Functions (Macros)

Functions and Workspaces: Variables

Functions (Macros)

Why Functions (Macros)

Hye-Chung Kum
Population Informatics Research Group
http://research.tamhsc.edu/pinformatics/
http://pinformatics.web.unc.edu/

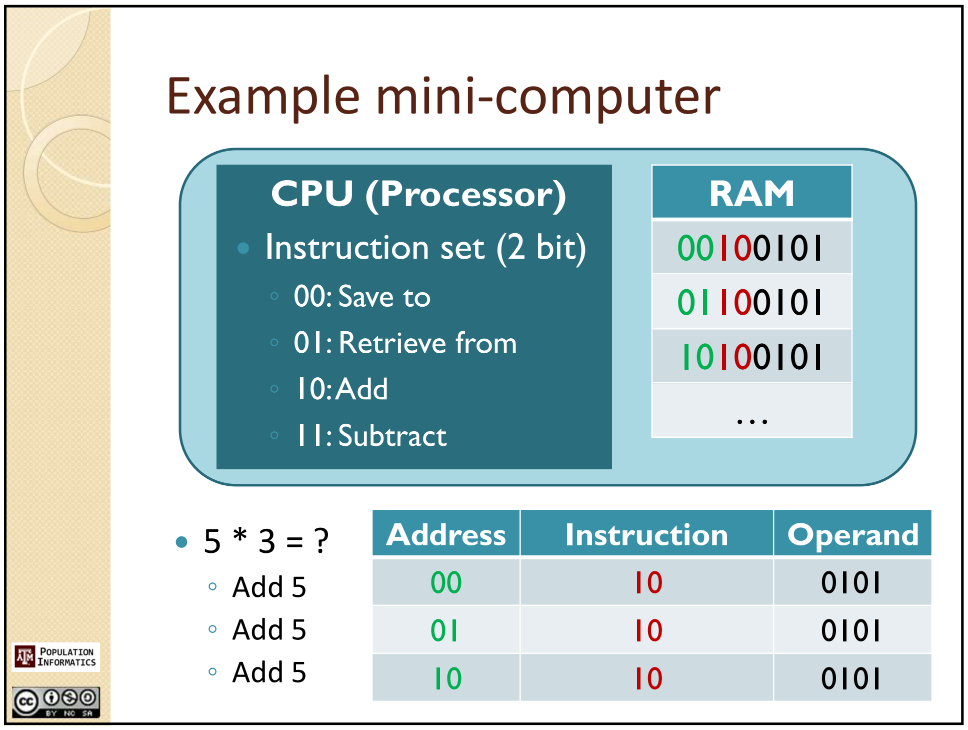
Course URL:
http://pinformatics.tamhsc.edu/phpm672

1

Programming

- Reusable code
- If you could not reuse code, writing exact steps for doing anything reasonable (usually takes MANY MANY lines of code) would take too much effort
- Programming works because
  - you write functions, small building blocks, that do small defined tasks correctly given certain input (parameters)
  - Then compose these functions together to carry out the complex task

2



Example mini-computer

CPU (Processor)

