General Questions

Q1. What is Perl?

Perl stands for Practical Extraction and Reporting Language. It was developed by Larry Wall and can run on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. Due to its strong text processing abilities, it was very popular, especially in the financial industry.

Q2. What are the various advantages and disadvantages of Perl?

Advantage:

- Perl is **easy to use** and **portable, cross-platform**.
- Perl is **extendable**. We can include various open-source packages and modules in a Perl program

Disadvantage:

- it is **an interpreted language** so the performance of the Perl program is slow.
- **Perl syntax is very flexible**, which is possible to reduce the readability of Perl programs.

Q3. Explain the various characteristics of Perl?

- Easy-to-code
- Open-source and OOP
- Portable and cross-platform
- Extendable
- Case-sensitive
- No distinction between the types of variables.
- Non-scalars can be used as loop indices.
- Powerful text manipulation API including Regular expressions.
- Supports high-level intrinsic operations – **Example**: stack Push/pop.

Q4. arguments of Perl interpreter

-e for Execute

-c to compile

-d to call the debugger on the file specified

-T for train model for security/input checking

-W for show all warning mode (or -w to show less warning)

-w (argument shows warning)

-p -n Implicit Loops

-i create a new file with the name of the original file and write data into it.

Q5. What is a Perl identifier?

A perl identifier is a name used to identify a variable, function, class, module, or other object. A perl variable name starts with either $, @ or % (percent sign) followed by zero or more letters, underscores, and digits

Q6. What are data types that perl supports?

scalars: $

arrays of scalars: @

hashes of scalars (associative arrays): %

Q7. What are hashes?

A Perl hash is a group of unordered key-value pairs; The keys are unique strings and values are scalar values; It is preceded with (%) sign.

Q8. Perl operators

- Arithmetic operators, + (plus), - (minus), * (asterisk), / (slash), % (percent)
- assignment operators: +=, -=, *= ...
- Increment/decrement operators: ++, --
- String concatenation: . (period or dot)
- comparison operators: ==, != (exclamation mark), >, <, >=
- logical operators: &&, ||, !
- Bitwise operators: &(and), |(or),^(ex-or),~(not),<<(shift left),>>(shift right)

Q9. String operators

eq	equal	**ne**	Not Equal
cmp	Comparison		
lt	Less than	**gt**	Greater than
le	Less than or equal	**ge**	Greater than or equal
.	Concatenation	**x**	Repetition

Q10. Explain Chomp, Chop, TK

Chomp: A chomp function removes the last characters from an expression (if it matches the value of $/) **/: slash**

Chop: A chop function removes the last character from expression

TK: an open source tool kit that is used to build web based applications using perl

Q11. Explain say() function in Perl?

The Perl say() function is not supported by older Perl versions (); It is like the Perl print() function with only one difference that it automatically adds a new line at the end.

Q12. Difference use and require in Perl?

USE: The method is used only for the modules (only to include .pm type file); The included objects are verified at the time of compilation. We don't need to specify the file extension; Loads the module at compile time.

REQUIRE: The method is used for both libraries and modules; The included objects are verified at the run time; We need to specify the file Extension.Loads at run-time.

Q13. loop control keywords are there in Perl?

next; last; redo

Q14. What is CPAN?

CPAN stands for Comprehensive Perl archive network. It is a repository which contains thousands of Perl modules.

Q15. What is scalar?

A scalar contains a single unit of data. They are preceded with a ($) sign.
A scalar contains a number, characters, reference, or a string. A reference is the address of the variable.

Q16. What are Perl strings?

Strings are an essential part in Perl. They are scalars so they start with $ sign.
Strings can be placed inside single or double quotes.

Q17. What is '->' in Perl?

In Perl, '->' symbol is an infix dereference operator. if the right hand side is an array subscript, hash key or a subroutine, then the left hand side must be a reference.

Q18. How do you check the return code of system call?

System calls "traditionally" returns 0 when successful and any number when it fails

Q19. How to debug Perl programs?

Start Perl manually with the Perl command and use the -d switch. Perl -d [program name]

Q20. Write code to create a directory if not there?

```
if (!(-d $dir and -e $dir)) {
      system('mkdir -p $dir');
}
```

Q21. How to do comment in Perl

for single line comment: use # (hash or sharp) before the line you want to comment
for multiple line comments:
use **=begin** (equal sign following begin) and **=cut** (equal cut) statement before and after the lines you want to comment.

