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

第九章　限制酶图谱和正则表达式

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 12 月

教学提纲

教学提纲

已经学习

- Perl 语言
  - 正则表达式相关：模式匹配，字符替换
  - 操作符：数字操作符，字符串操作符
- 生物信息学
  - 处理 FASTA 格式的文件
  - 在 DNA 序列中查找基序

即将学习

- Perl 语言
  - 正则表达式的基本理念
  - 操作符的优先级
- 生物信息学
  - 用正则表达式表征酶切数据
  - 根据用户要求制作酶切图谱

已经学习

- Perl 语言
  - 正则表达式相关：模式匹配，字符替换
  - 操作符：数字操作符，字符串操作符
- 生物信息学
  - 处理 FASTA 格式的文件
  - 在 DNA 序列中查找基序

即将学习

- Perl 语言
  - 正则表达式的基本理念
  - 操作符的优先级
- 生物信息学
  - 用正则表达式表征酶切数据
  - 根据用户要求制作酶切图谱

教学提纲

教学提纲

用一句“话”/一个字符串描述：

- 1 ～ 6 号染色体
- 除 X、Y、M 以外的所有染色体
- 任意一个碱基/核苷酸
- *Bst*YI 的切割序列（RGATCY）
- `<A-X-[ST](2)-X(0,1)-{V}`
- ORF（开放阅读框）

实现方法

正则表达式！

用一句“话”/一个字符串描述：

- 1 ～ 6 号染色体
- 除 X、Y、M 以外的所有染色体
- 任意一个碱基/核苷酸
- *Bst*YI 的切割序列（RGATCY）
- `<A-X-[ST](2)-X(0,1)-{V}`
- ORF（开放阅读框）

实现方法

正则表达式！

正则表达式

正则表达式，又称正规表示式、正规表示法、正规表达式、规则表达式、常规表示法（Regular Expression，在代码中常简写为 regex、regexp 或 RE），计算机科学的一个概念。

正则表达式使用单个字符串来描述、匹配一系列符合某个句法规则的字符串。

在很多文本编辑器里，正则表达式通常被用来检索、替换那些符合某个模式的文本。

教学提纲

```
1 if( $dna =~ /CT[CGT]ACG/  ) {
2   print "I found the motif!!\n";
3 }
```

解析

- /CT[CGT]ACG/：完整的正则表达式
- //：正则表达式界定符
- ACGT：ACGT 四个字符/四种碱基本身
- [CGT]：C 或者 G 或者 T
- 基序（motif）：CTCACG 或者 CTGACG 或者 CTTACG

```
1 if( $dna =~ /CT[CGT]ACG/  ) {
2   print "I found the motif!!\n";
3 }
```

解析

- `/CT[CGT]ACG/`：完整的正则表达式
- `//`：正则表达式界定符
- `ACGT`：ACGT 四个字符/四种碱基本身
- `[CGT]`：C 或者 G 或者 T
- 基序（motif）：CTCACG 或者 CTGACG 或者 CTTACG


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

/([Ii]f|and)* [AC]+.(and)?/

"You might," the candid hero admitted; "though such obtuseness would certainly be phenomenal. Still, the event is *possible*. So I will just ask you to grant one more Hypothetical."

"Very good. I'm quite willing to grant it, as soon as you've written it down. We will call it

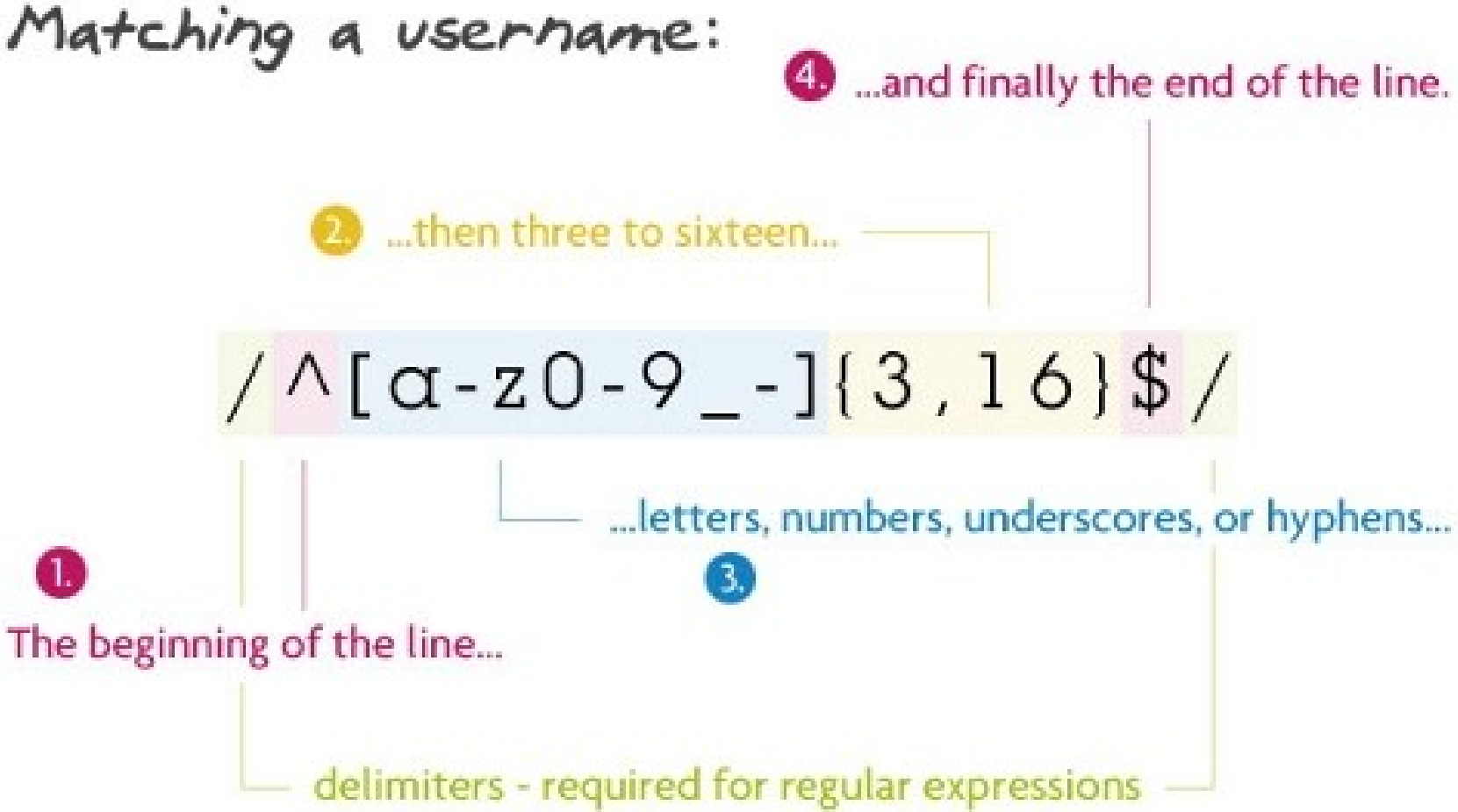
(D) If A and B and C are true, Z must be true.

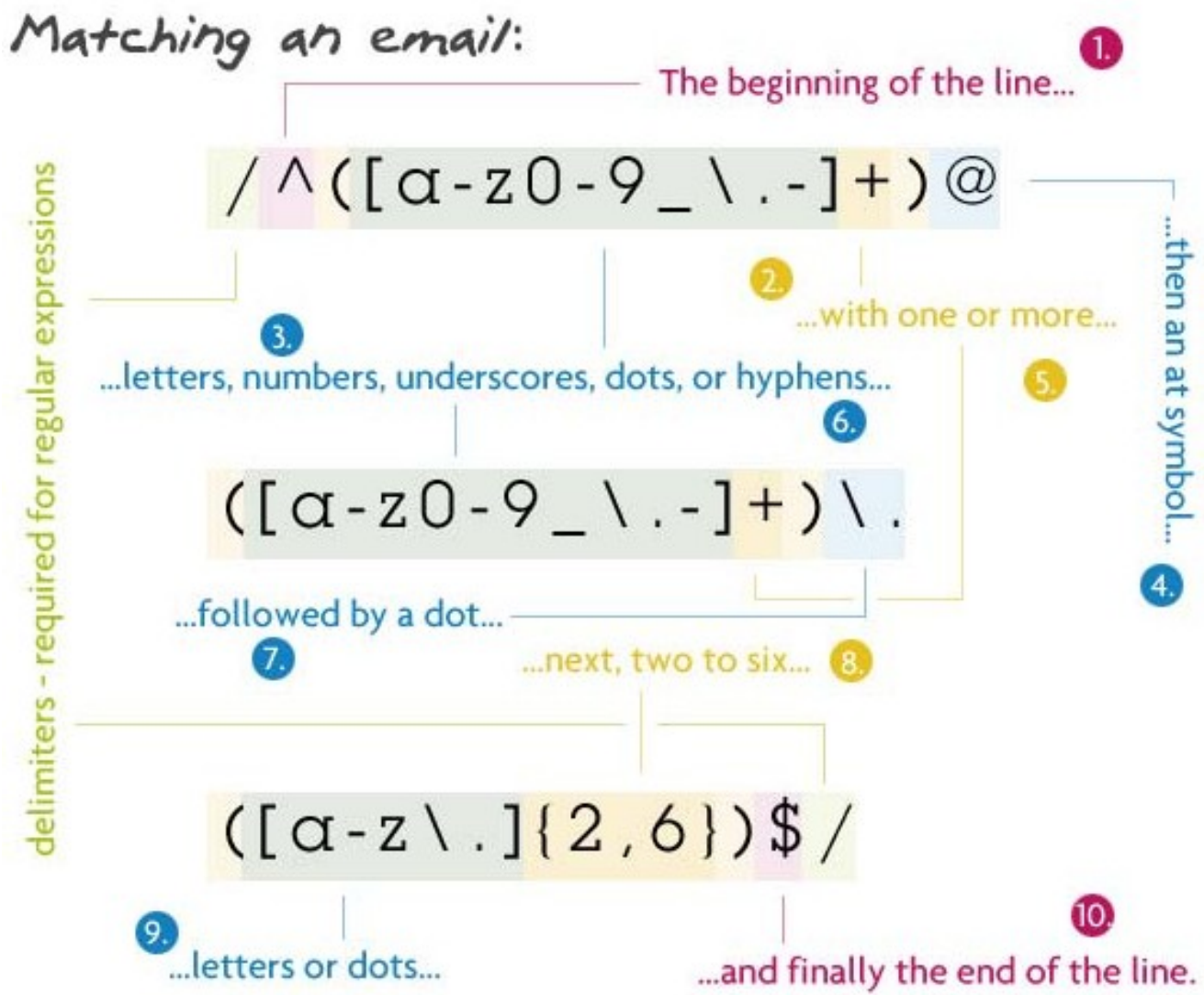
"Have you entered that in your note-book?"