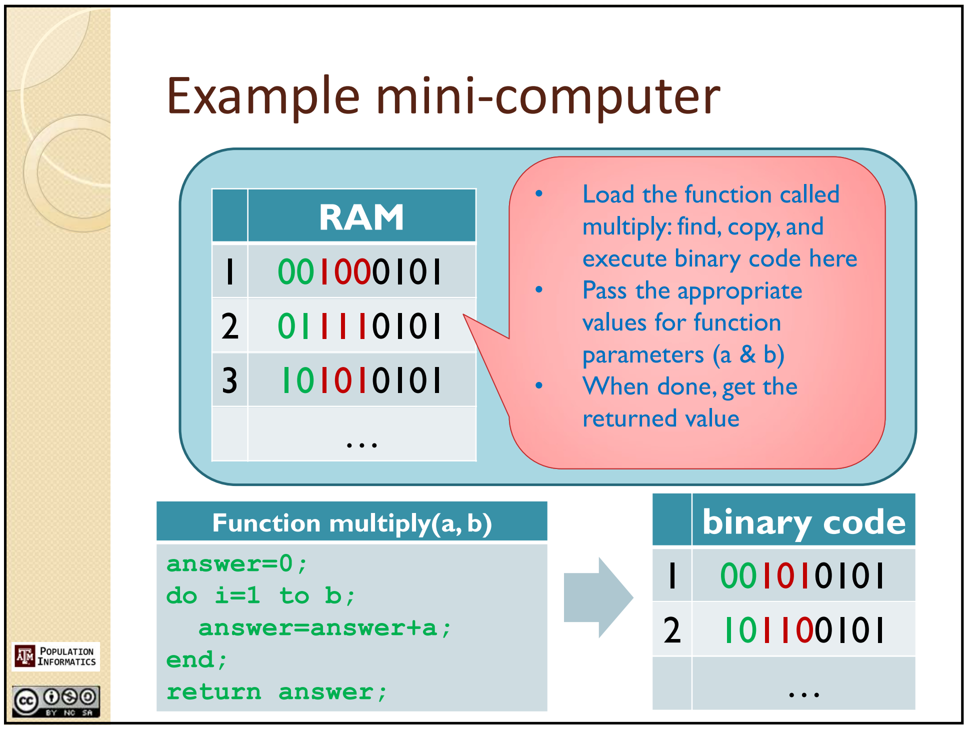
- Instruction set (2 bit)
  - 00: Save to
  - 01: Retrieve from
  - 10: Add
  - 11: Subtract

RAM
00100101
01100101
10100101
…

- 5 * 3 = ?
  - Add 5
  - Add 5
  - Add 5

Address	Instruction	Operand
00	10	0101
01	10	0101
10	10	0101

3



Example mini-computer

	RAM
1	001000101
2	011110101
3	101010101
	…

- Load the function called multiply: find, copy, and execute binary code here
- Pass the appropriate values for function parameters (a & b)
- When done, get the returned value

Function multiply(a, b)

```
answer=0;
do i=1 to b;
  answer=answer+a;
end;
return answer;
```

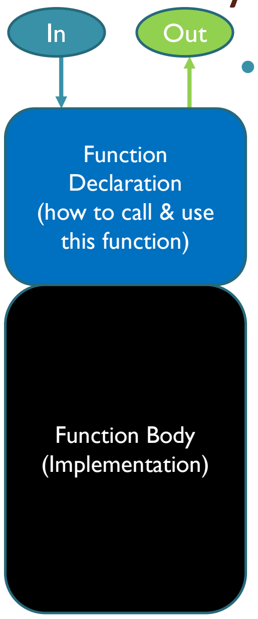
	binary code
1	001010101
2	101100101
	…

4

Why use Functions?

- Top-down design
  - Break a complex problem into simpler manageable problems
  - Solve simpler problems
  - Connect simple solutions to solve original problem
- Testing strategy
  - Call function with different inputs to find bugs in algorithm
  - Small components tested individually
  - Connect components later (system integration)
  - Try testing 10,000 lines of script code without functions !?!

Why use Functions?


In
Out
Function Declaration (how to call & use this function)
Function Body (Implementation)

- **Encapsulation**
  - Black box programming
  - Hides internal details of algorithm from users
  - Users typically only care about using the function to get results.
- Isolates computations, protects variables
  - Interaction through arguments
- Separates interface and implementation
  - Interface: what a function does
  - Implementation: how a function does it

Why use Functions?

