



# Edmonton SAS Users Group

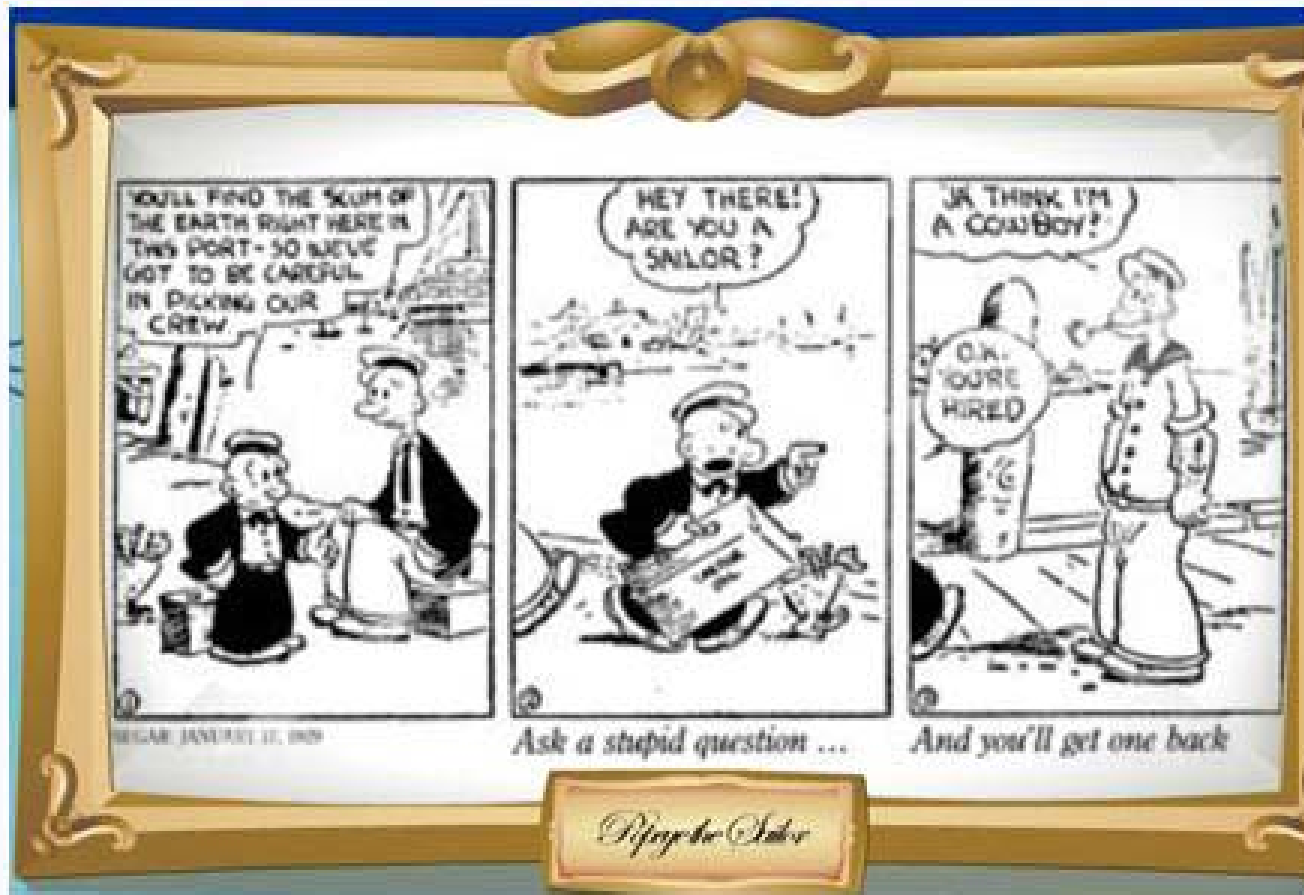
## March 11, 2009

**A Cup of Coffee and Proc FCMP:  
I Cannot Function Without Them**

---

Peter Eberhardt, Fernwood Consulting Group Inc.

