values HASH

Returns a list consisting of all the values of the named hash. (In a scalar context, returns the number of values.)

The values are returned in an apparently random order. The actual random order is subject to change in future versions of perl, but it is guaranteed to be the same order as either the keys or each function would produce on the same (unmodified) hash. Since Perl 5.8.1 the ordering is different even between different runs of Perl for security reasons (see Algorithmic Complexity Attacks in *perlsec*).

As a side effect, calling values() resets the HASH's internal iterator, see each. (In particular, calling values() in void context resets the iterator with no other overhead.)

Note that the values are not copied, which means modifying them will modify the contents of the hash:

```
for (values %hash)      { s/foo/bar/g }   # modifies %hash values
for (@hash{keys %hash}) { s/foo/bar/g }   # same
```

See also keys, each, and sort.

vec EXPR,OFFSET,BITS

Treats the string in EXPR as a bit vector made up of elements of width BITS, and returns the value of the element specified by OFFSET as an unsigned integer. BITS therefore specifies the number of bits that are reserved for each element in the bit vector. This must be a power of two from 1 to 32 (or 64, if your platform supports that).

If BITS is 8, "elements" coincide with bytes of the input string.

If BITS is 16 or more, bytes of the input string are grouped into chunks of size BITS/8, and each group is converted to a number as with pack()/unpack() with big-endian formats n/N (and analogously for BITS==64). See §**??** for details.

If bits is 4 or less, the string is broken into bytes, then the bits of each byte are broken into 8/BITS groups. Bits of a byte are numbered in a little-endian-ish way, as in 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80. For example, breaking the single input byte chr(0x36) into two groups gives a list (0x6, 0x3); breaking it into 4 groups gives (0x2, 0x1, 0x3, 0x0).

vec may also be assigned to, in which case parentheses are needed to give the expression the correct precedence as in

```
vec($image, $max_x * $x + $y, 8) = 3;
```

If the selected element is outside the string, the value 0 is returned. If an element off the end of the string is written to, Perl will first extend the string with sufficiently many zero bytes. It is an error to try to write off the beginning of the string (i.e. negative OFFSET).

The string should not contain any character with the value > 255 (which can only happen if you're using UTF-8 encoding). If it does, it will be treated as something which is not UTF-8 encoded. When the vec was assigned to, other parts of your program will also no longer consider the string to be UTF-8 encoded. In other words, if you do have such characters in your string, vec() will operate on the actual byte string, and not the conceptual character string.

Strings created with vec can also be manipulated with the logical operators |, &, ^, and ~. These operators will assume a bit vector operation is desired when both operands are strings. See Bitwise String Operators in *perlop*.

The following code will build up an ASCII string saying 'PerlPerlPerl'. The comments show the string after each step. Note that this code works in the same way on big-endian or little-endian machines.

```
my $foo = '';
vec($foo,  0, 32) = 0x5065726C;     # 'Perl'

# $foo eq "Perl" eq "\x50\x65\x72\x6C", 32 bits
print vec($foo, 0, 8);              # prints 80 == 0x50 == ord('P')
```