离散数学

目录

第 1 章 逻辑

1.1 命题

<u>命题(Proposition)</u>

逻辑(logic)规则给出数学语句的准确含义，这些规则用来区分有效和无效的数学**论证**。逻辑不仅对理解数学推理十分重要，而且在计算机科学中有许多应用，逻辑可用于**电路设计**、**程序构造**、**程序正确性证明**等方面。

命题(proposition)是逻辑的基本成分，一个**命题**是一个具有**真值**(truth value)的语句，命题可以为**真**也可以为**假**，但不能既为真又为假。

命题	非命题
I have a dog.	What day is today?
1 + 2 = 3	Shut the door!
Today is Wednesday.	1 + 2
It is snowing today.	x + 1 = 2

命题习惯上用字母 p、q、r、s 等来表示，如果一个命题是**真命题**，它的真值为真，用“**T**”表示；如果一个命题是**假命题**，它的真值为假，用“**F**”表示。

<u>非运算符(Negation Operator / NOT)</u>

非运算符“¬”只作用于一个命题，其作用是**反转命题的真值**。

真值表(truth table)可以给出命题真值之间的关系，在确定由简单命题组成的命题的真值时，真值表特别有用。

p	**¬p**
T	F
F	T

范例：¬p
p: It snowed last night. q: 2 + 3 = 6 ¬p: It didn't snow last night. ¬q: $2+3\neq 6$

合取运算符(Conjunction Operator / AND)

命题 p∧q 表示“**p 并且 q**”，当 p 和 q 都为真时命题为真，否则为假。真值表如下：

p	**q**	**p∧q**
T	T	T
T	F	F
F	T	F
F	F	F

范例：p∧q
p: 今天是星期五。 q: 今天下雨。

$p \wedge q$: 今天是星期五并且下雨。

析取运算符(Disjunction Operator / OR)

命题 $p \vee q$ 表示“**p 或 q**”，当 p 和 q 都为假时命题为假，否则为真。真值表如下：

p	**q**	$\mathbf{p \vee q}$
T	T	T
T	F	T
F	T	T
F	F	F

范例：$\mathbf{p \vee q}$
p: 开关坏了。
q: 灯泡坏了。
$p \vee q$: 开关坏了或者灯泡坏了。

异或运算符(Exclusive Or / XOR)

命题 $p \oplus q$ 表示 p 和 q 的**异或**，当 p 和 q 种恰有一个为真时命题为真，否则为假。真值表如下：

p	**q**	$\mathbf{p \oplus q}$
T	T	F
T	F	T

F	T	T
F	F	F

范例：p⊕q
p: 他现在在上海。 q: 他现在在北京。 p⊕q: 他现在在上海或北京。

范例：某地发生了一件谋杀案，警察通过排查确定杀人凶手必为 4 个嫌疑犯的一个。以下为 4 个嫌疑犯的供词。
A 说：不是我。
B 说：是 C。
C 说：是 D。
D 说：C 在胡说
已知 3 个人说了真话，1 个人说的是假话。
现在请根据这些信息，确定到底谁是凶手。

```python
def main():
    for killer in ['A', 'B', 'C', 'D']:
        if (killer != 'A') + (killer == 'C') \
            + (killer == 'D') + (killer != 'D') == 3:
            print(killer)

if __name__ == "__main__":
    main()
```

运行结果	C

1.2 复合命题

复合命题(Compound Proposition)

使用非运算符和已定义的各联结词可以构造**复合命题**。小括号用于规定复合命题中多个逻辑运算符的操作顺序，为了减少所需的小括号数量，规定了各联结词的**优先级**。

优先级	
1	¬
2	∧ / ∨
3	→ / ↔

蕴含运算符(Implication Operator)

命题 p → q 表示 **p 蕴含 q**，在 p 为真而 q 为假时命题为假，否则为真。其中 **p 称为前提**，**q 称为结论**。真值表如下：

p	q	p → q
T	T	T
T	F	F
F	T	T
F	F	T

表示 p → q 的术语有很多种，常见的有：

- if p, then q.
- p only if q.

- q is necessary for p.

