

Planning Your Mission to Mars, the Red Planet

[Part 2]

David J. Ritchie, Tim Winder, Craig Weber

©1998 by David J. Ritchie

[Note]

This was independently written to work with the Mars Simulation unit available from Interact, 5937 Darwin Court, Suite 106, Carlsbad, CA 92008.

Phone: (800)359-0961 Web: <http://www.teachinteract.com/>

The Mars Simulation is a copyrighted product of Interact.

They have not reviewed or endorsed this document.

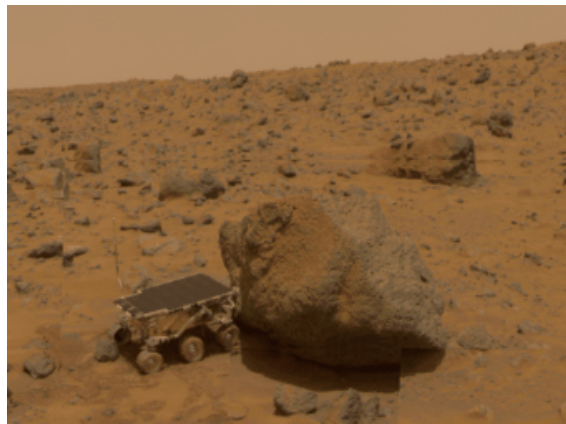
**Mars Mission Science Camp
School District 203, Naperville, IL**

July 14, 1998

Summary

This is the second of five sessions in which you will use the computer programming language "perl" to:

plan your mission to Mars, the red planet.



<http://www.jpl.nasa.gov/files/images/browse/81958b.gif>

1.0 Introduction

Remember from last time that we are trying to find a way to "pretend" to go to Mars so that we can determine whether or not our plans for food, etc., are good enough when we do our real trip.

We are doing the "pretend" by means of a computer simulation using the programming language perl.

Our goal is to teach you just enough perl that you can provide input to the Mars Mission simulation program. If you want to learn more, we will be happy to tell you how to go about doing that.

2.0 What is Perl? - Basic Lesson

Perl is a language that is used to give instructions to computers to make them do things, such as:

- to ask questions on the computer screen,
- to receive answers entered on the keyboard,
- to add, subtract, multiply, divide numbers to get results,
- to make calculations using previous numbers,
- to print the results of those calculations

Let's take each one of the above items and learn how to use perl to make your computer do it.

2.1 To Ask Questions on the Computer Screen

To ask a question on the computer screen, you use the "print" statement. Here is an example. Please try it. Make sure you type it exactly as it is written:

```
# Example p2s21e1.pl
# say welcome...
print ("Welcome to the Mars Mission Simulation!\n");
```

When you want to do something "special" like put out a carriage return, you enter a particular character preceeded by the "backslash". For example, the "backslash-n" in the above stands for "new line". Just for fun, you might try "backslash-a" to see what it does. (Hint: the "a" stands for "alarm").

Try this:

```
# Example p2s21e2.pl
# another output example
print ("Attention!\a\a\n", "Simulation is Ready\n");
```

2.2 To Receive Answers Entered on the Keyboard

To receive answer entered on the keyboard, use "standard input". This is abbreviated as <STDIN>. :

```
# Example p2s22e1.pl
# ask name of Mission Commander
print ("What is the name of the Mission Commander?\n");
$MissionCmdr = <STDIN>;
```

The above asks the question "What is the name of the Mission Commander?" and then stops and waits for input from the keyboard ("standard input"). When you type an answer and hit return, the characters you type (and the return key) get put into a thing called a variable whose name is \$MissionCmdr.

Let's print out what we received. To do that, enter the following:

```
# Example p2s22e2.pl
# ask name of Mission Commander
print ("What is the name of the Mission Commander?\n");
$MissionCmdr = <STDIN>;
print ("The Commander is $MissionCmdr, isn't it?\n");
```

In the previous example, notice that the return is printed because it was part of the contents of the \$MissionCmdr variable as it was entered.

2.3 To Add, Subtract, Multiply, Divide to Get Results

To add two numbers, do the following:

```
# Example p2s23e1.pl
# add two numbers
# $result = 4.3 + 2.4;
# show what we got
# print ("The result is: $result, isn't it?\n");
```

Notice that the output doesn't have a return anywhere in it. That's because the \$result variable wasn't entered from the keyboard where we press a return to complete the entry.

To subtract one number from another, do the following:

```
# Example p2s23e2.pl
# subtract the second number from the first
# $result = 4.3 - 2.4;
# show what we got
# print ("The result is: $result, isn't it?\n");
```

To multiply two numbers, do the following:

```
# Example p2s23e3.pl
# multiply two numbers
# $result = 4.0 * 2.0;
# show what we got
# print ("The result is: $result, isn't it?\n");
```

To divide two numbers, do the following:

```
# Example p2s23e4.pl
# divide two numbers
# $result = 4.0 / 2.0;
# show what we got
# print ("The result is: $result, isn't it?\n");
```

2.4 To Make Calculations Using Previous Numbers

To make using previous numbers, you need to remember the previous numbers in variables.