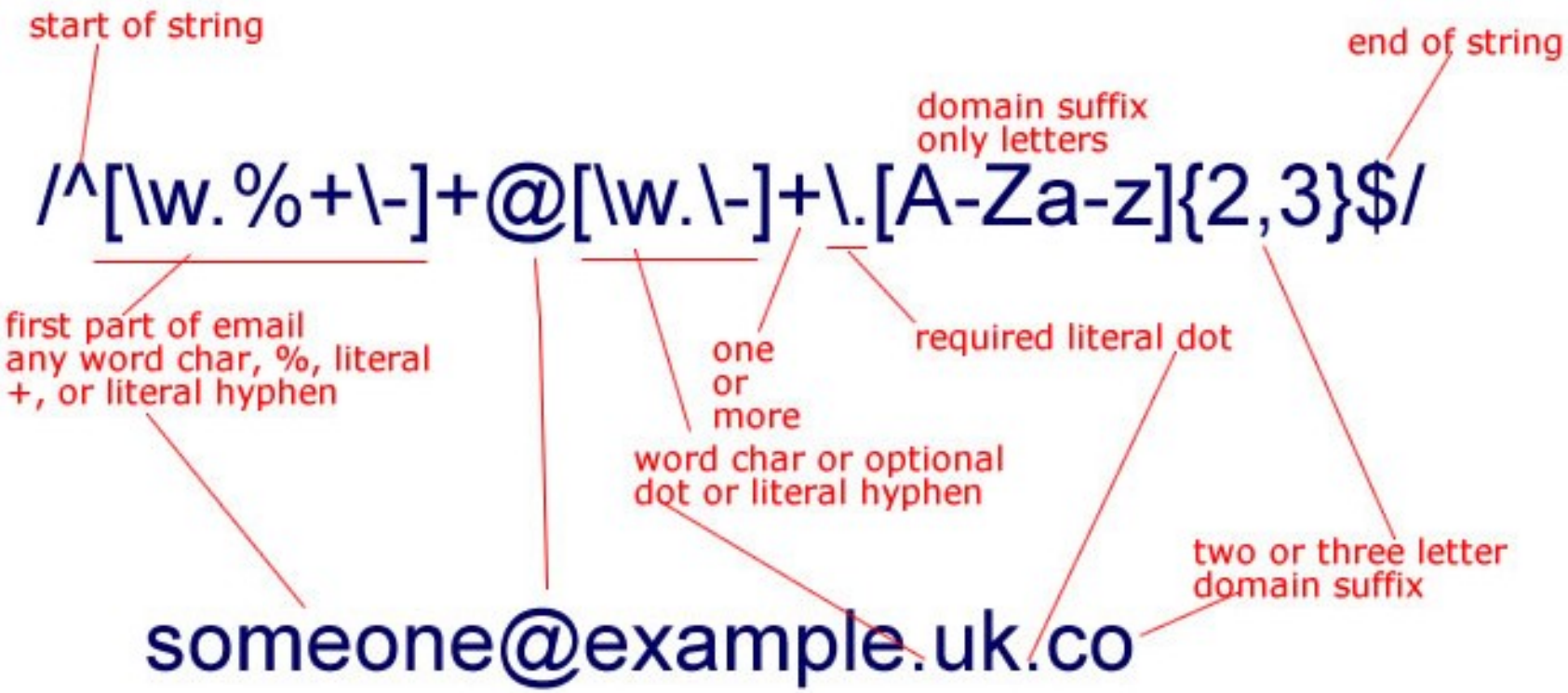
"I *have!*" Achilles joyfully exclaimed, as he ran the pencil into its sheath. "And at last we've got to the end of the ideal race-course! Now that you accept A and B and C and D, *of course* you accept Z."

"Do I?" said the Tortoise innocently. "Let's make that quite clear. I accept A and B and C and D. Suppose I *still* refuse to accept Z?"

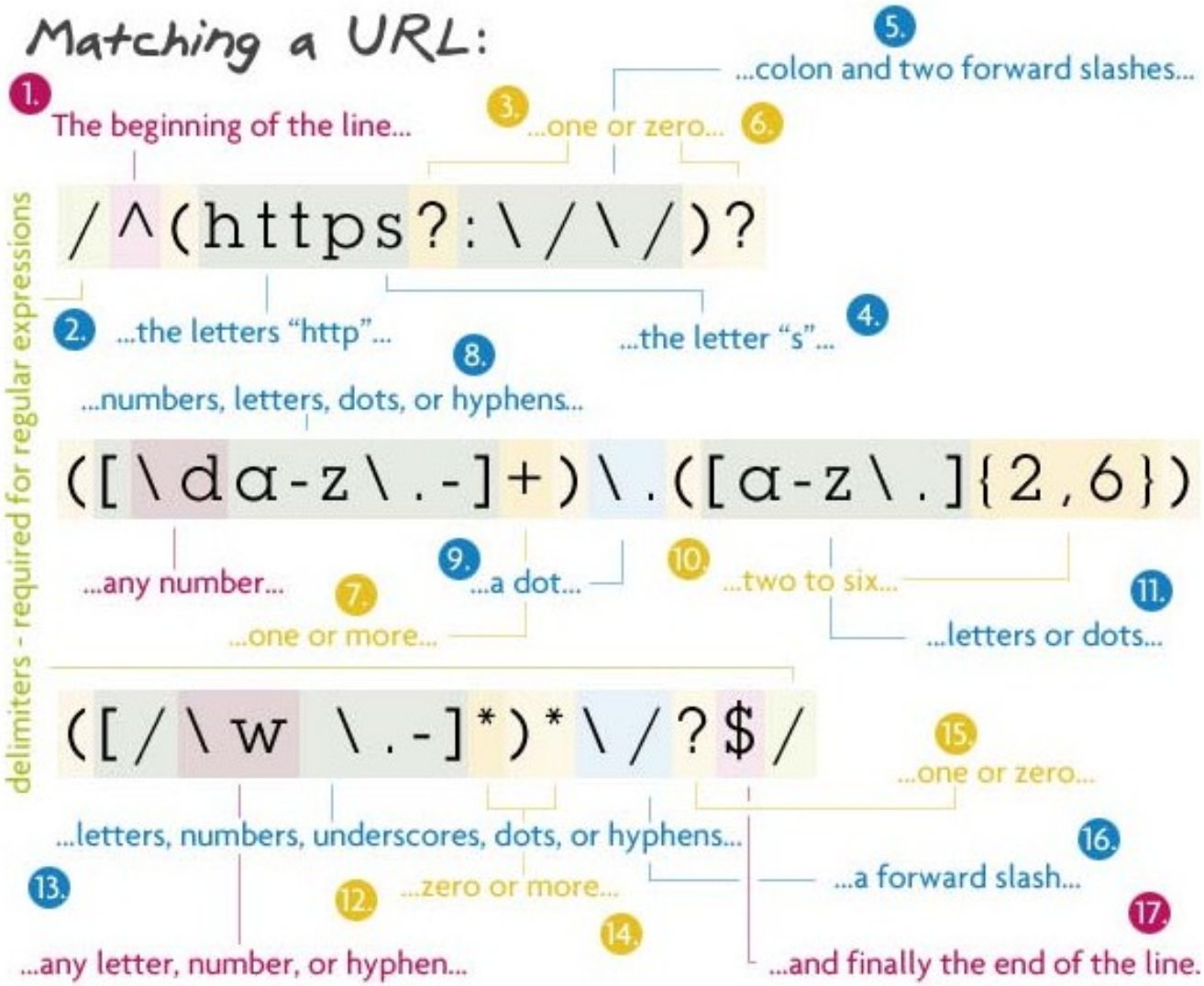
"Then Logic would take you be the throat, and *force* you to do it!" Achilles triumphantly replied. "Logic would tell you 'You can't help yourself. Now that you've accepted A and B and C and D, you MUST accept Z!' So you've no choice, you see."

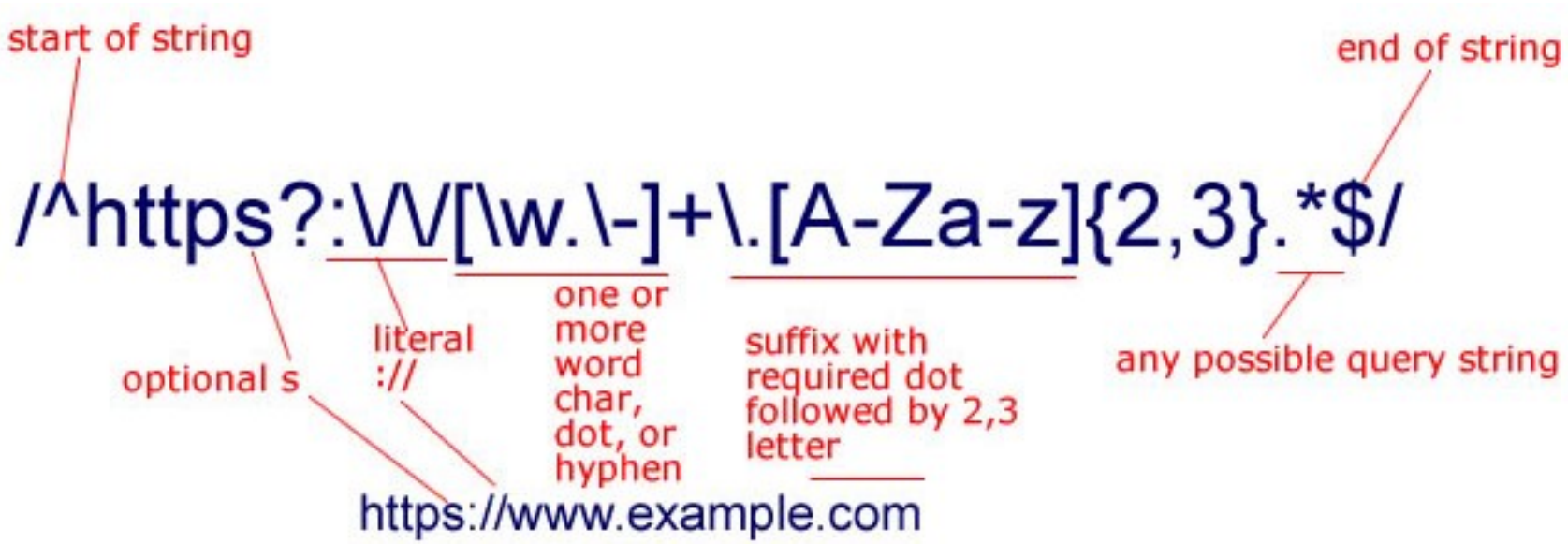

Matching a username:
4. ...and finally the end of the line.
2. ...then three to sixteen...
/^[a-z0-9_-]{3,16}$/
...letters, numbers, underscores, or hyphens...
3.
1.
The beginning of the line...
delimiters - required for regular expressions