Q22. How to substitute a particular string in a file containing millions of records?

```
perl -pi.bak -e 's/search_str/replace_str/g' filename
```

Q23. How to get the hash size?

```
scalar keys %test
```

Q24. What is the difference between die and exit?

Die() prints out stderr message in the terminal before exiting the program
Exit() just terminate the program without giving any message
Die also can evaluate expressions before exiting

Q25. What is meant by a 'pack' in perl?

The pack function is used to convert a list into a string,
according to a user-defined template (ex. a binary representation of a list).

Q26.

Q27. How can we add/remove elements in Array?

push: will add an element to the end of the Array
pop: removes the last element of an array
unshift: will add an element to the beginning of an Array
shift: removes the first element of an array

Q28. Define warn and confess function in Perl?

The warn function gives the warning on encountering an error but does not exit the script. Script keeps on running; The confess function is used within the Carp standard library for error handling.

Q29. Explain gmtime() epoch time function in Perl

This function works similar to localtime() with only one difference
that returned value is localized for the standard Greenwich time zone
The epoch time refers to the number of seconds after a specific date and time.
This specific date and time varies for different operating systems

Q30. What is dynamic scoping?

local($x) saves away the old value of the global variable $x and assigns a new value for the duration of the subroutine which is visible in other functions called from that subroutine. This is done at run-time, so is called dynamic scoping. local() always affects global variables, also called package variables or dynamic variables.
my($x) creates a new variable that is only visible in the current subroutine. This is done at compile-time, so it is called lexical or static scoping. my() always affects private variables, also called lexical variables or (improperly) static(ly scoped) variables.
What is the difference between 'my' and 'local' variable scope declarations?
Both of them are used to declare local variables.
The variables declared with 'my' can live only within the block
and cannot get its visibility inherited functions called within the block
The variable defined as 'local' can live within the block and have its visibility in the functions called within that block.

```
local $uustring = "local variable";
{
my $uustring = "my variable";
print("in block is:".$uustring."\n");  # output: in block is:my variable
print("local block is:".$::uustring."\n"); # local block is:local variable
}
```

Q31. What are Perl references?

A Perl reference is a scalar data type that holds the location of another value which could be scalar, arrays, or hashes. Because of its scalar nature, a reference can be used anywhere, a scalar can be used.

```
$a = \@b
@$a is @b
\ (back slash)
```

Q32. Write code to list down the array content?

```
for (@test) {
      print $_;
}
```

Q33. Define prefix dereferencer?

When we dereference a variable using a particular prefix, they are called prefix dereference.

$ - scalar variables

% - hash variables

@ - array variables

& - subroutines

Q34. How to implement stack in Perl?

stack is last in (push) first out (pop)

Q35. How do you navigate through an XML document?

Event based parser (XML::Parser) and memory based parser (XML::DOM (Document Object Model).

Q36. Why does Perl not have overloaded functions?

Because you can inspect the argument count, return context and object types all by yourself. the number of arguments is in @_ (at underscore)

Q37. Define @ISA, @EXPORT, @EXPORT_OK?

@ISA: each package has its own @ISA array, this array keep track of classes it is inheriting

@EXPORT: this array stores the subroutine to be exported from a module

@EXPORT_OK: this array stores the subroutine to be exported only on request

Q38. $a <=> $b

```
# sort lexically
my @articles = sort @files;
# same thing, but with explicit sort routine
my @articles = sort {$a cmp $b} @files;
# now case-insensitively
my @articles = sort {fc($a) cmp fc($b)} @files;
# same thing in reversed order
my @articles = sort {$b cmp $a} @files;
# sort numerically ascending
```

```
my @articles = sort {$a <=> $b} @files;
# sort numerically descending
my @articles = sort {$b <=> $a} @files;
# this sorts the %age hash by value instead of key
# using an in-line function
my @eldest = sort { $age{$b} <=> $age{$a} } keys %age;
```

Q39. common modules

xml::parser, net:ftp, net:telnet, socket, dbi, dbd etc. Time:ParseDate, Date::Format, Data::Dumper

