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

第四章　序列和字符串

伊现富（Yi Xianfu）

天津医科大学（TIJMU）
生物医学工程与技术学院

2016 年 11 月

教学提纲

教学提纲

Perl 语言基础

- 标量变量和数组变量
- 字符串操作（替换、翻译等）
- 从文件中读取数据

DNA 和蛋白质生物序列数据的处理

- 把 DNA 片段拼接起来
- 把 DNA 转录成 RNA
- 获取反向互补序列
- 从文件中读取序列
- 获取序列信息（碱基数目、GC 含量）

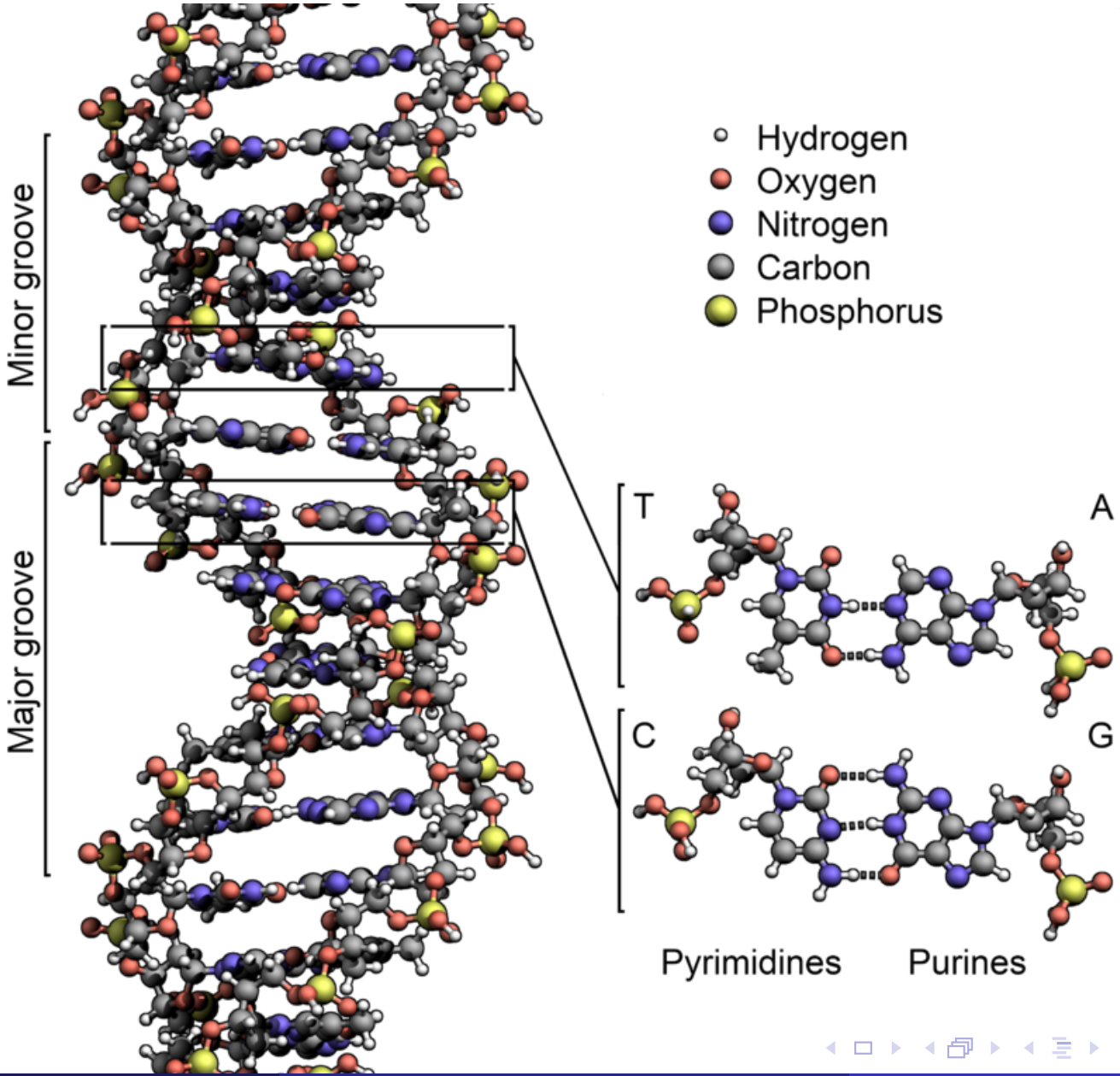
Perl 语言基础

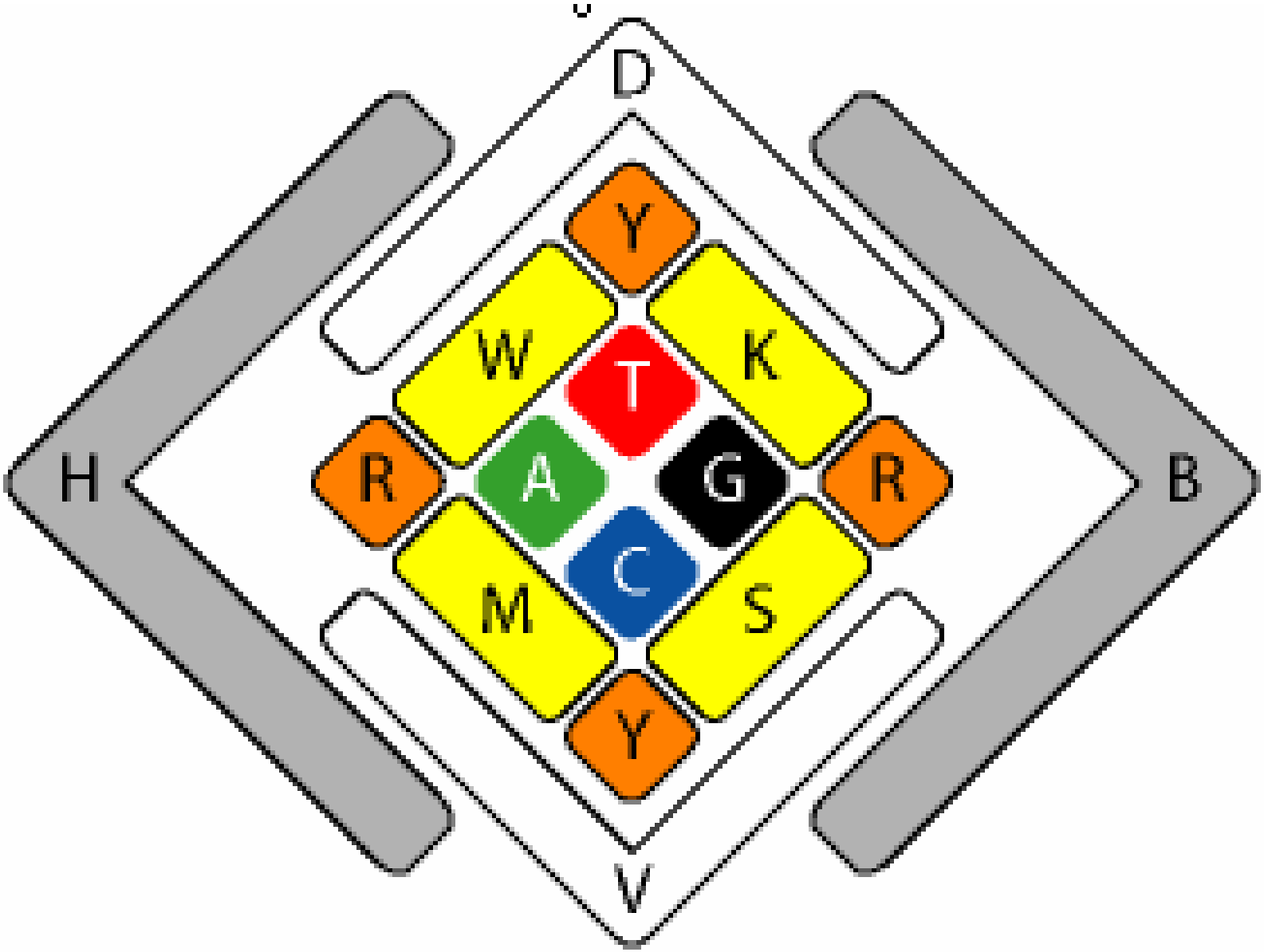
- 标量变量和数组变量
- 字符串操作（替换、翻译等）
- 从文件中读取数据

DNA 和蛋白质生物序列数据的处理

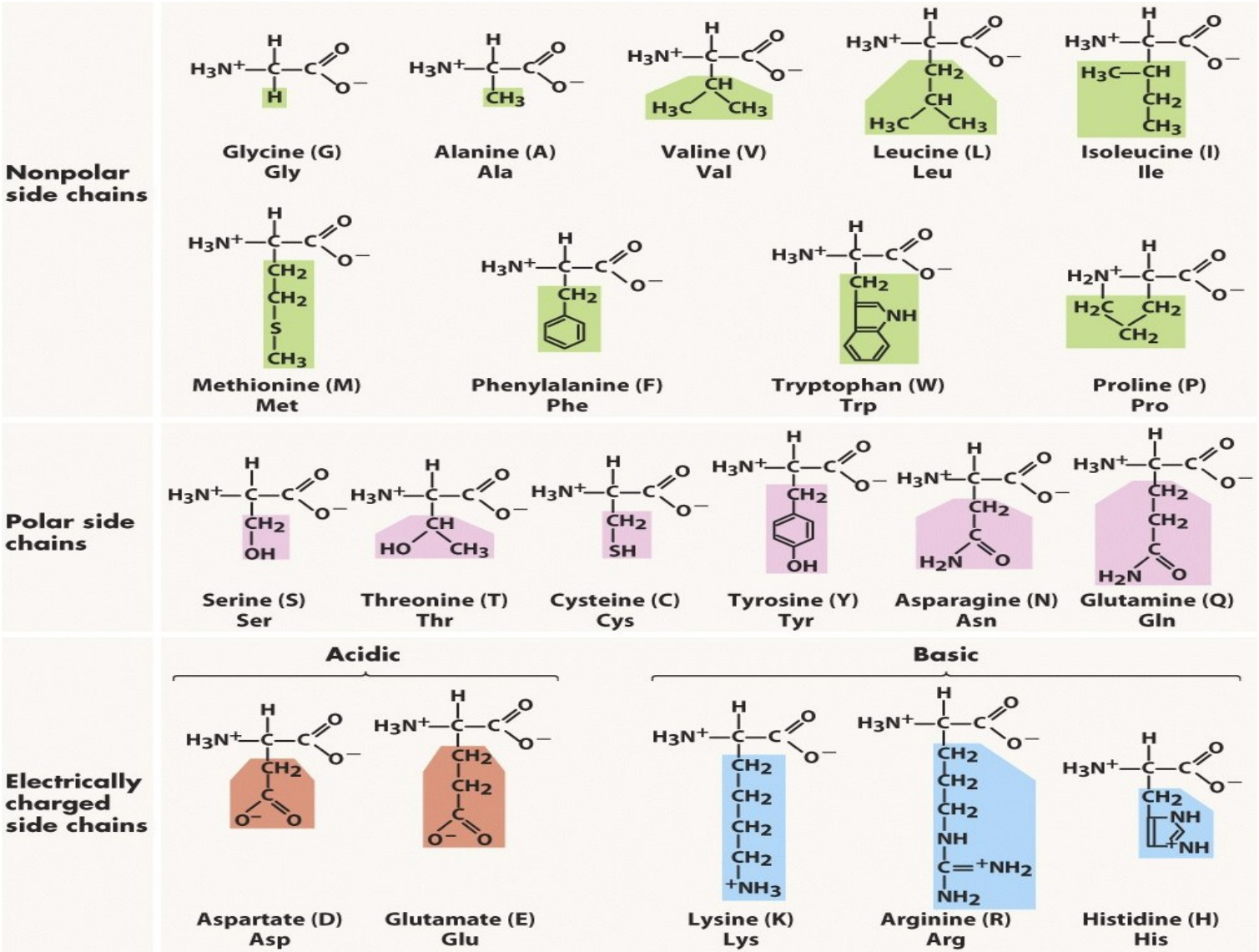
- 把 DNA 片段拼接起来
- 把 DNA 转录成 RNA
- 获取反向互补序列
- 从文件中读取序列
- 获取序列信息（碱基数目、GC 含量）

教学提纲


Minor groove
Major groove
Hydrogen
Oxygen
Nitrogen
Carbon
Phosphorus
T
A
C
G
Pyrimidines
Purines


D
H
B
V
Y
W
T
K
R
A
G
R
C
M
S
Y

Code	Represents	Complement
A	Adenine	T
G	Guanine	C
C	Cytosine	G
T	Thymine	A
Y	Pyrimidine (C or T)	R
R	Purine (A or G)	Y
W	weak (A or T)	W
S	strong (G or C)	S
K	keto (T or G)	M
M	amino (C or A)	K
D	A, G, T (not C)	H
V	A, C, G (not T)	B
H	A, C, T (not G)	D
B	C, G, T (not A)	V
X/N	any base	X/N
-	Gap	-


Nonpolar side chains
Glycine (G) Gly
Alanine (A) Ala
Valine (V) Val
Leucine (L) Leu
Isoleucine (I) Ile
Methionine (M) Met
Phenylalanine (F) Phe
Tryptophan (W) Trp
Proline (P) Pro
Polar side chains
Serine (S) Ser
Threonine (T) Thr
Cysteine (C) Cys
Tyrosine (Y) Tyr
Asparagine (N) Asn
Glutamine (Q) Gln
Electrically charged side chains
Acidic
Basic
Aspartate (D) Asp
Glutamate (E) Glu
Lysine (K) Lys
Arginine (R) Arg
Histidine (H) His

字符串

字符串（string），是由零个或多个字符组成的有限序列，是编程语言中表示文本的数据类型。

字符串操作

- 通常以串的整体作为操作对象，如：在串中查找某个子串、求取一个子串、在串的某个位置上插入一个子串以及删除一个子串等。
- 两个字符串相等的充要条件是：长度相等，并且各个对应位置上的字符都相等。
- 设 p、q 是两个串，求 q 在 p 中首次出现的位置的运算叫做模式匹配。
- 一个简单的字符串操作是“拼接”：也就是说先写一个字符串 S，随后在后面再写一个 T 得到 ST 这样一个过程。