# Global Forum in National Harbor



# Agenda

- What is a function?
- How do we create functions?
- How do we debug functions?
- How do we deploy functions?
- Review and Questions

# What is a function

# Function

Main Entry: **<sup>1</sup>func-tion**

Pronunciation: \<sup>1</sup>fəŋ(k)-shən\

Function: *noun*

Etymology: Latin *function-*, *functio* performance, from *fungi* to perform; probably akin to Sanskrit *bhunkte* he enjoys

**1:** professional or official position : occupation

**2:** the action for which a person or thing is specially fitted or used or for which a thing exists : purpose

**3:** any of a group of related actions contributing to a larger action ; *especially* : the normal and specific contribution of a bodily part to the economy of a living organism

Source: Mirriam-Webser Online

# Function

Main Entry: **<sup>1</sup>func-tion**

**4:** an official or formal ceremony or social gathering

**5a:** a mathematical correspondence that assigns exactly one element of one set to each element of the same or another set

**5b:** a variable (as a quality, trait, or measurement) that depends on and varies with another <height is a *function* of age> ; *also* : result <illnesses that are a *function* of stress>

**6:** characteristic behavior of a chemical compound due to a particular reactive unit ; *also* : functional group

# Function

Main Entry: **<sup>1</sup>func-tion**

**7:** a computer subroutine ; *specifically* : one that performs a calculation with variables provided by a program and supplies the program with a single result

## What is a function

- A function is a rule for transforming zero or more values called ***arguments***; the result of the transformation is called the value or result of the function. The value of the result is uniquely determined by the arguments. The transformation can also have side effects, that is, permanent changes in the values of arguments.



## What is a function

- The SAS 9.2 online help has a similar definition:

A **SAS function** performs a computation or system manipulation on arguments, and returns a value that can be used in an assignment statement or elsewhere in expressions

# What is a function

- Functions have names

- `rc = foo(2500, 3);`

- Names should be meaningful

- `rc = monthlyInterest(2500, 3);`
- `interestPaid = monthlyInterest(balance, iRate);`

# What is a function

## ■ Call Routines

- A **CALL routine** alters variable values or performs other system functions. CALL routines are similar to functions, but differ from functions in that you cannot use them in assignment statements or expressions.
- All SAS CALL routines are invoked with CALL statements. That is, the name of the routine must appear after the keyword CALL on the CALL statement

■ call monthlyInterest(balance, iRate, interestPaid);

# Agenda

- ☺ What is a function?
- How do we create functions?
- How do we debug functions?
- How do we deploy functions?
- Review and Questions

## How do we create functions

- If it's raining I wear my black shoes; otherwise I wear either my red shoes or my blue shoes

$$d = f(w) \begin{cases} \text{Black} & \text{if } w = \text{rain} \\ \text{Red or Blue} & \text{otherwise} \end{cases}$$

- “the value of the result is **uniquely** determined by the arguments”

## How do we create functions

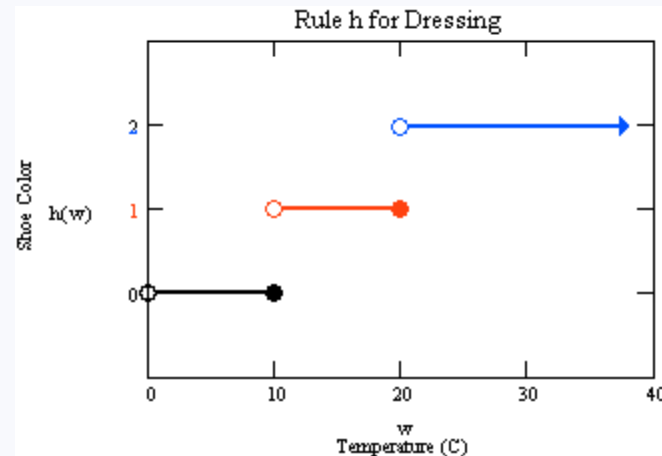
$g :=$

	$w$ ( $^{\circ}\text{C}$ )	$d$ (color)
	0	1
0	10	<b>0</b>
1	20	<b>1</b>
2	30	<b>2</b>

$$d = g(w)$$

- the value of the result is uniquely determined by the arguments
- The **domain** of the rule is finite (10, 20, 30)

# How do we create functions



$$d = h(w)$$

- the value of the result is uniquely determined by the arguments
- The **domain** of the rule is all temperatures greater than 0

## How do we create functions

$$d = m(w) = 2w - 1$$

Where you have numbered your shoes  $d = 1, 2, 3, \dots$   
according to some personal scheme

