

Perl II

Logic and control structures

Sofia Robb

What is truth?

`0` the number 0 is false

`"0"` the string 0 is false

`""` and `' '` an empty string is false

`my $x;` an undefined variable is false

everything else is **true**

Examples of truth

```
my $a;           # FALSE (not yet defined)
$x = 1;          # TRUE
$x = 0;          # FALSE
$x = "\n";       # TRUE (a single newline is non-empty)
$x = 'true';     # TRUE
$x = 'false';    # TRUE (!!!! "false" is a nonempty string)
$x = ' ';        # TRUE (a single space is non-empty)
$x = 0.0;        # FALSE (converts to string "0")
$x = '0.0';     # TRUE (watch out! The string "0.0" is not the
                # same as "0")
```

defined

`defined` lets you test whether a variable is defined.

```
if ( defined($x) ) {  
    print "$x is defined\n";  
}
```

Control structures

Control structures allow you to control if and how a line of code is executed.

You can create alternative branches in which different sets of statements are executed depending on the circumstances.

You can create various types of repetitive loops.

Control structures

So far you've seen a basic program, where every line is executed, in order, and only once.

```
my $x = 1;  
my $y = 2;  
my $z = $x + $y;  
print "$x + $y = $z\n";
```

Control structures

Here, the print statement is only executed some of the time.

```
my $x = 1;  
my $y = 2;  
if ($x == $y) {  
    print "$x and $y are equal\n";  
}
```

Components of a control structure

1. a keyword

2. a statement in parentheses

3. squiggly brackets

```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}
```

The part enclosed by the squiggly brackets is called a block.

Components of a control structure

When you program, build the structure first and then fill in.

1. a keyword

2. a statement in parentheses

3. squiggly brackets

```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}
```

4. now add the print statement

if

```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}
```

If `$x` is the same as `$y`, then the print statement will be executed.

or said another way:

If `($x == $y)` is true, then the print statement will be executed.

if — a common mistake

What happens if we write it this way?

```
if ($x = $y) {  
    print "$x and $y are equal\n";  
}
```

if — a common mistake

- 1 equals sign to make the left side equal the right side.
- 2 equals signs to test if the left side is equal to the right.

```
my $x;           # x is undefined
my $x = 1;       # x is now defined
if ($x == 1)     # is $x equal to 1?
if ($x = 1)      # (wrong)
```

`use warnings` will catch this error.

else

If the if statement is false, then the first print statement will be skipped and only the second print statement will be executed.

```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}  
else {  
    print "$x and $y aren't equal\n";  
}
```

elseif

Sometimes you want to test a series of conditions.

```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}  
elseif ($x > $y) {  
    print "$x is bigger than $y\n";  
}  
elseif ($x < $y) {  
    print "$x is smaller than $y\n";  
}
```

elseif

What if more than one condition is true?

```
if (1 == 1) {  
    print "$x and $y are equal\n";  
}  
elseif (2 > 0) {  
    print "2 is positive\n";  
}  
elseif (2 < 10) {  
    print "2 is smaller than 10\n";  
}
```

Logical operators

Use **and** and **or** to combine comparisons.

Operator

Meaning

and

TRUE if left side is TRUE and right side is TRUE

or

TRUE if left side is TRUE or right side is TRUE

Logical operator examples

```
if ($i < 100 and $i > 0) {  
    print "$i is the right size\n";  
}  
else {  
    print "out of bounds error!\n";  
}
```

```
if ($age < 10 or $age > 65) {  
    print "Your movie ticket is half price!\n";  
}
```

while

As long as `($x == $y)` is true, the print statement will be executed over and over again.

```
while ( $x == $y ) {  
    print "$x and $y are equal\n";  
}
```