

# 分子生物计算 (Perl 语言编程)

天津医科大学  
生物医学工程与技术学院

2016-2017 学年上学期 (秋)  
2014 级生信班

## 第二章 Perl 语言入门

伊现富 (Yi Xianfu)

天津医科大学 (TJMU)  
生物医学工程与技术学院

2016 年 11 月



# 教学提纲

1

引言

2

Perl 简介

3

变量

- 标量
- 数组
- 散列
- 内置变量

4

操作符

5

基本函数

6

判断语句

- if 语句
- unless 语句
- given-when 语句

7

循环语句

- foreach 语句
- for 语句
- while 语句
- until 语句

8

列表操作

- sort
- grep
- map
- 高效排序

9

检修脚本

10

回顾与总结

- 总结
- 思考题

# 教学提纲

## 1 引言

## 2 Perl 简介

## 3 变量

- 标量
- 数组
- 散列
- 内置变量

## 4 操作符

## 5 基本函数

## 6 判断语句

- if 语句
- unless 语句
- given-when 语句

## 7 循环语句

- foreach 语句
- for 语句
- while 语句
- until 语句

## 8 列表操作

- sort
- grep
- map
- 高效排序

## 9 检修脚本

- ## 10 回顾与总结
- 总结
  - 思考题



# 引言 | 计算机



计算机程序（Computer Program）是指一组指示计算机或其他具有信息处理能力装置每一步动作的指令，通常用某种程序设计语言编写，运行于某种目标体系结构上。

打个比方，一个程序就像一个用汉语（程序设计语言）写下的红烧肉菜谱（程序），用于指导懂汉语和烹饪手法的人（体系结构）来做这个菜。

通常，计算机程序要经过编译和链接而成为一种人们不易看清而计算机可解读的格式，然后运行。未经编译就可运行的程序，通常称之为脚本程序（script）。



编程语言 (programming language) , 是用来定义计算机程序的形式语言。它是一种被标准化的交流技巧, 用来向计算机发出指令。一种计算机语言让程序员能够准确地定义计算机所需要使用的数据, 并精确地定义在不同情况下所应当采取的行动。

与英语类似, 编程语言其实就是一门外语, 只不过是专门用于跟计算机进行沟通的外语。

在电脑领域已发明了上千种不同的编程语言, 而且每年仍有新的编程语言诞生。有许多用于特殊用途的语言, 只在特殊情况下使用。例如, PHP 专门用来显示网页; Perl 更适合文本处理; C 语言被广泛用于操作系统和编译器的开发 (所谓的系统编程) 。



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



## Perl

- Perl 是高级、通用、直译式、动态的程序语言
- Practical Extraction and Report Language, 实用摘录与报表语言
- Pathologically Eclectic Rubbish Lister, 病态折中式垃圾列表器
- 拉里·沃尔 (Larry Wall) , 1987 年 12 月 18 日

## 特性

- 具有动态语言强大灵活的特性
- 借用了 C、sed、awk、shell 等语言的特性，提供了许多冗余语法
- 使用了语言学的思维（泛型变量、动态数组、Hash 表等）
- 程序员可以忽略内部数据存储、类型、内存越界等细节
- 内部集成了正则表达式的功能
- 巨大的第三方代码库 CPAN (Comprehensive Perl Archive Network, Perl 综合典藏网)；搜索引擎 MetaCPAN

## Perl

- Perl 是高级、通用、直译式、动态的程序语言
- Practical Extraction and Report Language, 实用摘录与报表语言
- Pathologically Eclectic Rubbish Lister, 病态折中式垃圾列表器
- 拉里·沃尔 (Larry Wall) , 1987 年 12 月 18 日

## 特性

- 具有动态语言强大灵活的特性
- 借用了 C、sed、awk、shell 等语言的特性，提供了许多冗余语法
- 使用了语言学的思维（泛型变量、动态数组、Hash 表等）
- 程序员可以忽略内部数据存储、类型、内存越界等细节
- 内部集成了正则表达式的功能
- 巨大的第三方代码库 CPAN (Comprehensive Perl Archive Network, Perl 综合典藏网) ; 搜索引擎 MetaCPAN

## 简介

1954 年，程序员、系统管理员、语言学家和作家。

## 在 Perl 里的工作（来自 Perl 官方文档）

- 拉里对 Perl 如何表现的定义总是对的。这说明他对核心功能有最终否决权。
- 拉里可以日后改变对任何东西的看法，不论他以前是否使用了规则 1。（拉里总是对的，即使当他原来是错的。）

## 程序员的 3 个美德

- 懒惰；不耐烦；骄傲
- (出处：《Programming Perl》（《骆驼书》）第二版，沃尔（和共同作者 Randal L. Schwartz 与 Tom Christiansen）

## 简介

1954 年，程序员、系统管理员、语言学家和作家。

## 在 Perl 里的工作（来自 Perl 官方文档）

- 拉里对 Perl 如何表现的定义总是对的。这说明他对核心功能有最终否决权。
- 拉里可以日后改变对任何东西的看法，不论他以前是否使用了规则 1。（拉里总是对的，即使当他原来是错的。）

## 程序员的 3 个美德

- 懒惰；不耐烦；骄傲
- (出处：《Programming Perl》（《骆驼书》）第二版，沃尔（和共同作者 Randal L. Schwartz 与 Tom Christiansen）

## 简介

1954 年，程序员、系统管理员、语言学家和作家。

## 在 Perl 里的工作（来自 Perl 官方文档）

- 拉里对 Perl 如何表现的定义总是对的。这说明他对核心功能有最终否决权。
- 拉里可以日后改变对任何东西的看法，不论他以前是否使用了规则 1。（拉里总是对的，即使当他原来是错的。）

## 程序员的 3 个美德

- 懒惰；不耐烦；骄傲
- (出处：《Programming Perl》（《骆驼书》）第二版，沃尔（和共同作者 Randal L. Schwartz 与 Tom Christiansen）

## 书写规范

- Perl：程序语言本身
- perl：实际编译并运行程序的解释器
- PERL：错误的写法，外行的标志



## 应用

Perl 擅长处理整体来说“约有 90% 与文字处理有关，10% 与其他事物有关”的问题。

## 领域

- 网络编程、CGI (Common Gateway Interface, 通用网关接口)
- 图形编程
- 系统管理
- 生物信息
- ...



## 名号

- 脚本语言中的瑞士军刀
- Unix 中的王牌工具
- 一种拥有各种语言功能的梦幻脚本语言
- 名符其实的“胶水语言”
- 黑客语言
- 拯救人类基因组计划的“利器”
- 与 Python 并称为生物信息学家的语言
- ...



## 应用举例

**Asciiquarium** Asciiquarium is an aquarium/sea animation in ASCII art. Enjoy the mysteries of the sea from the safety of your own terminal!

**ack** ack is a code-searching tool, similar to grep but optimized for programmers searching large trees of source code. It runs in pure Perl, is highly portable, and runs on any platform that runs Perl.

**Mojo-Webqq** 使用 Perl 语言编写的 smartqq/webqq 客户端框架（非 GUI）。

**Mojo-Weixin** 使用 Perl 语言编写的微信/weixin/wechat 客户端框架（非 GUI）。



## 应用举例

- Bugzilla** Bugzilla is free and open source web-based bug-tracking software that is developed by an active group of volunteers in the Mozilla community, and used by thousands of projects and companies around the world.
- RT** RT is an enterprise-grade issue tracking system. It allows organizations to keep track of what needs to get done, who is working on which tasks, what's already been done, and when tasks were (or weren't) completed.
- Webmin** Webmin is a web-based interface for system administration for Unix. Using any browser that supports tables and forms, you can setup user accounts, Apache, internet services, DNS, file sharing and so on.
- Circos** Circos is a program for the generation of publication-quality, circularly composited renditions of genomic data and related annotations. Circos is particularly suited for visualizing alignments, conservation and intra and inter-chromosomal relationships.

# Perl | 应用 | Asciiquarium



ack!

Security alert

New in ack 2

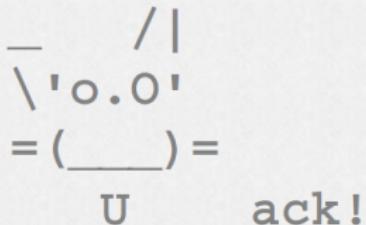
Why ack?

Install

Documentation

Community

More Tools



## ack 2.14 is a tool like grep, optimized for programmers

Designed for programmers with large heterogeneous trees of source code, ack is written purely in portable Perl 5 and takes advantage of the power of Perl's regular expressions.

### Top 5 reasons to use ack



#### Blazing fast

It's fast because it only searches the stuff it makes sense to search.



#### Better search

Searches entire trees by default while ignoring Subversion, Git and other VCS directories and other files that aren't your source code.



#### Designed for code search

Where grep is a general text search tool, ack is especially for the programmer searching source code. Common tasks take fewer keystrokes.



#### Highly portable

Ack is pure Perl, so it easily runs on a Windows installation Perl (like [Strawberry Perl](#)) without modifications.



#### Free and open

Ack costs nothing. It's 100% free and open source under [Artistic License v2.0](#).

### Security alert

ack versions 2.00 to 2.11\_02 are susceptible to a code execution exploit. Please [upgrade to 2.12 or higher ASAP](#). See the [security alert](#) for more information.

### How to install

ack is simple to install, via CPAN, package or simple download. [Read how](#).

### People love ack

'Every once in a while something comes along that improves an idea so much, you can't ignore it. Such a thing is Ack, the grep replacement.'