范例：p → q
p: 我去看电影。
q: 我买奶茶。
p → q: 如果我去看电影，那么我会买奶茶。

由 p → q 可以构造出几个相关的蕴含：

1. q → p 称为 p → q 的**逆命题**(converse)。
2. ¬q → ¬p 称为 p → q 的**倒置命题**(contrapositive)。

范例：逆命题与倒置命题
p: 今天是星期四。
q: 我今天有考试。
p → q: 如果今天是星期四，那么我今天有考试。
q → p：如果我今天有考试，那么今天是星期四。
¬q → ¬p：如果我今天没有考试，那么今天不是星期四。

双蕴含(Biconditional Operation)

命题 $p \leftrightarrow q$ 表示 **p 双向蕴含 q**，在 p 和 q 有相同的真值时命题为真，否则为假。真值表如下：

p	**q**	**$p \leftrightarrow q$**
T	T	T
T	F	F
F	T	F
F	F	T

双蕴含的真值表与异或的真值表正好相反，因此 $p \leftrightarrow q$ 与$\neg(p \oplus q)$相同。

1.3 逻辑等价

逻辑等价(Logical Equivalence)

两个不同的复合命题可能拥有完全相同的真值，则称这两个命题在逻辑上是等价的。如果无论复合命题中各个命题的真值是什么，复合命题的真值总是为真，这样的复合命题称为永真式(tautology)。如果真值永远为假的复合命题称为矛盾(contradiction)。

p	$\neg p$	$p \lor \neg p$	$p \land \neg p$
T	F	T	F
F	T	T	F

如果复合命题 s 和 r 是逻辑等价的，可表示为 $s \equiv r$。只有当 $s \leftrightarrow r$ 是永真式时，s 和 r 才是逻辑等价的。

范例：使用真值表证明$p \lor q \equiv \neg(\neg p \land \neg q)$

p	q	$p \lor q$	$\neg p$	$\neg q$	$\neg p \land \neg q$	$\neg(\neg p \land \neg q)$
T	T	T	F	F	F	T
T	F	T	F	T	F	T
F	T	T	T	F	F	T
F	F	F	T	T	T	F

一些重要的逻辑等价关系如下：

名称	等价关系
幂等律 (Idempotent Laws)	$p \land p \equiv p$ $p \lor p \equiv p$

恒等律 (Identity Laws)	$\mathbf{p \wedge T \equiv p}$ $\mathbf{p \vee F \equiv p}$
支配律 (Domination Laws)	$\mathbf{p \vee T \equiv T}$ $\mathbf{p \wedge F \equiv F}$
双非律 (Double Negation Law)	$\mathbf{\neg(\neg p) \equiv p}$
交换律 (Commutative Laws)	$\mathbf{p \wedge q \equiv q \wedge p}$ $\mathbf{p \vee q \equiv q \vee p}$
结合律 (Associative Laws)	$\mathbf{(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)}$ $\mathbf{(p \vee q) \vee r \equiv p \vee (q \vee r)}$
分配律 (Distributive Laws)	$\mathbf{p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)}$ $\mathbf{p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)}$
德摩根律 (De Morgan's Laws)	$\mathbf{\neg(p \wedge q) \equiv \neg p \vee \neg q}$ $\mathbf{\neg(p \vee q) \equiv \neg p \wedge \neg q}$
吸收律 (Absorption Laws)	$\mathbf{p \wedge (p \vee q) \equiv p}$ $\mathbf{p \vee (p \wedge q) \equiv p}$
条件恒等	$\mathbf{p \rightarrow q \equiv \neg p \vee q}$ $\mathbf{p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)}$

范例：证明$(\boldsymbol{p \wedge q}) \rightarrow \boldsymbol{p}$永真
$(p \wedge q) \rightarrow p$ $\equiv \neg(\mathrm{p} \wedge q) \vee \mathrm{p}$ $\equiv (\neg \mathrm{p} \vee \neg q) \vee \mathrm{p}$ $\equiv (\neg \mathrm{q} \vee \neg p) \vee \mathrm{p}$

$\equiv \neg q \vee (\neg p \vee p)$
$\equiv \neg q \vee T$
$\equiv T$

1.4 谓词和量词

谓词(Predicate)