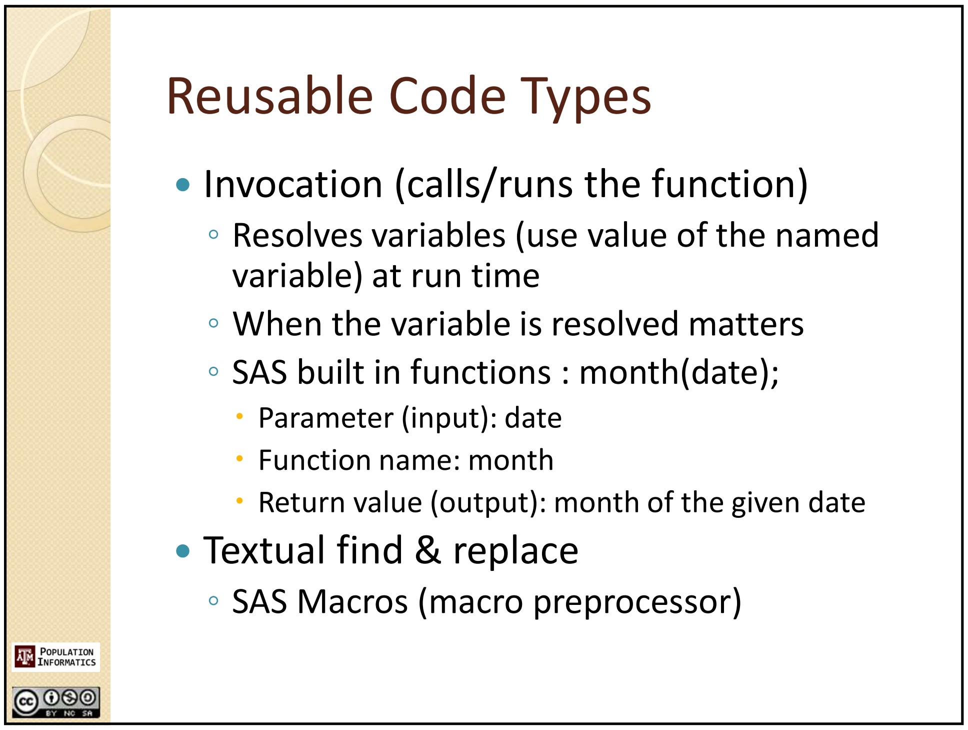
- Code reuse
  - Solve a problem once
  - Reuse your solution for similar problems
- Avoids repetitive typing
  - Consistency
  - Reduce Mistakes
  - Maintenance
    - Easier to fix one function than find and fix all locations of cut & paste code.

7

Why use Functions?

- Code sharing
  - Share your solution to a problem with others.
  - Collaboration
    - Team, organization, world
  - Another programmer only needs to know your function interface and behavior to use it.
  - Get solution from someone else
    - (and get caught easily if it's an assignment)

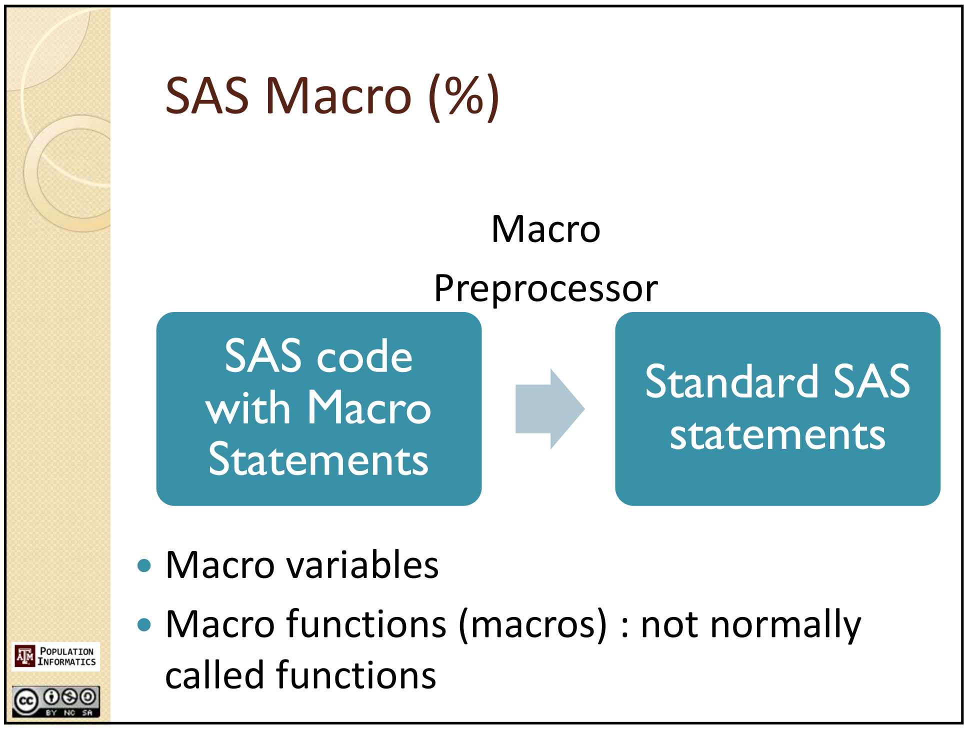
8



Reusable Code Types

- Invocation (calls/runs the function)
  - Resolves variables (use value of the named variable) at run time
  - When the variable is resolved matters
  - SAS built in functions : month(date);
    - Parameter (input): date
    - Function name: month
    - Return value (output): month of the given date
- Textual find & replace
  - SAS Macros (macro preprocessor)

