

Perl for Biologists

Session 2

March 11, 2015

Constants, variables and functions

Jaroslav Pillardy

"shebang" notation – path to the program to interpret the script, must be the first line and start with #!

using external
module/library

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

```
use warnings;
```

```
#this is my first Perl script
```

```
print "Hello, CBSU\n";
```

anything starting with # is
a comment, unless it is #!
in the first line

function to print out text

statement ends with a
semicolon

"shebang" notation – path to the program to interpret the script, must be the first line and start with #!

using external
module/library

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

parentheses can be always
omitted, unless it changes
the meaning of expression

```
use warnings;
```

```
#this is my first Perl script
```

```
print("Hello, CBSU\n");
```

anything starting with # is
a comment, unless it is #!
in the first line

function to print out text

statement ends with a
semicolon

Session 1 Exercises Review

1. Write a Perl program that prints your name and e-mail in the following format in one line:

first_name last_name <emailaddr@domain.edu>

`/home/jarekp/perl_01/exercise1.pl`

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

```
print 'Jarek Pillardy <jp86@cornell.edu>';
```

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

```
print "Jarek Pillardy <jp86\@cornell.edu>\n";
```

2. Are the following modules installed on your BioHPC Lab machine?

Net::Ping

XML::Special

Net::Telnet

CBSU::HDF5

```
>perl -MNet::Ping -e "print \"OK\n\"; "
```

Scalar Variables

Variable – a name for a block in computer memory holding *something*

Scalar variable – a variable containing only one element

Expression representing a constant value is a *literal*

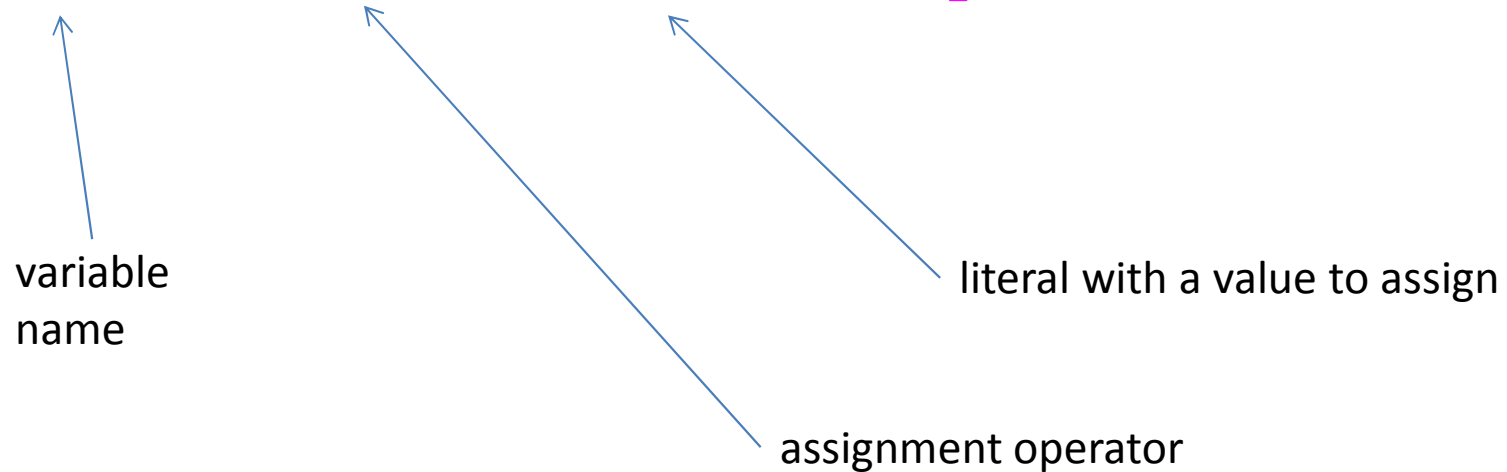
Scalar Variables

Scalar variables in Perl always start with \$

String variable:

```
$variable = "Jarek Pillardy";
```

variable
name



literal with a value to assign

assignment operator

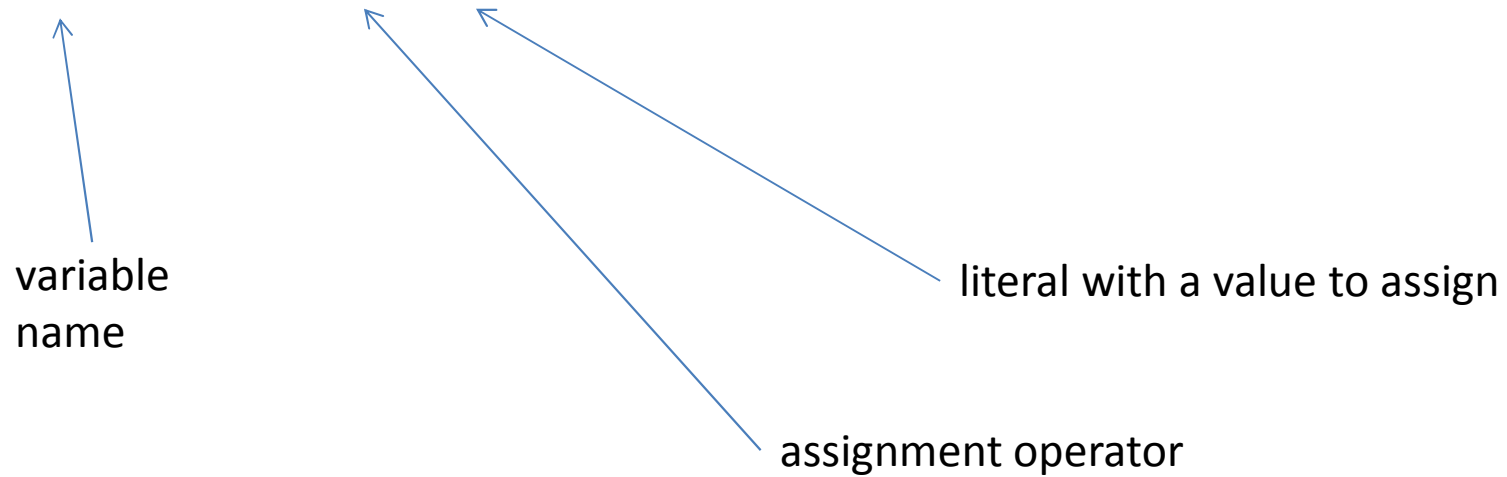
Scalar Variables

Scalar variables in Perl always start with \$

Numerical variable:

```
$variable = 55;
```

variable
name



literal with a value to assign

assignment operator

Variable names can contain letters, numbers and underscores

Case sensitive

Cannot start from number (digit)

```
$JarekPillardy
```

```
$jarekpillardy          #different than above
```

```
$jp86_cornell_edu
```

```
$123jarek               ← INVALID, starts with a number
```

```
$jp86@cornell.edu       ← INVALID, contains @ and .
```


script1.pl

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

$svar = "\"Hello, CBSU\"\\n";

$nvar = 55.55;

print $svar;

print $nvar;

print "\\n";
```

script1.pl

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

$svar = "\"Hello, CBSU\"\\n";

$nvar = 55.55;

print $svar;

print $nvar;

print "\\n";
```

All scripts for this session can be copied from
/home/jarekp/perl_02
in this case /home/jarekp/perl_02/script1.pl
>cp /home/jarekp/perl_02/script1.pl .
copies this script to your current directory

script1.pl

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

```
$svar = "\"Hello, CBSU\"\\n\";
```

```
$nvar = 55.55;
```

```
print $svar;
```

```
print $nvar;
```

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

```
[jarekp@cbsum1c2b014 perl_02]$ perl script1.pl
"Hello, CBSU"
55.55
[jarekp@cbsum1c2b014 perl_02]$
```

script2.pl

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

$svar = "\"Hello, CBSU\"\\n";

$nvar = 55.55;

print "$svar nvar=$nvar\\n";
```

script2.pl

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

$svar = "\"Hello, CBSU\"\\n";

$nvar = 55.55;

print "$svar nvar=$nvar\\n";
```

```
[jarekp@cbsum1c2b014 perl_02]$ perl script2.pl
"Hello, CBSU"
nvar=55.55
[jarekp@cbsum1c2b014 perl_02]$
```

