

Introduction to Bioinformatics

Prof. Dr. Nizamettin AYDIN

naydin@yildiz.edu.tr

<http://www3.yildiz.edu.tr/~naydin>

Introduction to Bioinformatics

Introduction to Perl

Learning objectives

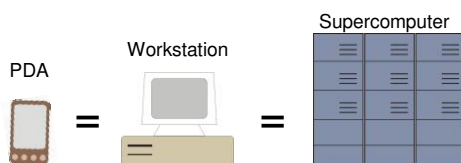
- After this lecture you should be able to understand :
 - sequence, iteration and selection;
 - basic building blocks of programming;
 - three C's: constants, comments and conditions;
 - use of variable containers;
 - use of some Perl operators and its pattern-matching technology;
 - Perl input/output
 - ...

Setting The Technological Scene

- One of the objectives of this course is..
 - to enable students to acquire an understanding of, and ability in, a programming language (Perl, Python) as the main enabler in the development of computer programs in the area of Bioinformatics.
- Modern computers are organised around two main components:
 - Hardware
 - Software

Introduction to the Computing

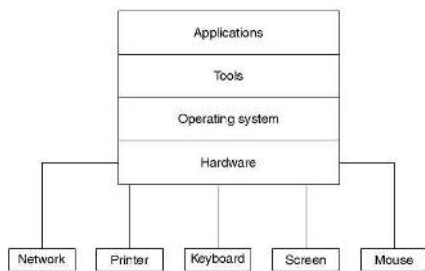
- Computer: electronic genius?
 - NO! Electronic idiot!
 - Does exactly what we tell it to, nothing more.
- All computers, given enough time and memory, are capable of computing exactly the same things.



Introduction to the Computing

- In theory, computer can **compute** anything
- that's possible to compute
 - given enough **memory** and **time**
- In practice, **solving problems** involves computing under constraints.
 - **time**
 - weather forecast, next frame of animation, ...
 - **cost**
 - cell phone, automotive engine controller, ...
 - **power**
 - cell phone, handheld video game, ...

Layers of Technology



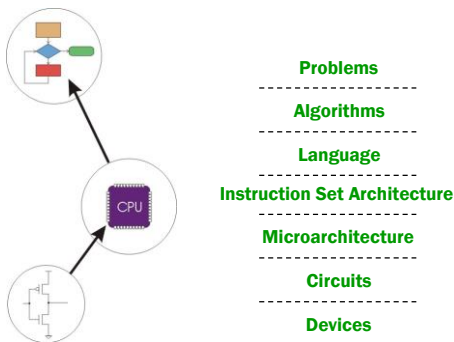
7

Layers of Technology

- Operating system...
 - Interacts directly with the hardware
 - Responsible for ensuring efficient use of hardware resources
- Tools...
 - Softwares that take advantage of what the operating system has to offer.
 - Programming languages, databases, editors, interface builders...
- Applications...
 - Most useful category of software
 - Web browsers, email clients, web servers, word processors, etc...

8

Transformations Between Layers



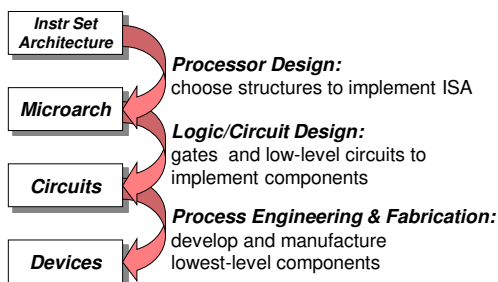
9

How do we solve a problem using a computer?

- A systematic sequence of transformations between layers of abstraction.
-
- Software Design:**
choose algorithms and data structures
- Programming:**
use language to express design
- Compiling/Interpreting:**
convert language to machine instructions

10

Deeper and Deeper...



11

Descriptions of Each Level...

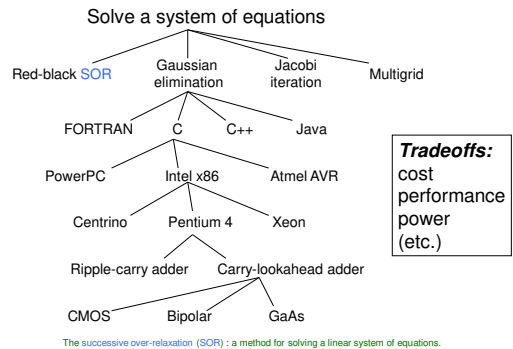
- Problem Statement
 - stated using "natural language"
 - may be ambiguous, imprecise
- Algorithm
 - step-by-step procedure, guaranteed to finish
 - definiteness, effective computability, finiteness
- Program
 - express the algorithm using a computer language
 - high-level language, low-level language
- Instruction Set Architecture (ISA)
 - specifies the set of instructions the computer can perform
 - data types, addressing mode

12

...Descriptions of Each Level

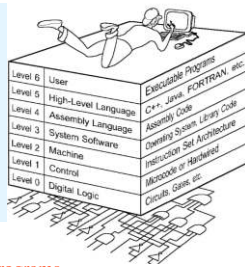
- Microarchitecture
 - detailed organization of a processor implementation
 - different implementations of a single ISA
- Logic Circuits
 - combine basic operations to realize microarchitecture
 - many different ways to implement a single function (e.g., addition)
- Devices
 - properties of materials, manufacturability

Many Choices at Each Level



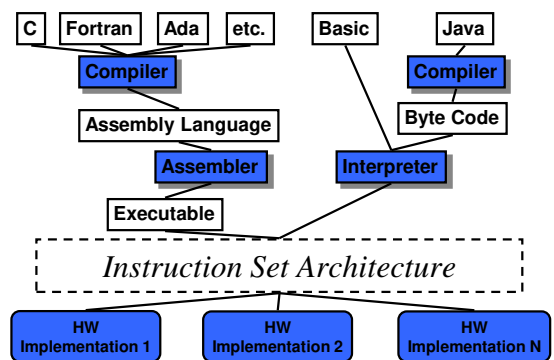
The Computer Level Hierarchy

- Each virtual machine layer is an abstraction of the level below it.
- The machines at each level execute their own particular instructions, calling upon machines at lower levels to perform tasks as required.
- Computer circuits ultimately carry out the work.



•Software?

- Program or collection of programs.
- Enables the hardware to process data.



Programming

- Methodologies for creating computer programs that perform a desired function.
 - Problem Solving
 - How do we figure out what to tell the computer to do?
 - Convert problem statement into algorithm, using **stepwise refinement**.
 - Convert algorithm into machine instructions.
 - Debugging
 - How do we figure out why it didn't work?
 - Examining registers and memory, setting breakpoints, etc.

Time spent on the first can reduce time spent on the second!

Stepwise Refinement

- Also known as **systematic decomposition**.
- Start with problem statement:
- Decompose task into a few simpler subtasks.
- Decompose each subtask into smaller subtasks, and these into even smaller subtasks, etc....

until you get to the machine instruction level.

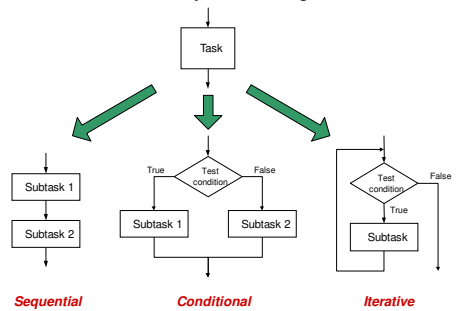
Problem Statement

- Because problem statements are written in English, they are sometimes ambiguous and/or incomplete.
 - Where is “file” located?
 - How big is it?
 - How do I know when I’ve reached the end?
 - How should final count be printed? A decimal number?
 - If the character is a letter, should I count both upper-case and lower-case occurrences?
- How do you resolve these issues?
 - Ask the person who wants the problem solved, or
 - Make a decision and document it.

19

Three Basic Constructs

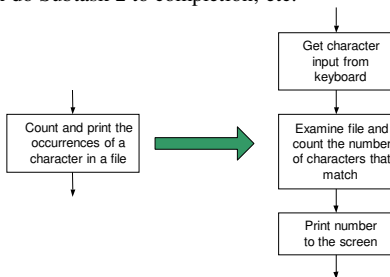
- There are three basic ways to decompose a task:



20

Sequential

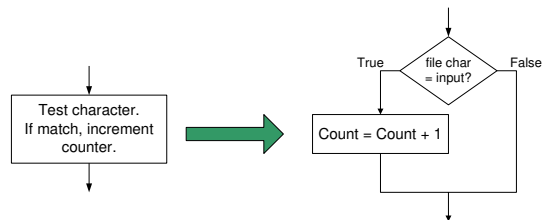
- Do Subtask 1 to completion, then do Subtask 2 to completion, etc.



21

Conditional

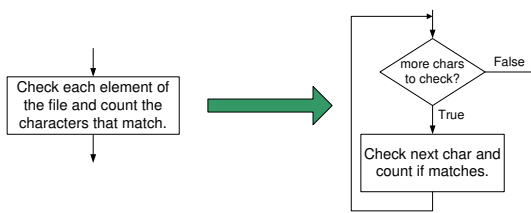
- If condition is true, do Subtask 1; else, do Subtask 2.



22

Iterative

- Do Subtask over and over, as long as the test condition is true.



23

Why Write Programs?

- Automate computer work that you do by hand
 - save time & reduce errors
- Run the same analysis on lots of similar data files
- Analyze data
- Make decisions
- Create new analysis methods

Why Perl?

- Fairly easy to learn the basics
- Many powerful functions for working with text: search & extract, modify, combine
- Can control other programs
- Free and available for **all** operating systems
- Most popular language in bioinformatics
- Many pre-built “modules” are available that do useful things

As a software tool: Perl

- **What Is Perl?**
- PERL is a “*Practical Extraction and Report Language*”
- (or *Pathologically Eclectic Rubbish Lister*)
 - freely available for Unix, MVS, VMS, MS/DOS, Macintosh, OS/2, Amiga, and other operating systems.
- Perl has powerful text manipulation functions.
 - It eclectically combines features and purposes of many command languages.
 - Perl has enjoyed popularity for programming World Wide Web electronic forms and generally as glue and gateway between systems, databases, and users.

26

History

- Originally written by Larry Wall at NASA’s Jet Propulsion Labs
 - to process mail on Unix systems
 - extended by a lot of people and many biologists!
- Started as ‘glue’ language,
 - for the use of Larry and officemates.
- It combines the best features of several languages.
- Version 1: December 18, 1987
- Current stable release is Perl 5.18.2
- Perl motto: **TMTOWTDI**-There’s **More Than One Way To Do It**

27

Strengths of Perl

- Very easy to learn
- Very portable
- High level language
- Powerful text processing
- It’s free
- What makes **Perl** a good programming language for Biological data?
 - Fast in file manipulation
 - DBI modules provide database bridge for other applications
 - CGI module provides easy web interface

28

Getting and Installing Perl

- <http://www.perl.org/>
- <http://www.perl.com/CPAN/>
- <http://www.activestate.com/>
- Perl tutorials:
 - <http://www.internetbiologists.org/IB-perl/index.html>
 - http://learn.perl.org/library/beginning_perl/
- Bioinformatics related web pages:
 - <http://www.geocities.com/bioinformaticsweb/index.html>
 - <http://glasnost.itcarlow.ie/~biobook/index.html>

29

What is Perl Used For

- CGI (common gateway interface) Programming (dynamically generating web pages).
 - (Example websites: www.amazon.com, www.slashdot.org, www.deja.com)
- Extracting data from one source and translating it to another format.
- Manipulating databases, simple search and replace operation.
- Data management in Human Genome Project
- Internet programming, automating administration tasks,etc.

30

Again: What Is Perl?

- **Interpreted** Language ?
 - (such as Basic, which needs another program called interpreted to process the code every time you want to run the program)
- **Compiled** language ?
 - (such as C, which uses a compiler to process the code before the code is ever run)
- Perl is in between like java:
 - interpreter reads and compiles the program at ones,
 - not into the specific machine code,
 - but into a special virtual machine code.
- It is also called **scripting** language

37

How to Write Perl Code

- Form a working folder (directory)
- Open Notepad (or any text editor) and type in the perl code following the convention
- Save the file with extension **pl**, or **plx**
- In file manager you double click on the file.
 - The program will run (probably a window will appear and disappear)
- Go to MSDOS prompt.
- Change to working directory and type **perl xxxx.pl**
- Or you use one of the IDEs

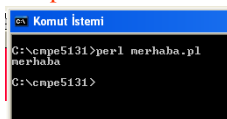
38

First Perl Program

- Here is a simple script to illustrate how a Perl program looks: (Print a message to the terminal Code:

```
#!/niPerl/bin -w
print "merhaba \n";
```

- Save this file as **merhaba.pl**
- Run it by typing
 - > **perl merhaba.pl**



39

Debugging Perl

- **Debugging**
 - Finding the errors and fixing them.
 - It is a specialized skill and it takes practice to become good at it.
- Among the beginner programmers, it is common banging your head against the keyboard for what seems like hours, only to discover the problem was actually in a completely different part of your script than where you were looking.

40

...Debugging...

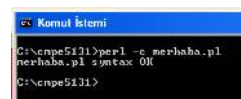
- Perl tries its best to tell you where the error is.
 - Read the error message carefully.
- Sometimes it gives multiple line errors.
- Go to the first line number and try to see the error.
- Try to look at an earlier line too,
 - sometimes the error doesn't trigger until a next statement.
- After fixing the first error, run the script again,
 - the next errors may have gone away.

41

...Debugging...

- Use the **-c** switch
 - to check for possible errors.
- By entering the command **perl -c scriptname** you make Perl to try to compile your script without actually running it
 - The **-c** switch compiles the script without invoking the warnings feature

```
print "merhaba \n";
>perl -c merhaba.pl
```



42

...Debugging...

```
print "merhaba \n";
```

```
>perl -c hmerhaba1.pl
```

43

...Debugging...

- Syntax Errors

```
pint "merhaba \n ";
```

```
>perl -c hmerhaba2.pl
```

44

Syntax and semantics

- A Perl program may be syntactically correct, but semantically wrong.
- Semantics has to do with meaning of language.
 - This means that the program satisfies the rules and regulations of language but does not do what you expected it to do.

```
print; "merhaba \n ";
```

```
> perl -c hmerhaba3.pl
```

45

...Debugging...

- Use the `-w` switch
 - to tell Perl to warn you about things it thinks might be problems in your script.

```
>perl -w hmerhaba3.pl
```

46

...Debugging...

- Use the `-cw` switches (combination of `-c` and `-w`)
 - the `-cw` compiles it with warnings turned on.
- In either case, you can get feedback on any problems your script might have, without having to actually run it which, may save you time in long and big scripts.

```
>perl -c -w hmerhaba3.pl  
>perl -cw hmerhaba3.pl
```

47

...Debugging...

- Try to isolate the problem.
 - By commenting out chunks of code, then rerunning the script, you can often narrow down where the problem is occurring.
 - Even better is to avoid the need for this by building and debugging the script in small increments.
 - Create a simple framework first, get it working, then add increasingly complex features on, testing each component before moving to the next.
 - In this way you will uncover bugs as you go, and it will usually be obvious where the bug resides; it is in the small section of code you just added.
 - While it may seem faster to code up the whole thing first, then do all the debugging at the end, it rarely works out that way.

48

...Debugging...

- Use the **strict pragma**
 - Perl is a great language for writing quick, one-off scripts, in part because of its default behavior of having new variables simply spring into existence on first mention.
 - This can lead to problems as your script grows, however.
 - A typo in the name of a variable will mean that your script is suddenly using a new, different variable from the one you intended, which can be a real head-scratcher to debug.

In computer programming, a **directive** or **pragma** (from "pragmatic") is a language construct that specifies how a compiler (or other translator) should process its input. Directives are not part of the grammar of a programming language, and may vary from compiler to compiler.

49

...Debugging...