Bugzilla@Mozilla - Main Page version 4.0.6+

Home | New | Browse | Search |  Search | [\[?\]](#) | Reports | Requests | New Account | Log In | Forgot Password

## Welcome to Bugzilla

  
[Get Help](#)

  
[File a Bug](#)

  
[Search](#)

  
[Open a New Account](#)

Enter a bug # or some search terms

[Quick Search](#) | [Quick Search help](#) | [Install the Quick Search plugin](#)

[Bugzilla User's Guide](#) | [Release Notes](#) | [Bugzilla Etiquette](#) | [Bug Writing Guidelines](#)

Home | New | Browse | Search |  Search | [\[?\]](#) | Reports | Requests | New Account | Log In | Forgot Password

[Privacy Policy](#)



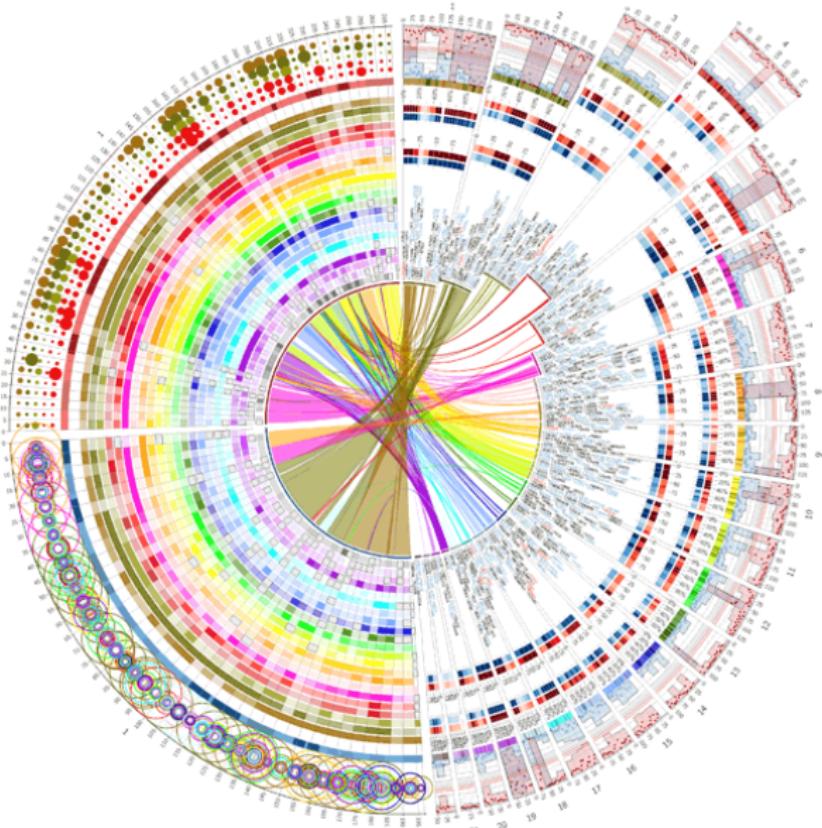
# Perl | 应用 | Webmin

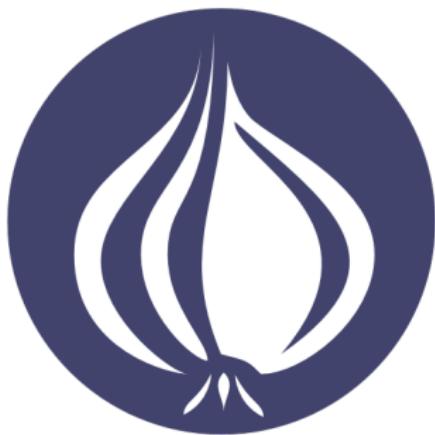
The screenshot shows the Webmin 1.740 configuration interface. The left sidebar contains a navigation tree with categories like Webmin, System, Servers, etc. The "Webmin Configuration" link under the Webmin category is highlighted with a red box. The main panel displays a grid of icons representing different configuration options:

IP Access Control	Ports and Addresses	Logging	Proxy Servers and Downloads
User Interface	Webmin Modules	Operating System and Environment	Language
Index Page Options	Upgrade Webmin	Authentication	Two-Factor Authentication
Reassign Modules	Edit Categories	Module Titles	Webmin Themes
Trusted Referrers	Anonymous Module Access	File Locking	Mobile Device Options
Blocked Hosts and Users	Background Status Collection	Advanced Options	Debugging Log File
Web Server Options	Webmin Scheduled Functions	Sending Email	SSL Encryption
Certificate Authority			



# Perl | 应用 | Circos





C P  N

A large, bold, black sans-serif font spelling out "CPAN". A small graphic of two stacked books forming a triangle is positioned between the "P" and the "N".

- **TMTOWTDI**: There's More Than One Way To Do It. 不只一种方法来做一件事。发音为“Tim Toady”。
- **TIMTOWTDIBSCINABTE**: There's more than one way to do it, but sometimes consistency is not a bad thing either. 不只一种方法来做一件事，但有时保持一致也不错。发音为“Tim Toady Bicarbonate”。
- Easy things should be easy, and hard things should be possible. 简单的事情应该是简单的，复杂的事情应该变得可能。



## 优点

- 很容易：容易使用，但学习 Perl 并不简单（入门容易精通难）
- 几乎不受限制：几乎没有什么事是 Perl 办不到的
- 速度通常很快

## 缺点

- 灵活、随意和“过度”的冗余语法
- write-only：代码有点难看，令人难以阅读
- 解释器耗费资源



## 优点

- 很容易：容易使用，但学习 Perl 并不简单（入门容易精通难）
- 几乎不受限制：几乎没有什么事是 Perl 办不到的
- 速度通常很快

## 缺点

- 灵活、随意和“过度”的冗余语法
- write-only：代码有点难看，令人难以阅读
- 解释器耗费资源



# Perl | 优缺点 | JAPH

```
1 #JAPH(Just another Perl hacker) program without
2 obfuscation:
3
4 #Embedding JAPH in opaque code:
5 $_= '987;s/^(\d+)/$1-1/e;$1?eval:print"Just another
6 Perl hacker,"';eval;
7
8 #Decoding JAPH from a transposed string literal:
9 $_="krJhruaesrltre c a cnP,ohet";$_.=$1,
10 print$_while s/(..)(.)/;
11
12 #Appearing as if it does something completely
13 unrelated to printing JAPH:
14 $_ = "wftedskaebjgdpjgidbsmnjgc";
15 tr/a-z/oh, turtleneck Phrase Jar!/; print;
```



# Perl | 优缺点 | JAPH

```
@P=split//".URRUU\c8R";@d=split//"\nrekcah xinU / lreP rehtona tsuj";sub p{  
@p{"r$p","u$p"}=(P,P);pipe"r$p","u$p";++$p;($q*=2)+=$f=!fork;map{$P=$P[$f^ord  
($p{$_})&6];$p{$_}=/^$P/ix?$P:close$_}keys%p}p;p;p;p;p;map{$p{$_}=~/^P./&&  
close$_}%p;wait until?:map{/^r/&&<$ >%p;$_=d[$q];sleep rand(2)if/$/;print
```

"=~('(?{'.('-)@.)@\_\*([]@!@/)(@)@-@),@(@@+@)'  
^']])@]`}`]@`@.@[)%[`}%)[@`@!#@%[').',"]`}))



## Perl | 优缺点 | JAPH

```
not exp log srand xor s qq qx xor  
s x x length uc ord and print chr  
ord for qw q join use sub tied qx  
xor eval xor print qq q q xor int  
eval lc q m cos and print chr ord  
for qw y abs ne open tied hex exp  
ref y m xor scalar srand print qq  
q q xor int eval lc qq y sqrt cos  
and print chr ord for qw x printf  
each return local x y or print qq  
s s and eval q s undef or oct xor  
time xor ref print chr int ord lc  
foreach qw y hex alarm chdir kill  
exec return y s gt sin sort split
```



# Perl | 优缺点 | JAPH

```
1          #!/usr/bin/perl
2          use strict;
3
4          my$f=           $[;my
5          $ch=0;sub        l{length}
6          sub r{join"",   reverse split
7          ("",$_[$[]])}sub ss{substr($_[0]
8          ,$_[1],$_[2])}sub be{$_=$_[0];p
9          (ss($_,$f,1));$f+=1()/2;$f%1
10         ();$f++if$ch%2;$ch++}my$q=r
11         ("\ntfgpfdfal,ce?bngbjn".
12         "axfvxf"); $_=$q; $q=~"
13         tr/f[a-z]/ [l-za-k]
14         /;my@ever=1..&l
15         ;my$mine=$q
16         ;sub p{
17             print
18             @_;
19         }
20
21         be $mine for @ever
```



## 易于编程

Perl 的一些特性，使得解决常见的生物信息学问题非常容易。

## 快速成型

程序员可以快速写出实用的 Perl 程序（尤其是频率低或需频繁修改）。

## 可移植性好

可以在不同的计算机、操作系统上运行 Perl 程序。

## 速度还可以

先用 Perl 编写程序，再用 C 重写需要提速的部分。（开发时间远比运行时间宝贵！）

## 易于维护

程序是死的，人是活的。（维护程序不比编写程序容易。）

## 易于编程

Perl 的一些特性，使得解决常见的生物信息学问题非常容易。

## 快速成型

程序员可以快速写出实用的 Perl 程序（尤其是频率低或需频繁修改）。

## 可移植性好

可以在不同的计算机、操作系统上运行 Perl 程序。

## 速度还可以

先用 Perl 编写程序，再用 C 重写需要提速的部分。（开发时间远比运行时间宝贵！）

## 程序容易维护

程序是死的，人是活的。（维护程序不比编写程序容易。）

## 易于编程

Perl 的一些特性，使得解决常见的生物信息学问题非常容易。

## 快速成型

程序员可以快速写出实用的 Perl 程序（尤其是频率低或需频繁修改）。

## 可移植性好

可以在不同的计算机、操作系统上运行 Perl 程序。

## 速度还可以

先用 Perl 编写程序，再用 C 重写需要提速的部分。（开发时间远比运行时间宝贵！）

## 程序容易维护

程序是死的，人是活的。（维护程序不比编写程序容易。）

## 易于编程

Perl 的一些特性，使得解决常见的生物信息学问题非常容易。

## 快速成型

程序员可以快速写出实用的 Perl 程序（尤其是频率低或需频繁修改）。

## 可移植性好

可以在不同的计算机、操作系统上运行 Perl 程序。

## 速度还可以

先用 Perl 编写程序，再用 C 重写需要提速的部分。（开发时间远比运行时间宝贵！）

## 程序容易维护

程序是死的，人是活的。（维护程序不比编写程序容易。）

## 易于编程

Perl 的一些特性，使得解决常见的生物信息学问题非常容易。

## 快速成型

程序员可以快速写出实用的 Perl 程序（尤其是频率低或需频繁修改）。

## 可移植性好

可以在不同的计算机、操作系统上运行 Perl 程序。

## 速度还可以

先用 Perl 编写程序，再用 C 重写需要提速的部分。（开发时间远比运行时间宝贵！）

## 程序容易维护

程序是死的，人是活的。（维护程序不比编写程序容易。）

## Perl5

- 1994 年——至今
- 最新版：v5.24.0 (2016-05-09)