命题逻辑并不能表达数学语言和自然语言中所有语句的确切含义。在数学表达式和计算机程序中经常可以看到含有变量的语句，例如“x > 3”、“x = y + 3”、“程序 x 正在运行”等。当变量值未指定时，这些语句既不为真也不为假。

利用 P(x)可以表示语句，其中 x 是变量，语句 P(x)可以说是命题函数 P 在 x 的值。一旦给变量 x 赋一个值，语句 P(x)就称为命题并具有真值。

通常使用大写字母 P、Q、R 等表示**谓词**，小写字母 x、y、z 等表示**变量**。

范例：谓词	
P(x): x + 3 = 6，P(3)的真值？	True
Q(x, y): x = y + 2，Q(4, 1)的真值？	False

量词(Quantifier)

量词用量化表示在何种程度上谓词对于一定范围的个体成立。

量词有**全称量词**(universal quantifier)和**存在量词**(existential quantifer)：

1. **全称量词**用“∀”表示“ALL”。$\forall_x P(x)$是一个命题，当范围内**所有**的 x 都能使语句 P(x)为真时，命题为真。

$$\forall_x P(x) = P(a_1) \wedge P(a_2) \wedge \ldots \wedge P(a_k)$$

范例：全称量词
假设 x 表示“全班所有学生”，P(x)表示“x 完成了作业”。 $\forall_x P(x)$：全班所有学生都完成了作业。

2. 存在量词用“∃”表示“Exist”。$\exists_x P(x)$是一个命题，当范围内存在至少一个 x 能够语句 P(x)为真时，命题为真。

$$\exists_x P(x) = P(a_1) \vee P(a_2) \vee \ldots \vee P(a_k)$$

范例：存在量词
假设 x 表示“全班所有学生”，P(x)表示“x 完成了作业”。 $\exists_x P(x)$：班里存在有一个学生完成了作业。

范例：嵌套量词
假设 x 表示“一个人”，P(x)表示“x 有父母”。 $\forall_x P(x)$：所有人都有父母。 $\exists_x P(x)$：存在至少有一个人有父母。 $\exists_x \exists_y (P(x) \wedge P(y))$：至少存在一个人 x 和一个人 y 有父母。

范例：P(x): x 是偶数，Q(x): x 能被 3 整除，x ∈ $\mathbb{Z}^+$。	
$\exists_x (P(x) \wedge Q(x))$的真值？	True
$\forall_x (P(x) \rightarrow \neg Q(x))$的真值？	False

<u>全称量词的否定</u>

否定运算符可以使用在全称量词上。两个等价关系如下：

1. $\neg \forall_x P(x) \equiv \exists_x \neg P(x)$
2. $\neg \exists_x P(x) \equiv \forall_x \neg P(x)$

范例：$\neg \forall_x \boldsymbol{P(x)}$与$\forall_x \neg \boldsymbol{P(x)}$

P(x): x will pass the course (x is a student).

$\neg\forall_x P(x)$: Not all students will pass the course.

$\forall_x \neg P(x)$: No student will pass the course.

范例：$\neg\exists_x \boldsymbol{P}(\boldsymbol{x})$与$\exists_x \neg\boldsymbol{P}(\boldsymbol{x})$

P(x): x will pass the course (x is a student).

$\neg\exists_x P(x)$: There does not exist a student that will pass the course.

$\exists_x \neg P(x)$: There exists a student that will not pass the course.

1.5 证明

证明(Proof)

证明方法非常重要，不仅因为它们可用于证明数学定理，而且在计算机科学中也有许多应用，包括验证程序正确性、建立安全的操作系统、人工智能领域做推论等。证明就是建立定理真实性的**有效论证**。

证明定理有很多方法：

1. 直接证明法(direct proof)

范例：直接证明法
定理：如果 n 是奇数，那么n^2也是奇数。 当 $n \in \mathbb{Z}$，$\forall_n (P(n) \rightarrow Q(n))$ $n^2 = (2k+1)^2$ $= 4k^2 + 4k + 1$ $= 2(2k^2 + 2k) + 1$