- Use the **strict pragma**
 - By putting the following line near the top of the script:
`use strict;`
 - you are telling Perl that you are willing to be held to a higher standard.
 - In particular, you're saying you are willing to declare all your variables before using them.
 - Besides protecting you from typos in your variable names (because the script will abort with an error message during the initial compilation phase if it encounters an undeclared variable),
 - this also lets you properly "scope" your variables, thereby limiting their visibility, rather than letting them be "global" variables that could conceivably interact with other variables of the same name elsewhere in the script.
 - All of this translates into big savings in debugging time.

50

...Debugging

- Resist the temptation to attribute the problem to some previously undiscovered bug in Perl.
 - Every novice Perl programmer eventually comes up against a bug that defies all efforts to identify and eradicate it.
 - As the programmer's frustration level mounts, an idea begins to creep into his or her head:
 - it must not be a problem in the script, but is something broken in Perl itself.
- It is almost certainly a bug in your script, not in Perl.

51

Return to our 1st program

- `#!/niPerl/bin -w`
- Every line starting with `#` is comment and ignored by Perl.
- However, `#` and `!` together at the start of the 1st line tell UNIX how the file should be run.
 - In this case the file should be passed to Perl interpreter, which lives in `niPerl/bin`
- Within Perl, and almost all other programming languages, each line in a program is referred to as a "statement".
 - Perl statements end with, and are separated from any other statements by, a semicolon, that is, the `“;”` character.

52

...1st program

- 2nd line `print "merhaba \n";`
- `print` function tells perl to display the given text.
- Text inside the quotes is not interpreted as code and is called **string**.
- `\n` is used to start a new line
`#!/niPerl/bin -w`
`print "merhaba \n";`

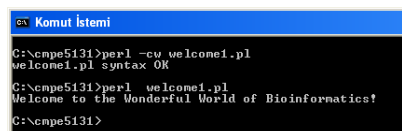
53

Program 1

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

```
>perl -cw welcome1.pl
```

```
>perl welcome1.pl
```

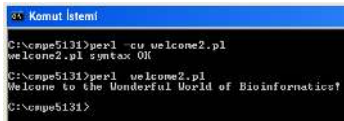


```
C:\ncmpe5131> perl -cw welcome1.pl
welcome1.pl syntax OK
C:\ncmpe5131> perl welcome1.pl
Welcome to the Wonderful World of Bioinformatics!
C:\ncmpe5131>
```

54

Another version of welcome

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



```
>perl -cw welcome1.pl
>perl welcome1.pl
```

55

Iteration (Repetition)

- Using the Perl while construct

```
# The 'iter1' 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;
}
```

56

Running forever ...

```
>perl iter1.pl
```

```
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!
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!
```

57

Iteration

- Rather than use the word **iteration** or repetition to refer to this mechanism, many programmers favour the use of the word **loop**.
 - In this context, **loop** is both a noun and a verb.
 - Typical programmer utterances might be “the loop prints the message five times” or “this program loops forever”.
 - Programs that loop forever, just like the forever program in this section, are referred to as an **infinite loop**.
 - The infinite loop is generally regarded as a *very bad thing*, and programmers are encouraged not to introduce such loops into programs.

58

Introducing variable containers

- Two **constant definitions** come after the comment lines:

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

 - Perl treats a value of 1 as **true** and a value of 0 as **false**.
- A **constant** is a container within a program whose value cannot be changed under any circumstance.
 - Although not required, it is a convention to give constants all **UPPERCASE** names.
- Perl is case-sensitive.
 - This means that when naming variables in Perl, case is significant.
 - So, “TRUE” is a different symbol to “true”.

59

Introducing variable containers

- The opposite of a constant is a **variable container**, or **variable** for short.
 - A variable's value can change over the lifetime of the program.
- When you need to change the value of an item, use a variable container.
- Perl has excellent support for all types of variable containers.
- The simplest type of variable container is the **scalar**.
 - Scalars can hold, a number, a word, a sentence or a disk-file.
- Within Perl programs, scalars are given a name prefixed with a dollar sign (\$).
 - Here are some example scalar names:
 - \$name, \$address, \$programming_101, \$z, \$abc, \$count

60

Variable containers and loops

- To demonstrate the use of variable containers within loops, a version of forever that displays ten messages and then stops can be created.

```
# The 'iter2' 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++;
}
```

61

Running ten times ...

>perl iter2.pl

```
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!
```

62

Variable containers and loops

- In Perl, it is not necessary to set the value of a variable container before it's used.
- Perl has a number of rules that are applied to the first usage of a variable container
 - Perl sets a scalar to zero if it is first used within a numeric context.
 - This feature can be very convenient.
 - However, it is always a good idea to give variable containers an explicit starting value, as it indicates precisely what the intentions for the variable are.
 - Any programmer reading the *tentimes* program should be in no doubt that the `$count` scalar is to be used within a numeric context.

63

Selection

- One of the basic building blocks of programming is selection.
- The use of a selection mechanism allows a program to choose one of a number of possible courses of action.
- Here's the general form of the selection statement in Perl:

```
if ( some condition is true )
{
    do something
}
else
{
    do something else
}
```

64

Using the Perl if construct

- Here is another variation on the forever program that prints the message fivetimes.

```
# 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;
    }
}
```

65

There Really Is MTOWTDI

- Where MTOWTDI stands for “**more than one way to do it**”.
- This philosophy is one of the great strengths of Perl, but care is needed.
- Next examples illustrate the good and the bad of this philosophy,
- Starting with a couple of not so good examples followed by a couple of much improved ones.

66

The oddeven program

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

67

The terrible program

- Here is another program that does exactly the same thing as **oddeven**.

```
#!/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"; } }
```
- Notice that the program statements that make up the terrible program are exactly the same as those that make up the **oddeven** program.
 - The difference between the two programs has to do with how they are laid out, or formatted.

68

The terrible program

- Like a lot of modern programming languages, Perl is classified as free format.
- This means that you can write a program using whatever formatting you prefer, as perl can just as easily process a well-formatted program, such as **oddeven**, as it can a poorly formatted program, such as **terrible**.
- Do yourself and everyone else a favour, and be sure to format your programs to be as readable as possible.
- Use plenty of **whitespace**, **blank lines** and **indentation** to make your programs easier to read.

69

The oddeven2 program

- Here is another version of **oddeven**.
- It produces exactly the same output as both **oddeven** and **terrible**:

```
#!/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";
    }
}
```

70

Using the modulus operator

- That percentage sign (%) is another Perl operator, the **modulus**, % operator.
- Given two numbers, “A % B” returns the **remainder** after A has been divided by B, assuming both are positive numbers.

```
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.
```

71

The oddeven3 program

- The following code is even shorter than the **oddeven2**.

```
#!/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 );
}
```

72

Processing Data Files

- The input operator in Perl is `<>`
 - When Perl encounters this operator within a program, it looks for and returns a line of input from standard input,
 - which is the name given to the mechanism that is currently providing input data to the program.
 - Unless Perl is told otherwise, the default input mechanism is the keyboard.
- A program takes a line of data from the keyboard whenever the input operator is used.
 - Consider the following program statement:


```
$line = <>;
```

 - A line is read from the keyboard and put into the `$line` scalar

73

Processing Data Files

- The following is a small program called `getlines` that exploits the program statement in the previous slide:


```
#!/usr/bin/perl -w
# The 'getlines' program which processes lines.

while ( $line = <> )
{
    print $line;
}
```
- The `getlines` program has a condition part that uses `<>` to look for and return a line from standard input.
 - The line, when available, is assigned to the `$line` scalar, which is then checked for truthness.

74

Processing Data Files

- It turns out that, in addition to using numerics to represent `true` and `false`, strings also have a truth value.
 - A string with no characters is `false`, otherwise it is `true`.
- In addition to standard input, Perl has
 - standard output,
 - the default place to display normal messages,
 - standard error,
 - the default place to display error messages.
 - Unless told otherwise, Perl uses the screen as the default for both standard output and standard error.
- To make things convenient, standard input, standard output and standard error go by the shorthand names of `STDIN`, `STDOUT` and `STDERR` respectively.

75

Processing Data Files

- Consider the following command line statement:

`> perl getlines terrible`

```
PS E:\lectures\2017-2\120\docs\Noorhouse\Codes\chapter3> perl getlines terrible
#!/usr/bin/perl -w
# The 'terrible' program - a poorly formatted 'oddeven'.
use constant HOMMANY => 4; $count = 0;
while ( $count < HOMMANY ) { $count++;
if ( $count == 1 ) { print "oddi\n"; } elsif ( $count == 2 )
{ print "even\n"; } elsif ( $count == 3 ) { print "oddi\n"; }
else # at this point $count is four.
{ print "even\n"; } }
```

`> perl getlines terrible welcome3`

```
PS E:\lectures\2017-2\120\docs\Noorhouse\Codes\chapter3> perl getlines terrible welcome3
#!/usr/bin/perl -w
# The 'terrible' program - a poorly formatted 'oddeven'.
use constant HOMMANY => 4; $count = 0;
while ( $count < HOMMANY ) { $count++;
if ( $count == 1 ) { print "oddi\n"; } elsif ( $count == 2 )
{ print "even\n"; } elsif ( $count == 3 ) { print "oddi\n"; }
else # at this point $count is four.
{ print "even\n"; } }
#!/usr/bin/perl -w
# The 'welcome3' program - our first "executable" (Perl) program.
print "Welcome to the Wonderful World of Bioinformatics!\n";
```

76

Processing Data Files

- As the `getlines` program uses
- When used in association with a named disk-file, it uses the contents of the disk-file as input, and there can be more than one named disk file.
- Programmers refer to the list of things on the command-line that follow a program name as its command-line arguments or parameters.
- The last invocation of `getlines` has two command-line arguments, the word “terrible” and the word “welcome3”.

77

Introducing Patterns

- Perl has another programming language built into it.
 - This language within a language makes extensive use of Perl’s regular expression, pattern-matching technology.
 - The Perl on-line documentation defines a regular expression to be simply a string that describes a pattern.
 - The pattern identifies what it is hoped to match.
 - The actual how of finding the pattern is taken care of by the Perl program.

78

Introducing Patterns

- A programming language that allows the programmer to specify what is required is often referred to as a **declarative** language.
 - The programmer **declares** what's required, and the technology works out the details.
- A programming language that allows the programmer to specify exactly how a result is to be arrived at is often referred to as a **procedural** language.
 - The programmer defines the **procedure** to be followed, and the technology blindly follows the instructions.
- Most programming languages can be classified as one or the other, either **declarative** or **procedural**.
- Remarkably, Perl can be one or the other, or both.

79

Introducing Patterns

- For introducing regular expressions, consider the following program, called **patterns**:

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

# The 'patterns' program - introducing
# regular expressions.

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

80

Introducing Patterns

- The **'patterns'** program is very similar to the **'getlines'** program except the **print** command within the loop's block.

```
print $line if $line =~ /even/;
```
- Here's the English language equivalent:
 - display the contents of the scalar called **\$line** if and only if the scalar called **\$line** contains the pattern **"even"**.
- In above statement, **=~** is called the **binding operator**.
- The **binding operator** compares something (usually a scalar variable container) against a pattern.
 - For now, a pattern is defined as any sequence of characters surrounded by the forward-leaning slash character **"/"**.
 - In the example above, the pattern is the word **"even"**.
 - If the contents of **\$line** contains the pattern **"even"** anywhere in the line, it is said to match.

81

Introducing Patterns

- When programmers refer to a character that surrounds something of interest, such as the forward-leaning slash surrounding the patterns in this section, they call that character a **delimiter**.
- The character **delimits** the something of interest.
- The **"/"** character is the default delimiter for regular expression patterns in Perl.

82

Running patterns ...

- To illustrate what's going on, try the following command-lines:

> perl patterns terrible

```
PS E:\Lecture5\26178-2\128\docs\Woorhouse\Codes\chapter3> perl patterns terrible
# The 'patterns' program - a poorly formatted 'oddeven'.
{ print "even\n"; } elsif ( $count == 3 ) { print "odd\n"; }
{ print "even\n"; } }
PS E:\Lecture5\26178-2\128\docs\Woorhouse\Codes\chapter3>
```

> perl patterns oddeven

```
PS E:\Lecture5\26178-2\128\docs\Woorhouse\Codes\chapter3> perl 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";
```

> perl patterns welcome2

```
PS E:\Lecture5\26178-2\128\docs\Woorhouse\Codes\chapter3> perl patterns welcome2
PS E:\Lecture5\26178-2\128\docs\Woorhouse\Codes\chapter3>
```

83

http://en.wikipedia.org/wiki/Regular_expression

<http://www.regular-expressions.info/tutorial.html>

<http://www.english.uga.edu/humcomp/perl/regex2a.html>

<http://www.perl.com/doc/manual/html/pod/perlre.html>

84

Input/Output

- Data entering a program is referred to as its **input**, while data produced by a program is its **output**.
 - Rather than refer to (and write) “input/output”, most programmers simply say “IO”, which is written as I/O.
- I/O facilities are often referred to as streams.
 - It is possible to have many streams associated with a program, with some of them classed as **input streams** and others classed as **output streams**.
 - As a minimum, every Perl program has three standard streams available to it.
 - **STDIN**, **STDOUT**, and **STDERR**

85

I/O-STDIN

- The standard input stream (**STDIN**) is the default place from which data enters a program.
- Typically, **STDIN** is the keyboard, but it can also be a disk-file.
- To read data from **STDIN**, use the input operator:

```
my $data = <STDIN>;
```

or

```
my $data = <>;
```
- Perl is smart enough to know that an “empty” input operator actually refers to **STDIN** by default.

86

I/O-STDIN

```
print "Enter a number: ";
$a = <STDIN>; #<> is okay too
print "Enter another number: ";
$b = <STDIN>; #<> is okay too
chomp $a; # chomp : removes \n from string
chomp $b; # chomp : removes \n from string
$c = $a + $b;
print "The sum of the numbers that you entered is $c";
```

- If **\$a** is omitted, it is assumed to be **\$_**

87

I/O-STDIN

```
print "Enter the username: ";
$username = <STDIN>;
chomp $username;
if ($username =~ /ibstudent/) {
    print "Welcome IB student!\n\n";
} else {
    print "Bad username, sorry!\n\n";
}
```

- Syntax of if, else statement:

```
if (a condition is met)
{do something;}
else {do something else;}
```

88

I/O- STDOUT

- The standard output stream (**STDOUT**) is the default place to which data is sent by a program.
- Typically, **STDOUT** is the screen, but it can also be a disk-file.
- To write data to **STDOUT**, use the output operator:

```
print STDOUT $data;
or
print $data;
```

- Perl is smart enough to know that **print** sends data to **STDOUT** by default.

