

# 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

2018-2019 学年上学期 (秋)  
2016 级生信班

## 第三章 编程的艺术

伊现富 (Yi Xianfu)

天津医科大学 (TIJMU)  
生物医学工程与技术学院

2018 年 11 月



# 教学提纲

1 引言

2 学习方法

3 编写程序

4 编程策略

5 编程过程

6 编程真言

7 回顾与总结

- 总结

- 思考题

1 引言

2 学习方法

3 编写程序

4 编程策略

5 编程过程

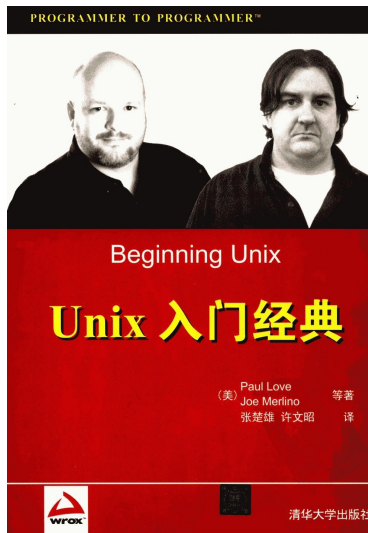
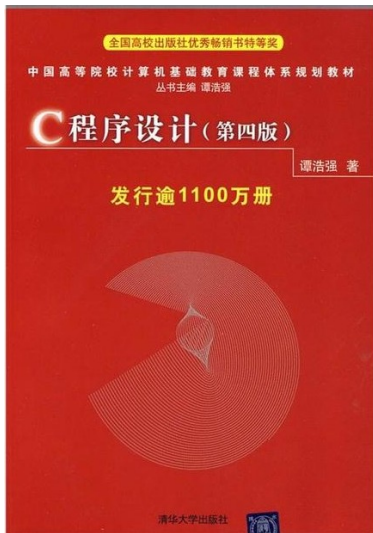
6 编程真言

7 回顾与总结

- 总结

- 思考题





# 教学提纲

- 1 引言
- 2 学习方法**
- 3 编写程序
- 4 编程策略

- 5 编程过程
- 6 编程真言
- 7 回顾与总结
  - 总结
  - 思考题



## 提问

学习编程的最佳方法是什么？

## 回答

- 取决于你要完成的任务
- 取决于你打算如何学习编程
- 取决于……



## 提问

学习编程的最佳方法是什么？

## 回答

- 取决于你要完成的任务
- 取决于你打算如何学习编程
- 取决于……





## 常见方法

- 参加 (XXX 新手) 培训班
- 阅读 (XXX 入门、30 天学会 XXX) 书籍
- 死啃手册
- 拜师学艺
- 研究经典程序
- .....
- 组合多种方法

## 五字真言

- 实践出真知！
- Experience is the best teacher.
- 不要只读书/看手册/读源代码，一定要亲自动手去编写、调试程序。

## 常见方法

- 参加 (XXX 新手) 培训班
- 阅读 (XXX 入门、30 天学会 XXX) 书籍
- 死啃手册
- 拜师学艺
- 研究经典程序
- .....
- 组合多种方法

## 五字真言

- 实践出真知！
- Experience is the best teacher.
- 不要只读书/看手册/读源代码，一定要亲自动手去编写、调试程序。

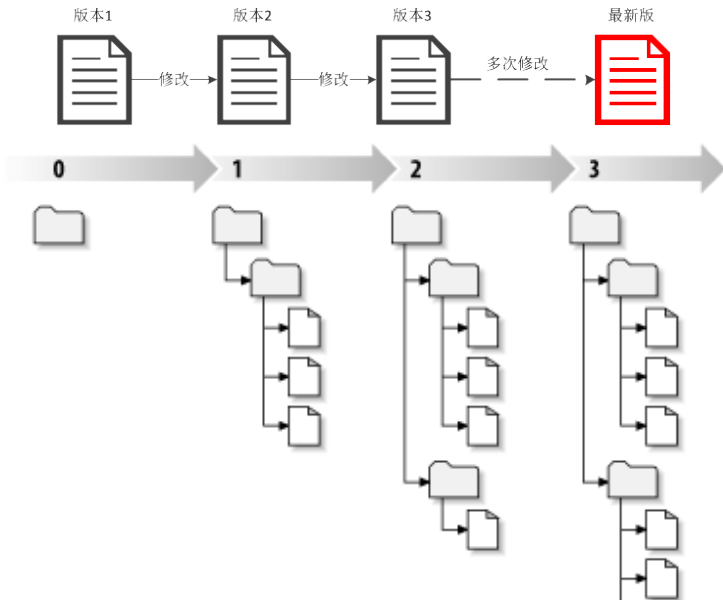
# 教学提纲

- 1 引言
- 2 学习方法
- 3 编写程序
- 4 编程策略
- 5 编程过程
- 6 编程真言
- 7 回顾与总结
  - 总结
  - 思考题





# 编程艺术 | 编写程序 | 版本控制



什么是版本控制？我真的需要吗？版本控制是一种记录若干文件内容变化，以便将来查阅特定版本修订情况的系统。

如果你是位图形或网页设计师，可能会需要保存某一幅图片或页面布局文件的所有修订版本（这或许是你非常渴望拥有的功能）。采用版本控制系统（VCS, Version Control System）是个明智的选择。有了它你就可以将某个文件回溯到之前的状态，甚至将整个项目都回退到过去某个时间点的状态。你可以比较文件的变化细节，查出最后是谁修改了哪个地方，从而导致出现怪异问题，又是谁在何时报告了某个功能缺陷等等。使用版本控制系统通常还意味着，就算你乱来一气把整个项目中的文件改的改删的删，你也照样可以轻松恢复到原先的样子。但额外增加的工作量却微乎其微。