## Perl6

- 2000 年——至今 (2015-12-25 发布 v1.0)
- 最新版：Rakudo Star 2016.10 (2016-10-23)

## 版本选择

- 推荐 Perl5 (关注 Perl6) : 兼容性好, 稳定, 模块丰富, 资源丰富, .....
- (Perl5 vs. Perl6) vs. (Python2 vs. Python3)

## Perl5

- 1994 年——至今
- 最新版：v5.24.0 (2016-05-09)

## Perl6

- 2000 年——至今 (2015-12-25 发布 v1.0)
- 最新版：Rakudo Star 2016.10 (2016-10-23)

## 版本选择

- 推荐 Perl5 (关注 Perl6) : 兼容性好, 稳定, 模块丰富, 资源丰富, .....
- (Perl5 vs. Perl6) vs. (Python2 vs. Python3)

## Perl5

- 1994 年——至今
- 最新版：v5.24.0 (2016-05-09)

## Perl6

- 2000 年——至今 (2015-12-25 发布 v1.0)
- 最新版：Rakudo Star 2016.10 (2016-10-23)

## 版本选择

- 推荐 Perl5 (关注 Perl6) : 兼容性好, 稳定, 模块丰富, 资源丰富, .....
- (Perl5 vs. Perl6) vs. (Python2 vs. Python3)

# Perl | 版本 | Perl5 vs. Perl6

## Inline::Perl5

Use Perl 5 code in a Perl 6 program.

Let you run all 170,000+ Perl 5 modules while letting you get on with new Perl 6 development.

## Inline::Perl6

Use Perl 6 in Perl 5 code.

## Blue Tiger

Perl 5 to Perl 6 Translator.

## Perl::ToPerl6

Transmogrify Perl5 code into Perl6.

An extensible framework for creating and applying coding standards to Perl source code.

# Perl | 版本 | Perl5 vs. Perl6

## Inline::Perl5

Use Perl 5 code in a Perl 6 program.

Let you run all 170,000+ Perl 5 modules while letting you get on with new Perl 6 development.

## Inline::Perl6

Use Perl 6 in Perl 5 code.

## Blue Tiger

Perl 5 to Perl 6 Translator.

## Perl::ToPerl6

Transmogrify Perl5 code into Perl6.

An extensible framework for creating and applying coding standards to Perl source code.

# Perl | 版本 | Perl5 vs. Perl6

## Inline::Perl5

Use Perl 5 code in a Perl 6 program.

Let you run all 170,000+ Perl 5 modules while letting you get on with new Perl 6 development.

## Inline::Perl6

Use Perl 6 in Perl 5 code.

## Blue Tiger

Perl 5 to Perl 6 Translator.

## Perl::ToPerl6

Transmogrify Perl5 code into Perl6.

An extensible framework for creating and applying coding standards to Perl source code.

## Inline::Perl5

Use Perl 5 code in a Perl 6 program.

Let you run all 170,000+ Perl 5 modules while letting you get on with new Perl 6 development.

## Inline::Perl6

Use Perl 6 in Perl 5 code.

## Blue Tiger

Perl 5 to Perl 6 Translator.

## Perl::ToPerl6

Transmogrify Perl5 code into Perl6.

An extensible framework for creating and applying coding standards to Perl source code.

## 查看版本确认是否已安装

```
1 perl -v
```

### 官网

<https://www.perl.org/>

### Unix/Linux/Mac OS X

已经预装！

### Windows

- Strawberry Perl 【推荐】
- ActivePerl

## 查看版本确认是否已安装

```
1 perl -v
```

## 官网

<https://www.perl.org/>

## Unix/Linux/Mac OS X

已经预装！

## Windows

- Strawberry Perl 【推荐】
- ActivePerl

## 查看版本确认是否已安装

```
1 perl -v
```

## 官网

<https://www.perl.org/>

## Unix/Linux/Mac OS X

已经预装！

## Windows

- Strawberry Perl 【推荐】
- ActivePerl

## 查看版本确认是否已安装

```
1 perl -v
```

## 官网

<https://www.perl.org/>

## Unix/Linux/Mac OS X

已经预装！

## Windows

- [Strawberry Perl 【推荐】](#)
- [ActivePerl](#)

## 基本策略

先学习基础知识，当需要时再学习相关主题。

80/20 定律：Perl 里面 80% 的功能可以用文档中 20% 的部分加以描述，而另外 20% 的功能却需要占据其他 80% 的篇幅。（扩展：80% 的任务可以用 20% 的知识解决，20% 的任务则需要剩余 80% 的知识。）

## 各种资源

- 自带文档 (`man perl`)
- 在线文档
- 新闻组
- FAQs
- 邮件列表
- 书籍
- .....

## 基本策略

先学习基础知识，当需要时再学习相关主题。

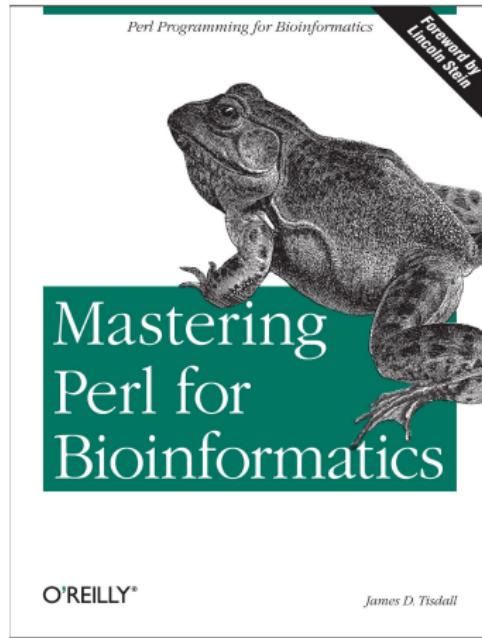
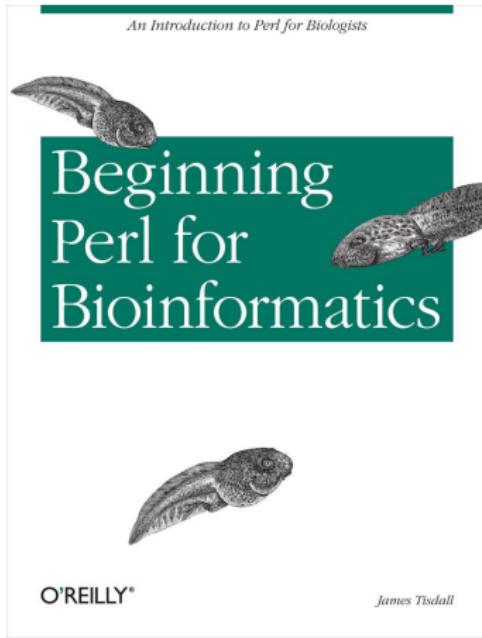
80/20 定律：Perl 里面 80% 的功能可以用文档中 20% 的部分加以描述，而另外 20% 的功能却需要占据其他 80% 的篇幅。（扩展：80% 的任务可以用 20% 的知识解决，20% 的任务则需要剩余 80% 的知识。）

## 各种资源

- 自带文档 (`man perl`)
- 在线文档
- 新闻组
- FAQs
- 邮件列表
- 书籍
- .....