9



SAS Macro (%)

Macro Preprocessor

SAS code with Macro Statements → Standard SAS statements

- Macro variables
- Macro functions (macros) : not normally called functions

10

Assignment 6 Objectives

- Read and write SAS macro variables
- Read, use, and modify SAS macro functions

11

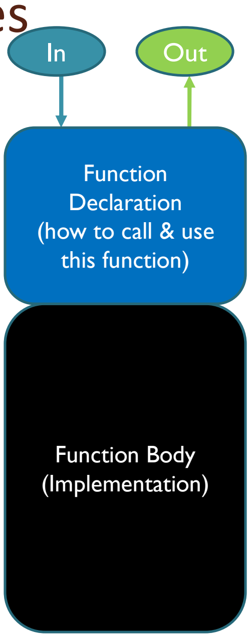
What is a workspace?

- The workspace is the set of variables that has been collected or instantiated during a session
- Session: one run of SAS (the time that you have been using SAS)
  - Batch mode: during the one run
- The two main workspace in SAS
  - SAS tables
  - Macro variables

12

Local vs Global Variables

- Based on scope of variable
  - Scope= workspace
- Global variables
  - Valid in all workspace
- Local variable
  - Valid in only the local workspace
  - For example inside a function or Macro


In
Out
Function Declaration (how to call & use this function)
Function Body (Implementation)

Macro Variables (older version)

- The name of a macro variable can be from one to eight characters.
- The name must begin with a letter or an underscore.
- Only letters, numbers, or underscores can follow the first letter.
- The content of macro variable can be up to 32K (in version 7, the limit is 64K).
- No macro variable can begin with SYS.
- No macro variable can have the same name as a SAS-supplied macro or macro function

Macro Variables

```
* Define a global macro variable;
%let varname = value;

* Use a defined macro variable;
keep &varname;
title  “&varname” ;  * must be double quotes;

* Resolves to be identical to;
keep value;
title  “value” ;

* Try examples;
```

15

Evaluating Expressions

```
* Integer arithmetic;
%let macro_var = %eval(expression);
%let age=%eval( 5+3 );

Myage=&age;
Myage=8;

* If float;
%let macro_var = %sysevalf(expression);
%let age=%sysevalf( 5.5+3 );
```

16

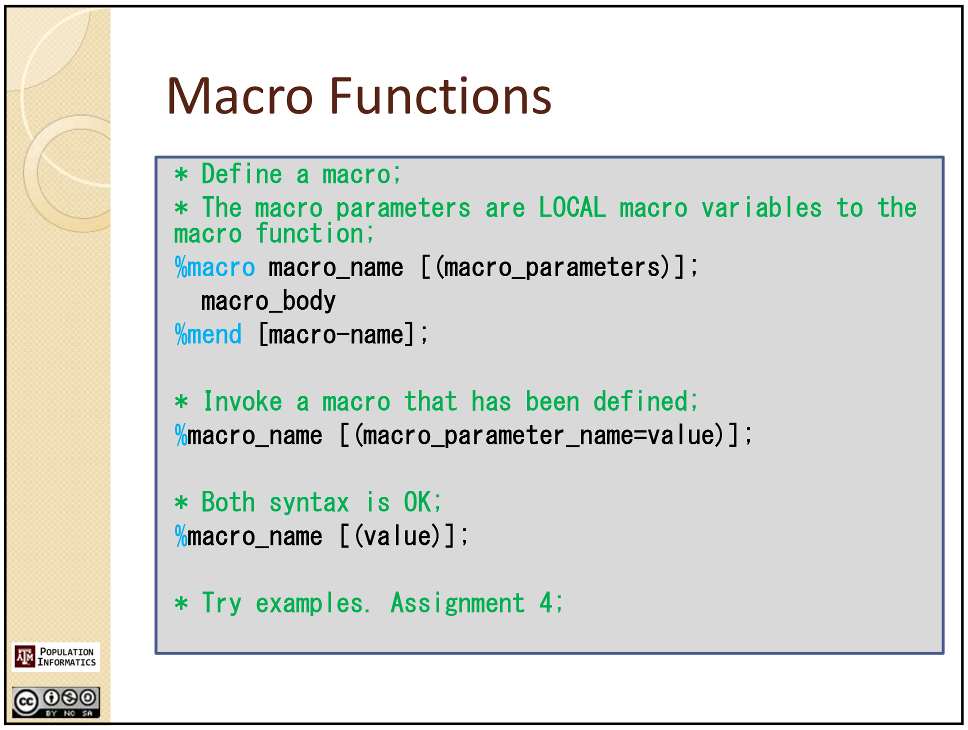
Moving data between Macro Variable & SAS Tables

```
CALL SYMPUT ( "macro_var_name" , value);
CALL SYMGET ( "macro_var_name" );
```

