Built-In Functions

Notations

- For each function, I will give its name and prototype.
  - prototype = number and type of arguments
- ARRAY means an actual named array (i.e., variable starting with @)
- LIST means any list of elements (i.e., a list literal, a named array, or an expression that returns a list)
  - Recall that a LIST can have 0, 1, or many values.
- HASH means a named hash variable
- any other type is a scalar
  - notation identifies the purpose of the argument
- For more info on any of these functions, run:
  - **`perldoc -f <func_name>`**

lc STRING; uc STRING; lcfirst STRING; ucfirst STRING

- **`lc`** and **`uc`** return a lower- or upper- cased version of STRING
- **`lcfirst`** and **`ucfirst`** return the STRING with the first character lower- or upper- cased
- **`$str = 'HELLO WORLD';`**
- **`$str = lc $str;  #hello world`**
- **`$str = ucfirst $str; #Hello world`**
- **`$hola = ucfirst lc 'HOLA';  #Hola`**
- Used internally by **`\L`**, **`\U`**, **`\l`**, and **`\u`**
  - **`$hola = "\u\LHOLA";`**

Mathematical Functions

- **abs NUM**
  - absolute value of NUM
- **sqrt NUM**
  - square root of NUM
- **cos RADIANS, sin RADIANS**
  - cosine or sine of value in radians
- **atan2 RADIANS**
  - arctangent of value in radians
  - tangent(X) = sin(X) / cos(X)
- **exp POWER**
  - e to the power of POWER
- **log NUM**
  - natural logarithm of NUM
  - log in base X of N = **log(N) / log(X);**

keys HASH; values HASH

- **keys** ← return list of all keys from HASH
  - seemingly random order
- **values** ← return list of all values from HASH
  - same 'random' order as keys produces
- **my %abbr_of = ('Jan' => 'January', 'Feb' => 'February', 'Mar' => 'March', …);**
- **keys (%abbr_of)** ← ('Nov', 'Jan', 'Oct', …)
- **values (%abbr_of)** ← ('November', 'January', 'October', …)
- In scalar context, both return number of key/value pairs in the hash
  - **my $months = keys %abbr_of; #$months = 12**
  - **my $months = values %abbr_of; #$months = 12**

length EXPR

- return number of characters in EXPR
- **$a = "Hello\n";**
  $b = length $a;
  $b ← 6
- if string is omitted, returns number of characters in $_
- Do Not use to find size of array or hash
  - What do you use?
  - What will happen if you try?

index STR, SUBSTR, OFFSET

- Look for first occurrence of SUBSTR within STR (starting at OFFSET)
  - OFFSET defaults to 0 if omitted
- Return first position within STR that SUBSTR is found.

```
my $x = index "Hello World\n", "o";
my $y = index "Hello World\n", "o", $x+1;
```

  - $x ← 4, $y ← 7
- Returns -1 if SUBSTR not found.
- **rindex** ← return last position found

substr EXPR, START, LENGTH

- returns the substring of EXPR starting at character START of length LENGTH
- `my $s = 'Hello World';`
- `my $ss = substr($s, 3, 5); #'lo Wo'`
- Negative start: start that many from the end
  - `$ss = substr ($s, -4, 3);  #'orl'`
- Omit length: return all remaining chars
  - `$ss = substr($s, 6);  #'World';`
- Negative length: leave that many off the end
  - `$ss = substr($s, 6, -2) #'Wor'`

Changing Substrings

- substr is one of the rare functions that you can assign to directly:
- `my $s = 'Hello World';`
- `substr($s, 6, 5) = 'Everybody';`
- $s now: 'Hello Everybody';
- String will automatically grow or shrink accordingly.
- Alternatively, use 4-arg substring:
- `my $s = 'Fred Flintstone';`
- `substr($s, 0, 4, 'Wilma');`
  - $s => 'Wilma Flintstone'

reverse LIST

- in list context, return a list consisting of elements in LIST, in opposite order
  - **`my @foo = (1 .. 10);`**
    `my @bar = reverse @foo;`
  - @foo ← (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
    @bar ← (10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
- in scalar context, concatenate all elements of LIST into a string, and return reverse of that string
  - **`my $rev = reverse @foo;`**
  - $rev ← '01987654321'
- Useful for reversing a single string
  - my $rstr = reverse 'abcd';
    - $rstr ← 'dcba';

