Subroutines in Perl

Jon-Michael Deldin

Dept. of Computer Science
University of Montana
jon-michael.deldin@mso.umt.edu

September 12, 2011

Outline

Topic

What is a subroutine?

A subroutine is a function in Perl that can take 0 or more arguments.

Remember these?

Functions (Math)

$$f(x) = x^2 + 2x - 1$$

$$f(3) = 14$$

Remember these?

Functions (Math)

$$f(x) = x^2 + 2x - 1$$

$$f(3) = 14$$

Functions (Perl)

```
sub f {
    my $x = shift;
    return $x**2 + 2*$x - 1;
}
print "f(3) = " . f(3);
```

Gives us

```
f(3) = 14
```

What can you do with them?

What can you do with them?

What can you do with them?

- Produce reusable code

What can you do with them?

- Produce reusable code
- Make your life easier

What can you do with them?

- Produce reusable code
- Make your life easier

What can you do with them?

- Produce reusable code
- Make your life easier

- `fasta_to_string('sequence.fna')`

What can you do with them?

- Produce reusable code
- Make your life easier

- `fasta_to_string('sequence.fna')`
- `fasta_to_array('sequence.fna')`

What can you do with them?

- Produce reusable code
- Make your life easier

- `fasta_to_string('sequence.fna')`
- `fasta_to_array('sequence.fna')`
- `clean_array(\@array)`

What can you do with them?

- Produce reusable code
- Make your life easier

- `fasta_to_string('sequence.fna')`
- `fasta_to_array('sequence.fna')`
- `clean_array(\@array)`
- `sequence_genome_and_make_coffee()`

Topic

Declaration

- use the sub keyword
- assign a name (follows variable naming rules)

Example definition

```
sub fasta_to_string {
    # code goes here
}
```

Arguments

- the x in $f(x)$

Arguments

- the x in $f(x)$
- as many as you want (even none)

Arguments

- the x in $f(x)$
- as many as you want (even none)
- passed in as an array with special name `@_`

Arguments

- the x in $f(x)$
- as many as you want (even none)
- passed in as an array with special name `@_`
- scalars only (there's a trick for arrays and hashes)

Example: Receiving arguments

Using shift

```
sub fasta_to_string {
    # first element of @_
    my $filename = shift;
    # ...
}
```

Example: Receiving arguments

Using shift

```
sub fasta_to_string {
    # first element of @_
    my $filename = shift;
    # ...
}
```

Using @_

```
sub fasta_to_string {
    # inline list for vars
    my ($filename) = @_;
    # ...
}
```

Example: Receiving more than one argument

If we needed two arguments, `$x`, `$y`, we would get args using multiple `shift` calls or `@_`.

Multiple `shift` calls

```
sub f {
    my $x = shift;
    my $y = shift;
    # ...
}
```

Example: Receiving more than one argument

If we needed two arguments, `$x`, `$y`, we would get args using multiple `shift` calls or `@_`.

Multiple `shift` calls

```
sub f {
    my $x = shift;
    my $y = shift;
    # ...
}
```

Using `@_`

```
sub f {
    my ($x, $y) = @_;
    # ...
}
```

Passing an array as an argument

- You can pass the array like a scalar if only one argument

Passing an array as an argument

- You can pass the array like a scalar if only one argument
- Otherwise, pass the array as a reference (similar to file handles)

Passing an array as an argument

- You can pass the array like a scalar if only one argument
- Otherwise, pass the array as a reference (similar to file handles)
- When passing an array to a function, use `\@my_array` as an argument

Passing an array as an argument

- You can pass the array like a scalar if only one argument
- Otherwise, pass the array as a reference (similar to file handles)
- When passing an array to a function, use `\@my_array` as an argument
- When receiving an array in the function, convert back to an array with `@{$array_ref}`

Example: Passing arrays as arguments

Example

```
sub square_array {
    my ($arrRef) = @_;
    my @arr = @{$arrRef};

    for (my $i = 0; $i < @arr; $i++) {
        $arr[$i] = $arr[$i]**2;
    }

    return \@arr;
}

my @nums = (0, 1, 2, 3, 4);
my @result = @{square_array(\@nums)};

print "ORIGINAL\n";
print join("\n", @nums);

print "\n\nNEW\n";
print join("\n", @result);
```

Output

```
ORIGINAL
0
1
2
3
4

NEW
0
1
4
9
16
```

Returning values

You can return the result of an operation, but it's not required.

Not returning

```
sub f {
    my $x = shift;
    print "x was $x!\n";
}
```

- prints a value to screen, but unusable

Returning values

You can return the result of an operation, but it's not required.

