

Computing Concepts for Bioinformatics



- Some PERL tricks
- Sequence Features
- Extracting features
- Displaying Features
- Working with Reports:
Bio::SearchIO

<http://amadeus.biosci.arizona.edu/~nirav>

Tricks: Strings & = ~ operator

- ~ = is known as the binding operator
- Can be used with m (match) and s (substitute)
- `$fun = ~ s/work/play/gi;`
- It can also be used for counting
- `$howmany = $fun = ~ s/work/play/gi;`
- The m operator returns TRUE (1) if there is a match or UNDEF (no value "") if mismatch
- `$count = ($found = ~ m/work/gi);`
- \$count can be 1 or UNDEFined
- The pattern can be a regular expression

Subroutines

- Many scripts have parts that need to be repeatedly executed
- Instead of writing the same code multiple times in the same script..
- Subroutines carry out the given task and return data and control back to the program
- The compartmentalization makes writing repeated tasks easy, safe and consistent

Subroutines

Flow of logic



Line 1

Line 2

Line 3

Line 4

Line 5



Sub find_pat

More subroutines

- Usually located at the bottom of the code (easier to locate)
- Declared by using `sub` before the name

```
sub gc_calc {  
    # Do stuff  
}
```
- To call/use them use the subroutine name `gc_calc()`;
- You can pass data/arguments to it `gc_calc($my_seq)`;
- Arguments are passed to subroutines using the array named `@_`
- Data is sent back using `return`:

```
return $gc_count;
```

gc subroutine

- To create the subroutine `gc_calc`:

```
sub gc_calc {  
  # Read 2 items ..sequence and length  
  my ($seqin,$len) = @_;  
  # Calculate GC  
  $gcount = $seqin =~ s/g/g/gi;  
  $ccount = $seqin =~ s/c/c/ci;  
  $gc_count = ($gcount+$ccount)/$len * 100;  
  # Return gc_count  
  return $gc_count;  
}
```

- Now you can call `gc_calc` anywhere in your program after reading a sequence and its length
`$gc_val = gc_calc($actual_seq,$length);`
- `$gc_val` now contains the calculated value

Flow Control: Next & Last

- The **next** and **last** operators allow you to modify the flow of your loops
- It is not at all uncommon to have a special case; you may want to skip it, or you may want to quit when you encounter it.
- The **next** operator would allow you to skip to the end of your current loop iteration, and start the next iteration.
- The **last** operator would allow you to skip to the end of your block, as if your test condition had returned false.

Next, last example

```
foreach $user (@users) {  
    # Skip nirav and susan  
    if ($user eq "nirav" or $user eq "susan") {  
        next; #Stays in the foreach loop; gets next  
item  
    }  
    # If gavin is found, do something, then stop  
looking  
    if ($user eq "gavin") {  
        print "Found the special user.\n";  
        # do some processing...  
  
        last;  
        #Jumps out of foreach loop  
    } else {  
        print "User not susan, nirav, or gavin\n";  
    }  
}
```

Sequence Features

- A *feature* is a region of interest in a specified nucleic or protein sequence.
- It has a specified start and end position.
- It has a name describing what type of feature it is, e.g. gene, CDS, promoter...
- Features may also explicitly or implicitly hold the name of the program or database that they are derived from.
- Feature Tables are groups of features, e.g. what you see in Genbank records.

Popular formats

- GenBank
- EMBL
- GFF
- Swissprot (not as popular)

Genbank

```
FEATURES             Location/Qualifiers
     source            1..297
                        /organism="Buchnera aphidicola"
                        /specific_host="Acyrthosiphon sp."
                        /db_xref="taxon:9"
     gene               1..297
                        /gene="thrB"
     CDS               <1..>297
                        /gene="thrB"
                        /EC_number="2.7.1.39"
                        /codon_start=3
                        /transl_table=11
                        /product="homoserine kinase"
                        /protein_id="BAA25384.1"
                        /db_xref="GI:3036933"
                        /translation="YLGGLQLILED SKIISQTIPNFKNWFWIVAWPGTKVPTAEARDI
LPK KYK KETCIKNSRYLAGFIHASYSQQPHLAARLMQDFIAEPYRIKLLPNYLY"
```

```
BASE COUNT      113 a      47 c      46 g      91 t
```

ORIGIN

```
   1 gttatctagg aggacttcag ctaatattag aagattcaaa aataataagt caaactattc
   61 caaattttaa aaattggttt tggatagtag cctggcctgg aactaaagtt cctactgcag
  121 aagcaagaga catactacca aaaaaatata aaaaagaaac atgtattaaa aatagtcggt
  181 atttagcagg ttttattcat gcttcataca gtcaacaacc tcacttagca gcacgattga
  241 tgcaagattt tatcgcagag ccatatcgta ttaaattatt acctaatat ttgtatg
```

```
//
```