89

I/O- STDOUT

- **STDOUT** can be altered outside your program, with the “**redirection**” operator.
- So if you were running your Perl program and you wanted to keep the output for review instead of letting it flash by on the screen, you could redirect **STDOUT** with the “**>**” symbol and have the output sent to a file like this:

```
>perl perltest.pl > output.txt;
```

90

I/O-STDOUT

- Writing the output to a particular file from within a Perl program:
 - Specify the file you want to use by opening a **filehandle** to it.
 - Use the new filehandle in print statement, instead of the default **STDOUT** filehandle.
 - When finished, close the **filehandle**.

```
open OUTPUT, ">output.txt";
print OUTPUT "hello world\n";
close OUTPUT;
```

91

I/O-STDOUT

- An example:
 - Following script creates a new web page, saved in the file: **c:/web/root/index.html**.

```
open HTML, ">c:/perlex/index.html";
print HTML "Content-Type: text/html\n\n";
print HTML "<html><head></head><body>";
print HTML "<h2>Written by Perl!</h2>";
print HTML "</body></html>";
close HTML;
```

92

FILEHANDLE

- A **filehandle** is a special type of variable that is associated with an output destination.
- It is used to tell your program where you want output to go.
 - open a file for reading
open FILEHANDLE, "chromosome2"
alternative form:
open FILEHANDLE, "< chromosome2"
 - open a file for writing
open FILEHANDLE, ">myprediction"
 - open a file for appending
open FILEHANDLE, ">>mypredictions"

93

FILEHANDLE

- Reading from file "DNA" and copying each line to "DNAcopy":

```
open IN, "DNA";
open OUT, ">DNAcopy";
while ($line = <IN>)
{
    print OUT $line;
}
```

- Syntax of while statement:

```
while (something is happening)
{do something;}
```

94

FILEHANDLE

- Reading from file "DNA" and copying each line to "DNAcopy":

```
open IN, "DNA" or die "Can't open input file: $!\n";
open OUT, ">DNAcopy" or die "Can't open output file: $!\n";
while ($line = <IN>)
{
    print OUT $line;
}
```

- Syntax of die function:

```
or die "output message"
|| die "output message"
```

- Variable **\$!** describes the error such as "file not found"

95

FILEHANDLE

- It is a good habit to close the things that you open.
- Close the filehandle once you are done with it.
- This will also happen automatically when your program ends.
- Some complex functions don't even work till after you close the files.
- Also as long as you keep files open you occupy the system's memory.

```
close IN or warn "Errors while closing filehandle: $!";
```

- Syntax of warn function:

```
or warn "output message"
|| warn "output message"
```

96

Perl Variables

- Perl programs use variables to store data in memory.
- Perl is a **typeless** language that doesn't force the programmer to distinguish between the types of data stored in a variable.
- Perl provides 3 built-in variable types:
 - Scalar
 - Array
 - Hash

97

Scalar variables

- In Perl the most basic variable type is a scalar variable.
- It holds a single value.
- Value can be any kind of data, including, but not limited to, string, integer, float, object and reference to other variables or sets of variables.
- Scalar variables are preceded with dollar sign (\$), and consist of only alpha-numeric characters.
- Following are all valid Perl variable assignments.

```
$lang = "Perl";      # <-- string. - Notice quotes
$version = 5.6;      # <-- float. - Notice lack of quotes
$year = 2001;        # <-- integer. /
$x = 10;
$value = $x + 1;
$number_of_items = 15;
$word = "hello";
$text = "This is a sentence but is still a scalar";
```

98

Array variables...

- Arrays are handy when you want to store more than one data in a single variable, but still want to be able to refer to them independently.
- Array in Perl is distinguished from scalar variables by its @ sign that proceeds its name.
- You can initialize an array variable by giving a list of values, each separated with comma inside the parenthesis:

```
@desimal = ("bir", "iki", "uc", ... "dokuz");
$array = ( 1, 2 );
@values = ( $x, $y, 3, 5);
```

99

...Array variables...

- You can also create an arbitrary array, and later re-assign it elements using a bracket ([]) operator.
 - Important: when you refer to individual elements of an array, you use \$ sign just like in scalar variables:

```
$ desimal [0] = "bir";
$ desimal [1] = "iki";
$ desimal [2] = "üç";
# ...
$ desimal [8]= "dokuz";
```

- Digits inside the [] are usually called array's index, or just index.
 - In Perl array indices start at 0, not 1.
 - That's why 10th element of an array has an index of 9

100

...Array variables

- Programming languages such as C/C++ require that all the elements of an array be of the same type, such as all integers, all characters, all strings.
- This is not the case with Perl.
 - You can mix all kinds of data types in an array:

```
$array[0] = "apple"; # <-- string
$array[1] = 12;      # <-- integer
$array[2] = 3.47     # <-- float
```

101

Appending elements to an array

- When you want to add an extra element to the end of the array, you will need to know the last index of the array.
- Special symbol, \$# can be prepended to the name of the array to get the last index number.
- For example:

```
$last_index = $#desimal;
$desimal[ $last_index + 1 ] = "on";
```

- or

```
$desimal[ $#desimal + 1 ] = "on";
```
- or

```
push (@desimal, "on");
```

102

Hash variables...

- Perl supports hash variables, which are also known as **associative arrays**.
 - Arrays, because they store multiple values, just like ordinary arrays.
 - Associative, because they associate the values of the element not with an index, but through names, also called **keys**.
- You generate keys yourself, and you can refer to those values with those keys.
- Distinguishing signature of a hash is a **%** sign, which is prepended to its name:

```
%person = (); # <-- creating an empty hash

%person = ( "l_name" => "AYDIN",
            "f_name" => "Nizamettin",
            "email" => 'naydin@yildiz.edu.tr' );
```

103

...Hash variables

- You can access the values of columns of the table individually using a **{ }** operator.
- Just like in arrays, we use not **%** sign, but **\$** to refer to individual variables:

```
$name = $person{"f_name"};
$email = $person{"l_name"};
```

104

Hash-related functions

- Just like in arrays, Perl provides several built-in functions for working with hashes.

keys() - returns all the keys (names) of the hash as an array:
@names = keys(%person);

values() - returns all the values of the hash as an array:
@values = values(%person);

delete() - deletes a key/value pair from the hash.
delete \$person{email};

exists() - returns a true value if a specific key of the hash really exists:
if (exists(\$person{f_name})) {
 # do something accordingly... }

105

String & Array...

- How to compute string length?
\$length = length \$variable;
length function returns the length of a string (number of characters)
- How to find position of character in a string?
\$length = rindex(\$variable, 'N') + 1;
rindex function returns the position of the first 'N' from the right
- Instead of 'rindex', 'index' can also be used
\$q = \$a . "?"; # "tag" the end of the string with '?'
\$x = index (\$q, '?'); # get the position of '?'

106

...String & Array

- Converting a string into an array:
 - Use 'qw' operator to create an array:
@hum_ubiq = qw(M Q I F V K T L T G K T);
Notice that the characters have to be separated by spaces
 - Use 'split' function:
\$ubiquitin = 'MQIFVKLTGTGKT';
@array = split(/,/, \$ubiquitin);
It splits string \$ubiquitin at a separator substring defined within slashes (/ / defines an empty string)

107

Example

- This is the amino acid sequence of human ubiquitin:
MQIFVKLTGTGKTITLEVEPSDTIENVKAKIQDKE
GIPPDQQRLLFAGKQLEDGRTLSDYNIQKESTLH
LVRLRLRG
- Human UBC gene encodes a precursor composed of nine direct repeats of this sequence, plus an additional valine residue (V) at the C terminus.
- Using Perl, create the sequence of the precursor, calculate its length, approximate molecular weight, and the corresponding number of nucleotides in mRNA, and finally print out the results.

108

precursor

- Main Entry: **pre·cur·sor**
- Pronunciation: \pri-'kər-sər, 'prē-, \
- Function: *noun*
- Etymology: Middle English *precursoure*, from Latin *praecursor*, from *praecurrere* to run before, from *prae-* pre- + *currere* to run — more at **CURRENT**
- Date: 15th century
- **1 a** : one that precedes and indicates the approach of another **b** : **PREDECESSOR**
- **2** : a substance, cell, or cellular component from which another substance, cell, or cellular component is formed
- **synonyms** see **FORERUNNER**
- — **pre·cur·so·ry** \-'kərs-rē, -'kər-sə- \ *adjective*

109

Example script 1

```
@ubi = qw(MQIFVKTLTGKTITLEVEPSDTI
ENVKAKIQDKEGIPPDQQRLIFAGKQLE
DGRITLSDYNIQKESTLHLVLRIRGG);
@pre_ubi = (@ubi) x 9;
push (@pre_ubi, 'V');
$length = @pre_ubi;
$mw = $length * 0.11;
$RNA_length = $length * 3;
print "The sequence of the human ubiquitin precursor
is:\n@pre_ubi\n";
print "Its length is $length amino acids.\n";
print "Its approximate molecular weight is $mw daltons.\n";
print "It is encoded by an mRNA of approximately
$RNA_length nucleotides.\n";
```

110

Example script 2

```
$ubi =
'MQIFVKTLTGKTITLEVEPSDTIENVKAKIQDKEGIPPDQ
QRLIFAGKQLEDGRITLSDYNIQKESTLHLVLRIRGG';
$pre_ubi = ($ubi x 9) . 'V';
$length = length ($pre_ubi);
$mw = $length * 0.11;
$RNA_length = $length * 3;
print "The sequence of the human ubiquitin precursor
is:\n$pre_ubi\n";
print "Its length is $length amino acids.\n";
print "Its approximate molecular weight is $mw daltons.\n";
print "It is encoded by an mRNA of approximately
$RNA_length nucleotides.\n";
```

111

Perl Operators...

- Arithmetic Operators
 - They perform some sort of mathematical functions

Operator	Function
+	Addition
-	Subtraction, Negative Numbers, Unary Negation
*	Multiplication
/	Division
%	Modulus
**	Exponent

112

...Perl Operators...

- To use these, you will place them in your statements like a mathematical expression.
- So, if you want to store the sum of two variables in a third variable, you would write something like this:

```
$advenue=20;
$ales=10;
$total_revenue = $advenue + $ales;
```

113

...Perl Operators...

- Assignment Operators

Operator	Function
=	Normal Assignment
+=	Add and Assign
-=	Subtract and Assign
*=	Multiply and Assign
/=	Divide and Assign
%=	Modulus and Assign
**=	Exponent and Assign

- Increment/Decrement

Operator	Function
++	Increment (Add 1)
--	Decrement (Subtract 1)

114

...Perl Operators...

- String Operators

Operator	Function
.	Concatenate Strings
.=	Concatenate and Assign

```
$a = "Hello";
$b = "World";
$c = $a . $b;    # $c is now "HelloWorld"
$d = $a . " " . $b; # $d is now "Hello World"
```

115

...Perl Operators...

- Numeric Comparison

- These operators are used to compare two numbers, but not to compare strings.
- These operators are typically used in some type of conditional statement that executes a block of code or initiates a loop.

Operator	Function
==	Equal to
!=	Not Equal to
>	Greater than
<	Less than
>=	Greater than or Equal to
<=	Less than or Equal to

116

...Perl Operators...

- String Comparison

- These are similar to the numerical comparisons, but they work with strings.

Operator	Function
eq	Equal to
ne	Not Equal to
gt	Greater than
lt	Less than
ge	Greater than or Equal to
le	Less than or Equal to

117

...Perl Operators...

- String Comparison

- The greater-than and less-than operators compare strings using alphabetical order.
- Something that starts with "a" is greater than something that starts with "c".
- Also, small letters are greater than capital letters.
- Thus, "hello" is greater than "Hello".
- So, that is how it will compare it.

118

...Perl Operators

- Logical Operators

- These are often used when you need to check more than one condition.

Operator	Function
&&	AND
	OR
!	NOT

- So, if you want to see if a number is less than or equal to 10, and also greater than zero:

```
$number=5; if (($number <= 10) && ($number > 0))
{
    ...code....
}
```

119

Perl Functions...

- Function is a collection of code (statements), which can be easily configured by passing lists of arguments.
- Functions in Perl are called [subroutines](#), and have a general syntax of:

```
sub (list of arguments) {
    list of statements to execute
    return some value }
```

120

...Perl Functions

- Let's create a simple function to compute the area of the triangle.
- This function will receive 2 arguments; triangles width and its height and returns final computed value:

```
sub ucgen ($genislik, $yukseklilik) {  
    $alan = $genislik * $yukseklilik / 2;  
    return $alan;  
}
```

- We can now call the above function with various arguments, and each time it will return a computed value for a triangle:

```
$alan1 = ucgen(3, 4);  
$alan2 = ucgen(45, 38);
```

121

Matching and Substitution in Perl...

- The ease and power of Perl's pattern matching is one its true strengths and a big reason why Perl is as popular as it is.
- Almost every script you write in Perl will have some kind of pattern matching operation because so often you want to seek something out, and then take an action when you find it.
- Matching and substitution are very important because this is how you do editing "on the fly".
- This is how you create content customized to your Web visitor.

122

...Matching and Substitution in Perl

- You need to be able to open HTML templates and swap in information pertaining to your visitor.
- Matching and then substituting is just the way to do it.
- Also in many other administrative tasks, such as searching through log files or web pages for particular words or sequences, pattern matching is the way to go.
- Pattern matching, in Perl at least, is the process of looking through sections of text for particular words, letters-within-words, character sequences, numbers, strings of numbers, html tags.
- These more complicated search expressions fall into the category of "**regular expressions**".

123

The Binding Operator...

- When you do a pattern match, you need three things:
 - the text you are searching through
 - the pattern you are looking for
 - a way of linking the pattern with the searched text
- As a simple example, let's say you want to see whether a string variable has the value of "success".
- Here's how you could write the problem in Perl:

```
$word = "success";  
if ( $word =~ m/success/ ) {  
    print "Found success\n";  
} else {  
    print "Did not find success\n";  
}
```

124

...The Binding Operator...

- The "**=~**" construct, called the **binding operator**, is what binds the string being searched with the pattern that specifies the search.
 - The binding operator links these two together and causes the search to take place.
- Next, the "**m/success/**" construct is the matching operator, **m/**, in action.
 - The "**m**" stands for matching to make it easy to remember. The slash characters here are the "**delimiters**".
 - They surround the specified pattern.
- In **m/success/**, the matching operator is looking for a match of the letter sequence: **success**.
- Generally, the value of the matching statement returns 1 if there was a match, and 0 if there wasn't.

125

...The Binding Operator

- **Negative Matching**
 - In some cases you are more interested in whether a pattern does **not** match a string rather than that it does. In this case you could write
 - if (! \$string =~ m/search text/) ...
 - but as usual, Perl makes it easier for you and offers you more than one way to do it.
- In this case, there's the "**negative**" binding operator, **!~**, so you could write this:
 - if (\$string !~ m/search text/) ...

126

Matching...

- Parentheses `()` group pattern elements.
- An asterisk `*` means that the preceding character, element, or group of elements may occur zero times, one time, or many times.
- A plus `+` means that the preceding element or group of elements must occur at least once.
- A question mark `?` matches zero or one times.
- So:
 - `/fr.*nd/` matches "frnd", "friend", "front and back"
 - `/fr.+nd/` matches "frond", "friend", "front and back" *but not* "frnd".
 - `/10*1/` matches "11", "101", "1001", "100000001".
 - `/b(an)*a/` matches "ba", "bana", "banana", "banananana"
 - `/flo?at/` matches "flat" and "float" *but not* "float"

127

...Matching...

- Square brackets `[]` match a class of single characters.
 - `[0123456789]` matches any single digit
 - `[0-9]` matches any single digit
 - `[0-9]+` matches any sequence of one or more digits
 - `[a-z]+` matches any lowercase word
 - `[A-Z]+` matches any uppercase word
 - `[ab n]*` matches the null string "", "b", any number of blanks, "nab a banana"

128

...Matching...

- `[^...]` matches characters that are *not* "...".
- `[^0-9]` matches any non-digit character.
- Curly braces allow more precise specification of repeated fields. For example
 - `[0-9]{6}` matches any sequence of 6 digits, and
 - `[0-9]{6,10}` matches any sequence of 6 to 10 digits.
- Patterns float, unless anchored. The caret `^` (outside `[]`) anchors a pattern to the beginning, and dollar-sign `$` anchors a pattern at the end, so:
 - `/at/` matches "at", "attention", "flat", & "flatter"
 - `/^at/` matches "at" & "attention" *but not* "flat"
 - `/at$/` matches "at" & "flat", *but not* "attention"
 - `/^at$/` matches "at" and nothing else.
 - `/^at$/i` matches "at", "At", "aT", and "AT".
 - `/^[\t]*$/` matches a "blank line", one that contains nothing or any combination of blanks and tabs.

129

...Matching...

