SAS

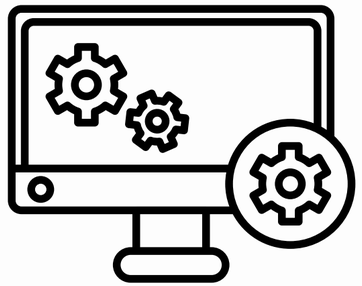
DATA STEP

CRASH COURSE

(Part 1)

@LearningwithJelly

Agenda



Intro/SAS Libraries

Length/Retain Statements

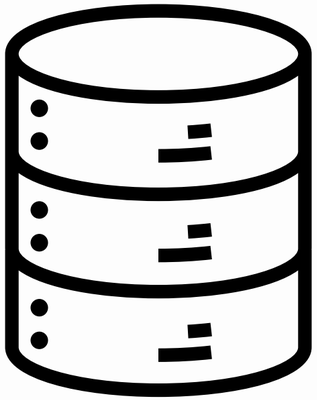
Input/Put Functions

Reading in DATA

Informats

Drop Statement

Conditional Statements



Variable Types

Common Functions

Do Statements

What is a DATA STEP?

- List of SAS Statements that start with a DATA statement

- What Can it Do?
  - Read in SAS Data Sets
  - Create SAS Data Sets
  - Transform Data (functions, create new vars, format data (informats)
  - and more!

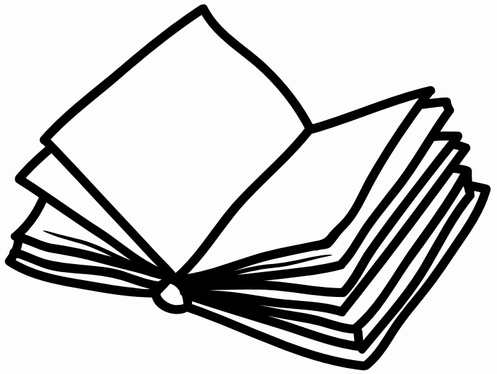
Creating a DATA Step

- Each statement ends with a semi-colon (;)
- Starts with the **DATA** keyword
- Typically ends with a RUN statement

```
DATA;
<other statements>;
RUN;
```

Getting DATA into SAS - SAS DATASETS on Your Computer

- In order to get your data into SAS you create a **library**
- A **library** is a collection of SAS data sets
- Create a library using the **libname statement**
- ***If your data is in non-sas dataset form you can read it in using infile statements (next section)***



Libname Statement

- Create a folder on your computer that has your SAS datasets in it (.sas7bdat)

- Create a library in SAS using the Libname Statement
  - Creates a libref (storage location for your library)
  - Max 8 bytes long for libref name (make the name of the libref 8 characters or less)
  - Name must also follow variable syntax rules
    - No special characters (except _)
    - Must start with a letter or underscore
    - No blanks

Libname Statement

SAS Downloaded on Your Computer

```
libname test 'C:\Documents\sasdatasets';
run;
```

SAS OnDemand for Academics

```
libname course '/home/u44796916/Course_Data';
run;
```

Library Naming Convention

- You can refer to datasets by **two-level or one-level names**

- Two Level Names
  - **LibraryName.DataSetName** (sashelp.cars)

- One Level Names
  - **DataSetName** (titanic)

- If you **do not** specify a library SAS assumes the library is the **WORK library**

- The **WORK library is TEMPORARY** and datasets will be REMOVED at the end of the SAS session

Importing Raw Data (non-SAS files)

IMPORTING RAW DATA - PROC IMPORT

- NON-SAS files (.csv, .txt. , .xslx etc.) need to be ‘specially’ imported into SAS using
  - **PROC IMPORT Procedure**
  - INFILE/INPUT statement combination

```
proc import datafile='/home/u44796916/titantic.csv'
     out = work.titanic /*these two lines are options not statements*/
     dbms = csv;
run;
```

Importing Data Using Infile/Input

- Infile statements within a DATA step can be used to read in RAW Data files

- INFILE statements are almost ALWAYS coupled with INPUT statements

- **INFILE statements are NEVER seen with SET statements**
- **INFILE reads in the file**
- **INPUT lists the variables and variable formats/informats that exist in the file (more on this later)**

```
data titanic2;
    infile '/home/u44796916/titantic.csv' delimiter=','
    firstobs=2 DSD;
    length name $50;
    input PassengerID $ Survived PClass Name $;
run;
```