- Create/reassign macro_var_name
- Same as %let except, can take values from sas table
- Value could be
  - A variable from a sas dataset
  - Constant
- Assigns the value at the end of the step
  - Run
  - Proc & Data
- Symget vs &
  - When the variable is resolved

Macro Functions

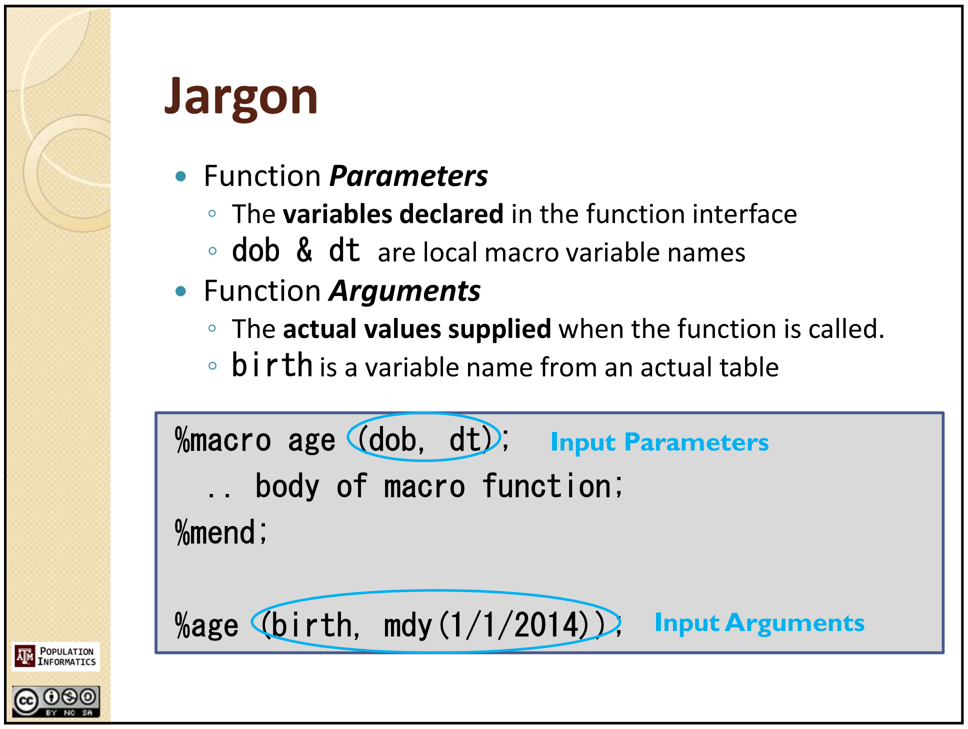
- Pro: Reusable code
  - Allows you to write a set of sas statements once, and then use them over and over again
- Con: more complicated code can lead to more difficulty in debugging
  - You MUST write modular code
  - First, write your program in normal SAS code
  - Test that it works
  - Then convert to SAS Macro
  - Test that the macro works