- The ***domain*** of the rule is all temperatures greater than or equal 1



## How do we create functions

- Make sure your result is unambiguous
- Identify the domain
- Learn PROC FCMP

## How do we create functions

### **PROC FCMP** option(s);

**ARRAY** array-name[dimensions] </NOSYMBOLS | variable(s) | constant(s) | (initial-value(s))>;

**ATTRIB** variable(s) <FORMAT=format-name LABEL='label' LENGTH=length>;

**FORMAT** variable(s) <format> <DEFAULT=default-format>;

**FUNCTION** function-name(argument-1, argument-2, ..., argument-n) <\$>  
<length>;

**LABEL** variable='label';

**LENGTH** variable(s)<\$> length <DEFAULT=n>;

**STRUCT** structure-name variable;

**SUBROUTINE** subroutine-name (argument-1, argument-2, ..., argument-n);  
OUTARGS out-argument-1, out-argument-2, ..., out-argument-n;

# How do we create functions

## **PROC FCMP** *option(s)*;

**ARRAY** array-name[dimensions] </NOSYMBOLS | variable(s) | constant(s) | (initial-value(s))>;

**ATTRIB** variable(s) <FORMAT=format-name LABEL='label' LENGTH=length>;

**FORMAT** variable(s) <format> <DEFAULT=default-format>;

**FUNCTION** **function-name(argument-1, argument-2, ..., argument-n) <\$>**  
**<length>;**

**LABEL** variable='label';

**LENGTH** variable(s)<\$> length <DEFAULT=n>;

**STRUCT** structure-name variable;

**SUBROUTINE** **subroutine-name (argument-1, argument-2, ..., argument-n);**

**OUTARGS** **out-argument-1, out-argument-2, ..., out-argument-n;**

## How do we create functions

```
PROC FCMP outlib=work.funcs.Test;
```

```
/* outlib= where will the functions be saved */
```

## How do we create functions

```
PROC FCMP outlib=work.funcs.Test;  
/* declare a function returning a number */  
function whatAml();  
    return(42);    /* return the number */  
endsub;
```

## How do we create functions

```
PROC FCMP outlib=work.funcs.Test;  
/* declare a function returning a number */  
function whatAml();  
    return(42);    /* return the number */  
endsub;  
  
/* declare a function returning a string */  
function whereAml() $;  
    return('In Test');    /* return the string */  
endsub;
```

## How do we create functions

```
PROC FCMP outlib=work.funcs.Test;  
/* declare a function returning a number */  
function whatAml();  
    return(42);    /* return the number */  
endsub;  
  
/* declare a function returning a string */  
function whereAml() $;  
    return('In Test');    /* return the string */  
endsub;  
quit;
```

## How do we create functions

```
1  proc fcmp
2      outlib=work.funcs.test;
3  function whatAmI(); /* declare a function returning a number */
4      return(42);      /* return the number */
5  endsub;
6
7  function whereAmI() $; /* declare a function returning a string */
8      return('In Test'); /* return the string */
9  endsub;
10
11 quit;
```

NOTE: Function whereAmI saved to work.funcs.test.

NOTE: Function whatAmI saved to work.funcs.test.

NOTE: PROCEDURE FCMP used (Total process time):

real time                    1.48 seconds

cpu time                     0.14 seconds



## How do we create functions

```
data _null_;  
    rci = whatAmI();  
    put rci=; /* should be 42 */  
    rcc = whereAmI();  
    put rcc=; /* should be In Test */  
  
run;
```

## How do we create functions

```
12
13 data _null_;
14     rci = whatAmI();
```

-----

68

**ERROR 68-185: The function WHATAMI is unknown, or cannot be accessed.**

```
15     put rci=;
16     rcc = whereAmI();
```

-----

68

**ERROR 68-185: The function WHEREAMI is unknown, or cannot be accessed.**

```
17     put rcc=;
18 run;
```

## How do we create functions

### ■ SAS option CMPLIB=

- Specifies the library where the functions reside

```
19  options cmplib=work.funcs;
20  data _null_;
21      rci = whatAmI();
22      put rci=; /* should be 42 */
23      rcc = whereAmI();
24      put rcc=; /* should be In Test */
25  run;
rci=42
rcc=In Test
```

