

TRIM A STRING

chop *VALUE* – Delete the last character from a string or list.
chomp *VALUE* – Delete any non-text line endings from a string or list.

The chop and chomp functions allow you to remove characters from strings. These functions are commonly used when working with strings that have been read from files or will be written to files.

Use the chop function to remove a single character from the end of a string. The chop function returns the value of the removed character. When you use the chop function with a list, the function will remove a character from the end of each string in the list, but will only return the value of the character removed from the last string.

The chomp function is similar to the chop function, but only removes non-text line ending characters,

such as newlines. Using the chomp function with a list will remove a line ending character from the end of every string in the list. The chomp function returns a result that indicates the total number of line ending characters removed from the list.

The chomp function is generally considered safer to use than the chop function since it does not remove any text data.

If you do not specify a variable for the chop or chomp function, the function will use the value of the default special variable, \$_.

Extra

The default special variable, \$_, temporarily stores data when no variable is specified. To save time when using the chop or chomp function with a string, assign the string to the \$_ variable. This allows you to omit the name of the variable when using the function, though it may make your code harder to read and understand.

TYPE THIS:

```
$_ = "abcdefg";
chop;
print;
```

RESULT:

```
abcdef
```

By default, the chomp function removes only one newline character from the end of a string. To have the chomp function remove all newline characters from the end of a string, use the special variable \$/ set to the null string "". This is useful for eliminating blank lines when working with strings that have been read from files.

TYPE THIS:

```
@names = ("Mary\n\n", " Pat");
$/="";
chomp @names;
print @names;
```

RESULT:

```
Mary Pat
```

To display the value returned by the chop or chomp function, assign the function to a variable and then display the results of the variable. This lets you see the character removed by the chop function or the number of line ending characters removed by the chomp function.

TYPE THIS:

```
@list = ("Jill\n", "Ronda\n", "April\n");
$removed = chomp @list;
print "$removed characters were removed.";
```

RESULT:

```
3 characters were removed.
```

USING THE CHOP FUNCTION

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

$message = "abcd";

chop $message;

print $message;
```

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

[scripts]\$ script.pl

abc

[scripts]\$

- 1Type the code that creates a variable to store the string or list you want to remove characters from.
- 2Type **chop** followed by the variable. Then type a semi-colon (;).
- 3Position the cursor where you want to type the code that uses the result of the chop function and type the code.
- 4Save and execute the script.
- Perl generates the result of using the chop function. The last character has been removed from the string(s).

USING THE CHOMP FUNCTION

UW PICO(tm) 3.5File: script.plModified

```
@names = ("Lindsay \n", " Rev \n", " Kyle \n");

chomp @names;

print @names;
```

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

[scripts]\$ script.pl

Lindsay Rev Kyle

[scripts]\$

- 1Type the code that creates a variable to store the string or list you want to remove line ending characters from.
- 2Type **chomp** followed by the variable. Then type a semi-colon (;).
- 3Position the cursor where you want to type the code that uses the result of the chomp function and type the code.
- 4Save and execute the script.
- Perl generates the result of using the chomp function. A line ending character has been removed from the end of the string(s).

FIND THE LENGTH OF A STRING

length *STRING* – Determine the number of characters in a string.

Perl's `length` function allows you to determine the number of characters in a string. The `length` function is often used to find the length of a string stored in a variable.

If the string you want to find the length of contains spaces or punctuation marks, the `length` function will count the spaces and punctuation marks as characters. The `length` function also counts any special characters in a string, such as tabs or newlines. For example, the `length` function would count six characters in the string `"hello\n"`.

If you use the `length` function with a numeric value, Perl will automatically convert the number to a string. If the number includes a decimal point, the decimal point will also be counted as a character. For example, the `length` function would count nine characters in the number `123456.78`.

If you do not specify a string in the `length` function, the function will return the length of the value stored in the default special variable, `$_`.

Extra

You can find the size of an array. To do so, you must evaluate the array in *scalar context*. The `scalar` function can be used to evaluate an array in scalar context and return the number of elements in the array.

TYPE THIS:

```
@newClients = ("Dana Lesh", "David Gregory", "Barry Pruett");
print scalar(@newClients);
```

RESULT:

3

You can find the number of elements in a *hash*. To find the size of a hash, the hash must be evaluated in *scalar context*. The `scalar` and `keys` functions can be used to evaluate a hash in scalar context and return the number of elements in the hash.

TYPE THIS:

```
%newClientsByLast = qw(Corder Mary Kyle Looper Harris Mark Johnson Jill);
print scalar keys %newClientsByLast;
```

RESULT:

4

FIND THE LENGTH OF A STRING

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

$name = "Tim Borek\n";

length
```

Get HelpExitWriteOutJustifyRead FileWhere isPrev PgNext PgCut TextUnCut TextCur PosTo Spell

1 Type the code that creates a variable and assigns it a value.

2 Position the cursor where you want to find the length of a string and type **length**.

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

$name = "Tim Borek\n";

length $name
```

Get HelpExitWriteOutJustifyRead FileWhere isPrev PgNext PgCut TextUnCut TextCur PosTo Spell

3 Type the name of the variable that stores the string you want to find the length of.

You can also type a string. Strings must be enclosed in quotation marks.

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

$name = "Tim Borek\n";

print "The length of the string is: ", length $name;
```

Get HelpExitWriteOutJustifyRead FileWhere isPrev PgNext PgCut TextUnCut TextCur PosTo Spell

4 Position the cursor where you want to type the code that uses the `length` function and type the code.

[scripts]\$ script.pl

The length of the string is: 10

[scripts]\$

5 Save and execute the script.

Perl generates the result of finding the length of a string.

CHANGE CASE OF A CHARACTER OR STRING

- lc** *STRING* – Change all text in a string to lowercase.
- lcfirst** *STRING* – Change first character in a string to lowercase.
- uc** *STRING* – Change all text in a string to uppercase.
- ucfirst** *STRING* – Change first character in a string to uppercase.

Perl provides functions that allow you to change the case of text in a string. Changing all the text in a string to the same case can help make the text easier to search and manage.

Use the **lc** function to change all the text in a string to lowercase. To change only the first character in a string to lowercase, use the **lcfirst** function.

To change all the text in a string to uppercase, use the **uc** function. The **ucfirst** function allows you to change the first character in a string to uppercase.

When using the **lcfirst** and **ucfirst** functions, keep in mind that Perl will apply the function to the first character in a string, even if the first character is not a letter. For example, if the first character in a string is a space or period, the string will not display any changes.