Matching an email:
1. The beginning of the line...
/^([a-z0-9_\.-]+)@
2. ...with one or more...
3. ...letters, numbers, underscores, dots, or hyphens...
5.
6.
([a-z0-9_\.-]+)\.
...then an at symbol...
4.
...followed by a dot...
7.
...next, two to six...
8.
([a-z\.]{2,6})$/
delimiters - required for regular expressions
9. ...letters or dots...
10. ...and finally the end of the line.


start of string
end of string
domain suffix
only letters
/^[\w.%+\-]+@[\w.\-]+\.[A-Za-z]{2,3}$/
first part of email
any word char, %, literal
+, or literal hyphen
one
or
more
word char or optional
dot or literal hyphen
required literal dot
two or three letter
domain suffix
someone@example.uk.co

酶切图谱 | 正则表达式 | 实例 | 网址




start of string
end of string
/^https?:\/\/[\w.\-]+\.[A-Za-z]{2,3}.*$/
optional s
literal ://
one or more word char, dot, or hyphen
suffix with required dot followed by 2,3 letter
any possible query string
https://www.example.com

```
/^
(?:ftp|https?):\/\/
(?:
  (?:(?:[\w\.\-\+!$&'\(\)*\+,;=]|%[0-9a-f]{2})+:)*
  (?:[\w\.\-\+%!$&'\(\)*\+,;=]|%[0-9a-f]{2})+@
)?
(?:
  (?:[a-z0-9\-\.]|%[0-9a-f]{2})+
  |(?:\[(?:[0-9a-f]{0,4}:)*(?:[0-9a-f]{0,4})\])
)
(?::[0-9]+)?
(?:[\/|\?]
  (?:[\w#!:\.\?\+=&@$'~*,;\/\(\)\[\]\-]|%[0-9a-f]{
*)?
```

教学提纲

正则理论

正则表达式可以用形式化语言理论的方式来表达。

正则表达式由常量和算子组成，它们分别表示字符串的集合和在这些集合上的运算。

形式语言

在数学、逻辑和计算机科学中，形式语言（formal language）是用精确的数学或机器可处理的公式定义的语言。

正则理论

正则表达式可以用形式化语言理论的方式来表达。

正则表达式由常量和算子组成，它们分别表示字符串的集合和在这些集合上的运算。

形式语言

在数学、逻辑和计算机科学中，形式语言（formal language）是用精确的数学或机器可处理的公式定义的语言。

空集 $\varnothing$ 表示集合 $\varnothing$

空串 ε 表示集合 $\{\varepsilon\}$

文字字符 Σ 中的 a 表示集合 {a}

串接 RS 表示集合 $\{\alpha\beta \mid \alpha \in R, \beta \in S\}$。例如：

```
{"ab", "c"}{"d", "ef"}
= {"abd", "abef", "cd", "cef"}。
```

选择 R|S 表示 R 和 S 的并集。例如：

```
{"ab", "c"}|{"ab", "d", "ef"}
={"ab", "c", "d", "ef"}。
```

Kleene 星号 R* 表示包含 ε 并且闭合在字符串串接下的 R 的最小超集。这是通过 R 中的零或多个字符串的串接得到的所有字符串的集合。例如：

```
{"ab", "c"}*
= {ε,"ab","c","abab","abc","cab","cc","ccc",
```

优先级定义

为了避免括号，假定 Kleene 星号有最高优先级，接着是串接，接着是并集。如果没有歧义则可以省略括号。

优先级实例

- `(ab)c` 可以写为 `abc`
- `a|(b(c*))` 可以写为 `a|bc*`。

优先级定义

为了避免括号，假定 Kleene 星号有最高优先级，接着是串接，接着是并集。如果没有歧义则可以省略括号。

优先级实例

- `(ab)c` 可以写为 `abc`
- `a|(b(c*))` 可以写为 `a|bc*`。

- a|b* 表示 {a, ε, b, bb, bbb, ...}。
- (a|b)* 表示由包括空串、任意数目个 a 或 b 字符组成的所有字符串的集合。
- ab*(c|ε) 表示开始于一个 a 接着零或多个 b 和最终可选的一个 c 的字符串的集合。

一个正则表达式通常被称为一个模式（pattern），为用来描述或者匹配一系列符合某个句法规则的字符串。

Handel、Händel 和 Haendel 这三个字符串，都可以由“H(a|ä|ae)ndel”这个模式来描述。

DNA 和 RNA 这两个字符串，都可以由“(D|R)NA”这个模式来描述。

选择

|（竖直分隔符）代表选择。例如“gray|grey”可以匹配 grey 或 gray。

数量限定

某个字符后的数量限定符用来限定前面这个字符允许出现的个数。最常见的数量限定符包括“+”、“?”和“*”（不加数量限定则代表出现一次且仅出现一次）：

- +（加号）代表前面的字符必须至少出现一次（1 次、或多次）。例如，“goo+gle”可以匹配 google、gooogle、goooogle 等。
- ?（问号）代表前面的字符最多只可以出现一次（0 次、或 1 次）。例如，“colou?r”可以匹配 color 或者 colour。
- *（星号）代表前面的字符可以不出现，也可以出现一次或者多次（0 次、或 1 次、或多次）。例如，“0*42”可以匹配 42、042、0042、00042 等。

匹配

圆括号可以用来定义操作符的范围和优先度。例如，“gr(a|e)y”等价于“gray|grey”，“(grand)?father”匹配 father 和 grandfather。

- 基本语法可以自由组合，因此，“H(ae?|ä)ndel”和“H(a|ae|ä)ndel”是相同的。
- 精确的语法可能因不同的工具（sed，grep 等）或程序（Perl，Python 等）而异。

Regular Expression Examples

Let Σ = {a, b}.

a = {a}
ab = {ab}
a | b = {a, b}
a^+ = {a, aa, aaa, ... }

ab* represents the set of strings having a single a followed by zero or more occurrences of b.
That is, it's {a, ab, abb, abbb, ... }

a (b | c) = {ab, ac}

(a | b) (c | d) = {ac, ad, bc, bd}
aa* = a^+ = {a, aa, aaa, ... }

正则表达式

- `/(abc|def)z*x/`
- `ACG.*GCA`

解析

- abc 或者 def 后面跟着 0 个或者多个 z，最后是一个 x
- ACG 后面跟着 0 个或者多个字符，最后是 GCA

实例

- abcx, abczx, abczzx, defx, defzx, defzzzzzx, ...
- 起始于 ACG、终止于 GCA 的 DNA 序列

正则表达式

- `/(abc|def)z*x/`
- `ACG.*GCA`

解析

- abc 或者 def 后面跟着 0 个或者多个 z，最后是一个 x
- ACG 后面跟着 0 个或者多个字符，最后是 GCA

实例

- abcx, abczx, abczzx, defx, defzx, defzzzzzx, ...
- 起始于 ACG、终止于 GCA 的 DNA 序列

正则表达式

- `/(abc|def)z*x/`
- `ACG.*GCA`

解析

- abc 或者 def 后面跟着 0 个或者多个 z，最后是一个 x
- ACG 后面跟着 0 个或者多个字符，最后是 GCA

实例

- abcx, abczx, abczzx, defx, defzx, defzzzzzx, ...
- 起始于 ACG、终止于 GCA 的 DNA 序列

正则表达式

- `/(abc|def)z*x/`
- `ACG.*GCA`

解析

- abc 或者 def 后面跟着 0 个或者多个 z，最后是一个 x
- ACG 后面跟着 0 个或者多个字符，最后是 GCA

实例

- abcx, abczx, abczzx, defx, defzx, defzzzzzx, ...
- 起始于 ACG、终止于 GCA 的 DNA 序列

元字符（metacharacter）

正则表达式语言由两种基本字符类型组成：原义（正常）文本字符和元字符。

元字符是一个或一组代替一个或多个字符的字符。

元字符是在正则表达式中具有特殊意义的专用字符，用来规定其前导字符（即位于元字符前面的字符）在目标对象中的出现模式。它使正则表达式具有处理能力。

字符	描述
\	将下一个字符标记为一个特殊字符、或一个原义字符、或一个向后引用、或一个八进制转义符。例如，“ n ”匹配字符“ n ”。“ \n ”匹配一个换行符。序列“ \\ ”匹配“ \ ”而“ \(”则匹配“ (”。
^	匹配输入字符串的开始位置。如果设置了RegExp对象的Multiline属性，^也匹配“ \n ”或“ \r ”之后的位置。
$	匹配输入字符串的结束位置。如果设置了RegExp对象的Multiline属性，$也匹配“ \n ”或“ \r ”之前的位置。
*	匹配前面的子表达式零次或多次。例如，zo*能匹配“ z ”、“ zo ”以及“ zoo ”。*等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，“ zo+ ”能匹配“ zo ”以及“ zoo ”，但不能匹配“ z ”。+等价于{1,}。
?	匹配前面的子表达式零次或一次。例如，“ do(es)? ”可以匹配“ do ”或“ does ”中的“ do ”。?等价于{0,1}。
{n}	n是一个非负整数。匹配确定的n次。例如，“ o{2} ”不能匹配“ Bob ”中的“ o ”，但是能匹配“ food ”中的两个o。
{n,}	n是一个非负整数。至少匹配n次。例如，“ o{2,} ”不能匹配“ Bob ”中的“ o ”，但能匹配“ foooood ”中的所有o。“ o{1,} ”等价于“ o+ ”。“ o{0,} ”则等价于“ o* ”。
{n,m}	m和n均为非负整数，其中$n<=m$。最少匹配n次且最多匹配m次。例如，“ o{1,3} ”将匹配“ fooooood ”中的前三个o。“ o{0,1} ”等价于“ o? ”。请注意在逗号和两个数之间不能有空格。
?	当该字符紧跟在任何一个其他限制符（*,+,?，{n}，{n,}，{n,m}）后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串“ oooo ”，“ o+? ”将匹配单个“ o ”，而“ o+ ”将匹配所有“ o ”。
.	匹配除“ \n ”之外的任何单个字符。要匹配包括“ \n ”在内的任何字符，请使用像“ (.|\n) ”的模式。