Variable Types

Variable Types

- SAS has **two** main variable types

- **Vars name** can be mixed-cased, must start with a letter or number, contain no blanks

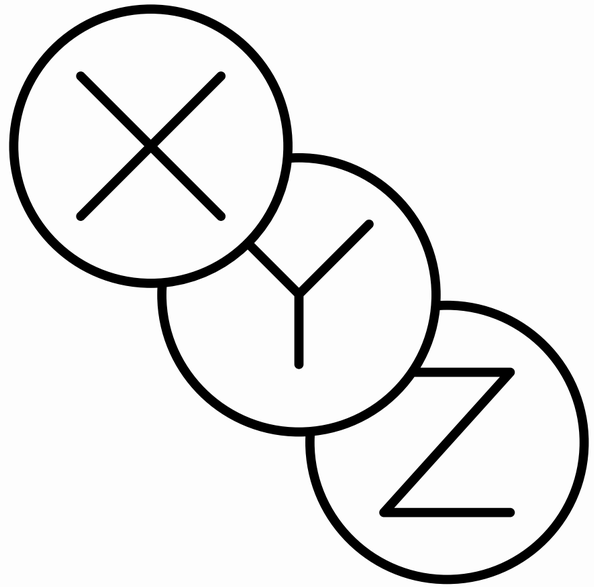
- 1) Character
  - Can contain numbers, letters, and special characters
  - Max of 32,767 characters long
  - **Left Aligned**
- 2) Numeric
  - Contains numbers
  - Stored as floating point numbers (decimals)
  - **Right Aligned**

CHARACTER **NUMERIC**

Team	nAtBat
Cleveland	293
Houston	315
Seattle	479

Creating New Variables

- Can create character or numeric variables (or dates)
- Can create vars using existing vars in the data set
- Can create new vars using functions (later in the lesson)
- **Most var creation is AFTER the SET or INFLIE/INPUT statements**
- For data professionals referred to as ‘feature engineering’



Creating New Variables

```
data baseball_country;
     set sashelp.baseball;
     Country = 'USA'; *adds a new var Country;
run;
```

Creating New Variables - Creating New Vars from Exisiting Vars

```
data baseball_salary;
    set sashelp.baseball;
    Salary_Full = Salary * 1000; *multiplies exisiting col by 1000;
run;
```

Creating New Variables - Creating New Vars using Functions

```
data air_test;
    set sashelp.air;
    Date_Month = Month(Date); *using the Month function;
run;
```

	DATE	AIR	Date_Month
1	JAN49	112	1
2	FEB49	118	2
3	MAR49	132	3
4	APR49	129	4

Length and Retain Statements

Length Statement

- Length statement specifies the bytes of a variable
- You can specify a dollar sign ($) after the variable name if it is a character
- Very important if you have ‘long’ variable values (i.e. name, address, etc.) so that it doesn’t get cut off in the dataset

```
data titanic2;
    infile '/home/u44796916/titantic.csv' delimiter=','
    firstobs=2 DSD;
    length name $50;
    input PassengerID $ Survived PClass Name $;
run;
```

Retain Statement

- Can be used to retain initial values of a variable (out of scope for this lesson)
- **Used to reorder variables in the output dataset**

```
data reorder_cars;
    retain mpg_highway mpg_city make;
    set sashelp.cars;
run;
```

BEFORE

Make	Model	Type
Acura	MDX	SUV
Acura	RSX Type S 2dr	Sedan

AFTER

mpg_highway	mpg_city	make
23	17	Acura
31	24	Acura

Informats & Raw Data

Informats

- **SAS can only read in structured data**
- Informats are normally used in the INPUT statement when reading in RAW data files
- When reading in **Unstructured data -- you need to use informats** (formats specific for unstructured data)
- Unstructured data can be data with symbols ($, commas, hyphens, decimal points, etc.)
- Dates are a common variable ‘type’ that needs to be read in using informats

Main Types of Informats

- **Character** - $informatw.
  - Dollar Sign Denotes a CHARACTER VARIABLE
- **Numeric** - informatw.d (d is decimal point)
- **Date** - infromatw (DATE9. MMDDYY10. etc.)