- 其它的常见操作包括在一个长字符串中搜索一个子串，排列一组字符串以及分析一个字符串。

字符串

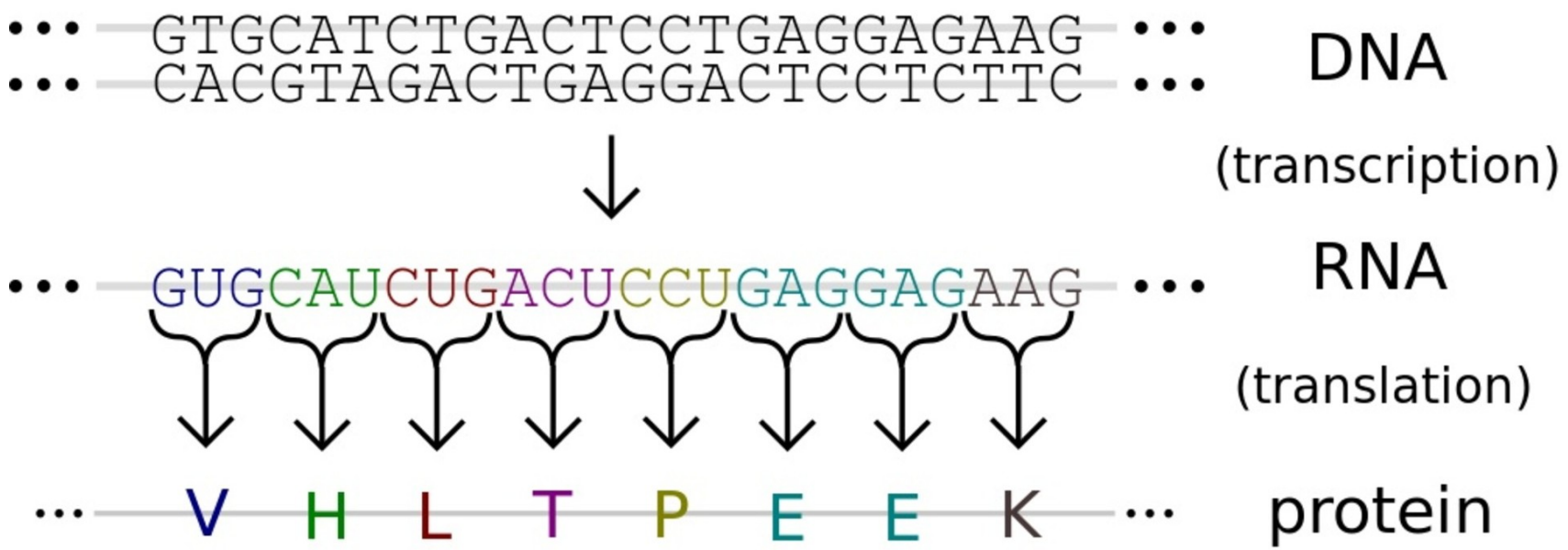
字符串（string），是由零个或多个字符组成的有限序列，是编程语言中表示文本的数据类型。

字符串操作

- 通常以串的整体作为操作对象，如：在串中查找某个子串、求取一个子串、在串的某个位置上插入一个子串以及删除一个子串等。
- 两个字符串相等的充要条件是：长度相等，并且各个对应位置上的字符都相等。
- 设 p、q 是两个串，求 q 在 p 中首次出现的位置的运算叫做模式匹配。
- 一个简单的字符串操作是“拼接”：也就是说先写一个字符串 S，随后在后面再写一个 T 得到 ST 这样一个过程。
- 其它的常见操作包括在一个长字符串中搜索一个子串，排列一组字符串以及分析一个字符串。

序列与字符串（问题转换）

生物信息学：（生物学）DNA/RNA/蛋白质序列 $\Longrightarrow$ 字符串（计算机科学）



教学提纲

教学提纲

```
1  #!/usr/bin/perl -w
2  # Example 4-1   Storing DNA in a variable,
     and printing it out
3
4  # First we store the DNA in a variable called
     $DNA
5  $DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
6
7  # Next, we print the DNA onto the screen
8  print $DNA;
9
10 # Finally, we'll specifically tell the
     program to exit.
11 exit;
```

- 在 Perl 中，变量就是要处理的数据的名称，使用该名称，你可以对数据进行完全的访问。
- 变量起名要清晰易懂；既然是存储 DNA 序列，起名为 `$DNA` 再自然不过了（或者 `$dna`、`$dna_seq`……）。
- 保存程序时一定要保存为 ASCII 或者纯文本格式。
- 运行程序：`perl example4-1.pl`（或者：`chmod 755 example4-1.pl; ./example4-1.pl`）。

教学提纲

所谓控制流，就是计算机是以什么顺序来执行程序中的语句的。

所有的程序都是从第一行开始执行，除非明确指明了其他的运行顺序，否则它将一条一条地按照顺序执行语句，直到程序的最后一行。

可以通过条件流程控制语句（if 等）、循环流程控制语句（while 等）等控制程序的执行顺序。

教学提纲

- 添加空行使程序更加易读。
- 以 # 起始进行注释。
- Perl 程序运行时，会把空行和注释忽略掉。
- 注释内容：程序的用途、作者及相关信息，代码每一部分的作用，代码的工作原理，……

没有注释的、赤裸裸的、完全等价的程序

```
1 #!/usr/bin/perl -w
2 $DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
3 print $DNA;
4 exit;
```

- 添加空行使程序更加易读。
- 以 # 起始进行注释。
- Perl 程序运行时，会把空行和注释忽略掉。
- 注释内容：程序的用途、作者及相关信息，代码每一部分的作用，代码的工作原理，……

没有注释的、赤裸裸的、完全等价的程序

```
1 #!/usr/bin/perl -w
2 $DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
3 print $DNA;
4 exit;
```