2. 反证法(proof by contrapositive)：由于 $p \rightarrow q \equiv \neg q \rightarrow \neg p$，因此可以通过证明原命题的**逆否命题**来反证原命题。

范例：反证法
定理：当 $x, y \in \mathbb{Z}$，如果 x*y 是偶数，那么 x 是偶数或 y 是偶数。 逆否命题：当 $x, y \in \mathbb{Z}$，如果 x 是奇数并且 y 是奇数，那么 x*y 是奇数。 $x * y = (2m+1) * (2n+1)$ $= 4mn + 2m + 2n + 1$ $= 2(2mn + m + n) + 1$

1.6 布尔代数

布尔代数(Boolean Algebra)

计算机和其它电子设备中的**电路**都有**输入**和**输出**，输入是 0 或 1，**输出也是 0 或 1**。电路可以用任何具有**两个不同状态**的基本元件来构造，例如开关和光学装置就是这样的原件，开关可位于开或关的位置，光学装置可能是点亮或未点亮。18 世纪，乔治 • 布尔（George Boole）给出了逻辑的基本规则。

电路的操作可以用**布尔函数**来定义，这样的**布尔函数对任意一组输入都能指出其输出的值**。

布尔代数提供的是**集合{0, 1}上的运算和规则**，布尔代数的规则类似于命题逻辑的规则。“**1**”相当于逻辑中的“**真**”，“**0**”相当于逻辑中的“**假**”。

布尔代码运算主要有三种：

1. **补**(complement)

x	$\overline{x}$
1	0
0	1

2. **布尔积**(boolean multiplication)

x	**y**	**x · y**
1	1	1
1	0	0
0	1	0
0	0	0

3. 布尔和(boolean addition)

x	y	x + y
1	1	1
1	0	1
0	1	1
0	0	0

范例：布尔表达式
当 x = y = 1，w = z = 0 时， $x \cdot y + (w + z) = 1 \cdot 1 + (0 + 0) = 1 + 0 = 1$ $x \cdot \overline{y} + z \cdot \overline{(w + z)} = 1 \cdot \overline{1} + 0 \cdot \overline{(0 + 0)} = 0 + 0 = 0$ $x \cdot \overline{y} + \overline{(\overline{x} + y + \overline{y}z)} = 1 \cdot \overline{1} + \overline{(\overline{1} + 1 + \overline{1} \cdot 0)} = 0 + \overline{1} = 0$

一些重要的布尔代数关系如下：

名称	等价关系
幂等律 (Idempotent Laws)	$\mathbf{x \cdot x = x}$ $\mathbf{x + x = x}$
恒等律 (Identity Laws)	$\mathbf{x \cdot 1 = x}$ $\mathbf{x + 0 = x}$
支配律 (Domination Laws)	$\mathbf{x \cdot 0 = 0}$ $\mathbf{x + 1 = 1}$
双非律 (Double Negation Law)	$\overline{\overline{x}} = x$

交换律 **(Commutative Laws)**	$\mathbf{x \cdot y = y \cdot x}$ $\mathbf{x + y = y + x}$
结合律 **(Associative Laws)**	$\mathbf{(x \cdot y) \cdot y = x \cdot (y \cdot z)}$ $\mathbf{(x + y) + y = x + (y + z)}$
分配律 **(Distributive Laws)**	$\mathbf{x \cdot (y + z) = (x \cdot y) + (x \cdot z)}$ $\mathbf{x + (y \cdot z) = (x + y) \cdot (x + z)}$
德摩根律 **(De Morgan's Laws)**	$\mathbf{\overline{(x \cdot y)} = \bar{x} + \bar{y}}$ $\mathbf{\overline{(x + y)} = \bar{x} \cdot \bar{y}}$
吸收律 **(Absorption Laws)**	$\mathbf{x \cdot (x + y) = x}$ $\mathbf{x + (x \cdot y) = x}$

范例：证明$\mathbf{xy + x\bar{y} = x}$
$xy + x\bar{y}$ $= x \cdot (y + \bar{y})$ $= x \cdot 1$ $= x$

布尔函数(Boolean Function)