- *The Backslash.* Other characters simply match themselves, but the characters `+.?*^$()[]\` and usually `/` must be escaped with a backslash `\` to be taken literally. Thus:
 - `/10.2/` matches "10Q2", "1052", and "10.2"
 - `/10\.\2/` matches "10.2" *but not* "10Q2" or "1052"
 - `/^+ /` matches one or more asterisks
 - `/A:\\DIR/` matches "A:\\DIR"
 - `/\\usr\\bin/` matches "\\usr/bin"
- If a backslash precedes an alphanumeric character, this sequence takes a special meaning, typically a short form of a `[]` character class. For example, `\\d` is the same as the `[0-9]` digits character class.
 - `/[-+]?\\d*\\.?\\d*/` is the same as
 - `/[-+]?[0-9]*\\.?\\d*/`
 - Either of the above matches decimal numbers: "-150", "-4.13", "3.1415", "+0000.00", etc.

130

...Matching

- A simple `\\s` specifies "white space", the same as the character class `[\\t\\n\\r\\f]` (blank, tab, newline, carriage return, form-feed). A character may be specified in hexadecimal as a `\\x` followed by two hexadecimal digits; `\\x1b` is the ESC character.
- A vertical bar `|` specifies "or".


```
if ($answer =~ /^y|^yes|^yeah/i ) { print
    "Affirmative!"; }
```

 prints "Affirmative!" for \$answer equal to "y" or "yes" or "yeah" (or "Y", "YeS", or "yessireebob, that's right").

131

Regular Expressions...

- All pattern matching in Perl is based on the concept of **regular expressions**.
- **Regular expressions** are an important part of computer science, and entire books are devoted to the topic.
- **Regular expressions** form a standard way of expressing almost any text pattern unambiguously.
- A mechanism to select specific strings from a set of character strings.
- A set of characters, metacharacters, and operators that define a string or group of strings in a search pattern.

132

...Regular Expressions

- A string containing wildcard characters and operations that define a set of one or more possible strings.
- A **regular expression** (abbreviated as **regexp**, **regex**, or **regxp**, with plural forms **regexps**, **regexes**, or **regexen**) is a string that describes or matches a set of strings, according to certain syntax rules.
- **Regular expressions** are used by many text editors and utilities to search and manipulate bodies of text based on certain patterns.
- Many programming languages support **regular expressions** for string manipulation

133

Strictness...

- Perl breaks a number of the “golden rules” of the traditional programming language
 - allows variables to be used before they are declared
 - Subroutines can be invoked before they are defined
- All variables are global by default
 - the use of **my** variables turns a global variable into a lexical.
 - By default, the use of **my** variables is optional
 - However, it is possible to have perl insist on the use of **my** variables, making their use mandatory.

134

...Strictness...

- This insistence is referred to as **strictness**, and is switched on by adding the following line to the top of a program:
`use strict;`
- This is a directive that
 - tells perl to insist on all variables being declared before they are used,
 - all subroutines be declared (or defined) before they are invoked.

135

...Strictness...

- As programs get bigger, they become harder to maintain.
 - The use of **use strict** helps keep things organised and reduces the risk of errors being introduced into programs.
 - Anything that helps reduce errors is a good thing, even if it is sometimes inflexible.
- Thinking about the scope of variables, and using **my** and **our** to control the visibility of variables, becomes important as a program grows in size.

136

...Strictness

- When strictness is enabled, perl checks the declaration of each of a program’s variables before execution occurs.
- Consider the following program:

```
#!/usr/bin/perl -w
# bstrict - demonstrating the effect of strictness.
use strict;
$message = "This is the message.\n";
print $message;
```
- In the code, **\$message** scalar is not declared as a lexical (**my**) or global (**our**) variable.

137

Results from bstrict...

- When an attempt is made to execute the **bstrict** program, perl complains that the strictness rules have been broken:

```
>perl -w strict.pl
Global symbol "$message" requires explicit package name at bstrict line 7.
Global symbol "$message" requires explicit package name at bstrict line 9.
Execution of bstrict aborted due to compilation errors.
>
```
- These “compilation errors” are fixed by declaring the **\$message** scalar as a **my** variable

```
my $message = "This is the message.\n";
```

138

...Results from beststrict

```
# beststrict - demonstrating the effect of strictness.
use strict;
my $message = "This is the message.\n";
print $message;
```

```
>perl -w mystrict.pl
This is the message.
>Exit code: 0
```

139

use subs

- Perl provides the `use subs` directive that ...
 - can be used in combination with `use strict`
 - to declare a list of subroutines at the top of the program

For example:

```
use strict;
use sub qw( drawline biod2mysql);
```

- The `use subs` directive declares a list of subroutine names that are later defined somewhere in the program's disk-file.

140

Perl One-Liners...

- Perl usually starts with the following line:

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

w: warning

– instructs perl to warn the programmer when it notices any dubious programming practices

- `-e` switch checks whether a module installed correctly or not:

```
>perl -e 'use ExampleModule'
```

e: execute

– instructs perl to execute the program statements included within the single quotes

141

...Perl One-Liners

- Other examples :

```
>perl -e 'print "Hello from a Perl one-liner.\n";'
```

– a single line of Perl code is provided to perl to execute immediately from the command-line

```
>perl -e 'printf "%0.2f\n", 30000 * .12;'
```

– turns perl into a simple command-line calculator

- The `printf` subroutine is a variant of the more common `print`, and prints to a specified format.

- The ability to use the `-e` switch on the command-line in this way creates what is known in the perl world as...

a one-liner.

142

Perl One-Liners: Equivalents...

- Another useful switch is `-n`, which, when used with `-e`, treats the one-liner as if it is enclosed with a loop.
- Consider this one-liner:

```
>perl -ne 'print if /ctgaatagcc/;' embl.data
```

which is equivalent to the following program statement:

```
while ( <> )
{
    print if /ctgaatagcc/;
}
```

143

...Perl One-Liners: Equivalents

- When the one-liner is executed, the following output is generated:

```
attgtaatat ctgaatagcc actgattttg taggcacctt tcagtccatc tagtgactaa
```

- Same function can also be implemented using `grep`:

```
>grep 'ctgaatagcc' embl.data
```

- When the `-n` switch is combined with `-p`, the loop has a `print` statement added to the end.

144

Perl One-Liners: More Options...

- Here is a one-liner that prints only those lines from the **embl.data** disk-file that **do not end in four digits**:

```
>perl -npe 'last if /\d{4}$/;' embl.data
```

- The above one-liner is equivalent to this program:

```
while ( <> )
{
    last if /\d{4}$/;
}
continue {
    print $_;
}
```

- This one-liner is a little harder to do with **grep**.

```
> grep -v '[0123456789][0123456789][0123456789][0123456789]$" embl.data
```

145

...Perl One-Liners: More Options...

- When executed, the following output is produced:

```
gccacagatt acaggaagtc atatttttag acctaaatca ctatctctta tctttcagca 60
agaaaagaac atttacttgg ttctgttccc tatccaagat tcagatgggtg aaacgagtga 120
tcatgcacct gatgaacgtg caaaaccaca gtcaagccat gacaaccccg attctacagtt 180
tgatgtlgaa actgccgatt ggtacgccta cagtgaaaac tatggcacaa gtgaagaaaa 240
acgtttgtt aagtttgtt caactcaaat tgacgagctt aaatcacgt acaagggtgc 300
agagatttac ctgatacgga atgaactcga ttattgggtg tttagcccta aagatgggtc 360
tagattcagc cctgactaca tctgtatcat taatgatgct gaaaatagtg aaatgtacta 420
tcaatgccta attgagccta aagtggttca ttgtctlgaa aaggatactt ggaagagaga 480
agttattgatt agtttgagtg atgaaagcca aattgttttt gatgcagatc aagatgattc 540
acaaaactat gttagttctt taatgaagat taagagccat gttataagg aagttaaatg 600
tttaggcctc aaattctaca ataccgaacc acgactcgaa tcagattttg ctattgatit 660
tcacaatagg atgccgagtt aatctaagtt tctcactgta acctgctgat tattatcttt 720
ttgtgaagtt gctacataat attgttttta agatcatgta ataaaaaagc cagctctata 780
ctgctttttt tattgcttaa aattatattc cgaatgcttg tcaaaactgc aagtatgcag 840
tcttgaccag gcatctaggg gtcgtctcag aattcggaaa ataaagcacg ctaaggcgta 900
gtcaccccg gactccccc gcccgaatga gcgagcttgc ttctgtcttg cagtgcagca 960
```

146

...Perl One-Liners: More Options

- The above one-liner is equivalent to this program:

```
while ( <> )
{
    last if /\d{4}$/;
}
continue {
    print $_;
}
```

- This one-liner is a little harder to do with **grep**.

```
> grep -v '[0123456789][0123456789][0123456789][0123456789]$" embl.data
```

147

Running Other Programs From Perl...

- There are two main ways to do this:

- By invoking the program in such a way that after execution, the calling program can determine whether the called program successfully executed.

- Perl's in-built system subroutine behaves in this way

- By invoking the program in such a way that after execution, any results from the called program are returned to the calling program.

- Perl's **backticks** and **qx//** operator behaves in this way

- Following example program demonstrates each of these mechanisms by invoking the DOS utility program, **dir**, that lists disk-files in the current directory

148

...Running Other Programs From Perl...

```
#!/usr/bin/perl -w
# pinvoke - demonstrating the invocation of other programs
# from Perl.
use strict;

my $result = system("dir *.*");
print "The result of the system call was as follows:\n$result\n";

$result = `dir *.*`; # warning: $result = 'dir *.*'; will not work
print "The result of the backticks call was as follows:\n$result\n";

$result = qx/dir *.*;
print "The result of the qx// call was as follows:\n$result\n";
```

149

...Running Other Programs From Perl

- The invocation of **system** results in the **dir** program executing. Any output from **dir** is displayed on screen (**STDOUT**) as normal.

- As **dir** executed successfully, a value of zero is returned to **pinvoke** and assigned to the **\$result** scalar.

- The **\$result** scalar is then printed to **STDOUT** as part of an appropriately worded message.

- If the **dir** program fails, the **\$result** scalar is set to **-1**.

- Perl's **backticks** (**`** and **`**) also execute external programs from within Perl.

- The results from the program are captured and returned to the program.

- In the **pinvoke** program, the results are assigned to the **\$result** scalar, and then printed to **STDOUT** as part of an appropriately worded message.

- The **qx//** operator is another way to invoke the **backticks** behaviour:

- it works exactly the same way as **backticks**

150

Results from pinvoke

Recovering from Errors...

- It is not always appropriate to **die** whenever an error occurs.
 - Sometimes it makes more sense to spot, and then recover from, an error.
- This is referred to as **exception handling**.
- Consider the following code:


```
my $first_filename = "itdoesnotexist.txt";

open FIRSTFILE, "$first_filename"
or die "Could not open $first_filename. Aborting.\n";
```

151

152

...Recovering from Errors...

- Executing the code


```
>perl -w errorrec.pl
Name "main::FIRSTFILE" used only once; possible typo at
errorrec.pl line 4.
Could not open itdoesnotexist.txt. Aborting.
>Exit code: 2
```
- This assumes that the **itdoesnotexist.txt** disk-file does not exist.
 - The program terminates as a result of the invocation of **die**.
- It is possible to protect this code by enclosing it within an **eval** block.

153

...Recovering from Errors...

- The in-built **eval** subroutine takes a block of code and executes (or evaluates it).
- When perl invokes **eval**, anything that happens within the **eval** block that would usually result in a program terminating is caught by perl and does not terminate the program.
- Here's exception handling by an **eval** block:


```
eval {
    my $first_filename = "itdoesnotexist.txt";
    open FIRSTFILE, "$first_filename"
    or die "Could not open $first_filename.
    Aborting.\n";
};
```

154

...Recovering from Errors

- If **die** is invoked within an **eval** block, the block immediately terminates and perl sets the internal **\$_** variable to the message generated by **die**.
- After the **eval** block, it is a simple matter to check the status of **\$_** and act appropriately.
- Adding the following **if** statement after the above **eval** block:


```
if ( $_ )
{
    print "Calling eval produced this message: $_";
}
```

 prints the following message to **STDOUT** when the **itdoesnotexist.txt** disk-file does not exist:


```
Calling eval produced this message: Could not open itdoesnotexist.txt. Aborting.
```

155

Sorting...

- Perl provides powerful in-built support for sorting.
 - sort** and **reverse**,
 - can be used to sort lists of strings or numbers into ascending order, descending order or any other customized order.
- Following examples demonstrate usage of **sort** and **reverse**.
 - In the following program a list of four short DNA sequences is assigned to an array called **@sequences**, which is then printed to **STDOUT**

156

...Sorting...

```
#!/usr/bin/perl -w
# sortexamples - how Perl's in-built sort subroutine works.

use strict;

my @sequences = qw( gctacataat attgttttta aattatattc cgaatgctgg );
print "Before sorting:\n\t-> @sequences\n";

my @sorted = sort @sequences;
my @reversed = sort { $b cmp $a } @sequences;
my @also_reversed = reverse sort @sequences;
print "Sorted order (default):\n\t-> @sorted\n";
print "Reversed order (using sort { \$b cmp \$a }):\n\t-> @reversed\n";
print "Reversed order (using reverse sort):\n\t-> @also_reversed\n";
```

157

...Sorting...

- Results from sort examples

>perl -w sort1.pl

Before sorting:

-> gctacataat attgttttta aattatattc cgaatgctgg

Sorted order (default):

-> aattatattc attgttttta cgaatgctgg gctacataat

Reversed order (using sort { \$b cmp \$a }):

-> gctacataat cgaatgctgg attgttttta aattatattc

Reversed order (using reverse sort):

-> gctacataat cgaatgctgg attgttttta aattatattc

>Exit code: 0

158

...Sorting

- my @sorted = sort @sequences;
 - created by invoking the in-built sort subroutine
 - sorts the array alphabetically in ascending order (from “a” through to “z”).
- my @reversed = sort { \$b cmp \$a } @sequences;
 - also created by invoking the in-built sort subroutine
 - sorts the array alphabetically in descending order (from “z” through to “a”).
- my @also_reversed = reverse sort @sequences;
 - created by first sorting the array, then reversing the sorted list by invoking the in-built reverse subroutine.
 - Note that the reverse subroutine reverses the order of elements in a list; it does not sort in reverse order.

159

Another Sorting Example

- It is also possible to sort in numerical order using sort
 - The following program defines a list of chromosome pair numbers and assigns them to another array, called @chromosomes, and the array is then printed to STDOUT:

```
my @chromosomes = qw( 17 5 13 21 1 2 22 15 );
print "Before sorting:\n\t-> @chromosomes\n";

@sorted = sort { $a <=> $b } @chromosomes;
@reversed = sort { $b <=> $a } @chromosomes;
print "Sorted order (using sort { \$a <=> \$b }):\n\t-> @sorted\n";
print "Reversed order (using sort { \$b <=> \$a }):\n\t-> @reversed\n";
```

160

And its results

```
>perl -w sort2.pl
Before sorting:
-> 17 5 13 21 1 2 22 15
Sorted order (using sort { $a <=> $b }):
-> 1 2 5 13 15 17 21 22
Reversed order (using sort { $b <=> $a }):
-> 22 21 17 15 13 5 2 1
>Exit code: 0
```

- To learn more, use the following command-line to read the on-line documentation for sort that comes with Perl:

```
>perldoc -f sort
>man sort
```

161

The sortfile Program...

