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

第八章　遗传密码

伊现富（Yi Xianfu）

天津医科大学（TIJMU）
生物医学工程与技术学院

2015 年 12 月

教学提纲

1. 引言
2. 散列
3. 数据结构和算法
4. 遗传密码
   - 密码子翻译成氨基酸
   - 遗传密码的冗余性
   - 使用散列表示遗传密码
5. DNA 翻译成蛋白质
6. 读取 FASTA 格式的 DNA
   - 序列格式
   - 读取 FASTA 文件的思路
   - 读取 FASTA 文件的子程序
   - 输出格式化的序列数据
   - 读取 FASTA 文件的主程序
   - DNA 翻译成蛋白质的主程序
7. 阅读框
   - 简介
   - 翻译阅读框
8. 回顾和总结
   - 总结
   - 思考题
9. 知识拓展
   - Perl 里面的复杂数据结构
   - 数据共享
   - 宽格式和长格式的数据
   - 数据处理实例

教学提纲

引言

已经学习

- 查找基序
- 模拟 DNA 突变
- 生成随机序列
- DNA 转录成 RNA

即将学习

- Perl 中的散列
- DNA 翻译成蛋白质
- 处理 FASTA 文件

引言

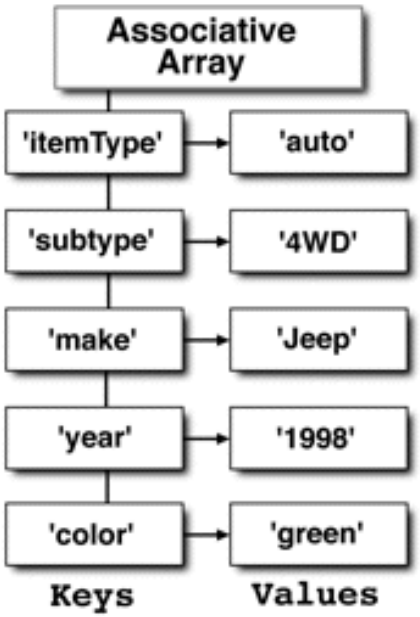
已经学习

- 查找基序
- 模拟 DNA 突变
- 生成随机序列
- DNA 转录成 RNA

即将学习

- Perl 中的散列
- DNA 翻译成蛋白质
- 处理 FASTA 文件

教学提纲



%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;

```
1  %classification = (
2     'dog',       'mammal',
3     'robin',     'bird',
4     'asp',       'reptile',
5  );
6
7  # 更加清晰易懂
8  %classification = (
9     'dog'    => 'mammal',
10    'robin'  => 'bird',
11    'asp'    => 'reptile',
12 );
```

- 胖箭头（=>）：对 Perl 而言，只是逗号的另一种写法，因此常被称为胖逗号
- 在任何需要逗号（,）的地方都可以用胖箭头（=>）代替，对 Perl 而言没有区别
- 胖箭头左边的任何裸字（一连串的字母、数字和下划线，但不得以数字开头）都会自动加上引号，因此胖箭头左边的裸字不需要加引号
- 在作为散列键使用时，如果花括号内只有裸字时，两边的引号也可以省略
- 列表末尾有一个额外的逗号，这种写法无伤大雅，而且便于维护（添加/删除）

```
1 # 给键赋值
2 $english_dictonary{'recreant'} = "One who
    calls out in surrender.";
3 # 散列（hash）：%english_dictonary
4 # 键（key）：标量，'recreant'
5 # 值（value）：标量，"One who calls out in
    surrender."
6
7 # 提取键对应的值
8 $english_dictonary{'recreant'};
9 $definition = $english_dictonary{'recreant'};
```

```
1  # 初始化散列
2  %hash = (a=>1, b=>2, c=>3);
3
4  # 获取键列表（所有的键）
5  @keys = keys %hash;
6  # 获取值列表（所有的值）
7  @values = values %hash;
8  # @keys和@values各自的顺序不定，但两者总是一一对应
9
10 # 获取散列元素（键值对）的个数
11 $count = keys %hash;
12 $count = values %hash;
13 # 列表上下文
```

```perl
# 逐项处理散列中的每一个元素
while ( ($key, $value) = each %hash ) {
  print "$key => $value\n";
}
# each返回的键值对的顺序是乱的，但它与keys和values
  返回的顺序相同，即散列的自然顺序

foreach my $key (sort keys %hash){
  my $value = $hash{$key};
  print "$key => $value\n";
  # print "$key => $hash{$key}\n";
}
```

```
1  # 真
2  if ($books{$someone}) {
3    print "$someone has at least one book
     checked out.\n";
4  }
5
6  # 假
7  # 现在没有借阅图书
8  $books{"barney"} = 0
9  # 这是张新办的借书证，从未借阅过图书
10 $books{"pebbles"} = undef;
```

```
1 # 删除指定的键及其相对应的值
2 delete $books{$person};
3 # 撤消$person的借书证
4 # delete之后，键不会出现在散列之中
5
6 $books{$person} = undef;
7 # 键依然存在于散列之中
```

```
1 %hash = (a=>1, b=>2, c=>3);
2
3 # 只适用于散列值不重复的情况，否则会导致重复的键
4 # “后发先至”：最后的键会覆盖之前的键
5 %inverse_hash = reverse %hash;
6 # %inverse_hash = (1=>a, 2=>b, 3=>c);
```

```perl
# 可以将单一散列元素内插到双引号包裹起来的字符串中
foreach $person (sort keys %books) {
  if ($books{$person}) {
    print "$person has $books{$person} items
  .\n";
  }
}

# 注意：不支持内插整个散列（"%books"）
print "%books";
# 包含6个字符的字符串：%books
```

defined

- 判断某个字符串是 undef 而不是空字符串
- 如果是 undef，返回假；否则返回真

exists

- 检查散列中是否存在某个键，和键对应的值无关
- 如果键存在，返回真；否则返回假

defined

- 判断某个字符串是 undef 而不是空字符串
- 如果是 undef，返回假；否则返回真

exists

- 检查散列中是否存在某个键，和键对应的值无关
- 如果键存在，返回真；否则返回假

```perl
#!/usr/bin/perl

use strict;
use warnings;
use utf8;

my %hash = (
    a => 1,
    b => 0,
    c => "0",
    d => "undef",
    e => undef,
);
```

```
1  if ( defined $hash{a} ) {
2      print "defined: $hash{a}(number) is TURE!\n";
3  }
4  if ( defined $hash{b} ) {
5      print "defined: $hash{b}(number) is TURE!\n";
6  }
7  if ( defined $hash{c} ) {
8      print "defined: $hash{c}(string) is TURE!\n";
9  }
10 if ( defined $hash{d} ) {
11     print "defined: $hash{d}(string) is TURE!\n";
12 }
13 unless ( defined $hash{e} ) {
14     print "defined: undef is FALSE!\n";
15 }
16 unless ( defined $hash{x} ) {
17     print "defined: key ?x? is FALSE!\n";
18 }
```

```
1  if ( exists $hash{a} ) {
2      print "exists: key a is TURE!\n";
3  }
4  if ( exists $hash{b} ) {
5      print "exists: key b is TURE!\n";
6  }
7  if ( exists $hash{c} ) {
8      print "exists: key c is TURE!\n";
9  }
10 if ( exists $hash{d} ) {
11     print "exists: key d is TURE!\n";
12 }
13 if ( exists $hash{e} ) {
14     print "exists: key e is TURE!\n";
15 }
16 unless ( exists $hash{x} ) {
17     print "exists: key ?x? is FALSE!\n";
18 }
```

```
1  if ( $hash{a} ) {
2      print "value: $hash{a}(number) is TURE!\n";
3  }
4  unless ( $hash{b} ) {
5      print "value: $hash{b}(number) is FALSE!\n";
6  }
7  unless ( $hash{c} ) {
8      print "value: $hash{c}(string) is FALSE!\n";
9  }
10 if ( $hash{d} ) {
11     print "value: $hash{d}(string) is TURE!\n";
12 }
13 unless ( $hash{e} ) {
14     print "value: undef is FALSE!\n";
15 }
16 unless ( $hash{x} ) {
17     print "value: key ?x? is FALSE!\n";
18 }
```

三大类

- 标量变量（scalar variable）：`$scalar`
- 数组（array）：`@array`
- 散列（hash, associative array）：`%hash`

数组 vs. 散列

- 数组元素：`$array[0]`
- 散列查找：`$hash{'key'}`

三大类

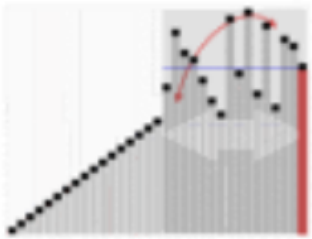
- 标量变量（scalar variable）：`$scalar`
- 数组（array）：`@array`
- 散列（hash, associative array）：`%hash`

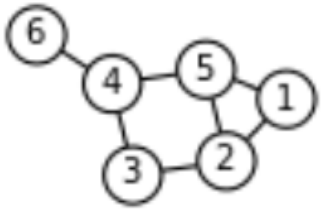
数组 vs. 散列

- 数组元素：`$array[0]`
- 散列查找：`$hash{'key'}`

教学提纲

$O(n^2)$





算法分析　　算法　　数据结构　　计算几何　　图论

算法

算法（algorithm）指定义良好的计算过程，它取一个或一组值作为输入，经过一系列定义好的计算过程，得到一个或一组输出。

在数学和计算机科学/算学之中，算法/演算法/算则法（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。

不同的算法通常需要不同的数据结构。

特征

高德纳在他的著作《计算机程序设计艺术》里对算法的特征归纳：

- 输入：一个算法必须有零个或以上输入量。
- 输出：一个算法应有一个或以上输出量，输出量是算法计算的结果。
- 明确性：算法的描述必须无歧义，以保证算法的实际执行结果是精确地匹配要求或期望，通常要求实际运行结果是确定的。
- 有限性：依据图灵的定义，一个算法是能够被任何图灵完备系统模拟的一串运算，而图灵机只有有限个状态、有限个输入符号和有限个转移函数（指令）。而一些定义更规定算法必须在有限个步骤内完成任务。
- 有效性：又称可行性。能够实现，算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现。

基本要素

算法的核心是创建问题抽象的模型和明确求解目标，之后可以根据具体的问题选择不同的模式和方法完成算法的设计。

形式化算法

算法是计算机处理信息的本质，因为计算机程序本质上是一个算法来告诉计算机确切的步骤来执行一个指定的任务。一般地，当算法在处理信息时，会从输入设备或数据的存储地址读取数据，把结果写入输出设备或某个存储地址供以后再调用。

常用实现方法

- 递归方法与迭代方法
- 顺序计算、并行计算和分布式计算
- 确定性算法和非确定性算法
- 精确求解和近似求解

基本要素

算法的核心是创建问题抽象的模型和明确求解目标，之后可以根据具体的问题选择不同的模式和方法完成算法的设计。

形式化算法

算法是计算机处理信息的本质，因为计算机程序本质上是一个算法来告诉计算机确切的步骤来执行一个指定的任务。一般地，当算法在处理信息时，会从输入设备或数据的存储地址读取数据，把结果写入输出设备或某个存储地址供以后再调用。

常用实现方法

- 递归方法与迭代方法
- 顺序计算、并行计算和分布式计算
- 确定性算法和非确定性算法
- 精确求解和近似求解

基本要素

算法的核心是创建问题抽象的模型和明确求解目标，之后可以根据具体的问题选择不同的模式和方法完成算法的设计。

形式化算法

算法是计算机处理信息的本质，因为计算机程序本质上是一个算法来告诉计算机确切的步骤来执行一个指定的任务。一般地，当算法在处理信息时，会从输入设备或数据的存储地址读取数据，把结果写入输出设备或某个存储地址供以后再调用。

常用实现方法

- 递归方法与迭代方法
- 顺序计算、并行计算和分布式计算
- 确定性算法和非确定性算法
- 精确求解和近似求解

常用设计模式

- 完全遍历法和不完全遍历法：把解空间的所有元素完全遍历一遍，逐个检测元素是否是我们要的解。
- 分治法：把一个问题分区成互相独立的多个部分分别求解的思路，便于进行并行计算。
- 动态规划法：当问题的整体最优解就是由局部最优解组成的时候，经常采用的一种方法。
- 贪婪算法：常见的近似求解思路。当问题的整体最优解不是（或无法证明是）由局部最优解组成，且对解的最优性没有要求的时候，可以采用的一种方法。
- 线性规划法：线性规划是最优化问题中的一个重要领域。很多最优化问题算法都可以分解为线性规划子问题，然后逐一求解。
- 简并法：把一个问题通过逻辑或数学推理，简化成与之等价或者近似的、相对简单的模型，进而求解的方法。

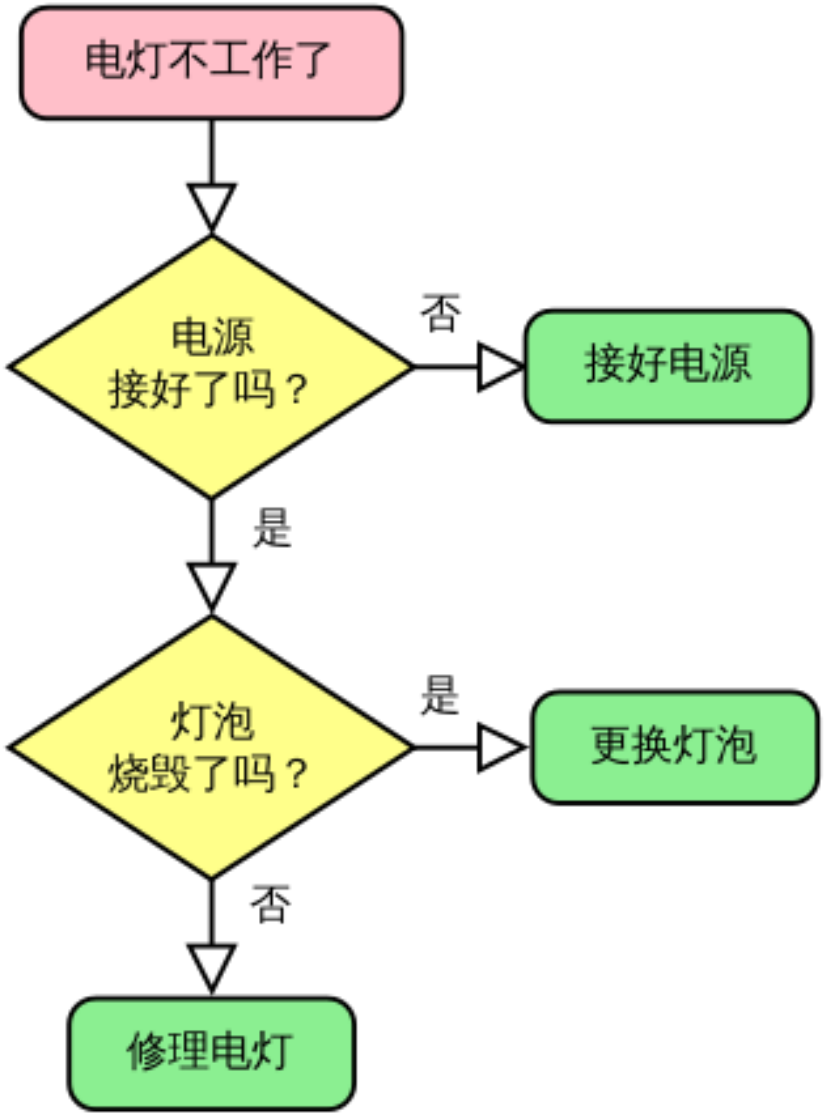
数据结构

数据结构（data structure）是计算机中存储、组织数据的方式。

数据结构意味着接口或封装：一个数据结构可被视为两个函数之间的接口，或者是由数据类型联合组成的存储内容的访问方法封装。

数据结构可透过程序语言所提供的数据类型、引用及其他操作加以实现。一个设计良好的数据结构，应该在尽可能使用较少的时间与空间资源的前提下，支持各种程序运行。

不同种类的数据结构适合不同种类的应用，部分数据结构甚至是为了解决特定问题而设计出来的。正确的数据结构选择可以提高算法的效率。在计算机程序设计的过程里，选择适当的数据结构是一项重要工作。许多大型系统的编写经验显示，程序设计的困难程度与最终成果的质量与表现，取决于是否选择了最适合的数据结构。



常见数据结构

- 数组（Array）
- 堆栈（Stack）
- 队列（Queue）
- 链表（Linked List）
- 树（Tree）
- 图（Graph）
- 堆（Heap）
- 散列表（Hash）

生物学问题

特定环境条件下、特性类型细胞中，人类基因组中的每一个基因是否表达？

实验方法与数据处理

- 基因芯片、RNA 测序……
- 芯片：探针 ⇒ 基因；亮度 ⇒ 表达值
- 测序：reads ⇒ 基因；reads 数 ⇒ 表达值
- 其他：质量控制，标准化，基因注释，……

最终数据

- 所有基因（30000）：基因名
- 表达基因（8000）的表达水平：具体数值
- 未表达基因（22000）的表达值：0

生物学问题

特定环境条件下、特性类型细胞中，人类基因组中的每一个基因是否表达?

实验方法与数据处理

- 基因芯片、RNA 测序……
- 芯片：探针 ⇒ 基因；亮度 ⇒ 表达值
- 测序：reads ⇒ 基因；reads 数 ⇒ 表达值
- 其他：质量控制，标准化，基因注释，……

最终数据

- 所有基因（30000）：基因名
- 表达基因（8000）的表达水平：具体数值
- 未表达基因（22000）的表达值：0

生物学问题

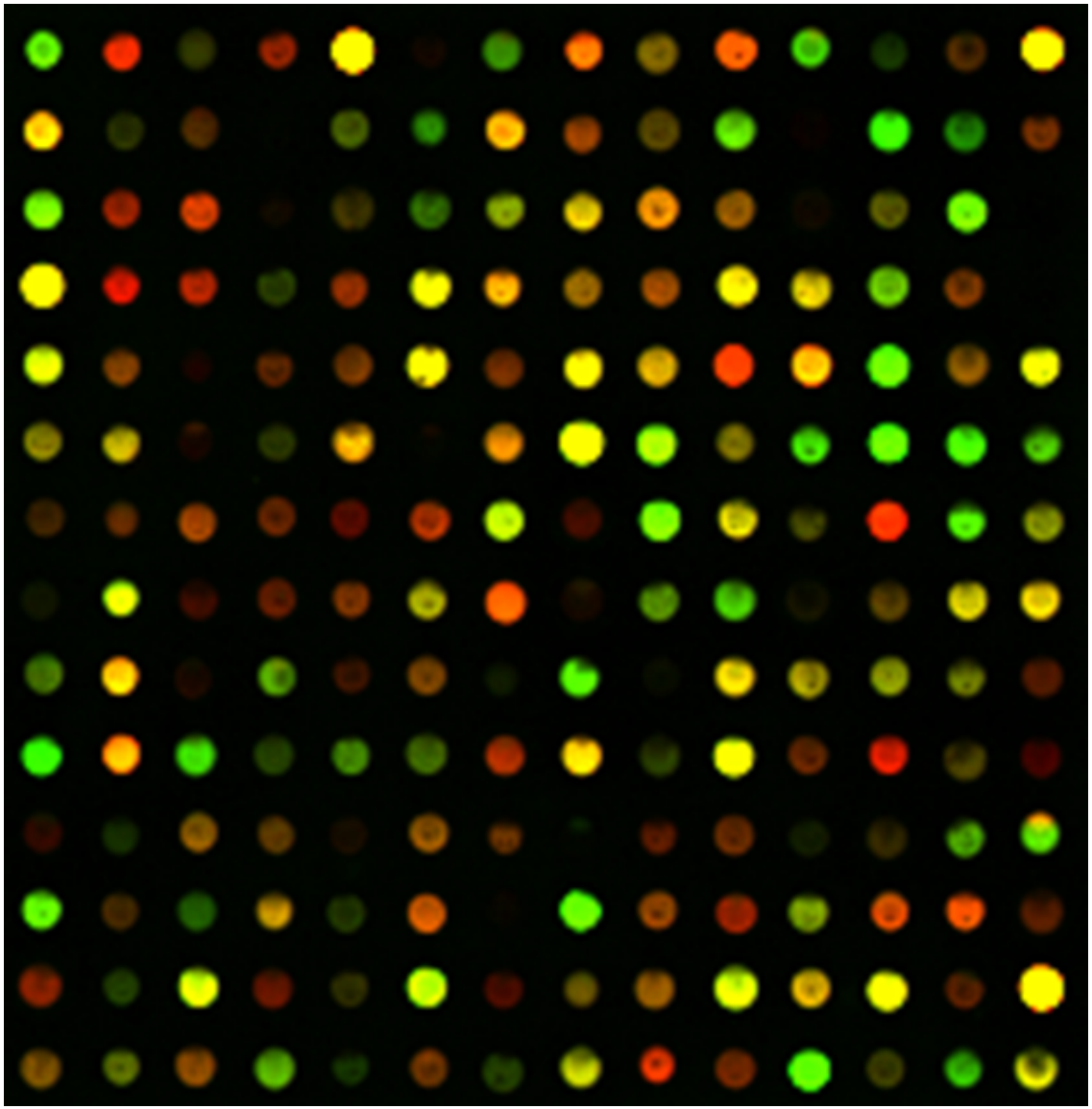
特定环境条件下、特性类型细胞中，人类基因组中的每一个基因是否表达?

