Calling a macro with and without parameters.
 Document macro functionality with comments
 Generate SAS Code conditionally by using the %IF-%THEN-%ELSE macro statements or
 iterative %DO statements.
 Use the SAS AUTOCALL facility to permanently store and call macros.
 Use macro functions.
 NOTES ON AUTOCALL FACILITY */
 libname maclib "/home/u60094620/advancedsas";
 options mstored sasmstore=maclib mautosource;
```

```
 %macro mystored_macro / STORE;
    %put calling my stored macro;
 %mend;

/* • Use macro functions. (%SCAN, %SUBSTR, %UPCASE) and quoting functions. (%NRSTR, %STR) */
%macro sashelp_dsn(dsn=%str(), /*dataset name*/
                   about=%str() /*description of the dataset*/);
   /*Reads in sashelp library datasets*/
   %if (%upcase(&dsn.) eq CARS) or (%upcase(&dsn.) eq PRDSALE) %then %do;
      %let financial = This dataset concerns financial data;
      %put %nrstr(%upcase(&dsn.)) = %upcase(&dsn.);
   %end;

   %else %do;
```

```
      %let financial = ;
      %put %nrstr(%upcase(&dsn.)) = %upcase(&dsn.);
   %end;

   data &dsn.;
      set sashelp.&dsn;
   run;

   title "&about.";
   title2 "&financial.";
   /*printing out the first 5 observations from the selected dataset*/
   proc print data=&dsn. (obs=5);
   run;
   title;
   title2;
%mend sashelp_dsn;
%sashelp_dsn(dsn=%str(cars),
             about=%str(Invenetory of a car dealership lot));

%sashelp_dsn(dsn=%str(iris),
             about=%str(Fishers famous dataset which contains 4 classes of instances where each class refers to a

/* Use macro evaluation functions. (%SYSEVALF)
 Use %SYSFUNC to execute DATA step functions within the SAS Macro Language.
 Debug macros.
 Trace the flow of execution with the MLOGIC option.
 Examine the generated SAS statements with the MPRINT option.
 Examine macro variable resolution with the SYMBOLGEN option.
 Use the %PUT statement to print information to the log. */
options mprint mlogic symbolgen;
%macro system_options_demo;
   proc sql noprint;
      create table months_years
         (year num(4), month num(2) format z2., row_num num(4));
      %let i = 0;
      %do g = 2000 %to 2023;
         %do h = 1 %to 12;
            %let i = %eval(&i + 1);
            insert into months_years
               values(&g.,&h.,&i.);
         %end;
      %end;
         select row_num, year, row_num-48
         into :max_row_num, :max_year, :start_row_num
         from months_years
         having row_num = max(row_num);

         select put(mdy(month,1,year),monyy7.)
         into :month_year_list separated by ', '
         from months_years(where=(row_num ge %eval(&max_row_num-48)));
    quit;
    %let k = 0;
    %do j = %eval(&start_row_num.) %to %eval(&max_row_num.);
       %let k = %eval(&k. + 1);

       proc sql noprint;
 select month, year into :month_&j., :year_&j.
 from months_years where row_num = %eval(&j.);quit;

       %put "month_&j. resolves to %cmpres(&&month_&j.)";
/*     %put "month_year_list resolves to %upcase(%scan(%superq(&month_year_list), &k., %str(,)))"; */
          %end;
%mend system_options_demo;
%system_options_demo;

/* Create data-driven programs using SAS Macro Language.
Create a series of macro variables.
 Use indirect reference to macro variables. (&&, etc.)
 Incorporate DICTONARY tables in data driven macros.
```

```
 Generate repetitive macro calls. */

%macro dict_example;
   %do i = 1 %to 20;
 proc sql noprint;
 select name into :col_name_&i from dictionary.columns
 where libname = "SASHELP" and memname = "BASEBALL" and varnum = %eval(&i.);
     quit;
 %put "The &i.th column is %cmpres(&&col_name_&i)";
   %end;
%mend dict_example;
%dict_example;

/*  3 - Advanced Techniques
 Process data using 1 and 2 dimensional arrays.
 • Define and use character arrays & numeric arrays.
 • Create variables with arrays.
 • Reference arrays within a DO loop.
 • Specify the array dimension with the DIM function.
 • Define arrays as temporary arrays.
 • Load initial values for an array from a SAS data set. */
data array_example_1;
   array lower_case{*} $ n1-n5;
   array upper_case{*} $ c1-c5;
   input lower_case{*};
      do i=1 to 5;
         upper_case{i} = upcase(lower_case{i});
      end;
   datalines;
   bob jeff tom frank hank;run;
data array_example_2;
   array population{2,5} c1p1-c1p5 c2p1-c2p5;
   input c1p1-c1p5 c2p1-c2p5;
   do i=1 to 2;
     do j=1 to 5;
       population{i,j}=round(population{i,j});
     end;
   end;
   datalines;
799.9 821.4 827.9 817.7 819.5
903.7 987.3 1089.5 1198.2 1335.2
875.8 982.1 98.2 93.5 967.7
901.3 1086.5 1259.2 1535.3 1575.5
;
run;
/* Process data using hash objects.
  Declare hash and hash iterator objects
  Dataset argument,Ordered argument,Multidata argument
 Use hash object methods
 definekey() definedata() definedone() */