## 生物信息学角度

*Beginning*  $\Rightarrow$  *Mastering Perl for Bioinformatics*



## 编程语言角度：代号

小骆驼 ⇒ 羊驼书 ⇒ 小羊驼母亲和她的孩子 ⇒ 大骆驼 ⇒ 黑豹书

## 英文名

*Learning Perl* ⇒ *Intermediate Perl* ⇒ *Mastering Perl* ⇒ *Programming Perl*  
⇒ *Advanced Perl Programming*

## 中文名

《Perl 语言入门》⇒ 《Perl 进阶》⇒ 《精通 Perl》⇒ 《Perl 语言编程》  
⇒ 《高级 Perl 编程》



## 编程语言角度：代号

小骆驼 ⇒ 羊驼书 ⇒ 小羊驼母亲和她的孩子 ⇒ 大骆驼 ⇒ 黑豹书

## 英文名

*Learning Perl* ⇒ *Intermediate Perl* ⇒ *Mastering Perl* ⇒ *Programming Perl*  
⇒ *Advanced Perl Programming*

## 中文名

《Perl 语言入门》⇒ 《Perl 进阶》⇒ 《精通 Perl》⇒ 《Perl 语言编程》  
⇒ 《高级 Perl 编程》



## 编程语言角度：代号

小骆驼 ⇒ 羊驼书 ⇒ 小羊驼母亲和她的孩子 ⇒ 大骆驼 ⇒ 黑豹书

## 英文名

*Learning Perl* ⇒ *Intermediate Perl* ⇒ *Mastering Perl* ⇒ *Programming Perl*  
⇒ *Advanced Perl Programming*

## 中文名

《Perl 语言入门》⇒ 《Perl 进阶》⇒ 《精通 Perl》⇒ 《Perl 语言编程》  
⇒ 《高级 Perl 编程》



## 编程语言角度：代号

小骆驼 ⇒ 羊驼书 ⇒ 小羊驼母亲和她的孩子 ⇒ 大骆驼 ⇒ 黑豹书

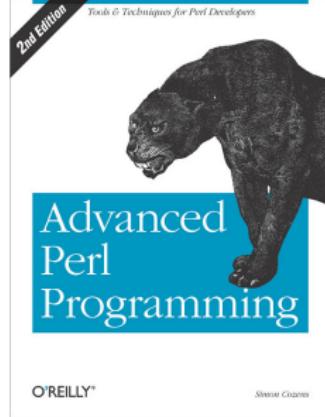
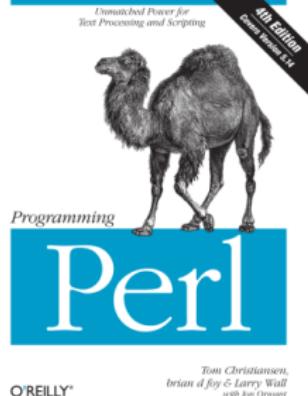
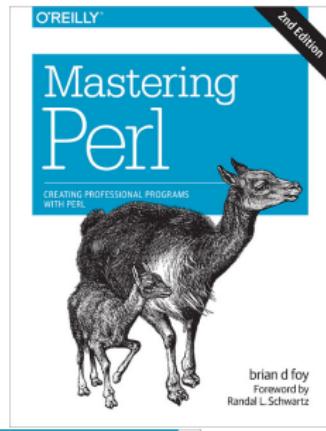
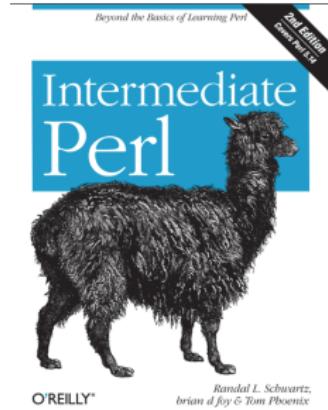
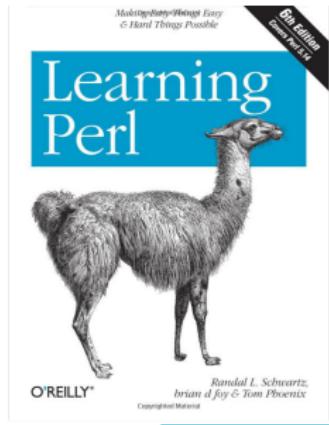
## 英文名

*Learning Perl* ⇒ *Intermediate Perl* ⇒ *Mastering Perl* ⇒ *Programming Perl*  
⇒ *Advanced Perl Programming*

## 中文名

《Perl 语言入门》⇒ 《Perl 进阶》⇒ 《精通 Perl》⇒ 《Perl 语言编程》  
⇒ 《高级 Perl 编程》





## 中文名

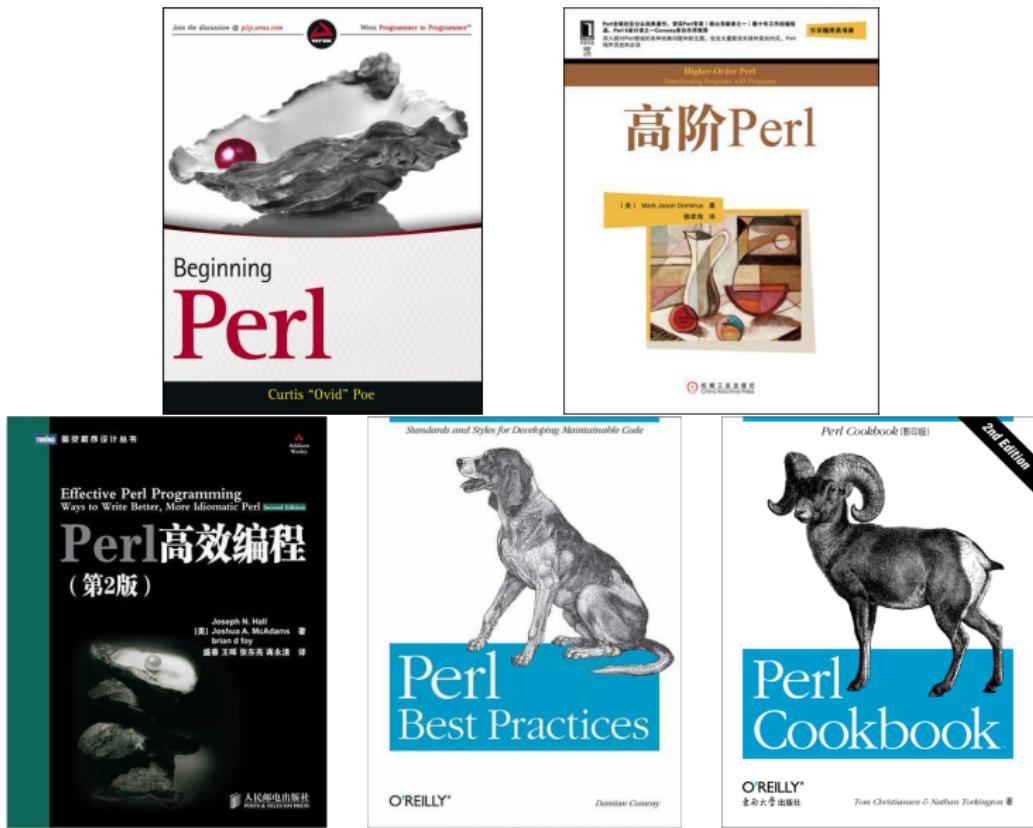
- 《Perl 入门经典》
- 《高阶 Perl》
- 《Perl 高效编程》
- 《Perl 最佳实践》
- *Perl Cookbook*

## 英文名

- *Beginning Perl*
- *Higher-Order Perl*
- *Effective Perl Programming*
- *Perl Best Practices*
- *Perl Cookbook*



# 书籍 | Perl



```
1 #!/usr/bin/perl
2 # Classic "Hello World" as done with Perl
3
4 use strict;
5 use warnings;
6
7 print "Hello World!\n";
```

```
1 # Step 1: 编写脚本
2 vim hello.pl
3 # Step 2: 修改权限
4 chmod 755 hello.pl
5 # Step 3: 运行脚本
6 perl hello.pl
7 ./hello.pl
```



```
1 #!/usr/bin/perl
2 # Classic "Hello World" as done with Perl
3
4 use strict;
5 use warnings;
6
7 print "Hello World!\n";
```

```
1 # Step 1: 编写脚本
2 vim hello.pl
3 # Step 2: 修改权限
4 chmod 755 hello.pl
5 # Step 3: 运行脚本
6 perl hello.pl
7 ./hello.pl
```



## Vim 插件

[perl-support.vim](#): Perl IDE – Write and run Perl-scripts using menus and hotkeys.

- insert various types of comments
- insert complete but empty statements (e.g. ‘if {} else {}’)
- insert often used code snippets (e.g. declarations, the opening of files, .. )
- insert the names of file tests, character classes, special Perl-variables and POSIX-signals
- read, write, maintain your own code snippets in a separate directory
- run scripts or run syntax check from within the editor
- show compilation errors in a quickfix window; navigate with hotkeys
- read perldoc for functions and modules
- run perltidy / run the profiler SmallProf
- test / explain regular expressions (needs Vim with Perl interface)



## 检查语法

```
1 perl -c script.pl
```

## 常见错误

- Missing right curly or square bracket: 括号不匹配
- 语句末尾少了分号
- 关键字（内置函数）拼写错误
- 变量类型符号错误/变量名拼写错误
- .....



## 检查语法

```
1 perl -c script.pl
```

## 常见错误

- Missing right curly or square bracket: 括号不匹配
- 语句末尾少了分号
- 关键字（内置函数）拼写错误
- 变量类型符号错误/变量名拼写错误
- .....



```
1 # 安装perltidy
2 sudo apt install perltidy
3 #sudo apt-get install perltidy
4
5 # 保留原始文件，格式化后的文件为*.tdy
6 perltidy script.pl
7
8 # 备份原始文件为*.bak, 格式化后的文件使用原文件名
9 perltidy -b script.pl
10
11 # perltidy使用帮助
12 man perltidy
13 perltidy -h
```



```
1 #!/usr/bin/perl
2
3 use strict;use warnings;use utf8;
4
5 for(my$i=1;$i<=9;$i++) {for(my$j=1;$j<=$i;$j++)
6   ++){print"$j"."x"."$i"."=". $i*$j;
7   if($j==$i) {print"\n"; } else{print"\t";}}}
```



# Perl | 格式化 | 后

```
1 #!/usr/bin/perl
2
3 use strict;
4 use warnings;
5 use utf8;
6
7 for ( my $i = 1 ; $i <= 9 ; $i++ ) {
8     for ( my $j = 1 ; $j <= $i ; $j++ ) {
9         print "$j" . "x" . "$i" . "=" . $i * $j;
10        if      ( $j == $i ) { print "\n"; }
11        else                  { print "\t"; }
12    }
13 }
```



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



# 变量

## 变量

Perl 是一种无类型语言 (untyped) , 换句话说, 在语言层面上, Perl 和大多数编程语言不同, 不把变量分成整数、字符、浮点数等等, 而只有一种能接受各种类型数据的“无类型”变量。

