Perl for Biologists

Session 10

May 6, 2015

Object Oriented Programming and BioPERL

Jaroslaw Pillardy

Subroutines and functions

Subroutine can be declared in Perl script as a named block of code:

```
sub sub_name
{
        code;
}
```

There is no difference between subroutine and function: declaration is the same and it **ALWAYS returns a value** (not always a useful one …)

Subroutines and functions

Subroutine can be called or referenced in two ways

By name as an object

```
&sub_name;
```

By name as a subroutine

```
sub_name(arg1, arg2, …);
```

Global and local variables: scope

Local variables, accessible only in a given code block can be declared using "**my**" keyword:

```
my $variable;
```

Local variable can be declared in ANY code block, not only in a subroutine.

Local variable declared in a code block is also declared in all child code blocks inside this code block

Using local variables wherever possible is a VERY good programming practice.

Arguments

In programming, there are two methods of passing variables to subroutines:

BY VALUE

The subroutine receives a **copy of the data**, and any changes made in a subroutine **DO NOT** affect original variables

BY REFERENCE

The subroutine receives **exactly the same variables** as listed in the arguments, any changes made in a subroutine **DO affect** original variables. This is Perl default.

Arguments

Variables passed to the subroutine are always passed in list context, i.e. as if they were one long array.

It works if we want to pass a few scalar variables followed by ONE array:

```
subroutine($var1, $var2, @arr);
```

We can recover variables in the subroutine if we know how many scalars are in the front:

```
my ($var1, $var2, @arr) = @_;
```

Arguments

... or by using references to arrays or hashes:

```
subroutine(\@arr1, $var2, \@arr2);
```

...

```
my ($arr1ref, $var2, $arr2ref) = @_;
@arr1 = @{$arr1ref};
@arr2 = @{$arr2ref};
```

Session 9 Exercises

1. Take a close look at script2b.pl. There are several potential problems with this script, find them and modify the script to fix the problems.

a) Function `opendir` may fail to open directory, but potential error is not handled at all.

b) What if a directory is empty? Subroutine `listfiles` variable `$n` will be zero in this case, triggering "division by zero" error. This case should be detected and handled separately.

c) The subroutine counts all objects in a directory except "." and ".." for the total number of files, which includes directories and symbolic links. Instead, only total number of files should be counted.

d) What if we have a directory ending with ".pl"? It will be counted as Perl script file, while instead only files with ".pl" extension should be counted.

/home/jarekp/perl_09/exercise1.pl

Session 9 Exercises

2. Modify script4.pl to eliminate problem with symbolic links circular references.

There are two ways to eliminated this problem:

a) Do not allow the script to follow symbolic links

   /home/jarekp/perl_09/exercise2a.pl

b) Limit the script to maximum recursion level (i.e. number of times the function calls itself).

   /home/jarekp/perl_09/exercise2b.pl

Session 9 Exercises

3. Write a program computing factorial of an integer specified in its command line. Use a recursive function to compute factorial, make sure it handles errors (like invalid input) properly.

Session 9 Exercises

/home/jarekp/perl_09/exercise3.pl

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

if($#ARGV != 0)
{
        print "USAGE: exercise3.pl number\n";
        exit;
}
my $n = $ARGV[0];
if($n !~ /^\d+$/)
{
        print "$n is not a positive integer\n";
        exit;
}

$nf = factorial($n);

print "$n! is $nf\n";

sub factorial
{
        my ($m) = @_;

        if($m <= 1){return 1;}
        return $m * factorial($m - 1);
}
```

Session 9 Exercises

4. Write a program that computes total GC content of all sequences in a given fasta file (file name should be given as an argument). Use subroutines to make the code clean, understandable and reusable. Test the program on fasta file from session 8 (/home/jarekp/perl_08/fasta_in.fa).

/home/jarekp/perl_09/exercise4.pl

Let's create a script to

1. Compute reverse-complement of a DNA string

2. Cut the a DNA string at a specified pattern (apply digestion enzyme) and print the number of fragments created

Lets do it using subroutines and functions

… and then convert it to an object-oriented code.

script1.pl (1)