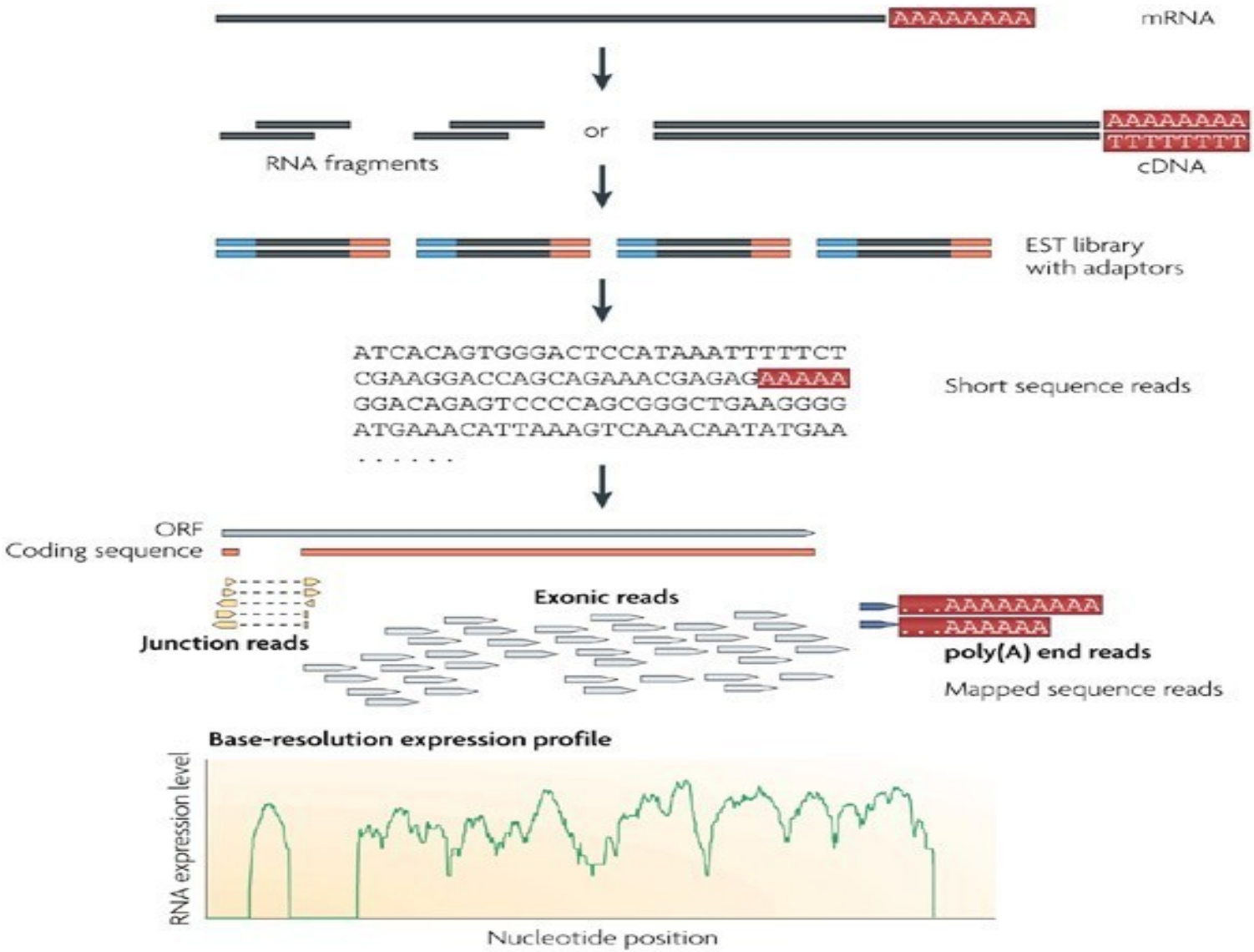
实验方法与数据处理

- 基因芯片、RNA 测序……
- 芯片：探针 ⇒ 基因；亮度 ⇒ 表达值
- 测序：reads ⇒ 基因；reads 数 ⇒ 表达值
- 其他：质量控制，标准化，基因注释，……

最终数据

- 所有基因（30000）：基因名
- 表达基因（8000）的表达水平：具体数值
- 未表达基因（22000）的表达值：0




AAAAAAAA
mRNA
or
RNA fragments
AAAAAAAA
TTTTTTTT
cDNA
EST library
with adaptors
ATCACAGTGGGACTCCATAAATTTTTCT
CGAAGGACCAGCAGAAACGAGAGAAAAA
GGACAGAGTCCCCAGCGGGCTGAAGGGG
ATGAAACATTAAAGTCAAACAATATGAA
Short sequence reads
ORF
Coding sequence
Exonic reads
Junction reads
...AAAAAAAAA
...AAAAAA
poly(A) end reads
Mapped sequence reads
Base-resolution expression profile
RNA expression level
Nucleotide position

具体实现

只在数组中存储表达基因（8000）的基因名，丢弃未表达基因的基因名

可能的问题

- 查找表达基因：平均比较 4000 次
- 查找未表达基因：必须比较 8000 次
- 查找未研究的基因：未表达？未研究？

具体实现

只在数组中存储表达基因（8000）的基因名，丢弃未表达基因的基因名

可能的问题

- 查找表达基因：平均比较 4000 次
- 查找未表达基因：必须比较 8000 次
- 查找未研究的基因：未表达？未研究？

改进方法

- 把 30000 个基因全部存储到数组中
- 把表达值附加在基因名的后面

存在的问题

- 每次查询：平均比较 15000 次
- 分割基因名和表达值

改进方法

- 把 30000 个基因全部存储到数组中
- 把表达值附加在基因名的后面

存在的问题

- 每次查询：平均比较 15000 次
- 分割基因名和表达值

具体实现

- 把所有基因名按照字母顺序排序后存储到数组中
- 使用折半查找的算法进行查找

优缺点

- 优点：只需大约 15 次循环比较即可
- 缺点：需要首先对列表进行排序
- 缺点：添加新元素时稍显繁琐

具体实现

- 把所有基因名按照字母顺序排序后存储到数组中
- 使用折半查找的算法进行查找

优缺点

- 优点：只需大约 15 次循环比较即可
- 缺点：需要首先对列表进行排序
- 缺点：添加新元素时稍显繁琐

具体实现

- 基因名 ⇒ 散列的键
- 表达值 ⇒ 散列的值

优缺点

- 优点：查找速度通常比折半查找快
- 优点：可以明确知道要查找的基因是否在数据集中存在
- 优点：添加或删除元素时非常方便
- 缺点：其中的元素没有经过排序

具体实现

- 基因名 ⇒ 散列的键
- 表达值 ⇒ 散列的值

优缺点

- 优点：查找速度通常比折半查找快
- 优点：可以明确知道要查找的基因是否在数据集中存在
- 优点：添加或删除元素时非常方便
- 缺点：其中的元素没有经过排序

未排序的数组

- 需要遍历数组，速度很慢
- 必须要把基因名和表达值分割开来

排序数组和折半查找

- 查找速度大幅提升
- 必须对列表进行排序
- 不停的插入元素、重新排序会非常慢

散列

- 键：基因名；值：表达值
- 相比折半查找有优势：速度快，明确是否被定义，添加或删除元素不需重排序
- 缺点：其中元素没有被排序

未排序的数组

- 需要遍历数组，速度很慢
- 必须要把基因名和表达值分割开来

排序数组和折半查找

- 查找速度大幅提升
- 必须对列表进行排序
- 不停的插入元素、重新排序会非常慢

散列

- 键：基因名；值：表达值
- 相比折半查找有优势：速度快，明确是否被定义，添加或删除元素不需重排序
- 缺点：其中元素没有被排序

未排序的数组

- 需要遍历数组，速度很慢
- 必须要把基因名和表达值分割开来

排序数组和折半查找

- 查找速度大幅提升
- 必须对列表进行排序
- 不停的插入元素、重新排序会非常慢

散列

- 键：基因名；值：表达值
- 相比折半查找有优势：速度快，明确是否被定义，添加或删除元素不需重排序
- 缺点：其中元素没有被排序

需求

- 看看某个元素是否在数据集中，元素的顺序无所谓
- 数据集需要排序、相对快速的查找，不需要频繁添加或删减元素
- 不需要对元素排序，但需要快速找到最新添加的元素
- 不需要对元素排序，但需要添加元素、移除“最老”的元素

数据结构

- 散列
- 排序数组结合折半查找
- 数组结合 push 和 pop 函数
- 数组结合 push 和 shift 函数

需求

- 看看某个元素是否在数据集中，元素的顺序无所谓
- 数据集需要排序、相对快速的查找，不需要频繁添加或删减元素
- 不需要对元素排序，但需要快速找到最新添加的元素
- 不需要对元素排序，但需要添加元素、移除“最老”的元素

数据结构

- 散列
- 排序数组结合折半查找
- 数组结合 push 和 pop 函数
- 数组结合 push 和 shift 函数

需求

- 看看某个元素是否在数据集中，元素的顺序无所谓
- 数据集需要排序、相对快速的查找，不需要频繁添加或删减元素
- 不需要对元素排序，但需要快速找到最新添加的元素
- 不需要对元素排序，但需要添加元素、移除“最老”的元素

数据结构

- 散列
- 排序数组结合折半查找
- 数组结合 push 和 pop 函数
- 数组结合 push 和 shift 函数

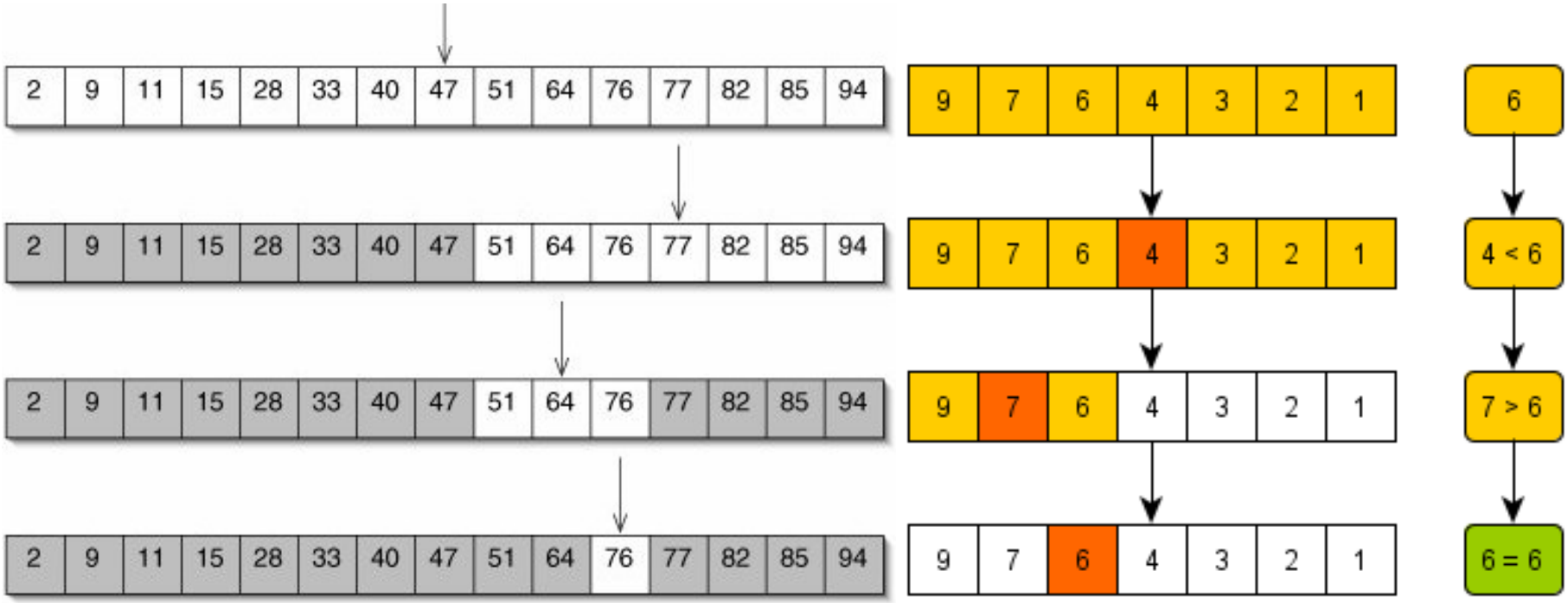
需求

- 看看某个元素是否在数据集中，元素的顺序无所谓
- 数据集需要排序、相对快速的查找，不需要频繁添加或删减元素
- 不需要对元素排序，但需要快速找到最新添加的元素
- 不需要对元素排序，但需要添加元素、移除“最老”的元素

数据结构

- 散列
- 排序数组结合折半查找
- 数组结合 push 和 pop 函数
- 数组结合 push 和 shift 函数

```
1  # 折半查找（binary search）的伪代码
2  Given a sorted array, and an element:
3
4  Until you find the element or discover it's not
     there,
5
6    Pick the midpoint of the array, $array[scalar(
     @array)/2]
7
8    Compare your element with the element at the
     midpoint
9
10   If that matches your element, you're done.
11
12   Else, ignore the half of the array that your
     element is not in
13 }
```


2 9 11 15 28 33 40 47 51 64 76 77 82 85 94
2 9 11 15 28 33 40 47 51 64 76 77 82 85 94
2 9 11 15 28 33 40 47 51 64 76 77 82 85 94
2 9 11 15 28 33 40 47 51 64 76 77 82 85 94
9 7 6 4 3 2 1
9 7 6 4 3 2 1
9 7 6 4 3 2 1
9 7 6 4 3 2 1
6
4 < 6
7 > 6
6 = 6

```perl
#!/usr/bin/perl

use strict;
use warnings;

my @numbers = map { $_ * 3 } ( 0 .. 1000000 );

sub search {
    my ( $numbers, $target ) = @_;
    for my $i ( 0 .. $#$numbers ) {
        return $i if $numbers->[$i] == $target;
    }
    return;
}

print search( \@numbers, 699 ), "\n";
print search( \@numbers, 28 ),  "\n";
```

```perl
#!/usr/bin/perl
use strict; use warnings;

my @numbers = map { $_ * 3 } ( 0 .. 1000000 );
sub search {
    my ( $numbers, $target ) = @_;
    return binary_search( $numbers, $target, 0, $#$numbers );
}
sub binary_search {
    my ( $numbers, $target, $low, $high ) = @_;
    return if $high < $low;
    my $middle = int( ( $low + $high ) / 2 );
    if ( $numbers->[$middle] > $target ) {
        return binary_search($numbers, $target, $low, $middle-1);
    }
    elsif ( $numbers->[$middle] < $target ) {
        return binary_search($numbers, $target, $middle+1, $high);
    }
    return $middle;
}
print search( \@numbers, 699 ), "\n";
print search( \@numbers, 28 ),  "\n";
```

```perl
# 两个字符串一样，返回0
'ZZZ' cmp 'ZZZ'

# 两个字符串按字母顺序排列，返回-1
'AAA' cmp 'ZZZ'

# 两个字符串按字母顺序的逆序排列，返回1
'ZZZ' cmp 'AAA'

# 按照字母顺序/逆序对由字符串构成的数组进行排序
@array = sort @array;
@array = sort { $a cmp $b } @array;
@array = sort { $b cmp $a } @array;

# 以升序/降序对由数字构成的数组进行排序:
@array = sort { $a <=> $b } @array;
@array = sort { $b <=> $a } @array;
```

```perl
# 查看某个散列值是否被定义
if ( defined $myhash{'mykey'} ) { ... }

# 对散列的键进行排序
@sorted_keys = sort keys %my_hash;

# 对散列的值进行排序
@sorted_values = sort values %my_hash;
```

散列

- Perl 的内置数据类型，易使用、易编程
- 对于程序员来说，懒惰的方法通常是最有效的方法（节省编程的时间通常要比节省程序运行的时间更加重要一些）

```perl
# 查看某个散列值是否被定义
if ( defined $myhash{'mykey'} ) { ... }

# 对散列的键进行排序
@sorted_keys = sort keys %my_hash;

# 对散列的值进行排序
@sorted_values = sort values %my_hash;
```

散列

- Perl 的内置数据类型，易使用、易编程
- 对于程序员来说，懒惰的方法通常是最有效的方法（节省编程的时间通常要比节省程序运行的时间更加重要一些）

数据库

数据库，简单来说可视为电子化的文件柜——存储电子文件的处所，用户可以对文件中的数据运行新增、截取、更新、删除等操作。适用于存储和访问海量数据。

数据库指的是以一定方式储存在一起、能为多个用户共享、具有尽可能小的冗余度、与应用程序彼此独立的数据集合。

数据库管理系统

数据库管理系统（Database Management System，DBMS）是为管理数据库而设计的电脑软件系统，一般具有存储、截取、安全保障、备份等基础功能。

数据库管理系统可以依据它所支持的数据库模型来作分类（关系式、XML），或依据所支持的电脑类型来作分类（服务器群集、移动电话），或依据所用查询语言来作分类（SQL、XQuery），或依据性能冲量重点来作分类（最大规模、最高运行速度），亦或其他的分类方式。不论使用哪种分类方式，一些 DBMS 能够跨类，例如，同时支持多种查询语言。

数据库

数据库，简单来说可视为电子化的文件柜——存储电子文件的处所，用户可以对文件中的数据运行新增、截取、更新、删除等操作。适用于存储和访问海量数据。

数据库指的是以一定方式储存在一起、能为多个用户共享、具有尽可能小的冗余度、与应用程序彼此独立的数据集合。

数据库管理系统

数据库管理系统（Database Management System，DBMS）是为管理数据库而设计的电脑软件系统，一般具有存储、截取、安全保障、备份等基础功能。

数据库管理系统可以依据它所支持的数据库模型来作分类（关系式、XML），或依据所支持的电脑类型来作分类（服务器群集、移动电话），或依据所用查询语言来作分类（SQL、XQuery），或依据性能冲量重点来作分类（最大规模、最高运行速度），亦或其他的分类方式。不论使用哪种分类方式，一些 DBMS 能够跨类，例如，同时支持多种查询语言。

关系数据库（relational database），是建立在关系模型基础上的数据库，借助于集合代数等数学概念和方法来处理数据库中的数据。关系数据库是存储和提取海量数据最流行的方法。

关系数据库把数据组织成表格进行存储。表是以行和列的形式组织起来的数据的集合。一个数据库包括一个或多个表。例如，可能有一个有关作者信息的名为 authors 的表。每列都包含特定类型的信息，如作者的姓氏。每行都包含有关特定作者的所有信息：姓、名、住址等等。在关系型数据库当中一个表就是一个关系，一个关系数据库可以包含多个表。

数据通常通过一种查询语言进行输入和提取，即结构化查询语言（SQL, Structured Query Language）语言。SQL 是 1974 年由 Boyce 和 Chamberlin 提出的一种介于关系代数与关系演算之间的结构化查询语言，是一个通用的、功能极强的关系型数据库语言。

School Table

ID	Name
S001	University of Technology
S002	University of Applied Science

Student Table

School ID	ID	Name	DOB
S001	UT-1000	Tommy	05/06/1995
S001	UT-1000	Better	16/04/1995
S002	UAS-1000	Linda	02/09/1995
S002	UAS-1000	Jonathan	22/06/1995

The DBI(Perl's Database Interface) is the standard database interface module for Perl. It defines a set of methods, variables and conventions that provide a consistent database interface independent of the actual database being used.

DBM（database management）是一种文件数据储存数据，由于采用散列结构进行链接，因此具有一些数据库的特点功能。与普通文本数据库相比，具有稳定、检索速度快和支持量大的优点。由于 DBM 是从 Unix 系统中移植来的，因此在 Unix/linux 系统中优点比较明显，而在 NT 系统中则不太理想，在 NT 中使用有时会令数据文件变得十分庞大。

DBM 数据库采用散列方式保存数据, 并与散列结合使用。

DBM

Perl 中简单的、内置的一种方法，来存储散列数据

初始化

启动后，把一个散列“绑定”到计算机硬盘上的一个文件

使用

像使用散列一样使用它（查找，添加、删除……）

特色

- 简单且非常有用的数据库
- 适用于键-值数据集

DBM

Perl 中简单的、内置的一种方法，来存储散列数据

初始化

启动后，把一个散列“绑定”到计算机硬盘上的一个文件

使用

像使用散列一样使用它（查找，添加、删除……）

特色

- 简单且非常有用的数据库
- 适用于键-值数据集

DBM

Perl 中简单的、内置的一种方法，来存储散列数据

初始化

启动后，把一个散列“绑定”到计算机硬盘上的一个文件

使用

像使用散列一样使用它（查找，添加、删除……）

特色

- 简单且非常有用的数据库
- 适用于键-值数据集

DBM

Perl 中简单的、内置的一种方法，来存储散列数据

初始化

启动后，把一个散列“绑定”到计算机硬盘上的一个文件

使用

像使用散列一样使用它（查找，添加、删除……）

特色

- 简单且非常有用的数据库
- 适用于键-值数据集

Problem

You want to create, populate, inspect, or delete values in a DBM database.

Solution

Use **dbmopen** or **tie** to open the database and make it accessible through a hash. Then use the hash as you normally would. When you're done, call **dbmclose** or **untie**.

Problem

You want to create, populate, inspect, or delete values in a DBM database.

Solution

Use **dbmopen** or **tie** to open the database and make it accessible through a hash. Then use the hash as you normally would. When you're done, call **dbmclose** or **untie**.