(pattern)	匹配pattern并获取这一匹配的子字符串。该子字符串用于向后引用。所获取的匹配可以从产生的Matches集合得到，在VBScript中使用SubMatches集合，在JScript中则使用$0...$9属性。要匹配圆括号字符，请使用“ \(”或“ \) ”。
(?:pattern)	匹配pattern但不获取匹配的子字符串，也就是说这是一个非获取匹配，不存储匹配的子字符串用于向后引用。这在使用或字符“ (|) ”来组合一个模式的各个部分是很有用。例如“ industr(?:y|ies) ”就是一个比“ industry|industries ”更简略的表达式。
(?=pattern)	正向肯定预查，在任何匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如，“ Windows(?=95|98|NT|2000) ”能匹配“ Windows2000 ”中的“ Windows ”，但不能匹配“ Windows3.1 ”中的“ Windows ”。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。
(?!pattern)	正向否定预查，在任何不匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如“ Windows(?!95|98|NT|2000) ”能匹配“ Windows3.1 ”中的“ Windows ”，但不能匹配“ Windows2000 ”中的“ Windows ”。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始
(?<=pattern)	反向肯定预查，与正向肯定预查类似，只是方向相反。例如，“ (?<=95|98|NT|2000)Windows ”能匹配“ 2000Windows ”中的“ Windows ”，但不能匹配“ 3.1Windows ”中的“ Windows ”。
(?<!pattern)	反向否定预查，与正向否定预查类似，只是方向相反。例如“ (?<!95|98|NT|2000)Windows ”能匹配“ 3.1Windows ”中的“ Windows ”，但不能匹配“ 2000Windows ”中的“ Windows ”。

x|y	匹配x或y。例如，“z|food”能匹配“z”或“food”。“(z|f)ood”则匹配“zood”或“food”。
[xyz]	字符集合（character class）。匹配所包含的任意一个字符。例如，“[abc]”可以匹配“plain”中的“a”。特殊字符仅有反斜线\保持特殊含义，用于转义字符。其它特殊字符如星号、加号、各种括号等均作为普通字符。脱字符^如果出现在首位则表示负值字符集合；如果出现在字符串中间就仅作为普通字符。连字符 - 如果出现在字符串中间表示字符范围描述；如果如果出现在首位则仅作为普通字符。
[^xyz]	排除型（negate）字符集合。匹配未列出的任意字符。例如，“[^abc]”可以匹配“plain”中的“plin”。
[a-z]	字符范围。匹配指定范围内的任意字符。例如，“[a-z]”可以匹配“a”到“z”范围内的任意小写字母字符。
[^a-z]	排除型的字符范围。匹配任何不在指定范围内的任意字符。例如，“[^a-z]”可以匹配任何不在“a”到“z”范围内的任意字符。
\b	匹配一个单词边界，也就是指单词和空格间的位置。例如，“er\b”可以匹配“never”中的“er”，但不能匹配“verb”中的“er”。
\B	匹配非单词边界。“er\B”能匹配“verb”中的“er”，但不能匹配“never”中的“er”。

\cx	匹配由x指明的控制字符。例如，\cM匹配一个Control-M或回车符。x的值必须为A-Z或a-z之一。否则，将c视为一个原义的“c”字符。
\d	匹配一个数字字符。等价于[0-9]。
\D	匹配一个非数字字符。等价于[^0-9]。
\f	匹配一个换页符。等价于\x0c和\cL。
\n	匹配一个换行符。等价于\x0a和\cJ。
\r	匹配一个回车符。等价于\x0d和\cM。
\s	匹配任何空白字符，包括空格、制表符、换页符等等。等价于[\f\n\r\t\v]。
\S	匹配任何非空白字符。等价于[^ \f\n\r\t\v]。
\t	匹配一个制表符。等价于\x09和\cI。
\v	匹配一个垂直制表符。等价于\x0b和\cK。
\w	匹配包括下划线的任何单词字符。等价于“[A-Za-z0-9_]”。
\W	匹配任何非单词字符。等价于“[^A-Za-z0-9_]”。

\x*n*	匹配*n*，其中*n*为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，“ \x41 ”匹配“ A ”。“ \x041 ”则等价于“ \x04&1 ”。正则表达式中可以使用ASCII编码。.
num	向后引用（back-reference）一个子字符串（substring），该子字符串与正则表达式的第num个用括号围起来的子表达式（subexpression）匹配。其中num是从1开始的正整数，其上限可能是99。例如：“ (.)\1 ”匹配两个连续的相同字符。
n	标识一个八进制转义值或一个向后引用。如果*n*之前至少*n*个获取的子表达式，则*n*为向后引用。否则，如果*n*为八进制数字（0-7），则*n*为一个八进制转义值。
nm	标识一个八进制转义值或一个向后引用。如果*nm*之前至少有*nm*个获得子表达式，则*nm*为向后引用。如果*nm*之前至少有*n*个获取，则*n*为一个后跟文字*m*的向后引用。如果前面的条件都不满足，若*n*和*m*均为八进制数字（0-7），则*nm*将匹配八进制转义值*nm*。
nml	如果*n*为八进制数字（0-3），且*m*和*l*均为八进制数字（0-7），则匹配八进制转义值*nml*。
\u*n*	匹配*n*，其中*n*是一个用四个十六进制数字表示的Unicode字符。例如，\u00A9匹配版权符号（©）。

优先权	符号
最高	\
高	( )、(?:)、(?=)、[ ]
中	*、+、?、{n}、{n,}、{m,n}
低	^、$、中介字符
最低	|

优先级

1. 最高等级：圆括号 `()`，用于分组和捕获；里面的东西比其他更有紧密性
2. 第二级：量词，星号 `*`、加号 `+`、问号 `?`、用花括号表示的量词 `{m,n}`；和它前面的条目紧密相连
3. 第三级：锚位和序列，`^`、`$`；单词里字母之间的紧密程度和锚位与字母之间的紧密程度是相同的
4. 第四级：择一竖线 `|`；把各种模式拆分成数个组件
5. 最低级别：原子，单独的字符、字符集 `\d`、反引用 `\1` 等；构成大多数基本的模式

基本原则

- 使用圆括号来分组（注意：圆括号同时也会有捕获的效果，因此尽可能使用非捕获圆括号来分组）

优先级

1. 最高等级：圆括号 `()`，用于分组和捕获；里面的东西比其他更有紧密性
2. 第二级：量词，星号 `*`、加号 `+`、问号 `?`、用花括号表示的量词 `{m,n}`；和它前面的条目紧密相连
3. 第三级：锚位和序列，`^`、`$`；单词里字母之间的紧密程度和锚位与字母之间的紧密程度是相同的
4. 第四级：择一竖线 `|`；把各种模式拆分成数个组件
5. 最低级别：原子，单独的字符、字符集 `\d`、反引用 `\1` 等；构成大多数基本的模式

基本原则

- 使用圆括号来分组（注意：圆括号同时也会有捕获的效果，因此尽可能使用非捕获圆括号来分组）

教学提纲

问题

- 匹配 1 ～ 6 号染色体
- 匹配除 X、Y、M 以外的所有染色体
- 匹配任意一个碱基/核苷酸
- *Bst*YI 的切割序列（RGATCY）
- `<A-X-[ST](2)-X(0,1)-{V}`

正则

- `/chr[1-6]/`
- `/chr[^xymXYM]/`
- `/[ACGTU]/`
- `/[AG]GATC[TC]/`
- `/^A.[ST]{2}.?[^V]/`

问题

- 匹配 1 ～ 6 号染色体
- 匹配除 X、Y、M 以外的所有染色体
- 匹配任意一个碱基/核苷酸
- *Bst*YI 的切割序列（RGATCY）
- <A-X-[ST](2)-X(0,1)-{V}>

正则

- `/chr[1-6]/`
- `/chr[^xymXYM]/`
- `/[ACGTU]/`
- `/[AG]GATC[TC]/`
- `/^A.[ST]{2}.?[^V]/`

问题

- 匹配 1 ～ 6 号染色体
- 匹配除 X、Y、M 以外的所有染色体
- 匹配任意一个碱基/核苷酸
- *Bst*YI 的切割序列（RGATCY）
- `<A-x-[ST](2)-x(0,1)-{V}`

正则

- `/chr[1-6]/`
- `/chr[^xymXYM]/`
- `/[ACGTU]/`
- `/[AG]GATC[TC]/`
- `/^A.[ST]{2}.?[^V]/`

问题

- 匹配 1 ～ 6 号染色体
- 匹配除 X、Y、M 以外的所有染色体
- 匹配任意一个碱基/核苷酸
- *Bst*YI 的切割序列（RGATCY）
- <A-X-[ST](2)-X(0,1)-{V}

正则

- `/chr[1-6]/`
- `/chr[^xymXYM]/`
- `/[ACGTU]/`
- `/[AG]GATC[TC]/`
- `/^A.[ST]{2}.?[^V]/`

问题

- 匹配 1 ～ 6 号染色体
- 匹配除 X、Y、M 以外的所有染色体
- 匹配任意一个碱基/核苷酸
- *Bst*YI 的切割序列（RGATCY）
- <A-X-[ST](2)-X(0,1)-{V}

正则

- `/chr[1-6]/`
- `/chr[^xymXYM]/`
- `/[ACGTU]/`
- `/[AG]GATC[TC]/`
- `/^A.[ST]{2}.?[^V]/`

问题

- 匹配 1 ～ 6 号染色体
- 匹配除 X、Y、M 以外的所有染色体
- 匹配任意一个碱基/核苷酸
- *Bst*YI 的切割序列（RGATCY）
- <A-X-[ST](2)-X(0,1)-{V}

正则

- `/chr[1-6]/`
- `/chr[^xymXYM]/`
- `/[ACGTU]/`
- `/[AG]GATC[TC]/`
- `/^A.[ST]{2}.?[^V]/`

问题

- 匹配 1 ～ 6 号染色体
- 匹配除 X、Y、M 以外的所有染色体
- 匹配任意一个碱基/核苷酸
- *Bst*YI 的切割序列（RGATCY）
- <A-X-[ST](2)-X(0,1)-{V}

正则

- `/chr[1-6]/`
- `/chr[^xymXYM]/`
- `/[ACGTU]/`
- `/[AG]GATC[TC]/`
- `/^A.[ST]{2}.?[^V]/`

