Scripting Languages

- Examples: Perl (Documentation: http://www.perl.org/docs.html) and Python (Documentation: http://docs.python.org/)

- Advantages:
  - Easy to code
  - Tons of prewritten scripts/modules developed for the bioinformatics community (e.g. BioPerl & BioPython)

- Not required for this class, but they are convenient for many applications discussed

- The best way to learn any language is to write some small scripts in it
Perl Basics: Command Line Input

• To invoke a Perl script from command line, type the following:

    perl PATH/FILE_NAME [ARG1 ARG2 ...]

• ARGs are optional parameters which are fed as an array of inputs into the program (more later)
Perl Basics: Syntax and Headers

- All variables should start with identifiers
  - $ for scalars
  - @ for arrays
  - % for hashes
- All lines should end with a semicolon (;)
- A script should begin with:
  ```
  #!PATH
  ```
  where `PATH` is the location of the Perl interpreter files
  - This is usually `/usr/bin/perl`
- To use premade modules, type `use MODULE_NAME`
Perl Basics: Variables

• Variables can be assigned values using the assignment operator (=)
  • $i = 3;
  • $j = 4;
  • $str = “This is a Perl string”;

• Standard operations can be performed on integers
  • $k = $i + $j; #$k = 7
Perl Basics: Strings

- Single quotes mean string is taken literally, double quotes mean string is interpreted

- Useful functions
  - Size
    - length($str)
  - Substring
    - substr($str, 3, 5)#takes the substring starting at index 3 with length 5
  - Concatenation
    - $str = "con"."cat"."enate"; #$str = “concatenate”
  - Splitting a string up
    - my @line = split(/a/, $str);#@line is an array containing 3 parts: “conc”, “ten”, and “te”
    - Any regular expression can fit in between the slashes in the split function
Perl Basics: Arrays

- Can be instantiated during use
- Array labeled with '@', while elements in the array labeled with '$' (e.g. $row_line[$0] is the 1st element in @row_line, which is “conc”)
- Maximum Index: $#array
- Size: scalar(@array)
Perl Basics: Conditionals and Loops

- Conditional:

  ```perl
  if(BOOLEAN){
    STATEMENTS;
  }
  elsif(BOOLEAN){
    STATEMENTS;
  }
  else{
    STATEMENTS;
  }
  ```

- For Loops:

  ```perl
  for($i = 1; $i <= $size; $i++){
    print "$i\n";
  }
  OR
  for $i (1 .. $size){
    print "$i\n";
  }
  ```

  Other types of loops occur too (e.g. while)
Perl Basics: File I/O

- To open a stream:
  ```perl
  open(F0, STRING_LOCATION);
  ```
- Start the STRING_LOCATION with '>' for write/overwrite, '>>' for write/append, or '<' for read (default – if nothing is placed before the file name)
- To read all at once into an array:
  ```perl
  my(@lines) = <F0>;
  ```
- To read line-by-line:
  ```perl
  while(<F0>){
    $line = $_;
    chomp($line);
    #Process as required
    print "$line
";
  }
  ```
- To write to a file, simply insert the stream between print and the text
  ```perl
  (e.g. print F1 "$line
";
  ``` #F1 is a write stream opened similarly to above)
Sample Script

Called from command line with:
perl ./PerlTest.pl TestArraySizes.txt

#!/usr/bin/perl

open F0, $ARGV[0]; #read in argument (path for file input) and open a stream to this path called F0
my @lines = <F0>; #read all the lines in F0 into an array called lines

my @row_line = split(/:/, $lines[0]); #break up the first line into two parts: "Number of Rows" and "5\n"
my @col_line = split(/:/, $lines[1]); #break up the second line into two parts: "Number of Columns" and "6\n"

my $row_size = $row_line[1]; #the second part of the row_line and col_line arrays has the size
my $col_size = $col_line[1];

chomp($row_size); #chomp gets rid of the \n new line character at the end of this,
chomp($col_size); #returning just "5" and "6" respectively

print "Number of Rows: $row_size\nNumber of Columns: $col_size\n"; #print to screen
#print '"$row_size $col_size"'; #single quotes aren't interpreted

for my $j (0; $j < $col_size; $j++) { #if you're on the first row, then set up the sum to be the column value
    $sum_matrix[0][$j] = $j;
}

for my $i (1; $i < $row_size; $i++) { #if you're on the first column, then set up the sum to be the row value
    $sum_matrix[$i][0] = $i;
}

for my $i (1 .. $row_size - 1) { #for all indices from 0 to (5-1) [all rows]
    for my $j (1 .. $col_size - 1) { #for all indices from 0 to (6-1) [all columns]
        $sum_matrix[$i][$j] = $sum_matrix[$i][$j - 1] + 1; #set up the sum to be the element directly to the left's value + 1
        if ($sum_matrix[$i][$j] != $sum_matrix[$i - 1][$j] + 1) { #this should equal the element directly above's value + 1
            print "Error: $sum_matrix[$i][\$j] does not equal ($sum_matrix[$i-1][\$j]+1)\n";
        }
    }
}

open F1, "~/home/nitin/PerlScriptOutput.txt"; #open an output stream and label it as F1

for my $i (0 .. scalar(@sum_matrix)) { #for all rows (scalar refers to the maximum element)
    for my $j (0 .. scalar(@$sum_matrix[1])) { #for all columns ($also refers to size)
        print F1 "$sum_matrix[\$i][\$j] \n"; #print the value of that index of the matrix to the F1 stream
    }
}
print F1 \n; #create a new line between rows
Python Basics: Syntax and Headers