Macro Functions

```
* Define a macro;
* The macro parameters are LOCAL macro variables to the
macro function;
%macro macro_name [(macro_parameters)];
  macro_body
%mend [macro-name];

* Invoke a macro that has been defined;
%macro_name [(macro_parameter_name=value)];

* Both syntax is OK;
%macro_name [(value)];

* Try examples. Assignment 4;
```

19



Jargon

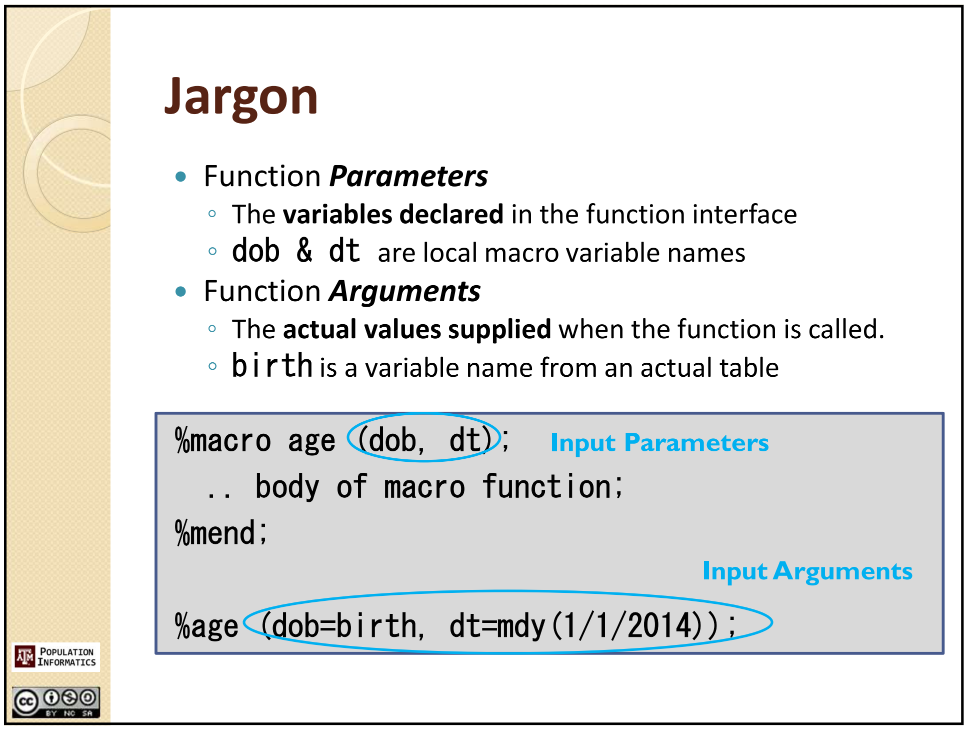
- Function ***Parameters***
  - The **variables declared** in the function interface
  - `dob & dt` are local macro variable names
- Function ***Arguments***
  - The **actual values supplied** when the function is called.
  - `birth` is a variable name from an actual table

```
%macro age (dob, dt);
  .. body of macro function;
%mend;

%age (birth, mdy(1/1/2014));
```