EMBL

```
FH Key Location/Qualifiers
FH
FT source 1..297
FT /db_xref="taxon:9"
FT /sequenced_mol="cDNA to mRNA"
FT /organism="Buchnera aphidicola"
FT /specific_host="Acyrthosiphon sp."
FT CDS <1..>297
FT /codon_start=3
FT /db_xref="SWISS-PROT:O66132"
FT /transl_table=11
FT /EC_number="2.7.1.39"
FT /gene="thrB"
FT /product="homoserine kinase"
FT /protein_id="BAA25384.1"
FT /translation="YLGGLQLILED SKIISQTIPNFKNWFWIVAUPGTKVPTAEARDIL
FT PKKYKKETCIKNSRYLAGFIHASYSQOPHLAARLMQDFIAEPYRIKLLPNYLY"
XX
SQ Sequence 297 BP; 113 A; 47 C; 46 G; 91 T; 0 other;
gttatctagg aggacttcag ctaatattag aagattcaaa aataataagt caaactattc 60
caaattttaa aaattggttt tggatagtag cctggcctgg aactaaagtt cctactgcag 120
aagcaagaga catactacca aaaaaatata aaaaagaaac atgtattaaa aatagtcggt 180
atttagcagg ttttattcat gcttcataca gtcaacaacc tcatctagca gcacgattga 240
tgcaagattt tatcgcagag ccatatcgta ttaaattatt acctaattat ttgtatg 297
```

//

GFF

##end-DNA

##gff-version 2.0

##date 2002-07-10

##Type DNA 91

91 EMBL gene 1090 2874 0.000 + . Sequence "91.1" ; gene "or7251" ; label or7251 ; note "13"

91 EMBL CDS 1090 2874 0.000 + . Sequence "91.2" ; gene "or7251" ; label or7251 ; note "GENE MODEL=glimmer; " ; note "NR Hit:

GFF

- General Feature Format (GFF) also known as Gene-Finding Format
- Format for describing genes and other features associated with DNA, RNA and Protein sequences.
- The current version level of GFF is Version 2.
- <http://www.sanger.ac.uk/Software/formats/GFF/>

GFF description

- The data is plain text file named with .gff extension
- The fields are tab (\t) separated and records are terminated by newline (\n)

```
SEQ1  EMBL  atg    103    105    .    +    0
SEQ1  EMBL  exon   103    172    .    +    0
SEQ1  EMBL  splice5 172    173    .    +    .
SEQ1  netgene splice5 172    173    0.94  +    .
SEQ1  genie  sp5-20 163    182    2.3   +    .
SEQ1  genie  sp5-10 168    177    2.1   +    .
SEQ2  graill ATG     17     19     2.1   -    0
```

- <seqname> <source> <feature> <start> <end>
<score> <strand> <frame> [attributes]
- < > are required fields, [] is optional
- Use . for score/strand/frame if empty/blank
- Use of space is not permitted in names i.e. CAG 12 should be written as CAG-12 or CAG_12

More GFF

- Comments begin with # and end with newline
- Meta information is stored using

```
##  
##DNA <seqname>  
##acggctcggattggcgctggatgatagatcagacgac  
##...  
##end-DNA
```

- The **attribute** field follows the same tag and value standard flattened into a single line and delimited by ;
- Strings with spaces must be enclosed with " "
- Author "Nirav Merchant"; Boring 100; Salsa 1

```
AE014076 EMBL gene 5886 6761 0,000 + . Sequence "AE014076.14" ; gene "atpG" ; note "BUsg007"  
AE014076 EMBL CDS 5886 6761 0,000 + . Sequence "AE014076.15" ; product "ATP synthase gamma chain" ; gene "atpG"  
; protein_id "AAM67579.1" ; transl_table 11 ; codon_start 1 ; translation "MASKKEIKDQIISVTNTKKITKAMEMVAVSKMRKTEERMRLGRPYSEIIKKVIHHWLQGSLEYKHSYLE  
KRNDKRIGIIVSTDRGLCGSLNLTNLFKQVLFKIQNFAKINIPCDLILFGLKSLSVFKLYGSSIISVTNLGETPDL SKLIGSIKIILEKYQNNQIDRLFIAYNKFHNKLSQYPKISQLLPLYNEKNIFSNNKIKWDYLYEPESK  
LILDTLFDRIYESQIYQSLENIASEQAARMVAMKTATDMSGNRIKELQLIYNKVRQANITQELTEIVAGASAVSVD" ; db_xref "GI:21622899"
```

So what is GFF good for ?