Discussion

Accessing a database as a hash is powerful but easy, giving you a persistent hash that sticks around after the program using it has finished running. It's also much faster than loading in a new hash every time; even if the hash has a million entries, your program starts up virtually instantaneously.

The program treats the database as though it were a normal hash. You can even call **keys** or **each** on it. Likewise, **exists** and **defined** are implemented for tied DBM hashes. Unlike a normal hash, a DBM hash does not distinguish between those two functions.

```perl
use DB_File;  # optional; overrides default
dbmopen %HASH, $FILENAME, 0666  # open database, accessed through %HASH
  or die "Can't open $FILENAME:$!\n";

$V = $HASH{$KEY};  # retrieve from database
$HASH{$KEY} = $VALUE;  # put value into database
if (exists $HASH{$KEY}) {  # check whether in database
  # ...
}
delete $HASH{$KEY};  # remove from database
dbmclose %HASH;  # close the database
```

```perl
use DB_File;  # load database module

tie %HASH, "DB_File", $FILENAME  # open database, to be accessed
  or die "Can't open $FILENAME:$!\n";  # through %HASH

$V = $HASH{$KEY};  # retrieve from database
$HASH{$KEY} = $VALUE;  # put value into database
if (exists $HASH{$KEY}) {  # check whether in database
  # ...
}
delete $HASH{$KEY};  # delete from database
untie %HASH;  # close the database
```

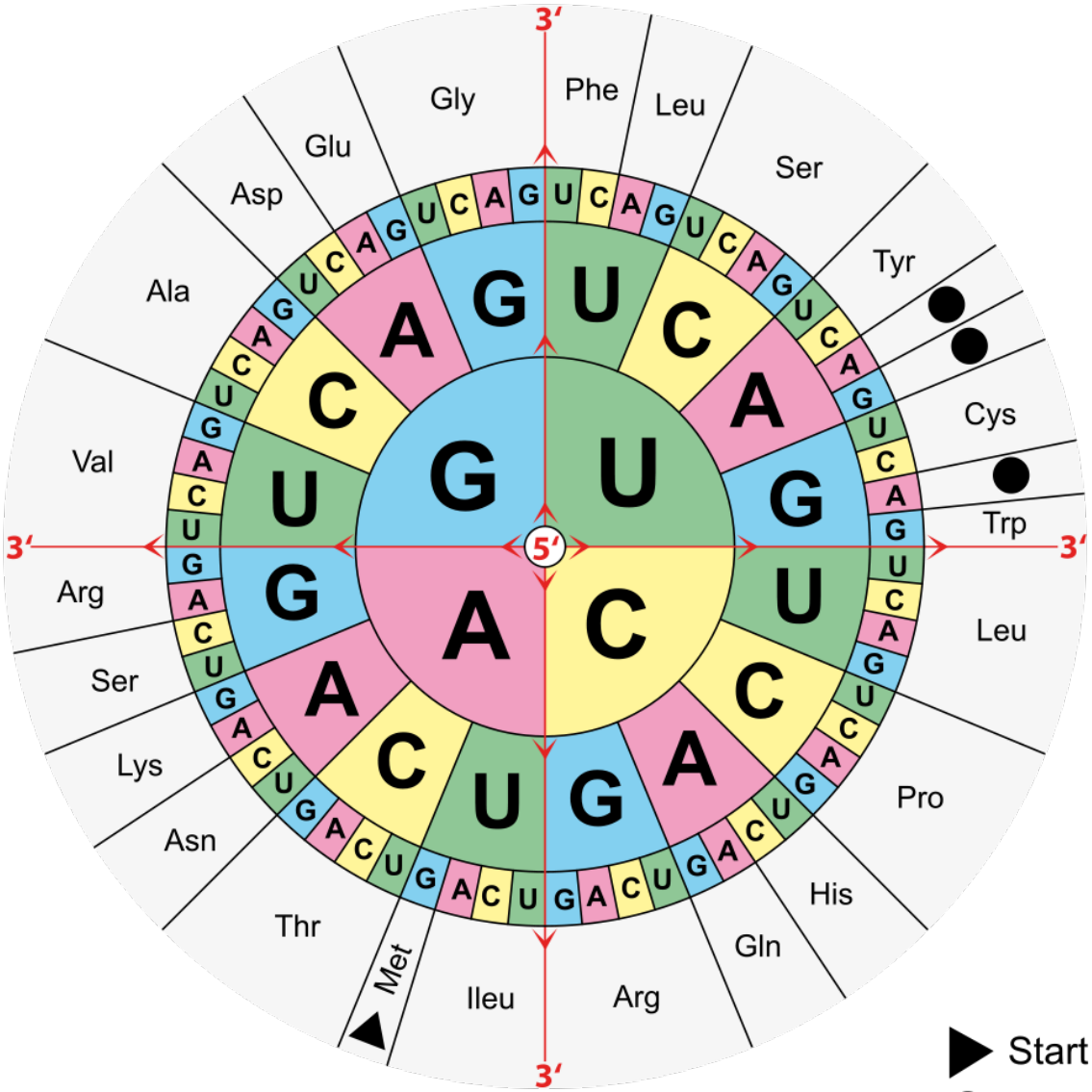
```perl
#!/usr/bin/perl

use warnings; use strict;
use DB_File;

my ( %h, $k, $v );
unlink "fruit";

tie %h, "DB_File", "fruit", O_RDWR | O_CREAT, 0666, $DB_HASH or
  die "Cannot open file 'fruit': $!\n";

# Add a few key/value pairs to the file
$h{"apple"}  = "red"; $h{"orange"} = "orange";
$h{"banana"} = "yellow"; $h{"tomato"} = "red";
# Check for existence of a key
print "Banana Exists\n\n" if $h{"banana"};
# Delete a key/value pair.
delete $h{"apple"};
# print the contents of the file
while ( ( $k, $v ) = each %h ) { print "$k -> $v\n" }

untie %h;
```

```perl
#!/usr/bin/perl
use warnings; use strict;

my %MYDB;
# 创建或打开DBM; 如果是新建DBM, 会自动生成mydb.dir和mydb.pag两个文件
dbmopen(%MYDB, "mydb", 0644)  or die "Can not access the database";

# 添加key/value对
$MYDB{name1} = "will1"; $MYDB{name2} = "will2";
# 读取内容
foreach my $key (keys %MYDB) {
  print "$key\t$MYDB{$key}\n";
}

# 关闭DBM
dbmclose(%MYDB) or die "Can not close database";
```

教学提纲

遗传密码（genetic code）是一组规则，将 DNA 或 mRNA 序列以三个核苷酸为一组的密码子（codon）翻译为蛋白质的氨基酸序列，用于蛋白质合成。

几乎所有的生物都使用同样的遗传密码，称为标准遗传密码；即使是非细胞结构的病毒，它们也是使用标准遗传密码。但是也有少数生物使用一些稍微不同的遗传密码。


3′
5′
3′
3′
Gly
Phe
Leu
Ser
Tyr
Cys
Trp
Leu
Pro
His
Gln
Arg
Ileu
Met
Thr
Asn
Lys
Ser
Arg
Val
Ala
Asp
Glu
Start
Stop

教学提纲

生物学目的

把三个核苷酸的密码子翻译成氨基酸。

程序实现

输入三字母的 DNA 密码子，返回一个单字母缩写表示的氨基酸。

生物学目的

把三个核苷酸的密码子翻译成氨基酸。

程序实现

输入三字母的 DNA 密码子，返回一个单字母缩写表示的氨基酸。

```perl
# codon2aa
#
# A subroutine to translate a DNA 3-character codon to an amino acid

sub codon2aa {
  my($codon) = @_;

     if ( $codon =~ /TCA/i ) { return 'S' } # Serine
  elsif ( $codon =~ /TCC/i ) { return 'S' } # Serine
  elsif ( $codon =~ /TCG/i ) { return 'S' } # Serine
  elsif ( $codon =~ /TCT/i ) { return 'S' } # Serine
  elsif ( $codon =~ /TTC/i ) { return 'F' } # Phenylalanine
  elsif ( $codon =~ /TTT/i ) { return 'F' } # Phenylalanine
  elsif ( $codon =~ /TTA/i ) { return 'L' } # Leucine
  elsif ( $codon =~ /TTG/i ) { return 'L' } # Leucine
```

```
16    elsif ( $codon =~ /TAC/i ) { return 'Y' } # Tyrosine
17    elsif ( $codon =~ /TAT/i ) { return 'Y' } # Tyrosine
18    elsif ( $codon =~ /TAA/i ) { return '_' } # Stop
19    elsif ( $codon =~ /TAG/i ) { return '_' } # Stop
20    elsif ( $codon =~ /TGC/i ) { return 'C' } # Cysteine
21    elsif ( $codon =~ /TGT/i ) { return 'C' } # Cysteine
22    elsif ( $codon =~ /TGA/i ) { return '_' } # Stop
23    elsif ( $codon =~ /TGG/i ) { return 'W' } # Tryptophan
24    elsif ( $codon =~ /CTA/i ) { return 'L' } # Leucine
25    elsif ( $codon =~ /CTC/i ) { return 'L' } # Leucine
26    elsif ( $codon =~ /CTG/i ) { return 'L' } # Leucine
27    elsif ( $codon =~ /CTT/i ) { return 'L' } # Leucine
28    elsif ( $codon =~ /CCA/i ) { return 'P' } # Proline
29    elsif ( $codon =~ /CCC/i ) { return 'P' } # Proline
30    elsif ( $codon =~ /CCG/i ) { return 'P' } # Proline
31    elsif ( $codon =~ /CCT/i ) { return 'P' } # Proline
32    elsif ( $codon =~ /CAC/i ) { return 'H' } # Histidine
33    elsif ( $codon =~ /CAT/i ) { return 'H' } # Histidine
```

```
34   elsif ( $codon =~ /CAA/i ) { return 'Q' } # Glutamine
35   elsif ( $codon =~ /CAG/i ) { return 'Q' } # Glutamine
36   elsif ( $codon =~ /CGA/i ) { return 'R' } # Arginine
37   elsif ( $codon =~ /CGC/i ) { return 'R' } # Arginine
38   elsif ( $codon =~ /CGG/i ) { return 'R' } # Arginine
39   elsif ( $codon =~ /CGT/i ) { return 'R' } # Arginine
40   elsif ( $codon =~ /ATA/i ) { return 'I' } # Isoleucine
41   elsif ( $codon =~ /ATC/i ) { return 'I' } # Isoleucine
42   elsif ( $codon =~ /ATT/i ) { return 'I' } # Isoleucine
43   elsif ( $codon =~ /ATG/i ) { return 'M' } # Methionine
44   elsif ( $codon =~ /ACA/i ) { return 'T' } # Threonine
45   elsif ( $codon =~ /ACC/i ) { return 'T' } # Threonine
46   elsif ( $codon =~ /ACG/i ) { return 'T' } # Threonine
47   elsif ( $codon =~ /ACT/i ) { return 'T' } # Threonine
48   elsif ( $codon =~ /AAC/i ) { return 'N' } # Asparagine
49   elsif ( $codon =~ /AAT/i ) { return 'N' } # Asparagine
50   elsif ( $codon =~ /AAA/i ) { return 'K' } # Lysine
51   elsif ( $codon =~ /AAG/i ) { return 'K' } # Lysine
```

```
52   elsif ( $codon =~ /AGC/i ) { return 'S' } # Serine
53   elsif ( $codon =~ /AGT/i ) { return 'S' } # Serine
54   elsif ( $codon =~ /AGA/i ) { return 'R' } # Arginine
55   elsif ( $codon =~ /AGG/i ) { return 'R' } # Arginine
56   elsif ( $codon =~ /GTA/i ) { return 'V' } # Valine
57   elsif ( $codon =~ /GTC/i ) { return 'V' } # Valine
58   elsif ( $codon =~ /GTG/i ) { return 'V' } # Valine
59   elsif ( $codon =~ /GTT/i ) { return 'V' } # Valine
60   elsif ( $codon =~ /GCA/i ) { return 'A' } # Alanine
61   elsif ( $codon =~ /GCC/i ) { return 'A' } # Alanine
62   elsif ( $codon =~ /GCG/i ) { return 'A' } # Alanine
63   elsif ( $codon =~ /GCT/i ) { return 'A' } # Alanine
64   elsif ( $codon =~ /GAC/i ) { return 'D' } # Aspartic
     Acid
65   elsif ( $codon =~ /GAT/i ) { return 'D' } # Aspartic
     Acid
```

```
66    elsif ( $codon =~ /GAA/i ) { return 'E' } # Glutamic
      Acid
67    elsif ( $codon =~ /GAG/i ) { return 'E' } # Glutamic
      Acid
68    elsif ( $codon =~ /GGA/i ) { return 'G' } # Glycine
69    elsif ( $codon =~ /GGC/i ) { return 'G' } # Glycine
70    elsif ( $codon =~ /GGG/i ) { return 'G' } # Glycine
71    elsif ( $codon =~ /GGT/i ) { return 'G' } # Glycine
72    else {
73      print STDERR "Bad codon \"$codon\"!!\n";
74      exit;
75    }
76  }
```

优点

- 代码清晰简单
- 排版布局美观
- 两者相得益彰，使整个翻译过程一目了然

缺点

- 大量的字符串比较
- 运行起来耗费时间

优点

- 代码清晰简单
- 排版布局美观
- 两者相得益彰，使整个翻译过程一目了然

缺点

- 大量的字符串比较
- 运行起来耗费时间

STDIN

- 标准输入（默认键盘）

STDOUT

- 标准输出（默认屏幕）
- print 默认使用 STDOUT
- print 可以使用一个文件句柄作为可选的参数

STDERR

- 标准错误输出（默认屏幕）

STDIN

- 标准输入（默认键盘）

STDOUT

- 标准输出（默认屏幕）
- print 默认使用 STDOUT
- print 可以使用一个文件句柄作为可选的参数

STDERR

- 标准错误输出（默认屏幕）

STDIN

- 标准输入（默认键盘）

STDOUT

- 标准输出（默认屏幕）
- print 默认使用 STDOUT
- print 可以使用一个文件句柄作为可选的参数

STDERR

- 标准错误输出（默认屏幕）

教学提纲

碱基

4 = A + C + G + T/U

氨基酸

21 = 20(氨基酸) + 1(起始，同 M) + 1(终止)

密码子

- 每 3 个碱基组成一个密码子
- 一个密码子编码一种氨基酸
- $4 \times 4 \times 4 = 64 > 21$

碱基

4 = A + C + G + T/U

氨基酸

21 = 20(氨基酸) + 1(起始，同 M) + 1(终止)

密码子

- 每 3 个碱基组成一个密码子
- 一个密码子编码一种氨基酸
- $4 \times 4 \times 4 = 64 > 21$

碱基

4 = A + C + G + T/U

氨基酸

21 = 20(氨基酸) + 1(起始，同 M) + 1(终止)

密码子

- 每 3 个碱基组成一个密码子
- 一个密码子编码一种氨基酸
- $4 \times 4 \times 4 = 64 > 21$

遗传密码 | 冗余性

nonpolar | polar | basic | acidic | (stop codon)

Standard genetic code

1st base	2nd base								3rd base
	U		C		A		G		
U	UUU	(Phe/F) Phenylalanine	UCU	(Ser/S) Serine	UAU	(Tyr/Y) Tyrosine	UGU	(Cys/C) Cysteine	U
	UUC		UCC		UAC		UGC		C
	UUA	(Leu/L) Leucine	UCA		UAA	Stop (*Ochre*)	UGA	Stop (*Opal*)	A
	UUG		UCG		UAG	Stop (*Amber*)	UGG	(Trp/W) Tryptophan	G
C	CUU		CCU	(Pro/P) Proline	CAU	(His/H) Histidine	CGU	(Arg/R) Arginine	U
	CUC		CCC		CAC		CGC		C
	CUA		CCA		CAA	(Gln/Q) Glutamine	CGA		A
	CUG		CCG		CAG		CGG		G
A	AUU	(Ile/I) Isoleucine	ACU	(Thr/T) Threonine	AAU	(Asn/N) Asparagine	AGU	(Ser/S) Serine	U
	AUC		ACC		AAC		AGC		C
	AUA		ACA		AAA	(Lys/K) Lysine	AGA	(Arg/R) Arginine	A
	AUG[A]	(Met/M) Methionine	ACG		AAG		AGG		G
G	GUU	(Val/V) Valine	GCU	(Ala/A) Alanine	GAU	(Asp/D) Aspartic acid	GGU	(Gly/G) Glycine	U
	GUC		GCC		GAC		GGC		C
	GUA		GCA		GAA	(Glu/E) Glutamic acid	GGA		A
	GUG		GCG		GAG		GGG		G

Inverse table (compressed using IUPAC notation)

Amino acid	Codons	Compressed	Amino acid	Codons	Compressed
Ala/A	GCU, GCC, GCA, GCG	GCN	Leu/L	UUA, UUG, CUU, CUC, CUA, CUG	YUR, CUN
Arg/R	CGU, CGC, CGA, CGG, AGA, AGG	CGN, MGR	Lys/K	AAA, AAG	AAR
Asn/N	AAU, AAC	AAY	Met/M	AUG	
Asp/D	GAU, GAC	GAY	Phe/F	UUU, UUC	UUY
Cys/C	UGU, UGC	UGY	Pro/P	CCU, CCC, CCA, CCG	CCN
Gln/Q	CAA, CAG	CAR	Ser/S	UCU, UCC, UCA, UCG, AGU, AGC	UCN, AGY
Glu/E	GAA, GAG	GAR	Thr/T	ACU, ACC, ACA, ACG	ACN
Gly/G	GGU, GGC, GGA, GGG	GGN	Trp/W	UGG	
His/H	CAU, CAC	CAY	Tyr/Y	UAU, UAC	UAY
Ile/I	AUU, AUC, AUA	AUH	Val/V	GUU, GUC, GUA, GUG	GUN
START	AUG		STOP	UAA, UGA, UAG	UAR, URA

```perl
# codon2aa
#
# A subroutine to translate a DNA 3-character codon to an amino
  acid
#   Version 2

sub codon2aa {
  my($codon) = @_;

     if ( $codon =~ /GC./i )     { return 'A' } # Alanine
  elsif ( $codon =~ /TG[TC]/i ) { return 'C' } # Cysteine
  elsif ( $codon =~ /GA[TC]/i ) { return 'D' } # Aspartic Acid
  elsif ( $codon =~ /GA[AG]/i ) { return 'E' } # Glutamic Acid
  elsif ( $codon =~ /TT[TC]/i ) { return 'F' } # Phenylalanine
  elsif ( $codon =~ /GG./i )    { return 'G' } # Glycine
  elsif ( $codon =~ /CA[TC]/i ) { return 'H' } # Histidine
  elsif ( $codon =~ /AT[TCA]/i ){ return 'I' } # Isoleucine
  elsif ( $codon =~ /AA[AG]/i ) { return 'K' } # Lysine
  elsif ( $codon =~ /TT[AG]|CT./i ) { return 'L'} # Leucine
  elsif ( $codon =~ /ATG/i )    { return 'M' } # Methionine
  elsif ( $codon =~ /AA[TC]/i)  { return 'N' } # Asparagine
```

```
21    elsif ( $codon =~ /CC./i )     { return 'P' } # Proline
22    elsif ( $codon =~ /CA[AG]/i )  { return 'Q' } # Glutamine
23    elsif ( $codon =~ /CG.|AG[AG]/i ) { return 'R' } # Arginine
24    elsif ( $codon =~ /TC.|AG[TC]/i ) { return 'S' } # Serine
25    elsif ( $codon =~ /AC./i )     { return 'T' } # Threonine
26    elsif ( $codon =~ /GT./i )     { return 'V' } # Valine
27    elsif ( $codon =~ /TGG/i )     { return 'W' } # Tryptophan
28    elsif ( $codon =~ /TA[TC]/i )  { return 'Y' } # Tyrosine
29    elsif ( $codon =~ /TA[AG]|TGA/i ) { return '_' } # Stop
30    else {
31      print STDERR "Bad codon \"$codon\"!!\n";
32      exit;
33    }
34  }
```

优势

- 使用正则表达式展示了遗传密码的冗余性
- 氨基酸的单字母代码是按照字母顺序排列的

正则表达式

- `/[TC]/`：匹配 T 或者 C
- `/./`：匹配换行符以外的任意一个字符（此处为 ACGT）
- `/T/i`：不区分大小写，匹配 T 或者 t
- `/TC.|AG[TC]/`：匹配 `/TC./`或者 `/AG[TC]/`
- `/TA[AG]|TGA/`：等同于 `/T(A[AG]|GA)/`

优势

- 使用正则表达式展示了遗传密码的冗余性
- 氨基酸的单字母代码是按照字母顺序排列的

正则表达式

- `/[TC]/`：匹配 T 或者 C
- `/./`：匹配换行符以外的任意一个字符（此处为 ACGT）
- `/T/i`：不区分大小写，匹配 T 或者 t
- `/TC.|AG[TC]/`：匹配 `/TC./`或者 `/AG[TC]/`
- `/TA[AG]|TGA/`：等同于 `/T(A[AG]|GA)/`