Input Parameters

Input Arguments

20

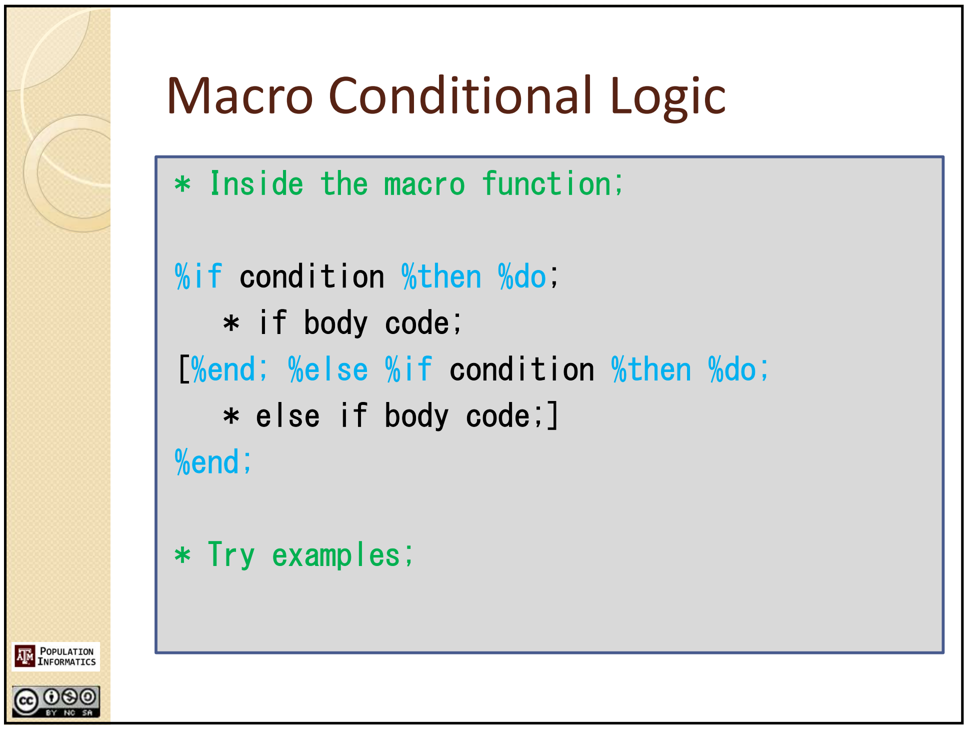


Jargon

- Function ***Parameters***
  - The **variables declared** in the function interface
  - `dob & dt` are local macro variable names
- Function ***Arguments***
  - The **actual values supplied** when the function is called.
  - `birth` is a variable name from an actual table

```
%macro age (dob, dt);    Input Parameters
  .. body of macro function;
%mend;
                                 Input Arguments
%age (dob=birth, dt=mdy(1/1/2014));
```

21



Macro Conditional Logic

```
* Inside the macro function;

%if condition %then %do;
   * if body code;
[%end; %else %if condition %then %do;
   * else if body code;]
%end;

* Try examples;
```

22

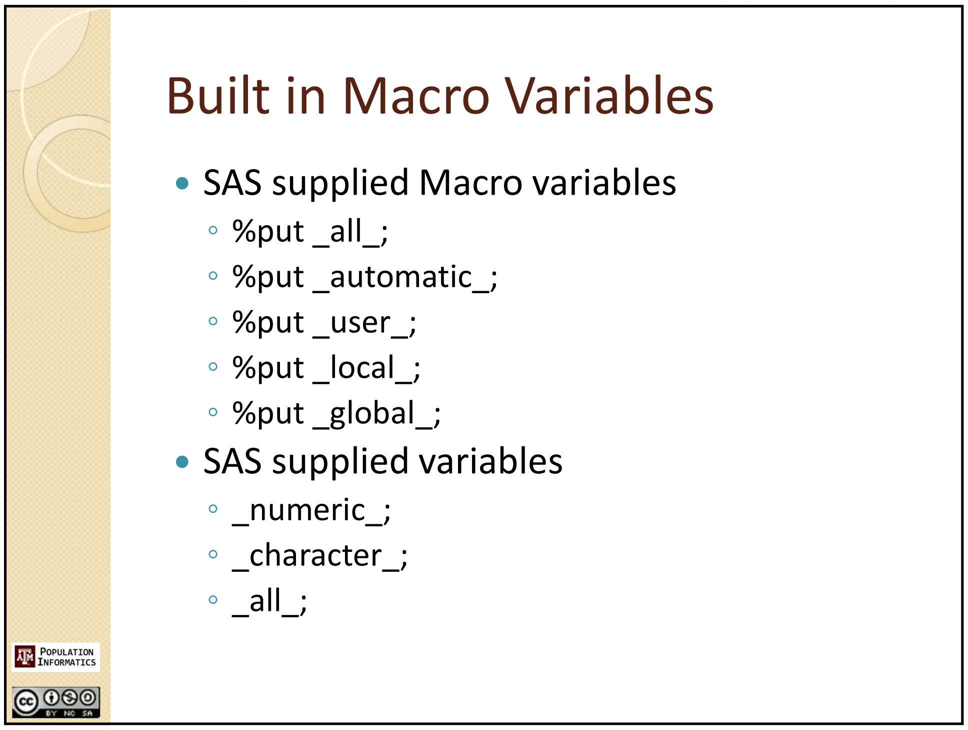
Macro Loops

```
* Inside the macro function;

%do i=istart %to iend;
   * if body code;
%end;

* Try examples;
```