教学提纲

```
1 #!/usr/bin/perl -w
```

命令解释

- 像是注释，但并不是注释。
- 告诉 Unix/Linux 计算机，这是一个 Perl 程序。
- 本质上是 Perl 语言解释器在文件系统中的绝对路径。
- 标志 `-w`，等同于 `use warnings;`。使 Perl 在遇到错误时打印出相关信息。
- 注意：错误信息中的行号不一定准确，但是可以作为参考（通常错误就在对应行的附近）。

```
1 #!/usr/bin/perl -w
```

命令解释

- 像是注释，但并不是注释。
- 告诉 Unix/Linux 计算机，这是一个 Perl 程序。
- 本质上是 Perl 语言解释器在文件系统中的绝对路径。
- 标志 `-w`，等同于 `use warnings;`。使 Perl 在遇到错误时打印出相关信息。
- 注意：错误信息中的行号不一定准确，但是可以作为参考（通常错误就在对应行的附近）。

```
1 #!/usr/bin/perl
2 #!/usr/bin/env perl
3 #!/usr/local/bin/perl
4 #!/bin/perl
5 ...
```

教学提纲

```
1 $DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
```

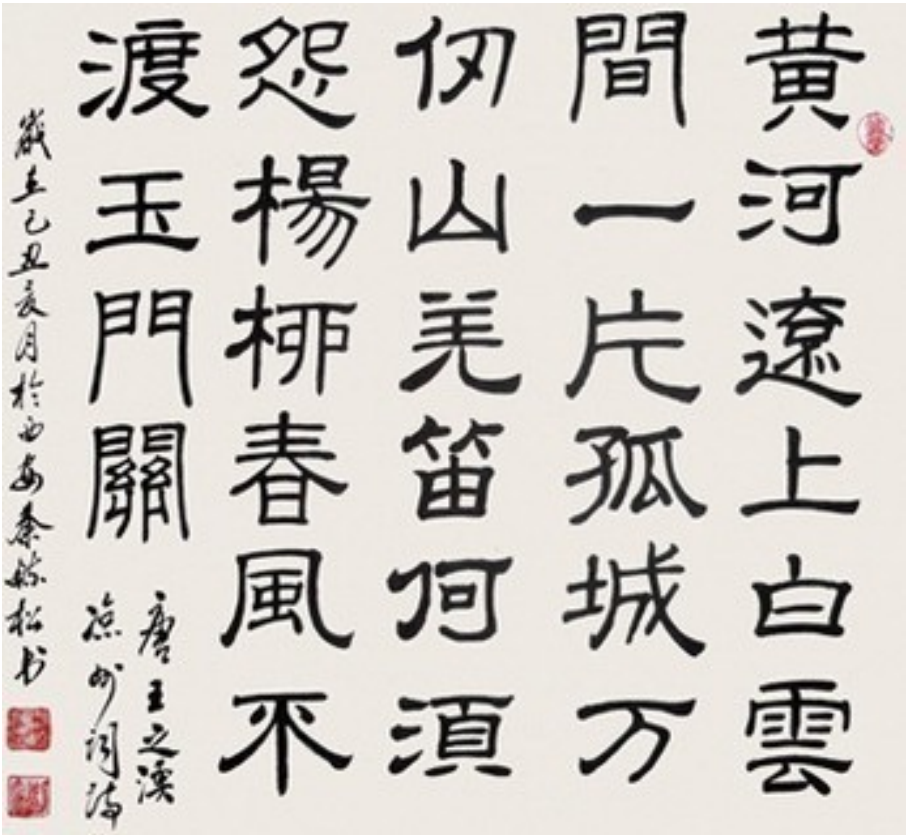
语句

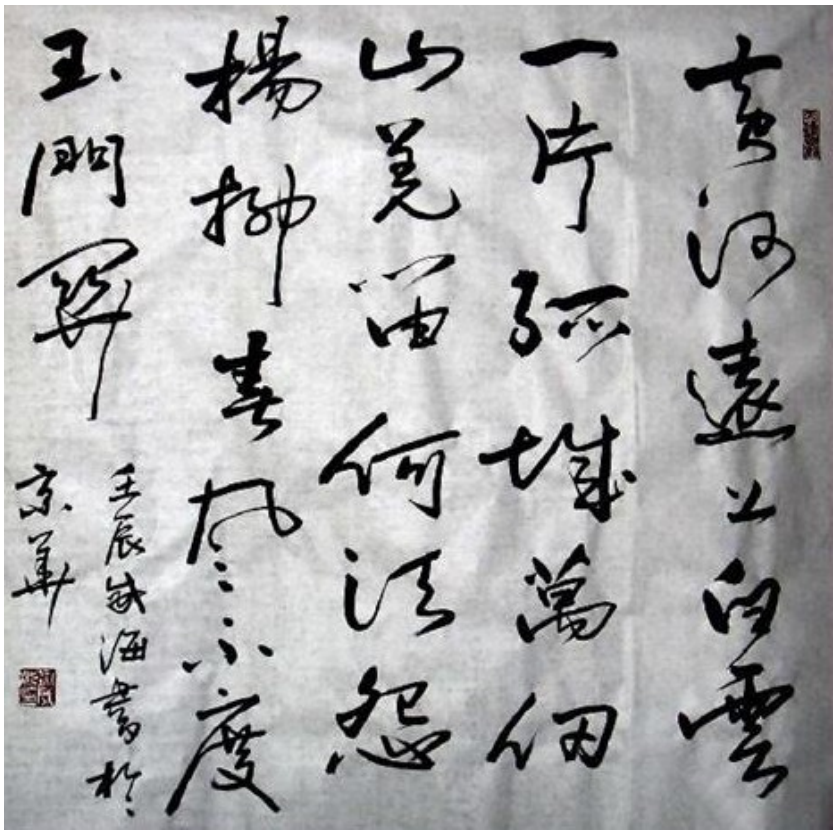
- 这行代码在 Perl 语言中叫做语句（statement）。
- 在 Perl 中，语句以分号；结尾（类似于英语中以句号．进行结尾）。
- 该行是一个赋值语句：把 DNA 序列存储到 `$DNA` 变量中。

```
1 $DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
```

语句

- 这行代码在 Perl 语言中叫做语句（statement）。
- 在 Perl 中，语句以分号；结尾（类似于英语中以句号．进行结尾）。
- 该行是一个赋值语句：把 DNA 序列存储到 `$DNA` 变量中。


黃河遠上白雲
間一片孤城万
仞山羌笛何須
怨楊柳春風不
渡玉門關


黃河遠上白雲
一片孤城萬仞
山羌笛何須怨
楊柳春風不度
玉門關


中国重名最多的姓名
张伟 299025 1
王伟 290619 2
王芳 277293 3
李伟 269453 4
李娜 258581 5
张敏 245553 6
李静 243644 7
王静 243339 8
刘伟 241621 9
王秀英 241189 10
中国重名最多的名
英 41000153 1
华 35266889 2
玉 34243152 3
秀 32120854 4
文 29623154 5
明 28531253 6
兰 25703851 7
金 22224840 8
国 22112399 9
春 22103501 10


2010后最热的30个名字
你家娃中了没?
子涵 欣怡 梓涵 晨曦 紫涵 诗涵
梦琪 嘉怡 子萱 雨涵 可馨 梓萱
思涵 思彤 心怡 雨萱 可欣 雨欣
涵 雨彤 雨轩 佳怡 梦瑶 诗琪
萱 雨馨 思琪 静怡 佳琪 一诺
子轩 浩宇 浩然 博文 宇轩 子涵
雨泽 皓轩 浩轩 俊杰 文博 浩
峻熙 子豪 天佑 俊熙 明轩 致远
睿 宇航 泽宇 鑫 一鸣 俊宇
硕 文轩 俊豪 子墨
@新华网

变量名

- Perl 中变量名的规范：只能由大小写字母、数字和下划线 _ 组成，且第一个字符不能是数字。
- 只要合法，什么样的变量名对于计算机来说都无所谓，都是一样的。
- 有意义的变量名可以清晰地表明程序中变量的作用，使程序易读、易理解。
- 精心选择有意义的变量名是一个好习惯。

标量变量

- 标量变量（scalar variable）：存储单个数据项目的变量，在 Perl 中以美元符号 $ 起始。
- 一个标量变量一次只能存储数据中的一个项目。
- 使用标量变量存储字符串或者数字（比如：hello，25、6.234、3.5E10、-0.8373）。

变量名

- Perl 中变量名的规范：只能由大小写字母、数字和下划线 _ 组成，且第一个字符不能是数字。
- 只要合法，什么样的变量名对于计算机来说都无所谓，都是一样的。
- 有意义的变量名可以清晰地表明程序中变量的作用，使程序易读、易理解。
- 精心选择有意义的变量名是一个好习惯。

标量变量

- 标量变量（scalar variable）：存储单个数据项目的变量，在 Perl 中以美元符号 $ 起始。
- 一个标量变量一次只能存储数据中的一个项目。
- 使用标量变量存储字符串或者数字（比如：hello，25、6.234、3.5E10、-0.8373）。

变量

- Perl 语言从 C 那里继承了一部分命名约定。局部变量与子例程通常使用用下划线分隔开的小写单词来命名；如果是私有变量或子例程，则在前面加一个下划线以表明其私有性；Perl 包中的变量命名则采用单词首字母大写的方式（像英文标题那样）；如果是声明的常量，需要全部大写；Perl 包的名字，除了 pragmata（如 strict），则一律采用“CamelCase”的方式。
- Naming convention (programming)

缩进与大括号

- K&R style, Allman style, Whitesmiths style, GNU style
- indent style

perlstyle

- `perldoc perlstyle`
- perlstyle - Perl style guide

变量

- Perl 语言从 C 那里继承了一部分命名约定。局部变量与子例程通常使用用下划线分隔开的小写单词来命名；如果是私有变量或子例程，则在前面加一个下划线以表明其私有性；Perl 包中的变量命名则采用单词首字母大写的方式（像英文标题那样）；如果是声明的常量，需要全部大写；Perl 包的名字，除了 pragmata（如 strict），则一律采用“CamelCase”的方式。
- Naming convention (programming)