- The following program takes any disk-file and sorts the lines in the disk-file in ascending order

```
#!/usr/bin/perl -w
# sortfile - sort the lines in any file.
use strict;
my @the_file;
while ( <> )
{
    chomp;
    push @the_file, $_;
}
my @sorted_file = sort @the_file;
foreach my $line ( @sorted_file )
{
    print "$line\n";
}
```

162

...The sortfile Program

- Before running **sortfile**
- After running **sortfile**

Icinde ve evlerde balkon
Bir tabut kadar yer tutar
Camasirlarinizi asarsiniz hazir kefen
Sezlongunuza uzanin olu
Gelecek zamanlarda
Oluleri balkonlara gomecekler
Insan rahat etmeyecek
Oldukten sonra da
Bana sormayin boyle nereye
Kosa kosa gidiyorum
Alnindan opmeye gidiyorum
Evleri balkonsuz yapan mimarlarin

```
cs Komut Istemi
C:\niPerl\samples\cse5555>perl sortfi
Alnindan opmeye gidiyorum
Bana sormayin boyle nereye
Bir tabut kadar yer tutar
Camasirlarinizi asarsiniz hazir kefen
Evleri balkonsuz yapan mimarlarin
Gelecek zamanlarda
Icinde ve evlerde balkon
Insan rahat etmeyecek
Kosa kosa gidiyorum
Oldukten sonra da
Oluleri balkonlara gomecekler
Sezlongunuza uzanin olu
C:\niPerl\samples\cse5555>_
```

- Same thing could also be done by using Linux sort utility in command-line:
sort sort.data

163

HERE Documents

- Consider the requirement to display the following text on screen in exactly the format shown from within a program:

Shotgun Sequencing

This is a relatively simple method of reading a genome sequence. It is "simple" because it does away with the need to locate individual DNA fragments on a map before they are sequenced.

The Shotgun Sequencing method relies on powerful computers to assemble the finished sequence.

164

Without HERE Documents

- Could be done by using a sequence of print statements as follows:

```
print "Shotgun Sequencing\n";
print "This is a relatively simple method of reading\n";
print "a genome sequence. It is \"simple\" because\n";
print "it does away with the need to locate\n";
print "individual DNA fragments on a map before\n";
print "they are sequenced.\n\n";
print "The Shotgun Sequencing method relies on\n";
print "powerful computers to assemble the finished\n";
print "sequence.\n";
```

165

Output

```
>perl -w shotgun1.pl
Shotgun Sequencing
```

This is a relatively simple method of reading a genome sequence. It is "simple" because it does away with the need to locate individual DNA fragments on a map before they are sequenced.

The Shotgun Sequencing method relies on powerful computers to assemble the finished sequence.

```
>Exit code: 0
```

166

With HERE Documents

- A better way to do this is to use Perl's HERE document mechanism
`my $shotgun_message = <<ENDSHOTMSG;`
`Shotgun Sequencing`

This is a relatively simple method of reading a genome sequence. It is "simple" because it does away with the need to locate individual DNA fragments on a map before they are sequenced.

The Shotgun Sequencing method relies on powerful computers to assemble the finished sequence.
ENDSHOTMSG

```
print $shotgun_message;
```

167

Output

```
>perl -w shotgun2.pl
Shotgun Sequencing
```

This is a relatively simple method of reading a genome sequence. It is "simple" because it does away with the need to locate individual DNA fragments on a map before they are sequenced.

The Shotgun Sequencing method relies on powerful computers to assemble the finished sequence.

```
>Exit code: 0
```

168

Even Better HERE Documents

- It is possible to improve previous program by removing the need for the \$shotgun message scalar and printing the HERE document directly, as follows:

```
print <<ENDSHOTMSG;
Shotgun Sequencing
```

This is a relatively simple method of reading a genome sequence. It is "simple" because it does away with the need to locate individual DNA fragments on a map before they are sequenced.

The Shotgun Sequencing method relies on powerful computers to assemble the finished sequence.

```
ENDSHOTMSG
```

169

Output

```
>perl -w shotgun3.pl
```

Shotgun Sequencing

This is a relatively simple method of reading a genome sequence. It is "simple" because it does away with the need to locate individual DNA fragments on a map before they are sequenced.

The Shotgun Sequencing method relies on powerful computers to assemble the finished sequence.

```
>Exit code: 0
```

170

HERE Documents

- HERE documents are useful,
 - especially when it comes to dynamically producing HTML documents.
 - This use of HERE documents will be discussed later

171

Downloading Datasets...

- Downloading from the Web
 - a highly interactive mechanism
 - useful for downloading individual data-files
 - cumbersome for downloading large number of data-files
- Some technologies allow the easy integration of data sources across the Internet.
- However, it is often convenient to download frequently used datasets and store them locally.

172

...Downloading Datasets...

- The advantages of downloading and storing datasets locally :
 - Ease of access
 - accessing data-files on a local hard disk easier than writing an interface routine to download them as needed from a – possibly congested – location on the Internet.
 - Speed
 - Local hard-disk access, even over a shared file system, is usually faster than operating through external networks to Internet locations.
 - When the processing is performed locally, it may be possible to allocate extra computational resources to the analysis.

173

...Downloading Datasets...

- Reliability
 - Accessing local hard-disk copies of data-files is more reliable than network connections and WWW servers.
- Stability
 - If the data changes frequently, it is often helpful to "freeze" it by downloading a copy and using it locally until all analyses are completed.
- Flexibility
 - Often the search facilities that exist on the WWW lack certain required functionality.
- Security
 - Data or results are often sensitive, and sending them to a remote, third-party Internet site may be unacceptable.

174

...Downloading Datasets...

- The disadvantages of downloading and storing datasets locally :
 - **Stale data**
 - The local copy is a one-time “snapshot” of the dataset at a particular point in time.
 - At some stage, it will need to be updated or replaced by newer data.
 - **Storage**
 - The dataset has to be stored somewhere, and some datasets can be large.
 - The Protein Databank (PDB) is close to four gigabytes, and the PDB is one of the smaller databases!
 - Consequently, storing multiple copies of the PDB is often impractical.
 - **Performance**
 - The centralised specialist services accessible from the WWW are often configured with dedicated parallelised systems, designed to service requests as quickly as possible.
 - If the stored dataset is designed with such systems in mind, it is unlikely that a local system will be able to match this advanced processing capability.
 - Consequently, some analyses may be slower locally when compared to those performed on the WWW.

175

...Downloading Datasets

- can be accomplished in a number of ways:
 - by using **established sequence analysis programs**, such as **EMBOSS**
 - <http://emboss.sourceforge.net/>
 - have specific methods for performing downloads.
 - Typically, datasets are accessed via a standard network connection to remote Internet sites.
 - Frequently, downloads are automated to occur at regular intervals.
 - The **wget** program, included with most Linux systems, can be used to do just this.
 - **wget** is an excellent example of GNU software as distributed by the Free Software Foundation.
 - <https://www.gnu.org/software/wget/>
 - <http://gnuwin32.sourceforge.net/packages/wget.htm>

176

Using wget to download PDB data-files

- To download a single data-file via anonymous FTP, simply provide the URL of the data-file required after the **wget** command.
 - For example, to download the two PDB structures, use these commands:


```
mkdir structures
cd structures
wget ftp://ftp.rcsb.org/pub/pdb/data/structures/all/pdb1m7t.ent.Z
wget ftp://ftp.rcsb.org/pub/pdb/data/structures/all/pdb1lqt.ent.Z
```

177

Mirroring a dataset

- **wget** can be used to mirror datasets.
 - to download the entire PDB, which is four gigabytes of data, stored in over 18000 data-files:


```
wget --mirror ftp://ftp.rcsb.org/pub/pdb/data/structures/all/pdb
```
- Such a command should be invoked only when there is a real need to mirror the PDB.
 - a download of this size takes a considerable amount of time and disk space.
 - If such a need exists, once complete, another invocation of the same command downloads only additions or updates to the PDB since the last mirror.

178

Smarter mirroring...

- The **wget** command (in slide 178) results in a deep directory tree.
 - The actual data-files are found in `structures/ftp.rcsb.org/pub/pdb/data/structures/all/pdb`
- Such a deep directory structure can be very inconvenient and frustrating to navigate.
- Following **wget** invocation can help with this problem:


```
wget --output-file=log --mirror --http-user=anonymous \
--http-passwd=email@where.ever.net \
--directory-prefix=structures/mmCIF \
--no-host-directories \
--cut-dirs=6 ftp://ftp.rcsb.org/pub/pdb/data/structures/all/pdb
```

179

...Smarter mirroring

- The **wget** command sets a number of options:
 - **--output-file**
 - a disk-file into which any message produced by **wget** is placed.
 - **--mirror**
 - turns on mirroring.
 - **--http-user**
 - sets the web username to use (if needed).
 - **--http-passwd**
 - sets the web password to use (if needed).
 - **--directory-prefix**
 - the place to put the downloaded data-files.
 - **--no-host-directories**
 - the instruction **not** to use the hostname when creating a mirrored directory structure, which is the “ftp.rcsb.org” part.
 - **--cut-dirs**
 - instructs **wget** to ignore the indicated number of directory levels.
 - In the previous example, six directory levels are to be ignored, that is, the “pub/pdb/data/structures/all/pdb” part.

180

Downloading a subset of a dataset...

- On many occasions, the entire contents of an FTP site might not be required
- `wget` can fetch a specific data-file, placing it in the current directory.
 - Use a command similar to this:
`wget ftp://beta.rcsb.org/pub/pdb/uniformity/data/mmCIF/all/1ger.cif.Z`
- While multiple URLs to data-files can be supplied on the command-line (separated by spaces), it is often more convenient to place the URLs in a data-file and use the “`--input file=`” switch.

181

...Downloading a subset of a dataset...

- The `pdbsselect` program takes the [PDB-Select](#) list produced in the [Non-Redundant Datasets](#) (discussed later), builds a list of URLs, removes the duplicates and then downloads them:

```
#!/usr/bin/perl
# pdbsselect <list of PDB IDs> - a program that takes a list of PDB ID
# codes; build a list of URLs for them;
# and automates the downloading of them
# using 'wget'.
use strict;
my $Base_URL = "ftp://ftp.rcsb.org/pub/pdb/data/structures/all/pdb";
my $Output_Dir = "structures";
open URL_LIST, ">pdb_select_url.lst"
```

182

...Downloading a subset of a dataset...

```
or die "Cannot write to file: 'pdb_select_url.lst'\n";
while ( <> )
{
if ( !Failed )
{
next;
}
}
$/ //g;
my ( $Structure, $Length ) = split ( ":", $_ );
my ( $ID, $Chain ) = split ( " ", $Structure );
$ID =~ tr /[A-Z]/[a-z]/;
print URL_LIST "$Base_URL/pdb$ID.ent.Z\n";
}
```

183

...Downloading a subset of a dataset...

```
close URL_LIST;
if ( !-e $Output_Dir )
{
system "mkdir $Output_Dir";
}
if ( !-w $Output_Dir or !-d $Output_Dir )
{
die "ERROR: Cannot access directory: '$Output_Dir'. Exiting\n";
}
system "sort -u pdb_select_url.lst > unique_urls.lst";
system "rm $Output_Dir/* > /dev/null";
system "wget --output-file=log --http-user=anonymous \
--http-passwd=email@some.where.net \
--directory-prefix=$Output_Dir -i unique_urls.lst";
```

184

...Downloading a subset of a dataset

- This program takes a list of [PDB ID codes](#) from [STDIN](#) and downloads them from the URL specified in the scalar variable `$Base_URL6`.
 - Those structures marked as [Failed](#) are skipped, otherwise a URL is built and written to the `pdbsselect_url.lst` file.
 - Duplicate structures are filtered out using the `sort -u` operating system utility.
 - Error-checking is performed to see if the output directory exists (otherwise it is created) and that the directory can be accessed.
 - All previous files in it are then deleted using the `rm` system call.
 - Finally, `wget` is invoked with the list of URLs.

185

The Protein Databank...

- The similarity between the amino acid sequence of a “new” protein and one previously characterized can give an indication of the function of the new protein.
 - Sequence search algorithms assume some groups of amino acids have similar functional roles and consequently, occur in both sequences.
 - It is also assumed that these amino acids have similar local structures (the amino acids arrangement in space).
- It is these structures that determine the function of a protein.
 - Although these assumptions are useful as a working model.

186

...The Protein Databank

- Determining the detailed structure of a protein is more difficult than finding a DNA or amino acid sequence.
- The aim of some structural studies
 - to know how the protein (or other biomolecule) “does what it does”
 - to alter its function.
 - for example, to design a small molecule that binds to the protein, more commonly known as a “drug”.

187

Determining Biomolecule Structures

- There are many methods used for gaining information about the structure of a biomolecule
- The two major methods by which the location of atoms can be determined to a useful accuracy
 - X-Ray Crystallography
 - Nuclear Magnetic Resonance (NMR).

188

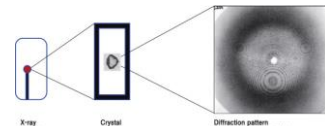
X-Ray Crystallography...

- a technique for determining the three-dimensional structure of molecules,
 - including complex biological macromolecules such as proteins and nucleic acids.
- a powerful tool in the elucidation of the three-dimensional structure of a molecule at atomic resolution.
- Data is collected by diffracting X-rays from a single crystal, which has an ordered, regularly repeating arrangement of atoms.
- Based on the diffraction pattern obtained from X-ray scattering off the periodic assembly of molecules or atoms in the crystal, the electron density can be reconstructed.

189

...X-Ray Crystallography...

- a tool used for determining the atomic and molecular structure of a crystal.
- The underlying principle is that the crystalline atoms cause a beam of X-rays to diffract into many specific directions.



- By measuring the angles and intensities of these diffracted beams, a crystallographer can produce a 3D picture of the density of electrons within the crystal.

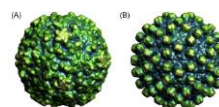
190

...X-Ray Crystallography...

- From this electron density image, the mean positions of the atoms in the crystal can be determined, as well as their chemical bonds, their disorder, and various other information.
 - The method revealed the structure and function of many biological molecules, including vitamins, drugs, proteins, and nucleic acids, such as DNA.
 - Note that the double helix structure of DNA discovered by James Watson and Francis Crick was revealed by X-ray crystallography.
- Recent advances in image reconstruction technology have made X-ray crystallography amenable to the structural analysis of much larger complexes, such as virus particles.

191

...X-Ray Crystallography



- Viral capsid structure obtained by X-ray crystallography.

(A) Poliovirus capsid with T=3 symmetry.

(B) Hepatitis B virus capsid with T=4

- The major shortcomings of X-ray crystallography
 - it is difficult to obtain a crystal of virus particles, which is a prerequisite for X-ray crystallography.
 - X-ray crystallography generally requires placing the samples in nonphysiological environments, which can occasionally lead to functionally irrelevant conformational changes.

192

Nuclear magnetic resonance...

- In NMR, no crystals are used in the process, and the protein remains in solution throughout the entire experiment.
- An intense and very linear magnetic field aligns the atomic nuclei of the protein into one of two spin states.
- A series of radio frequency pulses is used to perturb these by “flipping” some of the nuclei from one spin state to the other.
- As the total amount of energy absorbed is low, the protein remains undamaged and functions as normal.

193

...Nuclear magnetic resonance...

- Eventually, the “flipped” spin state of the nuclei realigns to the normal state, emitting a radio frequency pulse as it does so.
- The timing of this re-emission of energy is determined by the electronic environment in which the nucleus is embedded.
- A feature of this environment is the electrostatic shielding effects of the surrounding nuclei.
- The nuclei, in addition to the bonds linking them, can be identified by their spin decay properties.

194

...Nuclear magnetic resonance

- A problem with NMR methods is the size of the proteins that can be studied.
 - Using current techniques, this equates to a maximum of 200 amino acids.
 - This is low compared to the many hundreds of amino acids that can be studied using X-Ray Crystallography.
- The X-Ray Crystallography and NMR systems are complementary in many respects, as both determine, to a high accuracy, the coordinates of the atoms in protein structures.
 - If protein structures determined by X-Ray Crystallography and NMR are compared, they are generally consistent with each other and moreover are biologically plausible.
- This should give the researcher confidence when using them.

195

The Protein Databank...

- contains a large collection of previously determined biological structures.
 - For inclusion in the PDB, the spatial locations of the atoms have to be determined with sufficient accuracy to usefully describe protein structures.
- also includes experimental details of how the structure was determined, what publications and other databases to consult for more information on the structure, some “derived data” and details of any ill-defined regions.
 - While this information is meant to be included in the PDB, some of it may be missing, incomplete or incorrect for some database entries.

