Decision Making and Looping

Aims

- Understand if/elsif/else decision block
- Be able to use a foreach and while loop
- Understand and/or/not &&/||/!
- Using comparisons for decisions

Decision Making

- Programs are going to be pretty unhelpful long term if they don't at least try to do different things, dependent on data given.
- Contrary to what some people might think, computers are stupid, and will only do what you tell them. So, we need to tell them.
- Even perl programs

  bin/01-if_else.pl

Decision Making -if

- 'if' will be probably be the most called function you will ever write
- There are two ways of using it, the block form and the postfix form
- Let's take a look at them

Decision Making -if

- The block form

```
my $this = 'Hello';
my $that = 'Hello';
if ( $this ) {
  say $this;
}
```

- You can see that we are passing an array ( ) of arguments, the conditional, to *if*, which if true executes the block of code { }

Decision Making -if

- The postfix form

```
say $that if $that;
```

- Here we are passing an argument to *say*, and then also telling it to only *say* the argument if the conditional argument passed to *if* is true
- A word of advice, keep usage of postfix if to a minimum – **only** when very simple action and test

Decision Making - unless

- There is a negative version of if, unless. It has fallen out of favour, but can sometimes save you a lot of 'not's in your code

```
my $false;
unless ( $false ) {
  say 'I tested false';
}
say '0 is false' unless 0;
```

- Reason for not using, no elsunless block

Decision Making - else

- You are probably going to want to do something *else* if the *if* wasn't true.

```
my $seq = 'CCGGATCACTATGACCTGCTTTCG';
if ( $seq =~ m/ATG/ixms ) {
  say "$seq contains Methionine";
} else {
  say "$seq contains that damned cat again";
}
```

- This matches, so only the *if* block happens

Decision Making - else

```
my $clone = $seq;
$clone =~ s/atg/cat/gim;
if ( $clone =~ m/ATG/im ) {
  say "$clone contains Methionine";
} else {
  say "$clone contains that damned cat again";
}
```

- We have removed the ATG, so since it doesn't find that, we execute the else block

Decision Making - elsif

- So we have the possibility of doing 2 things, but can we do multiple options
  - Yes
- elsif allows us to put in extra optional truth statements

Decision Making - elsif

```
$clone =~ s/dog/aaa/gim;
if ( $clone =~ m/ATG/im ) {
  say "$clone contains Methionine";
} elsif ( $clone =~ m/dog/im ) {
  say "$clone from foreign planet, or this isn't DNA";
} elsif ( $clone =~ m/aaa/im ) {
  say "$clone has my favourite codon in it";
} else {
  say "$clone contains that damned cat again"; }
```

Decision Making - elsif

- You can have as many or as few *elsif* blocks as you like
- Each must have a condition and a block
- The tests stop as soon as a condition is true

```
$clone .= 'atg';
if ( $clone =~ m/ATG/im ) {
  say "$clone contains Methionine";
} elsif ( $clone =~ m/dog/im ) {…} … # never tested
```

Looping

- There are 4 loop keywords. *while*, *until*, *for* and *foreach*.
  - *while* and *until* are conditional based
  - *for* is iteration and conditional based (and less commonly used)
  - *foreach* is iteration based
- All can be rewritten to do the same as each other

bin/02-while_foreach.pl

Looping - while

```
my $true_comparison = 10;
my $count = 0;
while ( $count < $true_comparison ) {
  say 'while version : ' . $count;
  $count++; # increment count
}
```

- The syntax is similar to an *if* statement. The *while* keyword, followed by a conditional, and then a block of code

Looping - while

- The difference comes that *if* only executes it's block once.
- Every time the end of the block is reached, *while goes back to see if the conditional is still true,* and then re-executes until the conditional is false

```
while version : 0
...
while version : 9
```

Looping - until

- *until* is related to *while* in the way *unless* is to *if*

```
$count = 0;
until ( $count == $true_comparison ) {
  say 'until version : ' . $count;
  $count++;
}
```

- *until* keeps executing the block, whilst the conditional is false
- *until* is the least favoured of the two

Looping – while and until

- A warning – don't do the following, unless you have very good reason

```
while (1) {
…
}
until (0) {
…
}
```

- Both will never stop looping!