NOTE: DATA statement used (Total process time):

real time                   0.14 seconds

cpu time                    0.07 seconds

## How do we create functions

### ■ SAS option CMPLIB=

- Specifies the library where the functions reside

```
19  options cmplib=work.funcs; /* outlib=work.funcs.test; */
20  data _null_;
21      rci = whatAmI();
22      put rci=; /* should be 42 */
23      rcc = whereAmI();
24      put rcc=; /* should be In Test */
25  run;
rci=42
rcc=In Test
```

NOTE: DATA statement used (Total process time):

real time                   0.14 seconds

cpu time                    0.07 seconds

## How do we create functions

```
27  proc fcmp
28      outlib=work.funcs.test;  /* where will the functions be saved */
29  function whatAmI(startValue); /* declare a function returning a number */
```

*WARNING: Function whatAmI is already defined in packet test. Function whatAmI as defined in the current program will be used as default when packet is not specified.*

```
30      if missing(startValue)  then rc=.S;
31      else if startValue < 0  then rc=.Z;
32      else if startValue < 20 then rc=0;
33      else if startValue < 50 then rc=20;
34      else if startValue < 100 then rc=50;
35      else                      rc=100;
36      return(rc);
37  endsub;
38
39  quit;
```

NOTE: Function whatAmI saved to work.funcs.test.

## How do we create functions

```
42  data _null_;
43      iAm = .;
44      rci = whatAmI(iAM);
45      put iAM= rci=; /* should be .S */
46      iAm = -1;
47      rci = whatAmI(iAM);
48      put iAM= rci=; /* should be .Z */
49      iAm = 10;
50      rci = whatAmI(iAM);
51      put iAM= rci=; /* should be 0 */
52      iAm = 30;
53      rci = whatAmI(iAM);
54      put iAM= rci=; /* should be 20 */
55      iAm = 80;
56      rci = whatAmI(iAM);
57      put iAM= rci=; /* should be 50 */
```

## How do we create functions

```
58      iAm = 20000;  
59      rci = whatAmI(iAM);  
60      put iAM= rci=; /* should be 100 */  
61 run;
```

```
iAm=. rci=S  
iAm=-1 rci=Z  
iAm=10 rci=0  
iAm=30 rci=20  
iAm=80 rci=50  
iAm=20000 rci=100
```

```
if missing(startValue) then rc=.S;  
else if startValue < 0 then rc=.Z;  
else if startValue < 20 then rc=0;  
else if startValue < 50 then rc=20;  
else if startValue < 100 then rc=50;  
else rc=100;
```

NOTE: DATA statement used (Total process time):

```
real time          0.10 seconds  
cpu time           0.07 seconds
```

## How do we create functions

```
63  proc fcmp
64      outlib=work.funcs.test; /* where will the functions be saved */
65  subroutine whatAmI(startValue, rc); /* declare a subroutine          */
```

*WARNING: Function whatAmI is already defined in packet test. Function whatAmI as defined in the current program will be used as default when packet is not specified.*

```
67
```

```
68  if missing(startValue) then rc=.S;
```

*WARNING: The variable rc should not be the result of the '=' operation because it is a read-only argument to subroutine whatAmI. Any changes to this argument will not be returned from the subroutine whatAmI. Use the OUTARGS statement to allow return values from a subroutine.*



## How do we create functions

```
105 proc fcmp outlib=work.funcs.test;
```

```
106 subroutine whatAmI(startValue, rc); /* declare a subroutine *
```

*WARNING: Function whatAmI is already defined in packet test. Function whatAmI as defined in the current program will be used as default when packet is not specified. 107*

```
107.          outargs rc; /* argument rc will return a value */
```

```
108
```

```
109   if missing(startValue) then rc=.S;
```

```
110   else if startValue < 0 then rc=.Z;
```

```
111   else if startValue < 20 then rc=0;
```

```
112   else if startValue < 50 then rc=20;
```

```
113   else if startValue < 100 then rc=50;
```

```
114   else          rc=100;
```

```
115   endsub;
```

```
116   quit;
```

NOTE: Function whatAmI saved to work.funcs.test.

NOTE: PROCEDURE FCMP used (Total process time):

## How do we create functions

```
120 data _null_;
121     length rci 8.;
122     rci = -1;
123     iAm = .;
124     call whatAmI(iAM, rci);
125     put iAM= rci=; /* should be .S */
126     iAm = -1;
127     call whatAmI(iAM, rci);
128     put iAM= rci=; /* should be .Z */\
...