Q40. Array and list

Both list and array can be defined as a set of elements. The main difference between a list and an array in Perl is that a list is **immutable** i.e. it cannot be altered directly. In Perl, **a list is an array without a name**. Hence, most of the times array and list are used interchangeably. An array is mutable and its contents can grow, shrink in size, etc. Thus in order to change the contents of a list, we can store the list as an array. An array is a variable that provides dynamic storage for a list.

```
(1..9) is a list
@a = (1..9) is an array
```

Q41. How to create a three-dimensional array? How to add three arrays together?

```
# Array Declaration
my @arrayA = qw(1 0 0);
my @arrayB = qw(0 1 0);
my @arrayC = qw(0 0 1);
# Merging 3 arrays into One Final array
my @result = (\@arrayA, \@arrayB, \@arrayC);
my @threeArrays = (@arrayA, @arrayB, @arrayC);
```

Q42. File open, read, write, close

```
open (IN, "test.txt")||die "can't open file: test.txt\n";
my $tmpfile =$$."_tmp.txt";
open (OUT,">$tmpfile")||die "can't open tmpfile:$tmpfile\n";
```

```
while (<IN>) {
        my $lines = $_;
        $lines =~ s/name=\"cmd\"//;
        $lines =~ s/^\s+\<(\d+)/\t\<cmd name=\"$1\"/;
        print OUT $lines;
}
close (IN);
close (OUT);
```

Q43. Subroutine?

Call a subroutine: subroutine_name() or &subroutine_name
If the return statement is missing, then the subroutine implicitly returns the value of the last expression in its body.

Q44. Environment Variables

%ENV contains the value of all environment variables. $ENV{'PATH'}

Q45. Remove duplicate values from array

```
print join(" ", @array), "\n";
print join(" ", keys %{{ map { $_ => 1 } @array }}), "\n";
```

Q46. The difference between for() and foreach()?

The for statement has an initialization, condition check and increment expressions in its body and is used for general iterations performing operations involving a loop.
The foreach statement is particularly used to iterate through arrays and runs for the length of the array.

Q47. The difference between exec() and system()?

Both are to run system commands or other scripts.
exec() command stops the execution of the current process and starts the execution of the new process and does not return back to the stopped process.
system() command holds the execution of the current process, forks a new process and continues with the execution of the command specified and returns back to the process on hold to continue execution.

Q48. What is the use of command "use strict"?

Use strict command calls the strict pragma and is used to force checks on definition and usage of variables, references and other bare words used in the script. If unsafe or ambiguous statements are used, this command stops the execution of the script instead of just providing warnings.

Q49: how to get the element number of an array?

scalar @string or $#string + 1

Q50: Write an example explaining the use of symbol tables.

In perl the symbol table is a hash that contains the list of all the names defined in a namespace. It contains all the functions and variables. For every namespace the hash is named after the namespace followed by colons (Foo is package name).

```
print(Dumper(\%Foo::));
```

Q51: Alias, Typeglobs (What can be done for efficient parameter passing in perl?)

aliases are considered to be faster than references as they do not require any dereferencing.

```
$spud = "Wow!";
@spud = ("idaho", "russet");
*potato = *spud; # Alias potato to spud using typeglob assignment
print "$potato\n"; # prints "Wow!"
print @potato, "\n"; # prints "idaho russet"
```

Q52. How and what are closures implemented in perl?

The closure is a block of code that is used to capture the environments where it is defined. It specifically captures any lexical variables the block contains and uses defined in an outer space.

```
{
   my $seq = 3 ;
   sub sequence { $seq += 3 }
}
print $seq; # this is out of scope
print sequence; # this prints 6
```

```
print sequence; # this prints 9
```

Q53. Can a subroutine return either a scalar or an array? If so, how is this accomplished?

Yes, return @a ； return $a

It is not recommended to return an array, the better way is to return the reference of an array.

Q54. wantarray()

```
sub foo {
    return(wantarray() ? qw(A, B, C) : '1');
}
$result = foo();    # scalar context
@result = foo();    # array context
print("foo() in a  scalar context: $result\n");
print("foo() in an array  context: @result\n");
# foo() in a  scalar context: 1
# foo() in an array  context: A, B, C
```

Q55. explain the "defined or" operator? What convenience does it provide??