含有 n 个变量的布尔函数能够构造出 2^n 行的输入输出表。

范例：计算布尔函数$\mathbf{F(x, y, z) = xy + \bar{z}}$的值

x	**y**	**z**	**F(x, y, z)**
0	0	0	1
0	0	1	0
0	1	0	1

0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

NAND 和 AND 运算符

NAND 运算符用“↑”表示“**Not And**”，$x \uparrow y = \overline{x \cdot y}$。

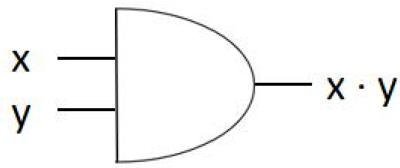
NOR 运算符用“↓”表示“**Not Or**”，$x \downarrow y = \overline{x + y}$。

1.7 逻辑门电路

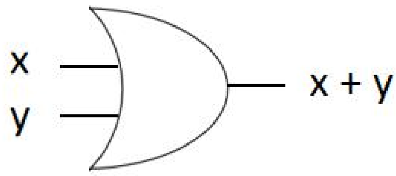
逻辑门电路(Logical Gate Circuit)

基础的**逻辑门**主要有三种：

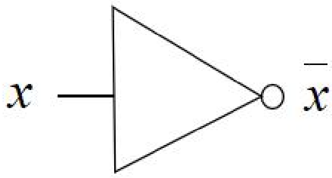
1. **与门**（**AND** gate）：



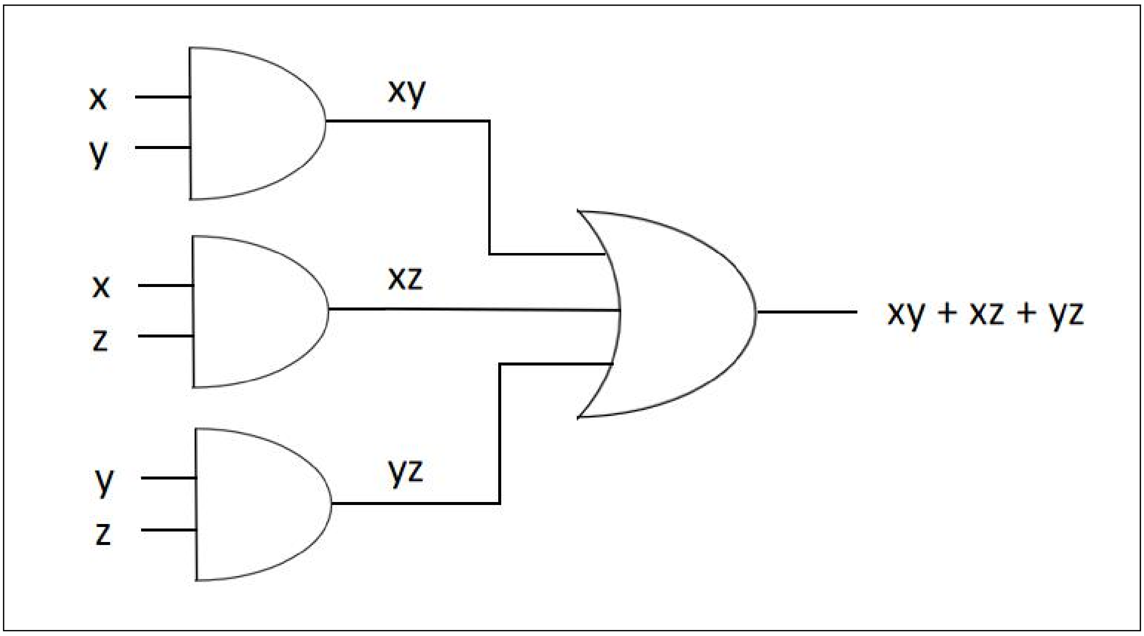
2. **或门**（**OR** gate）：



3. **非门**（**NOT** gate）：



范例：设计一个投票表决电路，三个人中有两人赞成即通过。赞成票为 1，否决表为 0。布尔函数为 F(x, y, z) = xy +xz +yz。


x
y
xy
x
z
xz
xy + xz + yz
y
z
yz

第 2 章 集合

2.1 集合

集合(Set)