- Easy to read
- Easy to parse
- Easy to exchange data between users and programs !
- Easy to visualize ! (more later)
- Sure ..its not XML ..but it works



Add a feature ..

- We can add features to a sequence using 2 methods
- Using the `Bio::SeqFeature::Generic` module, provide the start,end,tags and values ..

```
# Now add a new feature the hardway:
$snp = new Bio::SeqFeature::Generic (
    -start=>10 , -end=>11 , -primary=>"SNP",
    -source=>"RT-PCR", -tag=> { author=>"Nirav",junk=>"Booh yeah" }
);
```

```
#Now apply it to the sequence
$gb_seq->add_SeqFeature($snp);
```

- Build a GFF string and use that for adding a feature

```
#Now apply the feature a easy way using GFF
$gff_string = "TEST\tRT-PCR\tSNP\t12\t13\t.\t.\t.\tauthor Nirav;junk \t\"Booh Yeah\t\"";
$snp_gff = new Bio::SeqFeature::Generic ( -gff_String => $gff_string);
$gb_seq->add_SeqFeature($snp_gff);
```

- Once a feature has been defined (`$snp_gff`) we use the method `add_SeqFeature` to the given `Seq object` (`$gb_seq`)
`$gb_seq->add_SeqFeature($snp_gff);`

More about features

- Now that we have added the feature to the Seq object we can use SeqIO to write it to a file !
- Use the name `$file.out` to save the file, i.e. `small.gb` becomes `small.gb.out`

```
#Now that we have the new feature Create a genbank file  
# With the new feature  
$out_stream = new Bio::SeqIO (-file=>">$file.out", -format=>"GenBank");  
$out_stream->write_seq($gb_seq);
```

Use Bio::SeqFeature::Generic

```
#!/usr/local/bin/perl -w
# bpconvert.pl ver 1.0
#nirav@arl Nov 10 2005
# Converts genbank to fasta

use Bio::SeqIO;
use Bio::SeqFeature::Generic;

# Create a feature
$snp = new Bio::SeqFeature::Generic( -start=>10, -end=>11, -primary=>"SNP",
    -source=>"RT-PCR", -tag=> { author=>"Nirav", junk=>"test" });

$infile = Bio::SeqIO->new( -file => "<in.gb" , -format => GenBank);
$outfile = Bio::SeqIO->new( -file => ">out.embl" , -format => EMBL);

while($seq = $infile->next_seq) {

    $id = $seq->display_id();
    print "Working on $id\n";
    # Now write it
    $seq->add_SeqFeature($snp);

    $outfile->write_seq($seq);

}
```