许多人习惯用复制整个项目目录的方式来保存不同的版本，或许还会改名加上备份时间以示区别。这么做唯一的好处就是简单。不过坏处也不少：有时候会混淆所在的工作目录，一旦弄错文件丢了数据就没法恢复。



Git 是一个**分散式版本控制软件**，最初由**林纳斯·托瓦兹 (Linus Torvalds)**创作，于 2005 年以 GPL 释出。最初目的是为更好地管理 Linux 内核开发而设计。

Git 是用于 Linux 内核开发的版本控制工具。与 CVS、Subversion 一类的集中式版本控制工具不同，它采用了**分布式版本库**的做法，不需要服务器端软件，就可以运作版本控制，使得源代码的发布和交流极其方便。Git 最为出色的是它的合并追踪 (merge tracing) 能力。

在 Git 中的绝大多数操作都只需要访问本地文件和资源，不用连网。因为 Git 在本地磁盘上就保存着所有当前项目的历史更新，所以处理起来速度飞快。



Git 和其他版本控制系统的主要差别在于，Git 只关心文件数据的整体是否发生变化，而大多数其他系统则只关心文件内容的具体差异。

这类系统（CVS，Subversion，Perforce，Bazaar 等等）每次记录有哪些文件作了更新，以及都更新了哪些行的什么内容。

Git 并不保存这些前后变化的差异数据。实际上，Git 更像是把变化的文件作快照后，记录在一个微型的文件系统中。每次提交更新时，它会纵览一遍所有文件的指纹信息并对文件作一快照，然后保存一个指向这次快照的索引。为提高性能，若文件没有变化，Git 不会再次保存，而只对上次保存的快照作一链接。





```
1 # 安装Git
2 sudo apt install git-core
3 #sudo apt-get install git-core
4
5 # 使用帮助
6 man git
7 git --help
8 git help CMD
9
10 # 创建项目目录
11 mkdir ~/project
12 cd ~/project
```



```
1 # 创建Git仓库 (启动版本控制)
2 git init
3
4 # 创建编辑文件
5 vim script.pl
6 #print "Hello, world!";
7
8 # 添加需要进行版本控制的文件
9 git add script.pl
10 #git add .
11
12 # 提交改动信息
13 git commit -m "Say hello to the world."
```



```
1 # 修改文件
2 vim script.pl
3 #把Hello替换成Bye
4
5 # 添加改动信息
6 git add .
7
8 # 提交改动信息
9 git commit -m "Bye to the world."
10
11 # 查看提交历史
12 git log
13
14 # 版本回退
15 git reset --hard ID #不需要全部ID, 只需要有区分度
    的前几位即可
```



```
1 # 创建test分支并切换过去
2 git checkout -b test
3 #相当于两步: git branch test; git checkout test
4
5 # 修改文件
6 vim script.pl
7 #添加一行: print "Bye, world!";
8
9 # 添加改动信息
10 git add .
11
12 # 提交改动信息
13 git commit -m "And bye to the world."
```



```
1 # 切换回主分支
2 git checkout master
3
4 # 把test分支合并到主分支
5 git merge test
6 #可能需要手动修改后执行git add和git commit命令
7
8 # 删除test分支
9 git branch -d test
```



```
1 # 查看状态
2 git status
3
4 # 查看提交日志
5 git log
6
7 # 查看修改内容
8 git diff
9
10 # 删除文件
11 git rm
12
13 # 查看/创建标签
14 git tag
```



```
1 # 配置Git
2
3 #查看配置信息
4 git config --list
5
6 #彩色的 git 输出:
7 git config color.ui true
8 #显示历史记录时, 只显示一行注释信息
9 git config format.pretty oneline
10
11 #配置个人信息等
12 git config --global user.name "Yixf"
13 git config --global user.email "yixf@example.
    com"
14 git config --global core.editor vim
```



```
1 # 内置的图形化Git
2 gitk
3
4 # 忽略文件/文件夹
5 #将相关信息添加到.gitignore文件中
6 videos/
7 *.pdf
8 *.doc
```





## Basic Git Workflow Example

Initialize a new git repository, then stage all the files in the directory and finally commit the initial snapshot.

```
$ git init
$ git add .
$ git commit -m 'initial commit'
```

