The Basics

Getting started with Perl

Let's Get Started!

```
print "Welcome to the Wonderful World of Bioinformatics!\n";
```

Another version of welcome

```
print "Welcome ";
print "to ";
print "the ";
print "Wonderful ";
print "World ";
print "of ";
print "Bioinformatics!";
print "\n";
```

Maxim 3.1

Programs execute in sequential order

Maxim 3.2

Less is better

Maxim 3.3

If you can say something with fewer words, then do so

Running Perl programs

```
$ perl -c welcome

welcome syntax OK


$ perl welcome
```

Running Perl – Syntax Errors

```
String found where operator expected at welcome line 3,
near "pint "Welcome to the Wonderful World of
  Bioinformatics!\n""
(Do you need to predeclare pint?)
syntax error at welcome line 3,
near "pint "Welcome to the Wonderful World of
  Bioinformatics!\n""
welcome had compilation errors.
```

Syntax and semantics

```
print ; "Welcome to the Wonderful World of Bioinformatics!\n";
```

```
$ perl whoops

$ perl -c -w whoops

Useless use of a constant in void context at whoops line 1.
whoops syntax OK
```

Program: run thyself!

```
$ chmod u+x welcome3

#! /usr/bin/perl -w

$ ./welcome3

$ perl welcome3
```

Maxim 3.4

There's more than one way to do it

Iteration

Using the Perl while construct

```
#! /usr/bin/perl -w

# The 'forever' program - a (Perl) program,
# which does not stop until someone presses Ctrl-C.

use constant TRUE => 1;
use constant FALSE => 0;

while ( TRUE )
{
    print "Welcome to the Wonderful World of Bioinformatics!\n";
    sleep 1;
}
```

Running forever ...

```
$ chmod u+x forever
$ ./forever

Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
.
.
```

Maxim 3.5

Add comments to make future maintenance of a program easier for other programmers and for you

Maxim 3.6

When using constant values, refer to them with a nice, human-friendly name as opposed to the actual value

Maxim 3.7

Use blocks to group program statements together

More Iterations

Maxim 3.8

A condition can result in a value of true or false

Introducing variable containers

```
$name
$_address
$programming_101
$z
$abc
$count
```

Maxim 3.9

When you need to change the value of an item, use a variable container

Maxim 3.10

Don't be lazy: use good, descriptive names for variables

Variable containers and loops

```
#! /usr/bin/perl -w

# The 'tentimes' program - a (Perl) program,
# which stops after ten iterations.

use constant HOWMANY => 10;

$count = 0;

while ( $count < HOWMANY )
{
    print "Welcome to the Wonderful World of Bioinformatics!\n";
    $count++;
}
```

Running tentimes ...

```
$ chmod u+x tentimes
$ ./tentimes

Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
Welcome to the Wonderful World of Bioinformatics!
```

Selection

Using the Perl if construct

```
#! /usr/bin/perl -w

# The 'fivetimes' program - a (Perl) program,
# which stops after five iterations.

use constant TRUE => 1;
use constant FALSE => 0;

use constant HOWMANY => 5;

$count = 0;
while ( TRUE )
{
    $count++;
    print "Welcome to the Wonderful World of Bioinformatics!\n";
    if ( $count == HOWMANY )
    {
        last;
    }
}
```

There Really Is MTOWTDI

```
#! /usr/bin/perl -w

# The 'oddeven' program - a (Perl) program,
# which iterates four times, printing 'odd' when $count
# is an odd number, and 'even' when $count is an even
# number.

use constant HOWMANY => 4;

$count = 0;
while ( $count < HOWMANY )
{
    $count++;
    if ( $count == 1 )
    {
        print "odd\n";
    }
    elsif ( $count == 2 )
    {
        print "even\n";
    }
```

The oddeven program, cont.

```
    elsif ( $count == 3 )
    {
        print "odd\n";
    }
    else # at this point $count is four.
    {
        print "even\n";
    }
}
```

The terrible program

```
#! /usr/bin/perl -w
# The 'terrible' program - a poorly formatted 'oddeven'.
use constant HOWMANY => 4; $count = 0;
while ( $count < HOWMANY ) { $count++;
if ( $count == 1 ) { print "odd\n"; } elsif ( $count == 2 )
{ print "even\n"; } elsif ( $count == 3 ) { print "odd\n"; }
else # at this point $count is four.
{ print "even\n"; } }
```

Maxim 3.11

Use plenty of whitespace, blank-lines and indentation to make your programs easier to read

The oddeven2 program

```perl
#! /usr/bin/perl -w

# The 'oddeven2' program - another version of 'oddeven'.

use constant HOWMANY => 4;

$count = 0;

while ( $count < HOWMANY )
{
    $count++;
    if ( $count % 2 == 0 )
    {
        print "even\n";
    }
    else # $count % 2 is not zero.
    {
        print "odd\n";
    }
}
```

Using the modulus operator

```
print 5 % 2, "\n"; # prints a '1' on a line.
print 4 % 2, "\n"; # prints a '0' on a line.
print 7 % 4, "\n"; # prints a '3' on a line.
```

The oddeven3 program

```
#! /usr/bin/perl -w

# The 'oddeven3' program - yet another version of 'oddeven'.

use constant HOWMANY => 4;

$count = 0;

while ( $count < HOWMANY )
{
    $count++;
    print "even\n" if ( $count % 2 == 0 );
    print "odd\n" if ( $count % 2 != 0 );
}
```

Processing Data Files

```
<>;

$line = <>;
```

```
#! /usr/bin/perl -w

# The 'getlines' program which processes lines.

while ( $line = <> )
{
    print $line;
}
```

Running getlines ...

```
$ chmod u+x getlines

$ ./getlines
```

Asking getlines to do more

```
$ ./getlines terrible

$ ./getlines terrible welcome3
```

Introducing Patterns

```
#! /usr/bin/perl -w

# The 'patterns' program - introducing regular expressions.

while ( $line = <> )
{
    print $line if $line =~ /even/;
}
```

Running patterns ...

```
$ ./patterns terrible

# The 'terrible' program - a poorly formatted 'oddeven'.
{ print "even\n"; } elsif ( $count == 3 ) { print "odd\n"; }
{ print "even\n"; } }
```

```
$ ./patterns oddeven

# The 'oddeven' program - a (Perl) program,
# is an odd number, and 'even' when $count is an even
print "even\n";
print "even\n";
```

Maxim 3.12

Patterns tell perl what to look for, not how to find it

Where To From Here