Looping – foreach

- When you have an array of elements that you want to process, you can use *foreach*

```
my @codons = qw{AAA AAC AAG AAT ACA ACC
ACG ACT AGA AGC AGG AGT ATA ATC ATG ATT};
foreach my $codon ( @codons ) {
  say 'foreach version : ' . $codon;
}
```

- Foreach element (codon) in the array, assign to the $codon variable, execute the block with that variable

Looping – for

- *foreach* doesn't provide you with the index of the element in the array
- To do this, we use *for*

```
for (my $i = 0; $i < @codons; $i++ ) {
  say 'for version : ' . $codons[$i];
}
```

- This a combination of a conditional and an iteration

Looping – for

```
for (my $i = 0; $i < @codons; $i++ ) {}
```

- my $i = 0;
  - Start index variable with first index
- $i < @codons;
  - Conditional, keep going whilst $i < array length
- $i++
  - Increment index variable after each execution

Looping – for and foreach

- *for* and *foreach* are actually interchangeable due to historical wishes to be similar to other languages
- However, it is more common to see *for* replace *foreach* than the other way around

Looping – with instead of foreach

- You can write the code to use any one of the four to do all of the jobs e.g.

```
while ( my $codon = shift @codons ) {
  say 'while method of looping over array: ' . $codon;
}
```

- However, be warned, this would empty the @codons array, as it's 'shift'ing the codon
- The conditional fails when there is nothing to assign

Boolean Logic Operators

- Often, you are going to want to have more than one condition to be true (or false) in order for an action to occur
- We have access to the usual Boolean Logic operators to help here
  - *&& / and* = and
  - *|| / or* = or
  - *! / not* = not

bin/03-and_or_not.pl

Boolean - &&/and

- If you want an action when two or more conditions are true

```
my $true = 1;
my $also_true = 2;
my $false = 0;
my $also_false = 0;
if ( $true && $also_true ) {
  say 'true && also_true is true';
}
```

Boolean - &&/and

```
unless ( $true && $false ) {
  say 'true && false is not true'
} # note: using unless
unless ( $false and $also_false ) {
  say 'false and also_false not true';
}
```

- *&&* (or *and*) is looking for both parts to be true, and then makes the overall conditional true if all parts are

Boolean - ||/or

- If you want an action when at least one of the conditions is true

```
if ( $true || $also_true ) {
  say 'true || also_true is true';
}
if ( $true || $false ) {
  say 'true || false is true';
}
```

Boolean - ||/or

```
if ( $false or $true ) {
  say 'false or true is true';
}
unless ( $false or $also_false ) {
  say 'false or also_false is false';
}
```

- || (or *or*) is looking for either part to be true, and then makes the overall conditional true
- Shorts out at the first one it finds to be true

Boolean - !/not

- You may want to invert the truthfulness of something.

```
if ( ! $false ) {
  say 'false has been made true with !';
}
if ( not $also_false ) {
  say 'also_false has been made true with not';
}
```

- *!/not* always turns values to undef or 1

Boolean – Complex Conditionals

- We can use Boolean Logic to create complex conditionals
- Use () to group parts of conditionals

```
if ( $false
      ||
    ( $true && $also_true )
  ) {
  say 'I got to true in the end';
}
```

Boolean – Complex Conditionals

- does sequence contain a methionine and stop (may be a gene)

```
my $sequence = 'CCGGATCACTATGACCTG';
if ( $sequence =~ m/atg/im &&
    ( $sequence =~ m/TAA/im || $sequence =~ m/TAG/im || $sequence =~ m/TGA/im )
  ) {
   say 'contains a methionine codon and a stop codon';
}
```

Boolean – Complex Conditionals

- Or contains Methionine but not Glutamine

```
my $sequence = 'CCGGATCACTATGACCTG';
if ( $sequence =~ m/atg/im
     &&
    ! ( $sequence =~ m/CAA/im || $sequence =~ m/CAG/im )
  ) {
   say 'contains a methionine codon and not a glutamine codon';
}
```

Boolean – Complex Conditionals

- Note: either use &&/||/! or and/or/not, don't mix and match unless you have reason – they have different precedences and you will more than likely introduce a but
- So, as a rule of thumb, pick one, and stick with it.