196

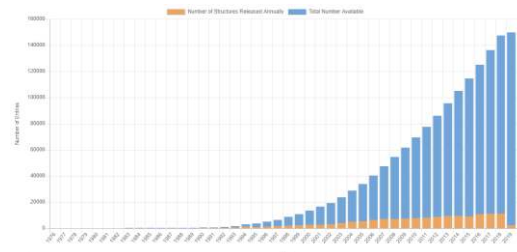
...The Protein Databank...

- one of the oldest bioscience data stores, dating back to 1971.
- It originally stored the 3D coordinates of protein structures as determined by the X-ray Crystallography method.
- Prior to the PDB, structures were typically published in journals, and many researchers re-entered the information manually into their computers so as to facilitate further manipulation of them.
- The original PDB data-file format adopted was a “flat” textual disk-file that was 80 columns wide.
- Today, the structures in the PDB are determined by either X-Ray Crystallography or NMR.
 - Often, many years of effort go into determining an individual structure.
- This is reflected in the growth of the number of entries in the PDB over some 40 years.

197

...The Protein Databank

- PDB Statistics:
 - Overall Growth of Released Structures Per Year



198

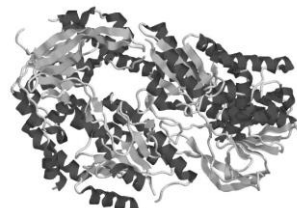
The PDB Data-file Formats

- available in one of two formats.
 - These formats are inter-convertible
- PDB flat file
 - The original, generic and highly unstructured PDB data-file format that is still widely used by researchers.
 - When biologists talk of “PDB files” or “PDB format”, they are referring to this data-file format.
 - The current standard format is the 2.3 version.
- mmCIF
 - The new PDB data-file format that is designed to offer a highly structured, modern replacement to the original PDB Flat File format.
 - The mmCIF format is often informally referred to as the “new PDB format”.

199

Example structures

- 1LQT
 - A modern, high-resolution “Oxidoreductase” enzyme structure produced using X-Ray Crystallographic techniques.



200

Example structures

- 1M7T
 - A modern protein structure of “Thioredoxin” produced using NMR.



201

Downloading PDB data-files

- PDB structure data-files can be downloaded from many web-site locations on the Internet.
- The RCSB web-site is always a good place to start:
 - <http://www.rcsb.org/pdb/>
- Alternatively, the EBI hosts a European mirror. Follow the links from:
 - <http://www.ebi.ac.uk/services/>
 to access the PDB from the EBI.

202

Accessing Data in PDB Entries

- There are some common sections to all PDB entries:
 - those concerned with
 - indexing,
 - bibliographic data,
 - notable features
 - 3D coordinates.
- Other sections are radically different from each other, as they depend on the experimental technique (X-Ray Crystallography or NMR) used to determine the structure.

203

Accessing Data in PDB Entries

- In a PDB data-file there is a left-right split (per line) and a top-bottom split (per data-file):
- Left-right
 - The left-most characters (a maximum of nine) on each line indicate what information is present on the right-hand side.
- Top-bottom
 - There is an upper HEADER section that contains the annotation about the structure (top) and a lower coordinates section that contains the 3D spatial locations of the atoms in the structure (bottom).

204

A short description of the most important fields in the PDB data-file

- **HEADER**
 - Contains a brief description of the structure, the date and the PDB ID code.
- **TITLE**
 - The title of the structure.
- **COMPND**
 - Brief details of the structure.
- **SOURCE**
 - Identifies which organism the structure came from.
- **KEYWDS**
 - Lists a set of useful words/phrases that describe the structure.
- **AUTHOR**
 - The scientists depositing the structure.
- **REVDAT**
 - The date of the last revision.

205

A short description of the most important fields in the PDB data-file

- **JRNL**
 - One or more literature references that describe the structure.
- **REMARK 1 through REMARK 999**
 - Details of the experimental methods used to determine the structure are contained in this subsection (see the example in the next section).
- **DBREF**
 - Cross links to other databases.
- **SEQRES**
 - The official amino acid sequence (protein, RNA or DNA) of the structure.
- **HELIX/SHEET**
 - Details of the regions of secondary structure found in the protein.
- **ATOM/HETATM**
 - The 3D spatial coordinates of particular atoms in the protein structure or other molecules such as water or co-factors.

206

Accessing PDB Annotation Data

- There are many examples of parsing data from the **HEADER** section of PDB data-files, all of which involve pattern matching.
 - Perl is exceptionally good at this.
- Two representative examples exploring
 - the relationship between the **resolution** of a structure and its **Free R value**, both of which are measures of the quality of the X-Ray Crystallographic structures.
 - the Free R value measures the agreement between the model and the observed x-ray reflection data.
 - The lower the Free R Value, the better the fit between the model and the observed data.
 - the database cross-referencing section used to link to other databases.

207

Free R and resolution...

- The **REMARK** tag, type 2 subsection stores **resolution**, whereas the **Free R value** is quoted in **REMARK** tag, type 3.
 - Here's a small extract from the 1LQT entry:


```
REMARK 2
REMARK 2 RESOLUTION, 1.05 ANGSTROMS.
```
- In NMR structures, **REMARK** tag, type 2 and type 3 are present, but the data in them is "NOT APPLICABLE" for **REMARK** tag, type 2 and "NULL" or free text for **REMARK** tag, type 3.
 - Extract from the 1M7T structure's **HEADER**:


```
REMARK 215 NMR STUDY
REMARK 215 THE COORDINATES IN THIS ENTRY WERE GENERATED FROM SOLUTION
REMARK 215 NMR DATA. PROTEIN DATA BANK CONVENTIONS REQUIRE THAT
REMARK 215 CRYSTI AND SCALE RECORDS BE INCLUDED, BUT THE VALUES ON
REMARK 215 THESE RECORDS ARE MEANINGLESS.
```

208

...Free R and resolution

- **Structural Refinement** is the process of iteratively fitting the model structure into the electron density map, and details of this refinement are stored in **REMARK** tag, type 3.
- Here is an extract :

```
.
.
REMARK 3 FIT TO DATA USED IN REFINEMENT.
REMARK 3 CROSS-VALIDATION METHOD : THROUGHOUT
REMARK 3 FREE R VALUE TEST SET SELECTION : RANDOM
REMARK 3 R VALUE (WORKING + TEST SET) : 0.134
REMARK 3 R VALUE (WORKING SET) : 0.134
REMARK 3 FREE R VALUE : 0.153
REMARK 3 FREE R VALUE TEST SET SIZE (%) : NULL
REMARK 3 FREE R VALUE TEST SET COUNT : 2200
```

209

A Perl program extracts the resolution and Free R Value from any PDB data-files

```
#!/usr/bin/perl -w
# free_res - Designed to extract the 'Free R Value' and 'Resolution'
# quantities from 'PDB data-files' containing structures
# produced by 'Diffraction'.
use strict;
my $PDB_Path = shift;
opendir ( INPUT_DIR, "$PDB_Path" )
    or die "Error: Cannot read from mmCIF directory: '$PDB_Path'\n";
my @PDB_dir = readdir INPUT_DIR;
close INPUT_DIR;
my @PDB_Files = grep /^pdb/, @PDB_dir;
foreach my $Current_PDB_File ( @PDB_Files )
{
    my $Free_R;
    my $Resolution;
    open ( PDB_FILE, "$PDB_Path/$Current_PDB_File" )
        or die "Cannot open PDB File: '$Current_PDB_File'\n";
```

210

A Perl program extracts the resolution and Free R Value from any PDB data-files

```
while ( <PDB_FILE> )
{
    if ( /^EXPDTA / and !/DIFFRACTION/ )
    {
        last;
    }
    if ( /^REMARK 2 RESOLUTION/ )
    {
        ( undef, undef, undef, $Resolution ) = split ( " ", $_ );
    }
    if ( /^REMARK 3 FREE R VALUE / )
    {
        $Free_R = substr ( $_, 47, 6 );
        $Free_R =~ s/ //g;
    }
}
```

211

A Perl program extracts the resolution and Free R Value from any PDB data-files

```
if ( $Free_R =~ /NULL/ or $Resolution eq "" )
{
    last;
}
else
{
    printf ( "%7s %4.2f %7.3f\n", $Current_PDB_File,
        $Resolution, $Free_R );
    last;
}
}
close ( PDB_FILE );
}
```

212

Non-redundant Datasets

- There may be many reasons for redundancy in a dataset.
 - **Scientific**
 - It is often advantageous to study molecules with similar structures.
 - This is a classic scientific investigative methodology: change a small part, then identify the change in structure or function to form hypotheses about the reasons for the change.
 - Consequently, researchers are encouraged to study similar molecules to those studied previously.
 - **Technological limitations**
 - In X-Ray Crystallography, it is easier to obtain the structure of a molecule that is similar to one that is already known, as molecules with similar conformations are likely to have similar crystallisation conditions.
 - This, conveniently, allows two of the most difficult aspects of using X-Ray Crystallography to be dealt with.

213

Reduction of redundancy

- There are two reasons for supporting the reduction of a database:
 - **Conceptually, to remove bias within the database.**
 - The statistical analysis based upon the non-redundant dataset will be more representative of all the items in the database, rather than just the largest dominant group.
 - **As a practical measure, to reduce the computational requirements caused by analysing examples that are unnecessary.**
 - For example, the PDB-Select structural non-redundant dataset contains approximately 1600 protein structures, whereas the entire PDB contained approximately 18,000.

214

Database cross references...

- The DBREF subsection gives a list of cross references to other Bioinformatics databases.
 - This makes it easier for researchers to integrate biological datasets.
- The second value on the DBREF line is the PDB identifier.
 - By examining this value, researchers and automatic parsing programs can tell to which structure the entry belongs.
- Example DBREF lines :


```
DBREF 1LQT A 1 456 GB 13882996 AAK47528 1 456
DBREF 1LQT B 1 456 GB 13882996 AAK47528 1 456

DBREF 1AFI 1 72 SWS P04129 MERP_SHIFL 20 91

DBREF 1M7T A 1 66 SWS P10599 THIO_HUMAN 0 65
DBREF 1M7T A 67 106 SWS P00274 THIO_ECOLI 68 107
```

215

...Database cross references...

- The PDB publishes a table of database names and their associated, abbreviated codes.

Database Name	Database Code
BioMagResBank	BMRB
BLOCKS	BLOCKS
European Molecular Biology Laboratory	EMBL
GenBank	GB
Genome Data Base	GD
Nucleic Acid Database	NDB
PROSITE	PROSIT
Protein Data Bank	PDB
Protein Identification Resource	PIR
SWISS-PROT	SWS
TREMBL	TREMBL

216

...Database cross references

- The DBREF lines identify the following fields, working from left to right:
 - PDB ID code.
 - Chain identifier (if needed).
 - The start of the sequence.
 - Insertion code.
 - End of the sequence.
 - The external database to which the cross reference refers.
 - The external database accession code.
 - The database external accession name.
 - The start, insertion and end of the sequence in the external database.

217

Coordinates section

- The coordinate data for the locations of atoms in the macromolecular structure is straightforward, especially when compared to the annotation contained in the HEADER section of the PDB data-file.
 - The coordinates are presented as points in space, the atoms they represent are actually in motion.
 - In crystallographic structures, isotropic B-factors, commonly referred to as "Temperature Factors", give us an idea of the vibration of the molecule.
 - For very high-resolution structures, Anisotropic Temperature Factors may be included in the ANISOU lines.
 - These provide an idea of the vibration of the molecule in the directions of the coordinate axes.
 - In NMR structures, the variation in position of a particular atom between different models in the ensemble can be used as a similar measure of motion or as an indication of the error between the minimisation models.
- Here is an example from 1M7T:

```
REMARK 210
REMARK 210 BEST REPRESENTATIVE CONFORMER IN THIS ENSEMBLE : 21
REMARK 210
```

218

Data section

- Referring to the 1LQT x-ray structure, an extract of lines from the coordinate section looks like this:

```
ATOM      1  N   ARG A  2      26.318   8.010  39.090   1.00  20.71   N
ANISOU    1  N   ARG A  2      2040  3071  2755   114   -339  -393   N
ATOM      2  CA  ARG A  2      25.150  -8.702  38.505   1.00  18.85   C
ANISOU    2  CA  ARG A  2      2029  2877  2455    67   -321  -209   C
ATOM      3  C   ARG A  2      24.846   8.176  37.123   1.00  17.23   C
ANISOU    3  C   ARG A  2      1689  2479  2479   143   -282  -258   C
ATOM      4  O   ARG A  2      25.151  -7.048  36.775   1.00  18.14   O
.
.
.
TER       7215      GLY A 456
ATOM      7216  N   ARG B  2      -19.423  25.709   6.980   1.00  21.57   N
ANISOU    7216  N   ARG B  2      2476  3022  2707   -165   -370    95   N
ATOM      7217  CA  ARG B  2      -18.718  26.510   8.024   1.00  19.01   C
ANISOU    7217  CA  ARG B  2      2127  2672  2424    -63   -285   91   C
ATOM      7218  C   ARG B  2      -17.250  26.207   8.002   1.00  17.22   C
ANISOU    7218  C   ARG B  2      1955  2392  2196    -91   -299  121   C
ATOM      7219  O   ARG B  2      -16.851  25.158   7.535   1.00  18.15   O
```

219

Data section

```
.
.
.
TER       14289      GLY B 456
HETATM 14290  C   ACT 1866      -13.075   1.733  10.218   1.00  27.25   C
ANISOU 14290  C   ACT 1866      3493  3560  3299   -39   -36  -44   C
.
.
.
CONECT14290142911429214293
CONECT1429114290
CONECT1429214290
TER
.
.
.
CONECT1469014663
MASTER      389    0  15  46  38    0  0  620280    2  401  72
END
```

220

Data section

- For the 1M7T NMR structure, an extract of lines from the coordinate section looks like this:

```
MODEL      1
ATOM      1  H1  MET A  1      3.110  -4.682  -3.025   1.00  0.00   N
ATOM      2  CA  MET A  1      2.546  -3.712  -2.053   1.00  0.00   C
ATOM      3  C   MET A  1      1.134  -3.295  -2.450   1.00  0.00   C
ATOM      4  O   MET A  1      0.882  -2.130  -2.758   1.00  0.00   O
ATOM      5  CB  MET A  1      3.466  -2.491  -2.002   1.00  0.00   C
ATOM      6  CG  MET A  1      3.781  -1.903  -3.370   1.00  0.00   C
ATOM      7  SD  MET A  1      4.256  -0.166  -3.285   1.00  0.00   S
ATOM      8  CE  MET A  1      6.004  -0.307  -2.920   1.00  0.00   C
ATOM      9  1H  MET A  1      2.906  -4.327  -3.980   1.00  0.00   H
ATOM     10  2H  MET A  1      2.650  -5.601  -2.859   1.00  0.00   H
ATOM     11  3H  MET A  1      4.134  -4.758  -2.858   1.00  0.00   H
ATOM     12  HA  MET A  1      2.517  -4.178  -2.079   1.00  0.00   H
ATOM     13  1HB  MET A  1      2.996  -1.724  -1.405   1.00  0.00   H
ATOM     14  2HB  MET A  1      4.397  -2.778  -1.536   1.00  0.00   H
ATOM     15  1HG  MET A  1      4.596  -2.461  -3.807   1.00  0.00   H
ATOM     16  2HG  MET A  1      2.907  -1.993  -3.998   1.00  0.00   H
ATOM     17  1HE  MET A  1      6.344  -1.302  -3.167   1.00  0.00   H
ATOM     18  2HE  MET A  1      6.169   0.120  -1.869   1.00  0.00   H
ATOM     19  3HE  MET A  1      6.553   0.416  -3.505   1.00  0.00   H
ATOM     20  N   VAL A  2      0.215  -4.256  -2.446   1.00  0.00   N
```

