



<http://www.perl.com>

Beginning Programming in Perl

copyright 1999 Edward Irvine. Freely distributable for educational use. Contact the author for commercial (magazines etc) use.

In this unit of study, you will learn how to write some simple programs in a language called "**Perl**" (Practical Extraction and Reporting Language). Perl is the language behind most interactive web pages - such as Hotmail and Yahoo!. Perl may seem a little weird at first but if you stick with it you may find you really like it (honest!). In fact, Perl was designed to be easy to learn.

Perl also has some very powerful features you can investigate yourself such as **objects** and **regular expression processing**. Perl has an unparalleled ability to integrate with internet services such as email, database, and html. Some of you may chose to use Perl with software projects further down the track.

Right now we are just interested in writing **simple** programs. Here is the path we will take:

1. Perl - copying out three Perl programs and getting them to run:
2. Perl - getting **input** and doing some Fortran.
3. Perl - selection: **if, if - else**. Nested if.
4. Perl - repetition: the **while** loop.
5. Perl - repetition: the **do** loop.

I have some expectations of you during this unit of work:

- I expect you to try.
- I expect you to not give up on the first attempt. Maybe your friend has worked out how to do it?
- I expect you not to copy other people's work. You may, however, copy their ideas, so long as you understand them.
- At the end of this unit of work, I want a neatly formatted print-out of all the programs you have written. This should be stapled and handed in.

For many of you, this unit of work will be a like going to the gym. You'll be working on certain parts that aren't used to being "worked out"; The only difference is that here we will be exercising the brain instead of biceps.

It might hurt. Remember: this is natural!

Perl: Day one.

This is an exercise to get you familiar with the programming environment. As you struggle to get these three programs running, you will learn about the importance of correct **syntax** as well. When you get a program running, save it as a separate file.

Program 1: *hello_world.plx*

```
# My first program in perl.
print "Hello, world!\n";
```

Add a few extra lines of your own to this, so that the computer outputs some poetry. eg:

```
print "Oh what a beautiful morning!\n";
print "Walk softly and carry a big fish!\n";
```

Here is a program that gets input from the user. <STDIN> is short for the "standard input" - in your case it is the keyboard. \$name is a **variable** that holds the string of characters that you type in, "chomp" strips the return key from the end of the string.

Program 2: *your_name.plx*

```
print "A program to ask you your name.\n" ;
print "Please enter your name: " ;
$name = <STDIN> ;
chomp ($name) ;
print "Hello, $name . Jolly good to meet you!\n" ;
```

See how the value of \$name is **substituted** into the print string? What do you think the output would look like if we didn't use the chomp function?

Program 3: *decision.plx*

This program makes a decision using the **if** statement. See how the program flow is divided up into "**blocks**" by the { and } symbols?

```
print "What is your name?" ;
$name = <STDIN> ;
chomp ($name) ;
if ($name eq "Murgatroid" ) {
    print "Ha! Weird name!\n" ;
    print "Goodbye!" ;
} else {
    print "Hello, $name . \n" ;
}
```

Perl: Day 2 and 3.

In perl, a "\$" variable can hold either a **string** or a **number** . Today we are going to learn about variables and variable substitution.

Here are some examples of **number** assignments (these lines are individual examples of assignments, not a program. So don't copy them out!)

```
$pi = 3.1415962 ;
$x = 200;
$y = -3;
$twopi = 2*3.1415962;
$another_two_pi = 2 * $pi ;
$x = (-$b + sqrt($b*$b - 4*$a$c))/(2*$a);
```

See how these assignments assigned a number to a variable? How many of these assignments used "variable substitution"? Did you copy out the lines above? Try to execute them as a Perl program? No? Good.

These assignments won't work properly:

```
$WeeklyAllowance = $2.50;           # error!
$test_percent = %99;                 # ditto!
```

Numeric assignments can only work with calculator style numbers.

Right then! Now for some programming tasks:

You will now have to do something that we often don't teach very well at school: Think for your self (!). Like going to the gym or riding a horse, this can be painful for the newbie.

Before you can teach a computer to do something, you will to have to be pretty certain about how to do it yourself. Do the task yourself and watch while you do it. You might try designing your program with paper and pencil and eraser. Accept that you will use the eraser a lot.

Write a program that:

1. Dollars.plx. Asks you for an amount in dollars and tells you what it was worth in cents.
2. Circle1.plx . Assigns the value 3.14159 to a variable called \$pi. It then works out and prints the area and circumference of a circle of radius 500 mm.
3. Circle2.plx. As above, but this time it asks you the value of the radius.
4. Average1.plx. Works out and prints the average of 7, 9, and 15.
5. Average2.plx. Like Average1.plx but it inputs three numbers from the user.
- *And for the intellectual Sumo out there* -
6. Cylinder1.plx: Works out the area and volume of a cylinder 12m high and 2m wide.
7. Cylinder2.plx: As above but asks the user for the width and height.
8. Quadratic1.plx: Asks the user for a, b and c and gives both the roots, if any.

Perl: Day 4.