教学提纲

```
1  #
2  # codon2aa
3  #
4  # A subroutine to translate a DNA 3-character
       codon to an amino acid
5  #   Version 3, using hash lookup
6
7  sub codon2aa {
8    my($codon) = @_;
9
10   $codon = uc $codon;
```

```
12  my(%genetic_code) = (
13
14  'TCA' => 'S',     # Serine
15  'TCC' => 'S',     # Serine
16  'TCG' => 'S',     # Serine
17  'TCT' => 'S',     # Serine
18  'TTC' => 'F',     # Phenylalanine
19  'TTT' => 'F',     # Phenylalanine
20  'TTA' => 'L',     # Leucine
21  'TTG' => 'L',     # Leucine
22  'TAC' => 'Y',     # Tyrosine
23  'TAT' => 'Y',     # Tyrosine
24  'TAA' => '_',     # Stop
25  'TAG' => '_',     # Stop
```

```
26    'TGC' => 'C',     # Cysteine
27    'TGT' => 'C',     # Cysteine
28    'TGA' => '_',     # Stop
29    'TGG' => 'W',     # Tryptophan
30    'CTA' => 'L',     # Leucine
31    'CTC' => 'L',     # Leucine
32    'CTG' => 'L',     # Leucine
33    'CTT' => 'L',     # Leucine
34    'CCA' => 'P',     # Proline
35    'CCC' => 'P',     # Proline
36    'CCG' => 'P',     # Proline
37    'CCT' => 'P',     # Proline
38    'CAC' => 'H',     # Histidine
39    'CAT' => 'H',     # Histidine
40    'CAA' => 'Q',     # Glutamine
41    'CAG' => 'Q',     # Glutamine
42    'CGA' => 'R',     # Arginine
43    'CGC' => 'R',     # Arginine
44    'CGG' => 'R',     # Arginine
45    'CGT' => 'R',     # Arginine
```

```
46    'ATA' => 'I',     # Isoleucine
47    'ATC' => 'I',     # Isoleucine
48    'ATT' => 'I',     # Isoleucine
49    'ATG' => 'M',     # Methionine
50    'ACA' => 'T',     # Threonine
51    'ACC' => 'T',     # Threonine
52    'ACG' => 'T',     # Threonine
53    'ACT' => 'T',     # Threonine
54    'AAC' => 'N',     # Asparagine
55    'AAT' => 'N',     # Asparagine
56    'AAA' => 'K',     # Lysine
57    'AAG' => 'K',     # Lysine
58    'AGC' => 'S',     # Serine
59    'AGT' => 'S',     # Serine
60    'AGA' => 'R',     # Arginine
61    'AGG' => 'R',     # Arginine
62    'GTA' => 'V',     # Valine
63    'GTC' => 'V',     # Valine
64    'GTG' => 'V',     # Valine
65    'GTT' => 'V',     # Valine
```

```
66    'GCA' => 'A',     # Alanine
67    'GCC' => 'A',     # Alanine
68    'GCG' => 'A',     # Alanine
69    'GCT' => 'A',     # Alanine
70    'GAC' => 'D',     # Aspartic Acid
71    'GAT' => 'D',     # Aspartic Acid
72    'GAA' => 'E',     # Glutamic Acid
73    'GAG' => 'E',     # Glutamic Acid
74    'GGA' => 'G',     # Glycine
75    'GGC' => 'G',     # Glycine
76    'GGG' => 'G',     # Glycine
77    'GGT' => 'G',     # Glycine
78    );
79
80    if(exists $genetic_code{$codon}) {
81      return $genetic_code{$codon};
82    }else{
83      print STDERR "Bad codon \"$codon\"!!\n";
84      exit;
85    }
```

代码解释

- `uc $codon`：把输入的参数转换为大写（子程序就可以同时处理大小写了）
- `exists $genetic_code{$condon}`：散列中存在 `$codon` 这个键就返回 true，否则返回 false

补充说明

- 散列的键必须是简单的标量值，不能使用正则表达式，不能使用字符集
- 使用子程序（做成“黑盒子”）对程序进行模块化组织
- 把 codon2aa 这个子程序放到 BeginPerlBioinfo.pm 模块文件中备用

代码解释

- `uc $codon`：把输入的参数转换为大写（子程序就可以同时处理大小写了）
- `exists $genetic_code{$condon}`：散列中存在 `$codon` 这个键就返回 true，否则返回 false

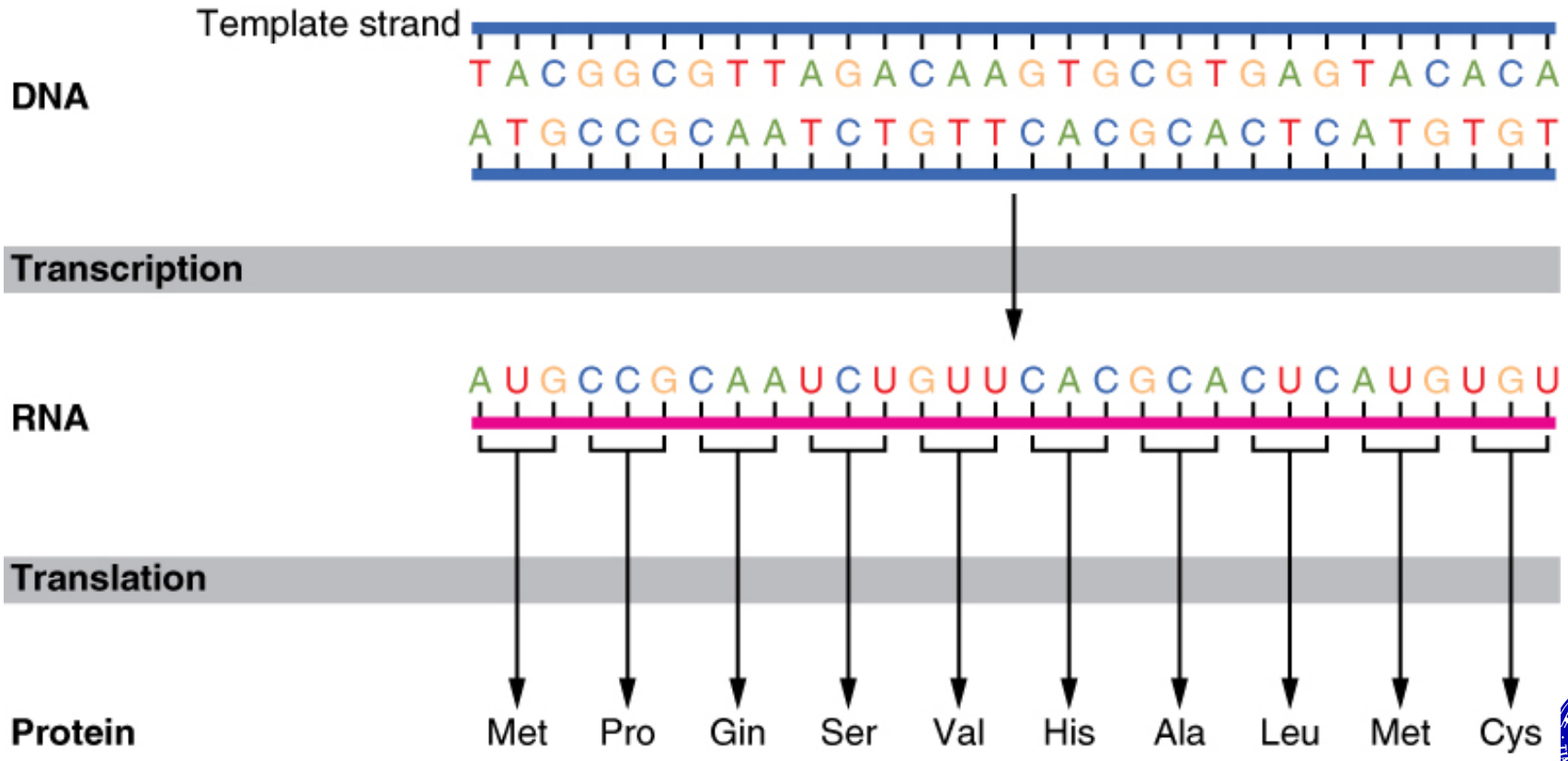
补充说明

- 散列的键必须是简单的标量值，不能使用正则表达式，不能使用字符集
- 使用子程序（做成“黑盒子”）对程序进行模块化组织
- 把 codon2aa 这个子程序放到 BeginPerlBioinfo.pm 模块文件中备用

教学提纲

目的

使用子程序 codon2aa 把整个 DNA 序列翻译成蛋白质。



DNA 翻译成蛋白质 | 程序 8.1.1

```
1  #!/usr/bin/perl -w
2  # Example 8-1   Translate DNA into protein
3
4  use strict;
5  use warnings;
6  use BeginPerlBioinfo;      # see Chapter 6
     about this module
7
8  # Initialize variables
9  my $dna = '
     CGACGTCTTCGTACGGGACTAGCTCGTGTCGGTCGC';
10 my $protein = '';
11 my $codon;
```

DNA 翻译成蛋白质 | 程序 8.1.2

```
13 # Translate each three-base codon into an
     amino acid, and append to a protein
14 for(my $i=0; $i < (length($dna) - 2) ; $i +=
     3) {
15     $codon = substr($dna,$i,3);
16     $protein .= codon2aa($codon);
17 }
18
19 print "I translated the DNA\n\n$dna\n\n  into
     the protein\n\n$protein\n\n";
20
21 exit;
```

DNA 翻译成蛋白质 | 程序 8.1 | 输出

```
1 I translated the DNA
2
3 CGACGTCTTCGTACGGGACTAGCTCGTGTCGGTCGC
4
5   into the protein
6
7 RRLRTGLARVGR
```

```
1 for (my $i = 0; $i < (length($dna) - 2); $i
    += 3) {
```

解释

- `my $i=0`：初始化计数器，限定作用域
- `$i<(length($dna)-2)`：判断是否继续翻译，密码子长度为 3，DNA 序列长 `length`，其中的碱基索引从 0 到 `length-1`
- `$i+=3`：计数器递增，一次处理 3 个碱基

```
1 for (my $i = 0; $i < (length($dna) - 2); $i
    += 3) {
```

解释

- `my $i=0`：初始化计数器，限定作用域
- `$i<(length($dna)-2)`：判断是否继续翻译，密码子长度为 3，DNA 序列长 `length`，其中的碱基索引从 0 到 `length-1`
- `$i+=3`：计数器递增，一次处理 3 个碱基

```
1 $codon = substr ($dna, $i, 3);
```

解释

- 从字符串 `$dna` 的位置 `$i` 开始向后提取 3 个字符
- 把提取的子字符串保存到变量 `$codon` 中

```
1 $codon = substr ($dna, $i, 3);
```

解释

- 从字符串 `$dna` 的位置 `$i` 开始向后提取 3 个字符
- 把提取的子字符串保存到变量 `$codon` 中

基本要求

- 输入：包含 DNA 的参数
- 返回：翻译后的肽链

问题简化

- 翻译整段 DNA（不需要提供指定起始和终止位点的参数）
- 移除不适合于子程序的 print 语句

DNA 翻译成蛋白质 | 程序 8.1 | 转换成子程序

基本要求

- 输入：包含 DNA 的参数
- 返回：翻译后的肽链

问题简化

- 翻译整段 DNA（不需要提供指定起始和终止位点的参数）
- 移除不适合于子程序的 print 语句

```perl
# dna2peptide
#
# A subroutine to translate DNA sequence into a peptide

sub dna2peptide {

  my($dna) = @_;

  use strict;
  use warnings;
  use BeginPerlBioinfo;      # see Chapter 6 about this module
```

```
13    # Initialize variables
14    my $protein = '';
15
16    # Translate each three-base codon to an
      amino acid, and append to a protein
17    for(my $i=0; $i < (length($dna) - 2) ; $i
      += 3) {
18       $protein .= codon2aa( substr($dna,$i,3));
19    }
20
21    return $protein;
22 }
```

易读 vs. 速度

- 使用 `$codon` 中间变量：更加易读
- 不使用 `$codon` 中间变量：避免字符串的复制，提高运行速度
- 当程序本身的易读性降低时，补充注释进行说明

use

- 可以在子程序中使用 use 载入模块
- 子程序的 use 可以和主程序的 use 存在冗余
- Perl 只会载入模块一次
- 子程序中载入模块有一定的优势（主程序中没有载入这些模块时）

易读 vs. 速度

- 使用 `$codon` 中间变量：更加易读
- 不使用 `$codon` 中间变量：避免字符串的复制，提高运行速度
- 当程序本身的易读性降低时，补充注释进行说明

use

- 可以在子程序中使用 use 载入模块
- 子程序的 use 可以和主程序的 use 存在冗余
- Perl 只会载入模块一次
- 子程序中载入模块有一定的优势（主程序中没有载入这些模块时）

教学提纲

教学提纲

读取 FASTA 格式的 DNA | 序列格式

- Plain sequence format
- FASTA format
- GenBank format
- EMBL format
- GCG format
- GCG-RSF (rich sequence format)
- IG format

Plain sequence format

A sequence in plain format may contain only IUPAC characters and spaces (no numbers!).

Note: A file in plain sequence format may only contain **one** sequence, while most other formats accept several sequences in one file.

An example sequence in plain format is:

```
ACAAGATGCCATTGTCCCCCGGCCTCCTGCTGCTGCTGCTCTCCGGGGCCACGGCCACCGCTGCCCTGCC
CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGC
CTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCATAGGAGAGG
AAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGCAATCCGCGCGCCGGGACAGAATGCC
CTGCAGGAACTTCTTCTGGAAGACCTTCTCCTCCTGCAAATAAAACCTCACCCATGAATGCTCACGCAAG
TTTAATTACAGACCTGAA
```

FASTA format

A sequence file in FASTA format can contain several sequences.
Each sequence in FASTA format begins with a single-line description, followed by lines of sequence data. The description line must begin with a greater-than (">") symbol in the first column.

An example sequence in FASTA format is:

```
>AB000263 |acc=AB000263|descr=Homo sapiens mRNA for prepro cortistatin like peptide, complete cds.|len=368
ACAAGATGCCATTGTCCCCCGGCCTCCTGCTGCTGCTGCTCTCCGGGGCCACGGCCACCGCTGCCCTGCC
CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGC
CTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCATAGGAGAGG
AAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGCAATCCGCGCGCCGGGACAGAATGCC
CTGCAGGAACTTCTTCTGGAAGACCTTCTCCTCCTGCAAATAAAACCTCACCCATGAATGCTCACGCAAG
TTTAATTACAGACCTGAA
```

GenBank format

A sequence file in GenBank format can contain several sequences.
One sequence in GenBank format starts with a line containing the word LOCUS and a number of annotation lines. The start of the sequence is marked by a line containing "ORIGIN" and the end of the sequence is marked by two slashes ("//").

An example sequence in GenBank format is:

```
LOCUS       AB000263                368 bp    mRNA    linear   PRI 05-FEB-1999
DEFINITION  Homo sapiens mRNA for prepro cortistatin like peptide, complete
            cds.
ACCESSION   AB000263
ORIGIN
        1 acaagatgcc attgtccccc ggcctcctgc tgctgctgct ctccggggcc acggccaccg
       61 ctgccctgcc cctggagggt ggccccaccg gccgagacag cgagcatatg caggaagcgg
      121 caggaataag gaaaagcagc ctcctgactt tcctcgcttg gtggtttgag tggacctccc
      181 aggccagtgc cgggcccctc ataggagagg aagctcggga ggtggccagg cggcaggaag
      241 gcgcaccccc ccagcaatcc gcgcgccggg acagaatgcc ctgcaggaac ttcttctgga
      301 agaccttctc ctcctgcaaa taaaacctca cccatgaatg ctcacgcaag tttaattaca
      361 gacctgaa
//
```

EMBL format

A sequence file in EMBL format can contain several sequences.
One sequence entry starts with an identifier line ("ID"), followed by further annotation lines. The start of the sequence is marked by a line starting with "SQ" and the end of the sequence is marked by two slashes ("//").

An example sequence in EMBL format is:

```
ID   AB000263 standard; RNA; PRI; 368 BP.
XX
AC   AB000263;
XX
DE   Homo sapiens mRNA for prepro cortistatin like peptide, complete cds.
XX
SQ   Sequence 368 BP;
     acaagatgcc attgtccccc ggcctcctgc tgctgctgct ctccggggcc acggccaccg        60
     ctgccctgcc cctggagggt ggccccaccg gccgagacag cgagcatatg caggaagcgg       120
     caggaataag gaaaagcagc ctcctgactt tcctcgcttg gtggtttgag tggacctccc       180
     aggccagtgc cgggcccctc ataggagagg aagctcggga ggtggccagg cggcaggaag       240
     gcgcaccccc ccagcaatcc gcgcgccggg acagaatgcc ctgcaggaac ttcttctgga       300
     agaccttctc ctcctgcaaa taaaacctca cccatgaatg ctcacgcaag tttaattaca       360
     gacctgaa                                                               368
//
```

GCG format

A sequence file in GCG format contains exactly one sequence, begins with annotation lines and the start of the sequence is marked by a line ending with two dot ("..") characters. This line also contains the sequence identifier, the sequence length and a checksum. This format should only be used if the file was created with the GCG package.

An example sequence in GCG format is:

```
ID   AB000263 standard; RNA; PRI; 368 BP.
XX
AC   AB000263;
XX
DE   Homo sapiens mRNA for prepro cortistatin like peptide, complete cds.
XX
SQ   Sequence 368 BP;
AB000263  Length: 368  Check: 4514  ..
       1  acaagatgcc attgtccccc ggcctcctgc tgctgctgct ctccggggcc acggccaccg
      61  ctgccctgcc cctggagggt ggccccaccg gccgagacag cgagcatatg caggaagcgg
     121  caggaataag gaaaagcagc ctcctgactt tcctcgcttg gtggtttgag tggacctccc
     181  aggccagtgc cgggcccctc ataggagagg aagctcggga ggtggccagg cggcaggaag
     241  gcgcacccca ccagcaatcc gcgcgccggg acagaatgcc ctgcaggaac ttcttctgga
     301  agaccttctc ctcctgcaaa taaaacctca cccatgaatg ctcacgcaag tttaattaca
     361  gacctgaa
```

IG format

A sequence file in IG format can contain several sequences, each consisting of a number of comment lines that must begin with a semicolon (";"), a line with the sequence name (it may not contain spaces!) and the sequence itself terminated with the termination character '1' for linear or '2' for circular sequences.

An example sequence in IG format is:

```
; comment
; comment
AB000263
ACAAGATGCCATTGTCCCCCGGCCTCCTGCTGCTGCTGCTCTCCGGGGCCACGGCCACCGCTGCCCTGCC
CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGC
CTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCATAGGAGAGG
AAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGCAATCCGCGCGCCGGGACAGAATGCC
CTGCAGGAACTTCTTCTGGAAGACCTTCTCCTCCTGCAAATAAAACCTCACCCATGAATGCTCACGCAAG
TTTAATTACAGACCTGAA1
```

教学提纲

读取文件的两种策略

1. 从打开的文件中一次读入一行，边读入、边处理
   - 大文件
   - 寻找小的信息片段
2. 一次性把整个文件都读入到数组中，然后对数组进行操作
   - 中、小文件
   - 容易遍历整个数据进行操作

读取 FASTA 文件 | 思路

要求

- 输入：FASTA 文件的文件名参数
- 返回：FASTA 文件中的序列数据

思路

- 一个子程序：打开、读取 FATSA 文件并提取序列数据
- 两个子程序
  1. 第一个子程序：打开并读取文件
  2. 第二个子程序：提取序列数据

读取 FASTA 文件 | 思路

要求

- 输入：FASTA 文件的文件名参数
- 返回：FASTA 文件中的序列数据

思路

- 一个子程序：打开、读取 FATSA 文件并提取序列数据
- 两个子程序
  1. 第一个子程序：打开并读取文件
  2. 第二个子程序：提取序列数据

```
subroutine get data from a file

  argument = filename

  open file
    if can't open, print error message and
  exit

  read in data and

  return @data
}
```

```
1  Subroutine extract sequence data from fasta file
2
3    argument = array of file data in fasta format
4
5      Discard all header lines
6      (and blank and comment lines for good measure)
7      If first character of first line is >, discard
       it
8
9    Read in the rest of the file, join in a scalar,
10     edit out nonsequence data
11
12   return sequence
13 }
```

读取 FASTA 文件 | 读取失败

如果文件不能被读取，可能的处理方法：

- 退出程序：比较极端
- 向用户询问文件名：人性化，不利于自动化
- 使用默认的文件：人性化，自动化
- 返回 false（空数组）：容易和空文件相混淆
- 其他处理方法：返回 undefined

教学提纲

读取 FASTA 文件 | 子程序

```
1  # get_file_data
2  # A subroutine to get data from a file given its
     filename
3
4  sub get_file_data {
5    my($filename) = @_;
6    use strict;
7    use warnings;
8    # Initialize variables
9    my @filedata = ( );
10   unless( open(GET_FILE_DATA, $filename) ) {
11     print STDERR "Cannot open file \"$filename\"\n\n";
12     exit;
13   }
14   @filedata = <GET_FILE_DATA>;
15   close GET_FILE_DATA;
16   return @filedata;
17 }
```

读取 FASTA 文件 | 子程序

```
31 # extract_sequence_from_fasta_data: A subroutine to
     extract FASTA sequence data from an array
32 sub extract_sequence_from_fasta_data {
33   my(@fasta_file_data) = @_;
34   use strict; use warnings;
35   my $sequence = ''; # Declare and initialize variables
36   foreach my $line (@fasta_file_data) {
37     # discard blank line
38     if ($line =~ /^\s*$/) { next;
39     # discard comment line
40     } elsif($line =~ /^\s*#/) { next;
41     # discard fasta header line
42     } elsif($line =~ /^>/) { next;
43     # keep line, add to sequence string
44     } else { $sequence .= $line; }
45   }
46   # remove non-sequence data (in this case, whitespace)
     from $sequence string
47   $sequence =~ s/\s//g; return $sequence;
48 }
```

```
1  # 使用my声明循环中使用的变量
2  foreach my $line (@fasta_file_data) {
3
4  # 匹配只有/没有空白的空行
5  if ($line =~ /^\s*$/) {
6
7  # 匹配注释行
8  } elsif($line =~ /^\s*#/) {
9
10 # 匹配FASTA标题行
11 } elsif($line =~ /^>/) {
12
13 # 删除空白字符（含换行符）
14 $sequence =~ s/\s//g;
```