缩进与大括号

- K&R style, Allman style, Whitesmiths style, GNU style
- indent style

perlstyle

- `perldoc perlstyle`
- perlstyle - Perl style guide

变量

- Perl 语言从 C 那里继承了一部分命名约定。局部变量与子例程通常使用用下划线分隔开的小写单词来命名；如果是私有变量或子例程，则在前面加一个下划线以表明其私有性；Perl 包中的变量命名则采用单词首字母大写的方式（像英文标题那样）；如果是声明的常量，需要全部大写；Perl 包的名字，除了 pragmata（如 strict），则一律采用“CamelCase”的方式。
- Naming convention (programming)

缩进与大括号

- K&R style, Allman style, Whitesmiths style, GNU style
- indent style

perlstyle

- `perldoc perlstyle`
- perlstyle - Perl style guide

字符串

- 在 Perl 中，把序列等放在引号（单引号或者双引号）中表明它是字符串。
- 单引号（' '）不会进行变量内插。
- 双引号（" "）能够进行变量内插，可以使用转义字符。

```
1 # 此处两者完全等价
2 $DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
3 $DNA = "ACGGGAGGACGGGAAAATTACTACGGCATTAGC";
4
5 # 此处结果完全不同（变量内插）
6 print '$DNA';
7 print "$DNA";
```

字符串

- 在 Perl 中，把序列等放在引号（单引号或者双引号）中表明它是字符串。
- 单引号（' '）不会进行变量内插。
- 双引号（" "）能够进行变量内插，可以使用转义字符。

```
1 # 此处两者完全等价
2 $DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
3 $DNA = "ACGGGAGGACGGGAAAATTACTACGGCATTAGC";
4
5 # 此处结果完全不同（变量内插）
6 print '$DNA';
7 print "$DNA";
```

赋值

- 使用等号 = 来把一个变量设成特定的值。
- = 叫做赋值操作符（assignment operator）。
- 赋值后就可以通过变量名来获取它的值了。
- 注意赋值语句中项目的顺序：变量在左边（lvalue），要赋给变量的值在右边（rvalue）。
- 牢记在 Perl 中 = 不表示相等（数学），而是进行赋值。

打印输出

- 使用 `print` 函数，它会把变量的值直接打印输出出来。
- `print` 处理的是标量变量。
- 默认是输出到计算机屏幕（标准输出设备，STDOUT）上。

```
1 # 两者效果相同
2 print $DNA;
3 print "$DNA";
4
5 # 不会输出序列
6 print '$DNA';
```

打印输出

- 使用 `print` 函数，它会把变量的值直接打印输出出来。
- `print` 处理的是标量变量。
- 默认是输出到计算机屏幕（标准输出设备，STDOUT）上。

```
1 # 两者效果相同
2 print $DNA;
3 print "$DNA";
4
5 # 不会输出序列
6 print '$DNA';
```

退出

- 使用 `exit;` 语句明确告诉计算机退出程序。
- 在 Perl 中，程序末尾的 `exit;` 并不是必需的。
- Perl 程序一旦运行到末尾，就会自动退出。

```
1 #!/usr/bin/perl -w
2
3 $DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
4
5 print $DNA;
6
7 exit;
```

教学提纲

拼接（concatenate）

- 把一个字符串附加到另一个字符串的末尾。
- 把 AT 和 GC 拼接起来，得到 ATGC。

生物学中的应用

- 把克隆插入到细胞载体中
- 把剪切后的外显子拼接起来（“剪接”中的“接”）
- ……

拼接（concatenate）

- 把一个字符串附加到另一个字符串的末尾。
- 把 AT 和 GC 拼接起来，得到 ATGC。

生物学中的应用

- 把克隆插入到细胞载体中
- 把剪切后的外显子拼接起来（“剪接”中的“接”）
- ……

```
1  #!/usr/bin/perl -w
2  # Example 4-2   Concatenating DNA
3
4  # Store two DNA fragments into two variables
     called $DNA1 and $DNA2
5  $DNA1 = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
6  $DNA2 = 'ATAGTGCCGTGAGAGTGATGTAGTA';
7
8  # Print the DNA onto the screen
9  print "Here are the original two DNA
     fragments:\n\n";
10
11 print $DNA1, "\n";
12
13 print $DNA2, "\n\n";
```

```
15 # Concatenate the DNA fragments into a third
     variable and print them
16 # Using "string interpolation"
17 $DNA3 = "$DNA1$DNA2";
18
19 print "Here is the concatenation of the first
     two fragments (version 1):\n\n";
20
21 print "$DNA3\n\n";
```

```
23 # An alternative way using the "dot operator
     ":
24 # Concatenate the DNA fragments into a third
     variable and print them
25 $DNA3 = $DNA1 . $DNA2;
26
27 print "Here is the concatenation of the first
     two fragments (version 2):\n\n";
28
29 print "$DNA3\n\n";
```

```
31 # Print the same thing without using the
     variable $DNA3
32 print "Here is the concatenation of the first
     two fragments (version 3):\n\n";
33
34 print $DNA1, $DNA2, "\n";
35
36 exit;
```

```
1  Here are the original two DNA fragments:
2
3  ACGGGAGGACGGGAAAATTACTACGGCATTAGC
4  ATAGTGCCGTGAGAGTGATGTAGTA
5
6  Here is the concatenation of the first two fragments (version 1)
     :
7
8  ACGGGAGGACGGGAAAATTACTACGGCATTAGCATAGTGCCGTGAGAGTGATGTAGTA
9
10 Here is the concatenation of the first two fragments (version 2)
     :
11
12 ACGGGAGGACGGGAAAATTACTACGGCATTAGCATAGTGCCGTGAGAGTGATGTAGTA
13
14 Here is the concatenation of the first two fragments (version 3)
     :
15
16 ACGGGAGGACGGGAAAATTACTACGGCATTAGCATAGTGCCGTGAGAGTGATGTAGTA
```

```
1 print $DNA1, "\n";
2 print $DNA2, "\n\n";
```

说明

- \n：换行符，定位到下一行的开头
- \n\n：两个新行（中间添加一个空行）
- 空行：没有任何打印输出的行（取决于操作系统）
- 包裹在双引号中的换行符表示它们是字符串的一部分
- "\n" 会输出换行符，'\n' 输出 \n 本身
- 逗号分隔列表中的项目，print 语句会输出列表中的所有项目

```
1 print $DNA1, "\n";
2 print $DNA2, "\n\n";
```

说明

- \n：换行符，定位到下一行的开头
- \n\n：两个新行（中间添加一个空行）
- 空行：没有任何打印输出的行（取决于操作系统）
- 包裹在双引号中的换行符表示它们是字符串的一部分
- "\n" 会输出换行符，'\n' 输出 \n 本身
- 逗号分隔列表中的项目，print 语句会输出列表中的所有项目

```
1 $DNA3 = "$DNA1$DNA2"; print "$DNA3\n\n";
2 $DNA3 = $DNA1 . $DNA2; print "$DNA3\n\n";
3 print $DNA1, $DNA2, "\n";
4 print "$DNA1$DNA2\n";
5 ...
```

说明

- 字符串内插（string interpolation）／变量替换：双引号会把字符串中的变量替换成变量的值（必要时可以使用大括号来保护变量）
- 点操作符：拼接字符串
- 使用 print 语句
- There's more than one way to do it.
- There are more than two ways to do it.

标量变量

- 标量变量可以存储字符串、整数、浮点数、布尔值等
- Perl 能够“智能”判断存储的是哪种类型的数据

```
1 $number = 17;
2 print $number, "\n";
```

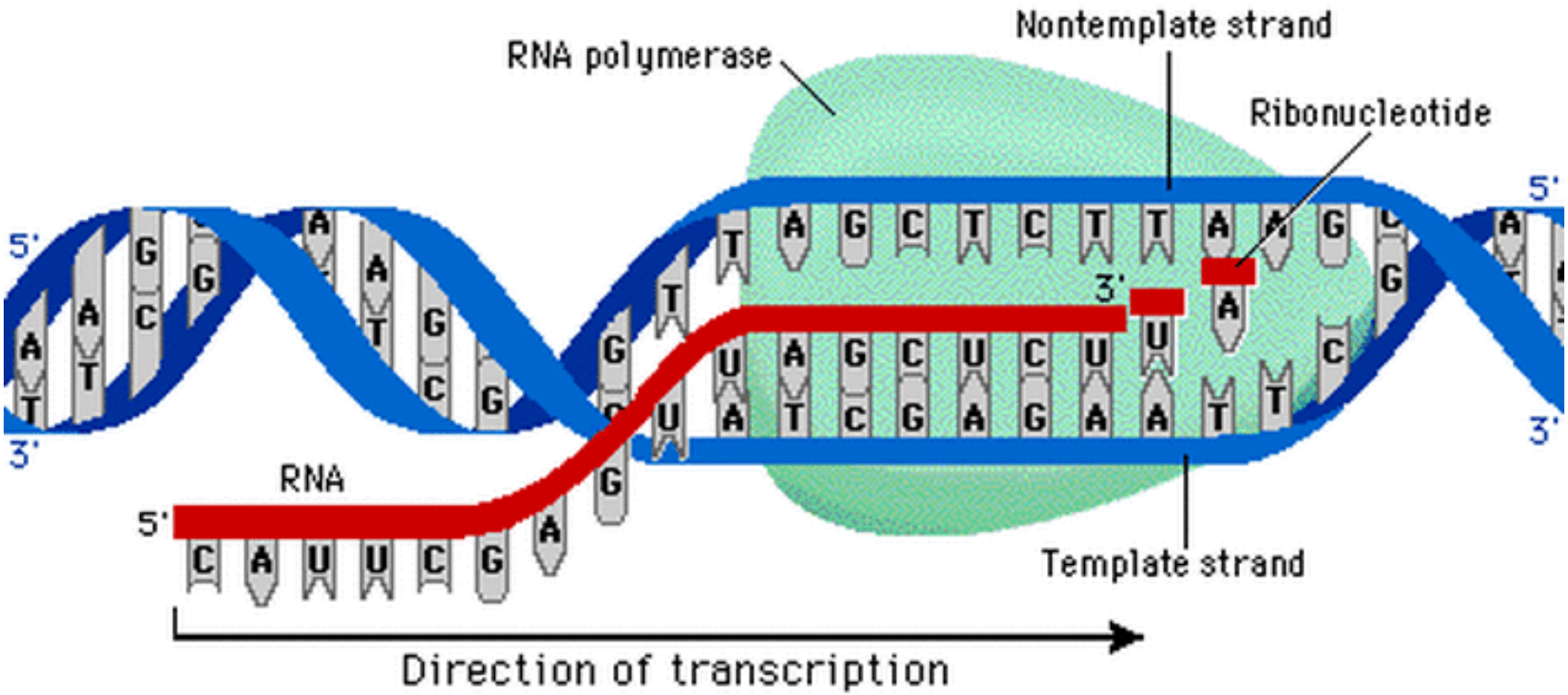
标量变量

- 标量变量可以存储字符串、整数、浮点数、布尔值等
- Perl 能够“智能”判断存储的是哪种类型的数据

```
1 $number = 17;
2 print $number, "\n";
```