If you do not specify a string for the **lc**, **uc**, **lcfirst** or **ucfirst** function, the function will use the default special variable, **\$_**.

Extra

The **lc**, **lcfirst**, **uc** and **ucfirst** functions can be used together to apply the results of one case function to another case function. This lets you change the case of a string in more than one way. For example, applying the **lc** function to a string and then applying the **ucfirst** function allows you to ensure only the first letter is uppercase.

TYPE THIS:

```
$wrongCaseName = "pHiLLiP";  
print ucfirst(lc($wrongCaseName));
```

RESULT:

Phillip

The **\u**, **\l**, **\U**, **\L** and **\E** backslash escape sequences can be used for changing the case of text in a literal string. Use **\u** and **\l** to change the character immediately following the backslash escape sequence to uppercase or lowercase, respectively. Characters between the **\U** and **\E** escape sequences will all be converted to uppercase, while all the characters between the **\L** and **\E** escape sequences will be converted to lowercase.

TYPE THIS:

```
print "a tree grows IN BROOKLYN";  
print "\n";  
print "\ua \Utree\E grows \lIN \LBROOKLYN\E";
```

RESULT:

```
a tree grows IN BROOKLYN  
A TREE grows iN brooklyn
```

CHANGE TO LOWERCASE

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl  
print lc "THIS IS LINE 1\n";  
print lcfirst "THIS IS LINE 2\n";
```

^G Get Help^O WriteOut^R Read File^Y Prev Pg^K Cut Text^C Cur Pos
^X Exit^J Justify^W Where is^V Next Pg^U UnCut Text^T To Spell

```
[scripts]$ script.pl  
this is line 1  
tHis IS LINE 2  
[scripts]$
```

- 1

To change an entire string to lowercase, type **lc** followed by the string enclosed in quotation marks.
- 2

To change the first letter of a string to lowercase, type **lcfirst** followed by the string enclosed in quotation marks.
- 3

Position the cursor where you want to type the code that generates the result of using the **lc** and **lcfirst** functions and type the code.
- 4

Save and execute the script.
- Perl generates the result of using the **lc** and **lcfirst** functions to change the case of characters.

CHANGE TO UPPERCASE

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl  
print uc "this is line 3\n";  
print ucfirst "this is line 4\n";
```

^G Get Help^O WriteOut^R Read File^Y Prev Pg^K Cut Text^C Cur Pos
^X Exit^J Justify^W Where is^V Next Pg^U UnCut Text^T To Spell

```
[scripts]$ script.pl  
THIS IS LINE 3  
This is line 4  
[scripts]$
```

- 1

To change an entire string to uppercase, type **uc** followed by the string enclosed in quotation marks.
- 2

To change the first letter of a string to uppercase, type **ucfirst** followed by the string enclosed in quotation marks.
- 3

Position the cursor where you want to type the code that generates the result of using the **uc** and **ucfirst** functions and type the code.
- 4

Save and execute the script.
- Perl generates the result of using the **uc** and **ucfirst** functions to change the case of characters.

DIVIDE A STRING

`split /PATTERN/, string, limit` – Divide a string into substrings.

The `split` function allows you to divide a string into smaller parts, called substrings. Strings are often divided to more easily process lines of information read from a file.

You must specify a delimiter to tell Perl where you want to divide a string. When used with the `split` function, the delimiter is a *regular expression* and can be any character or multiple characters. Perl will perform a pattern search in a string for the delimiter and then divide the string at the delimiter.

Perl returns the substrings it finds in a list and removes the delimiter you specified. You may

want to store the substrings as elements in an array to help make the data easier to work with. For information about arrays, see page 28.

The number of substrings Perl returns depends on the number of times Perl finds the delimiter in the string. For example, if Perl finds one instance of the delimiter, two substrings will be returned. The first substring would contain all the characters up to the delimiter and the second substring would contain all the characters after the delimiter.

If a string does not contain the delimiter you specified, Perl will return the entire string.

Apply It

You can use a positive integer to specify a limit for the `split` function. The limit you specify determines the maximum number of substrings Perl divides the string into.

```
TYPE THIS:
$nameList = "Tom:Steve:Bill:Ed:Lou:George:Phil";
@limitSplit = split /:/, $nameList, 3;
print "@limitSplit";
```

```
RESULT:
Tom Steve Bill:Ed:Lou:George:Phil
```

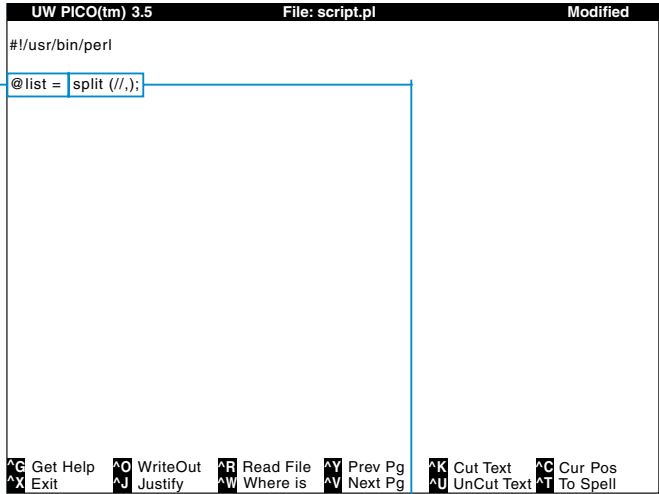
You can use the `split` function without specifying a delimiter or a string. If you do not specify a delimiter, Perl will divide the string at every whitespace character. Whitespace characters are characters that represent a separation between words, such as newlines, tabs and spaces.

If you do not specify a string for the `split` function, Perl will divide the string currently stored in the special variable `$_`.

```
TYPE THIS:
$_ = "Once upon a time.";
@splitList = split;
foreach (@splitList) { print; print " ^ "; }
```

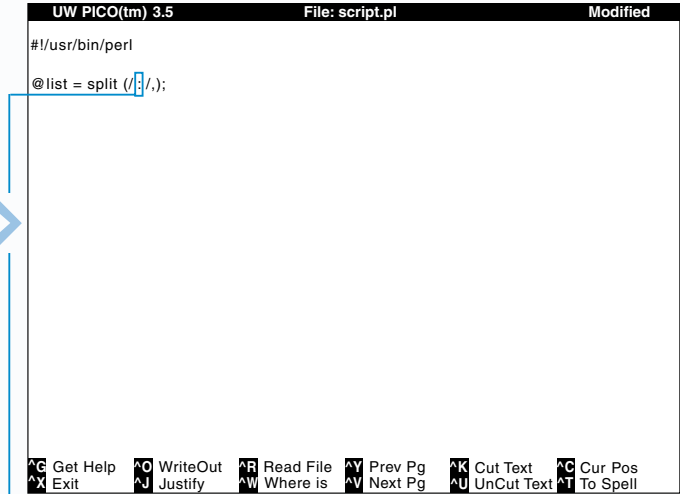
```
RESULT:
Once ^ upon ^ a ^ time. ^
```