教学提纲

格式化序列数据

- 目的：把序列数据格式化成适当长度
- 输入：序列和行的长度两个参数
- 输出：按指定要求格式化后的序列数据

```
1  # print_sequence
2  # A subroutine to format and print sequence
     data
3
4  sub print_sequence {
5    my($sequence, $length) = @_;
6    use strict;
7    use warnings;
8    # Print sequence in lines of $length
9    for ( my $pos = 0 ; $pos < length($sequence
     ) ; $pos += $length ) {
10     print substr($sequence, $pos, $length), "
     \n";
11   }
12 }
```

教学提纲

读取 FASTA 文件 | 程序 8.2.1

```
1  #!/usr/bin/perl -w
2  # Example 8-2   Read a fasta file and extract
       the sequence data
3
4  use strict;
5  use warnings;
6  use BeginPerlBioinfo;      # see Chapter 6
     about this module
7
8  # Declare and initialize variables
9  my @file_data = (  );
10 my $dna = '';
```

读取 FASTA 文件 | 程序 8.2.1

```
12 # Read in the contents of the file "sample.
     dna"
13 @file_data = get_file_data("sample.dna");
14
15 # Extract the sequence data from the contents
     of the file "sample.dna"
16 $dna = extract_sequence_from_fasta_data(
     @file_data);
17
18 # Print the sequence in lines 25 characters
     long
19 print_sequence($dna, 25);
20
21 exit;
```

读取 FASTA 文件 | 程序 8.2 | 输出

```
1  agatggcggcgctgaggggtcttgg
2  gggctctaggccggccacctactgg
3  tttgcagcggagacgacgcatgggg
4  cctgcgcaataggagtacgctgcct
5  gggaggcgtgactagaagcggaagt
6  ...
7  ccaacagcagccacagccatcacag
8  aagttagggcgcatccgtgaagatg
9  agggggcagtggcgtcatcaacagt
10 caaggagcctcctgaggctacagcc
11 acacctgagccactctcagatgagg
12 accta
```

教学提纲

DNA 翻译成蛋白质 | 程序 8.3.1

```
1  #!/usr/bin/perl -w
2  # Example 8-3   Read a fasta file and extract
       the DNA sequence data
3  # Translate it to protein and print it out in
       25-character-long lines
4
5  use strict;
6  use warnings;
7  use BeginPerlBioinfo;      # see Chapter 6
     about this module
8
9  # Initialize variables
10 my @file_data = (  );
11 my $dna = '';
12 my $protein = '';
```

DNA 翻译成蛋白质 | 程序 8.3.1

```
13 # Read in the contents of the file "sample.
     dna"
14 @file_data = get_file_data("sample.dna");
15
16 # Extract the sequence data from the contents
     of the file "sample.dna"
17 $dna = extract_sequence_from_fasta_data(
     @file_data);
18
19 # Translate the DNA to protein
20 $protein = dna2peptide($dna);
21
22 # Print the sequence in lines 25 characters
     long
23 print_sequence($protein, 25);
24
```

DNA 翻译成蛋白质 | 程序 8.3 | 输出

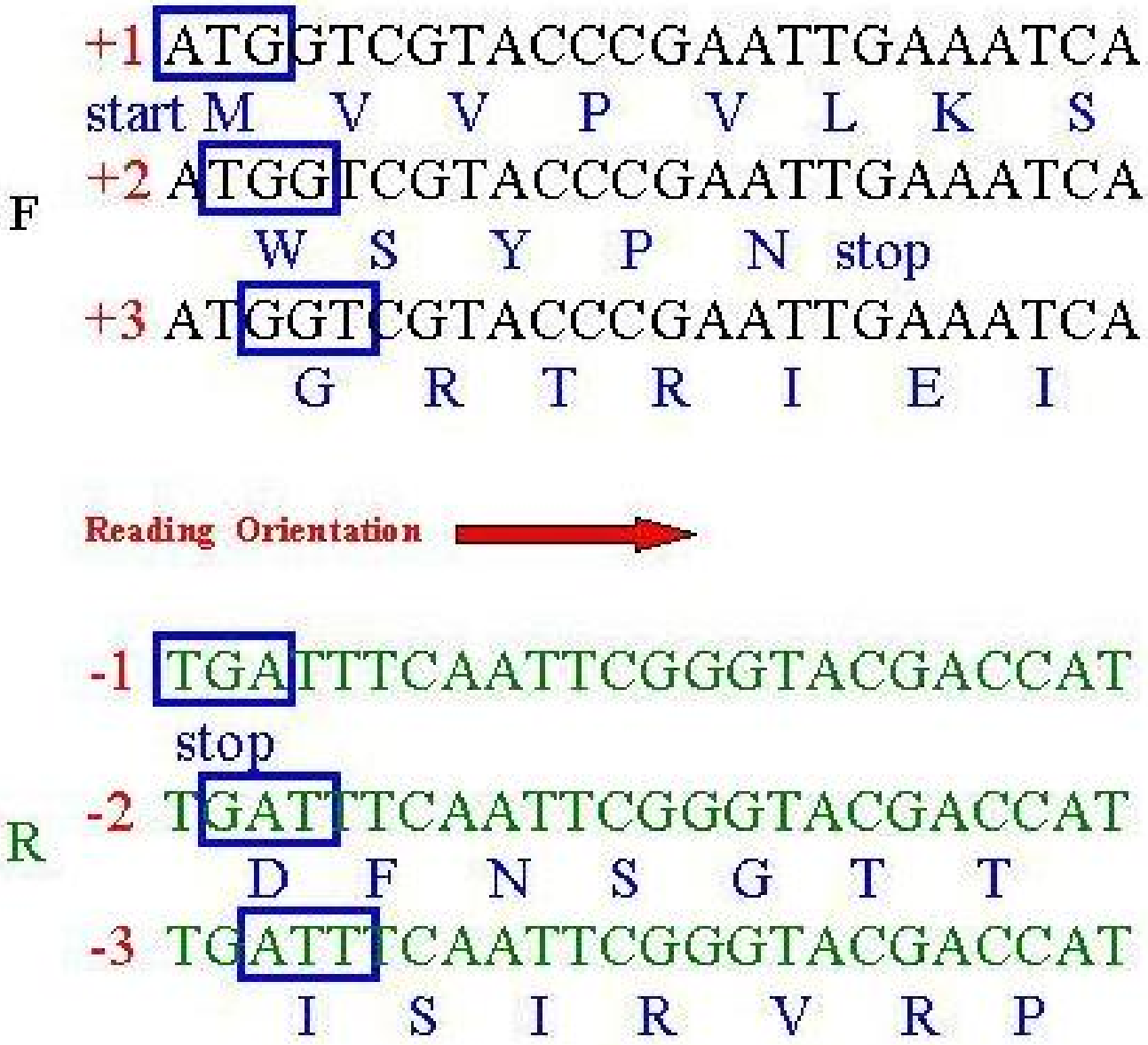
```
 1 RWRR_GVLGALGRPPTGLQRRRRMG
 2 PAQ_EYAAWEA_LEAEVVVGAFATA
 3 WDAAEWSVQVRGSLAGVVRECAGSG
 4 DMEGDGSDPEPPDAGEDSKSENGEN
 5 APIYCICRKPDINCFMIGCDNCNEW
 6 FHGDCIRITEKMAKAIREWYCRECR
 7 EKDPKLEIRYRHKKSRERDGNERDS
 8 SEPRDEGGGRKRPVPDPDLQRRAGS
 9 GTGVGAMLARGSASPHKSSPQPLVA
10 TPSQHHQQQQQQIKRSARMCGECEA
11 CRRTEDCGHCDFCRDMKKFGGPNKI
12 RQKCRLRQCQLRARESYKYFPSSLS
13 PVTPSESLPRPRRPLPTQQQPQPSQ
14 KLGRIREDEGAVASSTVKEPPEATA
15 TPEPLSDEDL
```

教学提纲

教学提纲

阅读框架（reading frame）是指 RNA 或 DNA 中，一组连续且不重复的 3 核苷酸密码子。每一条 mRNA 共有 3 种可能的阅读框架，而双股的 DNA 则每股各 3 种，共 6 种阅读框架。

同一序列具有多种阅读框架的现象使重叠基因（overlapping genes）可能存在，已知常见于某些细菌与病毒。至于人类细胞中则较少见。


+1 ATGGTCGTACCCCGAATTGAAAATCA
start M V V P V L K S
+2 ATGGTCGTACCCCGAATTGAAAATCA
F
W S Y P N stop
+3 ATGGTCGTACCCCGAATTGAAAATCA
G R T R I E I
Reading Orientation
-1 TGATTTCAATTCGGGTACGACCAT
stop
-2 TGATTTCAATTCGGGTACGACCAT
R
D F N S G T T
-3 TGATTTCAATTCGGGTACGACCAT
I S I R V R P

开放阅读框（open reading frame，ORF）是指在给定的阅读框架中，不包含终止密码子的一串序列。这段序列是生物个体的基因组中，可能作为蛋白质编码序列的部分。基因中的 ORF 包含并位于起始编码与终止编码之间。

由于一段 DNA 或 RNA 序列有多种不同读取方式，因此可能同时存在许多不同的开放阅读框架。

开放阅读框是判断 DNA 序列中存在基因的一个重要依据；基因识别程序需要进行开放阅读框的分析。

阅读框 | 动手之前的思考

基本思想

- 一切高大上的问题/任务都是纸老虎

解决思路

- 四处寻找已经写好的、可以完成该任务的（子）程序/工具
- 充分利用自己收集的子程序库
  - 有哪些相关的子程序可以直接使用
  - 有哪些相关的子程序需要进行修改/扩展
  - 还缺少哪些子程序需要编写添加

基本思想

- 一切高大上的问题/任务都是纸老虎

解决思路

- 四处寻找已经写好的、可以完成该任务的（子）程序/工具
- 充分利用自己收集的子程序库
  - 有哪些相关的子程序可以直接使用
  - 有哪些相关的子程序需要进行修改/扩展
  - 还缺少哪些子程序需要编写添加

已有的子程序

- get_file_data：打开读取文件
- extract_sequence_from_fasta_data：提取 FASTA 文件中的序列
- dna2peptide：把 DNA 翻译成多肽
  - codon2aa：把密码子翻译成氨基酸
- print_sequence：输出格式化的序列

尚缺的子程序

- revcom：计算反向互补序列
- translate_frame：翻译 DNA 的指定区间

已有的子程序

- get_file_data：打开读取文件
- extract_sequence_from_fasta_data：提取 FASTA 文件中的序列
- dna2peptide：把 DNA 翻译成多肽
  - codon2aa：把密码子翻译成氨基酸
- print_sequence：输出格式化的序列

尚缺的子程序

- revcom：计算反向互补序列
- translate_frame：翻译 DNA 的指定区间

教学提纲

```perl
# revcom
# A subroutine to compute the reverse complement of DNA sequence
sub revcom {
    my($dna) = @_;

    # First reverse the sequence
    my($revcom) = reverse($dna);

    # Next, complement the sequence, dealing with upper and lower case
    # A->T, T->A, C->G, G->C
    $revcom =~ tr/ACGTacgt/TGCAtgca/;

    return $revcom;
}
```

```
1 Given DNA sequence
2
3 subroutine translate_frame ( DNA, start, end )
4
5   return dna2peptide( substr( DNA, start, end - start +
    1 ) )
6 }
```

```
1 Given DNA sequence
2
3 subroutine translate_frame ( DNA, start, end )
4
5   # start and end are numbering the sequence from 1 to
    length
6   return dna2peptide( substr( DNA, start - 1, end -
    start + 1 ) )
7 }
```

```
1 Given DNA sequence
2
3 subroutine translate_frame ( DNA, start, end )
4
5   return dna2peptide( substr( DNA, start, end - start +
    1 ) )
6 }
```

```
1 Given DNA sequence
2
3 subroutine translate_frame ( DNA, start, end )
4
5   # start and end are numbering the sequence from 1 to
    length
6   return dna2peptide( substr( DNA, start - 1, end -
    start + 1 ) )
7 }
```

两种索引

- Perl 的处理方式：从 0 开始
- 生物学家的处理方式：从 1 开始

推荐策略

- 面向程序：以 0 起始的索引方式（计算简便）
- 面向用户：以 1 起始的索引方式（人性化）

两种索引

- Perl 的处理方式：从 0 开始
- 生物学家的处理方式：从 1 开始

推荐策略

- 面向程序：以 0 起始的索引方式（计算简便）
- 面向用户：以 1 起始的索引方式（人性化）

```perl
# translate_frame
# A subroutine to translate a frame of DNA

sub translate_frame {
  my($seq, $start, $end) = @_;
  #my $protein;

  # To make the subroutine easier to use, you won't need
   to specify the end point--it will just go to the end
  of the sequence by default.
  unless($end) {
    $end = length($seq);
  }

  # Finally, calculate and return the translation
  return dna2peptide ( substr ( $seq, $start - 1, $end -
  $start + 1 ) );
}
```

```perl
#!/usr/bin/perl -w
# Example 8-4   Translate a DNA sequence in all six reading frames

use strict;
use warnings;
use BeginPerlBioinfo;       # see Chapter 6 about this module

# Initialize variables
my @file_data = (  );
my $dna = '';
my $revcom = '';
my $protein = '';
```

```
14 # Read in the contents of the file "sample.dna"
15 @file_data = get_file_data("sample.dna");
16
17 # Extract the sequence data from the contents of
     the file "sample.dna"
18 $dna = extract_sequence_from_fasta_data(@file_data
     );
19
20 # Translate the DNA to protein in six reading
     frames
21 #   and print the protein in lines 70 characters
     long
22 print "\n -------Reading Frame 1--------\n\n";
23 $protein = translate_frame($dna, 1);
24 print_sequence($protein, 70);
```

```
26 print "\n -------Reading Frame 2---------\n\n"
     ;
27 $protein = translate_frame($dna, 2);
28 print_sequence($protein, 70);
29
30 print "\n -------Reading Frame 3---------\n\n"
     ;
31 $protein = translate_frame($dna, 3);
32 print_sequence($protein, 70);
```

```
34 # Calculate reverse complement
35 $revcom = revcom($dna);
36
37 print "\n -------Reading Frame 4--------\n\n";
38 $protein = translate_frame($revcom, 1);
39 print_sequence($protein, 70);
40
41 print "\n -------Reading Frame 5--------\n\n";
42 $protein = translate_frame($revcom, 2);
43 print_sequence($protein, 70);
44
45 print "\n -------Reading Frame 6--------\n\n";
46 $protein = translate_frame($revcom, 3);
47 print_sequence($protein, 70);
48
49 exit;
```

```
 -------Reading Frame 1--------

RWRR_GVLGALGRPPTGLQRRRRMGPAQ_EYAAWEA_LEAEVVVGAFATAWDAAEWSVQVRGSLAGVVRE
CAGSGDMEGDGSDPEPPDAGEDSKSENGENAPIYCICRKPDINCFMIGCDNCNEWFHGDCIRITEKMAKA
IREWYCRECREKDPKLEIRYRHKKSRERDGNERDSSEPRDEGGGRKRPVPDPDLQRRAGSGTGVGAMLAR
GSASPHKSSPQPLVATPSQHHQQQQQQIKRSARMCGECEACRRTEDCGHCDFCRDMKKFGGPNKIRQKCR
LRQCQLRARESYKYFPSSLSPVTPSESLPRPRRPLPTQQQPQPSQKLGRIREDEGAVASSTVKEPPEATA
TPEPLSDEDL

...

-------Reading Frame 6--------

GPHLRVAQVWL_PQEAP_LLMTPLPPHLHGCALTSVMAVAAVGWAVAGGALAGTLRASLVRARKGSTCTI
PGPAAGTGAAGTSAGSCWGPRTSSCPDRNHSDHSPQCADMPHTHHTCGLTV_SAAAAAGDAGWVWPPRAA
ERICGAKQSPEQAWPQPLSLTLPGAAGLDQGQASCALHPHPGAHCCPAHCHPAPVTSCADSESLAWGLSL
CTPDSTTPGWPWPSSQ_SGCSPHGTTHCSCHTRS_SS_CPVCGRCSRWAHSPHSRTCCPPRHLEALGLNH
LPPYLRSRRTPSRPPPATREPAQTTRRRPRRLQRRPQLLPLLVTPPRQRTPIAQAPCVVSAANQ_VAGLE
PPRPLSAAI
```

教学提纲

教学提纲

知识点

- 散列：键，值，初始化，检查定义/存在，排序，……
- Perl 中的数据类型：标量变量，数组，散列
- 正则表达式：元字符，字符集，修饰符，择一匹配
- 序列格式：plain，FASTA，GenBank，EMBL，……
- Perl 读取文件：两种策略，读取失败的处理方法
- 阅读框：6 种框架，开放阅读框
- 其他：数据结构，比较排序，文件句柄，循环，子字符串提取，索引，……

技能

- 熟练使用 Perl 中的散列
- 熟练使用 Perl 读取文件
- 能编写把 DNA 翻译成蛋白质的 Perl 程序

知识点

- 散列：键，值，初始化，检查定义/存在，排序，……
- Perl 中的数据类型：标量变量，数组，散列
- 正则表达式：元字符，字符集，修饰符，择一匹配
- 序列格式：plain，FASTA，GenBank，EMBL，……
- Perl 读取文件：两种策略，读取失败的处理方法
- 阅读框：6 种框架，开放阅读框
- 其他：数据结构，比较排序，文件句柄，循环，子字符串提取，索引，……

技能

- 熟练使用 Perl 中的散列
- 熟练使用 Perl 读取文件
- 能编写把 DNA 翻译成蛋白质的 Perl 程序

教学提纲

1. 比较 Perl 的三种常见数据类型。
2. 如何获取散列中的所有键/值?
3. 如何对字符串/数字进行正序/逆序的排序?
4. 举例说明不同任务下数据结构的选择。
5. 如何使用正则表达式表征密码子?
6. 列举并举例说明常见的序列格式。
7. 比较读取文件的两种策略。
8. 举例说明 6 种阅读框。
9. 比较两种不同的索引方式。
10. 举例说明子程序的使用。