Boolean – Assignment

- The || operator is really useful in variable assignment

```
my $truth = $false || $true;
say 'truth: ' . $truth;
```

- line up many, cut out as soon as a true value found

```
$truth = $false || $also_true || 'I am a truthful string';
```

- With the above you can give preferential values to a variable

Boolean – Assignment

- ||= (or assign) is an exceptionally common piece of code

```
$truth ||= $true;
say 'truth: ' . $truth;
$truth = 0; # remember, 0 is false
$truth ||= {};
say 'truth: ' . $truth;
```

- We'll see more of this sort of thing a lot when we explore functions and objects

Comparisons

- We have seen straight forward depending on truth, and regexes as our conditionals
- Don't forget, we can use any comparison tests as a conditional
  - Equal
  - Not Equal
  - Less than
  - Greater than...

bin/04-comparisons.pl

Comparisons - Equal

```
if ( 10 == 10 ) {
  say 10 == 10;
}

if ( 'cat' eq 'cat' ) {
  say 'cat' eq 'cat';
}
```

Comparisons – Not Equal

```
if ( 10 != 20 ) {
  say 10 != 20;
}

if ( 'cat' ne 'dog' ) {
  say 'cat' ne 'dog';
}
```

- you can continue for >< lt gt => <=

Decision Making and Looping

- Decisions and looping are two fundamental parts of programming.
- You need to instruct the program how to deal with the data given, and if there are choices to make, what rules to use.
- Hopefully, we have covered the syntax here so that you can at least write (or understand) some of these, and have some decisions made.
- The next workshop should help

Workshop
Let's Write a Script

Let's Write A Script

- Folder 08-lets_write_script
- All the instructions are in file

  bin/01-sequence_play.pl
- Along with some sequences already assigned to variables
- Work through each of the questions in the file (doing your work in it).
- There are challenges relating to all the 4 sections we have covered.
- *there is more than one way to do it*

Answers

- 1) lower case each string

```
say lc $sequence_1;
say lc $sequence_2;
say lc $sequence_3;
```

Answers

- 2) do any of the sequences contain methionine (atg)?

```
if ( $sequence_1 =~ m/atg/ixms ) {
  say 'sequence 1 contains methionine';
}
```

Answers

- 3) do any of the sequences contain a stop codon (taa,tag,tga)?

```
if ( $sequence_1 =~ m/taa/im || $sequence_1 =~ m/tag/im || $sequence_1 =~ m/tga/im ) {
  say 'sequence 1 contains stop';
}
```

Answers

- 4) what is the %age GC content of each string?

```
foreach my $seq ( $sequence_1, $sequence_2, $sequence_3 ) {
  say ( ( ( $seq =~ tr/GCgc/GCgc/ ) * 100 ) / length $seq );
}
```

Answers

- 5) Let the user enter a codon to exchange methionine for in sequence_1, and switch it

  Bonus points for challenging a non-dna base letter

Answers

```
print 'Enter a codon you would like to swap methionine (atg) for: ';
my $user_codon = <STDIN>;
chomp $user_codon;
while ( $user_codon =~ m/[bd-fi-su-z]/im ) {
  print 'The codon you entered contains non_dna bases. Please try again: ';
  $user_codon = <STDIN>;
  chomp $user_codon;
}
$sequence_1 =~ s/atg/$user_codon/im;
say $sequence_1;
```

Answers

- 6) BONUS: assuming sequence is always read 5' to 3', can you print out the reverse complement in the order it is read of each sequence

```
foreach my $seq ( $sequence_1, $sequence_2, $sequence_3 ) {
  $seq =~ tr/ACGTacgt/TGCAtgca/;
  $seq = reverse $seq;
  say $seq;
}
```