Create a new branch named featureA, then check it out so it is the active branch. then edit and stage some files and finally commit the new snapshot.

```
$ git branch featureA
$ git checkout featureA
$ (edit files)
$ git add (files)
$ git commit -m 'add feature A'
```

Switch back to the master branch, reverting the featureA changes you just made, then edit some files and commit your new changes directly in the master branch context.

```
$ git checkout master
$ (edit files)
$ git commit -a -m 'change files'
```

Merge the featureA changes into the master branch context, combining all your work. Finally delete the featureA branch.

```
$ git merge featureA
$ git branch -d featureA
```



GitHub 是一个共享虚拟主机服务，用于存放使用 Git 版本控制的软件代码和内容项目。

GitHub 同时提供付费账户和免费账户。这两种账户都可以建立公开的代码仓库，但是付费账户也可以建立私有的代码仓库。除了允许个人和组织建立和存取代码库以外，它也提供了一些方便社会化软件开发的功能，包括允许用户跟踪其他用户、组织、软件库的动态，对软件代码的改动和 Bug 提出评论等。GitHub 也提供了图表功能，用于显示开发者们怎样在代码库上工作以及软件的开发活跃程度。



截止到 2018 年 6 月，GitHub 已经有超过二千八百万注册用户和五千七百万代码仓库。事实上已经成为了**世界上最大的代码存放网站**。

2018 年 6 月 4 日晚上，美国科技公司微软宣布以 75 亿美元的股票收购 GitHub。

GitHub 里面的项目可以通过标准的 Git 命令进行访问和操作。同时，所有的 Git 命令都可以用到 GitHub 项目上面。



```
1 # 克隆仓库
2 #克隆本地仓库
3 git clone /path/to/repository
4 #克隆远程服务器上的仓库到本地
5 git clone username@host:/path/to/repository
6
7 # 把本地已有的仓库和服务器上的仓库关联起来
8 git remote add origin <server>
9
10 # 把本地库的内容推送到远程库
11 git push origin master
12
13 # 把远程库的内容更新到本地库
14 git pull
```

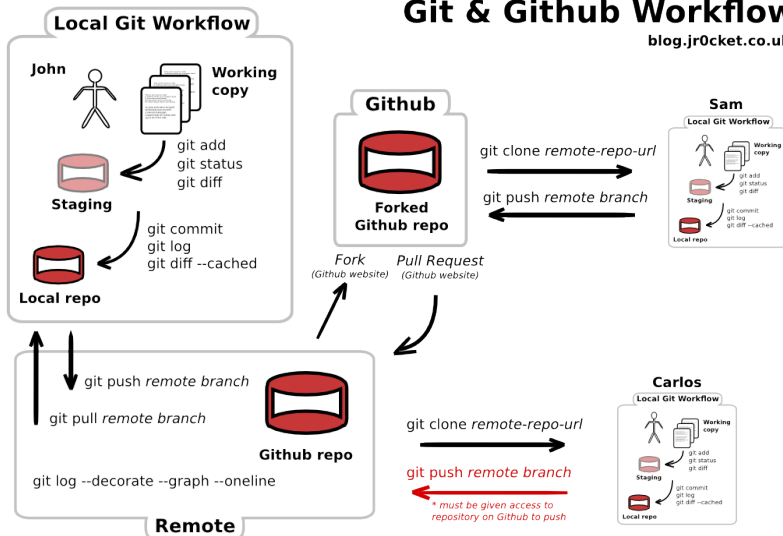


```
1 # 克隆GitHub仓库到本地计算机 (的当前目录下)
2 git clone https://github.com/Yixf-Education/
  project_Perl.git
3 cd project_Perl # 进入项目目录
4 # 更新本地仓库 (从GitHub中拉取最新变化)
5 #git pull
6
7 # 常规Git操作 (...)
8 #vim script.pl; git add script.pl
9 #git commit -m "Fix typos in script.pl"
10
11 # 推送修改 (把本地修改上传到GitHub仓库中)
12 git push -u origin master # 第一次: 关联本地的
  master和远端的origin
13 git push origin master # 以后: 就可以简化命令了
```



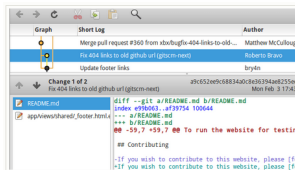
## Git & Github Workflow

blog.jr0cket.co.uk



## GUI Clients

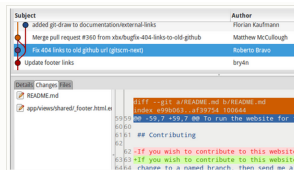
### ● GUI Clients



**gitg**

Platforms: Linux

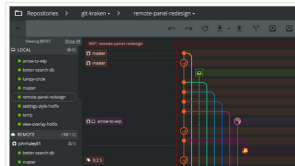
Price: Free



**gitg**

Platforms: Linux

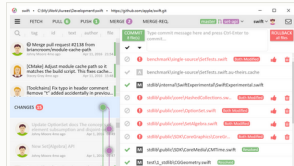
Price: Free



**GitKraken**

Platforms: Windows, Mac, Linux

Price: Free for non-commercial use



**Aureores**

Platforms: Windows

Price: \$69.99/user / Free for non-commercial use



- 出错并不可怕！（【编程初期】出错是非常正常的。）
- 一定不会对错误信息视而不见！
- 从第一个错误开始，逐个进行修复。
- 必要时进行一定的猜测。





- 使用 Perl 调试器：`perl -d script.pl`。
- 在程序中加入 `print` 语句，输出中间值。
- 选择性地注释掉部分代码。
- 使用相关的模块：`Benchmark`，`Data::Dumper`，`Smart::Comments`，……
- 组合使用多种调试方法
- ……