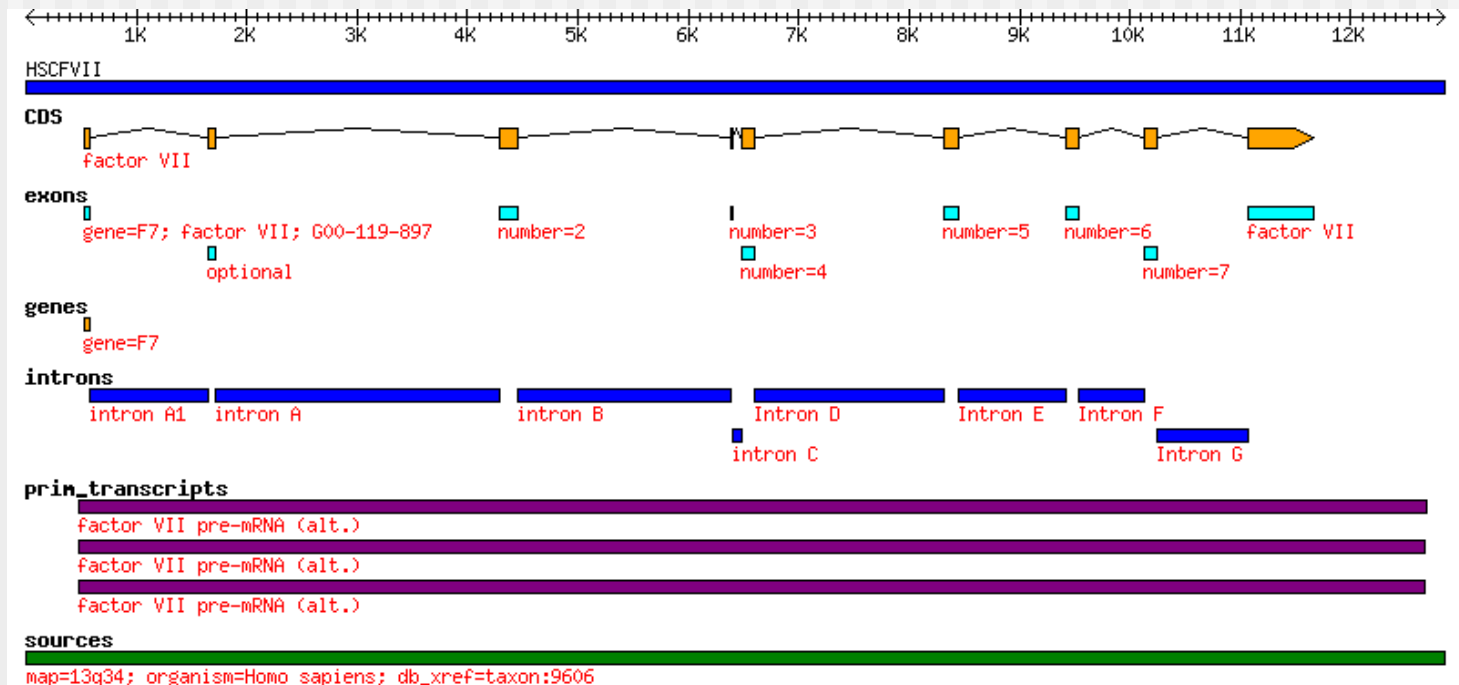
Displaying Features

- The BioPerl module `Bio::Graphics` gives us an easy way to graphically display features.
- The idea is that you create a `Bio::Graphics::Panel` object, then call the object's `add_track` method to add a graphical representation of a feature, with the desired colors, shapes, labels, etc.
- To save a `.png` file containing the graphic, you use the panel's `png` method:

```
print $panel->png();
```

Using Bio::Graphics

- Start by reading the Graphics HOWTO: <http://bioperl.org/HOWTOs/>



- There are also SeqIO and SearchIO HOWTOs, with example code

Working with Reports

- Just as the BioPerl SeqIO module gives you an easy way to read sequences from files of many formats, the SearchIO module can read many types of “Sequence Search” output such as BLAST, FASTA, HMMER.
- We will use SearchIO to parse a BLAST output file and extract some of the details – this is much easier than scanning a large BLAST output by eye!!

Using Command Line Options

- We have used GetOpt to retrieve command line *options* (e.g. *-name*)
- We will use this to specify the file to parse
- Example: Write a script using Bio::SearchIO to parse a BLAST output file. Be able to run it with *any* file without modifying the script:

```
my_parser -file blastfile1
```

```
...
```

```
my_parser -file blastfile1000
```

BLAST output

- Each sequence BLASTed is called a query
- Each match is called a subject
- Within each subject there can be several HSPs (High Scoring Pairs)

```
>gi|27717636|ref|XM_234908.1| Rattus norvegicus similar to mBLVR [Mus musculus]
(LOC314633), mRNA
      Length = 2199
```

```
Score = 161 bits (81), Expect = 3e-36
Identities = 81/81 (100%)
Strand = Plus / Minus
```

```
Query: 206  ttacagatggtggatacgcacatcagctgggcccgttcaacatcatagaagtgatgagt 265
          |||
Sbjct: 38930 ttacagatggtggatacgcacatcagctgggcccgttcaacatcatagaagtgatgagt 38871
```

```
Score = 149 bits (75), Expect = 1e-32
Identities = 75/75 (100%)
Strand = Plus / Minus
```

```
Query: 745  gttcgggtgctcttactccttggaaaccgcggtgggcaagaagctgatcgagcccctcac 804
          |||
Sbjct: 30287 gttcgggtgctcttactccttggaaaccgcggtgggcaagaagctgatcgagcccctcac 30228
```

Using Bio::SearchIO

From the BioPerl documentation:

```
use Bio::SearchIO;
```

```
# format can be 'fasta', 'blast'
```

```
$searchio = new Bio::SearchIO( -format => 'blastxml', -file  
=> 'blastout.xml' );
```

```
while ( my $result = $searchio->next_result() ) {
```

```
    while( my $hit = $result->next_hit ) {
```

```
        # process the Bio::Search::Hit::HitI object
```

```
            while( my $hsp = $hit->next_hsp ) {
```

```
                # process the Bio::Search::HSP::HSPI object
```

```
            }
```

```
        }
```

```
    }
```

Today's Assignment

- Go to <http://doc.bioperl.org/> to see which options to use to create a **SearchIO** object
- You will also want to look up the **Search::HSP** object to see which methods it has available
- Make your own class12 directory
- From `/home/student/2005/eeb/class12/` copy `srchio.pl` and `*.blastn` to your class12 directory and finish the script
- Test with both `.blastn` output files