INFORMAT Example

```
data mri;
    infile '/home/u44796916/MRI.txt' delimiter=' ' firstobs=2;
    input Patient_ID $4. MRI_Date MMDDYY7.;
run;
```

Patient_ID	MRI_Date
1	11663
2	11932

FORMAT Statement

- Informats are used to READ in the data
- Special FORMATS can be applied to make the variable readable (important for dates)

```
data mri;
    format MRI_Date MMDDYY7.;
    infile '/home/u44796916/MRI.txt' delimiter=' ' firstobs=2;
    input Patient_ID $4. MRI_Date MMDDYY7.;
run;
```

MRI_Date
11663
11932
11920

MRI_Date	Patient_ID
120791	1
090192	2
082092	3

Side Note: Order of Execution

- SAS Execution of statements -- important to know when adding statements to Data Steps (Source: SAS Documentation)

Structure of a DATA Step	Action
DATA statement	begins the step counts iterations
Data-reading statements: [1]	
INPUT	describes the arrangement of values in the input data record from a raw data source
SET	reads an observation from one or more SAS data sets
MERGE	joins observations from two or more SAS data sets into a single observation
MODIFY	replaces, deletes, or appends observations in an existing SAS data set in place
UPDATE	updates a master file by applying transactions

FORMAT Statement

- Informats are used to READ in the data
- Special FORMATS can be applied to make the variable readable (important for dates)

```
data mri;
    format MRI_Date MMDDYY7.;
    infile '/home/u44796916/MRI.txt' delimiter=' ' firstobs=2;
    input Patient_ID $4. MRI_Date MMDDYY7.;
run;
```

MRI_Date
11663
11932
11920

MRI_Date	Patient_ID
120791	1
090192	2
082092	3

Common Functions

Functions

- Functions in SAS can accept arguments, perform a function, and return values
- Mostly used in DATA statements, where expressions, and SQL (not in PROC steps except PROC REPORT)

Commonly Used Functions Found Here

ROUND Function	Rounds the first argument to the nearest multiple of the second argument, or to the nearest integer when the second argument is omitted.
SCAN Function	Returns the *n*th word from a character string.
STRIP Function	Returns a character string with all leading and trailing blanks removed.
SUBSTR (left of =) Function	Replaces character value contents.
SUBSTR (right of =) Function	Extracts a substring from an argument.
SUM Function	Returns the sum of the nonmissing arguments.
TIME Function	Returns the current time of day as a numeric SAS time value.
TIMEPART Function	Extracts a time value from a SAS datetime value.
TRANWRD Function	Replaces all occurrences of a substring in a character string.
TRANSLATE Function	Replaces specific characters in a character expression.
TODAY Function	Returns the current date as a numeric SAS date value.
UPCASE Function	Converts all lowercase single-width English alphabet letters in an argument to uppercase.
YEAR Function	Returns the year from a SAS date value.
YRDIF Function	Returns the difference in years between two dates according to specified day count conventions; returns a person's age.

Functions - Numeric

- Statistical Functions (avg, sum, min, max, stdev, median, etc)
- Round()
- Int() - integer
- Abs() - absolute value

```
data test_fish;
    set sashelp.fish;
    Total_Len = sum(Length1, Length2, Length3);
run;
```

```
data test_fish;
    set sashelp.fish;
    Width = round(Width);
run;
```

Functions - Dates

- QTR()
- MONTH()
- YEAR()
- YRDIF()
- TODAY()

```
data air_test;
     set sashelp.air;
     Year = Year(Date);
run;
```

DATE	AIR	Year
JAN49	112	1949
FEB49	118	1949
MAR49	132	1949
APR49	129	1949

Functions - Concatenation

- **CAT()** - concatenates strings WITHOUT removing spaces
- **CATT()** - concatenates strings WITH removal of TRAILING spaces
- **CATS()** - concatenates strings WITH removal of LEADING & TRAILING SPACES
- **CATX()** - concatenates strings WITH removal of LEADING & TRAILING SPACES and separated them using a CUSTOM delimiter

```
data test_cars;
    set sashelp.cars;
    Make_Model_Cats = cats(Make, Model);
    Make_Model_Catx = catx('-', Make, Model);
run;
```

Make_Model_Cats	Make_Model_Catx
AcuraMDX	Acura-MDX
AcuraRSX Type S 2dr	Acura-RSX Type S 2dr
AcuraTSX 4dr	Acura-TSX 4dr

Input/Put

PUT/INPUT Functions

- PUT() & INPUT() functions allows you to convert variable type (i.e. numeric to character or character to numeric)
- Things to keep in mind:
  - **PUT() always creates character variables**
  - **INPUT() can create character or numeric variables based on the informat**
  - **The source format must match the source variable type in PUT()**
  - **The source variable type for INPUT() must always be character variables**