问题

- 匹配 1 ～ 6 号染色体
- 匹配除 X、Y、M 以外的所有染色体
- 匹配任意一个碱基/核苷酸
- *Bst*YI 的切割序列（RGATCY）
- `<A-X-[ST](2)-X(0,1)-{V}`

正则

- `/chr[1-6]/`
- `/chr[^xymXYM]/`
- `/[ACGTU]/`
- `/[AG]GATC[TC]/`
- `/^A.[ST]{2}.?[^V]/`

问题

- 匹配 1 ～ 6 号染色体
- 匹配除 X、Y、M 以外的所有染色体
- 匹配任意一个碱基/核苷酸
- *Bst*YI 的切割序列（RGATCY）
- `<A-X-[ST](2)-X(0,1)-{V}`

正则

- `/chr[1-6]/`
- `/chr[^xymXYM]/`
- `/[ACGTU]/`
- `/[AG]GATC[TC]/`
- `/^A.[ST]{2}.?[^V]/`

问题

- 匹配 1 ～ 6 号染色体
- 匹配除 X、Y、M 以外的所有染色体
- 匹配任意一个碱基/核苷酸
- *Bst*YI 的切割序列（RGATCY）
- `<A-X-[ST](2)-X(0,1)-{V}`

正则

- `/chr[1-6]/`
- `/chr[^xymXYM]/`
- `/[ACGTU]/`
- `/[AG]GATC[TC]/`
- `/^A.[ST]{2}.?[^V]/`

问题

用正则表达式表征 ORF

要求

- ORF 起始于起始密码子（ATG）
- ORF 终止于终止密码子（TAA，TAG，TGA）
- 起始和终止密码子之间的碱基个数是 3 的倍数
- ORF 至少编码 30 个氨基酸
- 序列可能使用的是 U 而非 T
- 序列可能使用小写字母（甚至大小写混合）
- 序列中可能不止一个 ORF

参考（正确？）

```
/A[TU]G([ACGTU]{3}){29,}[TU](AA|AG|GA)/ig
```

问题

用正则表达式表征 ORF

要求

- ORF 起始于起始密码子（ATG）
- ORF 终止于终止密码子（TAA，TAG，TGA）
- 起始和终止密码子之间的碱基个数是 3 的倍数
- ORF 至少编码 30 个氨基酸
- 序列可能使用的是 U 而非 T
- 序列可能使用小写字母（甚至大小写混合）
- 序列中可能不止一个 ORF

参考（正确？）

```
/A[TU]G([ACGTU]{3}){29,}[TU](AA|AG|GA)/ig
```

问题

用正则表达式表征 ORF

要求

- ORF 起始于起始密码子（ATG）
- ORF 终止于终止密码子（TAA，TAG，TGA）
- 起始和终止密码子之间的碱基个数是 3 的倍数
- ORF 至少编码 30 个氨基酸
- 序列可能使用的是 U 而非 T
- 序列可能使用小写字母（甚至大小写混合）
- 序列中可能不止一个 ORF

参考（正确？）

```
/A[TU]G([ACGTU]{3}){29,}[TU](AA|AG|GA)/ig
```

教学提纲

教学提纲

限制酶

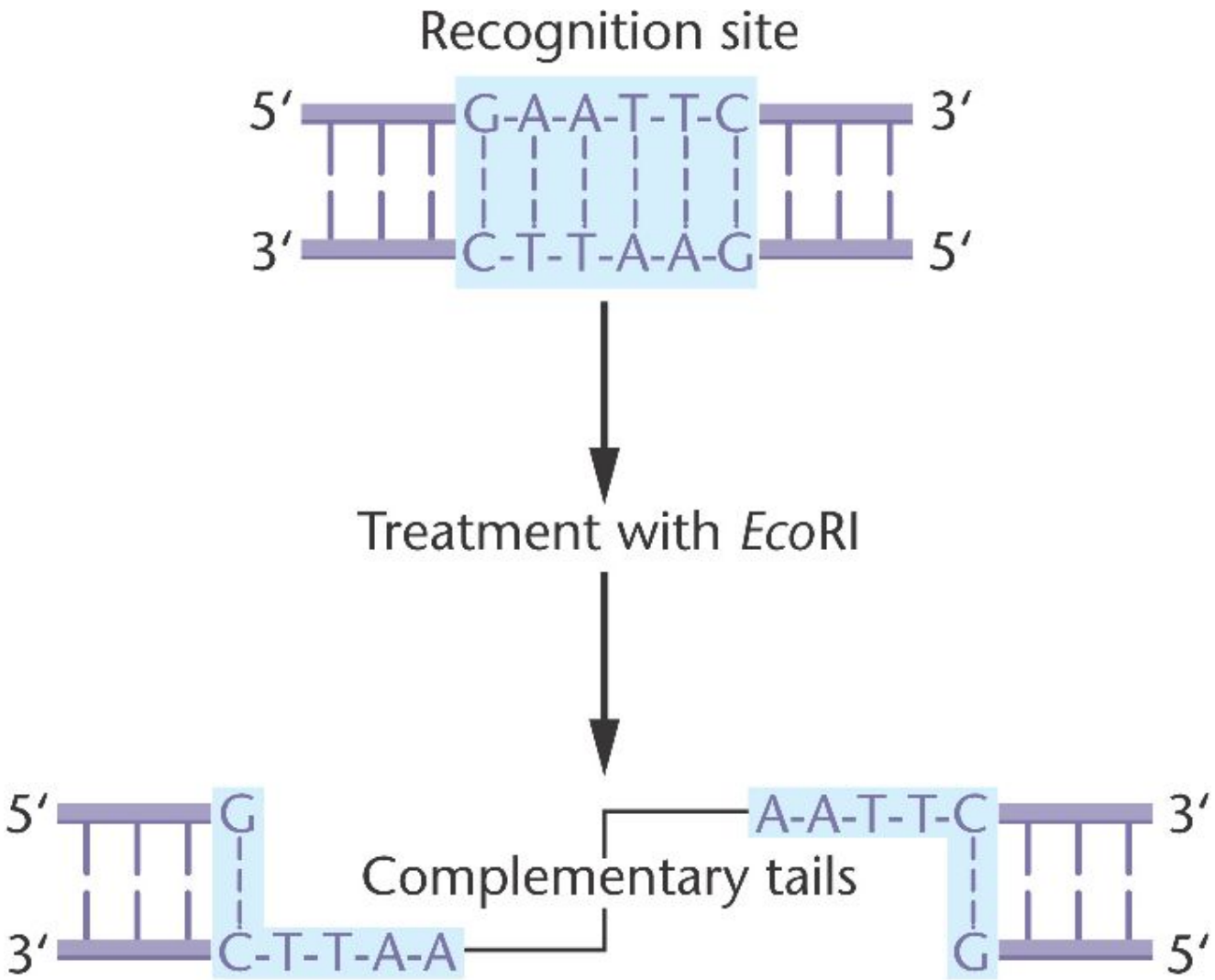
限制酶（restriction enzyme）又称限制内切酶或限制性内切酶（restriction endonuclease），全称限制性核酸内切酶，是一种能将双链 DNA 切开的酶。

切割方法是将糖类分子与磷酸之间的键切断，进而于两条 DNA 链上各产生一个切口，且不破坏核苷酸与碱基。切割形式有两种，分别是可产生具有突出单链 DNA 的黏状末端，以及末端平整无凸起的平滑末端。由于断开的 DNA 片段可由 DNA 连接酶黏合，因此染色体或 DNA 上不同的限制片段，可以经由剪接作用而结合在一起。

