

USING A SUBROUTINE

A subroutine contains a block of code that performs a specific task, such as displaying a message.

Using subroutines makes it easy to re-use sections of code. For example, you may have a subroutine that displays a warning message when a client enters invalid data into a form. Instead of retyping the section of code that displays the message for each field in the form, you can simply re-use the subroutine.

Subroutines allow you to group lines of code into smaller, more manageable sections. This makes it easier to understand and troubleshoot the code.

A subroutine is defined by using the sub keyword followed by the name of the subroutine. The block of code used in the subroutine is then enclosed in braces {}.

The block of code specified in a subroutine will not be executed until the subroutine is called in the script. To call a subroutine, you type the name of the subroutine followed by parentheses () where you want to execute the code specified in the subroutine.

Extra

You can call a subroutine from within another subroutine. When calling other subroutines from within a subroutine, be careful not to generate a loop that causes the subroutines to continuously call each other.

TYPE THIS:

```
$clientRecord = "jones0251432";
$invNumLen = 7;

print PutNumFirst($clientRecord);

sub PutNumFirst
{
    $capRecord = InitCap($_[0]);
    $newRecord = substr($capRecord,
        -$invNumLen) . substr($capRecord, 0, -$invNumLen);
    return $newRecord;
}

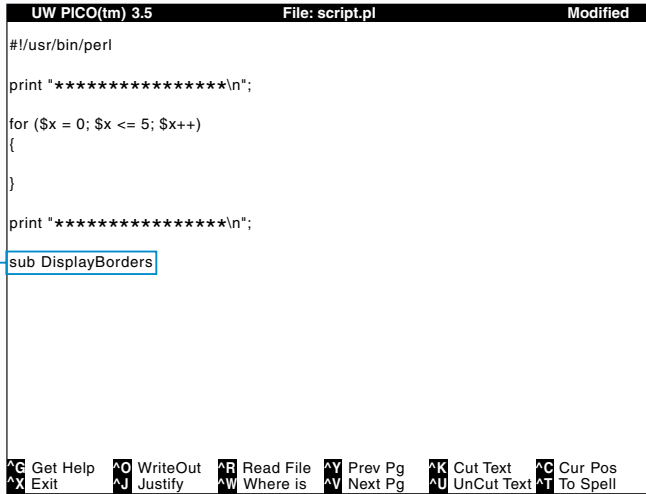
sub InitCap
{
    return "\u$_[0]";
}
```

RESULT:

0251432Jones

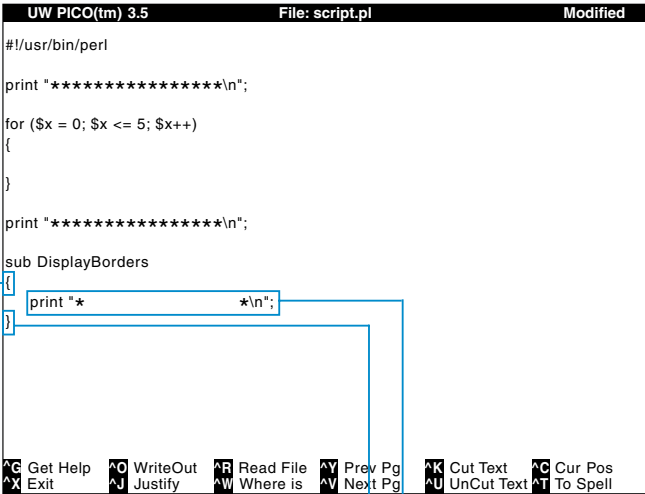
When naming a subroutine, you should combine multiple words and capitalize the initial letter of each word. This will help you distinguish subroutines from variables, statements, built-in functions and subroutines that Perl calls automatically, such as AUTOLOAD. The subroutine name should explain the purpose of the subroutine or describe the action it performs. For example, a subroutine that reads a file containing client records could be called ReadClientRecords.