# 教学提纲

- 1 引言
- 2 学习方法
- 3 编写程序
- 4 编程策略**

- 5 编程过程
- 6 编程真言
- 7 回顾与总结
  - 总结
  - 思考题



- ❶ 寻找现成的（免费/收费）程序：避免“重复发明轮子”
- ❷ 自己编写程序
  - ❶ 修改现成的程序（平时注意收集、整理程序）
  - ❷ 充分利用已有模块，快速“拼凑”程序
  - ❸ 从头编写完整的程序
- ❸ 请其他专家（无偿/有偿）编写程序
- ❹ 组合使用上述多种策略

## 注意

有时修改现成的程序可能会比从头编写一个完整的程序还要困难！

- 背景知识
- 处理思路
- 编程风格
- ...

- ❶ 寻找现成的（免费/收费）程序：避免“重复发明轮子”
- ❷ 自己编写程序
  - ❶ 修改现成的程序（平时注意收集、整理程序）
  - ❷ 充分利用已有模块，快速“拼凑”程序
  - ❸ 从头编写完整的程序
- ❸ 请其他专家（无偿/有偿）编写程序
- ❹ 组合使用上述多种策略

## 注意

有时修改现成的程序可能会比从头编写一个完整的程序还要困难！

- 背景知识
- 处理思路
- 编程风格
- ...

1 引言

2 学习方法

3 编写程序

4 编程策略

5 编程过程

6 编程真言

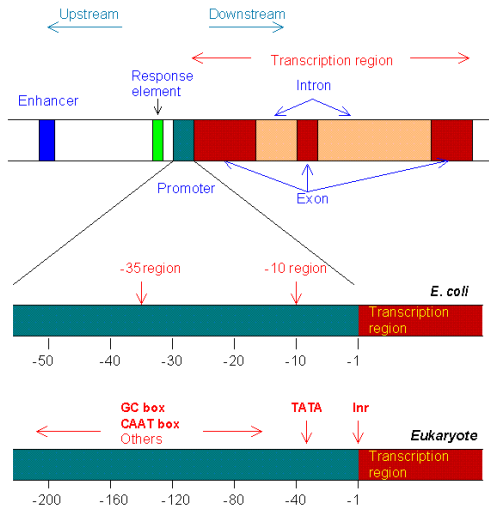
7 回顾与总结

- 总结

- 思考题



## 计算一个 DNA 序列中调控元件的数目。



- 分析任务属性，对其进行充分的理解（数量、频率、时间等）
- 确定输入数据，对其进行充分理解（数据量、格式等）
- 对程序进行整理构思（算法、数据结构等）
- 确定输出数据，包括输出方法（文件、图形化展示、管道）、格式等
- 进一步改善整体构思，根据输入、输出等信息添加细节内容
- 必要时编写伪代码整理思路
- **最后**才是动手编写程序代码



- 程序构思：在实际编程前首先要完成的关键步骤
- 分析任务属性：任务数量、任务频率、解决任务的时间限制等
- 确定输入数据：数据来源（文件、输入等）、数据数量、数据校验（文件存不存在、格式对不对）等
- 选择正确/合适的算法（速度、优劣）：针对每一个调控元件，在 DNA 序列中从头到尾进行查找；针对 DNA 序列的每一个位置，对每个调控元件进行查找
- 确定输出数据：输出形式、数据格式、人性化输出（用户提供文件名、易于解读……）等
- 选择编程范式：命令式编程（把一个大的问题/程序分割成多个微小、但却相互关联配合的部分/子程序），程序式编程，面向对象编程
- 编写伪代码：整理思路、优化构思、调整细节……





- 分析任务属性：任务数量、处理频率、时间限制等
- 确定输入输出：数据格式、数据量、数据校验、输出形式等
- 选择合适算法：算法速度、算法难易、对应数据结构等
- 编写伪代码：整理思路、优化构思、选择编程范式、调整细节等
- 编写程序代码：编辑、调试、运行、完善等



在数学和计算机科学/算学之中，算法（algorithm）为一个**计算的具**  
**体步骤**，常用于计算、数据处理和自动推理。精确而言，算法是一个表示为有限长列表的有效方法。算法应包含清晰定义的指令用于计算函数。