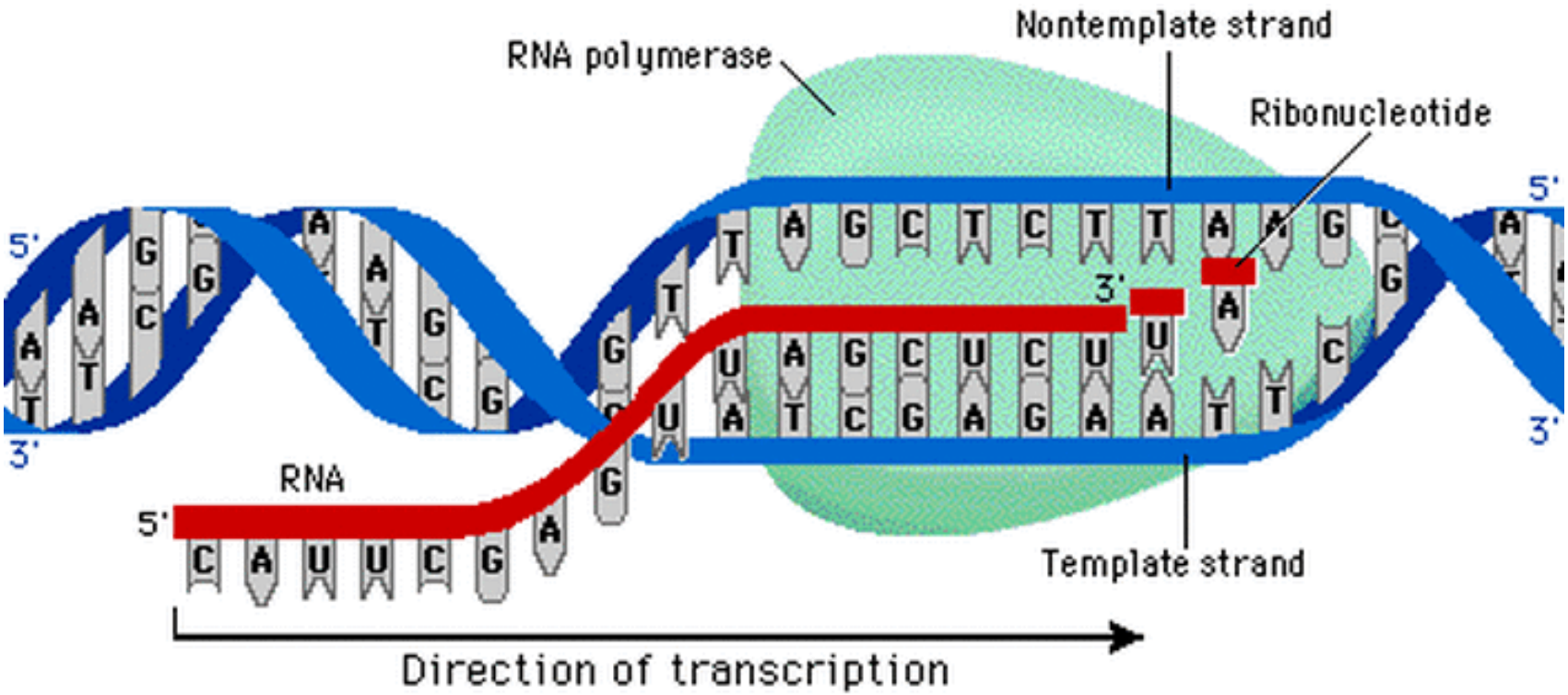
```
1  #!/usr/bin/perl -w
2
3  $DNA1 = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
4  $DNA2 = 'ATAGTGCCGTGAGAGTGATGTAGTA';
5
6  print $DNA1, "\n";
7  print $DNA2, "\n\n";
8
9  $DNA3 = "$DNA1$DNA2";
10 print "$DNA3\n\n";
11
12 $DNA3 = $DNA1 . $DNA2;
13 print "$DNA3\n\n";
14
15 print $DNA1, $DNA2, "\n";
16
17 exit;
```

教学提纲



问题简化

DNA 转录成 RNA $\Longrightarrow$ 把 DNA 中所有的 T 替换成 U



问题简化

DNA 转录成 RNA $\Longrightarrow$ 把 DNA 中所有的 T 替换成 U

```perl
#!/usr/bin/perl -w
# Example 4-3   Transcribing DNA into RNA

# The DNA
$DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';

# Print the DNA onto the screen
print "Here is the starting DNA:\n\n";

print "$DNA\n\n";
```

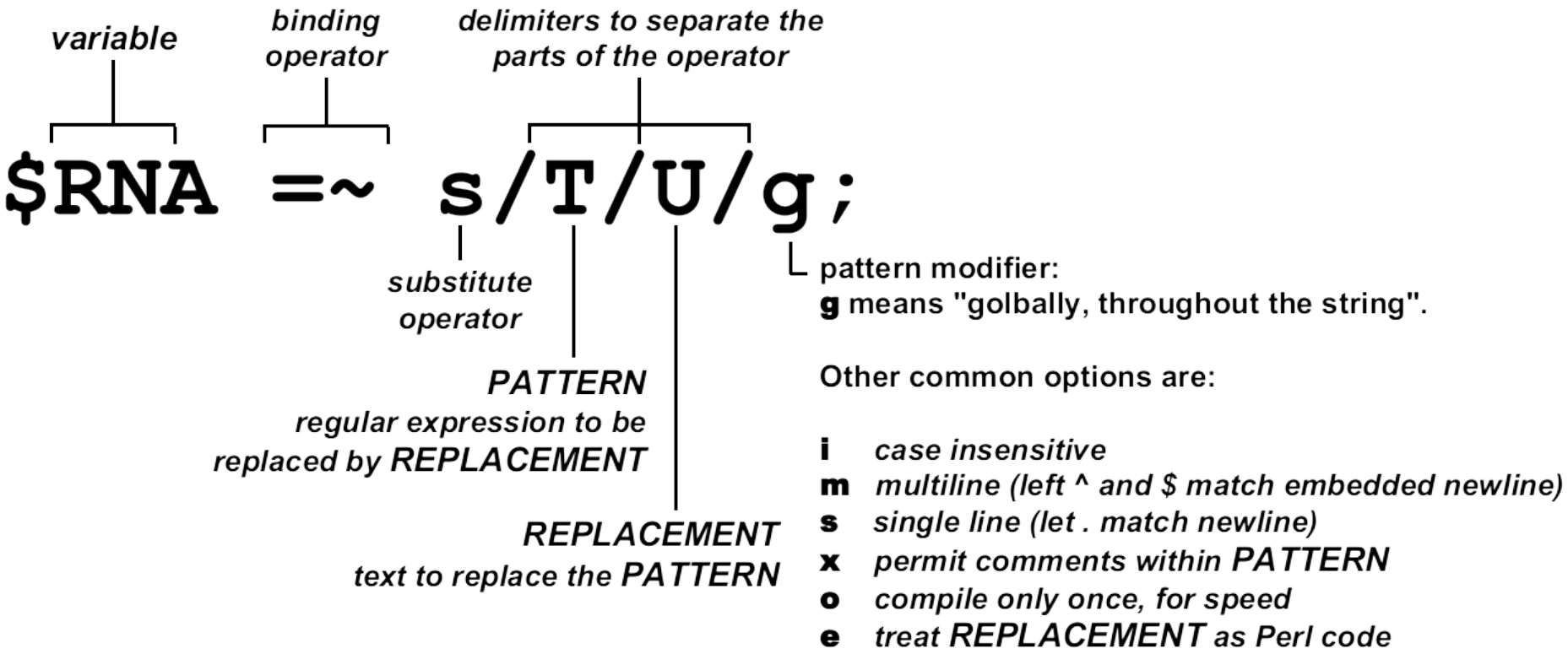
```
12 # Transcribe the DNA to RNA by substituting
     all T's with U's.
13 $RNA = $DNA;
14
15 $RNA =~ s/T/U/g;
16
17 # Print the RNA onto the screen
18 print "Here is the result of transcribing the
     DNA to RNA:\n\n";
19
20 print "$RNA\n";
21
22 # Exit the program.
23 exit;
```

```
1 Here is the starting DNA:
2
3 ACGGGAGGACGGGAAAATTACTACGGCATTAGC
4
5 Here is the result of transcribing the DNA to
     RNA:
6
7 ACGGGAGGACGGGAAAAUUACUACGGCAUUAGC
```