USING A SUBROUTINE



DEFINE A SUBROUTINE

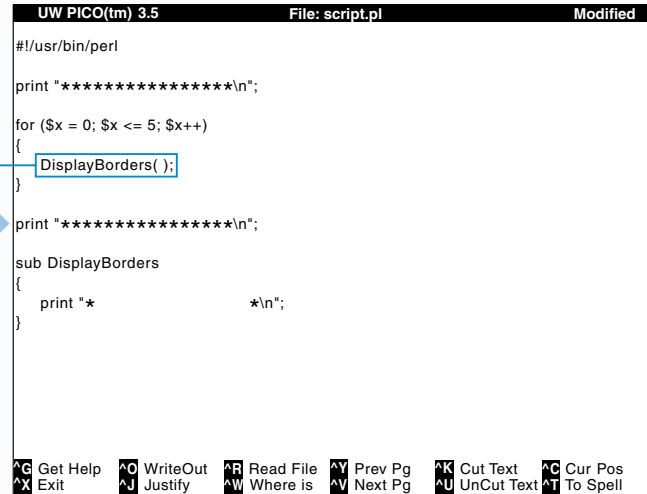
1 Type **sub** followed by the name of the subroutine you want to create. Then press Enter.



2 Type { and then press Enter.

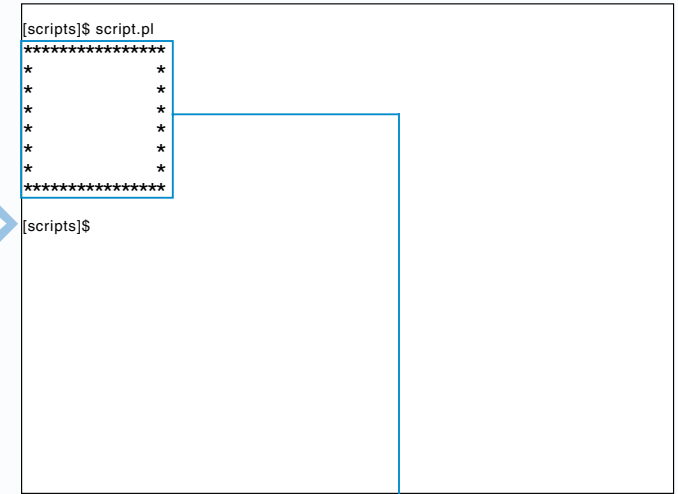
3 Press Tab and then type the block of code you want the subroutine to execute. Then press Enter.

4 Type } and then press Enter.



CALL A SUBROUTINE

5 Position the cursor where you want to call the subroutine and type the name of the subroutine followed by ();. Then press Enter.



6 Save and execute the script.

Perl generates the result of using a subroutine.

PASS ARGUMENTS TO A SUBROUTINE

Passing arguments to a subroutine you have created can make the subroutine more powerful. You can pass one or more arguments to a subroutine. For information about using subroutines, see page 106.

You include the arguments you want to pass to a subroutine in the code that calls the subroutine. The arguments are enclosed in the parentheses at the end of the subroutine name. When the subroutine is called, Perl uses the arguments to execute the code in the subroutine.

Perl automatically stores the arguments you pass to a subroutine in an array named `@_`. To access

an argument passed to a subroutine, you use `$_[0]` followed by the index of the argument enclosed in brackets. For example, `$_[0]` can be used to access the first argument passed to a subroutine.

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

Extra

The binary assignment operator (`||=`) allows you to assign a default value to an argument in a subroutine. This is useful when an argument is not passed to the subroutine. You should not assign a default value to an argument if the value of the argument can be zero or an empty string (`""`), since Perl will replace these values with the default value.

TYPE THIS:

```
Add(2);
sub Add
{
    $value1 = $_[0];
    $value2 = $_[1];
    $value2 ||= 10;
    print "$value1 + $value2 = ", ($value1 + $value2);
}
```

RESULT:

2 + 10 = 12

Perl uses subroutine prototypes to help you verify that the right types of arguments are passed to a subroutine. To use a subroutine prototype, list the character(s) the arguments must start with, such as `$`, `@` and `%`, in the `sub` statement. If an argument that does not start with a specified character is passed to the subroutine, an error message is generated.

TYPE THIS:

```
sub MergeIntoList($@);
$, = " , ";
print MergeIntoList " leaf", qw(orange red brown);
sub MergeIntoList($@)
{
    ($toMerge, @orig) = @_; map $_ .= $toMerge, @orig;
}
```

RESULT:

orange leaf, red leaf, brown leaf

PASS ARGUMENTS TO A SUBROUTINE

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl

print DisplayMessage("Please Wait..", 4);

sub DisplayMessage
{
    for ($x = 0; $x <= ; $x++)
    {
        $temp = $temp . "\n";
    }
}
```

1 Create and call the subroutine you want to pass arguments to.

2 In the code that calls the subroutine, position the cursor over the closing parenthesis and type each argument you want to pass to the subroutine, separated by a comma.

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl

print DisplayMessage("Please Wait..", 4);

sub DisplayMessage
{
    for ($x = 0; $x <= $_[0]; $x++)
    {
        $temp = $temp . $_[0] . "\n";
    }
}
```