算法中的指令描述的是一个计算，当其运行时能从一个初始状态和初始输入（可能为空）开始，经过一系列有限而清晰定义的状态最终产生输出并停止于一个终态。

程序所做的事情：获取文件、打开文件、读入数据、进行计算、输出结果；而算法就是此过程中**计算的思路**。



伪代码 (pseudocode)，又称为虚拟代码，是高层次描述算法的一种方法。它不是一种现实存在的编程语言；它可能综合使用多种编程语言的语法、保留字，甚至会用到自然语言。

它以编程语言的书写形式指明算法的职能。相比于程序语言，它更类似自然语言。它是半形式化、不标准的语言。我们可以将整个算法运行过程的结构用接近自然语言的形式（这里可以使用任何一种作者熟悉的文字，例如中文、英文，重点是将程序的意思表达出来）描述出来。使用伪代码，可以帮助我们更好得表述算法，不用拘泥于具体的实现。

人们在用不同的编程语言实现同一个算法时意识到，他们做出来的实现（而非功能）很不同。程序员要理解一个用他并不熟悉的编程语言编写的程序，可能是很困难的，因为程序语言的形式限制了程序员对程序关键部分的理解。伪代码就这样应运而生了。

当考虑算法功能（而不是其语言实现）时，伪代码常常得到应用。计算机科学在教学中通常使用伪代码，以使得所有的程序员都能理解。



伪代码是介于自然语言和编程语言之间的一种“中间语言”。

## 代码

```
1 sub getanswer {  
2     print "Type in your answer here :";  
3     my $answer = <STDIN>;  
4     chomp $answer;  
5     return $answer;  
6 }
```

## 伪代码

```
1 getanswer
```

伪代码是介于自然语言和编程语言之间的一种“中间语言”。

## 代码

```
1 sub getanswer {  
2   print "Type in your answer here :";  
3   my $answer = <STDIN>;  
4   chomp $answer;  
5   return $answer;  
6 }
```

## 伪代码

```
1 getanswer
```

```
1 get the name of DNAfile from the user
2
3 read in the DNA from the DNAfile
4
5 for each regulatory element
6     if element is in DNA, then
7         add one to the count
8
9 print count
```



- 注释是源代码的一部分，旨在帮助用户/程序员理解程序
- 从 # 开始到行末的所有内容都被看做是注释，会被 Perl 解释器忽略掉
- 首行的 `#!/usr/bin/perl` 不是注释，不会被 Perl 解释器忽略掉
- 注释内容：程序的目的、整体构思、使用实例、细节注释等
- 牢记：代码不止是被计算机看的，也会被人查看
- 可以通过注释掉伪代码把它们保留在程序中



# 教学提纲

1 引言

2 学习方法

3 编写程序

4 编程策略

5 编程过程

6 编程真言

7 回顾与总结

- 总结

- 思考题





## 方法论——学习

- 首次尝试某种事情时，最好从最基本的地方开始。你对**基础**理解得越好，以后你理解复杂问题时越轻松。
- **学习编程**和学习编程语言是截然不同的。应该掌握的不是答案，而是**思考方法**。学会靠自己的能力解决，而不是去记答案。
- 我们在做任何事情时，都需要 100 个小时后才能入门。
- 学习编程的两种方法：其一，是掌握工具，然后寻找能够用到的地方。其二，是找出要用的地方，然后掌握必要的工具。
- 任何事情都需要反复的**练习**，编程也是同样的道理。
- 在程序的世界中，**英语**才是标准语言。
- 程序中的用语和习惯很多都受到了**数学**的影响。
- 编程中的思考方式和技术，不同语言之间的差别并不大。



## 方法论——理念

- 在写任何程序之前最好先做个**计划**。想要达成目的，首先要思考从何处着手。
- 在拿到一个任务后，不要立即进行代码编写，而要先对任务进行**审视和分析**，确定合适的策略后方可高效地解决问题。
- 无论是多么大的目标，都有其第一步。如果不实际着手，就无法开始。而且，关于第一步应该做什么，需要自己来思考。
- “**先树立目标，再考虑手段**”是编程时的基本思想。“**先打造框架，再填充内容**”的想法是非常基础的。
- 不建议在弄清问题前急于寻找工具。
- 在大多数情况下，比起编写时花费的时间，**修改错误以及进行维护时需要花费的时间更长**。
- 在尝试新内容的时候，尽量从**小的试验**开始。
- 现实生活中司空见惯的事情在程序的世界中并不一定理所当然。

## 方法论——思路

- 当你受到挫折或者面临太大的挑战时：
  - ① **拆分**：把大问题拆成小问题，只考虑困难问题的一小部分
  - ② **搁置**：把它放到一边一段时间，先不去理它，过几天再回来
- **一次只对程序做小的一点改动**，这是因为我们最好一边做一边试验它是否好用。假如我们一次性把所有代码都写好，然后才发现它不工作，那我们要到哪里去找原因呢？
- 在我们希望推测程序的内部构造的时候，**稍加改造**后观察其变化是最基本的方法。
- 最好将**可变要素**控制为一个。如果同时修改两三个的话，就不知道是哪一项影响了最终的结果。
- 想要修正错误，从现在的运行**结果去推测原因**会比较快。
- 当我们要达成的事情比较复杂时，需要将达成的过程加以拆分，为每个过程设定目标，然后思考每个目标的达成手段。