```
#!/usr/local/bin/perl
use strict;
use warnings;

my $seqStr = "ACGGGCTGAATTCGGGGAATTTCCCTTACTAGAATTCAGCGGGACCCAGGGAGCCCC";

print revcom($seqStr), "\n";

print cut($seqStr, "GAATTC"), "\n";
```

```
sub cut
{
        my $this_seq_string = shift @_;
        my $pattern = shift @_;
        my @sites = split /$pattern/, $this_seq_string;
        return $#sites+1;
}

sub revcom
{
        my $this_seq_string = shift @_;
        my $result = reverse $this_seq_string;
        $result=~s/A/X/gi;
        $result=~s/T/A/gi;
        $result=~s/X/T/gi;
        $result=~s/C/X/gi;
        $result=~s/G/C/gi;
        $result=~s/X/G/gi;
        return $result
}
```

Re-write the code with Object Oriented PERL

Subroutine

script1.pl

```
my $seqStr = "ACGGGCTGAATTCGGGGAATTTCCCTTACTAGAATTCAGCGGGACCCAGGGAGCCCC";

print revcom($seqStr), "\n";

print cut($seqStr, "GAATTC"), "\n";
```

Object Oriented Perl

script2.pl

```
use mySeqAnalyzer;

my $seqObject = mySeqAnalyzer->new ("ACGGGCTGAATTCGGGGAATTTCCCTTACTAGAATTCAGCGGGACCCAGGGAGCCCC");

print $seqObject->revcom(), "\n";

print $seqObject->cut("GAATTC"), "\n";
```

Basics of Object Oriented Programming

Class: A template that defines the state and behavior of a data type.

Object: An instance of a class: a named part of memory organized according to class (template) definition. It is a pointer.

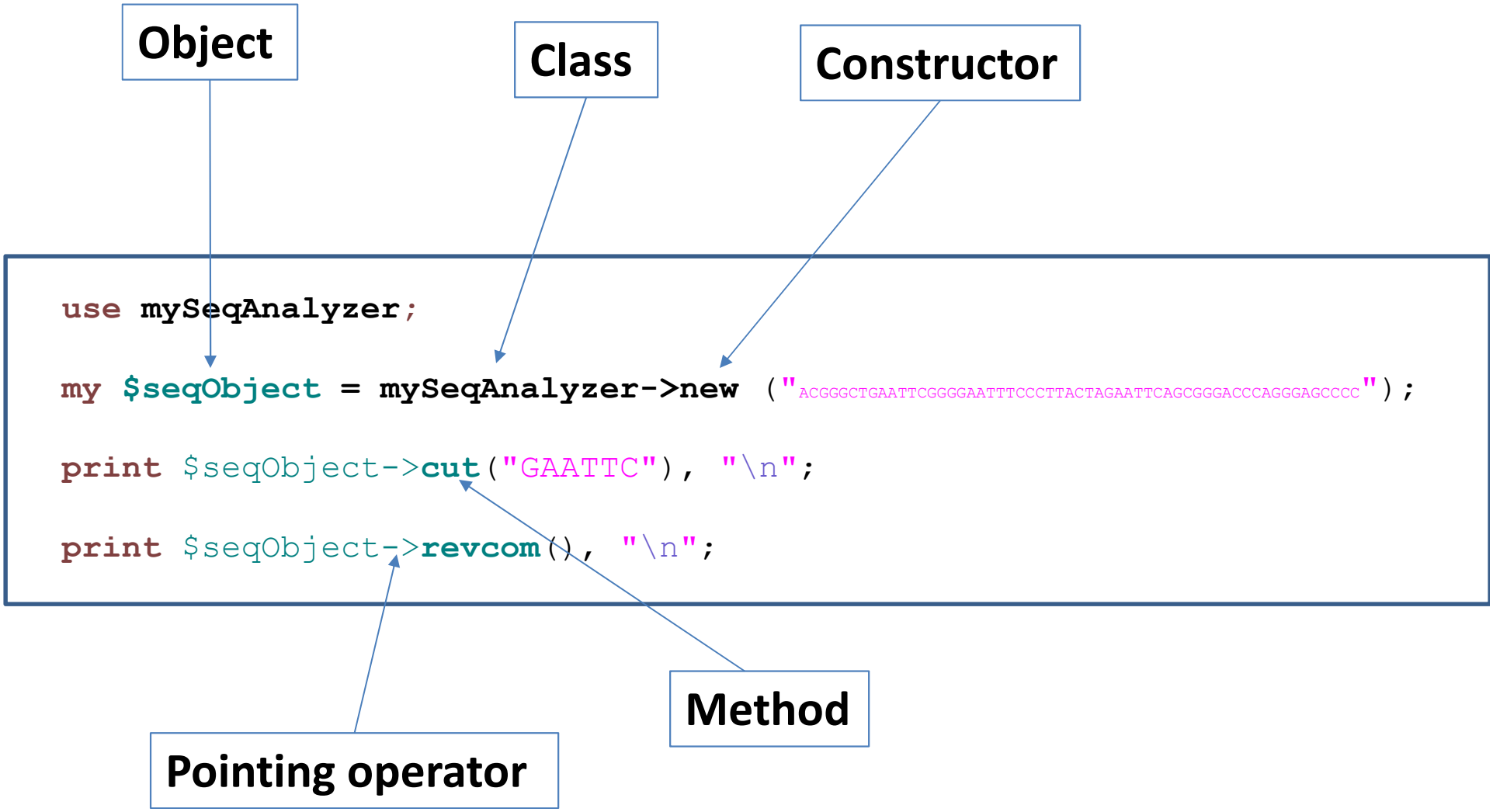
Constructor: A method to create an object, normally named "new".

Method: A function in a class (and object).

Field: A data field (variable) in an object. Usually private (hidden).

Property: An exposed data field (variable) in an object. In Perl same as field, in many other languages it is a method that gets or sets a variable in a way that it looks like a variable itself.

Syntax in Object Oriented PERL Programming



mySeqAnalyzer.pm (1)