221

Data section

```
.
.
.
TER       1659      VAL A 107
ENDMDL
MODEL      2
ATOM      1  N   MET A  1      2.750  -6.779  -1.627   1.00  0.00   N
ATOM      2  CA  MET A  1      2.487  -5.475  -2.290   1.00  0.00   C
.
.
.
TER       1660      VAL A 107
ENDMDL
```

222

Data section

- In each ATOM line, the fields are as follows:

COLUMNS	DATA TYPE	FIELD	DEFINITION
1 - 6	Record name	"ATOM(s)"	
7 - 11	Integer	serial	Atom serial number.
13 - 16	Atom	name	Atom name.
	Character	altLoc	Alternate location indicator.
18 - 20	Residue name	resName	Residue name.
22	Character	chainID	Chain identifier.
23 - 26	Integer	resSeq	Residue sequence number.
27	AChar	iCode	Code for insertion of residues.
31 - 38	Real(8,3)	x	Orthogonal coordinates for X in Angstroms.
39 - 46	Real(8,3)	y	Orthogonal coordinates for Y in

223

Extracting 3D co-ordinate data

- The technique involves extracting the three substrings from each line that contains the X, Y and Z coordinates.
- Assuming the data is in \$_, three invocations of Perl's substr subroutine do the trick:

```
my ( $X, $Y, $Z ) = ( substr( $_, 30, 8 ),  
                      substr( $_, 38, 8 ),  
                      substr( $_, 46, 8 ) );
```

224

The simple_coord_extract program

```
#!/usr/bin/perl -w  
# simple_coord_extract <PDB File> - Demonstrates the extraction of  
# C-Alpha co-ordinates from a PDB  
# data-file.  
  
use strict;  
  
while ( <> )  
{  
    if ( /^ATOM/ && substr( $_, 13, 4 ) eq "CA " )  
    {  
        my ( $X, $Y, $Z ) = ( substr( $_, 30, 8 ),  
                              substr( $_, 38, 8 ),  
                              substr( $_, 46, 8 ) );  
  
        $X =~ s/ //g;  
        $Y =~ s/ //g;  
        $Z =~ s/ //g;  
  
        print "X, Y & Z: $X, $Y, $Z\n";  
    }  
}
```

225

Results from simple_coord_extract ...

```
X, Y & Z: 25.150, -8.702, 38.505  
X, Y & Z: 23.675, -8.497, 35.069  
X, Y & Z: 20.747, -6.252, 34.332  
X, Y & Z: 17.545, -8.297, 34.292  
X, Y & Z: 15.182, -7.484, 31.454  
X, Y & Z: 11.736, -8.952, 30.942  
X, Y & Z: 10.261, -9.014, 27.451  
X, Y & Z: 6.507, -9.548, 27.173
```

226

Introducing Databases...

- Many modern computer systems store vast amounts of structured data.
- Typically, this data is held in a [database](#) system.
- Database
 - a collection of one or more [related tables](#).
- Table
 - a collection of one or more [rows](#) of data.
 - The rows of data are arranged in columns, with each intersection of a row and column containing a data item.
- Row
 - a collection of one or more data items, arranged in columns.
 - Within a row, the columns conform to a structure.

227

...Introducing Databases

- For example,
 - if the first column in a row holds a date, then every first column in every row must also hold a date.
 - if the second column holds a name, then every second column must also hold a name, and so on.
- The following data corresponds to the structure, in that there are two columns, the first holding a date, the second holding a name:

1960-12-21	P. Barry
1954-6-14	M. Moorhouse

228

Structured data

- Each column can be given a descriptive name.

```
-----
Discovery_Date      Scientist
-----
1960-12-21          P. Barry
1954-6-14           M. Moorhouse
1970-3-4            J. Blow
2001-12-27          J. Doe
```

- In addition, the structure requires that each data item held in a column be of a specific type.

```
-----
Column name        Type restriction
-----
Discovery_Date      a valid Date
Scientist            a String no longer than 64 characters
```

- This type information generally goes by one of two names: **metadata** or **schema**.

229

Relating tables...

- Extending the Discoveries table to include details of the discovery, an additional column is needed to hold the data

```
-----
Discovery_Date      Scientist      Discovery
-----
1960-12-21          P. Barry      Flying car
1954-6-14           M. Moorhouse  Telepathic sunglasses
1970-3-4            J. Blow      Self cleaning child
2001-12-27          J. Doe       Time travel
```

- The inclusion of this new column requires an update to the structure of the table

```
-----
Column name        Type restriction
-----
Discovery_Date      a valid Date
Scientist            a String no longer than 64 characters
Discovery            a String no longer than 128 characters
```

230

...Relating tables

```
-----
Column name        Type restriction
-----
Discovery_Date      a valid Date
Scientist            a String no longer than 64 characters
Discovery            a String no longer than 128 characters
Date_of_birth       a valid Date
Telephone_number     a String no longer than 16 characters
```

```
-----
Discovery_Date      Scientist      Discovery      Date_of_birth      Telephone_number
-----
1960-12-21          P. Barry      Flying car      1966-11-18          353-503-555-91910
1954-6-14           M. Moorhouse  Telepathic sunglasses  1970-3-24          00-44-81-555-3232
1970-3-4            J. Blow      Self cleaning child  1955-8-17          555-2837
2001-12-27          J. Doe       Time travel      1962-12-1          -
1974-3-17           M. Moorhouse  Memory swapping toupee  1970-3-24          00-44-81-555-3232
1999-12-31          M. Moorhouse  Twenty six hour clock  1958-7-12          416-555-2000
```

231

The problem with single-table databases

- Although the above table structure solves the problem of uniquely identifying each scientist, it introduces some other problems:
 - If a scientist is responsible for a large number of discoveries, their identification information has to be entered into every row of data that refers to them.
 - This is time-consuming and wasteful.
 - Every time identification information is added to a row for a particular scientist, it has to be entered in exactly the same way as the identification information added already.
 - Despite the best of efforts, this level of accuracy is often difficult to achieve.
 - If a scientist changes any identification information, every row in the table that refers to the scientist's discoveries has to be changed.
 - This is drudgery.

232

Solving the one table problem...

- The problems described in the previous section are solved by breaking the all-in-one Discoveries table into two tables.
- Here is a new structure for Discoveries:

```
-----
Column name        Type restriction
-----
Discovery_Date      a valid Date
Scientist_ID        a String no longer than 8 characters
Discovery            a String no longer than 128 characters
```

```
-----
Column name        Type restriction
-----
Scientist_ID        a String no longer than 8 characters
Scientist            a String no longer than 64 characters
Date_of_birth       a valid Date
Address              a String no longer than 256 characters
Telephone_number     a String no longer than 16 characters
```

233

...Solving the one table problem

```
-----
Discovery_Date      Scientist_ID      Discovery
-----
1954-6-14           MM               Telepathic sunglasses
1960-12-21          PB               Flying car
1969-8-1            PB               A cure for bad jokes
1970-3-4            JB               Self cleaning child
1974-3-17           MM               Memory swapping toupee
1999-12-31          MM2              Twenty six hour clock
2001-12-27          JD               Time travel
```

```
-----
Scientist_ID      Scientist      Date_of_birth      Address              Telephone_number
-----
JB                J. Blow       1955-8-17          Belfast, NI          555-2837
JD                J. Doe        1962-12-1          Sydney, AUS          -
MM                M. Moorhouse  1970-3-24          England, UK          00-44-81-555-3232
MM2              M. Moorhouse  1958-7-12          Toronto, CA          416-555-2000
PB                P. Barry      1966-11-18          Carlow, IRL          353-503-555-91910
```

234

Relational Databases

- Relating data in one table to that in another forms the basis of modern database theory.
 - It also explains why so many modern database technologies are referred to as **Relational Database Management Systems (RDBMS)**.
- When a collection of tables is designed to relate to each other they are collectively referred to as a **database**.
- It is usually a requirement to give the database a descriptive name.

235

Database system: a definition

- A database system is a computer program (or group of programs)
 - that provides a mechanism to define and manipulate
 - one or more databases
- A database system
 - allows databases, tables and columns to be created and named, and structures to be defined.
 - provides mechanisms to add, remove, update and interact with the data in the database.
- Data stored in tables can be searched, sorted, sliced, diced and cross-referenced.
- Reports can be generated, and calculations can be performed.

236

Available Database Systems

- Personal database systems:
 - Designed to run on PCs
 - Access, Paradox, FileMaker, dBase
- Enterprise database systems:
 - Designed to support efficient storage and retrieval of vast amount of data
 - Interbase, Ingres, SQL Server, Informix, DB2, Oracle
- Open source database systems:
 - Free!!! (Linux!!!)
 - PostgreSQL, MySQL

237

Choosing Database System

- Which type of database system is chosen depends on a number of factors, including (but not limited to):
 - The amount of data to be stored in the database.
 - Whether the data supports a small personal project or a large collaborative one.
 - How much funds (if any) are available towards the purchase of a database system.

238

SQL: The Language of Databases

- Defining data with SQL (structured query language)
- SQL provides two facilities:
 - A database definition Language (DDL)
 - provides a mechanism whereby databases can be created
 - A Data Manipulation Language (DML)
 - provides a mechanism to work with data in tables

239

Installing a database system

- MySQL is a modern, capable and SQL-enabled database system.
- It is Open Source and freely available for download from the MySQL web-site:
<http://www.mysql.com>
- It comes as a standard, installable component of most Linux distributions
- The following commands switch on MySQL on RedHat and RedHat-like Linux distributions
`chkconfig --add mysqld`
`chkconfig mysqld on`
- If the first `chkconfig` command produces an error messages like this:
`error reading information on service mysqld: No such file or directory`
this means that MySQL is not installed and the second command will also fail.

240

Installing a database system

- Once MySQL is installed, it needs to be configured.
- The first requirement is to assign a password to the MySQL superuser, known as “root”.
- The `mysqladmin` program does this, as follows:

```
mysqladmin -u root password 'passwordhere'
```

- It is now possible to securely access the MySQL Monitor command-line utility with the following command, providing the correct password when prompted:

```
mysql -u root -p
```

241

A Database Case Study: MER

- A small collection of SWISS-PROT and EMBL entries are taken from the Mer Operon, a bacterial gene cluster that is found in many bacteria for the detoxification of Mercury Hg²⁺ ions.
- These provide the raw data to a database, which is called MER.
- The MER database contains four tables:
 - `proteins` – A table of protein structure details, extracted from a collection of SWISS-PROT entries.
 - `dnas` – A table of DNA sequence details, extracted from a collection of EMBL entries.
 - `crossrefs` – A table that links the extracted protein structures to the extracted DNA sequences.
 - `citations` – A table of literature citations extracted from both the SWISS-PROT and EMBL DNA entries.

242

A Database Case Study: MER

- Once the raw data is in the database, SQL can be used to answer questions about the data.
- For instance:
 - How many protein structures in the database are longer than 200 amino acids in length?
 - How many DNA sequences in the database are longer than 4000 bases in length?
 - What's the largest DNA sequence in the database?
 - Which protein structures are cross-referenced with which DNA sequences?
 - Which literature citations reference the results from the previous question?

243

Creating the MER database...

- SQL queries can be entered directly at the MySQL Monitor prompt.

```
mysql> create database MER;
Query OK, 1 row affected (0.36 sec)
mysql> show databases;
```

```
+-----+
| Databases |
+-----+
| MER      |
| test     |
| mysql    |
+-----+
3 rows in set (0.00 sec)
```

- A list of databases is returned by MySQL.
- There are three identified databases:
 - `MER` – The just-created database that will store details on the extracted protein structures, DNA sequences, cross references and literature citations.
 - `test` – A small test database that is used by MySQL and other technologies to test the integrity of the MySQL installation.
 - `mysql` – The database that stores the internal “system information” used by the MySQL database system.

244

...Creating the MER database

- It is possible to use the MySQL superuser to create tables within the MER database.
- However, it is better practice to create a user within the database system to have authority over the database, and then perform all operations on the MER database as this user.
- The queries to do this are entered at the MySQL Monitor prompt.
- Here are the queries and the messages returned:

```
mysql> use mysql;
Database changed
mysql> grant all on MER.* to bbp identified by 'passwordhere';
Query OK, 0 rows affected (0.00 sec)
mysql> quit
Bye
```
- The first query tells MySQL that any subsequent queries are to be applied to the named database, which in this case is the `mysql` database.
- The second query does three things:
 - creates a new MySQL user called “bbp”.
 - assigns a password with the value of “passwordhere” to user “bbp”.
 - grants every available privilege relating to the MER database to “bbp”.

245

Adding tables to the MER database

create table proteins

```
(
  accession_number  varchar (6)   not null,
  code              varchar (4)   not null,
  species           varchar (5)   not null,
  last_date         date          not null,
  description       text          not null,
  sequence_header   varchar (75)  not null,
  sequence_length   int           not null,
  sequence_data     text          not null
)
```

```
$ mysql -u bbp -p MER < create_proteins.sql
```

246

Databases and Perl

- Why Program Databases?
 - Customised output handling
 - Programs can be written to post-process the results of any SQL query and display them in any number of preferred formats.
 - Customised input handling
 - Users of customised input handling programs do not need to know anything about SQL – all they need to know and understand is their data.
 - Extending SQL
 - Some tasks that are difficult or impossible to do with SQL can be programmed more easily.
 - Integrating MySQL into custom applications
 - Having the power of MySQL as a component of an application can be very powerful.

247

Perl Database Technologies

- A number of third-party CPAN modules provide access to MySQL from within a Perl program.
 - One such module is Net::MySQL by Hiroyuki Oyama, which provides a stable programming interface to MySQL functionality.
- In fact, nearly every database system provides a specific technology for programmers to use when programming their particular database.
 - This technology is referred to as an API, an application programming interface.
- Unfortunately, the effort expended in learning how to use Net::MySQL is of little use when a program has to be written to interface with Oracle or Sybase

248

Perl Database Technologies

- The DBI module provides a database independent interface for Perl.
 - By providing a generalised API, programmers can program at a “higher level” than the API provided by the database system, in effect insulating programs from changes to the database system.
- To connect the high-level DBI technology to a particular database system, a special driver converts the general DBI API into the database system-specific API.
 - These drivers are implemented as CPAN modules.