Since version 5.10.0 Perl has had the defined-or operator (‘//’). This will check if the variable is defined, and if it is not, Perl will execute the code on the right-side of the operator. We can use it to simplify our subroutine code：

```
my $rabbits = 0;
$rabbits //= 1;
```

Regular Expression

Matching regular expression operator, m///

Substitute regular expression operator, s///

Transliterate regular expression operator, tr///

meta characters

/pattern/ 结果

. 匹配除换行符以外的所有字符

x? 匹配 0 次或一次 x 字符串

x* 匹配 0 次或多次 x 字符串，但匹配可能的最少次数

x+ 匹配 1 次或多次 x 字符串，但匹配可能的最少次数

.* 匹配 0 次或一次的任何字符

.+ 匹配 1 次或多次的任何字符

{m} 匹配刚好是 m 个 的指定字符串

{m,n} 匹配在 m个 以上 n个 以下 的指定字符串

{m,} 匹配 m个 以上 的指定字符串

[] 匹配符合 [] 内的字符

[^] 匹配不符合 [] 内的字符

[0-9] 匹配所有数字字符

[a-z] 匹配所有小写字母字符

[^0-9] 匹配所有非数字字符

[^a-z] 匹配所有非小写字母字符

^ 匹配字符开头的字符

$ 匹配字符结尾的字符

\d 匹配一个数字的字符，和 [0-9] 语法一样

\d+ 匹配多个数字字符串，和 [0-9]+ 语法一样

\D 非数字，其他同 \d

\D+ 非数字，其他同 \d+

\w 英文字母或数字的字符串，和 [a-zA-Z0-9] 语法一样

\w+ 和 [a-zA-Z0-9]+ 语法一样

\W 非英文字母或数字的字符串，和 [^a-zA-Z0-9] 语法一样

\W+ 和 [^a-zA-Z0-9]+ 语法一样

\s 空格，和 [\n\t\r\f] 语法一样

\s+ 和 [\n\t\r\f]+ 一样

\S 非空格，和 [^\n\t\r\f] 语法一样

\S+ 和 [^\n\t\r\f]+ 语法一样
\b 匹配以英文字母,数字为边界的字符串
\B 匹配不以英文字母,数值为边界的字符串
a|b|c 匹配符合a字符 或是b字符 或是c字符 的字符串
abc 匹配含有 abc 的字符串
(pattern) () 这个符号会记住所找寻到的字符串，是一个很实用的语法。第一个 () 内所找到的字符串变成 $1 这个变量或是 \1 变量，第二个 () 内所找到的字符串变成 $2 这个变量或是 \2 变量，以此类推下去。
/pattern/i i 这个参数表示忽略英文大小写，也就是在匹配字符串的时候，不考虑英文的大小写问题。
\ 如果要在 pattern 模式中找寻一个特殊字符，如 "*"，则要在这个字符前加上 \ 符号，这样才会让特殊字符失效

原则1：正则表达式有三种不同形式(匹配(m/ /)，替换(s/ / /eg)和转换(tr/ / /))。
原则2：正则表达式仅对标量进行匹配($scalar =~ m/a/; 可以工作; @array =~ m/a/ 将把 @array作为标量对待，因此可能不会成功)。
原则3：正则表达式匹配一个给定模式的最早的可能匹配。缺省时，仅匹配或替换正则表达式一次($a = 'string string2'; $a =~ s/string/ /; 导致 $a = 'string 2')。
原则4：正则表达式能够处理双引号所能处理的任意和全部字符($a =~ m/$varb/ 在匹配前把varb扩展为变量；如果 $varb = 'a' $a = 'as'，$a =~ s/$varb/ /; 等价于 $a =~ s/a/ /;，执行结果使 $a = " s")。
原则5：正则表达式在求值过程中产生两种情况：结果状态和反向引用： $a=~ m/pattern/ 表示 $a 中是否有子串 pattern 出现，$a =~ s/(word1)(word2)/$2$1/ 则“调换”这两个单词。
原则6：正则表达式的核心能力在于通配符和多重匹配运算符以及它们如何操作。$a =~ m/\w+/ 匹配一个或多个单词字符；$a =~ m/\d/" 匹配零个或多个数字。
原则7：如果欲匹配不止一个字符集合，Perl使用 "|" 来增加灵活性。如果输入 m/(cat|dog)/ 则相当于“匹配字符串 cat 或者 dog。