Not returning

```
sub f {
    my $x = shift;
    print "x was $x!\n";
}
```

- prints a value to screen, but unusable

Returning a value

```
sub f {
    my $x = shift;
    my $result = $x * rand(10);
    return $result;
}

my $num = f(20);

print $num; # optional
```

. . . makes `$num` this value:

```
28.6895029394152
```

Topic

Planning

Planning

Planning

- `fasta_to_array`

Planning

- `fasta_to_array`
  - input filename

Planning

- `fasta_to_array`
  - input: filename
  - output: array

Planning

- `fasta_to_array`
  - input: filename
  - output: array

Planning

- `fasta_to_array`
  - input filename
  - output array

Process

Planning

- `fasta_to_array`
  - input filename
  - output array

Process

1. Open a file handle

Planning

- `fasta_to_array`
  - input: filename
  - output: array

Process

1. Open a file handle
2. Read the contents as an array

Planning

- `fasta_to_array`
  - input: filename
  - output: array

Process

1. Open a file handle
2. Read the contents as an array
3. Close the file handle

Planning

- `fasta_to_array`
  - input: filename
  - output: array

Process

1. Open a file handle
2. Read the contents as an array
3. Close the file handle
4. Return the array

Build the stub

Build functions iteratively!

Example (sequence.fa)

```
> very interesting sequence
ACTG
CCCC
AAAAAAAA
TTTT
```

Build the stub

Build functions iteratively!

Example (sequence.fa)

```
> very interesting sequence
ACTG
CCCC
AAAAAAAA
TTTT
```

function

```
use warnings;
use strict;

sub fasta_to_array {
    my $filename = shift;

    return (); # an empty list for now
}

my @seq = fasta_to_array('sequence.fa');
```

Add file handling

Code

```
use warnings;
use strict;

sub fasta_to_array {
    my $filename = shift;

    open my $fh, '<', $filename or die("Can't read $filename");
    close $fh;

    return ();
}

my @seq = fasta_to_array('sequence.fa');
```

Read in the file

Code

```
use warnings;
use strict;

sub fasta_to_array {
    my $filename = shift;

    open my $fh, '<', $filename or die("Can't read $filename");
    my @lines = <$fh>;
    close $fh;

  return \@lines; # a reference
}

my @seq = @{fasta_to_array('sequence.fa')};
print @seq;
```

Read in the file

Code

```
use warnings;
use strict;

sub fasta_to_array {
    my $filename = shift;

    open my $fh, '<', $filename or die("Can't read $filename");
    my @lines = <$fh>;
    close $fh;

  return \@lines; # a reference
}

my @seq = @{fasta_to_array('sequence.fa')};
print @seq;
```

Output

```
> very interesting sequence
ACTG
CCCC
AAAAAAAA
TTTT
```

Topic

Planning

Planning

Planning

- `clean_fasta`

Planning

- `clean_fasta`

  input array reference from `fasta_to_array`

Planning

- `clean_fasta`
  - input: array reference from `fasta_to_array`
  - output: array reference

Planning

- `clean_fasta`
  - input: array reference from `fasta_to_array`
  - output: array reference

Planning

- `clean_fasta`
  - input: array reference from `fasta_to_array`
  - output: array reference

Process

Planning

- `clean_fasta`
  - input: array reference from `fasta_to_array`
  - output: array reference

Process

1. Get the array reference

Planning

- `clean_fasta`
  - input: array reference from `fasta_to_array`
  - output: array reference

Process

1. Get the array reference
2. Get rid of the header line

Planning

- `clean_fasta`
  - input: array reference from `fasta_to_array`
  - output: array reference

Process

1. Get the array reference
2. Get rid of the header line
3. For each line, get rid of all whitespace

Planning

- `clean_fasta`
  - input: array reference from `fasta_to_array`
  - output: array reference

Process

1. Get the array reference
2. Get rid of the header line
3. For each line, get rid of all whitespace
4. Return the cleaned array

Stub

Code

```
sub clean_fasta {
    my $ref = shift;
    my @arr = @{$ref};

    return \@arr;
}

my @seq = @{clean_fasta(fasta_to_array('sequence.fa'))};

print @seq;
```

Output

```
> very interesting sequence
ACTG
CCCC
AAAAAAAA
TTTT
```

Drop the header line

Code

```
sub clean_fasta {
    my $ref = shift;
    my @arr = @{$ref};

    shift @arr;

    return \@arr;
}

my @seq = @{clean_fasta(fasta_to_array('sequence.fa'))};

print @seq;
```

Output

```
ACTG
CCCC
AAAAAAAA
TTTT
```

Get rid of whitespace

Code

```
sub fasta_to_array {
    my $filename = shift;

    open my $fh, '<', $filename or die("Can't read $filename");
    my @lines = <$fh>;
    close $fh;

    return \@lines; # or a reference, \@lines
}

sub clean_fasta {
    my $ref = shift;
    my @arr = @{$ref};

    # remove header line
    shift @arr;

    # get rid of newlines, tabs, spaces
    for (my $i = 0; $i < @arr; $i++) {
        $arr[$i] =~ s/[\t\n ]//g;
    }

    return \@arr;
}

my @seq = @{clean_fasta(fasta_to_array('sequence.fa'))};

print "- " . join("\n- ", @seq);
```

Output

```
- ACTG
- CCCC
- AAAAAAAA
- TTTT
```