- Perl 能够轻松处理 DNA 字符串等文本数据
- 常见操作：翻译、反转、替换、删除、排序等
- Perl 在生物信息学领域独领风骚的主要原因（之一）

```perl
# $RNA开始存储的其实是DNA
$RNA = $DNA;
# 替换后，$RNA存储的才是RNA
$RNA =~ s/T/U/g;

# 完全等价的语句
# $RNA存储的就真的只是RNA了
($RNA = $DNA) =~ s/T/U/g;
```


variable
binding operator
delimiters to separate the parts of the operator
$RNA =~ s/T/U/g;
substitute operator
PATTERN
regular expression to be replaced by REPLACEMENT
REPLACEMENT
text to replace the PATTERN
pattern modifier:
g means "golbally, throughout the string".
Other common options are:
i case insensitive
m multiline (left ^ and $ match embedded newline)
s single line (let . match newline)
x permit comments within PATTERN
o compile only once, for speed
e treat REPLACEMENT as Perl code

```
1  #!/usr/bin/perl -w
2
3  $DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
4
5  print "Here is the starting DNA:\n\n";
6  print "$DNA\n\n";
7
8  $RNA = $DNA;
9  $RNA =~ s/T/U/g;
10
11 print "Here is the result of transcribing the
       DNA to RNA:\n\n";
12 print "$RNA\n";
13
14 exit;
```

教学提纲

- http://www.perl.com/
- `perldoc -f print`（perldoc 手册：`man perldoc`）
- 文档太全，直接忽略掉对你来说毫无意义的内容吧
- 翻阅文档是学习 Perl 的绝佳途径（不要“骑着驴找驴”）

教学提纲

应用

- 给出一条链，输出另一条链
- 在查询 DNA 时，自动查询其反向互补序列
- 从基因的负链得到正链
- ……

```
1  #!/usr/bin/perl -w
2  # Example 4-4   Calculating the reverse
     complement of a strand of DNA
3
4  # The DNA
5  $DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
6
7  # Print the DNA onto the screen
8  print "Here is the starting DNA:\n\n";
9
10 print "$DNA\n\n";
```

```
12 # Calculate the reverse complement
13 #  Warning: this attempt will fail!
14 #
15 # First, copy the DNA into new variable $revcom
16 # (short for REVerse COMplement)
17 # Notice that variable names can use lowercase letters
     like
18 # "revcom" as well as uppercase like "DNA".  In fact,
19 # lowercase is more common.
20 #
21 # It doesn't matter if we first reverse the string and
     then
22 # do the complementation; or if we first do the
     complementation
23 # and then reverse the string.  Same result each time.
24 # So when we make the copy we'll do the reverse in the
     same statement.
```

```
27 $revcom = reverse $DNA;
28
29 #
30 # Next substitute all bases by their complements,
31 # A->T, T->A, G->C, C->G
32 #
33
34 $revcom =~ s/A/T/g;
35 $revcom =~ s/T/A/g;
36 $revcom =~ s/G/C/g;
37 $revcom =~ s/C/G/g;
38
39 # Print the reverse complement DNA onto the screen
40 print "Here is the reverse complement DNA:\n\n";
41
42 print "$revcom\n";
```

```
45 # Oh-oh, that didn't work right!
46 # Our reverse complement should have all the bases in it, since
     the
47 # original DNA had all the bases-but ours only has A and G!
48 #
49 # Do you see why?
50 #
51 # The problem is that the first two substitute commands above
     change
52 # all the A's to T's (so there are no A's) and then all the
53 # T's to A's (so all the original A's and T's are all now A's).
54 # Same thing happens to the G's and C's all turning into G's.
55 #
56
57 print "\nThat was a bad algorithm, and the reverse complement
     was wrong!\n";
58 print "Try again ... \n\n";
```

```
60 # Make a new copy of the DNA (see why we saved the
      original?)
61 $revcom = reverse $DNA;
62
63 # See the text for a discussion of tr///
64 $revcom =~ tr/ACGTacgt/TGCAtgca/;
65
66 # Print the reverse complement DNA onto the screen
67 print "Here is the reverse complement DNA:\n\n";
68
69 print "$revcom\n";
70
71 print "\nThis time it worked!\n\n";
72
73 exit;
```

```
1  Here is the starting DNA:
2
3  ACGGGAGGACGGGAAAATTACTACGGCATTAGC
4
5  Here is the reverse complement DNA:
6
7  GGAAAAGGGGAAGAAAAAAAGGGGAGGAGGGGA
8
9  That was a bad algorithm, and the reverse
     complement was wrong!
10 Try again ...
11
12 Here is the reverse complement DNA:
13
14 GCTAATGCCGTAGTAATTTTCCCGTCCTCCCGT
15
16 This time it worked!
```

相似的经历

1. 编写代码
2. 代码不工作
3. 解决问题（修正语法，重新思考、设计新的方法，……）

解决策略

- 检查代码的细节
- 查阅文档
- 检索（特性，模块，……）

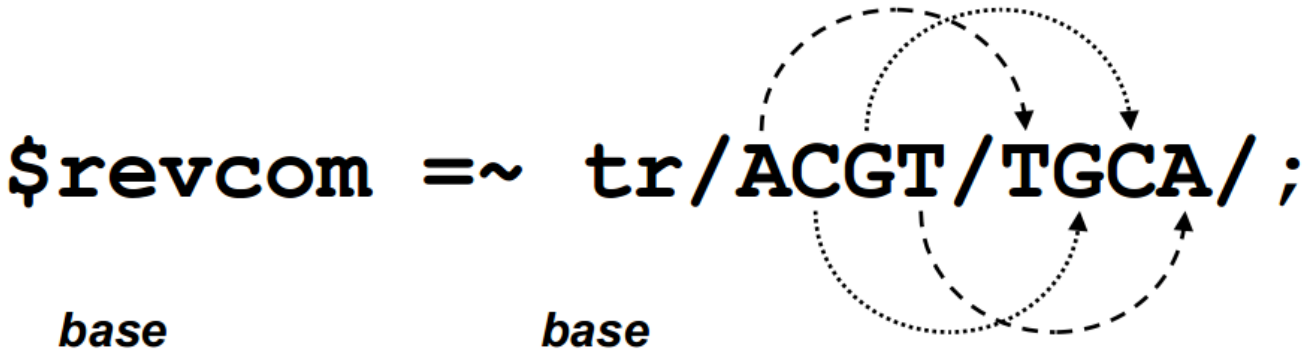
相似的经历

1. 编写代码
2. 代码不工作
3. 解决问题（修正语法，重新思考、设计新的方法，……）

解决策略

- 检查代码的细节
- 查阅文档
- 检索（特性，模块，……）

- reverse 函数：反转字符串等元素的顺序
- tr 函数：一次性把一个字符集翻译成新的字符



base	maps to	base
A	⋯⋯▸	T
C	⋯⋯▸	G
G	⋯⋯▸	C
T	⋯⋯▸	A

```
1  #!/usr/bin/perl -w
2
3  $DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
4
5  print "Here is the starting DNA:\n\n";
6  print "$DNA\n\n";
7
8  $revcom = reverse $DNA;
9  $revcom =~ tr/ACGTacgt/TGCAtgca/;
10
11 print "Here is the reverse complement DNA:\n\
     n";
12 print "$revcom\n";
13
14 exit;
```

已经学习

- 存储 DNA 序列
- 把 DNA 片段拼接起来
- 把 DNA 转录成 RNA
- 获取反向互补序列

即将学习

- 在 Perl 中使用蛋白质序列数据
- 从文件读取蛋白质序列数据
- Perl 语言中的数组
- Perl 语言中的上下文

序列和字符串 | 小结

已经学习

- 存储 DNA 序列
- 把 DNA 片段拼接起来
- 把 DNA 转录成 RNA
- 获取反向互补序列

即将学习

- 在 Perl 中使用蛋白质序列数据
- 从文件读取蛋白质序列数据
- Perl 语言中的数组
- Perl 语言中的上下文

教学提纲

```
1 MNIDDKLEGLFLKCGGIDEMQSSRTMVVMGGVSGQSTVSGELQD
2 SVLQDRSMPHQEILAADEVLQESEMRQQDMISHDELMVHEETVKNDEEQMETHERLPQ
3 GLQYALNVPISVKQEITFTDVSEQLMRDKKQIR
```

补充说明

- 认真组织文件和文件夹
- 仔细考虑文件和文件夹的命名
- 尽量仅通过文件名、而不需要打开文件就可以对文件保存的数据有所了解
- 比如：`NM_021964fragment.pep`

```
1 MNIDDKLEGLFLKCGGIDEMQSSRTMVVMGGVSGQSTVSGELQD
2 SVLQDRSMPHQEILAADEVLQESEMRQQDMISHDELMVHEETVKNDEEQMETHERLPQ
3 GLQYALNVPISVKQEITFTDVSEQLMRDKKQIR
```

补充说明

- 认真组织文件和文件夹
- 仔细考虑文件和文件夹的命名
- 尽量仅通过文件名、而不需要打开文件就可以对文件保存的数据有所了解
- 比如：`NM_021964fragment.pep`

```
1 #!/usr/bin/perl -w
2 # Example 4-5   Reading protein sequence data
     from a file
3
4 # The filename of the file containing the
    protein sequence data
5 $proteinfilename = 'NM_021964fragment.pep';
6
7 # First we have to "open" the file, and
    associate
8 # a "filehandle" with it.  We choose the
    filehandle
9 # PROTEINFILE for readability.
```

```
10 open( PROTEINFILE, $proteinfilename );
11
12 # Now we do the actual reading of the protein sequence
     data from the file,
13 # by using the angle brackets < and > to get the input
     from the
14 # filehandle.  We store the data into our variable
     $protein.
15 $protein = <PROTEINFILE>;
16
17 # Now that we've got our data, we can close the file.
18 close PROTEINFILE;
19
20 # Print the protein onto the screen
21 print "Here is the protein:\n\n";
22
23 print $protein;
24
25 exit;
```

```
1 Here is the protein:
2
3 MNIDDKLEGLFLKCGGIDEMQSSRTMVVMGGVSGQSTVSGELQD
```

```perl
# 第一步： 把文件和文件句柄关联起来，之后对文件的操作都通过文件句柄来进行
open( PROTEINFILE, $proteinfilename );
# 第二步：读取文件中的数据
$protein = <PROTEINFILE>;
# 第三步：把文件和文件句柄解关联
close PROTEINFILE;
```