Special Variable Types

Global Scalar Special Variables

	Name	Description
$_	**$ARG**	**The default input and pattern-searching space.**
$.	$NR	The current input line number of the last filehandle that was read. An explicit close on the filehandle resets the line number.
$/	$RS	The input record separator; newline by default. If set to the null string, it treats blank lines as delimiters.
$,	$OFS	The output field separator for the print operator.
$	$ORS	The output record separator for the print operator.
$"	$LIST_SEPARATOR	Like "$," except that it applies to list values interpolated into a double-quoted string (or similar interpreted string). Default is a space.
$;	$SUBSCRIPT_SEPARATOR	The subscript separator for multidimensional array emulation. Default is "\034".
$^L	$FORMAT_FORMFEED	What a format outputs to perform a form feed. Default is "\f".
$:	$FORMAT_LINE_BREAK_CHARACTERS	The current set of characters after which a string may be broken to fill continuation fields (starting with ^) in a format. Default is "\n"".
$^A	$ACCUMULATOR	The current value of the write accumulator for format lines
$#	$OFMT	Contains the output format for printed numbers (deprecated).

$?	$CHILD_ERROR	The status returned by the last pipe close, backtick (``) command, or system operator.
$!	$OS_ERROR or $ERRNO	If used in a numeric context, yields the current value of the errno variable, identifying the last system call error. If used in a string context, yields the corresponding system error string.
$@	$EVAL_ERROR	The Perl syntax error message from the last eval command.
$$	$PROCESS_ID or $PID	The pid of the Perl process running this script.
$<	$REAL_USER_ID or $UID	The real user ID (uid) of this process.
$>	$EFFECTIVE_USER_ID or $EUID	The effective user ID of this process.
$(	$REAL_GROUP_ID or $GID	The real group ID (gid) of this process.
$)	$EFFECTIVE_GROUP_ID or $EGID	The effective gid of this process.
$0	$PROGRAM_NAME	Contains the name of the file containing the Perl script being executed.
$[		The index of the first element in an array and of the first character in a substring. Default is 0.
$]	$PERL_VERSION	Returns the version plus patchlevel divided by 1000.
$^D	$DEBUGGING	The current value of the debugging flags.
$^E	$EXTENDED_OS_ERROR	Extended error message on some platforms.

$^F	$SYSTEM_FD_MAX	The maximum system file descriptor, ordinarily 2
$^H		Contains internal compiler hints enabled by certain pragmatic modules
$^I	$INPLACE_EDIT	The current value of the inplace-edit extension. Use undef to disable inplace editing
$^M		The contents of $M can be used as an emergency memory pool in case Perl dies with an out-of-memory error. Use of $M requires a special compilation of Perl. See the INSTALL document for more information
$^O	$OSNAME	Contains the name of the operating system that the current Perl binary was compiled for
$^P	$PERLDB	The internal flag that the debugger clears so that it doesn't debug itself
$^T	$BASETIME	The time at which the script began running, in seconds since the epoch.
$^W	$WARNING	The current value of the warning switch, either true or false.
$^X	$EXECUTABLE_NAME	The name that the Perl binary itself was executed as.
$ARGV		Contains the name of the current file when reading from <ARGV>

Global Array Special Variables

@ARGV	The array containing the command-line arguments intended for the script. **$#ARGV** is the last index of the array.

@INC	The array containing the list of places to look for Perl scripts to be evaluated by the do, require, or use constructs
@F	The array into which the input lines are split when the -a command-line switch is given
@_	The array stored all arguments passed to the function.

Global Hash Special Variables

%INC	The hash containing entries for the filename of each file that has been included via do or require
%ENV	The hash containing your current environment
%SIG	The hash used to set signal handlers for various signals

Global Special Filehandles

ARGV	The special filehandle that iterates over command line filenames in @ARGV. Usually written as the null filehandle in <>
STDERR	The special filehandle for standard error in any package.
STDIN	The special filehandle for standard input in any package.
STDOUT	The special filehandle for standard output in any package.
DATA	The special filehandle that refers to anything following the __END__ token in the file containing the script. Or, the special filehandle for anything following the __DATA__ token in a required file, as long as you're reading data in the same package __DATA__ was found in
_	The special filehandle used to cache the information from the last stat,

(underscore)	lstat, or file test operator.