Perl 中各种变量的运算也很自由, 数和含有数的字符串是等效的, 可以把数字字符串参与数学计算, 也可以反之, 让数字参与字符串的构成和操作。

## 类型

- 标量: scalar; 只包含一个元素的变量; 以 \$ 开头
- 数组: array; 含有任意数量元素的变量, 以其存储顺序作为索引; 以 @ 开头
- 散列: hash, associative array (关联数组); 像字典一样, 把不同的变量按照它们的逻辑关系组织起来, 并以作为“键”的变量进行索引; 以 % 开头

# 变量

## 变量

Perl 是一种无类型语言 (untyped) , 换句话说, 在语言层面上, Perl 和大多数编程语言不同, 不把变量分成整数、字符、浮点数等等, 而只有一种能接受各种类型数据的“无类型”变量。

Perl 中各种变量的运算也很自由, 数和含有数的字符串是等效的, 可以把数字字符串参与数学计算, 也可以反之, 让数字参与字符串的构成和操作。

## 类型

- 标量: scalar; 只包含一个元素的变量; 以 \$ 开头
- 数组: array; 含有任意数量元素的变量, 以其存储顺序作为索引; 以 @ 开头
- 散列: hash, associative array (关联数组); 像字典一样, 把不同的变量按照它们的逻辑关系组织起来, 并以作为“键”的变量进行索引; 以 % 开头

# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



# 变量 | 标量 | 语法

```
1 # 标量: $scalar
2
3 # 字符串, 双引号
4 $name = "Paul";
5
6 # 数字, 无引号
7 $age = 29;
8
9 # 字符串, 单引号
10 $where_to_find_him = 'http://www.weinstein.org';
```



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

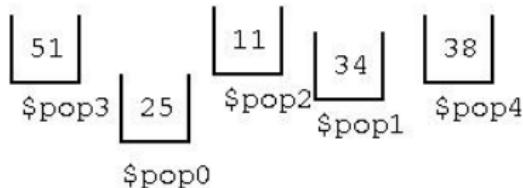
10 回顾与总结

- 总结

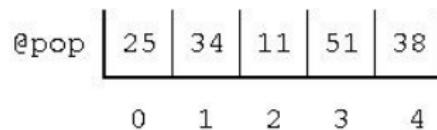
- 思考题



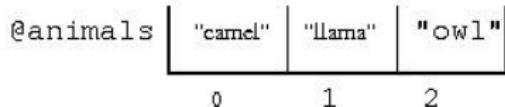
# 变量 | 数组 | 简介



Individual scalar variables



Array



# 变量 | 数组 | 语法

```
1 # 数组: @array
2 # 索引值从0开始
3
4 # 字符串
5 @authors = ("Paul", "Joe", "Jeremy", "Harley"
   );
6 $authors[4] = "Tom";
7
8 # 数字
9 @list = (1, 2, 3, 4);
10
11 # 解引用: $array[index]
12 $authors[4] # Tom
13 $list[0] # 1
```



## unshift

( A, C, G, T)

**shift**

push

pop

The diagram shows a 3D perspective view of an array with three elements. The first element contains a green plant icon, the second contains a red car icon, and the third contains the number '47'. Above the array, an arrow labeled 'Shift' points to the left, indicating that the entire array has shifted one position to the left. Below the array, an arrow labeled 'Unshift' points to the right, indicating that the array has been restored to its original state.

M

A 3D-style box divided into four sections. The first section contains a green potted plant icon. The second section contains a red car icon. The third section contains the number '47'. The fourth section contains the word 'plant'.



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

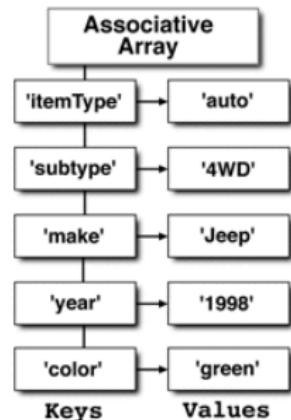
9 检修脚本

10 回顾与总结

- 总结

- 思考题





%Perez

Age	21
Name	"Jeff"
Eyes	"Br"
Temp	98.6

\$Perez{'Eyes'} = "Br";

%Kleiner

Age	21
Name	"Dave"
Eyes	"Blue"
Temp	99.0

\$Kleiner{'Temp'} = 99.0;



# 变量 | 散列 | 语法

```
1 # 散列: %hash
2
3 # 创建散列
4 %person = (
5 name => 'Paul',
6 age => '29',
7 url => 'http://www.weinstein.org',
8 )
9
10 # 提取键值: $hash{key}
11 $person{"age"} # 29
```



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



# 变量 | 内置变量

Perl 提供了大量的预定义变量。下面列举了常用的一些预定义变量：

\$_	在执行输入和模式搜索操作时使用的默认空格变量
\$. \$.	文件中最后处理的当前行号
\$@	由最近一个 eval() 运算符提供的 Perl 语法报错信息
\$!	获取当前错误信息值，常用于 die 命令
\$0	含有正在执行的程序名
\$\$	正在执行本脚本的 Perl 进程号
\$PERL_VERSION / \$^V	Perl 解释器的版本、子版本和修订版本信息
@ARGV ARGV	含有命令行参数
@INC	一个特殊的文件句柄，用于遍历 @ARGV 中出现的所有文件名
@_	库文件的搜索路径
%ENV	在子例程中， @_ 变量含有传给该子例程的变量内容
%SIG	关联数组型变量 %ENV 含有当前环境信息
	关联数组型变量 %SIG 含有指向信号内容的句柄



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



# 操作符 | 数字操作符

操作符	含义
+	加法
-	减法
*	乘法
/	除法
**	乘幂, 乘方
%	取模, 取余
<	小于
>	大于
==	等于
<=	小于等于
>=	大于等于
!=	不等于
<=>	比较。a<=>b: a 等于 b 时返回 0, a 大于 b 时返回 1, a 小于 b 时返回-1



# 操作符 | 字符串操作符

操作符	含义
.	连接。 "string1" . "string2"
x	重复。 "string" x number
lt	小于
gt	大于
eq	等于
le	小于等于
ge	大于等于
ne	不等于
cmp	比较。类似于数字比较的<=>



# 操作符 | 逻辑操作符

操作符	含义
&&	逻辑 AND, 与
	逻辑 OR, 或
!	逻辑 NOT, 非
?=	条件操作符



# 操作符 | 文件测试操作符

操作符	含义
-r	可读
-w	可写
-x	可执行
-e	存在
-z	存在但没有内容
-s	存在且有内容
-f	普通文件
-d	目录
-l	符号链接
-T	看起来像文本文件
-B	看起来像二进制文件
-M	最后被修改后至今的天数
-A	最后被访问后至今的天数
-C	最后 inode 变更后至今的天数



# 操作符 | 匹配操作符

操作符	含义
<code>=~</code>	绑定操作符，匹配
<code>!~</code>	绑定操作符，不匹配
<code>~~</code>	智能匹配操作符



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



# 基本函数 | print, chomp

```
1 # 向标准输出打印文本
2 print "Hello Again\n";
3
4 # 向一个具有文件句柄的文件打印文本
5 print FILE "Hello Again\n";
6
7 # 打印变量的值
8 print "How are on this day, the " . $date . "?\n";
```

```
1 # 删除变量末尾的（多个）换行符，返回删除的换行符的个数
2 chomp $name;
3 chomp @authors;
```

# 基本函数 | print, chomp

```
1 # 向标准输出打印文本
2 print "Hello Again\n";
3
4 # 向一个具有文件句柄的文件打印文本
5 print FILE "Hello Again\n";
6
7 # 打印变量的值
8 print "How are on this day, the " . $date . "?\n";
```

```
1 # 删除变量末尾的（多个）换行符，返回删除的换行符的个数
2 chomp $name;
3 chomp @authors;
```



# 基本函数 | join, split

```
1 # joining a number of strings together with a
   colon delimiter
2 $fields = join ':', $data_field1,
   $data_field2, $data_field3;
3
4 # splitting a string into substrings
5 ($field1, $field2) = split /:/, 'Hello:World'
   , 2;
6
7 # splitting a scalar and creating an array
8 @fields = split /:/, $raw_data;
```



# 基本函数 | open, close

```
1 # open the file and slurp its contents into
  an array and then close the file
2 open(FILE, "/etc/passwd");
3 @filedata = <FILE>; #囫囵吞枣
4 close(FILE);
5
6 open my $IN, '<', $file_in or die "$0 :
  failed to open input file '$file_in' : $!\n";
7 while(<$IN>) { #细嚼慢咽
8   chomp;
9   #...actions...
10 }
11 close $IN or warn "$0 : failed to close input
  file '$file_in' : $!\n";
```



# 基本函数 | opendir, readdir, closedir

```
1 #!/usr/bin/perl -w
2
3 print "Read user's home directory\n";
4 opendir(HOMEDIR, ".");
5 @ls = readdir HOMEDIR;
6 closedir(HOMEDIR);
7
8 print "Create file dirlist.txt with a
      directory listing of user's home dir\n";
9 open(FILE, ">dirlist.txt");
10 foreach $item (@ls) {
11     print FILE $item . "\n";
12 }
13 close(FILE);
14 print "All done\n\n";
```



# 基本函数 | my, local

```
1 # Global variable $name is given a name
2 $name = "Paul";
3
4 # Enter our loop
5 foreach (@filedata) {
6     # declare a new variable for just the loop
7     my $current_file;
8
9     # create a local version of name to
10    # temporarily assign values within the loop
11    # to
12    local $name;
13 }
```



