Week 2

18 January

• Reading for this week

• 24H Hour
  ○ Hour 3 - Controlling the Program's Flow
  ○ Hour 4 - Stacking Building Blocks: Lists and Arrays

• PP
  ○ ch 6: 25-29 - Arrays
  ○ ch 8: 49-59 - Conditional Constructs
    56-60 – Loop conditional constructs
Basic Perl

Arrays

- Identified by the @ symbol
  - @a and $a are completely different (they have different name spaces)
- An ordered set of scalars
- Initialized using parentheses with the elements separated by commas

```perl
# Define nucleotide bases
@base = ( "A", "C", "G", "T" );

foreach $nuc ( @base ) {
    print "$nuc is a valid base\n";
}
```
Basic Perl

Arrays

• Many languages require you to specify the size of an array. Perl extends the array as values are assigned
  ○ What does the following do?

```perl
$residue[4] = 'W';
$i = 0;
foreach $amino_acid ( @residue ) {
    print "amino acid $i = $amino_acid";
    $i++;
}
```

• Adding an element to an array immediately creates all the elements up to that one
Arrays

• DWIM: Do you need quotations?
  ○ *if you include variables that need interpolation*
  ○ *you are using strings that include punctuation (,@$"';) or spaces*

```perl
@residues = ( A,C,D,E,F,G,H,I,K,L,M,N,P,R,S,T,V,W,Y,Z );
$count = 0;
foreach $amino_acid (@residues) {
    $count++;
    print "$amino_acid ";
    unless ( $count % 5 ) { print "\n"; }  # note use of mod operator
}
```

Is this the right number or residues?

```perl
@residues = ( A,C,D,E,F,
              G,H,I,K,L,
              M,N,P,R,S,
              T,V,W,Y,Z );
$count = 0;
foreach $amino_acid (@residues) {
    $count++;
    print "$amino_acid ";
    unless ( $count % 5 ) { print "\n"; }
}
```
Arrays - Style

- make it easy to read and verify when you define arrays

```perl
@codon = (AAA, AAC, AAG, AAU, ACA, ACC, ACG, ACU, AGA, AGC, AGG, AGU, AUA, AUC, AUG, AUU,
          CAA, CAC, CAG, CAU, CCA, CCC, CGG, CGU, GGA, GCC, GGG, GGU, GUA, GUC, GUG, GUU,
          UAA, UAC, UAG, UAU, UCA, UCC, UCG, UCU, UGA, UGC, UGG, UGU, UUA, UUC, UUG, UUU );
```

```perl
@codon = (AAA, AAC, AAG, AAU,
          ACA, ACC, ACG, ACU,
          AGA, AGC, AGG, AGU,
          AUA, AUC, AUG, AUU,
          CAA, CAC, CAG, CAU,
          CCA, CCC, CGG, CGU,
          CGA, CGC, CGG, CGU,
          CUA, CUC, CUG, CUU,
          GAA, GAC, GAG, GAU,
          GCA, GCC, GCG, GCU,
          GGA, GGC, GGG, GGU,
          GUA, GUC, GUG, GUU,
          UAA, UAC, UAG, UAU,
          UCA, UCC, UCG, UCU,
          UGA, UGC, UGG, UGU,
          UUA, UUC, UUG, UUU );
```
Basic Perl

Arrays

• Iterating over arrays
  ○ Using foreach
    - Most common way to make loops with arrays

  ○ By removing items until the array is empty
    - (will discuss after we talk about push/pop/shift/unshift)
Basic Perl

Arrays

• Example: Print an array of letters in blocks of three

```perl
# print residues in blocks of three
@residues = ( A,C,D,E,F,G,H,I,K,L,M,N,P,R,S,T,V,W,Y,Z );

$count = 0;
foreach $amino_acid (@residues) {
    $count++;
    print "$amino_acid ";
    unless ( $count % 3 ) { print "\n"; }  # note use of mod operator
}
```

• How would I interactively get a number to print per line?
Basic Perl

Arrays

• Example - print a designated number per row

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

@residues = ( A,B,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,X,Y,Z );

# read number of symbols per line from terminal
print "How many residues per line?\n\n";
$aa_per_line = <>;

# print out in desired format

print "\nAmino Acid Residues\n";
$count = 0;
foreach @amino_acid ( @residues ) {
  $count++;
  print "$amino_acid ";
  unless ( $count % $aa_per_line ) { print "\n"; }
}

print "\n";
```
Basic Perl

Arrays

@list1 = (1, 2, 3, 4);

which is right?
   @list1 = [1, 2, 3, 4];
   @list1 = (1, 2, 3, 4);  
   wrong  
   right

$list1[2] = 

$x = @list1;
$x = ?

@b = ("one", "two");
@list1 = @b;

foreach $i (0 .. $#list1) {
    print "$i list1[$i] = $list1[$i]\n";
}

prints what?

pop @list1;
$x = pop @list1;
$x = ?

shift @list1;
$x = shift @list1;
$x = ?

$x = pop @list1;
unshift @list1, $x;

foreach$i (0 .. @list1) {
    print "$i list1[$i] = $list1[$i]\n";
}

prints what?

@list1 = [1, 2, 3, 4];
@list1 = (1, 2, 3, 4);

wrong  
right
Basic Perl

Arrays - Style

• use qw (mnemonic – quote word) to simplify array definitions
  ○ qw (quote word) treats a text as a list in which each word is separated by a comma, i.e.
  ○ @names = qw{ able baker charlie};
    is the same as
  @names = ( "able", "baker", "charlie" );
• the first character (your choice) following qw defines the beginning and end of list (list delimiter)
  ○ @names = qw( able baker charlie);
  ○ @names = qw{ able baker charlie};
  ○ @names = qw' able baker charlie';
  ○ @names = qw/ able baker charlie/;

@names = qw( able
    baker
    charlie
  );
Basic Perl

Arrays

• Special features of arrays
  ○ Parts of arrays are called slices. Uses @ prefix and continuous or discontinuous range in [ ]
    - Range operator @names = @directory[ 13 .. 23]
    - Comma @names = @directory[ 13, 23]