补充说明

- open 函数还有很多选项，用于精确指定如何使用文件
- 文件句柄通常使用大写字母
- < >：输入操作符，从文件中读取数据
- 好习惯：有 open 就有 close

```
1 # 第一步： 把文件和文件句柄关联起来，之后对文件的操作
    都通过文件句柄来进行
2 open( PROTEINFILE, $proteinfilename );
3 # 第二步：读取文件中的数据
4 $protein = <PROTEINFILE>;
5 # 第三步：把文件和文件句柄解关联
6 close PROTEINFILE;
```

补充说明

- open 函数还有很多选项，用于精确指定如何使用文件
- 文件句柄通常使用大写字母
- < >：输入操作符，从文件中读取数据
- 好习惯：有 open 就有 close

```
1  # 读取文件
2  open my $FH, '<', $filename or die "$0 : failed to
       open input file '$filename' : $!\n";
3  ... <$FH> ...
4  close $FH or warn "$0 : failed to close input file
       '$filename' : $!\n";
5
6  # 写入文件
7  open my $FH_OUT, '>', $fn_out or die "$0 : failed
     to open output file '$fn_out' : $!\n";
8  select $FH_OUT;
9  # OR: use $FH_OUT for every print
10 print $FH_OUT "something...";
11 ...
12 close $FH_OUT or warn "$0 : failed to close output
       file '$fn_out' : $!\n";
```

```
1  #!/usr/bin/perl -w
2  # Example 4-6   Reading protein sequence data
       from a file, take 2
3
4  # The filename of the file containing the
     protein sequence data
5  $proteinfilename = 'NM_021964fragment.pep';
6
7  # First we have to "open" the file, and
     associate
8  # a "filehandle" with it.  We choose the
     filehandle
9  # PROTEINFILE for readability.
10 open( PROTEINFILE, $proteinfilename );
```

```
12 # Now we do the actual reading of the protein
     sequence data from the file,
13 # by using the angle brackets < and > to get
   the input from the
14 # filehandle.  We store the data into our
   variable $protein.
15 #
16 # Since the file has three lines, and since
   the read only is
17 # returning one line, we'll read a line and
   print it, three times.
```

```
19 # First line
20 $protein = <PROTEINFILE>;
21
22 # Print the protein onto the screen
23 print "\nHere is the first line of the protein
     file:\n\n";
24
25 print $protein;
26
27 # Second line
28 $protein = <PROTEINFILE>;
29
30 # Print the protein onto the screen
31 print "\nHere is the second line of the protein
     file:\n\n";
32
33 print $protein;
```

```
35 # Third line
36 $protein = <PROTEINFILE>;
37
38 # Print the protein onto the screen
39 print "\nHere is the third line of the
     protein file:\n\n";
40
41 print $protein;
42
43 # Now that we've got our data, we can close
     the file.
44 close PROTEINFILE;
45
46 exit;
```

```
1  Here is the first line of the protein file:
2
3  MNIDDKLEGLFLKCGGIDEMQSSRTMVVMGGVSGQSTVSGELQD
4
5  Here is the second line of the protein file:
6
7  SVLQDRSMPHQEILAADEVLQESEMRQQDMISHDELMVHEETVKNDEEQMETHERLPQ
8
9  Here is the third line of the protein file:
10
11 GLQYALNVPISVKQEITFTDVSEQLMRDKKQIR
```

```
1  #!/usr/bin/perl -w
2
3  $proteinfilename = 'NM_021964fragment.pep';
4
5  open( PROTEINFILE, $proteinfilename );
6
7  $protein = <PROTEINFILE>;
8  print $protein;
9  $protein = <PROTEINFILE>;
10 print $protein;
11 $protein = <PROTEINFILE>;
12 print $protein;
13
14 close PROTEINFILE;
15
16 exit;
```

序列和字符串 | 读取文件 | 程序 4.6 | WRONG!

```perl
#!/usr/bin/perl -w

$proteinfilename = 'NM_021964fragment.pep';

open( PROTEINFILE, $proteinfilename );

$protein = <PROTEINFILE>;
$protein = <PROTEINFILE>;
$protein = <PROTEINFILE>;
print $protein;
# What is wrong?

close PROTEINFILE;

exit;
```

便捷之处

- 自动读取文件的下一行
- 程序记录读取到哪儿，需要读取哪一行

繁琐之处

- 一次只能读取输入文件的一行
- 如果一个文件有成千上万行怎么办？
- 解决办法
  - 数组
  - 循环

便捷之处

- 自动读取文件的下一行
- 程序记录读取到哪儿，需要读取哪一行

繁琐之处

- 一次只能读取输入文件的一行
- 如果一个文件有成千上万行怎么办？
- 解决办法
  - 数组
  - 循环

教学提纲

教学提纲

```
1 #!/usr/bin/perl -w
2 # Example 4-7   Reading protein sequence data
     from a file, take 3
3
4 # The filename of the file containing the
    protein sequence data
5 $proteinfilename = 'NM_021964fragment.pep';
6
7 # First we have to "open" the file
8 open( PROTEINFILE, $proteinfilename );
```

```
10 # Read the protein sequence data from the
     file, and store it
11 # into the array variable @protein
12 @protein = <PROTEINFILE>;
13
14 # Print the protein onto the screen
15 print @protein;
16
17 # Close the file.
18 close PROTEINFILE;
19
20 exit;
```

```
1 MNIDDKLEGLFLKCGGIDEMQSSRTMVVMGGVSGQSTVSGELQD
2 SVLQDRSMPHQEILAADEVLQESEMRQQDMISHDELMVHEETVKNDEEQMETHERLPQ
3 GLQYALNVPISVKQEITFTDVSEQLMRDKKQIR
```

```
1  #!/usr/bin/perl -w
2
3  $proteinfilename = 'NM_021964fragment.pep';
4
5  open( PROTEINFILE, $proteinfilename );
6
7  @protein = <PROTEINFILE>;
8  print @protein;
9
10 close PROTEINFILE;
11
12 exit;
```

教学提纲

数组（array）

- 存储多个标量值的变量
- 变量的值可以是数字、字符串等

标量 vs. 数组

- 标量：单数，以 `$` 起始（scalar）
- 数组：复数，以 `@` 起始（array）
- print 函数不仅可以处理标量变量，也可以处理数组变量
- 数组中的每一个元素都是标量值
- 通过索引/下标/偏移量/位置（从 0 开始）对数组中的元素进行访问

数组（array）

- 存储多个标量值的变量
- 变量的值可以是数字、字符串等

标量 vs. 数组

- 标量：单数，以 $ 起始（scalar）
- 数组：复数，以 @ 起始（array）
- print 函数不仅可以处理标量变量，也可以处理数组变量
- 数组中的每一个元素都是标量值
- 通过索引/下标/偏移量/位置（从 0 开始）对数组中的元素进行访问

```
1  # Here's one way to declare an array,
     initialized with a list of four scalar
     values.
2  @bases = ('A', 'C', 'G', 'T');
3
4  # Now we'll print each element of the array
5  print "Here are the array elements:";
6  print "\nFirst element: ";
7  print $bases[0];
8  print "\nSecond element: ";
9  print $bases[1];
10 print "\nThird element: ";
11 print $bases[2];
12 print "\nFourth element: ";
13 print $bases[3];
```

```
1 Here are the array elements:
2 First element: A
3 Second element: C
4 Third element: G
5 Fourth element: T
```

```
1  @bases = ('A', 'C', 'G', 'T');
2  print "\n\nHere are the array elements: ";
3
4  # 元素肩并肩地输出
5  print @bases;
6  #Here are the array elements: ACGT
7
8  # 输出用空格分隔的元素（注意print语句中的双引号）
9  print "@bases";
10 #Here are the array elements: A C G T
```