String Variables

Can be assigned both single quoted or double quoted strings

```
$variable1 = "Hello, CBSU\n";
```

```
$variable2 = 'Hello, CBSU\n';
```

String operators:

. Concatenation

```
$str1 = "Jarek" . " " . "Pillardy";  
$str1 = "Jarek Pillardy";      #same as above
```

x Repetition

```
$str2 = "AAGT" x 3;  
$str2 = "AAGTAAGTAAGT";      #same as above
```

Numerical Variables

Can be a floating point, integer, or non-decimal number

```
$variable1 = 1000000;           #integer
$variable1 = 1_000_000;         #integer, _ ignored
$variable1 = 1e+6;              #integer
$variable1 = 2.6182818285e-3;   #floating point
$variable1 = 0xffff34g;         #hexadecimal
$variable1 = 02351;             #octal
$variable1 = 0b101101;         #binary
```

ALL numerical variables are stored the same way in Perl
– as double precision floating point numbers

Numerical Operators

```
$variable1 = 1000000 + 222;
```

```
$variable1 = 1000000 * 222;
```

```
$variable1 = 1000000 - 222;
```

```
$variable1 = 1000000 / 222;
```

```
$variable1 = 121**3;           #power, =1771561
```

```
$variable1 = 1000000 % 222;    #modulus, =112
```

Numerical Built-In Functions

| | |
|--------------------------|-----------------------------|
| abs | #absolute value |
| sin cos tan atan2 | #trigonometry |
| exp log sqrt | #exponent, log, square root |
| int | #convert to int |
| hex oct | #convert to hex oct |
| srand rand | #random numbers |

```
$variable=100;  
print sqrt($variable);
```

String Built-In Functions (some)

| | |
|--|---|
| substr (\$var, \$start, \$length) | #substring, 0-based |
| chomp (\$var) | #removes trailing \n |
| index (\$var, \$str) | #position of \$str in \$var |
| reverse (\$var) | #reverse string |
| rindex (\$var, \$str) | #reverse index |
| uc (\$var) | lc (\$var) #uppercase, lowercase |

String Built-In Functions (some)

ord(\$var) #converts character to its
 #numerical ASCII value

\$num = **ord**("a") #\$num is now 97

chr(\$nvar) #converts int into corresponding
 #ASCII character

\$char = **chr**(99) #\$char is now "c"

ASCII Table

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|-----|----|-----|------------------------------------|-----|----|-----|-------|--------------|-----|----|-----|-------|----------|-----|----|-----|--------|------------|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 | | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | \$ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | (| (| 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 |) |) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [| [| 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 |] |] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | | DEL |

Source: www.LookupTables.com

Finding more about functions and modules

Use perldoc command

```
>perldoc -f ord
```

```
>perldoc Net::Telnet
```

Search perldoc on the web

<http://perldoc.perl.org/perlfunc.html>

Binary assignment

```
$variable = 1;  
$variable = $variable + 3;      #variable is now 4  
$variable += 3;                #variable is now 7,  
                                # same effect
```

```
$svar = "Jarek";  
$svar .= " Pillardy";          #variable is now  
                                # "Jarek Pillardy"
```

```
$svar = "My name is " . $svar;  
                                #"My name is Jarek Pillardy"
```

or

```
$svar .= "My name is ";        ← NOT SAME AS ABOVE  
                                #"Jarek PillardyMy name is "
```

script3.pl

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

$svar = "Hello, CBSU\n";
print "svar = $svar";
$nvar = 55.55;
print "nvar = $nvar\n";

$nvar += 10;
print "nvar is now $nvar\n";

$svar .= "Hello again\n";
print $svar;
$svar = "Hello first\n" . $svar;
print $svar;
```


script3.pl

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