Debugging Macros

- MPRINT
- SYMBOLGEN
- MLOGIC
- %put
- %include
  - config.sas

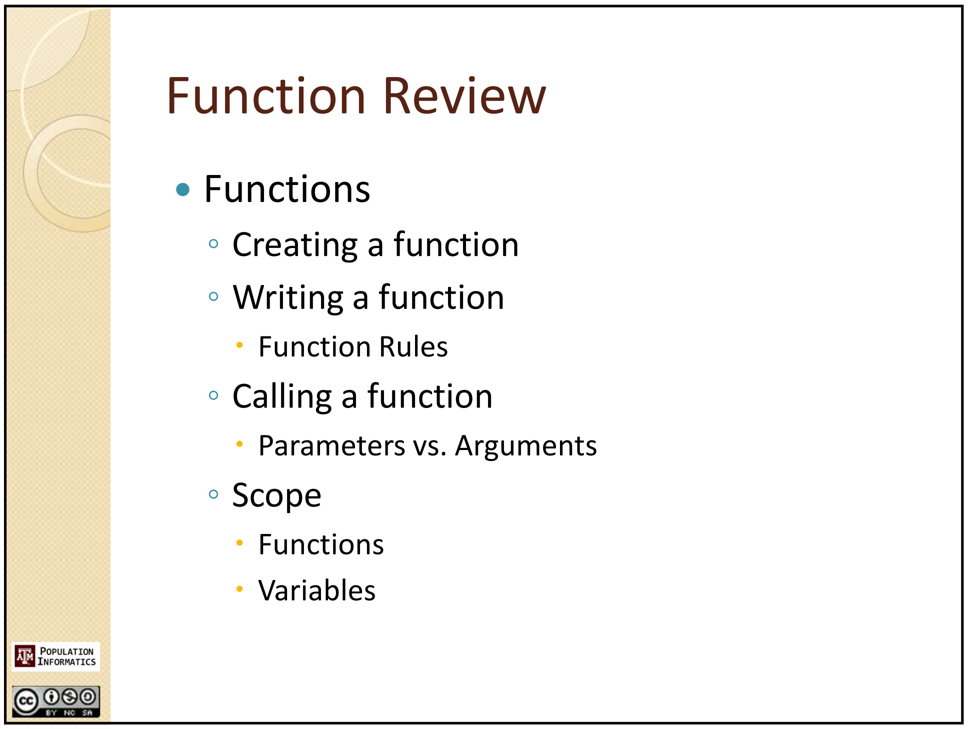
```
Options MPRINT MLOGIC SYMBOLGEN;
* Look at log;
```



Built in Macro Variables

- SAS supplied Macro variables
  - %put _all_;
  - %put _automatic_;
  - %put _user_;
  - %put _local_;
  - %put _global_;
- SAS supplied variables
  - _numeric_;
  - _character_;
  - _all_;

25



Function Review

- Functions
  - Creating a function
  - Writing a function
    - Function Rules
  - Calling a function
    - Parameters vs. Arguments
  - Scope
    - Functions
    - Variables

26

Programming …
Read.
Watch.
Do.
Repeat doing until you get the hang of it.

POPULATION INFORMATICS

From Assignment 6 on …

- Grading for style
  - Consistent style
  - Readable beautiful code
  - Good indentation
  - Good line breaks
  - Variable names
  - Comments
- For full grade: when you are done, go back and “EDIT” to make it readable and consistent before submission

POPULATION INFORMATICS

Assignment 6

- Objectives
  - Read and write SAS macro variables
  - Read, use, and modify SAS macro functions
- Lab 6
  - Start doing in class

Recoding

- It is perfectly fine to overwrite variable value in recoding.
  - acceptable and RECOMMENDED coding
    - county=compress(county)
  - It means take value from county, compress it, than save the new value into the county variable and overwrite what was there.

```
*clear blanks in county names;
ncounty=compress(county);
drop county;
rename ncounty=county;
```