下节预告

计算机

回顾正则表达式的相关知识：元字符，字符集，量词，修饰符，……

生物学

回顾限制性核酸内切酶的相关知识：定义，分类，特点，……

教学提纲

教学提纲

- arrays of arrays，数组的数组
- hashes of arrays，散列的元素为数组
- arrays of hashes，数组的元素为散列
- hashes of hashes，散列的元素为散列
- more elaborate constructs，更复杂的结构

```
@AoA = (
      [ "fred", "barney" ],
      [ "george", "jane", "elroy" ],
      [ "homer", "marge", "bart" ],
);
```

```
1  # reading from file
2  while ( <> ) {
3      push @AoA, [ split ];
4  }
5  # calling a function
6  for $i ( 1 .. 10 ) {
7      $AoA[$i] = [ somefunc($i) ];
8  }
9  # using temp vars
10 for $i ( 1 .. 10 ) {
11     @tmp = somefunc($i);
12     $AoA[$i] = [ @tmp ];
13 }
14 # add to an existing row
15 push @{ $AoA[0] }, "wilma", "betty";
```

```
1 # one element
2 $AoA[0][0] = "Fred";
3
4 # another element
5 $AoA[1][1] =~ s/(\w)/\u$1/;
```

```
1  # print the whole thing with refs
2  for $aref ( @AoA ) {
3    print "\t [ @$aref ],\n";
4  }
5  # print the whole thing with indices
6  for $i ( 0 .. $#AoA ) {
7    print "\t [ @{$AoA[$i]} ],\n";
8  }
9  # print the whole thing one at a time
10 for $i ( 0 .. $#AoA ) {
11   for $j ( 0 .. $#{ $AoA[$i] } ) {
12     print "element $i $j is $AoA[$i][$j]\n";
13   }
14 }
```

```
1 %HoA = (
2    flintstones => [ "fred", "barney" ],
3    jetsons     => [ "george", "jane", "elroy"
     ],
4    simpsons    => [ "homer", "marge", "bart"
     ],
5 );
```

```perl
# reading from file
# flintstones: fred barney wilma dino
while ( <> ) {
    next unless s/^(.*?):\s*//;
    $HoA{$1} = [ split ];
}

# reading from file; more temps
# flintstones: fred barney wilma dino
while ( $line = <> ) {
    ($who, $rest) = split /:\s*/, $line, 2;
    @fields = split ' ', $rest;
    $HoA{$who} = [ @fields ];
}
```

```
1  # calling a function that returns a list
2  for $group ( "simpsons", "jetsons", "
     flintstones" ) {
3    $HoA{$group} = [ get_family($group) ];
4  }
5  # likewise, but using temps
6  for $group ( "simpsons", "jetsons", "
     flintstones" ) {
7    @members = get_family($group);
8    $HoA{$group} = [ @members ];
9  }
10
11 # append new members to an existing family
12 push @{ $HoA{"flintstones"} }, "wilma", "
     betty";
```

```perl
# one element
$HoA{flintstones}[0] = "Fred";
# another element
$HoA{simpsons}[1] =~ s/(\w)/\u$1/;
# print the whole thing
foreach $family ( keys %HoA ) {
  print "$family: @{ $HoA{$family} }\n";
}
# print the whole thing with indices
foreach $family ( keys %HoA ) {
  print "family: ";
  foreach $i ( 0 .. $#{ $HoA{$family} } ) {
    print " $i = $HoA{$family}[$i]";
  }
  print "\n";
}
```

```
1 # print the whole thing sorted by number of
    members
2 foreach $family ( sort { @{$HoA{$b}} <=> @{
    $HoA{$a}} } keys %HoA ) {
3     print "$family: @{ $HoA{$family} }\n";
4 }
5
6 # print the whole thing sorted by number of
    members and name
7 foreach $family ( sort { @{$HoA{$b}} <=> @{
    $HoA{$a}} || $a cmp $b } keys %HoA  ) {
8     print "$family: ", join(", ", sort @{
    $HoA{$family} }), "\n";
9 }
```

```
1  @AoH = (
2     {
3        Lead   => "fred",
4        Friend => "barney",
5     },
6     {
7        Lead   => "george",
8        Wife   => "jane",
9        Son    => "elroy",
10    },
11    {
12       Lead   => "homer",
13       Wife   => "marge",
14       Son    => "bart",
15    },
16 );
```

```
1  # reading from file
2  # format: LEAD=fred FRIEND=barney
3  while ( <> ) {
4     $rec = {};
5     for $field ( split ) {
6        ($key, $value) = split /=/, $field;
7        $rec->{$key} = $value;
8     }
9     push @AoH, $rec;
10 }
11
12 # no temp
13 while ( <>  ) {
14    push @AoH, { split /[\s+=]/ };
15 }
```

```
1  # calling a function  that returns a key/
     value pair list, like
2  # "lead","fred","daughter","pebbles"
3  while ( %fields = getnextpairset()  ) {
4    push @AoH, { %fields };
5  }
6
7  # likewise, but using no temp vars
8  while (<>) {
9    push @AoH, { parsepairs($_) };
10 }
11
12 # add key/value to an element
13 $AoH[0]{pet} = "dino";
14 $AoH[2]{pet} = "santa's little helper";
```

```
1 # one element
2 $AoH[0]{lead} = "fred";
3
4 # another element
5 $AoH[1]{lead} =~ s/(\w)/\u$1/;
```

```perl
# print the whole thing with refs
for $href ( @AoH ) {
  print "{ ";
  for $role ( keys %$href ) {
    print "$role=$href->{$role} ";
  }
  print " }\n";
}
# print the whole thing with indices
for $i ( 0 .. $#AoH ) {
  print "$i is { ";
  for $role ( keys %{ $AoH[$i] } ) {
    print "$role=$AoH[$i]{$role} ";
  }
  print " }\n";
}
```

```
1 # print the whole thing one at a time
2 for $i ( 0 .. $#AoH ) {
3    for $role ( keys %{ $AoH[$i] } ) {
4       print "element $i $role is $AoH[$i]{$role
    }\n";
5    }
6 }
```

```
1  %HoH = (
2    flintstones => {
3      lead      => "fred",
4      pal       => "barney",
5    },
6    jetsons     => {
7      lead      => "george",
8      wife      => "jane",
9      "his boy" => "elroy",
10   },
11   simpsons    => {
12     lead      => "homer",
13     wife      => "marge",
14     kid       => "bart",
15   },
16 );
```

```
1  # reading from file
2  # flintstones: lead=fred pal=barney wife=
     wilma pet=dino
3  while ( <> ) {
4    next unless s/^(.*?):\s*//;
5    $who = $1;
6    for $field ( split ) {
7      ($key, $value) = split /=/, $field;
8      $HoH{$who}{$key} = $value;
9    }
10 }
```

```
1  # reading from file; more temps
2  # flintstones: lead=fred pal=barney wife=
     wilma pet=dino
3  while ( <> ) {
4     next unless s/^(.*?):\s*//;
5     $who = $1;
6     $rec = {};
7     $HoH{$who} = $rec;
8     for $field ( split ) {
9        ($key, $value) = split /=/, $field;
10       $rec->{$key} = $value;
11    }
12 }
```

```
1  # calling a function that returns a key,value
       hash
2  for $group ( "simpsons", "jetsons", "
     flintstones" ) {
3     $HoH{$group} = { get_family($group) };
4  }
5
6  # likewise, but using temps
7  for $group ( "simpsons", "jetsons", "
     flintstones" ) {
8     %members = get_family($group);
9     $HoH{$group} = { %members };
10 }
```

```
1 # append new members to an existing family
2 %new_folks = (
3   wife => "wilma",
4   pet  => "dino",
5 );
6
7 for $what (keys %new_folks) {
8   $HoH{flintstones}{$what} = $new_folks{$what
    };
9 }
```

```
1 # one element
2 $HoH{flintstones}{wife} = "wilma";
3
4 # another element
5 $HoH{simpsons}{lead} =~ s/(\w)/\u$1/;
```

```
1 # print the whole thing
2 foreach $family ( keys %HoH ) {
3   print "$family: { ";
4   for $role ( keys %{ $HoH{$family} } ) {
5     print "$role=$HoH{$family}{$role} ";
6   }
7   print " }\n";
8 }
```

```
1 # print the whole thing somewhat sorted
2 foreach $family ( sort keys %HoH ) {
3     print "$family: { ";
4     for $role ( sort keys %{ $HoH{$family} }
    ) {
5         print "$role=$HoH{$family}{$role} ";
6     }
7     print " }\n";
8 }
```

```
1 # print the whole thing sorted by number of
    members
2 foreach $family ( sort { keys %{$HoH{$b}} <=>
     keys %{$HoH{$a}} } keys %HoH  ) {
3     print "$family: { ";
4     for $role ( sort keys %{ $HoH{$family} }
    ) {
5         print "$role=$HoH{$family}{$role} ";
6     }
7     print " }\n";
8 }
```

```
1  # establish a sort order (rank) for each role
2  $i = 0;
3  for ( qw(lead wife son daughter pal pet) ) {
     $rank{$_} = ++$i }
4  # now print the whole thing sorted by number
     of members
5  foreach $family ( sort { keys %{ $HoH{$b}  }
     <=> keys %{ $HoH{$a} } } keys %HoH  ) {
6      print "$family: { ";
7      # and print these according to rank order
8      for $role ( sort { $rank{$a} <=> $rank{$b
     } } keys %{ $HoH{$family} } ) {
9          print "$role=$HoH{$family}{$role} ";
10     }
11     print " }\n";
12 }
```

```
1 $rec = {
2   TEXT       => $string,
3   SEQUENCE   => [ @old_values ],
4   LOOKUP     => { %some_table },
5   THATCODE   => \&some_function,
6   THISCODE   => sub { $_[0] ** $_[1] },
7   HANDLE     => \*STDOUT,
8 };
```

```
1  print $rec->{TEXT};
2  print $rec->{SEQUENCE}[0];
3  $last = pop @ { $rec->{SEQUENCE} };
4  print $rec->{LOOKUP}{"key"};
5  ($first_k, $first_v) = each %{ $rec->{LOOKUP}
       };
6  $answer = $rec->{THATCODE}->($arg);
7  $answer = $rec->{THISCODE}->($arg1, $arg2);
8  # careful of extra block braces on fh ref
9  print { $rec->{HANDLE} } "a string\n";
10 use FileHandle;
11 $rec->{HANDLE}->autoflush(1);
12 $rec->{HANDLE}->print(" a string\n");
```

```
%TV = (
  flintstones => {
    series   => "flintstones",
    nights   => [ qw(monday thursday friday) ],
    members  => [
      { name => "fred", role => "lead", age => 36, },
      { name => "wilma", role => "wife", age => 31, },
      { name => "pebbles", role => "kid", age => 4, },
    ],
  },
  jetsons     => {
    series   => "jetsons",
    nights   => [ qw(wednesday saturday) ],
    members  => [
      { name => "george", role => "lead", age => 41, },
      { name => "jane", role => "wife", age => 39, },
      { name => "elroy", role => "kid", age => 9, },
    ],
  },
  simpsons    => {
```

```
1  # reading from file
2  # here's a piece by piece build up
3  $rec = {};
4  $rec->{series} = "flintstones";
5  $rec->{nights} = [ find_days() ];
6  @members = ();
7  # assume this file in field=value syntax
8  while (<>) {
9    %fields = split /[\s=]+/;
10   push @members, { %fields };
11 }
12 $rec->{members} = [ @members ];
13 # now remember the whole thing
14 $TV{ $rec->{series} } = $rec;
```

```
1 #
    ##############################################################
2 # now, you might want to make interesting extra
    fields that
3 # include pointers back into the same data
    structure so if
4 # change one piece, it changes everywhere, like
    for example
5 # if you wanted a {kids} field that was a
    reference
6 # to an array of the kids' records without having
    duplicate
7 # records and thus update problems.
8 #
    ##############################################################
```

```
1  foreach $family (keys %TV) {
2    $rec = $TV{$family}; # temp pointer
3    @kids = ();
4    for $person ( @{ $rec->{members} } ) {
5      if ($person->{role} =~ /kid|son|daughter
     /) {
6        push @kids, $person;
7      }
8    }
9    # REMEMBER: $rec and $TV{$family} point to
     same data!!
10   $rec->{kids} = [ @kids ];
11 }
```

```
1 # you copied the array, but the array itself
    contains pointers
2 # to uncopied objects. this means that if you
     make bart get
3 # older via
4 $TV{simpsons}{kids}[0]{age}++;
5 # then this would also change in
6 print $TV{simpsons}{members}[2]{age};
7 # because $TV{simpsons}{kids}[0] and $TV{
    simpsons}{members}[2]
8 # both point to the same underlying anonymous
     hash table
```

```
1  # print the whole thing
2  foreach $family ( keys %TV ) {
3    print "the $family";
4    print " is on during @{ $TV{$family}{nights} }\n
     ";
5    print "its members are:\n";
6    for $who ( @{ $TV{$family}{members} } ) {
7      print " $who->{name} ($who->{role}), age $who
     ->{age}\n";
8    }
9    print "it turns out that $TV{$family}{lead} has
     ";
10   print scalar ( @{ $TV{$family}{kids} } ), " kids
      named ";
11   print join (", ", map { $_->{name} } @{ $TV{
     $family}{kids} });
12   print "\n";
13 }
```

教学提纲

What you should deliver to the statistician

For maximum speed in the analysis this is the information you should pass to a statistician:

1. The raw data.
2. A tidy data set
3. A code book describing each variable and its values in the tidy data set.
4. An explicit and exact recipe you used to go from 1 → 2,3

知识拓展 | datasharing | raw data

It is critical that you include the rawest form of the data that you have access to. Here are some examples of the raw form of data:

- The strange binary file your measurement machine spits out
- The unformatted Excel file with 10 worksheets the company you contracted with sent you
- The complicated JSON data you got from scraping the Twitter API
- The hand-entered numbers you collected looking through a microscope

You know the raw data is in the right format if you:

1. Ran no software on the data
2. Did not manipulate any of the numbers in the data
3. You did not remove any data from the data set
4. You did not summarize the data in any way

If you did any manipulation of the data at all it is not the raw form of the data. Reporting manipulated data as raw data is a very common way to slow down the analysis process, since the analyst will often have to do a forensic study of your data to figure out why the raw data looks weird.

The four general principles you should pay attention to are:

1. Each **variable** you measure should be in one **column**
2. Each different **observation** of that variable should be in a different **row**
3. There should be one **table** for each "**kind**" of variable
4. If you have multiple tables, they should include a column in the table that allows them to be **linked**

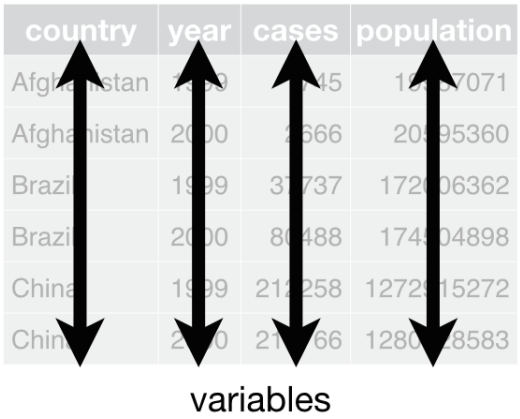
While these are the hard and fast rules, there are a number of other things that will make your data set much easier to handle. First is to include **a row** at the **top** of each data table/spreadsheet that contains **full variable names**. So if you measured age at diagnosis for patients, you would head that column with the name `AgeAtDiagnosis` instead of something like `ADx` or another abbreviation that may be hard for another person to understand.

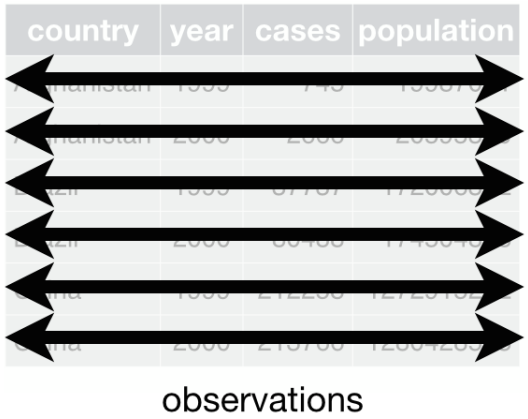
If you are sharing your data with the collaborator in Excel, the tidy data should be in one Excel file per table. They should not have multiple worksheets, no macros should be applied to the data, and no columns/ cells should be highlighted. Alternatively share the data in a CSV or TAB-delimited text file.

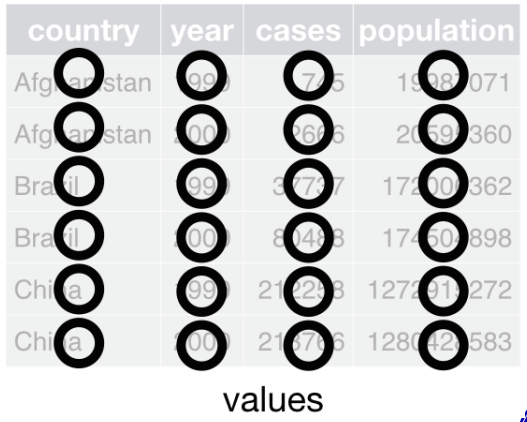
Here is an example of how this would work from genomics. Suppose that for 20 people you have collected gene expression measurements with RNA-sequencing. You have also collected demographic and clinical information about the patients including their age, treatment, and diagnosis. You would have one table/spreadsheet that contains the clinical/demographic information. It would have four columns (patient id, age, treatment, diagnosis) and 21 rows (a row with variable names, then one row for every patient). You would also have one spreadsheet for the summarized genomic data. Usually this type of data is summarized at the level of the number of counts per exon. Suppose you have 100,000 exons, then you would have a table/ spreadsheet that had 21 rows (a row for gene names, and one row for each patient) and 100,001 columns (one column for patient ids and one column for each data type).

知识拓展 | datasharing | tidy data

1. Each variable forms a column (Each variable in the data set is placed in its own column)
2. Each observation forms a row (Each observation is placed in its own row)
3. Each value is placed in its own cell
4. Each type of observational unit forms a table







	treatmenta	treatmentb
John Smith	—	2
Jane Doe	16	11
Mary Johnson	3	1

	John Smith	Jane Doe	Mary Johnson
treatmenta	—	16	3
treatmentb	2	11	1

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

country	year	m014	m1524	m2534	m3544	m4554	m5564	m65	mu	f014
AD	2000	0	0	1	0	0	0	0	—	—
AE	2000	2	4	4	6	5	12	10	—	3
AF	2000	52	228	183	149	129	94	80	—	93
AG	2000	0	0	0	0	0	0	1	—	1
AL	2000	2	19	21	14	24	19	16	—	3
AM	2000	2	152	130	131	63	26	21	—	1
AN	2000	0	0	1	2	0	0	0	—	0
AO	2000	186	999	1003	912	482	312	194	—	247
AR	2000	97	278	594	402	419	368	330	—	121
AS	2000	—	—	—	—	1	1	—	—	—

country	year	column	cases
AD	2000	m014	0
AD	2000	m1524	0
AD	2000	m2534	1
AD	2000	m3544	0
AD	2000	m4554	0
AD	2000	m5564	0
AD	2000	m65	0
AE	2000	m014	2
AE	2000	m1524	4
AE	2000	m2534	4
AE	2000	m3544	6
AE	2000	m4554	5
AE	2000	m5564	12
AE	2000	m65	10
AE	2000	f014	3

(a) Molten data

country	year	sex	age	cases
AD	2000	m	0-14	0
AD	2000	m	15-24	0
AD	2000	m	25-34	1
AD	2000	m	35-44	0
AD	2000	m	45-54	0
AD	2000	m	55-64	0
AD	2000	m	65+	0
AE	2000	m	0-14	2
AE	2000	m	15-24	4
AE	2000	m	25-34	4
AE	2000	m	35-44	6
AE	2000	m	45-54	5
AE	2000	m	55-64	12
AE	2000	m	65+	10
AE	2000	f	0-14	3

(b) Tidy data

知识拓展 | datasharing | cook book

For almost any data set, the measurements you calculate will need to be described in more detail than you will sneak into the spreadsheet. The code book contains this information. At minimum it should contain:

1. Information about the variables (including units!) in the data set not contained in the tidy data
2. Information about the summary choices you made
3. Information about the experimental study design you used

A common format for this document is a Word (or Markdown) file. There should be a section called “Study design” that has a thorough description of how you collected the data. There is a section called “Code book” that describes each variable and its units.

In our genomics example, the analyst would want to know what the unit of measurement for each clinical/demographic variable is (age in years, treatment by name/dose, level of diagnosis and how heterogeneous). They would also want to know how you picked the exons you used for summarizing the genomic data (UCSC/Ensembl, etc.). They would also want to know any other information about how you did the data collection/study design. For example, are these the first 20 patients that walked into the clinic? Are they 20 highly selected patients by some characteristic like age? Are they randomized to treatments?

知识拓展 | datasharing | code variables

When you put variables into a spreadsheet there are several main categories you will run into depending on their data type:

1. Continuous
2. Ordinal
3. Categorical
4. Missing
5. Censored

Continuous variables are anything measured on a quantitative scale that could be any fractional number. An example would be something like weight measured in kg. Ordinal data are data that have a fixed, small (< 100) number of levels but are ordered. This could be for example survey responses where the choices are: poor, fair, good. Categorical data are data where there are multiple categories, but they aren't ordered. One example would be sex: male or female. Missing data are data that are missing and you don't know the mechanism. You should code missing values as `NA`. Censored data/make up/ throw away missing observations.

In general, try to avoid coding categorical or ordinal variables as numbers. When you enter the value for sex in the tidy data, it should be ”male” or ”female”. The ordinal values in the data set should be ”poor”, ”fair”, and ”good” not 1, 2 ,3. This will avoid potential mixups about which direction effects go and will help identify coding errors.

Always encode every piece of information about your observations using **text**. For example, if you are storing data in Excel and use a form of colored text or cell background formatting to indicate information about an observation (”red variable entries were observed in experiment 1.”) then this information will not be exported (and will be lost!) when the data is exported as raw text. Every piece of data should be encoded as actual text that can be exported.

You may have heard this before, but reproducibility is kind of a big deal in computational science. That means, when you submit your paper, the reviewers and the rest of the world should be able to exactly replicate the analyses from raw data all the way to final results. If you are trying to be efficient, you will likely perform some summarization/ data analysis steps before the data can be considered tidy.

The ideal thing for you to do when performing summarization is to create a computer script (in `R`, `Python`, or something else) that takes the raw data as input and produces the tidy data you are sharing as output. You can try running your script a couple of times and see if the code produces the same output.

知识拓展 | datasharing | instruction list/script

In many cases, the person who collected the data has incentive to make it tidy for a statistician to speed the process of collaboration. They may not know how to code in a scripting language. In that case, what you should provide the statistician is something called pseudocode. It should look something like:

1. Step 1 - take the raw file, run version 3.1.2 of summarize software with parameters a=1, b=2, c=3
2. Step 2 - run the software separately for each sample
3. Step 3 - take column three of outputfile.out for each sample and that is the corresponding row in the output data set

You should also include information about which **system** (Mac/ Windows/Linux) you used the software on and whether you tried it more than once to confirm it gave the same results. Ideally, you will run this by a fellow student/labmate to confirm that they can obtain the same output file you did.

When you turn over a properly tidied data set it dramatically decreases the workload on the statistician. So hopefully they will get back to you much sooner. But most careful statisticians will check your recipe, ask questions about steps you performed, and try to confirm that they can obtain the same tidy data that you did with, at minimum, spot checks.

知识拓展 | datasharing | expect from the analyst

You should then expect from the statistician:

1. An analysis script that performs each of the analyses (not just instructions)
2. The exact computer code they used to run the analysis
3. All output files/figures they generated.

This is the information you will use in the supplement to establish reproducibility and precision of your results. Each of the steps in the analysis should be clearly explained and you should ask questions when you don't understand what the analyst did. It is the responsibility of both the statistician and the scientist to understand the statistical analysis. You may not be able to perform the exact analyses without the statistician's code, but you should be able to explain why the statistician performed each step to a labmate/your principal investigator.