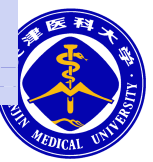
```perl
# pop: 从数组的末尾拿掉一个元素
@bases = ('A', 'C', 'G', 'T');
$base1 = pop @bases;
print "Here's the element removed from the
  end: ";
print $base1, "\n\n";
print "Here's the remaining array of bases: "
  ;
print "@bases";
```

```
Here's the element removed from the end: T

Here's the remaining array of bases: A C G
```

```
1 # pop: 从数组的末尾拿掉一个元素
2 @bases = ('A', 'C', 'G', 'T');
3 $base1 = pop @bases;
4 print "Here's the element removed from the
    end: ";
5 print $base1, "\n\n";
6 print "Here's the remaining array of bases: "
    ;
7 print "@bases";
```

```
1 Here's the element removed from the end: T
2
3 Here's the remaining array of bases: A C G
```



```
1 # shift: 从数组的开头拿掉一个元素
2 @bases = ('A', 'C', 'G', 'T');
3 $base2 = shift @bases;
4 print "Here's an element removed from the
    beginning: ";
5 print $base2, "\n\n";
6 print "Here's the remaining array of bases: "
    ;
7 print "@bases";
```

```
1 Here's an element removed from the beginning:
    A
2
3 Here's the remaining array of bases: C G T
```

```perl
# shift: 从数组的开头拿掉一个元素
@bases = ('A', 'C', 'G', 'T');
$base2 = shift @bases;
print "Here's an element removed from the
  beginning: ";
print $base2, "\n\n";
print "Here's the remaining array of bases: "
  ;
print "@bases";
```

```
Here's an element removed from the beginning:
   A

Here's the remaining array of bases: C G T
```

```
1 # unshift: 把一个元素添加到数组的开头
2 @bases = ('A', 'C', 'G', 'T');
3 $base1 = pop @bases;
4 unshift (@bases, $base1);
5 print "Here's the element from the end put on
     the beginning: ";
6 print "@bases\n\n";
```

```
1 Here's the element from the end put on the
    beginning: T A C G
```

```perl
# unshift: 把一个元素添加到数组的开头
@bases = ('A', 'C', 'G', 'T');
$base1 = pop @bases;
unshift (@bases, $base1);
print "Here's the element from the end put on
   the beginning: ";
print "@bases\n\n";
```

```
Here's the element from the end put on the
  beginning: T A C G
```

```perl
# push: 把一个元素添加到数组的末尾
@bases = ('A', 'C', 'G', 'T');
$base2 = shift @bases;
push (@bases, $base2);
print "Here's the element from the beginning
  put on the end: ";
print "@bases\n\n";
```

```
Here's the element from the beginning put on
  the end: C G T A
```

```
1 # push: 把一个元素添加到数组的末尾
2 @bases = ('A', 'C', 'G', 'T');
3 $base2 = shift @bases;
4 push (@bases, $base2);
5 print "Here's the element from the beginning
    put on the end: ";
6 print "@bases\n\n";
```

```
1 Here's the element from the beginning put on
    the end: C G T A
```

```perl
# reverse: 反转数组
@bases = ('A', 'C', 'G', 'T');
@reverse = reverse @bases;
print "Here's the array in reverse: ";
print "@reverse\n";
```

```
Here's the array in reverse: T G C A
```

```
1 # reverse: 反转数组
2 @bases = ('A', 'C', 'G', 'T');
3 @reverse = reverse @bases;
4 print "Here's the array in reverse: ";
5 print "@reverse\n";
```

```
1 Here's the array in reverse: T G C A
```

```
1 # scalar @array: 获取数组的长度（数组中元素的个数）
2 @bases = ('A', 'C', 'G', 'T');
3 print "Here's the length of the array: ";
4 print scalar @bases, "\n";
```

```
1 Here's the length of the array: 4
```

```
1 # scalar @array: 获取数组的长度（数组中元素的个数）
2 @bases = ('A', 'C', 'G', 'T');
3 print "Here's the length of the array: ";
4 print scalar @bases, "\n";
```

```
1 Here's the length of the array: 4
```

```
1 # splice: 在数组的任意一个位置插入一个元素（或者删除
    任意一个或多个元素）
2 @bases = ('A', 'C', 'G', 'T');
3 splice ( @bases, 2, 0, 'X' );
4 print "Here's the array with an element
    inserted after the 2nd element:
5 ";
6 print "@bases\n";
```

```
1 Here's the array with an element inserted
    after the 2nd element: A C X G T
```

```
1 # splice: 在数组的任意一个位置插入一个元素（或者删除
    任意一个或多个元素）
2 @bases = ('A', 'C', 'G', 'T');
3 splice ( @bases, 2, 0, 'X' );
4 print "Here's the array with an element
    inserted after the 2nd element:
5 ";
6 print "@bases\n";
```

```
1 Here's the array with an element inserted
    after the 2nd element: A C X G T
```

```perl
my @bases = ( "A", "C", "G", "T" );

splice ( @bases, 4, 0, "U" );
print "@bases\n"; # A C G T U

splice ( @bases, 3, 1, "U" );
print "@bases\n"; # A C G U U

splice ( @bases, 3, 1 );
print "@bases\n"; # A C G U

splice ( @bases, 2 );
print "@bases\n"; # A C

splice(@bases); # 清空数组，等同于:
undef(@bases); 或 @bases=();
```

```
1  push(@a,$x,$y)
2  splice(@a,@a,0,$x,$y)
3
4  pop(@a)
5  splice(@a,-1)
6
7  shift(@a)
8  splice(@a,0,1)
9
10 unshift(@a,$x,$y)
11 splice(@a,0,0,$x,$y)
12
13 $a[$i] = $y
14 splice(@a,$i,1,$y)
```

教学提纲

笑话

- 小刘喜欢讲笑话，他的笑话说不完。
- 小刘老是闹笑话，他的笑话说不完。

难吃

- 一点盐都没放，鱼太难吃了。
- 全都是毛毛刺，鱼太难吃了。

谁也赢不了

- 中国乒乓球谁也赢不了！
- 中国足球谁也赢不了！

笑话

- 小刘喜欢讲笑话，他的笑话说不完。
- 小刘老是闹笑话，他的笑话说不完。

难吃

- 一点盐都没放，鱼太难吃了。
- 全都是毛毛刺，鱼太难吃了。

谁也赢不了

- 中国乒乓球谁也赢不了！
- 中国足球谁也赢不了！

笑话

- 小刘喜欢讲笑话，他的笑话说不完。
- 小刘老是闹笑话，他的笑话说不完。

难吃

- 一点盐都没放，鱼太难吃了。
- 全都是毛毛刺，鱼太难吃了。

谁也赢不了

- 中国乒乓球谁也赢不了！
- 中国足球谁也赢不了！

意思

小明送给领导红包。

领导：“你这是什么意思？” 小明：“没什么意思，意思意思。”
领导：“你这就不够意思了。” 小明：“小意思，小意思。”
领导：“你这人真有意思。” 小明：“其实也没有别的意思。”
领导：“那我就不好意思了。” 小明：“是我不好意思。”

上下文环境

- Perl 语言中的上下文环境类似于自然语言中的语境。
- Perl 语言中有两种上下文环境：标量上下文和列表上下文。
- Perl 语言中许多操作符的表现依赖于它所处的上下文环境。

```perl
#!/usr/bin/perl -w
# Example 4-8   Demonstration of "scalar context"
  and "list context"

@bases = ( 'A', 'C', 'G', 'T' );
print "@bases\n";
#A C G T

$a = @bases;
print $a, "\n";
#4

($a) = @bases;
print $a, "\n";
#A

exit;
```

```
1 $a = @bases;
```

说明

- 数组是一种列表
- 语句的左边是一个标量变量，表明这是一个标量上下文（scalar context）
- 在标量上下文中，数组会对其大小进行求值（即获得数组中的元素个数）

获取数组元素个数

```
1 $number = @array;
2 #明确指定是标量上下文
3 $number = scalar @array;
```

```
1 $a = @bases;
```

说明

- 数组是一种列表
- 语句的左边是一个标量变量，表明这是一个标量上下文（scalar context）
- 在标量上下文中，数组会对其大小进行求值（即获得数组中的元素个数）

获取数组元素个数

```
1 $number = @array;
2 #明确指定是标量上下文
3 $number = scalar @array;
```

```
1 $a = @bases;
```

说明

- 数组是一种列表
- 语句的左边是一个标量变量，表明这是一个标量上下文（scalar context）
- 在标量上下文中，数组会对其大小进行求值（即获得数组中的元素个数）

获取数组元素个数

```
1 $number = @array;
2 #明确指定是标量上下文
3 $number = scalar @array;
```

```
1 ($a) = @bases;
```

说明

- 数组是一种列表
- 语句的左边是一个列表（该列表仅有 `$a` 一个变量），表明这是一个列表上下文（list context）
- 在列表上下文中，数组会把它的元素展开成一个列表
- 如果左边没有足够的变量用来赋值，那么就只有数组中的部分元素会被赋值给变量
- 如果左边变量的个数多于数组中的元素，多出来的变量将不会被赋值，处于未初始化状态

```
1 ($a) = @bases;
```

说明

- 数组是一种列表
- 语句的左边是一个列表（该列表仅有 `$a` 一个变量），表明这是一个列表上下文（list context）
- 在列表上下文中，数组会把它的元素展开成一个列表
- 如果左边没有足够的变量用来赋值，那么就只有数组中的部分元素会被赋值给变量
- 如果左边变量的个数多于数组中的元素，多出来的变量将不会被赋值，处于未初始化状态

```perl
my @bases = ('A', 'C', 'G', 'T');

# 参数为负数
splice (@bases, 1, -1);    # A T
splice (@bases, -1, 1);    # A C G
splice (@bases, -2, -1);   # A C T

# 上下文
my @base = splice (@bases, 1, 2);
print "@base"; # C G
my $base = splice (@bases, 1, 2);
print "$base"; # G
my $base = splice (@bases, 1, 0);
print "$base"; # undef
```

教学提纲

教学提纲

知识点

- Perl 语言基础：命令解释，注释，语句，运行，单引号与双引号，赋值，文档，读取文件，……
- 变量：标量，数组
- 字符串操作：拼接，替换，翻译，反转
- 数组：初始化，索引，常见操作
- 上下文：标量上下文，列表上下文

技能

- 能够编写 Perl 程序：存储 DNA 序列、拼接 DNA 片段、把 DNA 转录成 RNA、获取 DNA 的反向互补序列。
- 能够编写 Perl 程序：从文件中读取所需数据。
- 掌握 Perl 语言中数组的常见操作。
- 理解并能熟练应用标量上下文和列表上下文。

知识点

- Perl 语言基础：命令解释，注释，语句，运行，单引号与双引号，赋值，文档，读取文件，……
- 变量：标量，数组
- 字符串操作：拼接，替换，翻译，反转
- 数组：初始化，索引，常见操作
- 上下文：标量上下文，列表上下文

技能

- 能够编写 Perl 程序：存储 DNA 序列、拼接 DNA 片段、把 DNA 转录成 RNA、获取 DNA 的反向互补序列。
- 能够编写 Perl 程序：从文件中读取所需数据。
- 掌握 Perl 语言中数组的常见操作。
- 理解并能熟练应用标量上下文和列表上下文。

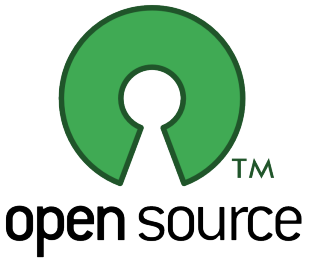
教学提纲

1. 举例说明拼接 DNA 片段的不同方法。
2. 举例说明 Perl 语言中双引号和单引号的异同。
3. 举例说明 Perl 语言中替换和翻译的函数及其语法。
4. 在 Perl 语言中如何从文件读取数据？
5. 举例说明数组的常见操作：初始化，头尾操作，反转，获取元素个数，等。
6. 举例说明 Perl 语言中的标量上下文和列表上下文。

下节预告

- 回顾 shell 的条件流程控制和迭代流程控制。
- 已经学习了读取文件，那么怎样写入文件呢？
- 回顾正则表达式的基本知识点。
- 如何在序列中查找基序？
- 如何计算序列中的核苷酸频率？

Powered by



TEX LATEX XETEX Beamer