

Perl: Subroutines

The HTML lecture notes have more details and examples

EIW - Perl Subroutines

1

Perl Subroutines and **sub**

- Use **sub** to create a new subroutine

```
sub subname {  
    statement1;  
    statement2;  
    # you can put more statements here  
}
```

subroutine name

subroutine body

EIW - Perl Subroutines

2

Arguments!

- You don't specify arguments (or even how many there are)!
- The values passed in all come packaged up in the array `@_;`
 - the subroutine gets its own `@_` array, this is not the same one available elsewhere in the Perl program.

EIW - Perl Subroutines

3

Return Value

- A Perl subroutine can return a single *thing*.
- The *thing* can be a scalar, a list (array) or a hash.
- There is a **return** keyword, but you don't have to use it!
 - The value of the last expression evaluated in the subroutine is returned.

EIW - Perl Subroutines

4

First Subroutine

```
sub foo {  
    "Hi Dave";  
}
```

Pretty useless, but a valid subroutine.

No return statement, but the return value is "Hi Dave"

EIW - Perl Subroutines

5

Calling a Perl subroutine

- In the old days, the name of a subroutine started with '&' (doesn't have to any more).
- These are all valid calls of the subroutine:

```
&foo;          foo(1,2,3,4);  
&foo("Hello"); foo 11.75;
```

EIW - Perl Subroutines

6

Our first subroutine `foo` is useless

```
$x = foo(3.141593,"Pi");
```

- Our subroutine doesn't do anything with the values passed in.
 - it's not an error to pass some stuff anyway!

EIW - Perl Subroutines

7

Global Variables

- Any variable that is created outside of a subroutine is global.
- Perl subroutines can access/change global variables.

NOTE: The latest versions of Perl want you to *declare* all variables explicitly, but unless you set the "strict" option you don't have to.

EIW - Perl Subroutines

8

A real subroutine

```
# here we define the subroutine printcelsius
sub printcelsius {
    $degc = ($degf - 32) * (5/9);
    print "$degf degrees fahrenheit" .
        " is $degc degrees celsius\n";
}
```

EIW - Perl Subroutines

9

```
# perl program to calculate degrees celsius
# using a subroutine.
# ask the user for a temp in degrees fahr.
print "Enter degrees fahrenheit:\n";

# read in the value from stdin
# (and chop off the newline)
chop($degf = <STDIN>);

# call the subroutine printcelsius
printcelsius();
```

EIW - Perl Subroutines

10

Subroutine Location

- Subroutine definitions can be placed anywhere in the file.
- Typically you group them all together in one place.
 - This way the "main program" is also in one place".
 - This is nice for humans, Perl doesn't care!

EIW - Perl Subroutines

11

Getting at parameter values

When a subroutine is called, Perl collects the parameter values and puts them in to a special version of `@_` that is usable only by the subroutine.

```
foo(1, "Hi");
```

inside `foo`,

```
$_[0] is 1 and $_[1] is "Hi"
```

EIW - Perl Subroutines

12

`@_, $_[0], $_[1]`

Help Meeeee.....

- Typically you assign the parameter values to private variable right away, so you don't need to mess with @_.

```
($a,$b) = @_;  
# now $a is the first value,  
# $b is the second
```

EIW - Perl Subroutines

13

Private Variables

- You can use the my keyword to declare a variable as private to the subroutine:

```
my($a,$b);  
my($size) = 11;  
my(@foo) = (1,3..11);
```

EIW - Perl Subroutines

14

Typical Subroutine

```
sub min {  
    my($a,$b) = @_;  
    if ($a < $b) {  
        $a;  
    } else {  
        $b;  
    }  
}
```

EIW - Perl Subroutines

15

Another Version of `min`

```
sub min {  
    my($a,$b) = @_;  
    if ($a < $b) {  
        return $a;  
    } else {  
        return($b);  
    }  
}
```

EIW - Perl Subroutines

16

perlish versions of `min`

```
sub min {  
    ($_[0]<$_[1])?$_[0]:$_[1];  
}  
  
sub min {  
    (sort {$a<=>$b;} @_ )[0];  
}
```

EIW - Perl Subroutines

17

What about passing an array?

- You can pass an entire array to a subroutine:
- `foo(@somevals);`
- Or a few scalars followed by an array:
- `foo($a,$b,@somevals);`
- Everything is put in the `@_` array that the subroutine gets...

EIW - Perl Subroutines

18

Exercise

- Write a subroutine that prints out an HTML table (one row) with the values passed in as the contents of the table cells.

```
maketable("Hello",12,"last cell");
```

EIW - Perl Subroutines

19

One Solution

```
sub maketable {  
    local(@vals) = @_;  
    my($i);  
  
    print "<TABLE><TR>\n";  
    for ($i=0;$i<=$#vals;$i++) {  
        print "<TD>$vals[$i]</TD>\n";  
    }  
    print "</TR></TABLE>\n";  
}
```

EIW - Perl Subroutines

20

A *perlsh* solution

```
sub maketable {  
    print "<TABLE><TR>\n<TD>" .  
        join("</TD>\n<TD>",<_> .  
        "</TD>\n</TR></TABLE>";  
}
```

EIW - Perl Subroutines

21

A First Perl CGI

- Write a Perl *program* that prints out the line:
Content-type: text/html
- then prints out a blank line.
- then calls
`maketable("This", "is", "a", "table");`

EIW - Perl Subroutines

22

Running the CGI

- The file containing the Perl program must have the definition for the subroutine **maketable**.
- The file must be named something **.pl**
- The file must be in a "virtual folder" that your web server can access (the same place you put **file.pl** will work!)

EIW - Perl Subroutines

23

Make sure you can do this!

- Next week in lab you will need to be able to write simple Perl CGI programs!
- Will involve arrays and hashes (assoc. arrays).
- Programs that contain simple subroutines.

EIW - Perl Subroutines

24