Function Call	Raw Type	Raw Value	Returned Type	Returned Value
A PUT(name, $10.);	**char, char format**	'Richard'	**char always**	'Richard '
B PUT(age, 4.);	num, num format	30	**char always**	' 30'
C PUT(name, $nickname.);	**char, char format**	'Richard'	**char always**	'Rick'
D INPUT(agechar, 4.);	**char always**	'30'	num, num informat	30
E INPUT(agechar, $4.);	**char always**	'30'	**char, char informat**	' 30'
F INPUT(cost,comma7.);	**char always**	'100,541'	num, num informat	100541

Source SAS Blog

PUT/INPUT Functions (Example)

- PUT() and INPUT() can be used together to get the desired output
- This converts the numeric ID column to a character value
- INPUT() always needs to take in a CHARACTER and PUT() always outputs a CHARACTER

```
data new_test;
    set test_fx;
    new_ID = put(ID, $4.);
run;
```

OR

```
data new_test;
    set test_fx;
    new_ID = put(ID, 4.);
    ID_Char = input(new_ID, $4.);
    drop new_ID;
run;
```

	ID	Age	Salary	ID_Char
1	1001	37	51250	1001
2	1002	52	63100	1002

DROP

Statement/Option

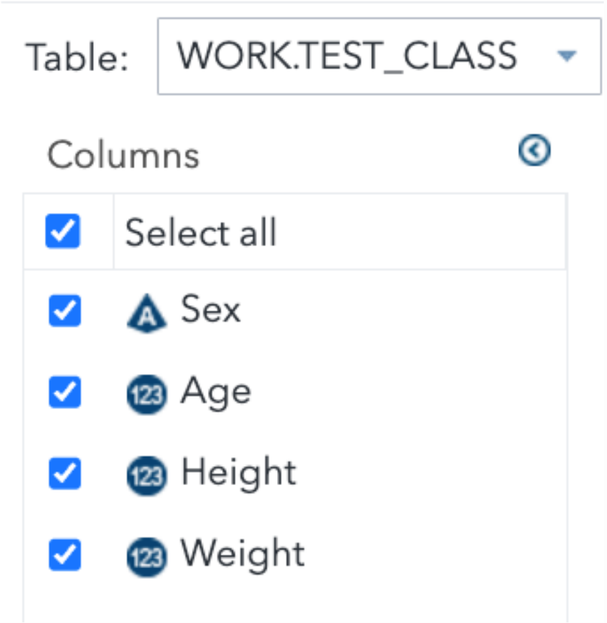
DROP= / KEEP = OPTIONS

- You can use the DROP/KEEP options in the SET statement to decide what variables to keep or drop

```
data test_class;
    set sashelp.class (drop=Name);
run;
```

OR

```
data test_class;
    set sashelp.class (keep=Age Height Sex Weight);
run;
```



DROP/KEEP statements

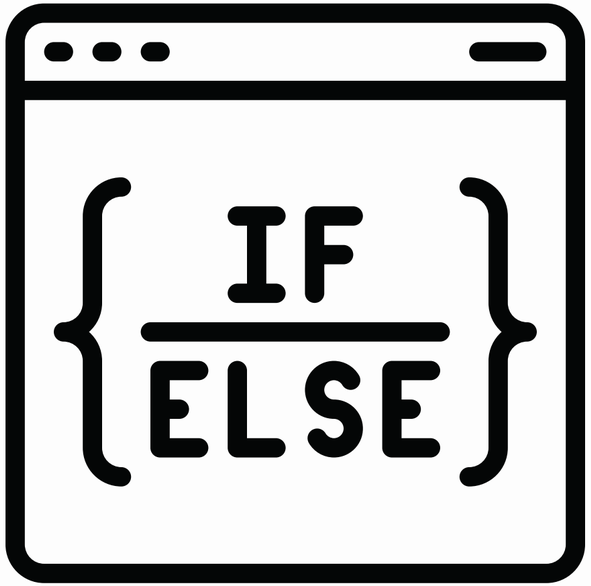
- No parenthesis or equal signs. Usually executes at different times in the code than drop/keep options in the SET statement

```
data new_test;
    set test_fx;
    new_ID = put(ID, $4.);
    drop ID;
run;
```