DIVIDE A STRING



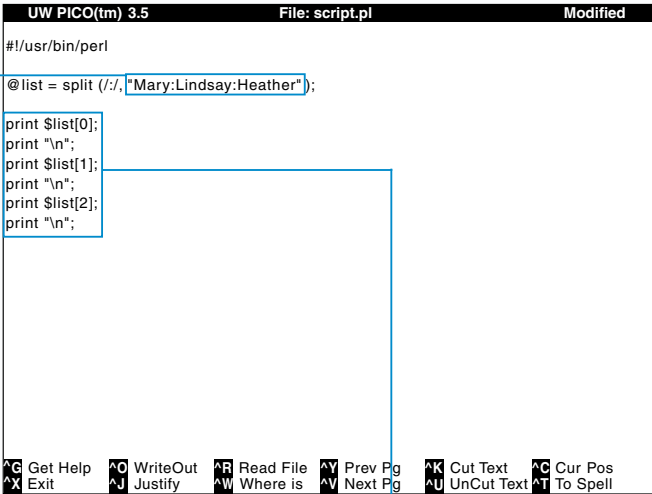
1 To store the list of substrings Perl returns in an array, type a name for the array followed by `=`.

2 Type `split (/ ,);`.



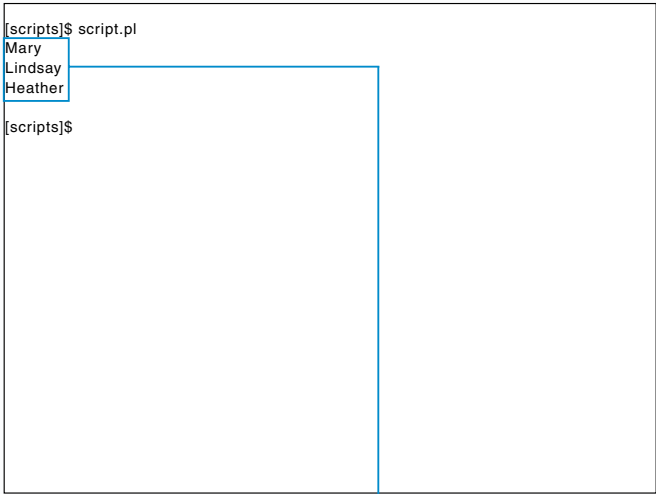
3 Position the cursor over the second forward slash (/) and type the character you want to use as a delimiter.

Note: You can specify multiple characters for a delimiter.



4 Position the cursor over the closing parenthesis and type the string you want to divide, enclosed in quotation marks.

5 Position the cursor where you want to type the code that accesses each substring and type the code.



6 Save and execute the script.

Perl generates the result of dividing a string.

FORMAT A STRING

`sprintf` *FORMATSTRING, LIST* – Format a string.

The `sprintf` function formats strings. The `sprintf` function requires two arguments—a format string and a list of values you want to format. A format string contains the formatting rules that will be applied to the values. Each formatting rule in the format string must have a corresponding value in the list.

Formatting rules are specified using a percentage symbol (%) and a letter that indicates the type of formatting to be applied. Formatting rules may also include numbers that indicate a minimum width or a precision for values. For example, the formatting rule `%8s` formats a string value so it will have a minimum

width of eight characters. The formatting rule `%.2f` formats a floating point number with a precision of two decimal places. This is useful for lining up text and numbers, improving the appearance of data.

A format string can also contain characters you do not want to format. For example, to include text in a string with formatted numbers, type the text in the format string. Perl will replace the formatting rules in the format string with the formatted numbers.

The `sprintf` function uses the same formatting rules as the `printf` function. For information about the `printf` function, see page 114.

Extra The following chart displays letters that you can use to indicate the type of formatting you want to apply.

LETTER:	MEANING:
c	Single character, by value
d	Decimal integer (2, -5)
e	Scientific notation, lowercase (2.53e-19)
E	Scientific notation, uppercase (2.53E-19)
f	Floating-point number (450.067)
g	Floating-point number, scientific notation, lowercase (2.92e-7)
G	Floating-point number, scientific notation, uppercase (2.92E-7)
ld	Long decimal integer (47483647)
lo	Long octal integer (723452365)
lu	Long unsigned decimal integer (42683201)
lx	Long hexadecimal integer (4ad3c2ff)
o	Octal integer (276)
s	String ("hello there")
u	Unsigned decimal integer (298)
x	Hexadecimal integer, lowercase (2af)
X	Hexadecimal integer, uppercase (2AF)

FORMAT A STRING

UW PICO(tm) 3.5File: script.plModified

#!/usr/bin/perl

\$message = sprintf "%8s %7.2f %3d \n";

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

UW PICO(tm) 3.5File: script.plModified

#!/usr/bin/perl

\$message = sprintf "%8s %7.2f %3d \n", "Jill";

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

UW PICO(tm) 3.5File: script.plModified

#!/usr/bin/perl

\$message = sprintf "%8s %7.2f %3d \n", "Jill", 13.4567, 10;

print \$message;

\$message = sprintf "%8s %7.2f %3d \n", "Maureen", 11.4823, 5;

print \$message;

\$message = sprintf "%8s %7.2f %3d \n", "David", 10.2369, 12;

print \$message;

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

[scripts]\$ script.pl

Jill 13.46 10

Maureen 11.48 5

David 10.24 12

[scripts]\$

- 1 Type the variable that will store the result of the `sprintf` function.
- 2 Type `= sprintf "";`
- 3 Position the cursor over the closing quotation mark and type the formatting rules you want to use.
- Note: You can also include characters you do not want to format.

- 4 Position the cursor over the semi-colon and type a comma (,) followed by the value you want to use the first formatting rule you specified.
- Text values must be enclosed in quotation marks.

- 5 Repeat step 4 for each formatting rule you specified.
- 6 Position the cursor where you want to type the code that uses the string and type the code.

- 7 Save and execute the script.
- Perl generates the result of using the `sprintf` function.

EXTRACT A SUBSTRING

substr *STRING, POSITION, length, replacement* – Extract a substring from a string.

The `substr` function allows you to access a specific part of a string, called a substring.

