

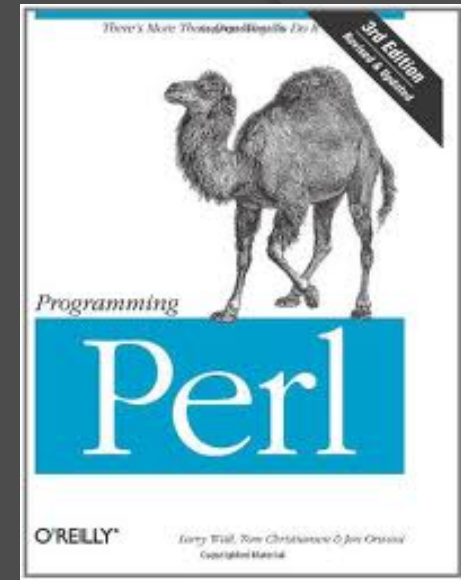
# PERL Bioinformatics

Nicholas E. Navin, Ph.D.  
Department of Genetics  
Department of Bioinformatics

TA: Dr. Yong Wang

# UNIX Background and History

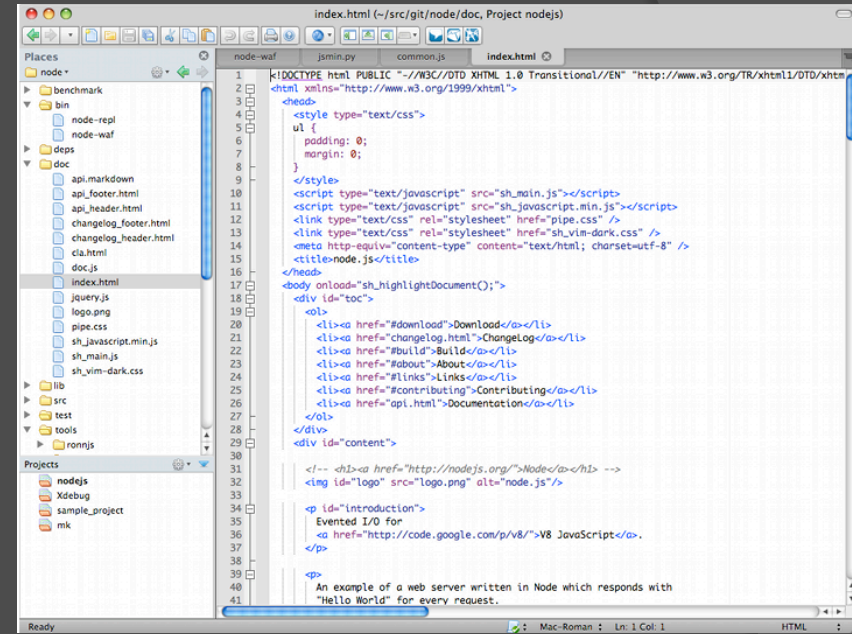
- PERL – ‘Practical Extraction and Reporting Language’
- Developed by Larry Wall in 1987 for UNIX operating systems to overcome the limitations of the standard UNIX tools
- Perl is a programming language that was designed for quickly manipulating text files
- Perl became the first major scripting language for the world wide web in the 1990’s and was used for most forms online
- Unlike other programming languages (C, C++) perl is an interpreted language, which means that the code does not need to be compiled before running.
- Perl can be run on any computer system and is pre-installed on all Apple OSX computers
- Perl scripts can easily be uploaded to servers or HPC to run programs that process large data sets



Larry Wall

# Graphical Interface for Perl Programming

- For this course we will write all of our code in KOMODO EDIT, a graphical user interface program, and then we will run the code in UNIX (or DOS on PCs)
- Komodo EDIT provides real-time feedback if you make errors while writing the code
- Komodo EDIT is an open source editor that is developed by Activestate and can be downloaded for free for APPLE or PC computers:  
<http://www.activestate.com/komodo-edit>
- Go ahead and download and install the application now



*Komodo Edit*, a graphical interface for editing perl code



**ActiveState**  
**Komodo® Edit 7**  
The open source editor for Python, PHP, Ruby, Javascript, Perl and Web development.