You create the variables by simply using them. You can name the variables almost anything you want. The case of the name is important. The variable \$TotalFood is different from the variable \$totalfood so be careful with that.

Variables that hold a single number or a single group of letters begin with a \$. You can check the value of a variable and do something different if it is less than or equal to zero (for example) by using if statements.

Do the following:

```
# Example p2s24e1.pl
# remember numbers in variables
$TotalFood = 10.0;
$DailyFood = 12.0;
#
# Calculate with those numbers
$TotalFood = $TotalFood - $DailyFood;
# show what we got
print ("The Total Food is: $TotalFood, isn't it?\n");
# use the result
if ($TotalFood <= 0) {print ("Egad!, no food\a\a\a\n");}
```

3.0 What is Perl? - Intermediate Lesson

You saw above how you could do basic things with Perl. You can also do things at an intermediate level.

For example, you can:

- Make loops,
- use other people's code, provided that they write the code in the form of a subroutine,

Let's take each one of the above items and learn how to use perl to make your computer do it.

3.1 Make Loops

Here's an example of a loop. See if you can figure out what it does:

```
# Example p2s31e1.pl
# set how much food we've got
$TotalFood = 10.0;
$DailyFood = 2.0;
#
# for each day that goes by (up to ten days)
foreach $Day (1..10) {
    $TotalFood = $TotalFood - $DailyFood;
    print ("Day $Day: Food left: $TotalFood \n");
    if ($TotalFood <= 0) {print ("Egad!, no food\a\a\a\n");}
}
```

3.2 Call Subroutines

When someone else (such as your perl consultant) writes perl code, you can use it very easily provided it is in the form of a subroutine.

You and the person writing the subroutine must agree on the names of variables. In the example below, we have decided that we will both use the variables \$TotalFood, \$DailyFood and \$Day.

You put values into the variables. Then, you call the subroutine simply by using it in a perl statement. When you do that, it is as if you wrote all that code yourself and put it in the place where you made the subroutine call.

In the example below, you have provided a certain amount of food, water, and oxygen for a woman crew member. You wish to know if it is enough to last for ten days. So, you call the EatFood, DrinkWater, and BreatheOxygen subroutines ten times.

Will it be enough? Did you provide correct data for the simulation? Does the simulation do the right thing? What do you think?

```
# Example p2s32e1.pl
# say how much total food, water and oxygen we've got
  $TotalFood    = 20.0;
  $TotalWater   = 40.0;
  $TotalOxygen  = 20.0;
#
# say how much a woman crew member needs each day
  $DailyFood    = 2.55;
  $DailyWater   = 4.25;
  $DailyOxygen  = 1.70;
  $AlarmFood    = 0;
  $AlarmWater   = 0;
  $AlarmOxygen  = 0;
```

```

#
#   say we are beginning the Human Factors Simulation
print ("Beginning Human Factors Simulation\n",
      "--Woman Crew Member.\n\n");
#
#   for each day from day 1 through day 10,
#   pretend to eat, drink and breathe
foreach $Day (1..10) {
    EatFood();
    DrinkWater();
    BreatheOxygen();
}
#
#   print food, water and oxygen at the end of 10 days
print ("At the end of ten days, amounts left are:\n",
      "Food: $TotalFood\n",
      "Water: $TotalWater\n",
      "Oxygen: $TotalOxygen\n");
exit;

```

```

#####
#   Subroutines
sub   EatFood {
#
#   Compute total food now left
    $TotalFood = $TotalFood - $DailyFood;
#
#   if we have eaten all the food, print out the day
    if ($TotalFood <= 0 and $AlarmFood == 0) {
        $AlarmFood = 1;
        print ("Egad!, no food\a\n");
        print ("Day $Day:  No food left!\n");
    }
    return;
}
#####
# subroutine to Drink Water
sub   DrinkWater {
#
#   Compute total water now left
    $TotalWater = $TotalWater - $DailyWater;
#
#   if we have drunk all the water, print out the day
    if ($TotalWater <= 0 and $AlarmWater == 0) {
        $AlarmWater = 1;
        print ("Egad!, no water\a\n");
        print ("Day $Day:  No water left!\n");
    }
    return;
}
}

```

```

#####
# subroutine to breathe oxygen
sub BreatheOxygen {
#
#   Compute total food now left
    $TotalOxygen = $TotalOxygen - $DailyOxygen;
#
#   if we have breathed all the oxygen, print out the day
    if ($TotalOxygen <= 0 and $AlarmOxygen == 0) {
        $AlarmOxygen = 1;
        print ("Egad!, no oxygen\n");
        print ("Day $Day:  No oxygen left!\n");
    }
    return;
}
}

```