data _null_;
   if 0 then set sashelp.cars;
   if _n_ = 1 then do;
      declare Hash hCars(ordered:'d', multidata:'y');
      hCars.definekey('Make','Model','MSRP');
      hCars.definedata('Origin','Type','Make','Model','MSRP');
      hCars.definedone();
   end;
   set sashelp.cars end=end;
   hCars.add();
   if end then
      hCars.output(dataset: 'hash_example_1');
run;
/*find() add()output(), Use hash iterator object methods
 first()next()last()prev() */
data hash_example_2;
   set sashelp.cars end=end;
```

```
   if _n_ = 1 then do;
      if 0 then set sashelp.cars(keep=msrp horsepower);

      declare hash hCars(dataset:'sashelp.cars');
      declare hiter hCarsIter('hCars');
      hCars.definekey('make','model');
      hCars.definedata('make','model','msrp','horsepower');
      hCars.definedone();

      declare hash hCarsSum();
      hCarsSum.definekey('make');
      hCarsSum.definedata('make','total_make','total_price');
      hCarsSum.definedone();

   end;

   if hCars.find() ne 0 then do;
       match = 0;
       rc = hCarsIter.first();
       do while(rc = 0);
          if model eq model then do;
             match=1;
             rc=1;
          end;
          else rc = hCarsIter.next();
       end;
       if match ne 1 then do;
          call missing(msrp,horsepower);
          output;
          delete;
       end;
    end;

    if hCarsSum.find() ne 0 then do;
        total_make = 0;
        total_price = 0;
    end;
    total_make = total_make + 1;
    total_price = total_price + msrp;
    hCarsSum.replace();
    output;
    if end then hCarsSum.output(dataset: 'hash_example_3');
run;
/* • Use hash objects as lookup tables. */
data _null_;
   if _n_ = 0 then set sashelp.shoes (keep=product sales);
   declare hash hShoes(dataset:'sashelp.shoes',ordered:'a',multidata:'y');
   hShoes.definekey('product');
   hShoes.definedata('product','sales');
   hShoes.definedone();

   product = 'Slipper';
   rc = hShoes.find();
   put rc= product= sales=;
   anotherProduct = .;
   rc = hShoes.has_next(RESULT: anotherProduct);
   do while(anotherProduct ne 0);
      rc = hShoes.find_next();
      put rc= product= sales= anotherProduct=;
      rc = hShoes.has_next(RESULT: anotherProduct);
   end;   stop;run;
/* • Use hash objects to create sorted data sets. */
/* • Use hash iterator objects to access data in forward or reverse key order. */

data _null_;
   if 0 then set sashelp.cars;
   if _n_ = 1 then do;
      declare Hash hCars(ordered:'a', multidata:'y');
      hCars.definekey('Make','Model','MSRP');
```

```
      hCars.definedata('Origin','Type','Make','Model','MSRP');
      hCars.definedone();
   end;
   set sashelp.cars end=end;
   hCars.add();
   if end then
      hCars.output(dataset: 'hash_example_4');
run;


/* Use SAS utility procedures.
 Specify a template using the PICTURE statement within the FORMAT Procedure
 Specify templates for date, time, and datetime values using directives.
 Specify templates for numeric values using digit selectors.
 PICTURE statement options: round, default, datatype, multiplier, prefix */
proc format;
   picture dollarpic LOW-HIGH = '000,000.00' (prefix='$' fill=' ');
run;

proc print data=sashelp.cars(obs=10) noobs;
   format msrp dollarpic.;
   var make model msrp;
run;

/* • Create custom functions with the FCMP procedure
 Create character and numeric custom functions with single or multiple
 arguments.
 Create custom functions based on conditional processing.
 Use custom functions with the global option CMPLIB=. */

proc fcmp outlib=work.funcs.trial;
   function priceWithTaxFunc(price);
     static taxRate;
     if (taxRate eq .) then
        taxRate = (7.25 / 100);
        taxedPrice = (price * (1+taxRate));
     return (taxedPrice);
   endsub;
run;

option CMPLIB=work.funcs;
proc fcmp data=sashelp.cars out=carPriceWithTax;
    format msrpWithTax dollar8.;
    msrpWithTax = priceWithTaxFunc(msrp);run;
    /* Use advanced functions.
 Finding strings or words with the FINDC/FINDW function
 Counting strings or words with the COUNT/COUNTC/COUNTW
 Retrieve previous values with the LAG function. */
data holiday_1;
   set sashelp.holiday;
   containDayPosition = findw(desc, 'Day');
   containListPosition = findc(desc, 'z,y,x,w');
   countDay = count(lowcase(desc), 'day');
   countList = countc(lowcase(desc), 'z,y,x,w');
   countWords = countw(desc);
   lastHoliday = lag(desc);
run;
/* • Regular expression pattern matching with PRX functions
```