## 方法论——补遗——通用

- 精通的人往往无法理解不懂的人。
- 所谓的**最佳方法**，会根据程序的不同、人的不同，甚至情绪的不同而发生变化。
- 进行准备工作也要分具体情况，避免做出多余的准备。到底应该做多少准备，需要根据具体情况而定。这需要经验的积累。
- 如果有精力的话，最好能想出两个以上的做法，将它们加以比较，然后选出其中较好的那个。如此一来就不需要陷入思维怪圈，而且做法行不通时也不用过于紧张。
- 永远正确的做法是不存在的。
- 当面对两个不同的方法时，一方独领风骚的情况非常少见。一般都是各有优势，需要根据当时的情况加以选择，即需要采取“扬长避短”的做法。
- 每个人都有着不同的生活状态，对事物抱着不同的看法、不同的感受。

## 方法论——补遗——编程

- 程序不是按你想的运行，而是按你写的运行。
- “开始无法按照预期运行，然后一遍遍地修改再试”——这种循环是不可避免的。先写再修正，再写再修正……
- 适合编程的人和不适合编程的人之间存在显著的差异。差异不在于记忆力的强弱、脑子的灵活程度或者计算能力的高低这些单纯的内容，而在于思考方式与世界观的差别。与其说谁优谁劣，不如说是性格或者个性的问题。
- 编程需要养成的习惯：
  - 从结果出发
  - 提问的方法非常重要，需要好好对问题加以变形
  - 带着目的，将其分解为更容易立即着手的小目标
  - 对出现的手段是否符合自己的目的加以确认
  - 准备多个选项，进行比对、选择

## 计算机科学家

- 计算机科学家与数学家类似，他们使用形式语言来描述理念（特别是计算）；与工程师类似，他们设计产品，将元件组装成系统，对不同的方案进行评估选择；与自然科学家类似，他们观察复杂系统的行为，构建科学家说，并检验其预测。
- 作为计算机科学家，最重要的技能就是**问题求解**。问题求解是发现问题、创造性地思考解决方法以及清晰准确地表达解决问题的能力。实践证明，学习编程的过程，正是训练问题求解能力的绝佳机会。



## 程序员

- 好的程序员不愿意重复做同一件事情，应尽全力找出省事的方案。
- 勤奋的程序员是不对的。最终能够成功的，是知道如何**偷懒**的人。
- **自我探索并解决问题的精神**被公认为是一个软件工程师应有的基本素质之一。
- **新手**在进行编程时，或者编写复杂的内容时，就应该一步一个脚印，频繁地进行测试。而**专家**在编写简单的内容时，一次进行大量的改造反而有助于把握好整体结构，也便于提高效率。



## 计算机

- 计算机是人的工具，只有人才知道他自己想让计算机做什么。
- 计算机就是对一切内容都通过计算进行处理的东西，计算之外的内容完全不明白。
- 计算机是由“运行程序的装置”和“大量的内存”这两者组成的。
- 计算机是按照程序中的内容操纵存储区，让外部设备运转的设备。
- 计算机只能理解数字。
- 计算机无非就是一种“**自动化设备**”，也就是处理复杂内容的帮手。当你感到麻烦时，不妨想想能否交给计算机来处理。“麻烦”在编程中是极为重要的信号。





## 程序

- 计算机程序是一组让计算机执行某种动作的指令。计算机程序就是一系列告诉没有知觉的硬件做什么事情的命令。程序有点像**思想**。思想告诉你做什么，计算机程序告诉计算机做什么。
- 程序究竟能做什么？它只能修改保存在存储区中的数字。程序就是由操纵存储区的命令组合而成的。程序就是告诉计算机“按什么顺序让哪个存储区记住哪个数字”的东西，它所能做的只有**操纵存储区**罢了。
- 所谓程序，就是记录着如何改变存储区所存数值的文章。通过操纵存储区，能够对与存储区相连的设备施加影响。程序是按照“以什么顺序”“向哪个存储区”“保存哪个数字”写下的东西。
- 软件就是计算机程序的集合。



## 编程与编程语言

- 编程会培养**创造能力、逻辑能力和解决问题的能力**，从中学到的技巧对于学校和工作都很有用。学习编程是一种很好的**思维训练**。
- 一种编程语言就是一种特定的与计算机**交谈的方式**，它使用计算机和人都能理解的指令。
- 程序设计过程就是**加工处理数据**的过程。输入数据给程序，经过程序处理后，输出处理结果。
- 在不同的编程语言中，程序的细节有所不同，但几乎所有编程语言中都会出现以下几类**基本指令**：