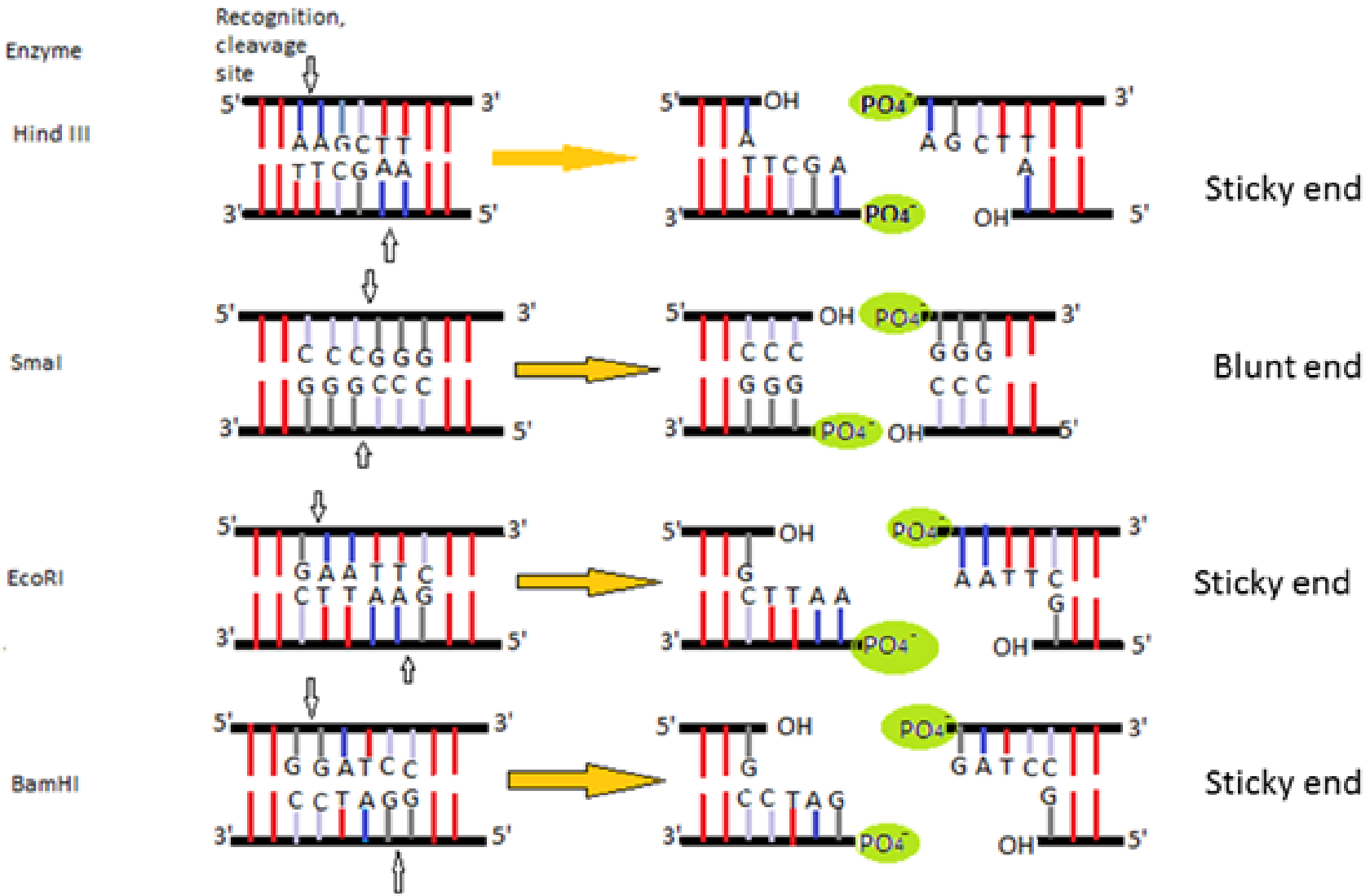
限制酶在分子生物学与遗传工程领域有广泛的应用。

根据限制酶的结构、辅因子的需求切位与作用方式，可将限制酶分为三种类型，分别是第一型（Type I）、第二型（Type II）及第三型（Type III）。

- 第一型限制酶：同时具有修饰（modification）及识别切割（restriction）的作用；另有识别（recognize）DNA 上特定碱基序列的能力，通常其切割位（cleavage site）距离识别位（recognition site）可达数千个碱基之远，并不能准确定位切割位点，所以并不常用。例如：*Eco*B、*Eco*K。
- 第二型限制酶：只具有识别切割的作用，修饰作用由其他酶进行。所识别的位置多为短的回文序列（palindrome sequence）；所剪切的碱基序列通常即为所识别的序列。是遗传工程上实用性较高的限制酶种类。例如：*Eco*RI、*Hin*dIII。
- 第三型限制酶：与第一型限制酶类似，同时具有修饰及识别切割的作用。可识别短的不对称序列，切割位与识别序列约距 24-26 个碱基对，并不能准确定位切割位点，所以并不常用。例如：*Eco*P1、*Hin*fIII。


Recognition site
5′
G-A-A-T-T-C
3′
3′
C-T-T-A-A-G
5′
Treatment with EcoRI
5′
G
3′
C-T-T-A-A
Complementary tails
A-A-T-T-C
3′
G
5′

酶切图谱 | 限制酶 | 实例


Enzyme
Recognition, cleavage site
Hind III
AAGCTT
TTCGAA
Sticky end
SmaI
CCCGGG
GGGCCC
Blunt end
EcoRI
GAATTC
CTTAAG
Sticky end
BamHI
GGATCC
CCTAGG
Sticky end
5'
3'
OH
PO4

教学提纲

编写一个在实验室中非常有用的程序

- 在 DNA 序列中查找限制酶
- 报告限制酶酶切图谱
- 找到该限制酶在 DNA 序列上的精确位点

DNA 序列

- 让用户指定序列（直接输入，指定文件，……）
- **从 DNA 样本文件中读入序列**

限制酶数据

- **限制酶数据库（REBASE）**
- 网址：http://rebase.neb.com/rebase/rebase.html

DNA 序列

- 让用户指定序列（直接输入，指定文件，……）
- **从 DNA 样本文件中读入序列**

限制酶数据

- **限制酶数据库（REBASE）**
- 网址：http://rebase.neb.com/rebase/rebase.html

用正则表达式表征限制酶

- **把数据库语言翻译成正则表达式**

存储限制酶数据

- **散列（酶的名字为键，酶切位点为值），** 或者 DBM 文件
- **存储几个限制酶，** 或者存储所有限制酶

接受用户的查询

- **向用户询问限制酶的名字（更加人性化）**
- 让用户直接输入正则表达式（高定制性）

用正则表达式表征限制酶

- **把数据库语言翻译成正则表达式**

存储限制酶数据

- **散列（酶的名字为键，酶切位点为值）**，或者 DBM 文件
- **存储几个限制酶**，或者存储所有限制酶

接受用户的查询

- 向用户询问限制酶的名字（更加人性化）
- 让用户直接输入正则表达式（高定制性）

用正则表达式表征限制酶

- **把数据库语言翻译成正则表达式**

存储限制酶数据

- **散列（酶的名字为键，酶切位点为值）**，或者 DBM 文件
- **存储几个限制酶**，或者存储所有限制酶

接受用户的查询

- **向用户询问限制酶的名字（更加人性化）**
- 让用户直接输入正则表达式（高定制性）

报告酶切图谱

- **输出位置及在该位置找到的限制酶的名字（易于流程化、进一步处理）**
- 图形化展示：输出序列，在序列上用线标明限制酶的位置（更加直观）

- 把限制酶数据翻译成正则表达式
- 把限制酶存储到散列中（键为名字，值为用正则表达式表征的切割序列）
- 从文件中读入 DNA 序列数据
- 提示用户输入限制酶的名字
- 查找正则表达式所有的出现及出现位置
- 输出找到的位置列表

已经学习

- 散列的使用
- 从文件中读取数据（读取 FASTA 文件中的序列）
- 捕获用户的键盘输入
- 模式匹配

尚需解决

- 把限制酶数据翻译成正则表达式
- 捕获模式匹配的位置信息

已经学习

- 散列的使用
- 从文件中读取数据（读取 FASTA 文件中的序列）
- 捕获用户的键盘输入
- 模式匹配

尚需解决

- 把限制酶数据翻译成正则表达式
- 捕获模式匹配的位置信息

教学提纲

酶切图谱 | 限制酶数据 | REBASE

REBASE®

The Restriction Enzyme Database

http://rebase.neb.com - CITING REBASE...

Choose search category and enter keyword:

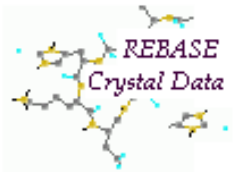
use percent sign as wildcard and quotes around phrases

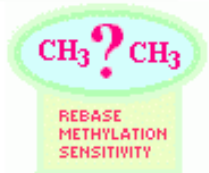
author starting with

Go Clear

enzyme name or #:

Go Clear





酶切图谱 | 限制酶数据 | REBASE

1. Commercial Sources (commdata)
2. GCG (also SEQTools, GeneDoc, and BioEdit) (gcg)
3. GCG (with references) (gcgenz,gcgref)
4. Nucleic Acids Research (nar)
5. Prototypes (proto)
6. Type II (Oligo/Primer Premier/SimVector) (type2,type2ref)
7. Type II (with isoschizomers) (gtype2,type2ref)
8. Type II (tabbed output) (itype2,type2ref)
9. All Enzymes with ref list at bottom (Redasoft Plasmid) (allenz)
10. All Enzymes (each with its refs) (withref)
11. All Enzymes (parsed references) (parsrefs)
12. All Enzymes (sorted by microorganism) (orgref)
13. IGSuite (bionet)
14. Alphazymes (azymes)
15. Cutzymes (cutzymes)
16. Staden (staden)
17. DNA Strider, DNAssist, Sequence Quickie-Calc, Serial Cloner (strider)
18. DNA Strider (commercial), DNAssist, Sequence Quickie-Calc, Serial Cloner (striderc)
19. MacVector, Vector NTI, PC/Gene (Bairoch) Format (consistent with PROSITE, SWISS-PROT, EMBL, ENZYME, ECD, EPD, HAEMB data banks) (bairoch)
20. DNASIS (dnasis)
21. Published Refs (pubrefs, pubrefsa)

bionet

An alphabetical listing of Type II restriction enzymes, their prototypes, and recognition sequences (with cut sites indicated).

EXAMPLE:

enzyme name (prototype)	recognition sequence
AaaI (XmaIII)	C^GGCCG
AacI (BamHI)	GGATCC
AatI (StuI)	AGG^CCT
AatII	GACGT^C
AcaII (BamHI)	GGATCC

```
REBASE version 510                                              bionet.510

    =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
    REBASE, The Restriction Enzyme Database   http://rebase.neb.com
    Copyright (c)  Dr. Richard J. Roberts, 2015.   All rights reserved.
    =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

Rich Roberts                                                    Sep 29 2015

AaaI (XmaIII)                     C^GGCCG
AacLI (BamHI)                     GGATCC
AaeI (BamHI)                      GGATCC
AagI (ClaI)                       AT^CGAT
AanI (PsiI)                       TTA^TAA
AaqI (ApaLI)                      GTGCAC
AarI                              CACCTGCNNNN^
AarI                              ^NNNNNNNNGCAGGTG
AasI (DrdI)                       GACNNNN^NNGTC
AatI (StuI)                       AGG^CCT
AatII                             GACGT^C
AauI (Bsp1407I)                   T^GTACA
AbaI (BclI)                       T^GATCA
Aba6411II                         CRRTAAG
```