Global Special Constants

__END__	Indicates the logical end of your program. Any following text is ignored, but may be read via the DATA filehandle
__FILE__	Represents the filename at the point in your program where it's used. Not interpolated into strings
__LINE__	Represents the current line number. Not interpolated into strings
__PACKAGE__	Represents the current package name at compile time, or undefined if there is no current package. Not interpolated into strings

Regular Expression Special Variables

$digit		Contains the text matched by the corresponding set of parentheses in the last pattern matched. For example, $1 matches whatever was contained in the first set of parentheses in the previous regular expression
$&	$MATCH	The string matched by the last successful pattern match
$`	$PREMATCH	The string preceding whatever was matched by the last successful pattern match
$'	$POSTMATCH	The string following whatever was matched by the last successful pattern match
$+	$LAST_PAREN _MATCH	The last bracket matched by the last search pattern. This is useful if you don't know which of a set of alternative patterns was matched. For example : /Version: (.*)|Revision: (.*)/ &&

		($rev = $+)

Filehandle Special Variables

$|	$OUTPUT_AUTOFLUSH	If set to nonzero, forces an fflush(3) after every write or print on the currently selected output channel
$%	$FORMAT_PAGE_NUMBER	The current page number of the currently selected output channel
$=	$FORMAT_LINES_PER_PAGE	The current page length (printable lines) of the currently selected output channel. Default is 60
$-	$FORMAT_LINES_LEFT	The number of lines left on the page of the currently selected output channel
$~	$FORMAT_NAME	The name of the current report format for the currently selected output channel. Default is the name of the filehandle
$^	$FORMAT_TOP_NAME	The name of the current top-of-page format for the currently selected output channel. Default is the name of the filehandle with _TOP appended

Built-in functions

eval()

```
$print = "print (\"hello,world\\n\");";
eval ($print);
if ($@) {
    print(“Error!\n”)
}
```

system()

```
@proglist = ("echo", "hello,world!");
print(system(@proglist));
```

fork()

```
$retval = fork();
if ($retval ==0) {
      print("this is in child process\n");
      sleep(5);
} else {
      print("parent process existed.\n");
}
```

pipe()

exec()

similar with system() but the main process will exist after run exec()

die()

End the process with the die message input.

```
die ("Cannot open input file");
```

warn()

similar with die() but not exist the process

exit(return code)

exit(4);

kill()

```
kill(9, 123456)          # kill(signal, processID)
```

sleep()

waitpid()

```
$procid = fork();
if ($procid == 0) {      # this is the child process
  print ("this line is printed first\n");
  exit(1);
} else {     # this is the parent process
  waitpid ($procid, 1);
  print ("this line is printed last\n");
}
```

chroot()

times()

returns the program running time

index(string, substring, position)

```
my $string = "good morning!";
my $substring = "oo";
print(index($string, $substring, 1));
```

rindex()

similar with index() but start at the right of the string.

length()

```
print(length("hello!"),"\n");
```

pos()

```
$string =~ m/L/g;
print(pos($string),"\n"); #returns the position of last matching
(starting at 1)
```

substr()

```
print(substr($string, 2, 4),"\n");
```

lc() uc() lcfirst() ucfirst()

lower, upper, lower first, upper first

quotemeta()

```
$string = quotemeta($string);  ← same as → $string =~
s/(\W)/\\$1/g;
```

join()

```
my @string = ("haha", "tim", "jia");
print(join("|", @string));
```

sprintf()

```
$num = 26;
$outstr = sprintf("%d = %x hexadecimal or %o octal\n",$num, $num, $num);
print ($outstr);
```

chop()

remove last character and returns the last removed character (if it is @array)

chomp()

check if the last character is $/ and remove the last character if it is. returns the number of removed characters.

crypt()

调用语法 result = crypt (original, salt);
解说 用DES算法加密字符串，original是将要加密的字符串，salt是两个字符的字符串，定义如何改变DES算法，以使更难解码。返回值为加密后的串。

hex()

调用语法 decnum = hex (hexnum);
解说 将十六进制数(字符串形式)转化为十进制数。

int()

调用语法 intnum = int (floatnum);
解说 将浮点数舍去小数部分转化为整型数。

oct()