**输入** 从键盘、文件或者其他设备中获取数据

**输出** 将数据显示到屏幕，保存到文件中，发送到网络上等

**数学** 进行基本数学操作，如加法或乘法

**条件执行** 检查某种条件的状态，并执行相应的代码

**重复** 重复执行某种动作，往往在重复中有一些变化

- 可以把编程看作一个**将大而复杂的任务分解为更小的子任务的过程**，不断分解，直到任务简单到足以由基本指令组合完成。

## 编程的本质

- 要编写程序，就必须知道编程语言的规则。语法指的是程序中文字的组织 and 顺序。虽然语法和词汇的使用方法不得不记，但仅仅记住这些是无法写出文章的。记住编程语言和记住**编程方法**不是一回事。
- 计算机上的诸多功能，都是通过**用程序修改存储区**的方法来实现的。
- 操纵外部装置需要使用存储区，使用外部装置进行操作时还是要用存储区。向外部装置传递信息或者发出指示的时候，只要操纵存储区即可。
- “如果这样，如果那样”这种逻辑，在计算机看来，不过是数字之间加减乘除的结合罢了。
- 编程的本质：不断在“易懂但量大”和“复杂却量少”之间挣扎。



## 算法与数据结构

- 编程的核心是数据结构。 **程序 = 数据结构 + 算法**
- 所谓数据结构就是指一类数据的内部构成，即一类数据由哪些成员数据构成，以什么方式构成，有什么特点。
- 算法好比武术中的种种套路，告诉你先干什么，后干什么。数据结构却好比我们手中的兵刃，性能好坏导致使用效果大相径庭。
- 在开始一个程序代码的编写前，选择合适的数据结构可以帮助我们更好地完成和实现目标功能，并且有助于后期算法的实现。



## 注释

- 在复杂和难以理解的地方加入**注释**也是编程的一种好习惯。好的注释可以充当程序的说明文档。
- 标有注释的地方，就是没有注释便难以理解的地方。不得不添加注释，并不是件好事。注释毕竟不是程序，不但麻烦而且难以信赖。
- “虽然修改了程序，但是没有修改注释”的情况屡见不鲜。这会导致注释中的内容根本不正确，比没有注释更可怕。注释是在没有其他更好的手段进行说明时不得已而采取的措施。
- 程序过长比较难懂时，才必须写下类似于“这里的程序是这种作用”的注释。在这种情况下，我们可以通过使用局部程序，做到即使不使用注释也能让程序更易懂。
- 如果你必须要在某处写下注释，就可能暗示着那里存在问题。



## 调试

- 在修复程序时，需要考虑“怎样的错误才会导致这样的结果”。
- **纠错方法**：其一，是观察现有代码，找出错误。其二，是通过观察程序的运行结果，推理出程序的问题所在。
- 出现错误时，我们不能单单询问“该怎么办才能改好”，因为这样做事情不会有任何进展。此时可以导入“从结果出发即兴思考”的方法，将问题变形为“要出现这样的结果，程序必须存在怎样的错误”后，才算是朝着问题的解决前进了一步。
- **学习调试**可能会带来挫折感，但它是一个有价值的技能，并在编程以外还有很多用途。
- 一个程序中可能出现**3 种类型的错误**：语法错误、运行时错误和语义错误。
- 每当你试验新的语言特性时，应当试着故意犯错。这种试验会帮你记住所读的内容，也能帮你学会调试，因为这样能看到不同的出错消息代表着什么。现在故意犯错总比今后在编程中意外出错好。

## 变量

- 变量是计算机程序中用来保存东西的一种方式。
- 变量指一个存储信息的地方，就像贴在东西上的标签。
- 变量就是一种给事物加标签的方法，从而让我们以后可以使用它们。
- 变量就是一个可以存储常量的**存储器**。一个变量就像是计算机内存中的信息**标签**。
- 选择什么样的名字取决于你需要让这个变量名有多么大的含义。
- 作用域是指一个变量的可见范围。变量的作用域控制它在函数内外的可见性。



## 条件与循环

- 条件就是用来做比较的程序语句。
- “有条件运行”说到底不过是“最多只能循环 1 次的循环”。
- 有条件运行本身不过是循环的一种，但实际进行编程时，循环和有条件运行则会被视为不同的东西。
- for 循环是针对指定长度的循环，而 while 循环则用于你事先不知道何时停止循环的情况。
- 一般而言，缩短程序就意味着需要增加同一行的重复运行次数。能用循环的时候就用循环，减少行数，增加同一行的运行次数。
- 在使用循环时，如何平衡“简单却冗长”与“复杂却简短”这两者的关系，是个永恒的话题。这里没有正确答案，需要根据具体情况进行分析，找出最有利的方案。





## 函数

- 函数就是让 Perl/Python 做某些事情的一段代码。它们是**重用代码**的一种方式。
- 创建函数来重用代码，使用类和对象把代码划分成小块。函数还可以按模块的方式组织起来。
- 函数就是一个给出了调用接口的自包含模块。函数名其实就是函数的标识。
- 某些功能用函数实现可以有以下好处：
  - 可以使程序逻辑清晰，代码可读性好
  - 用一个可读性好的名字给函数命名，可以帮助理解程序代码的含义