249

Preparing Perl

DBI and DBD::mysql modules need to be installed

```
$ man DBI
$ man DBD::mysql
```

```
$ find `perl -Te 'print "@INC"'` -name '*.pm' -print | grep 'DBI.pm'
$ find `perl -Te 'print "@INC"'` -name '*.pm' -print | grep 'mysql.pm'
```

```
$ locate DBI.pm
$ locate mysql.pm
```

DBI (previously called DBperl) is a database independent interface module for Perl.
DBD: Data Base Description

250

Checking the DBI installation

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

# check_drivers - check which drivers are installed with DBI.

use strict;

use DBI;

my @drivers = DBI->available_drivers;

foreach my $driver ( @drivers )
{
    print "Driver: $driver installed.\n";
}
```

251

Programming Databases With DBI

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

```
# show_tables - list the tables within the MER database.
# Uses "DBI::dump_results" to display results.
```

```
use strict;
use DBI qw( :utils );
use constant DATABASE => "DBI:mysql:MER";
use constant DB_USER => "bbp";
use constant DB_PASS => "passwordhere";
my $dbh = DBI->connect( DATABASE, DB_USER, DB_PASS )
    or die "Connect failed: ", $DBI::errstr, "\n";

my $sql = "show tables";
my $sth = $dbh->prepare( $sql );
$sth->execute;
print dump_results( $sth ), "\n";
$sth->finish;
$dbh->disconnect;
```

- DATABASE
 - Identifies the data source to use
- DB_USER
 - Identifies the username to use when connecting to the data source
- DB_PASS
 - Identifies the password to use when authenticating to the data source

252

The Sequence Retrieval System...

- Sequence Retrieval System (SRS) is a web-based database integration system that allows for the querying of data contained in a multitude of databases, all through a single user interface.
- This makes the individual databases appear as if they are really one big relational database, organised with different subsections:
 - one called SWISS-PROT,
 - one called EMBL,
 - one called PDB,
 - ...

253

...The Sequence Retrieval System...

- SRS makes it very easy to query the entire data set, using common search terms that work across all the different databases, regardless of what they are.
- Everything contained within the SRS is “tied together” by the web-based interface.
- Figure in the next slide is the database selection page from the EBI’s SRS web-site, which can be navigated to from the following Internet address:
 - <http://srs.ebi.ac.uk>
 - SRS is a trademark and the intellectual property of Lion Bioscience

254

EBI's SRS Database Selection Page

[illegible]

255

...The Sequence Retrieval System

- SRS is important for two reasons:
 - It is a useful and convenient service that every Bioinformatician should know about.
 - It is an excellent example of what can be created when the World Wide Web, databases and programming languages are combined.
- Warning:
 - Don't create a new data format unless absolutely necessary.
 - Use an existing format whenever possible

256

Web Technologies

- The WWW was invented in 1991 by Tim Berners-Lee.
- The ability to publish data and applications on the Internet, in the form of custom web pages, is now considered an essential skill in many disciplines, including Biology.
- The development infrastructure of the World Wide Web (WWW) is well established and well understood.
- There is a standard set of infrastructural components (as suggested by Tim Berners-Lee):

257

The Web Development Infrastructure...

- The web server
 - a program that when loaded onto a computer system, provides for the publication of data and applications
 - often referred to collectively as **content**
 - Examples (apache, Jigsaw, and Microsoft's IIS)
- The web client
 - a program that can request content from a web server and display the content within a graphical window, providing a mechanism whereby user can interact with the contents
 - The common name for the web client is **web browser**
 - Examples (Chrome, Mozilla, MS Internet Explorer, KDE Konqueror, Opera, Lynx, ...)

258

...The Web Development Infrastructure

- Transport protocol
 - the “language” that the web server and web client use when communicating with each other.
 - Think of this as the set of rules and regulations to which the client and server must adhere.
 - The transport protocol employed by the WWW is called [HyperText Transport Protocol \(HTTP\)](#)
- The content
 - the data and applications published by the web server
 - this is textual data formatted to conform to one of the [HyperText Mark-up Language standards \(HTML\)](#)
 - HTML can be enhanced with embedded graphics.
 - Data published in the form of HTML is often referred to as [HTML pages](#) or [web pages](#)

259

Additional components...

- Client-side programming
 - a technology used to program the web client, providing a way to enhance the user’s interactive experience.
 - Java applets, JavaScript, Macromedia Flash, ...
- Server-side programming
 - a technology used to program the web server, providing a mechanism to extend the services provided by the web server.
 - Java Servlets, JSP, Python, ASP, PHP, Perl, ...
- Backend database technology
 - a place to store the data to be published, which is accessed by the server-side programming technology.
 - MySQL, ...

260

...Additional components

- The acronym [LAMP](#) is used to describe the favoured WWW development infrastructure of many programmers.
 - The letters that form the acronym are taken from the words [Linux](#), [Apache](#), [MySQL](#) and [Perl/Python/PHP](#).
 - O’Reilly & Associates provides an excellent [LAMP](#) web-site, available on-line at
 - <http://www.onlamp.com>.
- The additional components turn the standard web development infrastructure into a dynamic and powerful application development environment.
- One of the reasons the WWW is so popular is the fact that creating content is so straightforward
 - Adding a programming language into the mix allows even more to be accomplished

261

Creating Content For The WWW...

- There are a number of techniques employed to create HTML:
 - [Creating content manually](#)
 - Any text editor can be used to create HTML, since HTML is mostly text.
 - Special tags within the text guide the web browser when it comes to displaying the web page on screen.
 - The tags are also textual and any text editor can produce them
 - [Advantages and disadvantages](#):
 - provides the maximum amount of flexibility as the creator has complete control over the process.
 - to know what’s going on behind the scenes, so learning HTML is highly recommended
 - can be time-consuming and tedious, as the creator of the page has to write the content as well as decide which tags to use and where

262

...Creating Content For The WWW...

- [Creating content visually](#)
 - Special-purpose editors can create HTML pages visually, displaying the web page as it will appear in the web browser as it is edited.
 - Netscape Composer, Microsoft FrontPage and Macromedia Dreamweaver,
 - [Advantages and disadvantages](#):
 - no need to know anything about HTML.
 - The editor adds the required tags to the text that’s entered by the user
 - unnecessary tags added,
 - HTML pages are larger

263

...Creating Content For The WWW

- [Creating content dynamically](#)
 - Since HTML is text, it is also possible to create HTML from a program.
 - [Advantages and disadvantages](#):
 - HTML pages produced in this way can sometimes be useful when combined with a web server that allows for server-side programming of a backend database
 - needs a web page creator to write a program to produce even the simplest of pages
- A useful web-site:
 - <http://www.htmlprimer.com>

264

A Simple HTML Page...

- Content of [simple-m.html](#) created manually

```
<HTML>
<HEAD>
<TITLE>A Simple HTML Page</TITLE>
</HEAD>
<BODY>
This is as simple a web page as there is.
</BODY>
</HTML>
```

265

...A Simple HTML Page

- Content of [simple-k.html](#) created visually
 - by using KompoZer
 - <http://www.kompozer.net/>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta content="text/html; charset=ISO-8859-1" http-
equiv="content-type">
<title> A Simple HTML Page</title>
</head>
<body>
This is as simple a web page as there is.
</body>
</html>
```

266

Producing HTML...

- Producing HTML with a Perl program using a HERE document:

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

# produce_simple - produces the "simple.html" web page using
# a HERE document.

use strict;

print <<WEBPAGE;
<HTML>
<HEAD>
<TITLE>A Simple HTML Page</TITLE>
</HEAD>
<BODY>
This is as simple a web page as there is.
</BODY>
</HTML>
WEBPAGE
```

267

...Producing HTML...

- HTML file produced by the program:

```
<HTML>
<HEAD>
<TITLE>A Simple HTML Page</TITLE>
</HEAD>
<BODY>
This is as simple a web page as there is.
</BODY>
</HTML>
```

268

...Producing HTML

- Another version of HTML generation
 - written to use Perl's standard CGI module

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

# produce_simpleCGI - produces the "simple.html" web page using
# Perl's standard CGI module.

use strict;

use CGI qw( :standard );

print start_html( 'A Simple HTML Page' ),
      "This is as simple a web page as there is.",
      end_html;
```

269

- The CGI module is designed to make the production of HTML as convenient as possible.
- [start_html](#) subroutine produces the tags that appear at the start of the web page.
- [end_html](#) subroutine produces the following HTML, representing tags that conclude a web page:
[</body></html>](#)

270

Results from produce_simpleCGI

- HTML file produced by the program:

```
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">
<head>
<title>A Simple HTML Page</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
This is as simple a web page as there is.
</body>
</html>
```

Extra stuff at the start is optional. Extra tags tell the web browser exactly which version of HTML the web page conforms to. The CGI module includes these tags for web browser to optimise its behaviour to the version of HTML identified

271

Static creation of WWW content

- **simple.html** web page is **static**
- If the web page is put on a web server it always appear in exactly the same way every time it is accessed.
 - It is **static**, and remains unchanged until someone takes the time to change it.
- It rarely makes sense to create such a web page with a program unless you have a special requirement.
 - Create static web pages either manually or visually

272

The dynamic creation of WWW content

- When the web page includes content that is not static, it is referred to as **dynamic** web page.
 - For example a page including current date and time
- It is not possible to creat a web page either manually or visually that includes dynamic content, and
 - this is where **server side programming technologies** come into their own.

273

The dynamic creation of WWW content

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

# whattimeisit - create a dynamic web page that includes the
# current date/time.

use strict;

use CGI qw( :standard );

print start_html( 'What Date and Time Is It?' ),
      "The current date/time is: ", scalar localtime,
      end_html;
```

274

Results from whattimeisit

```
>perl -wT whattime.pl
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-
US"><head><title>What Date and Time Is It?</title>
</head><body>The current date/time is: Thu Mar 29 18:56:17
2007</body></html>>Exit code: 0
```

- This web page, if served up by a web server, changes with each serving, as it is **dynamic**.

275

And some time later

```
>perl -wT whattime.pl
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US"
xml:lang="en-US"><head><title>What Date and Time Is It?</title>
</head><body>The current date/time is: Thu Mar 29 18:59:59
2007</body></html>>Exit code: 0
```

276

- Note that use of the “**T**” command-line option at the start of the program.
 - This switches on Perl’s **taint mode**,
 - which enables a set of special security checks on the behaviour of the program.
- If a server-side program does something that could potentially be exploited and, as a consequence, pose a security treat, Perl refuses to execute the program when **taint mode** is enabled.
- Always enable “**taint mode**” for server-side programs
- Test your web-site on localhost prior to deployment on the Internet

277

Sending Data To A Web Server...

- Switch on taint mode on the Perl command line
- Use CGI module, importing (at least) the standart set of subroutines
- Ensure the first print statement within the program is “**print header**”;
- Envelope any output sent to STDOUT with calls to the **start_html** and **end_html** subroutines
- Create a static web page to invoke the server-side program, providing input as necessary

278

...Sending Data To A Web Server

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

# The 'match_emb1CGI' program - check a sequence against the EMBL
#                               database entry stored in the
#                               emb1.data.out data-file on the
#                               web server.

use strict;

use CGI qw/:standard/;

print header;

open EMBLENTRY, "emb1.data.out"
or die "No data-file: have you executed prepare_emb1?";

my $sequence = <EMBLENTRY>;

close EMBLENTRY;
```

279

match_emb1CGI, cont.

```
print start_html( "The results of your search are in!");
print "Length of sequence is: <b>". length $sequence,
      "</b> characters.<p>";
print h3( "Here is the result of your search:" );

my $to_check = param( "shortsequence" );

$to_check = lc $to_check;

if ( $sequence =~ /$to_check/ )
{
    print "Found. The EMBL data extract contains: <b>$to_check</b>.";
}
else
{
    print "Sorry. No match found for: <b>$to_check</b>.";
}

print p, hr, p;
print "Press <b>Back</b> on your browser to try another search.";
print end_html;
```

280

A Search HTML Page

```
<HTML>
<HEAD>
<TITLE>Search the Sequence for a Match</TITLE>
</HEAD>
<BODY>
Please enter a sequence to match against:<p>
<FORM ACTION="/cgi-bin/match_emb1CGI">
<p>
<textarea name="shortsequence" rows="4" cols="60"></textarea>
</p>
<p>
<input type="reset" value="Clear">
<input type="submit" value="Try it!">
</p>
</FORM>
</BODY>
</HTML>
```

281

The “Search the Sequence for a Match” web page



282

Installing CGIs on a Web Server

```
$ su
$ cp mersearch.html /var/www/html
$ cp match_embICGI /var/www/cgi-bin
$ chmod +x /var/www/cgi-bin/match_embI
$ cp embl.data.out /var/www/cgi-bin
$ <Ctrl-D>
```

The "Results of your search are in!" web page



283

284

The "Sorry! Not Found" web page



285

Using a HERE document

```
print <<MERFORM;

Please enter another sequence to match against:<p>
<FORM ACTION="/cgi-bin/match_embICGIbetter">
<p>
<textarea name="shortsequence" rows="4" cols="60"></textarea>
</p>
<p>
<input type="reset" value="Clear">
<input type="submit" value="Try it!">
</p>
</FORM>
MERFORM
```

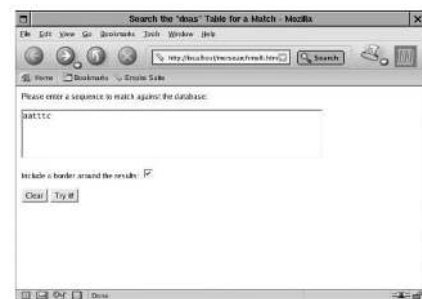
286

Better version: "Results of your search are in!" web page



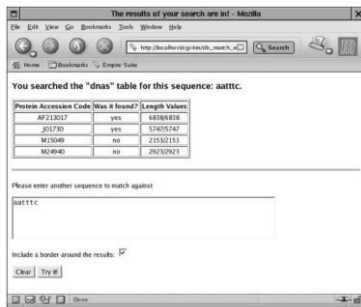
287

Searching all the entries in the dnas table



288

The ``results'' of the multiple search on the dnass table



Protein Accession Code	Was it found?	Length Value
AF213017	yes	48388488
201790	yes	57405740
M15040	no	22502151
M04040	no	29232923

Please enter another sequence to match against:

aatttc

Include a border around the results: ☒

Clear Try #

289

Installing DB Multi-Search

```
$ su
$ cp mersearchmulti.html /var/www/html
$ cp db_match_emb1CGI /var/www/cgi-bin
$ chmod +x /var/www/cgi-bin/db_match_emb1CGI
$ cp /home/barryp/DbUtilsMER.pm /var/www/cgi-bin
$ <Ctrl-D>
```

290

Web Automation

- Imagine you have 100 sequences to check.
- If it takes average 1 minutes to enter the sequence into text area, entering 100 sequences requires 100 minutes
- Why not automate it to save time
- Perl module **WWW::Mechanize** allows programmer to automate interactions with any web-site

Strategy to follow when automating interactions with any web page

- Load the web page of interest into a graphical browser
- View the HTML used to display the web page by selecting the **Page Source** option from browser's **View** menu
- Read the HTML and make a note of the names of the interface elements and form buttons that are of interest
- Write a Perl program that uses **WWW::Mechanize** to interact with the web page (based on **automatch**, if needed)
- Use an appropriate regular expression to extract the **interesting bits** from the results returned from the web server

291

292

The automatch program...

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

# The 'automatch' program - check a collection of sequences against
# the 'mersearchmulti.html' web page.

use strict;

use constant URL => "http://pblinux.itcarlow.ie/mersearchmulti.html";

use WWW::Mechanize;

my $browser = WWW::Mechanize->new;

while ( my $seq = <> )
{
    chomp( $seq );

    print "Now processing: '$seq'.\n";
```

293

...The automatch program

```
$browser->get( URL );
$browser->form( 1 );
$browser->field( "shortsequence", $seq );
$browser->submit;
if ( $browser->success )
{
    my $content = $browser->content;
    while ( $content =~
        m[<tr align="CENTER"
        /><td>(\w+?)</td><td>yes</td><td>]g )
    {
        print "Accession code: $1 matched '$seq'.\n";
    }
}
else
{
    print "Something went wrong: HTTP status code: ",
        $browser->status, "\n";
}
}
```

294

Running the automatch program

```
$ chmod +x automatch
$ ./automatch sequences.txt
```

Results from automatch

```
Now processing: 'attccgattaggcgta'.
Now processing: 'aattc'.
  Accession code: AF213017 matched 'aattc'.
  Accession code: J01730 matched 'aattc'.
  Accession code: M24940 matched 'aattc'.
Now processing: 'aatgggc'.
Now processing: 'aaattt'.
```

Results from automatch

```
Accession code: AF213017 matched 'aaattt'.
Accession code: J01730 matched 'aaattt'.
Accession code: M24940 matched 'aaattt'.
Now processing: 'acgatccgaagtagcaacc'.
Accession code: M15049 matched 'acgatccgaagtagcaacc'.
Now processing: 'gggcccaaa'.
Now processing: 'atcgatcg'.
Now processing: 'tcatgcacctgatgaacgtgcaaaaccacag'.
  Accession code: AF213017 matched 'tcatgcacctgatgaacgtgcaaaaccacag'.
.
.
Now processing: 'ccaaat'.
  Accession code: AF213017 matched 'ccaaat'.
  Accession code: J01730 matched 'ccaaat'.
  Accession code: M24940 matched 'ccaaat'.
```

295

296

Viewing the source of the mersearchmulti.html web page



297