# Anatomy of a Simple Perl Program

```
#!/usr/bin/perl  
#comment line  
print "Hello Class! \n" ;
```

**Header line** tells Perl where the binary application is located

# hash indicates **comments**, this line will not be interpreted by Perl

Tells computer to print the subsequent text to the screen

Text in quotes will be printed to the screen

\n is a newline character that will start a new line

**Semicolon** is required at the end of every line in Perl

# Data Types in Perl

Perl has 3 data types:

- Scalars \$
- Arrays @
- Hashes %

**Scalars** store integer numbers, floats (decimals numbers) and strings (characters or words)

**Arrays** are vectors that store linear series of scalar data. They can be indexed using numbers

**Hashes** store collections of data that can be indexed using words or characters

# Scalar Variables

- ◉ Scalar Variables are used to store data
- ◉ They are declared using the \$ sign followed by a name of the variable, and an equal sign to assign the value
- ◉ In most computer languages you must declare each variables as an integer value, float or string, but Perl automatically determines the data type for you

\$float = 2.7;      This is a float variable that can  
contain decimals

\$integer = 200;      This is an integer value

\$string = "hello everybody";      This is a string which can  
contain characters and words

print \$float;  
print \$integer;      Print the values out to  
Print \$string;      the screen

# Arrays

8	9	3	4	1.99	chr	cat	dog
0	1	2	3	4	5	6	7
							Index

- Arrays are vectors that store a series of data
- Any scalar variable can be stored in an array (integers, floats or strings)

```
@array = (8 , 9 , 3 , 4, 1.99, "chr", "cat, "dog" );
```

Initialize an array with some data

```
$length = @array;
```

Determine the length of the array

```
print @array;
```

Print all contents of the array

```
print $array[0];  
print $array[5];
```

Print the values out to the screen

```
print $length;
```

Print the array length out to the screen

# Hash Variables

7.22	cat	9	red	dog	ant	18	ATG
------	-----	---	-----	-----	-----	----	-----

Dec   animal   num   gem   animal   insect   num2   dna   Hash Keys

- Hashes data collections that can be indexed with keys (not numbers)
- Hashes are initialized with the % character, followed by a key and value

```
%hash= ("dec", 7.22, "animal", "cat", "num", 9, "gem", "red",  
"animal", "dog", "insect", "ant", "num2", 18, "dna", "ATG");
```

Initialize a  
hash with  
some data

```
Print $hash{"gem"};
```

```
Print $hash{"insect"};
```

```
Print $hash{"DNA"};
```

Print the key values of several  
hash variables



# Mathematical Operators

- Mathematical operators can be performed on any integer or float scalar variables

Operator	Operation
+	addition
-	subtraction
*	multiplication
**	exponential
/	division
%	modulus
++	increment
--	decrement

```
$add = 15 + 22;
```

```
$sub = 87.43 - 7.43;
```

```
$mul = 87 * 54;
```

```
$pow = 2 ** 10;
```

```
$mod = 10 % 7;
```

```
$inc = 5;  
$inc++;
```

```
$dec = 5;  
$dec--;
```

# Logical Operators

- Logical operators are used to evaluate expression
- In computer science 0 = false, and 1 = true

Operator	Operation
&&	AND
	OR
!	NOT

```
$false = 0;  
$true = 1;
```

```
$ans = !$true;           FALSE
```

```
$ans = $true && $true;    TRUE  
$ans = $true && $false;   FALSE  
$ans = $false && $false;  FALSE
```

```
$ans = $true || $true;   TRUE  
$ans = $true || $false;  TRUE  
$ans = $false || $false; FALSE
```

# Numerical Comparators

- These operators allow comparisons of numerical values
- They return a true (1) or false (0) value when the test is performed

Operator	Test
==	equality
!=	inequality
>	Greater than
<	Less than
>=	Greater than Equal to
<=	Less than Equal to

```
$five = 5;  
$ten = 10;
```

```
$ans = ($five == $five);  
$ans = ($five == $ten);  
$ans = ($five != $five);  
$ans = ($five != $ten);
```

```
TRUE  
FALSE  
FALSE  
TRUE
```

```
$ans = ($five < $ten);  
$ans = ($five >= $ten);
```

```
TRUE  
FALSE
```

# Strings

- Strings are characters or words that are declared as scalars \$

Operator	Test
\$string1.\$string2	concatenate
length(\$string)	Find length of a string
lc(\$string)	Convert string to lowercase
uc(\$string)	Convert string to uppercase
Index(\$string1, \$string2)	Find location of string1 in string2
substr(\$string, offset, length)	Find a substring in a string

```
$dna1 = "AAAATATAATTT";
```

```
$dna2 = "CCCCGCGCGC";
```

```
$combine = $dna1.$dna2;
```

```
$len_dna1 = length($dna1);
```

```
$low_dna1 = lc($dna1);
```

```
$index = index($dna1, "TTT");
```

```
$subdna = substr($dna1, 5, 4);
```

```
AAAATATAATTTCCCCGCGCGC
```

```
12
```

```
aaaatataattt
```

```
9
```

```
ATAA
```

# String Matching with Regular Expression

- Regular Expression is a powerful tool in Perl for matching strings

command	function
<code>\$string =~ m/pattern/g</code>	Match pattern in string, return true or false
<code>\$string =~ s/pattern/replace/g</code>	Replace pattern in string

```
$dna1 = "GAAATTTTAA";
```

```
$dna1 =~ m/TTTT/g      TRUE
$dna1 =~ m/AT+TA/g     TRUE
$dna1 =~ m/AT?TA/g     FALSE
$dna1 =~ m/^G/g        TRUE
```

```
$dna1 =~ s/TTTT/CCCC/  GAAACCCCAA
$dna1 =~ s/T//g        GAAAAA
$dna1 =~ s/[G|T]A/CC/g CCAATTCCA
```

