Perl

By Terrence Tan, James Epstein, Naomi Staley, and Thomas Williams

What is Perl

- Perl is a general purpose programming language
- Perl is both dynamic and interpreted
- Perl is used for web design, file management, and general object oriented programming

Perl Syntax: Variables

```
my $sword = "victory"; # Declaration of a scalar variable holding a string
my $bow = 9; # Declaration of a scalar variable holding an integer
my @axe = ("Frodo","Sam","Merry","Pippin"); # Declaration of an array variable
```

- Three types of variables: scalar, array, hash
- Scalars hold basic data types such as integers, floats, and strings
- Arrays hold an ordered group of other variables
- Hashes hold a set of key-value pairs

Perl Syntax: Subroutines and Classes

```perl
package Node; #from here to next package call is Node

sub new
{
    my $class = shift;
    my $self = {
        _next  => undef, #nodes hold two fields, _next points to next node
        data => undef,   #data holds a scalar/reference to non-scalar
    };
    bless $self, $class;
    return $self;
}
```

- Methods are defined using the "sub" keyword
- Classes use the keyword "package"
- Classes are set using the keyword "bless"

Perl Syntax: Statements

```
until($start > $end) { #if the key isn't found, the start index gets         higher than the ending one
    my $mid = int(($end + $start)/2); #the mid is an index, so it must be an int.

    if ($key > $array[])   #if the key is larger than midpoint value
        {$start = $mid+1;} #redo, but starting with the midpoint \+1
    elsif ($key < $array[]) #if the key is smaller,
        {$end = $mid \-1;} #redo with the endpoint as the mid-1
    elsif ($key == $array[]) #if the key is the midpoint value,
        {return $mid;} \#return the midpoint
    }
return \-1;
}
```

- "Java-style" for loops
- Standard while loops, plus an additional "until" loop
- "if", "elsif" and "else" keywords for if statements

Scoping

```
test(1,2);

sub test{
    #the variables $i and $j only exist in test
    #the global value of i gets written over by the local value
    my ($i, $j) =@_;
    print "value of i: ", $i, "\n";
    test2();
    $id = 3;
    print "id changed ", $id, "\n";
    print "value of c: ", $c, "\n";
}

sub test2{
    #since this is a local variable it can be accessed from any other
    #function that calls it. (like test for example)
    local $id = 2;
    print "id ", $id, "\n";
}
```

Reference Passing

```
my @testArray = (1,1,2,4,8,9,10,13,18,21,21,36); # Pre-sorted 12 element array
print("Original Array: @testArray\n");

my $target1 = binarySearch(\@testArray,1,0,scalar(@testArray)-1); # Should return 0
my $target9 = binarySearch(\@testArray,9,0,scalar(@testArray)-1); # Should return 5
my $target21 = binarySearch(\@testArray,21,0,scalar(@testArray)-1); # Should return 9
my $target36 = binarySearch(\@testArray,36,0,scalar(@testArray)-1); # Should return 11
my $target0 = binarySearch(\@testArray,0,0,scalar(@testArray)-1); # Should return -1

print("(0,5,10,11,-1) = ($target1,$target9,$target21,$target36,$target0)?\n");

sub binarySearch {
    # The recursive binary search subroutine

    my ($ref_array,$target,$min,$max) = @_;
    my @array = @{$ref_array};
    my $index = int $min + (($max-$min) / 2);

    if ($array[$index] == $target) {
        # Target found, return index
        return($index);
    }
    elsif ($max-$min <= 0) {
        # Target not in array, return -1
        return(-1);
    }
    elsif ($array[$index] > $target) {
        # Target is smaller than pivot value, search lower half
        return binarySearch(\@array,$target,$min,$index-1);
    }
    else{
        # Target is larger than pivot value, search upper half
        return binarySearch(\@array,$target,$index+1,$max);
    }
}
```

Functions

```
#a function is treated like another data type
#it can be set to a variable
my $foo = sub {
    return "this is a function"; };
#and retreived
print $foo -> ().".\n";

#foo is actually a reference to the function
sub funct
{
    my $foo = shift; #shift means take the argument given to the funtion and assign to r-value
    print ($foo->()." inside another function.\n"); #$ref -> (args) is the general form of using a reference as a function
}

funct($foo);
```

- Functions declarations return a reference to the subroutine
- Subroutine references can be passed around as normal scalars

Haikus

```
my $heart = "broken";          # my heart gets broken
if ("she" != "my lover"){      # if she is not my lover
     return("to the bar");     # return to the bar
}
```

```
our $perl = "better"; #Our perl is better
print("ing text is easier"); #printing is easier
warn "python to hide"; #warn python to hide
```