```
$svar = "Hello, CBSU\n";
```

```
print "svar = $svar";
```

```
$nvar = 55.55;
```

```
print "nvar = $nvar\n";
```

```
$nvar += 10;
```

```
print "nvar is now $nvar\n";
```

```
$svar .= "Hello again\n";
```

```
print $svar;
```

```
$svar = "Hello first\n" . $svar;
```

```
print $svar;
```

```
[jarekp@cbsum1c2b014 perl_02]$ perl script3.pl
svar = Hello, CBSU
nvar = 55.55
nvar is now 65.55
Hello, CBSU
Hello again
Hello first
Hello, CBSU
Hello again
[jarekp@cbsum1c2b014 perl_02]$
```

script3a.pl

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

$sva1 = "Hello, CBSU\n";
print "sva1 = $sva1\n";

$sva1 = $sva1;
chomp($sva1);
print "sva1 = $sva1\n";

$sva1 = substr($sva1, 0, 5);
print "sva1 = $sva1\n";

print index($sva1, ",") . "\n";
print index($sva1, ",") . "\n";

print uc($sva1) . "\n";
```

script3a.pl

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

$svar = "Hello, CBSU\n";
print "svar = $svar\n";

$svar1 = $svar;
chomp($svar1);
print "svar1 = $svar1\n";

$svar1 = substr($svar1, 0, 5);
print "svar1 = $svar1\n";

print index($svar, ",") . "\n";
print index($svar1, ",") . "\n";

print uc($svar1) . "\n";
```

```
[jarekp@cbsum1c2b014 perl_02]$ perl script3a.pl
svar = Hello, CBSU
```

```
svar1 = Hello, CBSU
svar1 = Hello
5
-1
HELLO
```

```
[jarekp@cbsum1c2b014 perl_02]$
```

Automatic Variable Conversion or *Variable Interpolation*

Perl is a context-based language, variables will be converted (or *interpolated*) as needed

```
$nvar = 55.5;  
$svar = "The number nvar is " . $nvar;
```

Perl expects string since
string operation is being
used

`$nvar` is converted to string and
concatenated with preceding string

Automatic Variable Conversion or *Variable Interpolation*

Perl is a context-based language, variables will be converted (or *interpolated*) as needed

```
$nvar = 55.5;  
$svar = "2variable6str ";
```

```
$nnn = $nvar * $svar;
```

*#\$nnn is now 55.5*2 = 111*

Perl expects number since
numeric operation is being
used

\$svar is converted to number, all trailing
letters and non-numbers are discarded, if there
are no starting numbers the result is 0

script4.pl

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

$nvar = 55.5;
$svar = "The number nvar is " .
$nvar;
print "$svar\n";

$nvar = 55.5;
$svar = "2variable1str 3a";
$nnn = $nvar * $svar;
print "$nnn\n";

print "55.5" . 2 * 7;
print "\n";
```

script4.pl

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

```
$nvar = 55.5;  
$svar = "The number nvar is " .  
$nvar;  
print "$svar\n";
```

```
$nvar = 55.5;  
$svar = "2variable1str 3a";  
$nnn = $nvar * $svar;  
print "$nnn\n";
```

```
print "55.5" . 2 * 7;  
print "\n";
```

```
[jarekp@cbsum1c2b014 perl_02]$ perl script4.pl  
The number nvar is 55.5  
111  
55.514  
[jarekp@cbsum1c2b014 perl_02]$
```

script5.pl

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

```
use warnings;
```

```
$nvar = 55.5;
```

```
$svar = "The number nvar is " .
```

```
$nvar;
```

```
print "$svar\n";
```

```
$nvar = 55.5;
```

```
$svar = "2variable1str 3a";
```

```
$nnn = $nvar * $svar;
```

```
print "$nnn\n";
```

```
print "55.5" . 2 * 7;
```

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


script5.pl

```
[jarekp@cbsum1c2b014 perl_02]$ perl script5.pl
The number nvar is 55.5
Argument "2variable1str 3a" isn't numeric in multiplication (*) at script5.pl line 11.
111
55.514
[jarekp@cbsum1c2b014 perl_02]$
```