You must specify the name of the string containing the substring you want to extract. The position of the substring must also be specified. Perl finds the position of a substring by counting the number of characters from the beginning of the string. Characters in a string are numbered starting at zero (0). Keep in mind that if the value of the special variable `$[` has been changed, the string characters will not be numbered starting at 0. For more information about special variables, see page 34.

Although not necessary, you can tell Perl the length of a substring you want to extract. If you do not specify the length of a substring, the `substr` function will return a substring that begins at the position you specified and ends at the end of the string.

The `substr` function also allows you to specify replacement text for a substring, which is useful when you want to alter the contents of a string. The replacement text does not have to be the same length as the substring.

Apply It

When using the `substr` function, a negative number can be used to specify the position of a substring. If you specify a negative number for the position of a substring, Perl locates the substring by counting from the end of the string instead of the beginning.

You can use a negative number to specify the length of the substring you want to extract. When a negative number is used, Perl removes that number of characters from the end of the substring.

TYPE THIS:

```
$string = "This is a string.";
$negpos = substr($string, -17, 4);
print $negpos;
```

RESULT:

This

TYPE THIS:

```
$string = "This is a string.";
$neglen = substr($string, 0, -7);
print $neglen;
```

RESULT:

This is a

EXTRACT A SUBSTRING

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl
$string = "This is a string";
print substr($string, 4);
print "\n\n";
```

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

- 1 Type the code that creates the string variable and assigns it a value.
- 2 To extract a substring, type `substr();`.
- 3 Position the cursor over the closing parenthesis and type the name of the string followed by a comma (,). Then type the position of the substring you want to extract.
- 4 Position the cursor where you want to type the code that uses the substring and type the code.

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl
$string = "This is a string";
print substr($string, 4);
print "\n\n";
print substr($string, 4, 3);
print "\n\n";
```

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

- 5 To extract a substring of a specific length, perform steps 2 and 3.
- 6 Position the cursor over the closing parenthesis and type a comma (,) followed by the length of the substring you want to extract.
- 7 Position the cursor where you want to type the code that uses the substring and type the code.

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl
$string = "This is a string";
print substr($string, 4);
print "\n\n";
print substr($string, 4, 3);
print "\n\n";
substr($string, 10, 6, "line of text");
print $string;
```

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

- 8 To extract a substring of a specific length and specify replacement text, perform steps 5 and 6.
- 9 Position the cursor over the closing parenthesis and type a comma (,) followed by the replacement text enclosed in double quotation marks.
- 10 Position the cursor where you want to type the code that uses the new string and type the code.

[scripts]\$ script.pl

is a string

is

This is a line of text

[scripts]\$

- 11 Save and execute the script.
- Perl generates the result of extracting substrings.

SEARCH FOR A SUBSTRING

index *STRING, SUBSTRING, position* – Find the location of a substring.
rindex *STRING, SUBSTRING, position* – Find the location of a substring, searching from the end of a string.

The `index` and `rindex` functions allow you to find the location of a substring in a string. To search for a substring, you must specify the string you want to search and the substring you want to search for.

When determining the starting position, keep in mind that the characters in a string are numbered starting at zero (0). However, if the value of the special variable `$[` has been changed, the character numbering may not begin at 0.

Use the `index` function to start a search from the beginning of a string. To have a search begin from the end of a string, use the `rindex` function. The `rindex` function is useful when you want to determine the location of the last occurrence of a substring in a string.

When the `index` and `rindex` functions are used, Perl returns a value that indicates the location of the substring you searched for. A return value of 0 indicates that the substring is located at the beginning of the string. If Perl cannot find the substring, the return value will be -1.

Although not necessary, Perl allows you to specify a starting position when searching for a substring.

Apply It

You may want to use the `substr` function to extract a substring that is a specific number of characters from a substring you find using the `index` or `rindex` function. In the example below, we extract a directory name from a URL. The `rindex` function is used to find the last occurrence of a forward slash (/) in the string. We then add 1 to the return value and use the new value as the position where the `substr` function will extract the directory name.

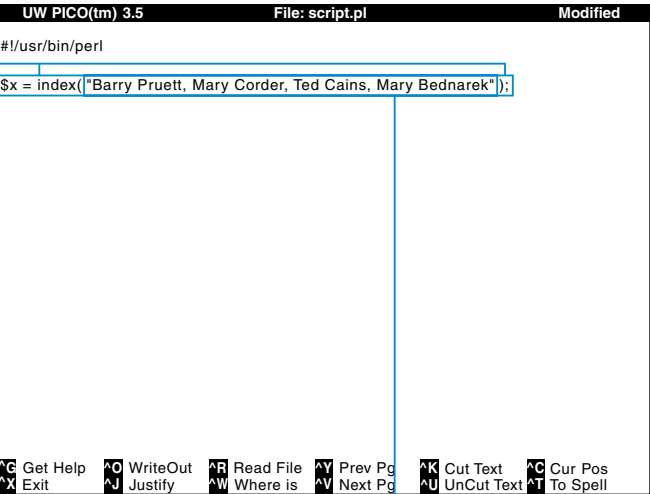
TYPE THIS:

```
$salesUrl = "http://www.abccorp.com/Sales2000";  
$position = rindex($salesUrl, "/") + 1;  
$directory = substr($salesUrl, $position);  
print "URL = $salesUrl\n";  
print "Directory name = $directory\n";
```

RESULT:

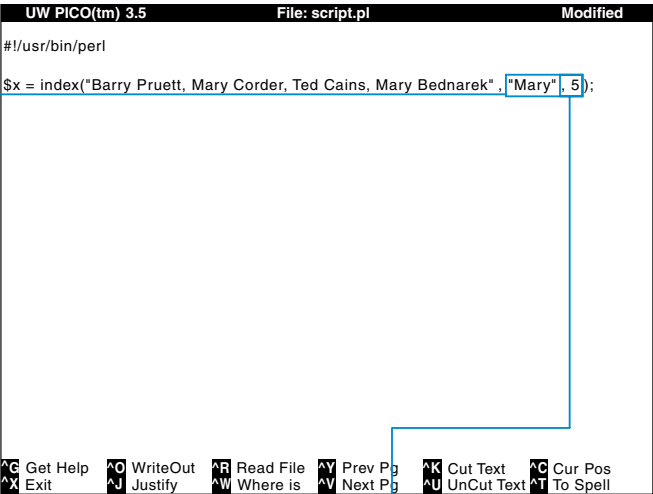
```
URL = http://www.abccorp.com/Sales2000  
Directory name = Sales2000
```

SEARCH FOR A SUBSTRING



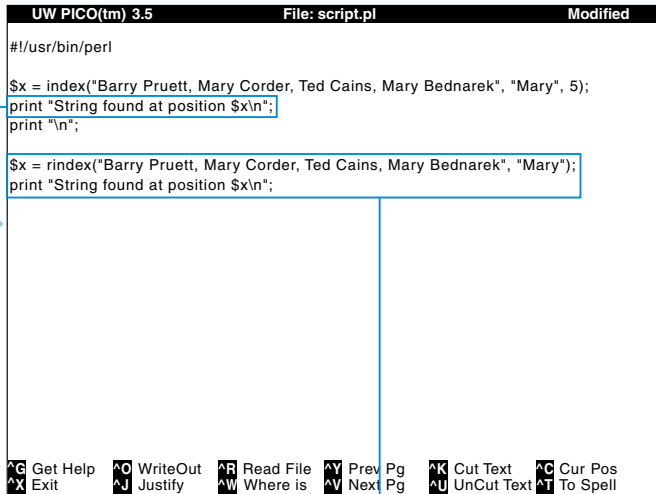
1 To begin searching for a substring from the beginning of a string, type the name of the variable you want to use to store the return value followed by `= index()`;

2 Position the cursor over the closing parenthesis and type the string you want to search, enclosed in quotation marks.



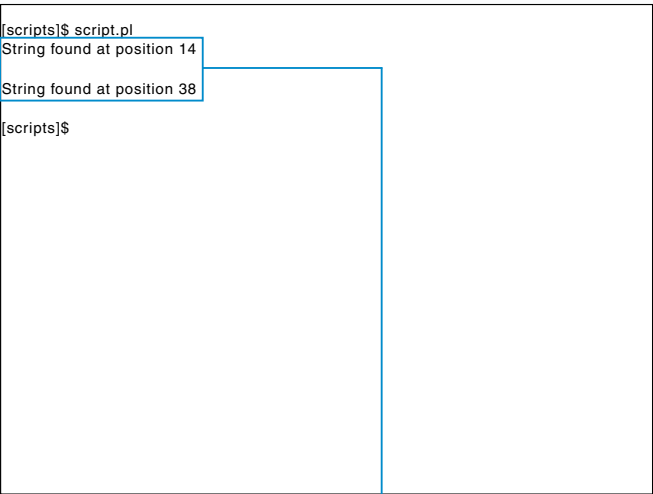
3 Type a comma (,) followed by the substring you want to search for, enclosed in quotation marks.

4 To specify a starting position for the search, type a comma (,) followed by the position in the string where you want to start the search.



5 Position the cursor where you want to type the code that uses the result of the `index` function and type the code.

6 To begin searching for a substring from the end of a string, perform steps 1 to 5, except type `= rindex()`; in step 1.



7 Save and execute the script.

Perl generates the result of finding the location of a substring in a string.

USING MATHEMATICAL FUNCTIONS

- rand** *NUMBER* – Generate a random number.
- int** *NUMBER* – Convert a non-integer to an integer.
- oct** *NUMBER* – Convert an octal value to a decimal value.

Perl includes several built-in functions that you can use in your scripts to perform mathematical calculations.

The **rand** function generates a random fractional number between 0 and a number you specify. The number you specify should be a positive number. If you do not specify a number, the **rand** function will generate a random number between 0 and 1.

The **int** function allows you to convert a non-integer value to an integer. Keep in mind that the **int** function only removes numbers following the decimal point and does not round the number to the nearest integer

- hex** *NUMBER* – Convert a hexadecimal value to a decimal value.
- abs** *NUMBER* – Convert a negative number to an absolute value.

value. The **int** function is often used with the **rand** function. For example, you can use the **rand** function to generate a random fractional number and then use the **int** function to convert the number to an integer.

The **oct** or **hex** function converts an octal or hexadecimal value to its equivalent decimal value. The **abs** function converts a negative number to a positive, or absolute, value.

If you do not specify a value for the **int**, **oct**, **hex** or **abs** functions, Perl will use the value of the default special variable, `$_`.

Extra

When working with numbers, Perl assumes you are working in base 10, or decimal notation. Perl also supports base 8, octal notation, and base 16, hexadecimal notation. To work in octal notation, precede each octal number by a zero (0). To work in hexadecimal notation, precede each hexadecimal number by a zero (0) and the letter x.

Example:

Decimal	Octal	Hexadecimal
16	020	0x10
100	0144	0x64
255	0377	0xFF
2000	03720	0x7D0

You can use the **srand** function to ensure that the **rand** function generates a different random number each time the script is run. The **rand** function will generate random numbers based on the value you specify in the **srand** function. The value of many special variables cannot be predicted, so you should specify a special variable, such as `$$`, in the **srand** function.

TYPE THIS:

```
srand($$);
@suits = ("clubs", "diamonds", "hearts", "spades");
@cards = ("ace", 2..10, "jack", "queen", "king");
for (1..6)
{
    print "$cards[int(rand 13)] $suits[int(rand 4)] ";
}
```

RESULT:

king diamonds 9 clubs 5 diamonds 7 hearts
7 clubs 5 spades

USING MATHEMATICAL FUNCTIONS

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl
print rand 10, "\n";
print int 3.567, "\n";
```

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl
print rand 10, "\n";
print int 3.567, "\n";
print oct 45, "\n";
print hex FF, "\n";
```

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl
print rand 10, "\n";
print int 3.567, "\n";
print oct 45, "\n";
print hex FF, "\n";
print abs -45, "\n";
```

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

[scripts]\$ script.pl

2.60769712273031
3
37
255
45

[scripts]\$

GENERATE A RANDOM NUMBER

- 1 Type **rand** followed by the number you want to use to generate a random number.
- 2 Position the cursor where you want to type the code that uses the random number and type the code.

CONVERT TO AN INTEGER

- 3 Type **int** followed by the non-integer you want to convert to an integer.
- 4 Position the cursor where you want to type the integer and type the code.

CONVERT TO A DECIMAL VALUE

- 5 Type **oct** followed by the octal number you want to convert to a decimal value.
- 6 Position the cursor where you want to type the code that uses the decimal value and type the code.
- To convert a hexadecimal number to a decimal value, perform steps 5 and 6 except type **hex** in step 5.

CONVERT TO AN ABSOLUTE VALUE

- 7 Type **abs** followed by the negative number you want to convert to an absolute value.

- 8 Position the cursor where you want to type the absolute value and type the code.

- 9 Save and execute the script.

Perl generates the results of using mathematical functions.

WORK WITH UNDEFINED VARIABLES

undef VARIABLE – Undefine a variable.
defined VARIABLE – Determine if a variable is defined.