```perl
@directory = ( Michael, Aditi, Damion, Doug, Emre, Greg, Hao );
@names = @directory[ 2..4 ];
print "@names\n";
```
Basic Perl

Arrays

• Printing out arrays
  ○ Print entire array
  ○ Print entire array in double quotes
  ○ Print each element in a foreach loop
  ○ Print each element by index

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

@names = ( "Michael", "Reazur", "Greg", "Ying", "Prasad" );

# print the entire array
print "entire array - ";
print @names;

# print in quotes (interpolate)
print "\n\ninterpolated array - ";
print "@names\n";

# print in a loop
print "\nforeach loop\n";
$n = 0;
foreach $name ( @names ) {
  $n++;
  print "$n $name\n";
}

# print in loop using an index
print "\nindexed loop\n";
foreach $i ( 0..$#names ) {
  print "$i $names[$i]\n";
}
```

entire array - Michael Reazur Greg Ying Prasad
interpolated array - Michael Reazur Greg Ying Prasad
foreach loop
1 Michael
2 Reazur
3 Greg
4 Ying
5 Prasad
indexed loop
0 Michael
1 Reazur
2 Greg
3 Ying
4 Prasad
Basic Perl

Arrays

- Access functions for arrays
  - `push` - add a new element to the end of the array
  - `pop` - remove the last element from the array
  - `shift` - remove the first element from an array
  - `unshift` - opposite of shift, adds a new element to beginning of array

- Think of an array as a stack of books. Push adds a book to the top, pop takes it back off, shift pulls the bottom book out and the stack slides down.

```
0 1 2 3
A C G T
  push
0 1 2 3
A C G T
  shift
0 1 2 3
C G T
0 1 2 3
A C G
  pop
0 1 2 3
A C G
  unshift
0 1 2 3
T A C G
```

Basic Perl

Arrays

- pop and shift
  - remove an element from an array from the array
  - stacks
    - last in - first out lists (LIFO) use pop
  - queues (theater line order)
    - first in - first out lists (FIFO) use shift

```perl
# demonstrate pop
@base = ( "A", "C", "G", "T" );

$one_base = pop @base;
print "the last base was $one_base\n";
print "the remaining bases are @base\n";

# demonstrate shift
@base = ( "A", "C", "G", "T" );

$one_base = shift @base;
print "the last base was $one_base\n";
print "the remaining bases are @base\n";
```
Basic Perl

Arrays

• Iterating until a array is empty
• a little more complicated, useful when your array is a (possibly variable) list of things that have to be done – a "to do" list

```perl
# iterate until array is empty
@base = ( "A", "C", "G", "T" );

while ( @base ) {
    $b = pop @base; # remember the value of @base is the number of items in the array
    print "base $b\n";
}
```
Arrays

• Stacks and Queues

○ Say I want to calculate the length of each sequence as a fraction of the total length, i.e.,
  - 1 aaaa 40%
  - 2 bbbbbbb 60%

○ I can't do anything until I have the total, so I save in a queue/stack

```perl
#!/usr/local/bin/perl
$total = 0;
while ( $line = <> ) {
    $len = length( $line );
    print "$len $line";
    push @sequences, $line;
    push @lengths, $len;
    $total += $len;
}
```

Four ways to access the array of lines:
1. From beginning using shift
2. From beginning using index (or foreach)
3. From end using pop
4. From end using index
Arrays

• Table of amino acid one and three letter codes

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

@symbols = qw/ ala asx cys asp glu phe gly his ile met asn pro gln arg ser thr val trp unk tyr glx /;

@residues = ( A,B,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,X,Y,Z );

# read number of symbols per line from terminal
print "How many residues per line?\n\n";
$aa_per_line = <>;

# print out in desired format
print "Amino Acid Residues\n";
$count = 0;
foreach $i ( 0..$#symbols ) {
    print "$residues[$i] $symbols[$i] ";
    unless ( $i % $aa_per_line ) { print "\n"; }
}
print "\n";
```

How can I sort by three letter code? Is there an easier way to prevent errors?
Basic Perl

Sorting

• One of the most important computer activities – closely related to searching
• perl supports very sophisticated sorting, for now I'll keep it simple

• sort @array will sort the values of an array alphabetically
  o to save the sorted version you have to store in a new array

```perl
$symbols = qw/ ala asx cys asp glu
  phe gly his ile met
  asn pro gln arg ser
  thr val trp unk tyr
  glx
  /;

# new array in alphabetical order
@sorted_symbols = sort @symbols;

# alphabetize without copying the array
foreach $aa ( sort @symbols ) {
  print "$aa\n";
}
```
Basic Perl

Arrays

• Sorting arrays
  ○ `sort @list` sorts the contents of the array `@list` by value
  ○ implicitly sorts using the `cmp` (string comparison) operator which sorts alphabetically
  ○ can sort in other ways, we will discuss later

```perl
@a = ( 3, 2, 1 );
@b = sort( @a );           # @b is now ( 1, 2, 3 )

@c = ( 5, 10, 15, 20 );
@d = sort @c;
print "sorted @d\n";

@names = ( "dennis", "abigail", "clark", "bertha" );
foreach $person ( sort @names ) {
    # print in alphabetical order
    print "$person\n";
}
```