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

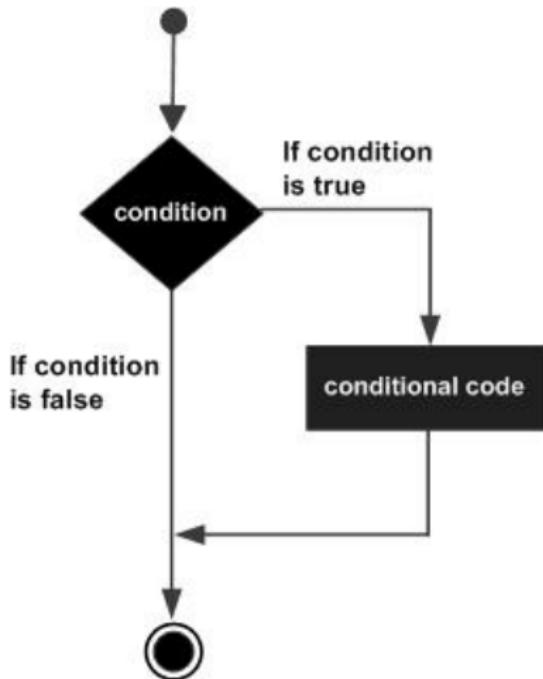
10 回顾与总结

- 总结

- 思考题



# 判断语句 | if | 逻辑流程

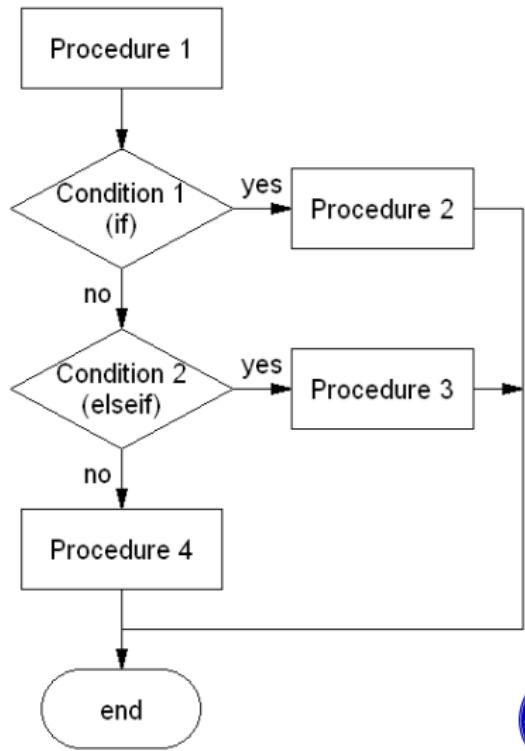
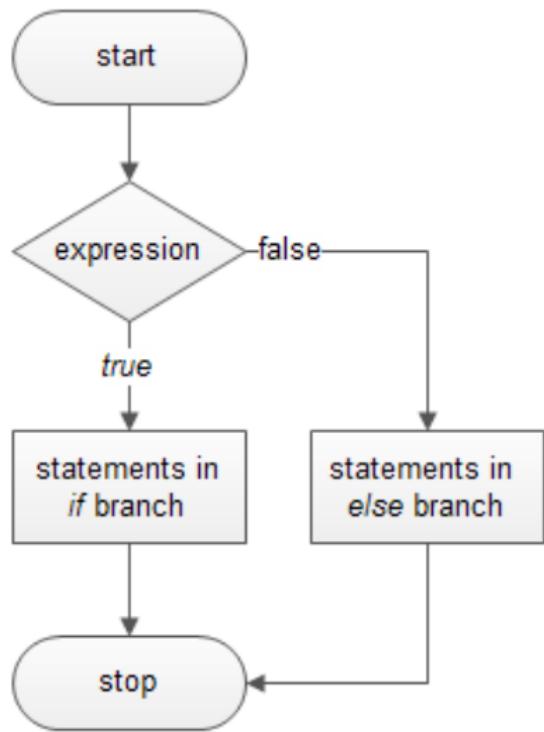


# 判断语句 | if | 语法

```
1 # if区块
2 if ($hour > 22) {
3     print "should sleep...\n";
4 }
5
6 # if语句
7 print "hello" if $guest >= 1;
```



# 判断语句 | if-else | 逻辑流程



# 判断语句 | if-else | 语法

```
1 if ($name eq "Paul") {  
2     print "Hi Paul\n";  
3 }  
4 elsif ($name eq "Joe") {  
5     print "Hi Joe\n";  
6 }  
7 elsif ($name eq "Jeremy") {  
8     print "Hi Jeremy\n";  
9 }  
10 else {  
11     print "Sorry, have we meet before?";  
12 }
```



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

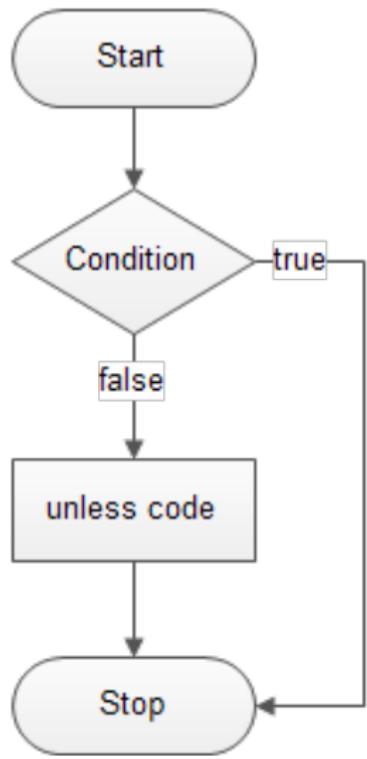
10 回顾与总结

- 总结

- 思考题



## 判断语句 | unless | 逻辑流程



# 判断语句 | unless | 语法

```
1 # unless区块
2 unless ($credit > 100) {
3     print "You can not graduate!\n";
4 }
5
6 # unless语句
7 print "eat!\n" unless $food==0;
```



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

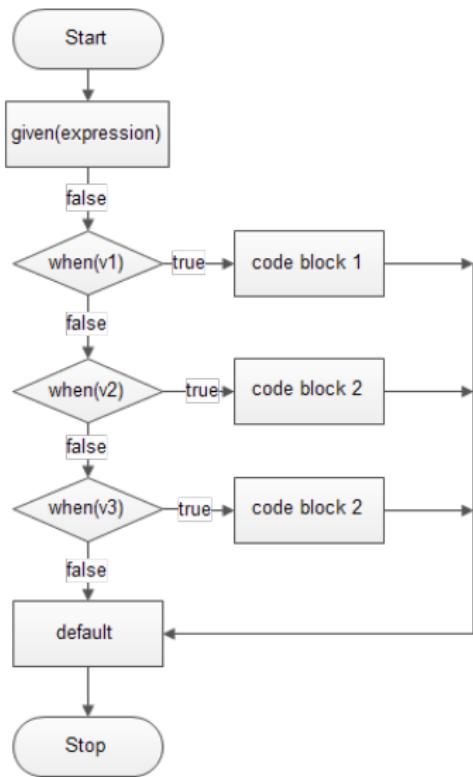
10 回顾与总结

- 总结

- 思考题



## 判断语句 | given-when | 逻辑流程



# 判断语句 | given-when | 语法

```
1 use 5.010;
2 given ($foo) {
3     say "a" when "a";
4     when (/b/) {say "b";}
5     default {say "not match";}
6 }
```



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7

## 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8

## 列表操作

- sort

- grep

- map

- 高效排序

9

## 检修脚本

10

## 回顾与总结

- 总结

- 思考题



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7

## 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8

## 列表操作

- sort

- grep

- map

- 高效排序

9

## 检修脚本

10

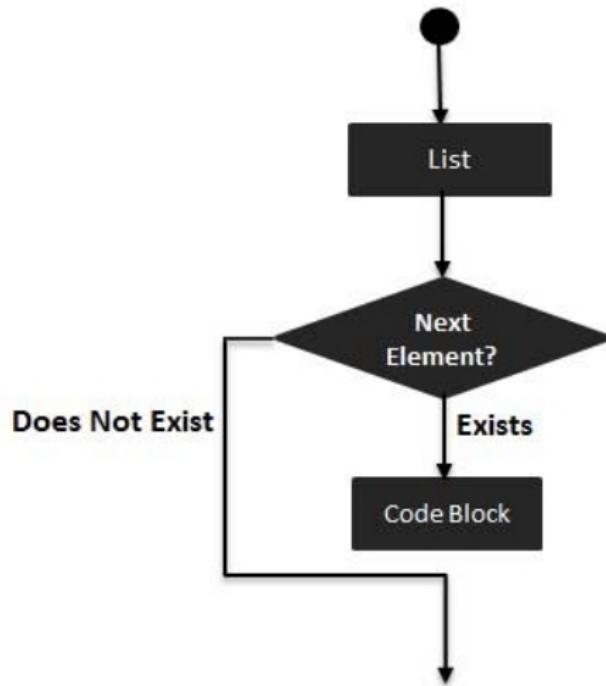
## 回顾与总结

- 总结

- 思考题



# 循环语句 | foreach | 逻辑流程



# 循环语句 | foreach | 语法

```
1 @group = 1..10;
2
3 # foreach循环
4 foreach my $element (@group) {
5     print "$element\n";
6 }
7
8 # 等价的for循环
9 for (@group) {
10    print "$_\n";
11 }
12 print "$_\n" for @group;
```