RegEx	Test
+	Can be any number of characters
?	Single character
^	Start of line
\$	End of line
[A B c]	Subset of characters
[A-E]	Series of Letters

# Conditional IF/ELSE Statements

- The IF operator in Perl will evaluate a statement and only execute a command if the statement is TRUE

**If ( test-expression) { command to execute if true }**

- Optionally, you can add an ELSE statement **else { execute if expression is false }**

```
$mendel = "monk";
```

```
if ($mendel eq "monk")  
    { print "mendel is a monk";}  
if ($mendel eq "acrobat")  
    { print "mendel is an acrobat";}
```

TRUE  
Mendel is a monk

```
if ($mendel eq "acrobat")  
    { print "mendel is an acrobat";}  
else  
    {print "mendel is a monk";}
```

FALSE  
Mendel is a  
monk

# FOR Loops

- The **FOR** loop will executed a command a specified number of times
- The format is:

*for (initializer, condition, increment)  
{ command statement }*

\$up = 10;	up	down	i
\$down = 10;			
	11	9	0
for( \$i =0; \$i<10; \$i++)	12	8	1
{	13	7	2
\$up++;	14	6	3
\$down--;	15	5	4
	16	4	5
print "\$up \n";	17	3	6
print "\$down \n";	18	2	7
}	19	1	8
	20	0	9

# WHILE / UNTIL Loops

- While and Until loops will continue to loop indefinitely until a condition is met in which the program can exit the loop

*while/unless (condition) { command statement }*

- While loops assume a condition is TRUE and exits when it becomes FALSE
- Unless loops assume a condition is FALSE and exit when it becomes TRUE

```
$num = 0;
```

```
while( $num < 10)
```

```
{
```

```
  $num++;
```

```
  print $num;
```

```
}
```

```
until( $num == 10)
```

```
{
```

```
  $num++;
```

```
  print $num;
```

```
}
```

OUTPUT

1

2

3

4

5

6

7

8

9

TRUE

FALSE



# Foreach element in an Array

- Foreach is a special command in Perl that allows you to traverse all elements within an array, similar to a for loop, but using less code
- The limitation is that FOREACH does not keep track of the index

```
@arr = (5, 10, 15, 20, 25, 30, 35, 40, 45)
```

```
foreach $val(@arr)  
{
```

```
    @arr[$val] = @arr[$val]+0.5;
```

```
    print "@arr[$val] \n";
```

```
}
```

OUTPUT

5.5

10.5

15.5

20.5

25.5

30.5

35.5

40.5

45.5

# File Input

- Perl has commands for *reading in* and *writing out* text files
- @ARGV is an array that takes input when the program is run
- To input the filename into the program, you run the program followed by the name of the input filename
- Ex.

```
perl program.pl einstein.txt
```

```
$infile= $ARGV[0];
```

```
Open(TXT, "<$infile");
```

```
@text = <TXT>;  
print "$text[2]";
```

```
close(TXT)
```

Output:  
violent opposition

einstein.txt

Great spirits have  
always encountered  
violent opposition  
from mediocre minds

-albert einstein

# File Output

- @ARGV can both input and output file names
- Ex.

```
perl program.pl einstein.txt output.txt
```

```
$infile= $ARGV[0];  
$outfile = $ARGV[1];  
  
Open(TXT, "<$infile");  
Open(OUT, ">$outfile");  
  
@text = <TXT>;  
  
print OUT "$text[2]";  
  
close(TXT)
```

einstein.txt

Great spirits have  
always encountered  
violent opposition  
from mediocre minds

-albert einstein

output.txt

violent opposition

# Subfunctions

- When code becomes long, it is often advantageous to break the code down into subfunctions (sub), which are executed using the `&` character
- Variables can be passed to the subfunction, and returned with the **return** command
- Within the subfunction the `$_[0]` syntax is used to access passed variables

```
#!/usr/bin/perl
```

```
$var = 10;  
$result = &calculation($var);  
Print $result;
```

```
sub calculation  
{  
    $num = $_[0] * 66;  
    return($num);  
}
```

OUTPUT  
660

# PERL Bioinformatics Workshop

The workshop for today can be found by directing your web browser to this URL

**<http://www.navinlab.com/bioperl>**

Follow the instructions on the website to complete the workshops and don't be afraid to ask for help

**Note:** This is a long workshop and it is very unlikely that you will be able to finish it during the class

Please finish all sections as homework before the next class, the website can be accessed from anywhere

PS The UNIX workshop from last week has been moved to:

**<http://www.navinlab.com/biounix>**