任务

- 读入 bionet 这个文件
- 获取每一个酶的名字和识别位点
- （简化问题）把小括号中的限制酶的名字丢掉

```
1  Discard header lines
2
3  For each data line:
4
5    remove parenthesized names, for simplicity's
     sake
6
7    get and store the name and the recognition site
8
9    Translate the recognition sites to regular
     expressions
10     --but keep the recognition site, for printing
     out results
11 }
12
13 return the names, recognition sites, and the
     regular expressions
```

```
1 # Discard header lines
2 # This keeps reading lines, up to a line
    containing "Rich Roberts"
3 foreach line
4   if /Rich Roberts/
5     break out of the foreach loop
6 }
```

```
For each data line:

  # Split the two or three (if there's a parenthesized name) fields
  @fields = split( " ", $_ );
  # Get and store the name and the recognition site
  $name = shift @fields;
  $site = pop @fields;

  # Translate the recognition sites to regular expressions
    --but keep the recognition site, for printing out results
}

return the names, recognition sites, and the regular expressions
```

```perl
# 提取用空白分隔的单词，保存到数组
# 处理的是存储在特殊变量$_中的行
@fields = split( " ", $_ );

# 提取数组中的第一个元素
$name = shift @fields;

# 提取数组中的最后一个元素
$site = pop @fields;
```

```
REBASE version 510                                              bionet.510

    =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
    REBASE, The Restriction Enzyme Database   http://rebase.neb.com
    Copyright (c)  Dr. Richard J. Roberts, 2015.   All rights reserved.
    =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

Rich Roberts                                                    Sep 29 2015

AaaI (XmaIII)                     C^GGCCG
AacLI (BamHI)                     GGATCC
AaeI (BamHI)                      GGATCC
AagI (ClaI)                       AT^CGAT
AanI (PsiI)                       TTA^TAA
AaqI (ApaLI)                      GTGCAC
AarI                              CACCTGCNNNN^
AarI                              ^NNNNNNNNGCAGGTG
AasI (DrdI)                       GACNNNN^NNGTC
AatI (StuI)                       AGG^CCT
AatII                             GACGT^C
AauI (Bsp1407I)                   T^GTACA
AbaI (BclI)                       T^GATCA
Aba6411II                         CRRTAAG
```

```
1 # Example 9-1 Translate IUB ambiguity codes
    to regular expressions
2 # IUB_to_regexp
3 #
4 # A subroutine that, given a sequence with
    IUB ambiguity codes,
5 # outputs a translation with IUB codes
    changed to regular expressions
```

```
7  # These are the IUB ambiguity codes
8  # (Eur. J. Biochem. 150: 1-5, 1985):
9  # R = G or A
10 # Y = C or T
11 # M = A or C
12 # K = G or T
13 # S = G or C
14 # W = A or T
15 # B = not A (C or G or T)
16 # D = not C (A or G or T)
17 # H = not G (A or C or T)
18 # V = not T (A or C or G)
19 # N = A or C or G or T
```

```
21 sub IUB_to_regexp {
22
23     my ($iub) = @_;
24
25     my $regular_expression = '';
```

```
27      my %iub2character_class = (
28
29          A => 'A',
30          C => 'C',
31          G => 'G',
32          T => 'T',
33          R => '[GA]',
34          Y => '[CT]',
35          M => '[AC]',
36          K => '[GT]',
37          S => '[GC]',
38          W => '[AT]',
39          B => '[CGT]',
40          D => '[AGT]',
41          H => '[ACT]',
42          V => '[ACG]',
43          N => '[ACGT]',
44      );
```

```
46      # Remove the ^ signs from the recognition
    sites
47      $iub =~ s/\^//g;
48
49      # Translate each character in the iub
    sequence
50      for ( my $i = 0 ; $i < length($iub) ; ++
    $i ) {
51          $regular_expression .=
    $iub2character_class{ substr( $iub, $i, 1 )
     };
52      }
53
54      return $regular_expression;
55  }
```

目标

对于 REBASE 文件的每一行，返回三个数据：酶的名字、识别位点和正则表达式

策略一

- 使用数组：把三个数据项连续的存储在一起
- 读入数据：从数组中读取成组的三个项目
- 查询：有点困难……

策略二

- 使用散列：键为酶的名字，值为用空格分隔的识别位点和正则表达式
- 查询：快速
- 提取信息：使用 split 函数

目标

对于 REBASE 文件的每一行，返回三个数据：酶的名字、识别位点和正则表达式

策略一

- 使用数组：把三个数据项连续的存储在一起
- 读入数据：从数组中读取成组的三个项目
- 查询：有点困难……

策略二

- 使用散列：键为酶的名字，值为用空格分隔的识别位点和正则表达式
- 查询：快速
- 提取信息：使用 split 函数

目标

对于 REBASE 文件的每一行，返回三个数据：酶的名字、识别位点和正则表达式

策略一

- 使用数组：把三个数据项连续的存储在一起
- 读入数据：从数组中读取成组的三个项目
- 查询：有点困难……

策略二

- 使用散列：键为酶的名字，值为用空格分隔的识别位点和正则表达式
- 查询：快速
- 提取信息：使用 split 函数

```
1 # Example 9-2 Subroutine to parse a REBASE
    datafile
2 # parseREBASE-Parse REBASE bionet file
3 #
4 # A subroutine to return a hash where
5 #     key    = restriction enzyme name
6 #     value = whitespace-separated recognition
     site and regular expression
```

```
 8 sub parseREBASE {
 9
10     my ($rebasefile) = @_;
11
12     use strict;
13     use warnings;
14     use BeginPerlBioinfo;     # see Chapter 6
   about this module
15
16     # Declare variables
17     #my @rebasefile  = ();
18     my %rebase_hash = ();
19     my $name;
20     my $site;
21     my $regexp;
```

```
23      # Read in the REBASE file
24      my $rebase_filehandle = open_file($rebasefile)
    ;
25
26      while (<$rebase_filehandle>) {
27
28          # Discard header lines
29          ( 1 .. /Rich Roberts/ ) and next;
30
31          # Discard blank lines
32          /^\s*$/ and next;
33
34          # Split the two (or three if includes
    parenthesized name) fields
35          my @fields = split( " ", $_ );
36
37          # Get and store the name and the
    recognition site
```

```
39          # Remove parenthesized names, for simplicity's
    sake,
40          # by not saving the middle field, if any,
41          # just the first and last
42          $name = shift @fields;
43
44          $site = pop @fields;
45
46          # Translate the recognition sites to regular
    expressions
47          $regexp = IUB_to_regexp($site);
48
49          # Store the data into the hash
50          $rebase_hash{$name} = "$site $regexp";
51      }
52
53      # Return the hash containing the reformatted REBASE
    data
54      return %rebase_hash;
55  }
```

```
57 ##############################################################
58 # Subroutines
59 ##############################################################
60
61 # open_file
62 #
63 #    - given filename, set filehandle
64
65 sub open_file {
66
67     my ($filename) = @_;
68     my $fh;
69
70     unless ( open( $fh, $filename ) ) {
71         print "Cannot open file $filename\n";
72         exit;
73     }
74     return $fh;
75 }
```

教学提纲

```
1 ( 1 .. /Rich Roberts/ ) and next;
```

说明

- ..：范围操作符（range operator）
- 跳过头信息（从第一行到包含“Rich Roberts”的行）
- and：逻辑操作符

```
1 ( 1 .. /Rich Roberts/ ) and next;
```

说明

- ..：范围操作符（range operator）
- 跳过头信息（从第一行到包含“Rich Roberts”的行）
- and：逻辑操作符

```
1  foreach my $number (100..200) {
2      print "$number is a very big number\n";
3  }
4
5  foreach my $letter ("a".."z", "A".."Z") {
6      print "I know the letter $letter\n";
7  }
8
9  # a..z            = 26 combinations
10 # aa..zz          = 676 combinations
11 # aaa..zzz        = 17576 combinations
12 # aaaa..zzzz      = 456976 combinations
```

逻辑操作符（logical operator）

- and：测试两个条件是不是都为真
- or：测试是不是至少有一个条件为真
- not：否定操作符，测试某个条件是不是为假

补充说明

- 与 and、or 和 not 相关的操作符分别是 &&、|| 和 !；它们的优先级不同
- 对优先级不确定时，用小括号把表达式包裹起来，确保语句的执行结果和预期一样

逻辑操作符（logical operator）

- and：测试两个条件是不是都为真
- or：测试是不是至少有一个条件为真
- not：否定操作符，测试某个条件是不是为假

补充说明

- 与 and、or 和 not 相关的操作符分别是 &&、|| 和 !；它们的优先级不同
- 对优先级不确定时，用小括号把表达式包裹起来，确保语句的执行结果和预期一样

```
1  # and: 只有当两个条件都为真时，整个语句才为真
2  if( $string eq 'kinase' and $num == 3 ) {
3     ...
4  }
5
6  # or: 如果两个条件中至少有一个为真，整个语句就为真
7  if( $string eq 'kinase' or $num == 3 ) {
8     ...
9  }
10
11 # not: 条件为假，被not否定后，整个语句为真
12 if( not 6 == 9 ) {
13    ...
14 }
```

and 的求值顺序：类似 if 语句

- 首先对左边的参数进行求值
  - 如果左边为真，对右边的参数求值并返回结果
  - 如果左边为假，右边的参数永远不会被求值

```
1 if( $verbose  ) {
2   print $helpful_but_verbose_message;
3 }
4
5 $verbose and print
    $helpful_but_verbose_message;
```

and 的求值顺序：类似 if 语句

- 首先对左边的参数进行求值
  - 如果左边为真，对右边的参数求值并返回结果
  - 如果左边为假，右边的参数永远不会被求值

```
1 if( $verbose  ) {
2   print $helpful_but_verbose_message;
3 }
4
5 $verbose and print
    $helpful_but_verbose_message;
```

or 的求值顺序

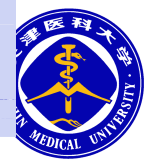
- 首先对左边的参数进行求值
  - 如果左边为真，就直接返回结果
  - 如果左边为假，对右边的参数求值并返回结果

```
1 unless(open(MYFILE, $file)) {
2   print "I cannot open file $file\n";
3   exit;
4 }
5
6 open(MYFILE, $file) or die "I cannot open
    file $file: $!";
```

or 的求值顺序

- 首先对左边的参数进行求值
  - 如果左边为真，就直接返回结果
  - 如果左边为假，对右边的参数求值并返回结果

```
1 unless(open(MYFILE, $file)) {
2   print "I cannot open file $file\n";
3   exit;
4 }
5
6 open(MYFILE, $file) or die "I cannot open
    file $file: $!";
```



```
1 # 跳过头部的信息行
2 ( 1 .. /Rich Roberts/ ) and next;
3
4 # 跳过空白行
5 /^\s*$/ and next;
```

next

The **next** function immediately sends a loop back to check its condition statement and start over. No code from the rest of the block is run. You can use this to skip over certain values when processing a list.

```
1 my @days = ("Mon","Tue","Wed","Thur","Fri","
    Sat","Sun");
2 foreach my $day (@days) {
3     next if ($day eq "Sun" or $day eq "Sat");
4     print "Ho hum it's $day - time to go to
    work\n";
5 }
```

next

The **next** function immediately sends a loop back to check its condition statement and start over. No code from the rest of the block is run. You can use this to skip over certain values when processing a list.

```
1 my @days = ("Mon","Tue","Wed","Thur","Fri","
    Sat","Sun");
2 foreach my $day (@days) {
3     next if ($day eq "Sun" or $day eq "Sat");
4     print "Ho hum it's $day - time to go to
    work\n";
5 }
```

last

To break out of a loop completely you can use the **last** function. This exits the loop immediately and moves on to the next statement below the loop. Using **last** you can break out of what would otherwise appear to be an infinite loop.

```
1  while (1) { # Infinite loop
2      my $guess = int(rand(10));
3      if ($guess==5) {
4          print "You guessed!\n";
5          last;
6      }
7      else {
8          print "No, not $guess, try again!\n";
9      }
10 }
```

last

To break out of a loop completely you can use the **last** function. This exits the loop immediately and moves on to the next statement below the loop. Using **last** you can break out of what would otherwise appear to be an infinite loop.

```
1  while (1) { # Infinite loop
2      my $guess = int(rand(10));
3      if ($guess==5) {
4          print "You guessed!\n";
5          last;
6      }
7      else {
8          print "No, not $guess, try again!\n";
9      }
10 }
```

tag

Next and last affect whatever is the closest enclosing loop block, but sometimes you want to have a loop within a loop, and be able to use next or last on the outer loop. To do this you can optionally tag a loop with a name which you can use with next and last to move around in nested loops.

```
YEARS: foreach my $year (2000..2005) {
  if (($year % 4) != 0) {
    foreach my $month (1..12) {
      if ($month == 7) { # I only work until July
        print "Yippee, 6 months off!\n";
        next YEARS;
      }
      print "I do some work in month $month of $year\n";
    }
  }
  print "Ooh $year is a leapyear!\n";
}
```

tag

Next and last affect whatever is the closest enclosing loop block, but sometimes you want to have a loop within a loop, and be able to use next or last on the outer loop. To do this you can optionally tag a loop with a name which you can use with next and last to move around in nested loops.

```
1  YEARS: foreach my $year (2000..2005) {
2    if (($year % 4) != 0) {
3      foreach my $month (1..12) {
4        if ($month == 7) { # I only work until July
5          print "Yippee, 6 months off!\n";
6          next YEARS;
7        }
8        print "I do some work in month $month of $year\n";
9      }
10   }
11   print "Ooh $year is a leapyear!\n";
12 }
```