## 模块

- 在 Perl/Python 中，模块是给别的程序提供有用的代码的一种方式。
- Perl/Python 的模块就是一些函数、类和变量的组合。模块用来把函数、变量，以及其他东西分组，组织成更大的、更强的程序。
- 模块要先被引用/载入然后才能使用。
- 使用扩展模块通常采用**示例驱动方法**，即参照扩展模块使用说明文档中的示例，通过简单改造示例的方法即实现我们自己期望的功能。



## 引用与子程序

- 引用就是一个指向某个数据结构（可以是一个简单变量、一个数组或一个哈希等）的标量值，通过引用可以间接访问原始数据。一个引用中所存储的是其引用对象的地址。
- 通过使用引用，不但可以缩减代码，还可以方便理解程序。
- 使用局部程序可以缩短程序。有时使用局部程序虽然不能缩短程序，但却能让程序更易懂。
- 当你做出新的局部程序时，建议你使用类似的小程序来检查代码是否正确。需要寻找错误时，当然是程序越小越方便。
- 局部程序是编写大型程序时的利器。



## 面向对象编程

- **对象是程序组织代码的方法**，它把复杂的想法拆分开来使其更容易被理解。**类是一组函数和变量**。对象是由“类”定义的，可以把“类”当成一种把对象分组归类的方法，可以把一个对象想象成是一个类家族中的一员。
- 类和对象是给函数分组的好方法，它们帮助我们程序分成小段来分别思考。
- “特征”就是一个类家族中的所有成员（还有它的子类）共同的特点。
- 有些类中的函数什么也不做，这其实在编程中很常见。某种意义上讲，这是一种共识，确保一个类的所有子类都提供同样的功能，尽管有时子类中的函数什么也不做。
- 不必强迫自己去理解面向对象。思考方式到头来不过是一种工具罢了。面向对象是工具，面向对象的程序设计语言也是工具。最重要的是“面向对象能为我们做什么”，而不是“面向对象是什么”。

## 其他

- 在计算机显示器上看到的所有东西都是由**像素**组成的，它们是很小的、方块的点。一个像素就是屏幕上的一个点，也就是可以表现出的最小元素。
- 所谓**比较字符的大小**，其实就比较它们在计算机内部存储编码的大小。字母是按照字母表顺序排列的，并且小写字母排列在大写字母的后面。
- 先画一张图，再画一个稍稍有点变化的图，这就让人感觉它在移动。这就是**动画**。
- 实际上，你计算机上的所有东西都是以文件的形式保存。
- 通过参数化运行方式，程序处理数据的方式变得更加灵活，做到了代码与具体文件名无关，代码运行的方式通过参数开关进行控制。
- 在计算个数时从 1 开始比较好，既自然又方便。除了计算个数以外，都是从 0 开始比较方便。

# 教学提纲

- 1 引言
- 2 学习方法
- 3 编写程序
- 4 编程策略
- 5 编程过程
- 6 编程真言
- 7 回顾与总结
  - 总结
  - 思考题



# 教学提纲

- 1 引言
- 2 学习方法
- 3 编写程序
- 4 编程策略

- 5 编程过程
- 6 编程真言
- 7 回顾与总结
  - 总结
  - 思考题



## 知识点

- 学习编程的方法：培训班、读书、看手册、拜师、研究程序、……
- 编程的基本流程：编辑-运行-修正
- 版本控制：Git, GitHub
- 调试程序：调试器、print、注释、模块、……
- 编程策略：找现成的程序、自己编写程序、请别人帮忙、……
- 编程的基本步骤：构思（输入、算法、输出）、编程
- 编程前：伪代码；编程中：注释

## 技能

- 熟练应用编程的基本策略、步骤和流程
- 能够使用 Git 和 GitHub 进行版本控制
- 能够用不同的方法调试程序



## 知识点

- 学习编程的方法：培训班、读书、看手册、拜师、研究程序、……
- 编程的基本流程：编辑-运行-修正
- 版本控制：Git, GitHub
- 调试程序：调试器、print、注释、模块、……
- 编程策略：找现成的程序、自己编写程序、请别人帮忙、……
- 编程的基本步骤：构思（输入、算法、输出）、编程
- 编程前：伪代码；编程中：注释

## 技能

- 熟练应用编程的基本策略、步骤和流程
- 能够使用 Git 和 GitHub 进行版本控制
- 能够用不同的方法调试程序

# 教学提纲

- 1 引言
- 2 学习方法
- 3 编写程序
- 4 编程策略

- 5 编程过程
- 6 编程真言
- 7 回顾与总结
  - 总结
  - 思考题



- 1 总结学习编程语言的方法。
- 2 编写程序的基本流程是什么？
- 3 如何使用 Git 进行版本控制？
- 4 总结调试程序的方法。
- 5 总结常用的编程策略。
- 6 编程的基本步骤是什么？需要构思哪些内容？
- 7 使用伪代码和注释有哪些优势？



# 下节预告

## 编程相关

回顾 shell/Perl 中的以下知识点：

- 变量
- 数组

## 生物学相关

回顾生物学中的以下知识点：

- DNA 的组成
- DNA 的转录
- DNA 的反向互补
- 蛋白质的组成





TEX

LATEX

X<sub>Y</sub>TEX

Beamer