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7

## 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8

## 列表操作

- sort

- grep

- map

- 高效排序

9

## 检修脚本

10

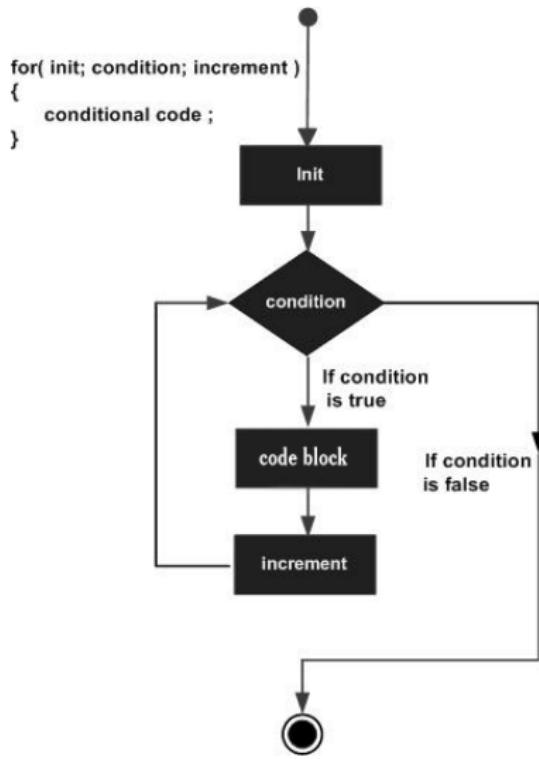
## 回顾与总结

- 总结

- 思考题



## 循环语句 | for | 逻辑流程



# 循环语句 | for | 语法

```
1 # 从1数到10
2 for ($i = 1; $i <= 10; $i++) {
3     print "I can count to $i!\n";
4 }
```



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7

## 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8

## 列表操作

- sort

- grep

- map

- 高效排序

9

## 检修脚本

10

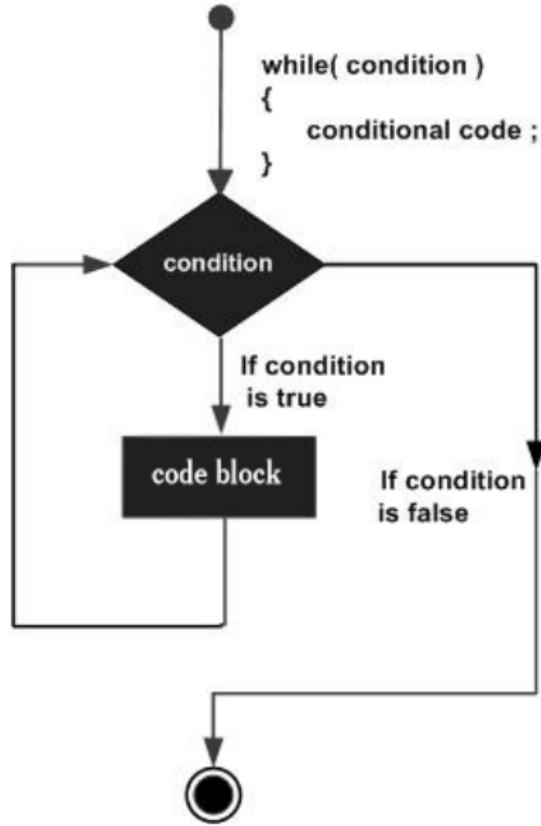
## 回顾与总结

- 总结

- 思考题



## 循环语句 | while | 逻辑流程

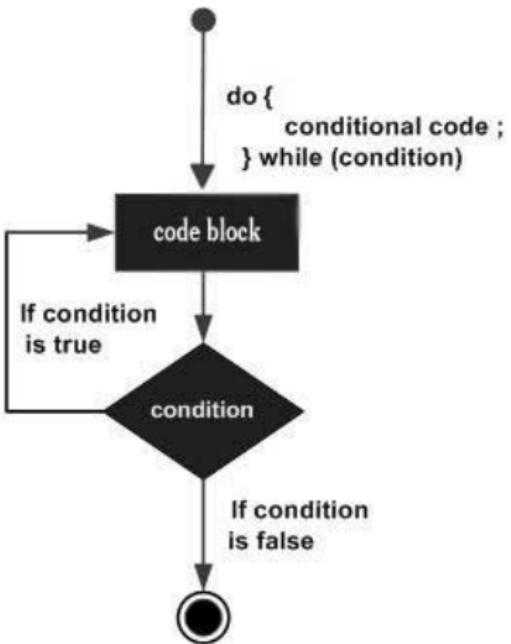


# 循环语句 | while | 语法

```
1 $i = 0;
2 while ($i < 10) {
3     print "$i\n";
4     $i++;
5 }
```



# 循环语句 | do-while | 逻辑流程



# 循环语句 | do-while | 语法

```
1 $i = 0;  
2 do {  
3     print "$i\n";  
4     $i = $i + 1;  
5 } while ($i < 10);
```



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7

## 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8

## 列表操作

- sort

- grep

- map

- 高效排序

9

## 检修脚本

10

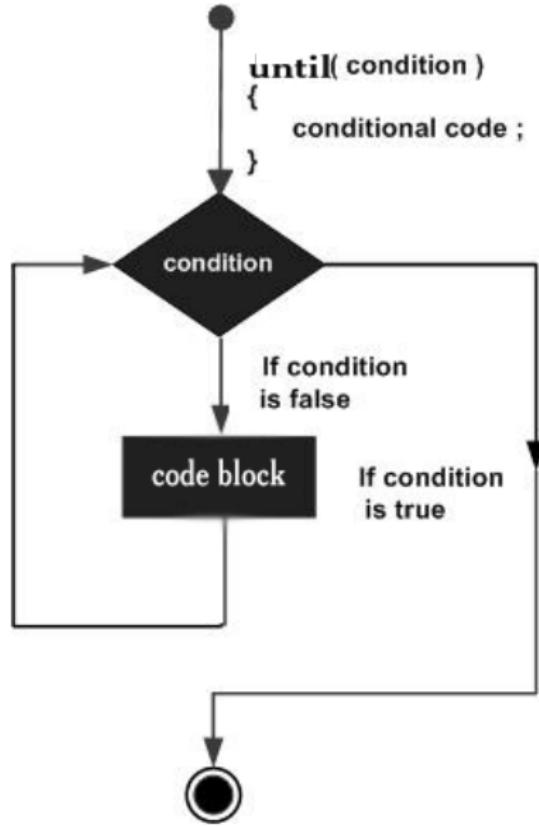
## 回顾与总结

- 总结

- 思考题



## 循环语句 | until | 逻辑流程

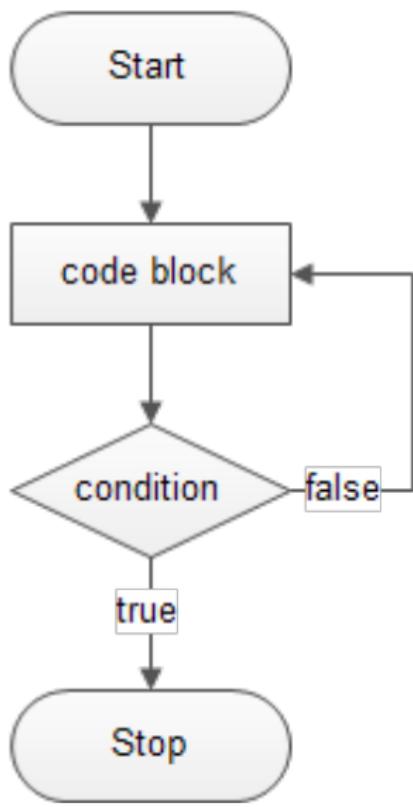


# 循环语句 | until | 语法

```
1 $i = 0;
2 until ($i == 10) {
3     print "$i\n";
4     $i++;
5 }
```



## 循环语句 | do-until | 逻辑流程



# 循环语句 | do-until | 语法

```
1 $i = 0;  
2 do {  
3     print "$i\n";  
4     $i++;  
5 } until ($i == 10);
```



## 比较

- 比较 while 和 do-while
- 比较 until 和 do-until

## 提示

- 如果一开始的条件测试就为假，两者的表现有何差异？



## 比较

- 比较 while 和 do-while
- 比较 until 和 do-until

## 提示

- 如果一开始的条件测试就为假，两者的表现有何差异？



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



## 列表操作

基于原来的列表创建一个新的列表：

`sort` 将给定列表排序后返回一个新的已排序列表

- 按字母顺序排序
- 按数字顺序排序
- 反向排序
- 运用 `sort` 子程序进行排序

`grep` 筛选列表以寻找符合标准的表项

- 基于原列表筛选大于 0 的元素建立一个新列表

`map` 将一个列表转换成另一个列表

- 将原列表中的所有元素都乘以 2 得到新列表



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



# 列表操作 | sort | 语法

```
1 sort LIST
2 sort BLOCK LIST
3 sort SUBNAME LIST
```

```
1 @passengers = sort @passengers;
2 @passengers = sort { $a->{age} <=> $b->{age} }
    } @passengers;
3 @passengers = sort women_and_children_first
    @passengers;
```



# 列表操作 | sort | 语法

```
1 sort LIST
2 sort BLOCK LIST
3 sort SUBNAME LIST
```

```
1 @passengers = sort @passengers;
2 @passengers = sort { $a->{age} <=> $b->{age} }
    } @passengers;
3 @passengers = sort women_and_children_first
    @passengers;
```



# 列表操作 | sort | 默认排序

```
1 # sort默认是通过字符串的比较来进行排序
2 my @list = sort qw(this is a list);
3 print "@list";
4 # a is list this
5
6 # sort默认将字符串分成单个字符来进行排序
7 print join ",", sort qw/1 9 10 99 222/;
8 # 1,10,222,9,99
9
10 # 使用sort代码块将数字按照数字顺序进行排序
11 print join ",", sort { $a <=gt; $b } qw /1 9 10
   99 222/;
12 # 1,9,10,99,222
```



# 列表操作 | sort | 反向排序

```
1 # 执行效率较低: 升序排列列表 + 反转列表 = 降序排列列表
2 my @reversed_names = reverse sort @names;
3
4 # 直接反向排序列表
5 # 反向排序字符串
6 my @reversed_names = sort { $b cmp $a }
    @names;
7 # 降序排序数字
8 my @descending = sort { $b <= $a } @numbers;
```