Variables can contain **strings** as well as **numbers**. Here are some examples of **string** assignments (these lines are individual examples, not a program. Don't copy them out).

```
$name = "Fredreekah";    # set $name to the "Fredreekah"
$name = <STDIN>;         # get $name from the keyboard.
chomp($name);           # and strip the Enter Key.
$initial = "F";
$x = "48 crash";
$y = "" ;               #ie: nothing;
$z = " " ;              #ie: a single space;
$first = "Murgatroid";
$last = "Muggins";
$fullname = "John" . " " . "Smith";
#Important to know! The period "." operator joins strings.
$fullname2 = $first . " " . $last;
$TwoNames = $fullname . " " . $fullname2;
# Question to ponder: What is $TwoNames now?
```

You're not going to go off and copy out the lines above now, are you? Good.

Write a program that:

1. `FirstName.plx`. Inputs your first name and prints it out.
2. `SecondName.plx` Asks you your first name, then asks you your second. It then prints your full name.
3. `ExamAverage.plx`. Asks you your first name, then your second name, and then your last three exam percentages. It Tells you your full name and what your average was.

And for the intellectual giants who are way, way ahead:

4. `Naughty.plx`. This is a line writing tool for naughty students. It prints out 100 lines of "I must resist the urge to tease my teachers. " (Hint use a while loop).
5. `NaughtyBad.plx`. This is a more flexible line writing tool. It asks the user what the contents of the line is to be, and how many lines are required. Then it prints them.
6. Can you find out how to print those lines to a text file instead of the screen (hint: look for a function called `open`).

Perl. Day 5

Selection.

So far all the programs you have done have simply started up the top and done each statement, one after the other. This is **sequence**. Now it is time for some **if** statements - these introduce the concept of **selection** or program branching. Sharpen your pencils. . .

Logical Operators:

To compare numbers:

<	Less than.	>	_____
<=	_____	>=	_____
==	_____	!=	_____

And to compare strings:

lt	Less than.	ne	_____
le	_____	ge	_____
eq	_____	nt	_____

Boolean Operators:

&& And || Or (no spaces between)

But first, some logic exercises to clear up some cobwebs:

True or False?

(5 < 6)	(5 >= 5)	("Costello" gt "Abbott")
True && True	True && False	False && True
False && False	True True	True False
False True	False False	True && True && True
((7 < 10) && (5 != 2))	((3 == 2) ("hello" lt "hi"))	

Perl Day 5 (continued)

Here is a snippet with an example of an if block:

```
if ($day eq "Thursday") {
    $balance = $balance + 300;
    $task = "Late Night Shopping with Granny";
}
print "your balance is $balance and your task is $task\n";
```

Here is another snippet:

```
if ($balance < 100) {
    $venue = "McDonalds";
    $balance = $balance - 10;
} else {
    $venue = "Maxims";
    $balance = $balance - 80;
}
```

And a complete program:

```
#-----
# get the bank balance
print "Enter your bank balance:";
$balance = <STDIN>;
chomp($balance);
#now decide on a venue:
if ($balance < 50) {
    $venue = "Hmm. Roast Lamb at Mums";
} else {
    print "Good to see you are not totally broke\n";
    if($balance < 100) {
        $venue = "McDonalds";
        $balance = $balance - 10;
    } else {
        $venue = "Maxims\n";
        $balance = $balance - 80;
    }
}
#Tell them where to go:
print "Your venue tonight is $venue and your bank balance";
print " will be $balance\n";
print "goodbye!\n"
#-----
```

Pause for a second. Wait until you really understand the sample program above. See how an else is optional?

Perl Day 6 and 7

Did you **really** understand the sample program above? Good.

Tasks:

1. Temperature1.plx Write a program that inputs the temperature and prints "Too Hot" if the temperature is above 30 and "Too Cold" if below.
2. Temperature2.plx As above, but prints "Just right" above 20 and below 30.
3. YouPassed.plx Asks you for your mark. If you got above 80, it prints "You passed". If you got 80 or below, the program exits without saying anything.
4. BlindDate.plx A program that asks a few questions and recommends if you should go on the blind date or not. Use your imagination, but keep it clean.
5. LeapYear.plx A leap year is every fourth year. Thus, 1904 was a leap year. However, century years must pass an extra test: they must be divisible by 400 as well. Thus, 1900 was not a leap year, but 2000 was. Your program should ask you for a year and then tell you if it is a leap year or not.

(Hint: the modulus operator % gives the remainder of a division.

Thus after the statement

```
$x = 13 % 10 ;
```

\$x is now 3.

Perl: Day 8

Repetition is one of the most common programming structures. In perl, we use two types of loops: **while**, and **do-while**. (There are others as well).

For example:

```
print "Count down from what number?\n";
$a = <STDIN>;
chomp $a;
while ( $a >= 0 ) {
    print "$a \n";
    $a = $a - 1;
}
```

Second example:

```
$x = 0;
do {
    print "Your score is $x\n";
    if ($x == 50) {
        print "Half century!\n";
    }
    if ($x == 100) {
        print "You have scored a century.\n";
        print "Take a bow!\n";
    }
    $x = $x + 1;
} while ($x <= 120);
printf "Goodbye!\n";
```

You can have loops (inside loops (inside loops (inside loops...))).

Tasks:

1. Write a program that prints a list of numbers from 0 to 1000.
2. Write a program that prints a list of even numbers and their squares from 0 to 1000. Output should look like so:

0	0
2	4
4	16 etc etc
3. Write a program that reads a list of numbers until 999 is read and then prints the total and average of all the numbers added together (be sure not to add the 999!).
4. Write a program that has the user guess a computer generated random number between 0 and 200. The computer will respond to an incorrect guess by saying "Too high" or "Too low".