```
#!/usr/local/bin/perl
use strict;
use warnings;

package mySeqAnalyzer;

sub new
{
    my $class = shift;
    my $self = {
                    _seqstr => shift
                };
    return bless $self;
}
```

```
sub cut
{
      my ($self, $pattern) = @_;
      my @sites = split /$pattern/, $self->{_seqstr};
      return $#sites+1;
}

sub revcom
{
         my ($self) = @_;
         my $result = reverse $self->{_seqstr};
         $result=~s/A/X/gi;
         $result=~s/T/A/gi;
         $result=~s/X/T/gi;
         $result=~s/C/X/gi;
         $result=~s/G/C/gi;
         $result=~s/X/G/gi;
         return $result;
}
1;
```

script2.pl

```
#!/usr/local/bin/perl
use strict;
use warnings;

use mySeqAnalyzer;

my $seqObject = mySeqAnalyzer->new
("ACGGGCTGAATTCGGGGAATTTCCCTTACTAGAATTCAGCGGGACCCAGGGAGCCCC");

#lets print an object
print "seqObject is " . $seqObject . "\n";
#lets print results of methods
print "revcom() is\n";
print $seqObject->revcom(), "\n";
print "cut(\"GAATTC\") is\n";
print $seqObject->cut("GAATTC"), "\n";
#lets print internal data (variable)
print "field _seqstr is\n";
print $seqObject->{_seqstr} . "\n";
```

Re-write the code with Object Oriented PERL

Subroutine

```
my $seqStr = "ACGGGCTGAATTCGGGGAATTTCCCTTACTAGAATTCAGCGGGACCCAGGGAGCCCC";