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

```
use warnings;
```

```
$nvar = 55.5;
$svar = "The number nvar is " . $nvar;
print "$svar\n";
```

```
$nvar = 55.5;
$svar = "2variable1str 3a";
$nnn = $nvar * $svar;
print "$nnn\n";
```

```
print "55.5" . 2 * 7;
print "\n";
```

script6.pl

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

```
use warnings;
```

```
$nvar = 2;  
print "$nvar\n";
```

```
$nvar1 = $nvar * 10;  
print "$nvar1\n";
```

```
$nvar1 = $nvar * 010;  
print "$nvar1\n";
```

```
$nvar1 = $nvar * "010";  
print "$nvar1\n";
```

script6.pl

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

```
use warnings;
```

```
$nvar = 2;
```

```
print "$nvar\n";
```

```
$nvar1 = $nvar * 10;
```

```
print "$nvar1\n";
```

```
$nvar1 = $nvar * 010;
```

```
print "$nvar1\n";
```

```
$nvar1 = $nvar * "010";
```

```
print "$nvar1\n";
```

```
[jarekp@cbsum1c2b014 perl_02]$ perl script6.pl
2
20
16
20
[jarekp@cbsum1c2b014 perl_02]$
```

What if we use a variable that has not been declared?

```
print "=>$newvar<=\n";
```

What if we use a variable that has not been declared?

```
print "=>$newvar<=\n";
```

No problem, any new variable is assigned a special value: *undef*

It will interpolate to

an empty string in string context

0 in numerical context

BEWARE: USING UNINITIALIZED VARIABLE IS A VERY COMMON SOURCE OF ERRORS. USE WARNINGS, IT HELPS.

script7.pl

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

```
use warnings;
```

```
print ">$newvar<=<\n";
```

script7.pl

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

```
use warnings;
```

```
print "=>$newvar<=<=\\n";
```

```
[jarekp@cbsum1c2b014 perl_02]$ perl script7.pl
```

```
Name "main::newvar" used only once: possible typo at script7.pl line 5.
```

```
Use of uninitialized value $newvar in concatenation (.) or string at script7.pl line 5.
```

```
==><=
```

```
[jarekp@cbsum1c2b014 perl_02]$
```

Numbers

ALL numbers are represented in Perl as *double-precision floating point* numbers

On 64 bit machines each takes 8 bytes = 64 bits

script8.pl

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

```
$nvar = 2**1023;
```

```
print "$nvar\n";
```

```
$nvar = 2**1024;
```

```
print "$nvar\n";
```

```
$nvar = 2**-1074;
```

```
print "$nvar\n";
```

```
$nvar = 2**-1075;
```

```
print "$nvar\n";
```

script8.pl

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

```
$nvar = 2**1023;
```

```
print "$nvar\n";
```

```
$nvar = 2**1024;
```

```
print "$nvar\n";
```

```
$nvar = 2**-1074;
```

```
print "$nvar\n";
```

```
$nvar = 2**-1075;
```

```
print "$nvar\n";
```

overflow

underflow

```
[jarekp@cbsum1c2b014 perl_02]$ perl script8.pl
8.98846567431158e+307
inf
4.94065645841247e-324
0
[jarekp@cbsum1c2b014 perl_02]$
```

script9.pl

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

```
$nvar = log(10);  
print "$nvar\n";  
$nvar = $nvar + 1e-14;  
print "$nvar\n";  
$nvar = $nvar + 1e-15;  
print "$nvar\n";
```

script9.pl

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

```
$nvar = log(10);  
print "$nvar\n";  
$nvar = $nvar + 1e-14;  
print "$nvar\n";  
$nvar = $nvar + 1e-15;  
print "$nvar\n";
```

15 digit accuracy:

1e-14 + 1 different than 1

1e-15 + 1 same as 1

```
[jarekp@cbsum1c2b014 perl_02]$ perl script9.pl  
2.30258509299405  
2.30258509299406  
2.30258509299406  
[jarekp@cbsum1c2b014 perl_02]$
```

Exercises

1. In a Perl program create a string representing a 54 bp DNA strand consisting of 6 repeats, store it in a variable. Create another variable containing the above DNA reversed. Create the third variable storing a subsequence of the original sequence from position 31 to position 47. Print all three.

Hint: Use string functions and operators to create strings from a repeat.

2. Use perldoc to find out how rand() and srand() functions work. Write a Perl program that produces a 17 character string composed of random lower case letters, store it in a variable and print it out. Run the program several times and compare the results.

Hint: use chr(), int() functions and ASCII table.