Using the undef function allows you to undefine a scalar, hash or array variable. When a variable is undefined, the variable still exists, but it will have a value of zero if it is used in a numeric calculation. If the variable is used in a string, it will have a value of a zero-length string ("").

Undefining a variable is useful for making a variable used in a block of code available to be used again in a later part of the script. Keep in mind that undefining a variable will not free up space in the computer's memory on most UNIX systems.

You should not use the undef function when using special variables, since this can produce unexpected results.

The defined function allows you to determine if a variable is defined. If the variable stores a defined value, the defined function will return a value of 1. If the variable stores an undefined value, the defined function will return a value of 0.

Determining if a variable is defined is useful when you want to check if a variable currently stores a value before overwriting the variable with a new value.

Apply It

The undef function can be used as a placeholder when assigning a list to another list that contains variables. This is useful when a list contains values that the script will not need to use again, since Perl will discard the value assigned to the undef function. In this example, the second and forth names are undefined, so the names in the second and forth positions of the @studentNames array are discarded.

TYPE THIS:

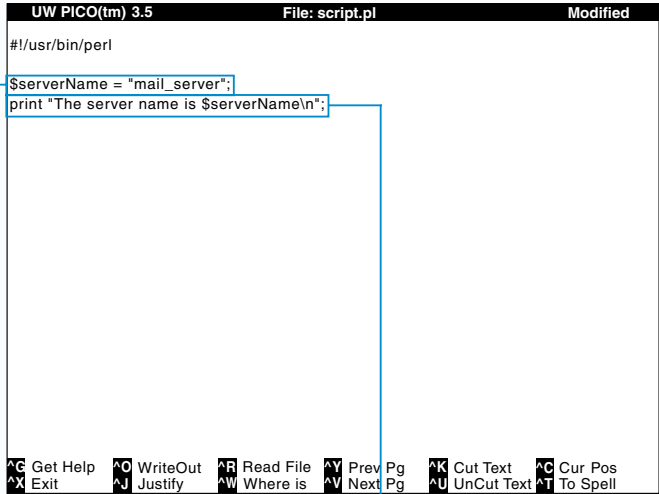
```
@studentNames = (Barry, April, Ted, Lindsay, Craig, Mark, Kyle);
$, = "\n";
print Discard2and4 (@studentNames);

sub Discard2and4
{
    ($first, undef, $third, undef, @theRest) = @_;
    return $first, $third, @theRest;
}
```

RESULT:

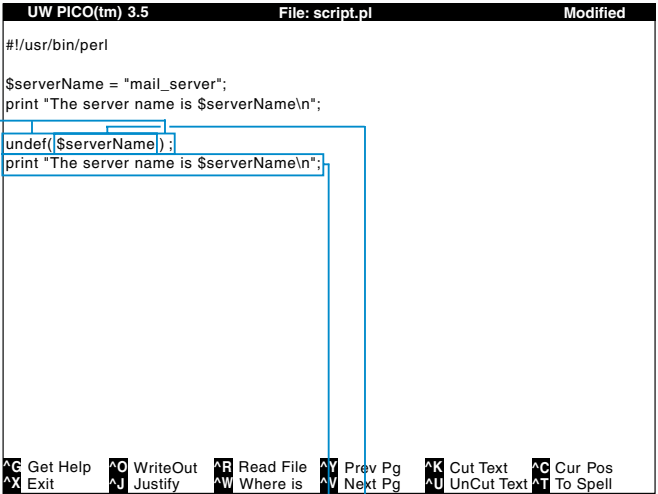
```
Barry
Ted
Craig
Mark
Kyle
```

WORK WITH UNDEFINED VARIABLES



1 Type the code that creates a variable and assigns it a value.

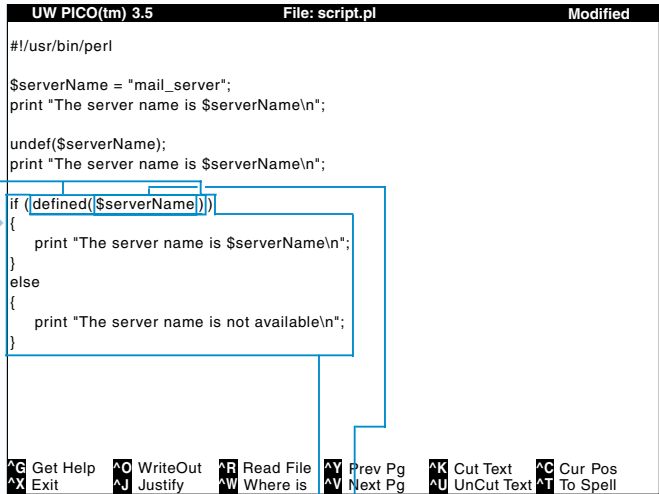
2 Position the cursor where you want to type the code that uses the variable and type the code.



3 To undefine a variable, type **undef();**.

4 Position the cursor over the closing parenthesis and type the name of the variable you want to undefine.

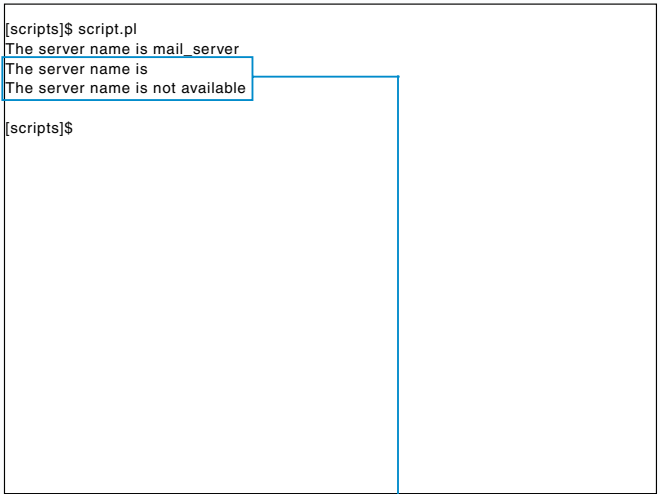
5 Position the cursor where you want to type the code that uses the undefined variable and type the code.



6 To determine if a variable is defined, type **defined()**.

7 Position the cursor over the closing parenthesis and type the name of the variable you want to check.

8 Position the cursor where you want to type the code that uses the defined function and type the code.



9 Save and execute the script.

Perl generates the results of using the undef and defined functions.

WORK WITH TIME

time – Determine the number of seconds since the beginning of the epoch.
localtime *EXPRESSION* – Determine the current time.
times – Determine the running time for a script.