print $seqStr;
```

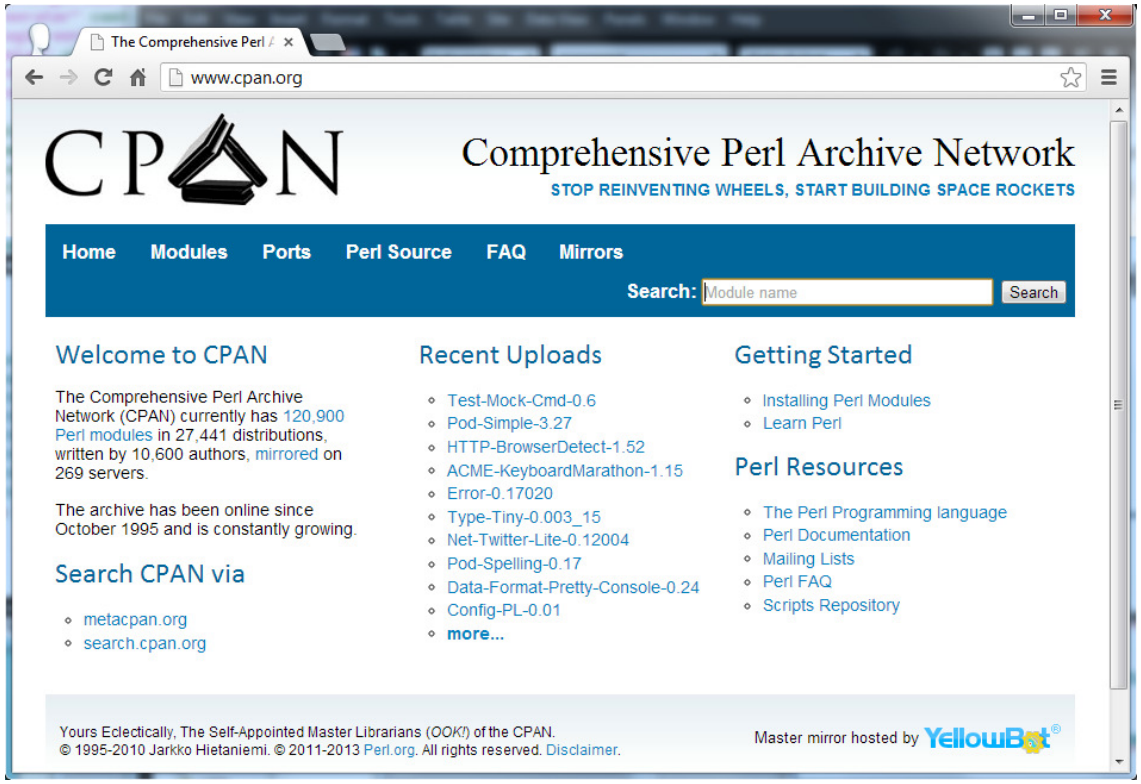
String

Object Oriented PERL

```
use mySeqAnalyzer;

my $seqObject = mySeqAnalyzer->new ("ACGGGCTGAATTCGGGGAATTTCCCTTACTAGAATTCAGCGGGACCCAGGGAGCCCC");


print $seqObject->{_seqstr};

print $seqObject->seq();
```

Reference to an object

CPAN (www.cpan.org) : The largest PERL module depository



Before you write your own PERL function, you might want to check CPAN to see if it is already available.

Installation of PERL modules

Shared location (require root privilege):
/usr/local/lib/perl5/site_perl/5.16.2/x86_64-linux-thread-multi
/usr/local/lib/perl5/site_perl/5.16.2
/usr/local/lib/perl5/5.16.2/x86_64-linux-thread-multi
/usr/local/lib/perl5/5.16.2

Local (in user home directory):
/home/jarekp/perl5/lib/perl5

Paths of installed PERL modules are defined in the @INC array

$ perl -V

```
/usr/local/lib/perl5/site_perl/5.16.2/x86_64-linux-thread-multi
/usr/local/lib/perl5/site_perl/5.16.2
/usr/local/lib/perl5/5.16.2/x86_64-linux-thread-multi
/usr/local/lib/perl5/5.16.2
```

Paths of installed PERL modules are defined in the @INC array

If module not found, you will see error message like:

```
$ perl -e "use xxxxx;"
Can't locate xxxxx.pm in @INC (@INC contains:
/usr/local/lib/perl5/site_perl/5.16.2/x86_64-linux-
thread-multi /usr/local/lib/perl5/site_perl/5.16.2
/usr/local/lib/perl5/5.16.2/x86_64-linux-thread-multi
/usr/local/lib/perl5/5.16.2 .) at -e line 1.
```

PERL modules can be installed locally

On Linux, run the command "cpan" .
First time running cpan, you will need to configure the cpan. When prompted for questions, you can use the default answer by simply press "return". Type "exit" and press "return" after cpan configuration is finished.

At this point, you will need to logout and login