```

iAm=. rci=S

iAm=-1 rci=Z

iAm=10 rci=0

iAm=30 rci=20

iAm=80 rci=50

iAm=20000 rci=100

```
if missing(startValue) then rc=.S;
else if startValue < 0 then rc=.Z;
else if startValue < 20 then rc=0;
else if startValue < 50 then rc=20;
else if startValue < 100 then rc=50;
else rc=100;
```

**NOTE: DATA statement used (Total process time):**

# How do we create functions

## ■ Passing arguments

- By value
  - A ‘copy’ of the variable is passed in. Any changes to the argument are NOT passed back.
- By reference
  - The ‘address’ of the variable is passed in. Any changes to the argument ARE passed back.
  - outargs

# How do we create functions

- Passing arguments
  - Arrays
    - Always passed by reference

# How do we create functions

- How to choose between a function and a call routine
  - Do you need to return a value?
  - Do you need to return more than one value?

# How do we create functions

- Multiple Return Values
  - Outargs arg1, arg2, ..., argN

# How do we create functions

## ■ *Two Tables*

- *Monthly special payments. Only one payment date per month (~300,000 records per month)*

Column	Description
ID	Unique entity identifier
GROUP	Group in which ID is a member
PAYRULE	Payment rule
PAYDATE	Payment Date. There is only one payment date per month for everyone
PAYAMOUNT	Amount of the Payment

- *Daily transactions (~150 Million records per year)*

Column	Description
ID	Unique entity identifier.
GROUP	Group in which ID is a member
SERVICECODE	A code identifying the service
SERVICEDATE	Service Date. These are the actual date of service
SERVICEAMOUNT	Amount of the Payment

## How do we create functions

```
proc fcmp outlib=work.Funcs.Dates;  
  
subroutine datesInMonth(inDate, startDate, endDate, days);  
    outargs startDate, endDate, days;  
  
    nextMonth = intnx('month', inDate, 1);  
    startDate = intnx('month', nextMonth, -1);  
    endDate   = nextMonth - 1;  
    days      = nextMonth - startDate;  
    return;  
  
endsub;  
quit;
```



## How do we create functions

```
data cntlinPaydates;
  retain fmtName 'payDates';
  set payDates;
  call missing(start, end, days);
  call datesInMonth(payDate, start, end, days);
  label = paydate;
  format payDate start end yymmdd10.;

run;

proc format cntlin=cntlinPaydates;
run;

data test;
  set allVisits;
  paydate = input(put(servicedate, paydates.), 7.);
  format paydate yymmdd10.;

run;
```

Select distinct paydate from paydates

# How do we create functions

## ■ Other things

- Recursion – reference the paper by Sekosky
- Arrays
- No SET/MERGE etc

# Agenda

- ☺ What is a function?
- ☺ How do we create functions?
- How do we debug functions?
- How do we deploy functions?
- Review and Questions

## How do we debug functions

- Be careful
- Cannot use the SAS dataset debugger
- PUT statement
  - To LOG
  - To PRINT

## How do we debug functions

```
154 proc fcmp outlib=work.Funcs.Dates;
155   subroutine datesInMonth(inDate, startDate, endDate, days);
156       outargs startDate, endDate, days;
157       FILE log
158       put inDate yymmdd10. ;
159       nextMonth = intnx('month', inDate, 1);
160       put nextMonth yymmdd10.;
161       startDate = intnx('month', nextMonth, -1);
162       put startDate yymmdd10.;
163       endDate = nextMonth - 1;
164       put endDate yymmdd10.;
165       days = nextMonth - startDate;
166       put days;
167       return;
168   endsub;
169   QUIT;
```

## How do we debug functions

```
170 data _null_;
171     format firstDay lastDay yymmdd10.
172         Numdays .;
173     call missing(firstDay, lastDay, numdays);
174     PUT 'calling routine: ';
175     call datesInMonth('15jan2009'd, firstDay, lastDay, numdays);
176     PUT 'end of routine: ';
177     put _all_;
178
179     PUT 'calling routine: ';
180     call datesInMonth('15FEB2009'd, firstDay, lastDay, numdays);
181     PUT 'end of routine: ';
182     put _all_;
183 run;
```

# How do we debug functions

calling routine:

2009-01-15

```
put inDate yymmdd10. ;
```

2009-02-01

```
put nextMonth yymmdd10.;
```

2009-01-01

```
put startDate yymmdd10.;
```

2009-01-31

```
put endDate yymmdd10.;
```

31

```
put days;
```

end of routine:

```
firstDay=2009-01-01 lastDay=2009-01-31 numdays=31 _ERROR_=0 _N_=1
```

calling routine:

2009-02-15

2009-03-01

2009-02-01

2009-02-28

28

end of routine:

```
firstDay=2009-02-01 lastDay=2009-02-28 numdays=28 _ERROR_=0 _N_=1
```

# How do we debug functions

- Functions can be called from within PROC FCMP
  - No outlib= statement
  - Useful first pass through so you do not have the overhead of writing out the function



# How do we debug functions

**184 proc fcmp ;**

**185 subroutine datesInMonth(inDate, startDate, endDate, days);**

**186     outargs startDate, endDate, days;**

**187     FILE log;**

**188     put inDate yymmdd10. ;**

**189     nextMonth = intnx('month', inDate, 1);**

**190     put nextMonth yymmdd10.;**

**191     startDate = intnx('month', nextMonth, -1);**

**192     put startDate yymmdd10.;**

**193     endDate = nextMonth - 1;**

**194     put endDate yymmdd10.;**

**195     days = nextMonth - startDate;**

**196     put days;**

**197     return;**

**198 endsub;**

## How do we debug functions

```
199  format testDate1 testDate2 yymmdd10.;
200  format days 3.;
201  call datesInMonth('15jan2009'd, testDate1, testDate2, days);
202  put 'date1: ' testDate1 ' date2: ' testDate2 ' days: ' days;
203
204  QUIT;
```

```
2009-01-15
2009-02-01
2009-01-01
2009-01-31
31
date1: 2009-01-01 date2: 2009-01-31 days: 31  put 'date1: ' testDate1
NOTE: PROCEDURE FCMP used (Total process time):
      real time          0.12 seconds
      cpu time           0.03 seconds
```

# Agenda

- ☺ What is a function?
- ☺ How do we create functions?
- ☺ How do we debug functions?
- How do we deploy functions?
- Review and Questions

## How do we deploy functions

- Functions are stored in SAS datasets
- OUTLIB=
  - Tells SAS where to save the functions
- CMPLIB=
  - Tells SAS where to find the functions

## How do we deploy functions

### ■ ***OUTLIB= work.funcs.Dates.***

- *work.funcs is the SAS dataset*
- *Dates is the function package*

### ■ **PACKAGE**

- A collection of functions and subroutines, each with a unique name
- It is possible to have the same function name in two different packages on the same data set; however, There is no way within a DATA step to specify which package to use.
- It is possible to access different versions of the same function if they are in packages in different data sets

# How do we deploy functions

## ■ Different versions.

- One group may have a larger set of parameters on some functions

- **Risk:**

- ```
call datesInMonth('15jan2009'd, testDate1, testDate2, days);  
interestPaid = monthlyInterest(balance, iRate);
```

- **Everyone else:**

- ```
call datesInMonth('15jan2009'd, testDate1, testDate2);  
interestPaid = monthlyInterest(balance, iRate);
```

## How do we deploy functions

### ■ Different datasets

- Risk specific: `allfuncs.risk.dates`
- All others: `allfunc.company.dates`.

### ■ Set the CMPLIB search path

- **Risk:**

Search order from right to left

`cmplib=( allfunc.company.dates allfuncs.risk.dates)`

Then Looks here

Looks here first

- **Everyone else:**

`cmplib= allfunc.company.dates.`

# Agenda

- ☺ What is a function?
- ☺ How do we create functions?
- ☺ How do we debug functions?
- ☺ How do we deploy functions?
- Review and Questions



# Review

- What is a function?
  - Unique mapping of inputs to output
  - Domain
  - Functions and Call routines
- How do we create functions?
- How do we debug functions?
- How do we deploy functions?
- Review and Questions

# Review

- What is a function?
  - Unique mapping of inputs to output
  - Domain
  - Functions and Call routines
- How do we create functions?
- How do we debug functions?
- How do we deploy functions?
- Review and Questions

# Review

- ☺ What is a function?
- ☺ How do we create functions?
- ☺ How do we debug functions?
- ☺ How do we deploy functions?
- Review and Questions

## Questions

***Peter Eberhardt***

**peter@fernwood.ca**



2009 Washington, DC  
22-25 March 2009