Using the time function with the localtime function allows you to access the current time from a computer's clock.

The time function returns the value of the number of non-leap seconds since the beginning of the epoch, which dates from January 1, 1970 for most operating systems. While this value is not very useful on its own, it is necessary for other functions that access time, such as the localtime function.

The localtime function uses the value returned by the time function to determine the current time.

The localtime function returns a list of nine numerical values that each represent one part of the time and date. The values are always returned in the same order—current second, minute, hour, day of the month, month, year, day of the week and day of the year. The final value in the list is a 1 or a 0, indicating whether daylight savings time is in effect or not.

The numerical values returned by the localtime function start at 0. For example, months are numbered from 0 to 11, so the month of July is returned as the value 6.

Extra

You can find the running time of a script in seconds. This lets you test the efficiency of a script and can help you determine whether you need to improve your code. The times function returns a list of four values that represent the running time of a script. The first value is the amount of time a script runs as observed by the user, while the second value is the amount of time that a script uses the computer's CPU. The third and fourth values represent the amount of time that any other processes started by the script have been running for the user and computer, respectively.

TYPE THIS:

```
 srand;
foreach (1 .. 300)
{
    $passWord = "";
    $passWord .= (a .. z)[int(rand 26)] foreach (1 .. 8);
    system "echo $passWord >> passfile.dat";
}

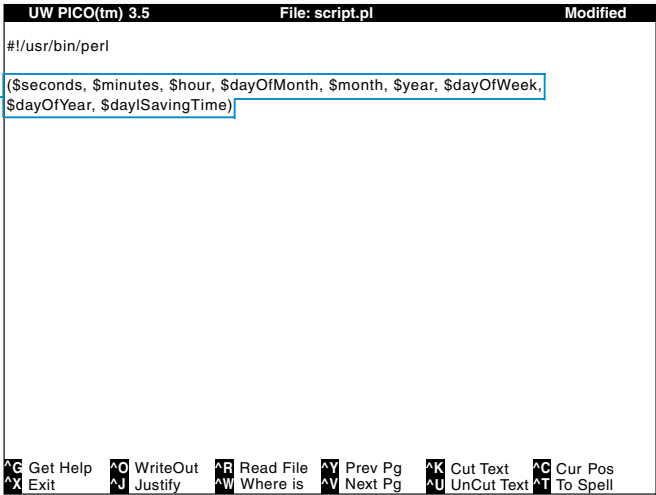
($usertime, $systemtime, $otheruser, $othersystem) = times();

print("User time: ", $usertime, "\n",
      "System time: ", $systemtime, "\n",
      "Other Process time (user): ", $otheruser, "\n",
      "Other Process time (system): ", $othersystem, "\n");
```

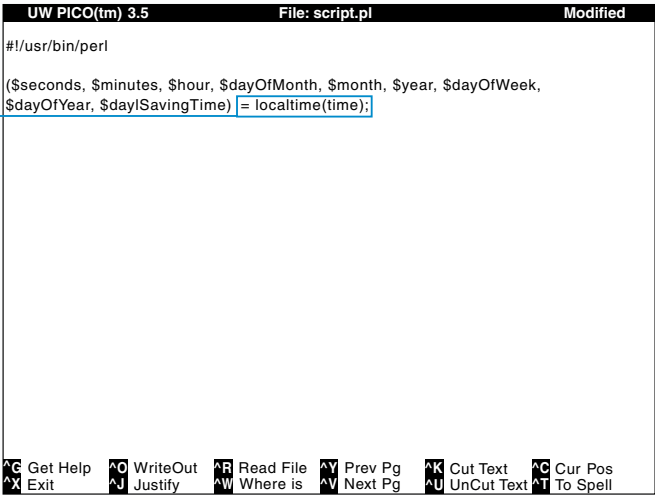
RESULT:

```
User time: 0.13
System time: 0.02
Other Process time (user): 1.24
Other Process time (system): 0.95
```

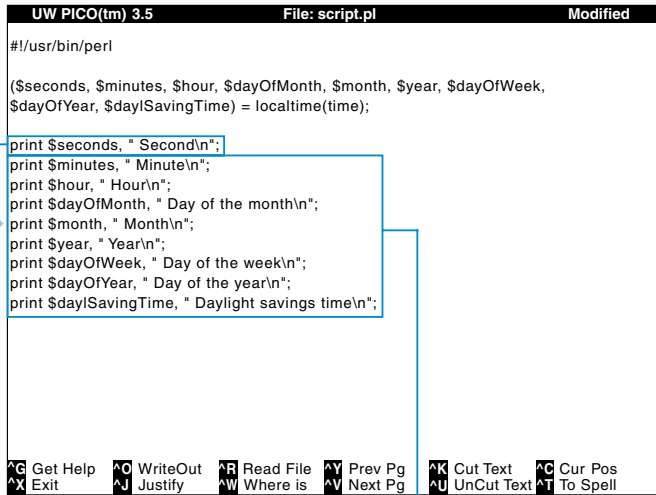
WORK WITH TIME



1 Type the list of nine variables you want to use to represent each part of the current time.

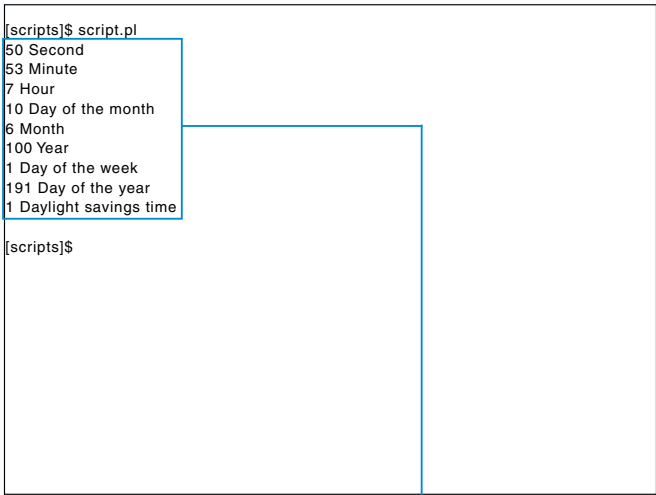


2 To determine the current time, type = localtime(time); and then press Enter.



3 Position the cursor where you want to type the code that uses a variable representing the part of the time you want to access and type the code.

4 Repeat step 4 for each part of the time you want to access.



5 Save and execute the script.

Perl generates the results of using the localtime function.

USING ASCII VALUES

chr *NUMBER* – Determine the character for an ASCII value.
ord *EXPRESSION* – Determine the ASCII value of a character.