3 Position the cursor where you want to access an argument in the subroutine and type `$_[0]`.

4 Position the cursor over the closing bracket and type the index of the argument you want to access.

5 Repeat steps 3 and 4 for each argument you want to access.

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl

print DisplayMessage("Please Wait..", 4);

sub DisplayMessage
{
    for ($x = 0; $x <= $_[1]; $x++)
    {
        $temp = $temp . $_[0] . "\n";
    }
    return $temp;
}
```

6 Position the cursor where you want to return a result from the subroutine and type `return` followed by the expression whose result you want to return.

```
[scripts]$ script.pl
Please Wait..
Please Wait..
Please Wait..
Please Wait..
Please Wait..
[scripts]$
```

7 Save and execute the script.

Perl generates the result of passing arguments to a subroutine and returning a result.

SET THE VARIABLE SCOPE

my VARIABLE – Set a private scope for a variable.
local VARIABLE – Set a private scope for a variable.

The scope of a variable determines which parts of a script can access the value stored in the variable.

By default, all variables have a global scope, which means variables can be accessed throughout the script. A variable with a private scope is a variable that can only be accessed within a specific block of code, such as a subroutine. Using a private scope helps ensure the script uses the correct value for a variable that appears more than once in the script.

Use the `my` or `local` function to set a private scope for a variable. The `my` function creates a

variable that can be accessed only within the block of code containing the `my` function. The `local` function is similar to the `my` function, but also makes the variable available to any subroutines accessed from the block of code containing the `local` function.

Using the `my` or `local` function with a variable will not affect any variables with the same name that appear in other parts of the script.

If you do not assign a value to a variable that uses the `my` or `local` function, the variable is automatically assigned a null value.

Extra

If you use a single statement to declare several variables that you want to restrict to the current subroutine, enclose the variables in parentheses before applying the `my` or `local` function. If you do not enclose the list of variables in parentheses, the `my` or `local` function will affect only the first variable in the list.

TYPE THIS:

```
$fileSize = 300;
$tempSpace = 120;
print "\$fileSize = $fileSize, \$tempSpace = $tempSpace\n";
ReadFile();
sub ReadFile
{
    my($fileSize, $tempSpace) = (900, 640);
    print "\$fileSize = $fileSize, \$tempSpace = $tempSpace\n";
}
```

RESULT:

```
$fileSize = 300, $tempSpace = 120
$fileSize = 900, $tempSpace = 640
```

You can enter the `use strict 'vars';` declaration below `#!/usr/bin/perl` in your script to ensure that all your variables are properly declared. This helps you avoid accidentally using an undefined variable in a script due to misspelling the variable name. When you use the declaration, you must use the `my` function with each variable you declare in the script. If the script attempts to access a variable that is not properly declared, an error message will be generated.

SET THE VARIABLE SCOPE

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl

$x = 10;
$y = 10;

DisplayValue();

sub DisplayValue
{
    my $x = 20;
    DisplayAnotherValue();
}

sub DisplayAnotherValue
{
}
```

USING THE MY FUNCTION

- 1 To create a variable that will only be used in the current block of code, type `my` followed by a name for the variable.
- 2 To assign a value to the variable, type `=` followed by the value you want to assign. Then type a semi-colon (;).

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl

$x = 10;
$y = 10;

DisplayValue();

sub DisplayValue
{
    my $x = 20;
    local $y = 20;
    DisplayAnotherValue();
}

sub DisplayAnotherValue
{
}
```

USING THE LOCAL FUNCTION

- 3 To create a variable that will only be used in the current block of code or any subroutine called in the block of code, type `local` followed by a name for the variable.
- 4 To assign a value to the variable, type `=` followed by the value you want to assign. Then type a semi-colon (;).

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl

$x = 10;
$y = 10;

DisplayValue();

sub DisplayValue
{
    my $x = 20;
    local $y = 20;
    print "\$x = $x, \$y = $y\n";
    DisplayAnotherValue();
}

sub DisplayAnotherValue
{
    print "\$x = $x, \$y = $y\n";
}
```

- 5 Position the cursor where you want to type the code that uses the variables and type the code.

```
[scripts]$ script.pl
$x = 20, $y = 20
$x = 10, $y = 20

[scripts]$
```

- 6 Save and execute the script.
- Perl generates the result of setting the variable scope.