Conditional Statements IF/THEN/ELSE

Conditional Statements

- Conditional Statements are used for
  - Filtering data
  - Creating new variables
  - In reducing code redundancy and enhancing efficiency (more in next session DO Groups)



Conditional Statements

- Conditional Statements are used for
  - **Filtering data**
  - Creating new variables
  - In reducing code redundancy and enhancing efficiency (more in next session DO Groups)

```
data asia;
    set sashelp.cars;
    if origin = 'Asia' then output;
run;
```

Conditional Statements

- Conditional Statements are used for
  - **Filtering data creating MULTIPLE data sets**
  - Creating new variables
  - In reducing code redundancy and enhancing efficiency (more in next session DO Groups)

```
data asia europe other;
    set sashelp.cars;
    if origin ='Asia' then output asia;
    else if origin = 'Europe' then output europe;
    else output other;
run;
```

Conditional Statements

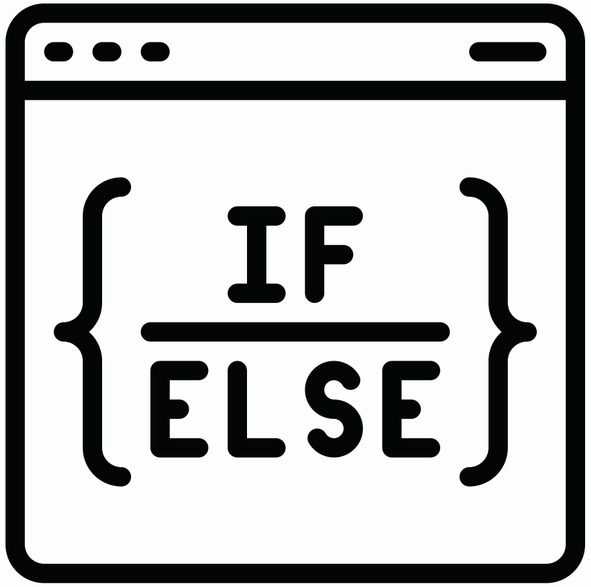
- Conditional Statements are used for
  - Filtering data creating MULTIPLE data sets
  - **Creating new variables**
  - In reducing code redundancy and enhancing efficiency (more in next session DO Groups)

```
data country_filter;
    set sashelp.cars;
    if origin = 'USA' then US_Flag = 1;
    else US_Flag = 0;
run;
```

Origin	US_Flag
Europe	0
Europe	0
USA	1
USA	1

Conditional Statements

- Conditional Statements are used for
  - Filtering data creating MULTIPLE data sets
  - Creating new variables
  - **In reducing code redundancy and enhancing efficiency (more in next session DO Groups)**



Do Statement

DO Statements

- Simple form of DO Group Processing
- **Repetitively executes a line(s) based on certain conditions (can be used with if/then/else statements)**
- Can be used as a ‘loop’ to reduce code redundancy (similar to FOR and WHILE loops in other languages)

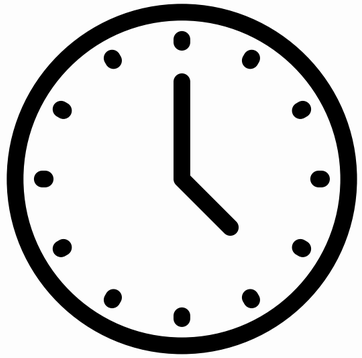
IF/THEN/DO Statements

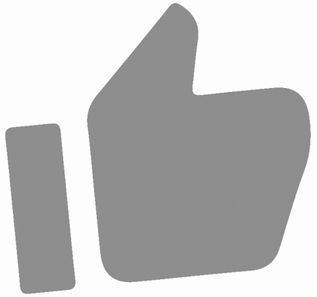
- DO groups start with a DO statement and ends with an END statement
- **If performing MORE THAN 1 action you need a DO GROUP**
- EACH ACTION is a STATEMENT and must end with a semicolon

```
data do_example;
     set sashelp.cars;
     if origin = 'USA' then
     do;
          Domestic = 1;
          New_Origin = 'United States';
     end;
     else Domestic = 0;
run;
```

DO Statements

- Simple form of DO Group Processing
- Repetitively executes a line(s) based on certain conditions (can be used with if/then/else statements)
- **Can be used as a ‘loop’ to reduce code redundancy (similar to FOR and WHILE loops in other languages) (NEXT Lecture)**





THANK YOU!

@LEARNINGWITHJELLY