ASCII (American Standard Code for Information Interchange) is a numbering system that assigns numerical values to letters, numbers and other characters. Many programs save, read and exchange data using the ASCII numbering system. Using the **chr** and **ord** functions allows you to convert between ASCII values and characters.

The **chr** function converts a specified ASCII value to a character. Perl allows you to manipulate the results of the **chr** function as you would manipulate strings. For example, you can join characters returned by the **chr** function together using the *concatenation operator* (**.**).

The **ord** function converts a specified character to an ASCII value. If a string of multiple characters is

specified, the **ord** function will return only the ASCII value of the first character.

All characters on a computer, including non-displayable characters such as tab and newline, have an ASCII value. To convert a non-displayable character to an ASCII value, you must provide the backslash escape sequence for the character. For a list of commonly used backslash escape sequences, see the top of page 19.

If you do not specify an ASCII value or character for the **chr** or **ord** function, the function will use the value of the default special variable, **\$_**.

USING ASCII VALUES

UW PICO(tm) 3.5File: script.plModified

#!/usr/bin/perl

print chr(37);

print "n";

print ord "H";

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

[scripts]\$ script.pl

%72

[scripts]\$

- 1

To convert an ASCII value to a character, type **chr** followed by the ASCII value you want to convert, enclosed in parentheses.
- 2

To convert a character to an ASCII value, type **ord** followed by the character you want to convert, enclosed in quotation marks.
- 3

Type the code that generates the results of using the **chr** and **ord** functions.
- 4

Save and execute the script.
- Perl generates the results of using the **chr** and **ord** functions.

SUMMARY OF ASCII VALUES

ASCII VALUE:	CHARACTER:	ASCII VALUE:	CHARACTER:	ASCII VALUE:	CHARACTER:
0	null	43	+	86	V
1	start of heading	44	,	87	W
2	start of text	45	-	88	X
3	end of text	46	.	89	Y
4	end of transmission	47	/	90	Z
5	enquiry	48	0	91	[
6	acknowledge	49	1	92	\
7	bell	50	2	93]
8	backspace	51	3	94	^
9	horizontal tab	52	4	95	_
10	newline	53	5	96	`
11	vertical tab	54	6	97	a
12	new page	55	7	98	b
13	carriage return	56	8	99	c
14	shift out	57	9	100	d
15	shift in	58	:	101	e
16	data link escape	59	;	102	f
17	device control 1	60	<	103	g
18	device control 2	61	=	104	h
19	device control 3	62	>	105	i
20	device control 4	63	?	106	j
21	negative acknowledge	64	@	107	k
22	synchronous idle	65	A	108	l
23	end of transmission block	66	B	109	m
24	cancel	67	C	110	n
25	end of medium	68	D	111	o
26	substitute	69	E	112	p
27	escape	70	F	113	q
28	file separator	71	G	114	r
29	group separator	72	H	115	s
30	record separator	73	I	116	t
31	unit separator	74	J	117	u
32	space	75	K	118	v
33	!	76	L	119	w
34	"	77	M	120	x
35	#	78	N	121	y
36	\$	79	O	122	z
37	%	80	P	123	{
38	&	81	Q	124	
39	'	82	R	125	}
40	(83	S	126	~
41)	84	T	127	delete
42	*	85	U		

EVALUATE CODE

`eval` *STRING* – Evaluate a string.
`eval` { *BLOCK* } – Evaluate a block of code.

The `eval` function allows you to evaluate a string or block of code as a separate Perl script. This is useful for determining if your code contains an error. If the code contains an error, the `eval` function will fail, but the rest of the script will continue to run.

The `eval` function is often used to evaluate strings stored in variables. To evaluate a block of code, you must enclose the code in braces { }. If you do not specify a string or block in the `eval` function, the function will evaluate the contents of the default special variable, `$_`.

Variables created in the main script can be accessed by the string or block being evaluated. Variables created in the `eval` function can be accessed outside of the `eval` function, unless the variables were created using the `my` or `local` functions. For information about the `my` and `local` functions, see page 110.

Although not required by Perl, you can use the `return` statement to specify the value you want a block of code to return. If you do not use the `return` statement, Perl will return the value of the last statement evaluated in the block.

Apply It

The `eval` function can handle errors that would ordinarily stop the execution of a script. When a string or block generates an error, the `eval` function returns an undefined value and stores the error in the `$@` special variable, while continuing to process the script. You can access the information in the `$@` special variable to locate the error in your script.

TYPE THIS:

```
$numbers = "print foreach (1 .. 5);";  
eval $numbers; print "$@\n";  
  
$numbers = "open(MYDATA, \"mydatafile.dat\") or die $!;";  
eval $numbers; print "$@\n";  
print "Program is still running";
```

RESULT:

```
12345  
Died at (eval 2) line 1.  
  
Program is still running
```

EVALUATE CODE

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl  
  
$function = 'print';  
$message = "This is a test of the eval function.\n\n";  
$argument = ("{$message}");  
  
$statement = $function . $argument;  
eval $statement;
```

1

Type the code that creates the variables you want to use in an evaluation and assigns their values.

2

Position the cursor where you want to evaluate a string and type `eval` ;.

3

Position the cursor over the semi-colon and type the name of the variable that stores the string you want to evaluate.

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl  
  
$function = 'print';  
$message = "This is a test of the eval function.\n\n";  
$argument = ("{$message}");  
  
$statement = $function . $argument;  
eval $statement;  
  
$numNeeded = eval  
{  
    $clientNum = 2578;  
    $flyerNum = 2467;  
    ($clientNum - $flyerNum);  
};
```

4

Type the name of the variable you want to store the result of the evaluation followed by `= eval` and then press Enter.

5

Type { and then press Enter.

6

Press Tab and type the code you want to evaluate.

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl  
  
$function = 'print';  
$message = "This is a test of the eval function.\n\n";  
$argument = ("{$message}");  
  
$statement = $function . $argument;  
eval $statement;  
  
$numNeeded = eval  
{  
    $clientNum = 2578;  
    $flyerNum = 2467;  
    return($clientNum - $flyerNum);  
};  
print "Number of flyers needed: $numNeeded\n";
```

7

Position the cursor in front of the statement whose result you want to return and type `return`.

8

Position the cursor on the line directly below the last line of code in the block you want to evaluate and type `};`.

9

Position the cursor where you want to type the code that uses the result of the `eval` function and type the code.

[scripts]\$ script.pl

This is a test of the eval function.

Number of flyers needed: 111

[scripts]\$

10

Save and execute the script.

11

Perl generates the result of evaluating a string and a block of code.