-X FILE

- A series of "file test" operators. Take either a filename or an open filehandle
  - If no argument given, uses $_
- **`-f 'file.txt'`** ← true if file.txt is a plain file
- **`-d $path`** ← true if $path is a directory
- **`-e $path`** ← true if $path exists
- **`-r $file`** ← true if file is readable
- **`-w $file`** ← true if the file is writable
- **`-s $file`** ← size (in bytes) of $file
- **`-M $file`** ← days since $file last modified
- For the full list, Camel Chapter 29
  - **`perldoc –f –X`**

-X Common Error

- Argument passed to file tests must be full relative path to the file or directory.
- tests will fail if file does not exist in the current working directory

```
opendir(my $dh, 'temp') or die ...;
while (my $file = readdir($dh)) {
    if (-r $file){ ... }
    # Wrong!  No $file in current
    # directory!
    if (-r "temp/$file") { ... }
    # Correct
}
```

–X efficiency

- **`-X`** actually performs a **`stat()`** on the filehandle, and stores 13 pieces of information.
- **`stat()`** is rather costly, so it's best to do it as infrequently as possible.
- After a **`stat()`**, the information is stored in memory, and accessible via the special filehandle: _
- ```
  if (-e $file and –r _) {
      print "$file exists and is readable\n";
  }
  ```
- Note this is NOT the same as **`–r $_`** !!

sort LIST

- returns LIST sorted in "ASCIIbetical" order.
  - Empty string first, then sort by ASCII chart
  - does not affect LIST that is passed in
  - note that by ASCII chart, "100" comes before "99"
    - and A-Z all come before any of a-z
- **`my @f = ('Banana','Apple','Carrot');`**
- **`my @sorted = sort @f;`**
- @sorted = ('Apple', 'Banana', 'Carrot')
- **`my @nums = sort (97 .. 102);`**
- @nums = (100, 101, 102, 97, 98, 99)

Advanced Sorting

- You can tell sort how you want a list sorted
  - Write a small subroutine describing the sort order
  - Perl will call this subroutine repeatedly, each time assigning $a and $b to be two elements from the list to sort
- In your subroutine, compare $a and $b however you need to.
- If $a should come before $b in sort order, return –1.
- If $b should be first, return 1.
- if order of $a and $b doesn't matter, return 0

```
sub by_number {
  if ($a < $b) {return –1;}
  elsif ($a > $b) {return 1;}
  else {return 0;}
}
```
```

Using Your Own Sort

- Now that we have that function, use it in sort:
  - `my @nums = (4, 2, 9, 10, 14, 11);`
  - `my @sorted = sort by_number @nums;`
  - @sorted Ł (2, 4, 9, 10, 11, 14);
- Look at that function again…

```
if ($a < $b) {return –1;}
elsif ($a > $b) {return 1;}
else {return 0;}
```

- This can be simplified quite a bit.
  - `return ($a <=> $b);`

Simplifying Further

- We now have:

```
sub by_number{
   return ($a <=> $b);
}
```

- All Perl blocks return the last value evaluated. Therefore, **`return`** keyword is optional
- When sort function is that simple, don't even need to declare it:
- `my @sorted = sort {$a <=> $b} @nums;`
- Excellent description of sorting in Llama chapter 15
  - `perldoc -f sort`
  - `perldoc -q sort`

More Examples

- Sort a series of people based on their ages
- `my %age_of =`
  `(Jess=>20, Megan=>19, Mike=>22, Paul=>29);`
- `my @kids =`
  `sort {$age_of{$b}<=>$age_of{$a}} keys %age_of;`
- @kids => ('Paul', 'Mike', 'Jess', 'Megan')
- Sort a series of strings based on their last letter:
- `my @s = ('Foo', 'Bar', 'Baz', 'Biff', 'Bam');`
- `sub by_last {`
  `my $last_a = substr($a, -1, 0);`
  `my $last_b = substr($b, -1, 0);`
  `return $last_a cmp $last_b;`
  `}`
- `my @sorted = sort by_last @s;`
- @sorted => ('Biff', 'Bam', 'Foo', 'Bar', 'Baz')