调用语法 decnum = oct (octnum);
解说 将八进制数(字符串形式)或十六进制数("0x.."形式)转化为十进制数。

ord()

调用语法 asciival = ord (char);
解说 返回单个字符的ASCII值，与PASCAL中同名函数类似。

chr()

调用语法 $char = chr (asciival);
解说 返回ASCII值的相应字符，与PASCAL中同名函数类似。

defined()

调用语法 retval = defined (expr);
解说 判断一个变量、数组或数组的一个元素是否已经被赋值。expr为变量名、数组名或一个数组元素。如果已定义，返回真，否则返回假。

undef()

调用语法 retval = undef (expr);
解说 取消变量、数组或数组元素甚至子程序的定义，回收其空间。返回值始终为未定义值，此值与空串等效。

grep()

调用语法 @foundlist = grep (pattern, @searchlist);
解说 与同名的UNIX查找工具类似，grep函数在列表中抽取与指定模式匹配的元素，参数pattern为欲查找的模式，返回值是匹配元素的列表。

例子 @list = ("This", "is", "a", "test");
@foundlist = grep(/^[tT]/, @list);
结果 @foundlist = ("This", "test");

splice()

调用语法 @retval = splice (@array, slipelements, length, @newlist);
解说　拼接函数可以向列表（数组）中间插入元素、删除子列表或替换子列表。参数skipelements是拼接前跳过的元素数目，length是被替换的元素数，newlist是将要拼接进来的列表。当newlist的长度大于length时，后面的元素自动后移，反之则向前缩进。因此，当length=0时，就相当于向列表中插入元素，而形如语句
splice (@array, -1, 0, "Hello");
则向数组末尾添加元素。而当newlist为空时就相当于删除子列表，这时，如果length为空，就从第skipelements个元素后全部删除，而删除最后一个元素则为：splice (@array, -1);这种情况下，返回值为被删去的元素列表。

shift()

调用语法 element = shift (@arrayvar);
解说 删去数组第一个元素，剩下元素前移，返回被删去的元素。不加参数时，缺省地对@ARGV进行操作。

unshift()

调用语法 count = unshift (@arrayver, elements);
解说 作用与shift相反，在数组arrayvar开头增加一个或多个元素，返回值为结果(列表)的长度。等价于splice (@array, 0, 0, elements);

push()

调用语法 push (@arrayvar, elements);
解说 在数组末尾增加一个或多个元素。等价于slice (@array, @array, 0, elements);

pop()

调用语法 element = pop (@arrayvar);
解说 与push作用相反，删去列表最后一个元素，并将其作为返回值，当列表已空，则返回“未定义值”(即空串)。

split()

调用语法 @list = split (pattern, string, maxlength);
解说 将字符串分割成一组元素的列表。每匹配一次pattern，就开始一个新元素，但pattern本身不包含在元素中。maxlength是可选项，当指定它时，达到该长度就不再分割。

sort()

调用语法 @sorted = sort (@list);
解说 按字母次序给列表排序。

reverse()

调用语法 @reversed = reverse (@list);
解说 按字母反序给列表排序。

map()

调用语法 @resultlist = map (expr, @list);
解说 此函数在Perl5中定义，可以把列表中的各个元素作为表达式expr的操作数进行运算，其本身不改变，结果作为返回值。在表达式expr中，系统变量$_代表各个元素。

```
@list = (100, 200, 300);
@results = map ($_+1, @list);
@results = map (&mysub($_), @list);
```

keys()

@list = keys (%assoc_array);
解说 返回关联数组无序的下标列表。

values()

调用语法 @list = values (%assoc_array);
解说 返回关联数组无序的值列表。

each()

调用语法 @pair = each (%assoc_array);
解说 返回两个元素的列表--键值对（即下标和相应的值），同样无序。当关联数组已空，则返回空列表。

delete()

调用语法 element = delete (assoc_array_item);

解说 删除关联数组中的元素，并将其值作为返回值。

```
%array = ("foo", 26, "bar", 17");
$retval = delete ($array{"foo"});
```

结果 $retval = 26;

exists()

调用语法 result = exists (element);

解说 在Perl5中定义，判断关联数组中是否存在某元素，若存在，返回非零值(真)，否则返回零值(假)。

例子 $result = exists ($myarray{$mykey});