# 列表操作 | sort | 复杂排序

```
1 my @employees = (
2     {
3         name      => 'Sally Jones',
4         years     => 4,
5         payscale  => 4,
6     },
7     {
8         name      => 'Abby Hoffman',
9         years     => 1,
10        payscale  => 10,
11    },
12    {
13        name      => 'Jack Johnson',
14        years     => 4,
15        payscale  => 5,
16    },
17    {
18        name      => 'Mr. Magnate',
19        years     => 12,
20        payscale  => 1,
21    },
22 );
```



# 列表操作 | sort | 复杂排序

```
1 @employees =
2     sort {
3         $b->{years}      <=> $a->{years}
4         ||
5         $a->{payscale} <=> $b->{payscale}
6         ||
7         $a->{name}       cmp $b->{name}
8     }
9     @employees;
```



# 列表操作 | sort | 复杂排序

```
1 # 将sort代码块放到子程序中，并用子程序的名称替代sort  
2 # 代码块  
3  
4 sub by_seniority_then_pay_then_name {  
5     $b->{years}      <=> $a->{years}  
6     ||  
7     $a->{payscale} <=> $b->{payscale}  
8     ||  
9     $a->{name}       cmp $b->{name}  
10 }  
11  
12 @employees = sort  
13     by_seniority_then_pay_then_name @employees;  
14 # 当排序条件很复杂时，使用一个命名的排序子程序：  
15 # 1. 提高代码的可读性  
16 # 2. 便于重复使用
```



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



## grep

生成一个新的列表，该列表中的所有元素由一个给定列表中所有满足筛选标准的元素组成。

```
1 NEWLIST = grep BLOCK      LIST;
2 NEWLIST = grep EXPRESSION, LIST;
3 # 注意逗号的有无!
```

```
1 my @greater = grep { $_[ > 0 } @numbers;
2 my @greater = grep $_[ > 0, @numbers;
3 my @greater = grep ( $_[ > 0, @numbers );
```



## grep

生成一个新的列表，该列表中的所有元素由一个给定列表中所有满足筛选标准的元素组成。

```
1 NEWLIST = grep BLOCK      LIST;
2 NEWLIST = grep EXPRESSION, LIST;
3 # 注意逗号的有无!
```

```
1 my @greater = grep { $_ > 0 } @numbers;
2 my @greater = grep $_ > 0, @numbers;
3 my @greater = grep ( $_ > 0, @numbers );
```



## 原理

在使用 grep 函数时，通过轮流地把每个元素设置成 `$_` 的方式，可以实现对列表中每个元素的迭代处理。grep 函数只返回在 BLOCK 或 EXPRESSION 中值为真的那些元素。可以在 grep 函数中使用任意复杂的表达式。

```
1 # 获取回文字符串
2 my @palindromes = grep { uc eq reverse uc }
  @words;
3
4 # 查找给定列表中以元音字母开头的单词
5 my @starts_with_vowels = grep { /^[aeiou]/ }
  @words;
```



# 列表操作 | grep | 组合 sort

```
1 my @numbers = ( 13, 3, -2, 7, 270, 19, -3.2,
2   10.1 );
3 my @result = sort { $a <=gt; $b } grep { $_ >=
4   10 } @numbers;
5 print join ", ", @result;
6 # 10.1, 13, 19, 270
7
8 # 更加清晰易读
9 my @result = sort { $a <=gt; $b }
10           grep { $_ >= 10 } @numbers;
```



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



## map

在想要改变整个列表内容时使用 map。它逐个对给定列表中的元素进行处理，然后新建一个列表，并将修改后的值返回到新的列表中。

```
1 NEWLIST = map BLOCK           LIST;
2 NEWLIST = map EXPRESSION, LIST;
```

```
1 # 将列表中的每一个字母大写
2 my @UPPER = map { uc } @words;
3
4 my @roots = map { sqrt($_) }
5             grep { $_ > 0 } @numbers;
```



## map

在想要改变整个列表内容时使用 map。它逐个对给定列表中的元素进行处理，然后新建一个列表，并将修改后的值返回到新的列表中。

```
1 NEWLIST = map BLOCK           LIST;
2 NEWLIST = map EXPRESSION, LIST;
```

```
1 # 将列表中的每一个字母大写
2 my @UPPER = map { uc } @words;
3
4 my @roots = map { sqrt($_) }
5                 grep { $_ > 0 } @numbers;
```



# 列表操作 | map | 实例

```
1 # 将华氏温度转换为摄氏温度
2 my @fahrenheit = (
3     'absolute zero'      => -459.67,
4     'freezing water'     => 32,
5     'body temperature'   => 98.6,
6     'boiling water'      => 212,
7 );
8
9 my %celsius = map
10    { $_ => 5 / 9 * ($fahrenheit{$_} - 32) }
11    keys %fahrenheit;
```



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



## Orcish Maneuver

```
1 keys my %or_cache = @in;
2 @out = sort {
3     ($or_cache{$a} ||= foo($a))
4     cmp
5     ($or_cache{$b} ||= foo($b))
6 } @in;
```



# 列表操作 | 高效排序

## Schwartzian transform

```
1 # The general form of the Schwartzian Transform:  
2 @sorted = map { $_[0] }  
3         sort { $a->[1] cmp $b->[1] }  
4         map { [$_, foo($_)] }  
5         @unsorted;  
6  
7 # sort the DNA list ("ACGT", "A", "AC", "ACG")  
8 # according to the sequence length  
9 @sorted_DNA = map { $_[0] }  
10        # 3. extract DNA sequence  
11        sort { $a->[1] <=> $b->[1] }  
12        # 2. use numeric comparison  
13        map { [$_, length($_)] }  
14        # 1. calculate the length of the DNA  
15        @unsorted_DNA;
```

# 列表操作 | 高效排序

## Schwartzian transform

```
1 # James|007|Spy
2 # Number 6|6|Ex-spy
3 # Agent 99|99|Spy with unknown name
4 # Napoleon Solo|11|Uncle spy
5 # Unknown|666|Maybe a spy
6
7 my @sorted = map { $_[0] }
8     # 3. undecorate
9     sort { $a->[1] <=> $b->[1] }
10    # 2. sort
11    map { /\|(\d+)/; [ $_[1], $1 ] }
12        # 1. decorate
13        <>;
```

## Guttman-Rosler Transform

```
1 # James|007|Spy
2 # Number 6|6|Ex-spy
3 # Agent 99|99|Spy with unknown name
4 # Napoleon Solo|11|Uncle spy
5 # Unknown|666|Maybe a spy
6
7 my @sorted = map { substr $_, 4 } 
8     sort
9     map { /\|(\d+)/; pack("A4", $1).$_ } <>;
```



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



# 检修脚本

```
1 # 不洁模式
2 #!/usr/bin/perl -T
3
4 # 打开警告
5 use warnings;
6
7 # 严格模式，语法更加规范
8 use strict;
9
10 # 开启诊断功能：提供扩展描述来指明具体哪部分程序出了问题，以
11 # 及关于如何解决这些问题的建议
12 # 对于初学者非常有用，有经验的程序员通常将其省略
13 use diagnostics;
14
15 # 启用UTF-8编码
use utf8;
```



# 检修脚本

```
1 # 检查语法  
2 perl -c script.pl  
3  
4 # 格式化脚本  
5 perltidy script.pl  
6  
7 # 调试脚本  
8 perl -d script.pl
```



命令	功能
s	仅执行脚本中的一行
n	按步执行以避免进入子程序
c	继续运行直至遇到断点
r	继续运行直至从当前子程序中执行返回命令
v	显示当前行前后的代码
b	设置断点
x	显示变量的值
w	设置监视表达式
T	显示栈回溯追踪
L	列出所有的断点
B	删除断点
R	重新启动脚本以便再次测试



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



## 知识点

- Perl 语言简介：基本知识，中心思想，优缺点，语法结构
- 变量：标量，数组，散列，内置变量
- 操作符：数字、字符串、逻辑、文件测试、匹配操作符
- 基本函数：print, chomp, join, split, open, close, my
- 判断语句：if, unless, given-when
- 循环语句：foreach, for, while, until
- 列表操作：sort, grep, map
- 检修脚本：检查语法，格式化脚本，调试脚本

## 技能

- 掌握 Perl 语言的基本语法
- 使用 Perl 编写简单的应用脚本

# 教学提纲

1 引言

2 Perl 简介

3 变量

- 标量

- 数组

- 散列

- 内置变量

4 操作符

5 基本函数

6 判断语句

- if 语句

- unless 语句

- given-when 语句

7 循环语句

- foreach 语句

- for 语句

- while 语句

- until 语句

8 列表操作

- sort

- grep

- map

- 高效排序

9 检修脚本

10 回顾与总结

- 总结

- 思考题



- ① Perl 语言的中心思想是什么？
- ② Perl 中的变量主要有哪三大类？
- ③ 列举 Perl 中的各种操作符。
- ④ chomp, join, split 的作用是什么？
- ⑤ 在 Perl 中如何和文件进行交互？
- ⑥ Perl 中的判断语句有哪些，语法是怎样的？
- ⑦ Perl 中的循环语句有哪些，语法是怎样的？
- ⑧ 列举 Perl 中进行列表操作的函数。
- ⑨ 如何调试 Perl 脚本？



- 在面对问题的时候，你是如何构思解决策略的？
- 在编程中，你是怎样保存和备份程序的？
- 你是如何一步一步从零开始写出一个可用的程序的？



# Powered by



T<sub>E</sub>X L<sup>A</sup>T<sub>E</sub>X X<sub>E</sub>T<sub>E</sub>X Beamer