教学提纲

```
1  # Get DNA
2  get_file_data
3
4  extract_sequence_from_fasta_data
5
6  # Get the REBASE data into a hash, from file
     "bionet"
7  parseREBASE('bionet');
8
9  for each user query
10
11   If query is defined in the hash
12     Get positions of query in DNA
13
14   Report on positions, if any
15 }
```

子程序

- 已有：extract_sequence_from_fasta_data
- 尚缺：在 DNA 中查询并返回匹配的位置

```
Given arguments $query and $dna

while ( $dna =~ /$query/ig ) {
  save the position of the match
}

return @positions
```

子程序

- 已有：extract_sequence_from_fasta_data
- 尚缺：在 DNA 中查询并返回匹配的位置

```
1 Given arguments $query and $dna
2
3 while ( $dna =~ /$query/ig ) {
4   save the position of the match
5 }
6
7 return @positions
```

```perl
# Find locations of a match of a regular expression in a string
# return an array of positions where the regular expression appears in the string
sub match_positions {
    my ( $regexp, $sequence ) = @_;
    use strict;
    use BeginPerlBioinfo;
    # Declare variables
    my @positions = ();
    # Determine positions of regular expression matches
    while ( $sequence =~ /$regexp/ig ) {
        push( @positions, pos($sequence) - length( $&) + 1 );
    }
    return @positions;
}
```

特殊变量

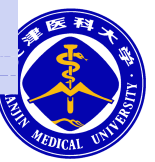
- `$&`：字符串中实际匹配模式的部分
- `$`` `：字符串中匹配模式前面的所有内容
- `$'`：字符串中匹配模式后面的所有内容

```
1 my $string = "abcdef";
2 while ( $string =~ /cd/g ) {
3     print "Prematch: $`\n";
4     print "Match: $&\n";
5     print "Posmatch: $'\n";
6 }
7 # $`:ab; $&:cd; $':ef
```

特殊变量

- `$&`: 字符串中实际匹配模式的部分
- `$`` `: 字符串中匹配模式前面的所有内容
- `$'`: 字符串中匹配模式后面的所有内容

```
1 my $string = "abcdef";
2 while ( $string =~ /cd/g ) {
3     print "Prematch: $`\n";
4     print "Match: $&\n";
5     print "Posmatch: $'\n";
6 }
7 # $`:ab; $&:cd; $':ef
```



pos

- pos：返回匹配序列后面第一个字符的索引位置
- pos - length：匹配序列第一个字符的索引位置

```
my $string = "abcdef";
# 0-index: 0 1 2 3 4 5
# string : a b c d e f
# 1-index: 1 2 3 4 5 6
while ( $string =~ /cd/g ) {
    my $pos    = pos($string);
    my $start0 = $pos - length($&);
    my $start1 = $pos - length($&) + 1;
    print "Number for pos(function): $pos\n";
    print "Number for start(0-index): $start0\n";
    print "Number for start(1-index): $start1\n";
}
# 4 2 3
```

pos

- pos：返回匹配序列后面第一个字符的索引位置
- pos - length：匹配序列第一个字符的索引位置

```
1  my $string = "abcdef";
2  # 0-index: 0 1 2 3 4 5
3  # string : a b c d e f
4  # 1-index: 1 2 3 4 5 6
5  while ( $string =~ /cd/g ) {
6      my $pos    = pos($string);
7      my $start0 = $pos - length($&);
8      my $start1 = $pos - length($&) + 1;
9      print "Number for pos(function): $pos\n";
10     print "Number for start(0-index): $start0\n";
11     print "Number for start(1-index): $start1\n";
12 }
13 # 4 2 3
```

```
1  #!/usr/bin/perl -w
2  # Example 9-3   Make restriction map from user
     queries on names of restriction enzymes
3
4  use strict;
5  use warnings;
6  use BeginPerlBioinfo;     # see Chapter 6 about
     this module
7
8  # Declare and initialize variables
9  my %rebase_hash       = ();
10 my @file_data         = ();
11 my $query             = '';
12 my $dna               = '';
13 my $recognition_site = '';
14 my $regexp            = '';
15 my @locations         = ();
```

```
17 # Read in the file "sample.dna"
18 @file_data = get_file_data("sample.dna");
19
20 # Extract the DNA sequence data from the
     contents of the file "sample.dna"
21 $dna = extract_sequence_from_fasta_data(
     @file_data);
22
23 # Get the REBASE data into a hash, from file
     "bionet"
24 %rebase_hash = parseREBASE('bionet');
25
26 # Prompt user for restriction enzyme names,
     create restriction map
```

```
27 do {
28     print "Search for what restriction site
     for (or quit)?: ";
29
30     $query = <STDIN>;
31
32     chomp $query;
33
34     # Exit if empty query
35     if ( $query =~ /^\s*$/ ) {
36
37         exit;
38     }
```

```
40      # Perform the search in the DNA sequence
41      if ( exists $rebase_hash{$query} ) {
42
43          ( $recognition_site, $regexp ) =
    split( " ", $rebase_hash{$query} );
44
45          # Create the restriction map
46          @locations = match_positions( $regexp
    , $dna );
```

```
48            # Report the restriction map to the user
49            if (@locations) {
50                print "Searching for $query
    $recognition_site $regexp\n";
51                print "A restriction site for $query
    at locations:\n";
52                print join( " ", @locations ), "\n";
53            }
54            else {
55                print "A restriction enzyme $query is
    not in the DNA:\n";
56            }
57        }
58        print "\n";
59    } until ( $query =~ /quit/ );
60
61    exit;
```

```
Search for what restriction enzyme (or quit)?: AceI
Searching for AceI G^CWGC GC[AT]GC
A restriction site for AceI at locations:
54 94 582 660 696 702 840 855 957

Search for what restriction enzyme (or quit)?: AccII
Searching for AccII CG^CG CGCG
A restriction site for AccII at locations:
181

Search for what restriction enzyme (or quit)?: AaeI
A restriction site for AaeI is not in the DNA:

Search for what restriction enzyme (or quit)?: quit
```

教学提纲

优先级（precedence）

在数学和计算机科学中，运算次序（也称为运算顺序、运算子优先级）是指决定在表示式中的哪一运算子首先被执行的规则。

比如，在四则运算中，一般有先乘除后加减的规定。就是说在 $2+3\times4$ 这样的式子中，按规定会先对 3 和 4 作乘法，得出 12，然后再把 2 和 12 加起来，最后就得出 14。

酶切图谱 | 优先级 | 规定

Associativity	Operators
left	Terms and list operators (leftward)
left	->
nonassoc	++ --
right	**
right	! ~ n + - (unary)
left	=~ !~
left	* / % x
left	+ - .
left	<< >>
nonassoc	named unary operators
nonassoc	< > <= >= lt gt le ge
nonassoc	== != <=> eq ne cmp

酶切图谱 | 优先级 | 规定

left	&
left	|^
left	&&
left	||
nonassoc	.. ...
right	?:
right	= += -= *= etc. (assignment operators)
left	, =>
nonassoc	List operators (rightward)
right	not
left	and
left	or xor

实例

- $10 + 8/4 - 2 \times 3 = 10$
- $10 + (8/4) - (2 \times 3) = 6$
- $10 + (8/4 - 2) \times 3 = 10$

基本原则

- （在复杂的表达式中）使用括号明确优先级
- 好处：不用背/查优先级表，避免大量的程序调试，……

实例

- $10 + 8/4 - 2 \times 3 = 10$
- $10 + (8/4) - (2 \times 3) = 6$
- $10 + (8/4 - 2) \times 3 = 10$

基本原则

- （在复杂的表达式中）使用括号明确优先级
- 好处：不用背/查优先级表，避免大量的程序调试，……

教学提纲

教学提纲

知识点

- 正则表达式基础：基本概念、基础理论、基本语法
- 正则表达式应用：元字符，解析，构建
- 操作符：范围操作符，逻辑操作符，优先级
- 逻辑操作符：求值顺序，应用
- 模式匹配：特殊变量，pos 函数

技能

- 能够把 IUB 代码翻译成正则表达式
- 能够编写制作酶切图谱相关的 Perl 程序

知识点

- 正则表达式基础：基本概念、基础理论、基本语法
- 正则表达式应用：元字符，解析，构建
- 操作符：范围操作符，逻辑操作符，优先级
- 逻辑操作符：求值顺序，应用
- 模式匹配：特殊变量，pos 函数

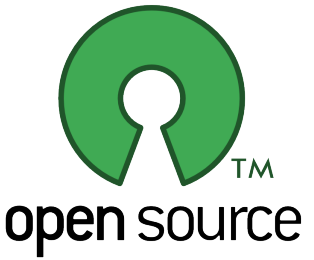
技能

- 能够把 IUB 代码翻译成正则表达式
- 能够编写制作酶切图谱相关的 Perl 程序

教学提纲

1. 总结正则表达式的基本运算。
2. 总结正则表达式的基本语法。
3. 举例说明正则表达式中的元字符。
4. 解析正则表达式实例。
5. 根据要求编写正则表达式。
6. 举例说明范围操作符的使用。
7. 列举常见的逻辑操作符，并解释其求值顺序。
8. 列举模式匹配中的特殊变量，举例进行说明。
9. 如何明确复杂表达式中操作的优先级?

Powered by



TEX LATEX XETEX Beamer