The first function calculates the length of the string to be appended by using `strlen`. In the second, you specify the length of the string yourself. The third function processes its arguments like `sprintf` and appends the formatted output. The fourth function works like `vsprintf`. You can specify the address and length of an array of SVs instead of the va_list argument. The fifth function extends the string stored in the first SV with the string stored in the second SV. It also forces the second SV to be interpreted as a string.

The `sv_cat*()` functions are not generic enough to operate on values that have "magic". See Magic Virtual Tables later in this document.

If you know the name of a scalar variable, you can get a pointer to its SV by using the following:

```
    SV*  get_sv("package::varname", FALSE);
```

This returns NULL if the variable does not exist.

If you want to know if this variable (or any other SV) is actually `defined`, you can call:

```
    SvOK(SV*)
```

The scalar `undef` value is stored in an SV instance called `PL_sv_undef`.

Its address can be used whenever an `SV*` is needed. Make sure that you don't try to compare a random sv with `&PL_sv_undef`. For example when interfacing Perl code, it'll work correctly for:

```
  foo(undef);
```

But won't work when called as:

```
  $x = undef;
  foo($x);
```

So to repeat always use SvOK() to check whether an sv is defined.

Also you have to be careful when using `&PL_sv_undef` as a value in AVs or HVs (see AVs, HVs and undefined values).

There are also the two values `PL_sv_yes` and `PL_sv_no`, which contain boolean TRUE and FALSE values, respectively. Like `PL_sv_undef`, their addresses can be used whenever an `SV*` is needed.

Do not be fooled into thinking that `(SV *) 0` is the same as `&PL_sv_undef`. Take this code:

```
    SV* sv = (SV*) 0;
    if (I-am-to-return-a-real-value) {
            sv = sv_2mortal(newSViv(42));
    }
    sv_setsv(ST(0), sv);
```

This code tries to return a new SV (which contains the value 42) if it should return a real value, or undef otherwise. Instead it has returned a NULL pointer which, somewhere down the line, will cause a segmentation violation, bus error, or just weird results. Change the zero to `&PL_sv_undef` in the first line and all will be well.

To free an SV that you've created, call `SvREFCNT_dec(SV*)`. Normally this call is not necessary (see Reference Counts and Mortality).