```
$ perl -V
/home/jarekp/perl5/lib/perl5
/usr/local/lib/perl5/site_perl/5.16.2/x86_64-linux-thread-multi
/usr/local/lib/perl5/site_perl/5.16.2
/usr/local/lib/perl5/5.16.2/x86_64-linux-thread-multi
/usr/local/lib/perl5/5.16.2
```

A new path was added

Sometimes automatic configuration gets confused …

If you don't see new path leading to /home/xxxx/perl5 in perl -V

Edit your /home/xxxx/.bashrc file and add

```
export PERL_LOCAL_LIB_ROOT="$PERL_LOCAL_LIB_ROOT:/home/xxxx/perl5";
export PERL_MB_OPT="--install_base /home/xxxx/perl5";
export PERL_MM_OPT="INSTALL_BASE=/home/xxxx/perl5";
export PERL5LIB="/home/xxxx/perl5/lib/perl5:$PERL5LIB";
export PATH="/home/xxxx/perl5/bin:$PATH";
```

Log out and log in and then rerun perl -V.

Where are your CPAN settings?

On Linux they are in your home directory, usually under

/home/xxxx/.local/share/.cpan

CPAN will also modify your startup script (/home/jarekp/.bashrc):

```
export PERL_LOCAL_LIB_ROOT="$PERL_LOCAL_LIB_ROOT:/home/jarekp/perl5";
export PERL_MB_OPT="--install_base /home/jarekp/perl5";
export PERL_MM_OPT="INSTALL_BASE=/home/jarekp/perl5";
export PERL5LIB="/home/jarekp/perl5/lib/perl5:$PERL5LIB";
export PATH="/home/jarekp/perl5/bin:$PATH";
```

If you ever want to start fresh with no local Perl modules

- Remove directory /home/xxxx/.local/share/.cpan
- Remove Perl export lines from your /home/xxxx/.bashrc
- Remove directory tree /home/xxxx/perl5
- Log out and log in

1. To install a PERL module, use “cpan install Module_Name

e.g.

```
cpan install String::Random
```

2. After installation, you can verify the installation by

```
perldoc String::Random
```

Or

```
perl –e "use String::Random"
```

Or

```
perl -MString::Random -e "print \"OK\n\"";
```

Example: Generate a random DNA sequence

1. **Install PERL module String::Random**
   cpan install String::Random

2. **Read the documentation of this module**
   perldoc String::Random

3. **Write a script.**

script3.pl

```
#!/usr/bin/perl
use strict;
use warnings;