教学提纲

知识拓展 | data | wide vs. long

wide-format

Wide data has a column for each variable.

long-format

Long-format data has a column for possible variable types and a column for the values of those variables.

wide vs. long

- It turns out that you need wide-format data for some types of data analysis and long-format data for others.
- In reality, you need long-format data much more commonly than wide-format data.
- But people often find it easier to record their data in wide format.

知识拓展 | data | wide vs. long

wide-format

Wide data has a column for each variable.

long-format

Long-format data has a column for possible variable types and a column for the values of those variables.

wide vs. long

- It turns out that you need wide-format data for some types of data analysis and long-format data for others.
- In reality, you need long-format data much more commonly than wide-format data.
- But people often find it easier to record their data in wide format.

知识拓展 | data | wide vs. long

wide-format

Wide data has a column for each variable.

long-format

Long-format data has a column for possible variable types and a column for the values of those variables.

wide vs. long

- It turns out that you need wide-format data for some types of data analysis and long-format data for others.
- In reality, you need long-format data much more commonly than wide-format data.
- But people often find it easier to record their data in wide format.

知识拓展 | data | wide vs. long

subject	sex	control	cond1	cond2
1	M	7.9	12.3	10.7
2	F	6.3	10.6	11.1
3	F	9.5	13.1	13.8
4	M	11.5	13.4	12.9

subject	sex	condition	measurement
1	M	control	7.9
1	M	cond1	12.3
1	M	cond2	10.7
2	F	control	6.3
2	F	cond1	10.6
2	F	cond2	11.1
3	F	control	9.5
3	F	cond1	13.1
3	F	cond2	13.8
4	M	control	11.5
4	M	cond1	13.4
4	M	cond2	12.9

知识拓展 | data | wide vs. long

gene	A1	A2	B1	B2
GENE_1	35.41	36.60	29.96	29.73
GENE_2	36.60	23.45	24.39	24.74
GENE_3	29.96	23.30	32.17	25.94
GENE_4	29.73	22.84	31.66	26.22
GENE_5	34.46	22.79	31.39	24.75
GENE_6	35.66	21.37	31.34	24.72
GENE_7	33.28	21.74	31.10	25.39
GENE_8	33.03	22.96	30.90	25.65
GENE_9	26.58	22.87	31.14	30.57
GENE_10	26.05	25.18	31.03	29.99
CONTROL	26.60	25.60	26.03	25.79

gene	sample_id	ct_value
GENE_1	A1	35.41
GENE_2	A1	36.60
GENE_3	A1	29.96
GENE_4	A1	29.73
GENE_5	A1	34.46
GENE_6	A1	35.66
GENE_7	A1	33.28
GENE_8	A1	33.03
GENE_9	A1	26.58
GENE_10	A1	26.05
CONTROL	A1	26.60
GENE_1	A2	36.60
GENE_2	A2	23.45
GENE_3	A2	23.30

知识拓展 | data | wide vs. long

dcast formula `dcast(aql, month + day ~ variable, value.var = "value")`

ID variables (left side of formula)	Variable to swing into column names (right side of formula)	Values (value.var)

Long-format data

month	day	variable	value
5	1	ozone	41
5	2	ozone	36
5	3	ozone	12
5	4	ozone	18
5	5	ozone	NA
5	6	ozone	28

Wide-format data

month	day	ozone	solar.r	wind	temp
5	1	41	190	7.4	67
5	2	36	118	8.0	72
5	3	12	149	12.6	74
5	4	18	313	11.5	62
5	5	NA	NA	14.3	56
5	6	28	NA	14.9	66

知识拓展 | data | wide2long

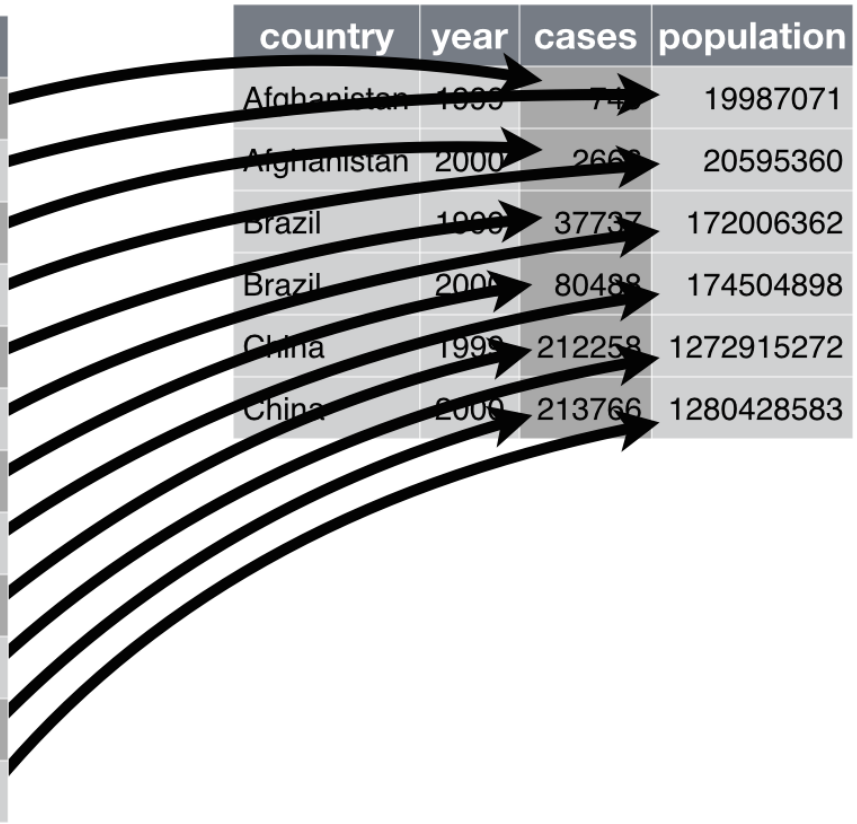
country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

知识拓展 | data | long2wide

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

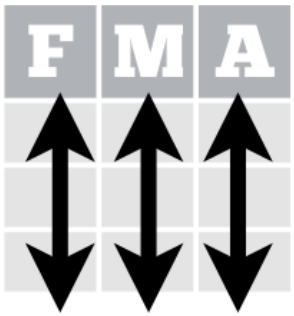


country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

教学提纲

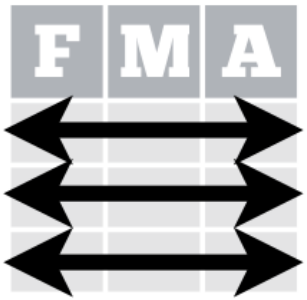
知识拓展 | example | Tidy data

In a tidy data set:



Each **variable** is saved in its own **column**

&

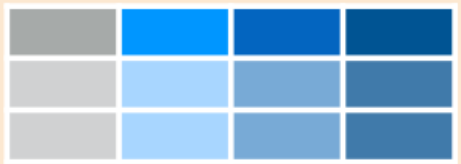


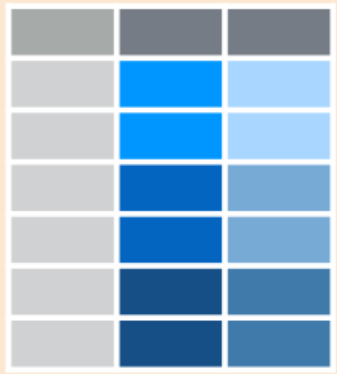
Each **observation** is saved in its own **row**

知识拓展 | example | wide vs. long

	id	name	e6	e7	e8	e9	final
1	2013052101	连明	86	82	83	85	88
2	2013052102	李秉宣	88	85	83	86	77
3	2013052103	何雨泽	84	85	84	85	79
4	2013052104	陈泽川	80	89	84	85	60
5	2013052105	张韬	85	85	82	84	90
6	2013052106	李发金	89	85	88	89	94
7	2013052107	杨屹顶	83	85	80	81	75
8	2013052109	高奕	84	87	81	82	70
9	2013052110	孙少省	91	89	85	86	69
10	2013052112	王晨	78	78	75	75	52
11	2013052113	高杨	79	77	75	75	49
12	2013052114	李嘉良	76	76	75	75	48
13	2013052115	刘晓宇	81	82	80	80	85
14	2013052116	赵天晨	89	88	86	85	91
15	2013052118	彭永林	90	89	89	88	84
16	2013052119	江姝婷	91	94	88	89	95
17	2013052120	朱依青	93	95	92	90	79
18	2013052121	韦秋霞	88	83	86	87	91
19	2013052122	陆玉妹	91	92	89	95	86
20	2013052123	钱碧平	86	84	82	83	71

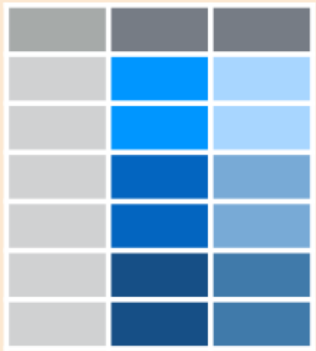
	id	name	class	score
1	2013052101	连明	e6	86
2	2013052102	李秉宣	e6	88
3	2013052103	何雨泽	e6	84
4	2013052104	陈泽川	e6	80
5	2013052105	张韬	e6	85
6	2013052106	李发金	e6	89
7	2013052107	杨屹顶	e6	83
8	2013052109	高奕	e6	84
9	2013052110	孙少省	e6	91
10	2013052112	王晨	e6	78
11	2013052113	高杨	e6	79
12	2013052114	李嘉良	e6	76
13	2013052115	刘晓宇	e6	81
14	2013052116	赵天晨	e6	89
15	2013052118	彭永林	e6	90
16	2013052119	江姝婷	e6	91
17	2013052120	朱依青	e6	93
18	2013052121	韦秋霞	e6	88
19	2013052122	陆玉妹	e6	91
20	2013052123	钱碧平	e6	86

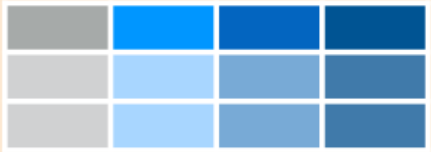




tidyr::**gather(cases, "year", "n", 2:4)**

Gather columns into rows.





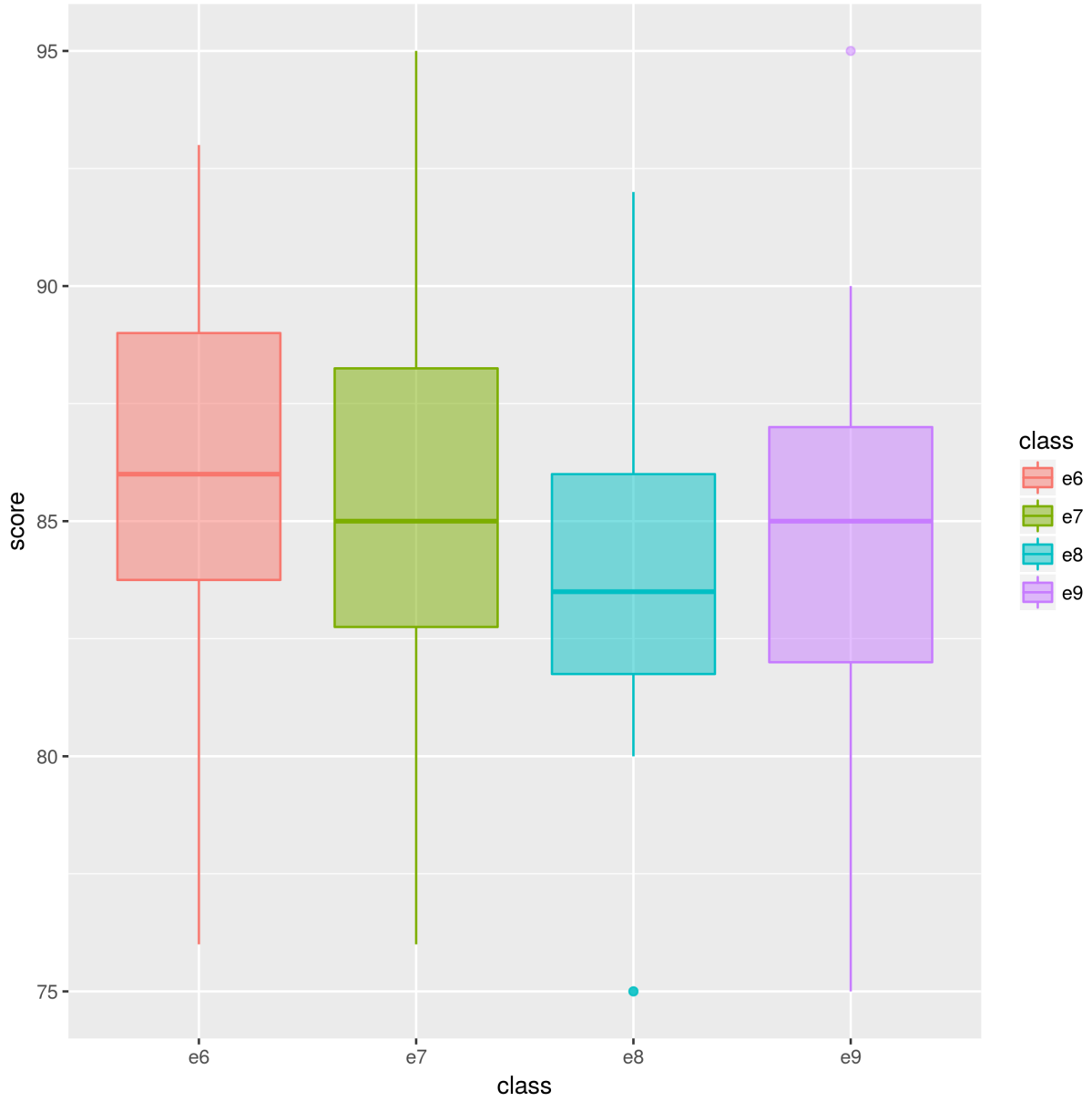
tidyr::**spread(pollution, size, amount)**

Spread rows into columns.

```
1  library(readr)
2  library(tidyr)
3  library(dplyr)
4  library(stringr)
5  library(ggplot2)
6  library(ComplexHeatmap)
7  library(circlize)
8
9  score_wide <- read_csv("score_2013.csv")
10 score_long <- gather(score_wide, "class", "
     score", 3:7)
11
12 score_long <- score_long %>% mutate(id2=as.
     factor(str_replace(id, "20130521", "")))
```

```
1 score_long2 <- score_long %>% filter(str_
    detect(class, "e"))
2
3 gb <- ggplot(score_long2, aes(class, score))
4 gb <- gb + geom_boxplot(aes(fill=class, color
    =class), alpha=0.5)
5
6 ggsave("exp_boxplot.png", gb)
```

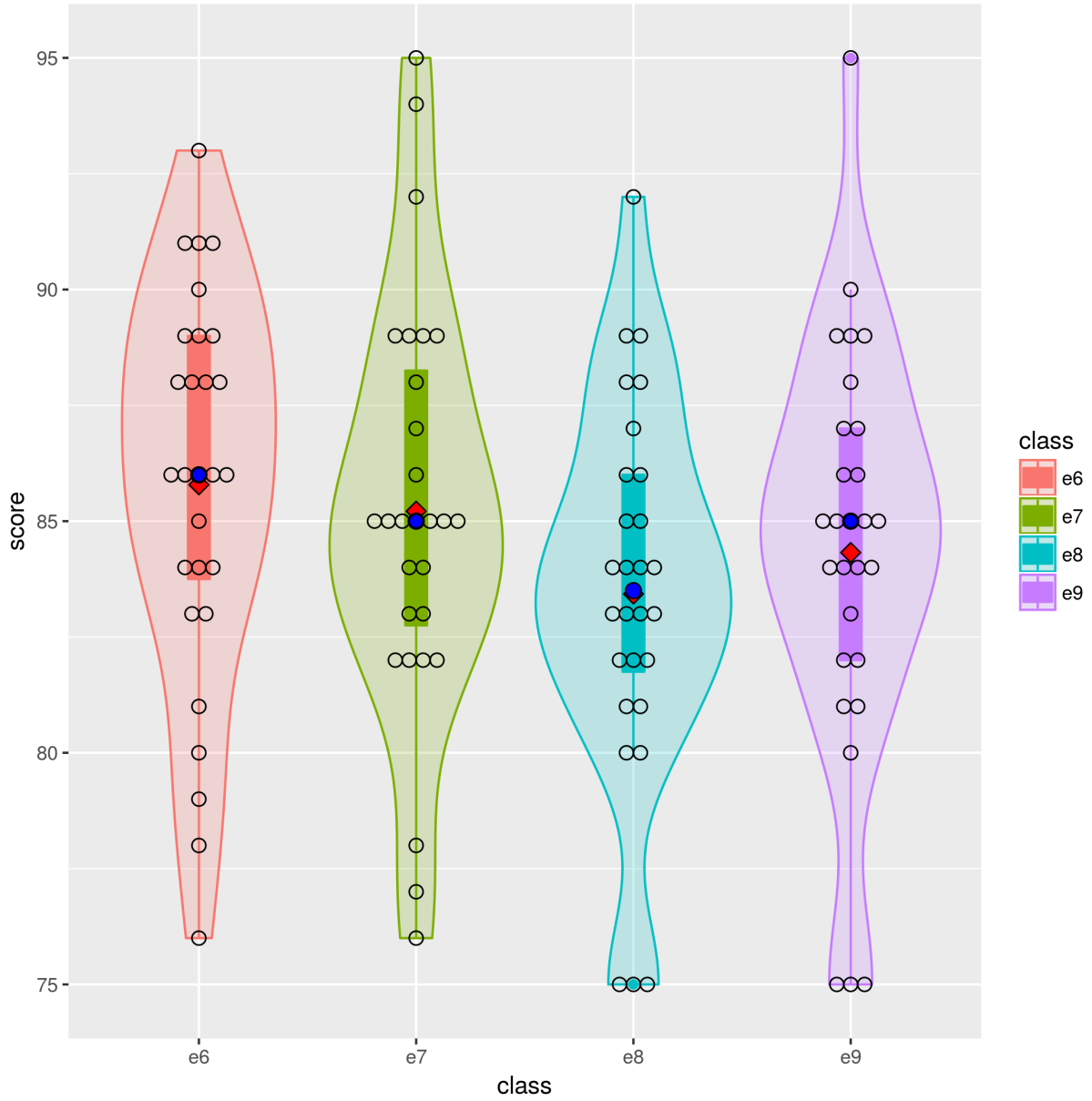
知识拓展 | example | R | plot | boxplot



知识拓展 | example | R | violin

```
1 gv <- ggplot(score_long2, aes(class, score))
2 gv <- gv + geom_violin(aes(fill=class, color=
    class), alpha=0.2)
3 gv <- gv + geom_boxplot(aes(fill=class, color
    =class), width=0.1)
4 gv <- gv + stat_summary(fun.y="mean", geom="
    point", shape=23, size=3, fill="red")
5 gv <- gv + stat_summary(fun.y="median", geom=
    "point", shape=21, size=3, fill="blue")
6 gv <- gv + geom_dotplot(binaxis="y", binwidth
    =0.3, stackdir="center", fill=NA)
7
8 ggsave("exp_violin.png", gv)
```