• Spacing is important – off by one can throw off the whole program

• Should begin with:

  ```
  #!<PATH>
  ```

  where `<PATH>` is the location of the Python interpreter files (usually `/usr/bin/env python`)

• To use premade modules, type `import <MODULE_NAME>`
Python Basics: Command Line Input

- To invoke a Python script from command line, type the following:

  ```
  python <PATH/FILE_NAME> [ARG1 ARG2 ...]
  ```

- ARGs are optional parameters which are fed as an array of inputs into the program (more later)
Python Basics: Variables

- Variables can be assigned values using the assignment operator (=)
  - \( i = 3 \)
  - \( j = k = 4 \)
  - \( \text{str} = \text{"This is a Python string"} \)
- Standard operations can be performed on integers
  - \( l = i + j; \# l = 7 \)
Python Basics: Strings

- Immutable (can't be changed directly, but can be modified and stored)

- Useful functions
  - Size
    - `len(str)`
  - Substring
    - `str[3:7]` #takes the substring starting at index 3 with length 5
  - Concatenation
    - `str = "con" + "cat" + "enate"` #str = “concatenate”
  - Splitting a string up
    - `line = str.split("a");` #line is an array containing 3 parts: "conc", "ten", and "te"
Python Basics: Arrays

- Should be instantiated prior to usage
- Access: array[i] is the ith element in array
- Size: len(array)
Python Basics: Conditionals and Loops

- Conditional:
  
  ```python
  if (BOOLEAN):
    STATEMENTS
  elif (BOOLEAN):
    STATEMENTS
  else:
    STATEMENTS
  ```

- For Loops:
  
  ```python
  for i in range(1, size):
    print '%d' % i
  ```

- Other types of loops occur too (e.g. while)
Python Basics: File I/O

- To open a stream:
  ```python
  F0 = open(STRING_LOCATION)
  ```
- Have a second parameter with 'w' for write/overwrite, 'a' for write/append, or 'r' for read (default – if nothing is placed before the file name)
- To read all at once into an array:
  ```python
  lines = F0.readlines()
  ```
- To read line-by-line:
  ```python
  line = F0.readline()
  for line in F0:
    #Process as required
    print line
  ```
- To write to a file,
  ```python
  print >>F1 "line" #F1 is a write stream opened similarly to above
  OR
  F1.write("line")
  ```
import sys

F0 = open(sys.argv[1])  # read in argument (path for file input) and open a stream to this path called F0
lines = F0.readlines()  # read all the lines in F0 into an array called lines

row_line = lines[0].split(":")  # break up the first line into two parts: "Number of Rows", and "5\n"
col_line = lines[1].split(":")  # break up the second line into two parts: "Number of Columns", and "6\n"

row_size = int(row_line[1])  # the second part of the row_line and col_line arrays has the size
col_size = int(col_line[1])

print "Number of Rows: %d" % row_size
print "Number of Columns: %d" % col_size  # print to screen

try:
    # instantiate the sum_matrix array before using it
    # faster if numpy module is there
    from numpy import *
    sum_matrix=zeros((row_size,col_size))
except ImportError:
    # else, use a list
    sum_matrix=[]
    for i in range(row_size):
        sum_matrix+=[[0]*col_size]

for j in range(col_size):  # if you're on the first row, then set up the sum to be the column value
    sum_matrix[0][j] = j

for i in range(1, row_size):  # if you're on the first column, then set up the sum to be the row value
    sum_matrix[i][0] = i

for i in range(1, row_size):  # for all indices from 0 to (5-1) [all rows]
    for j in range(1, col_size):  # for all indices from 0 to (6-1) [all columns]
        sum_matrix[i][j] = sum_matrix[i][j-1] + 1;  # set up the sum to be the element directly to the left's value + 1
        if(sum_matrix[i][j] != sum_matrix[i-1][j] + 1):  # this should equal the element directly above's value + 1
            print "Error: %d does not equal %d" % (sum_matrix[i][j], sum_matrix[i-1][j] + 1)

F1 = open("/home/nitin/PythonScriptOutput.txt", "w"):  # open an output stream and label it as F1

for i in range(len(sum_matrix)):  # for all rows (scalar refers to the maximum element)
    print >>F1, ".join("%s" % el for el in sum_matrix[i]),  # print the value of that index of the matrix to the F1 stream
F1.write('\n')
Questions?