use String::Random;
my $RandomSeq =  String::Random->new();
my $seqstr= $RandomSeq->randregex('[ACGT]{1000}');
print $seqstr, "\n";
```

Introduction to BioPERL

http://www.bioperl.org

Bio::Seq object

>gi|24940137|emb|AJ419826.1| Coffea arabica mRNA for rubisco small subunit
ATTCCCTTGCTGTTATTAGAAGAAAAAAGGAAGGGAACGAGCTAGCGAGAATGGCATCCTCAATGATCTC
CTCGGCAGCTGTTGCCACCACCACCAGGGCCAGCCCTGCTCAAGCTAGCATGGTTGCACCCTTCAACGGC
CTCAAAGCCGCTTCTTCATTCCCCATTTCCAAGAAGTCCGTCGACATTACTTCCCTTGCCACCAACGGTG
GAAGAGTCCAGTGCATGCAGGTGTGGCCACCAAGGGGACTGAAGAAGTACGAGACTTTGTCATATCTTCC
AGATCTCACCGACGAGCAATTGCTCAAGGAAATTGATTACCTTATCCGCAGTGGATGGGTTCCTTGCTTG
GAATTCGAGTTGGAGAAAGGATTTGTGTACCGTGAATACCACAGGTCACCGGGATACTATGACGGACGCT

Properties:

1. **display_id** : gi|24940137|emb|AJ419826.1|
2. **desc**: Coffea arabica mRNA for rubisco small subunit
3. **seq**: ATTCCCTTG……
4. **alphabet**: dna ('dna', 'rna', or 'protein')

Bio::Seq object

>gi|24940137|emb|AJ419826.1| Coffea arabica mRNA for rubisco small subunit
ATTCCCTTGCTGTTATTAGAAGAAAAAAGGAAGGGAACGAGCTAGCGAGAATGGCATCCTCAATGATCTC
CTCGGCAGCTGTTGCCACCACCACCAGGGCCAGCCCTGCTCAAGCTAGCATGGTTGCACCCTTCAACGGC
CTCAAAGCCGCTTCTTCATTCCCCATTTCCAAGAAGTCCGTCGACATTACTTCCCTTGCCACCAACGGTG
GAAGAGTCCAGTGCATGCAGGTGTGGCCACCAAGGGGACTGAAGAAGTACGAGACTTTGTCATATCTTCC
AGATCTCACCGACGAGCAATTGCTCAAGGAAATTGATTACCTTATCCGCAGTGGATGGGTTCCTTGCTTG
GAATTCGAGTTGGAGAAAGGATTTGTGTACCGTGAATACCACAGGTCACCGGGATACTATGACGGACGCT

Methods:

1. **display_id ()**: get or set id. E.g. $seqobj->display_id("newID");
2. **desc()**: get or set description line.
3. **seq()**: get or set sequence string
4. **alphabet()**: get or set alphabet

Bio::Seq object

>gi|24940137|emb|AJ419826.1| Coffea arabica mRNA for rubisco small subunit
ATTCCCTTGCTGTTATTAGAAGAAAAAAGGAAGGGAACGAGCTAGCGAGAATGGCATCCTCAATGATCTC
CTCGGCAGCTGTTGCCACCACCACCAGGGCCAGCCCTGCTCAAGCTAGCATGGTTGCACCCTTCAACGGC
CTCAAAGCCGCTTCTTCATTCCCCATTTCCAAGAAGTCCGTCGACATTACTTCCCTTGCCACCAACGGTG
GAAGAGTCCAGTGCATGCAGGTGTGGCCACCAAGGGGACTGAAGAAGTACGAGACTTTGTCATATCTTCC
AGATCTCACCGACGAGCAATTGCTCAAGGAAATTGATTACCTTATCCGCAGTGGATGGGTTCCTTGCTTG
GAATTCGAGTTGGAGAAAGGATTTGTGTACCGTGAATACCACAGGTCACCGGGATACTATGACGGACGCT

Other Methods:

1. **revcom ()**: return reverse-complement sequence object
2. **translate()**: translate.
3. **subseq()**: return a substring of the sequence
4. **trunc()**: return a sequence object with part of the sequence

A simple example:

script4.pl

```
#!/usr/local/bin/perl
use strict;
use warnings;

use String::Random;
use Bio::Seq;

my $RandomSeq =  String::Random->new();
my $seqstr= $RandomSeq->randregex('[ACGT]{1000}');

my $seqObject = Bio::Seq->new (-seq => $seqstr,
                               -display_id => "myseq1",
                               -desc => "This is an example",
                               -alphabet => "dna");

print $seqObject->seq(), "\n\n";
print $seqObject->length(), "\n\n";
print $seqObject->display_id(), "\n\n";
print $seqObject->translate(-frame=>0)->seq(), "\n\n";

my $newseq = $seqObject->trunc(10, 100)->revcom();

print $newseq->translate(-frame=>1)->seq(), "\n";
```

A simple example:

script4.pl

```
#!/usr/local/bin/perl
use strict;
use warnings;
```

A Constructor:

```
my $seqObject = Bio::Seq->new (-seq => $seqstr,
                               -display_id => "myseq1",
                               -desc => "This is an example",
                               -alphabet => "dna");
```

```
print $seqObject->seq(), "\n\n";
print $seqObject->length(), "\n\n";
print $seqObject->display_id(), "\n\n";
print $seqObject->translate(-frame=>0)->seq(), "\n\n";

my $newseq = $seqObject->trunc(10, 100)->revcom();

print $newseq->translate(-frame=>1)->seq(), "\n";
```

Returned Data Type of the Method

```
$seqObject->seq();
```
string

```
$seqObject->subseq(10, 100);
```
string

```
$seqObject->trunc(10, 100);
```
object

```
$seqObject->translate(-frame=>2);
```
object

```
$seqObject->revcom();
```
object

```
$seqObject->revcom()->seq();
```
string

Do not print object reference

```
print $seqObject->revcom();
```

No

```
$seqObject_r = $seqObject->revcom();
print $seqObject_r ->seq();
```

Yes

```
print $seqObject->revcom()->seq();
```

Yes

A simple example:

script4.pl

```
#!/usr/local/bin/perl
use strict;
use warnings;
```

Some methods:

```
print $seqObject->seq(), "\n\n";
print $seqObject->length(), "\n\n";
print $seqObject->display_id(), "\n\n";
print $seqObject->translate(-frame=>0)->seq(), "\n\n";

my $newseq = $seqObject->trunc(10, 100)->revcom();

print $newseq->translate(-frame=>1)->seq(), "\n";
```


```
print $newseq->translate(-frame=>1)->seq(), "\n";
```

Continue the simple example:

6-FRAME Translation

```
print $seqObject->translate(-frame=>0)->seq();
print $seqObject->translate(-frame=>1)->seq();
print $seqObject->translate(-frame=>2)->seq();


my $seqObject_r = $seqObject->revcom();


print $seqObject_r->translate(-frame=>0)->seq();
print $seqObject_r->translate(-frame=>1)->seq();
print $seqObject_r->translate(-frame=>2)->seq();
```

Alternative ways to create the sequence objects

From network database (e.g. NCBI Genbank)

script5a.pl

```
#!/usr/local/bin/perl
use strict;
use warnings;

use Bio::Perl;

my $db = Bio::DB::GenBank->new();
my $seqObject = $db->get_Seq_by_acc('X78121');


print $seqObject->seq(), "\n\n";
print $seqObject->length(), "\n\n";
print $seqObject->display_id(), "\n\n";
print $seqObject->translate(-frame=>0)->seq(), "\n\n";

my $newseq = $seqObject->trunc(10, 100)->revcom();

print $newseq->translate(-frame=>1)->seq(), "\n";
```

:: designates subclass in class, DB is a subclass in Bio class, GenBank is subclass in DB subclass of Bio class.

Works similar as directory path.

Alternative ways to create the sequence objects

From file

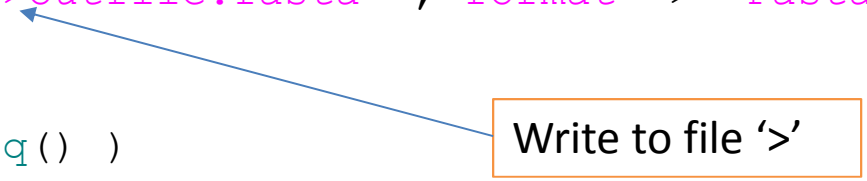
script5b.pl

```
#!/usr/local/bin/perl
use strict;
use warnings;

use Bio::SeqIO;

my $in = Bio::SeqIO->new(-file => "inputfile.fasta" ,-format => 'Fasta');

while ( my $seqObject = $in->next_seq() )
{
        print "\n-------------------------------\n";
        print $seqObject->seq(), "\n\n";
        print $seqObject->length(), "\n\n";
        print $seqObject->display_id(), "\n\n";
        print $seqObject->translate(-frame=>0)->seq(), "\n\n";

        my $newseq = $seqObject->trunc(10, 100)->revcom();

        print $newseq->translate(-frame=>1)->seq(), "\n";
}
```

Writing sequence to a file, changing sequence ids

script5c.pl

```
#!/usr/local/bin/perl
use strict;
use warnings;

use Bio::SeqIO;

my $in = Bio::SeqIO->new(-file => "inputfile.fasta" ,-format => 'Fasta');
my $out = Bio::SeqIO->new(-file => ">outfile.fasta" ,-format => 'Fasta');

my $n = 0;
while ( my $seqObject = $in->next_seq() )
{
        $n++;
        print $seqObject->display_id(), "\n";
        $seqObject->display_id("seq_$n");
        $seqObject->desc("sequence $n");
        $out->write_seq($seqObject);
}
$in->close();
$out->close();
```



Exercises

1. Write a script to read a DNA FASTA file and write it as protein sequence (after translation) into another fasta file. Use yeast_orf.fasta to as the input file.
   HINT: Modify script5c.pl.

2. Use String::Random to create 10 1kb random DNA sequences, and write to a new FASTA file.
   HINT: Use cpan to install locally String::Random. Create 10 random 1kb sequences in a loop and write them to a FASTAfile.

3. Add to mySeqAnalyzer.pm a method to print sequence stored in the object, name it seq(), same as in BioPERL.