In general, what's the distinction between a *function* and a *procedure*?

In Perl?

In general, functions return values and procedures do not.
In Perl this distinction is not usually made.

In Perl-speak, what's the difference between a *function* and a *subroutine*?

Subroutine always indicates a user-defined functions, while *function* is more general, possibly indicating a built-in function.

Create a new subroutine.

```
sub funcname {
    #body
}
```

Are forward declarations required?

Not usually. Subroutine definitions can be placed in any order.

Call a subroutine.

&funcname;

What is the result of a subroutine?

The last value computed.

What's the *void context*?

The context in which the result is not being used or stored.

What's the result of `print`?

The boolean success of the operation.

Call a subroutine with arguments.

Access arguments in a subroutine.

```
@funcname(arg1, ..., argN);
```

The arguments are available inside the subroutine as the `@_` array.

What happens if a function is called with the wrong number of arguments.

Extra arguments are ignored. Missing arguments are `undef`.

What is the default scope of variables?

How can this behavior be changed?

Global.

The `my` keyword creates lexical scopes.

Declare multiple lexically scoped variables.

```
my($ident1, ..., $identN)
```

What's a common idiom for creating lexically scoped arguments to subroutines with meaningful names?

```
sub funcname {
    my($ident1, ..., $identN);
    ($ident1, ..., $identN) = @_
    #more code
}
```