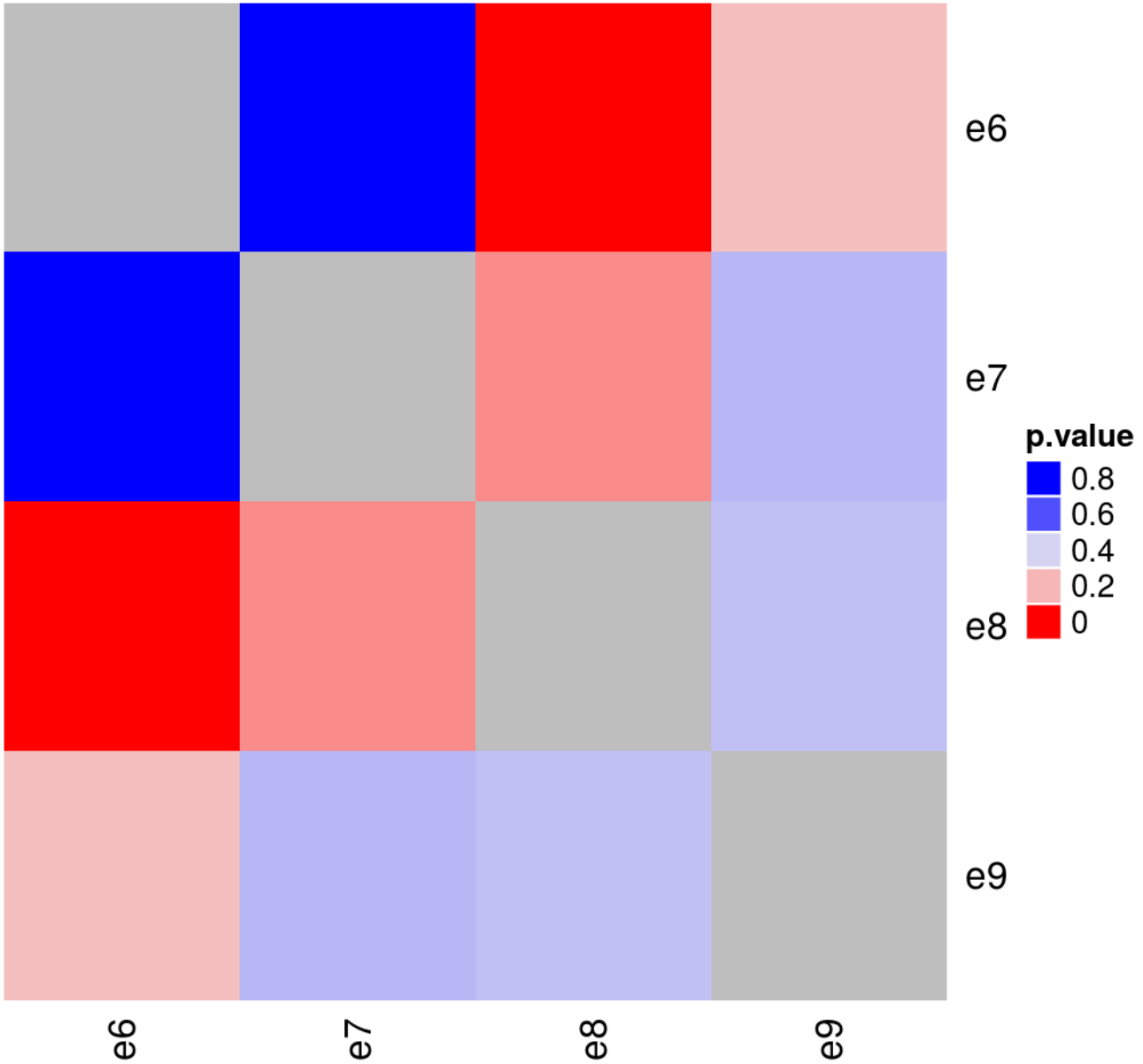
知识拓展 | example | R | plot | violin



知识拓展 | example | R | heatmap

```
1  tri2mat <- function(tp, pf) {
2    tp <- cbind(tp, rep(NA, nrow(tp)))
3    tp <- rbind(rep(NA, ncol(tp)), tp)
4    colnames(tp) <- pf; rownames(tp) <- pf
5    tp[upper.tri(tp)] <- t(tp)[upper.tri(tp)]
6    tp
7  }
8
9  t_test <- pairwise.t.test(score_long2$score, score_long2
     $class, p.adjust.method="none", pool.sd=FALSE)
10 tp <- t_test$p.value
11 pf <- levels(as.factor(score_long2$class))
12 tm <- tri2mat(tp, pf)
13 col = colorRamp2(seq(min(tm, na.rm=TRUE), max(tm, na.rm=
     TRUE), length=3), c("red", "#EEEEEE", "blue"), space="
     RGB")
14 png("exp_heatmap.png", width=1024, height=960, res=200)
15 Heatmap(tm, col=col, heatmap_legend_param=list(title="p.
     value"), cluster_rows=FALSE, cluster_columns=FALSE)
16 dev.off()
```

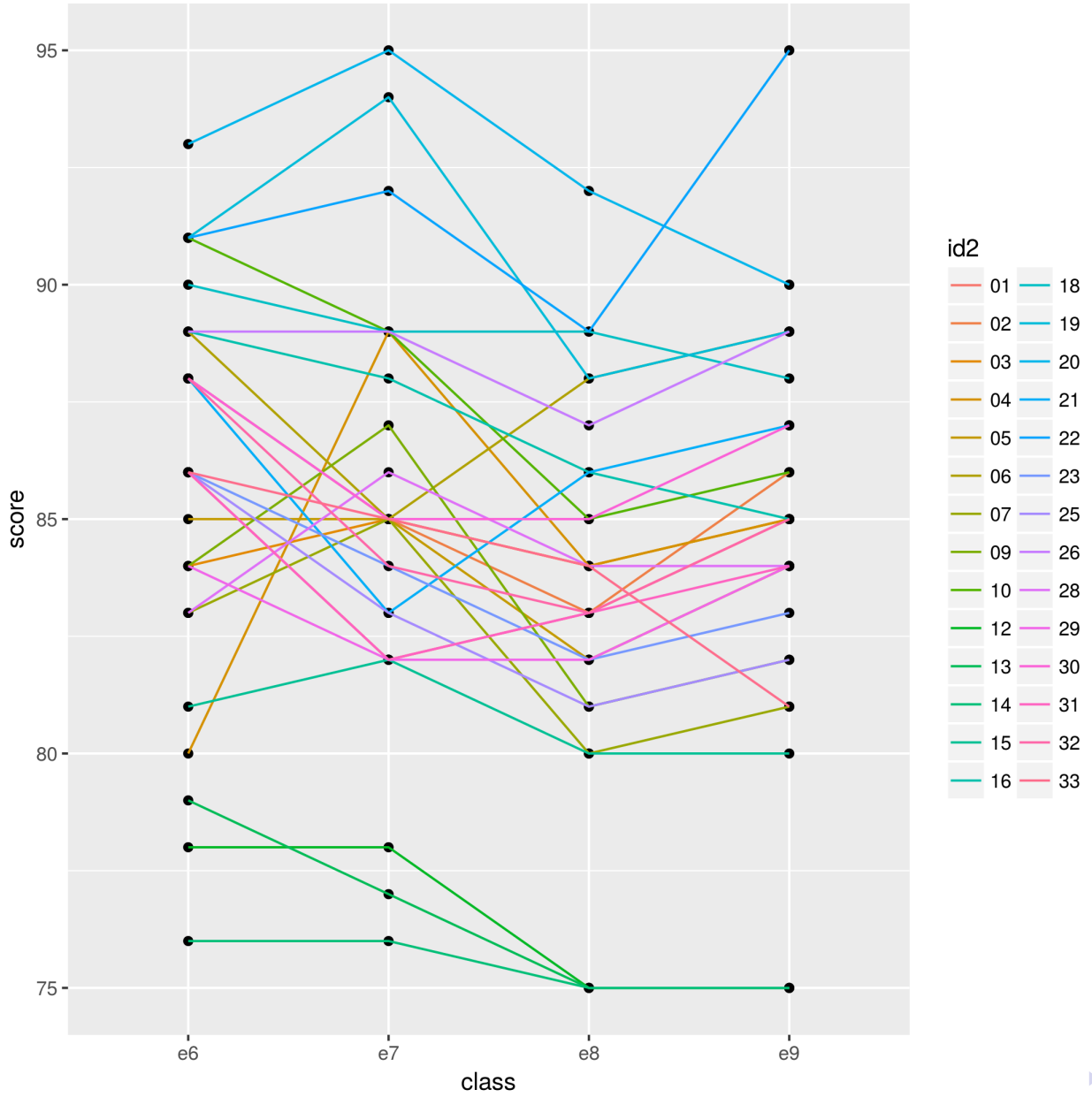
知识拓展 | example | R | plot | heatmap



知识拓展 | example | R | line

```
1 gl <- ggplot(score_long2, aes(class, score,
    group=id2))
2 gl <- gl + geom_point()
3 gl <- gl + geom_line(aes(color=id2))
4
5 ggsave("id_lineplot.png", gl)
```

知识拓展 | example | R | plot | line



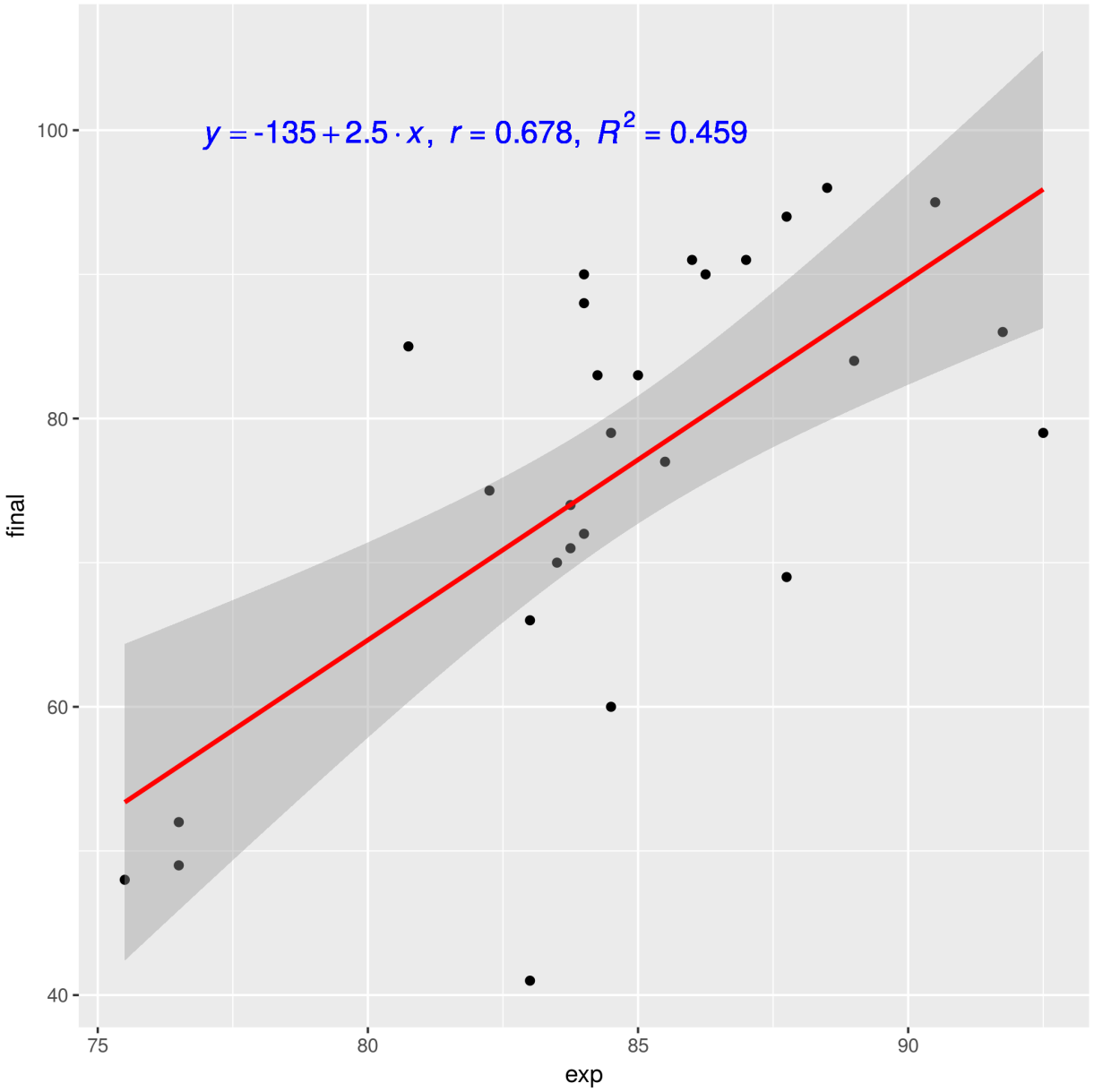
知识拓展 | example | R | correlation(1/3)

```
1 lm_eqn <- function(df) {
2   m <- lm(final ~ exp, df)
3   eq <- substitute(italic(y) == a + b %.%
    italic(x) *","* ~~italic(r)~"="~r1 *","* ~~
    italic(R)^2~"="~r2,
4                    list(a = format(coef(m)
    [1], digits=2),
5                         b = format(coef(m)
    [2], digits=2),
6                         r1 = format(sqrt(
    summary(m)$r.squared), digits=3),
7                         r2 = format(summary(m
    )$r.squared, digits=3)))
8   as.character(as.expression(eq))
9 }
```

知识拓展 | example | R | correlation(2/3)

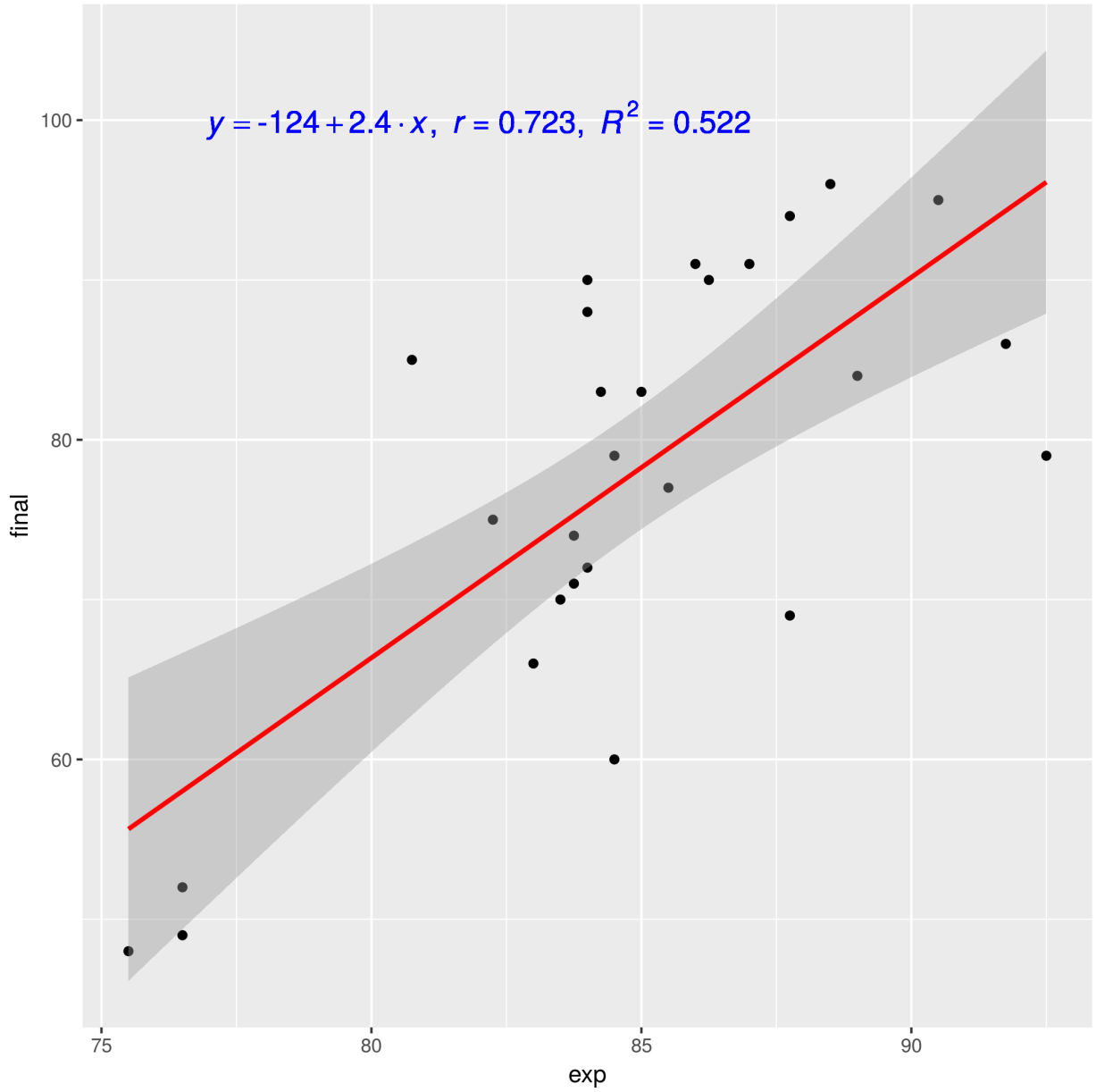
```
1  plot_cor <- function (df){
2    gr <- ggplot(df, aes(exp, final))
3    gr <- gr + geom_point()
4    gr <- gr + geom_smooth(method="lm", color="
     red", formula=y~x)
5    gr <- gr + geom_text(x=82, y=100, label=lm_
     eqn(df), parse=TRUE, size=5, color="blue")
6  }
7
8  score_wide2 <- score_wide %>% group_by(id,
     name,final) %>% summarise(exp=mean(c(e6,e7,
     e8,e9)))
9  gr <- plot_cor(score_wide2)
10 ggsave("correlation_plot.png", gr)
```

知识拓展 | example | R | plot | correlation(1/2)



```
1 id1 <- score_wide2[which.min(score_wide2$
    final),]$id
2 score_wide3 <- score_wide2 %>% filter(id!=id1
    )
3
4 gr2 <- plot_cor(score_wide3)
5
6 ggsave("correlation_plot_correct.png", gr2)
```

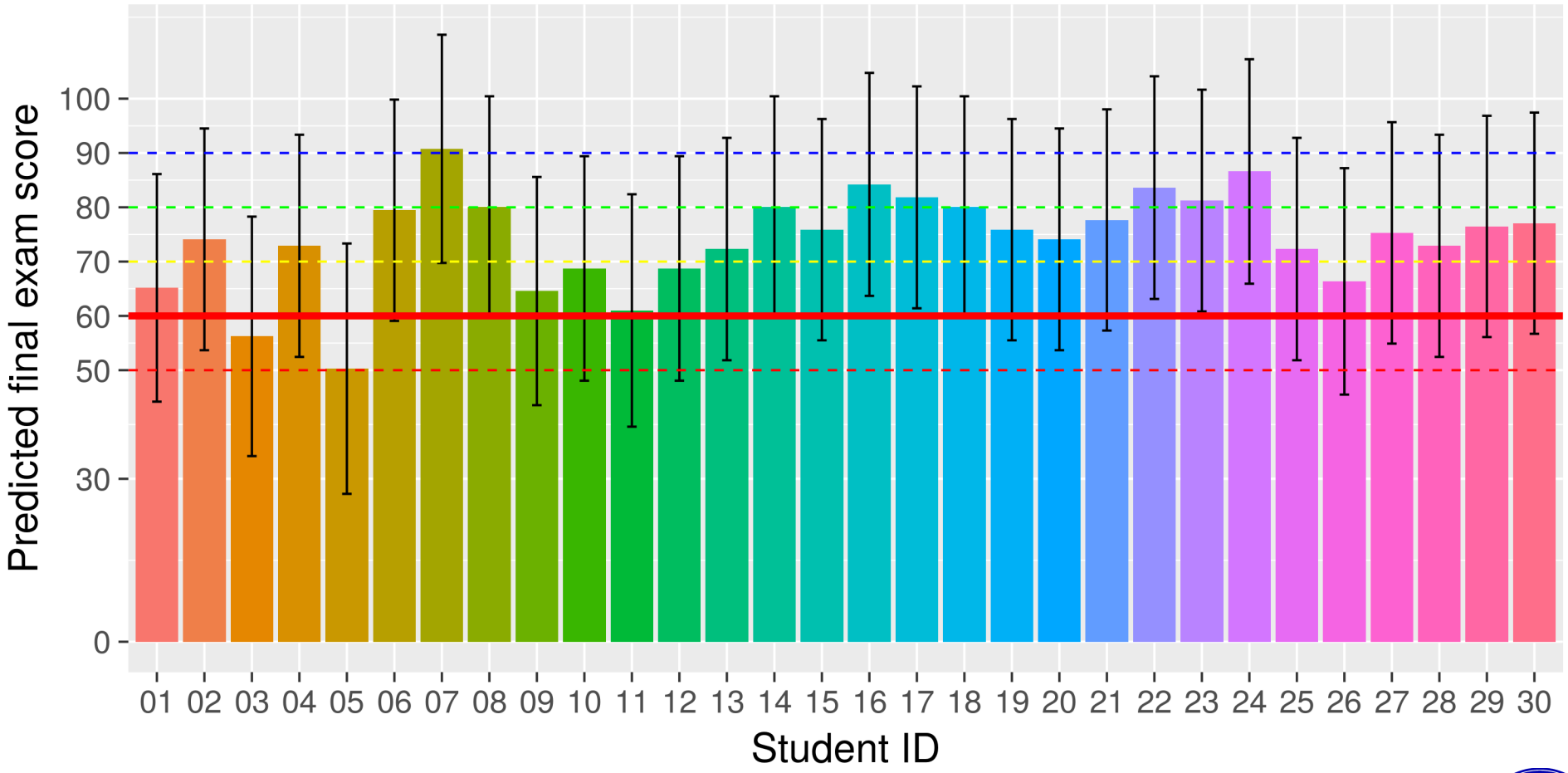
知识拓展 | example | R | plot | correlation(2/2)



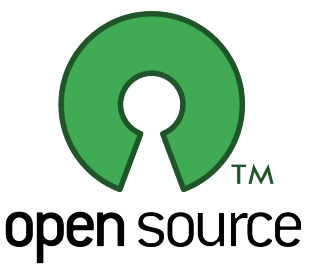
知识拓展 | example | R | bar

```
1  m <- lm(final ~ exp, score_wide3)
2  score2014 <- read_csv("score_2014.csv")
3  mp <- predict(m, score2014, interval="prediction")
4  s4 <- bind_cols(score2014, as.data.frame(mp))
5  colnames(s4) <- c("ID", "name", "exp", "final", "fl", "fu")
6  s4 <- s4 %>% mutate(id=as.factor(str_replace(ID, "20140521", "")
     ))
7
8  g <- ggplot(s4, aes(x=id, y=final, fill=id))
9  g <- g + theme_gray(base_size=18)
10 g <- g + geom_bar(stat="identity")
11 g <- g + geom_errorbar(aes(ymin=fl, ymax=fu), width=0.2)
12 g <- g + geom_hline(yintercept=60, color="red", size=1.5)
13 g <- g + geom_hline(yintercept=c(50, 70, 80, 90), color=c("red",
     "yellow", "green", "blue"), size=0.5, linetype=2)
14 g <- g + scale_y_continuous(breaks=c(0, 30, seq(50,100,10)))
15 g <- g + labs(x="Student ID", y="Predicted final exam score")
16 g <- g + guides(fill=FALSE)
17 ggsave("prediction_2014.png", g, width=10, height=5)
```

知识拓展 | example | R | plot | barplot



Powered by



TEX LATEX XETEX Beamer