集合是对象的**唯一**的、**无序**的聚集，通常一个集合中的对象都具有**相似**的性质。**对象**也称为集合的**元素**（element）或**成员**（member）。

通常用大写字母表示集合，小写字母表示元素。$a \in A$ 表示 a 是集合 A 中的元素，$a \notin A$ 表示 a 不是集合 A 中的元素。

使用**花名册方法**（roster method）列出集合中的元素，可以用于描述集合。

范例：花名册方法
小写元音字母集合 $V = \{a, e, i, o, u\}$
小于 10 的正奇数集合 $O = \{1, 3, 5, 7, 9\}$
小于 100 的非负整数集合 $A = \{0, 1, 2, 3, ..., 99\}$

集合构造器（set builder）通过描述元素具有的形式来描述集合。

范例：集合构造器
小于 10 的正整数 $A = \{x \mid x < 10\}$

有一些常用的特殊符号可用于描述指定的集合：

符号	含义
$\mathbb{N}$	自然数集 $\{0, 1, 2, 3, ...\}$
$\mathbb{Z}$	整数集 $\{..., -2, -1, 0, 1, 2, ...\}$
$\mathbb{Z}^+$	正整数集 $\{1, 2, 3, ...\}$

$\mathbb{Q}$	有理数集$\{p/q \mid p \in \mathbb{Z}, q \in \mathbb{Z}\ (q \neq 0)\}$
$\mathbb{Q}^+$	正有理数集
$\mathbb{R}$	实数集
$\mathbb{R}^+$	正实数集
$\mathbb{C}$	复数集
$\emptyset$	空集{}

基数(Cardinality)

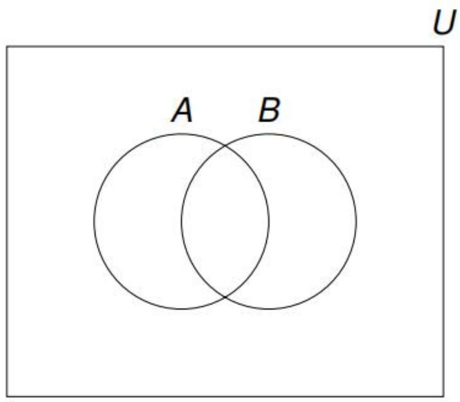
基数表示**有限集合**中元素的**个数**，集合 A 的基数记为“|A|”。

范例：基数
英语字母集合 A，$\lvert A\rvert = 26$ 空集$\emptyset$，$\lvert\emptyset\rvert = 0$

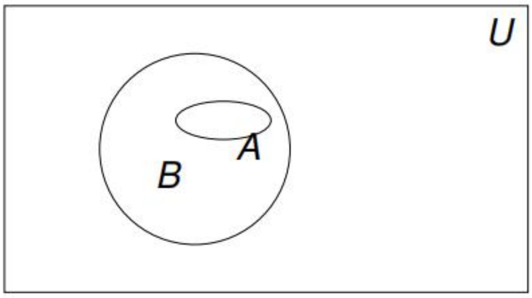
文氏图(Venn Diagram)

集合还可以使用**文氏图**来表示。

全集(universal set)包含所研究问题中**所有的元素**，用符号“U”表示。



假设有两个集合 A 和 B，如果 A 中的所有元素都 B 中，那么 A 就是 B 的子集，表示为“$A \subseteq B$”。如果 A 中有一个元素不在 B 中，那么 A 就不是 B 的子集，表示为“$A \not\subseteq B$”。



只有当两个集合互相为对方的子集时，那么这两个集合相等，即：

$$\mathbf{A = B \text{ iff } A \subseteq B \text{ and } B \subseteq A}$$

如果 $A \subseteq B$，并且 B 中有一个元素不是 A 的元素，那么称 A 是 B 的**真子集**(proper subset)，表示为“$A \subset B$”。

<u>幂集(Power Set)</u>

一个集合中是可以包含另一个集合的，如$\{\{1\},\{1,2\},\{1,2,3\}\}$。需要注意，$1 \neq \{1\} \neq \{\{1\}\}$。

幂集用于表示**一个集合所有子集的集合**，集合 A 的幂集表示为 P(A)。

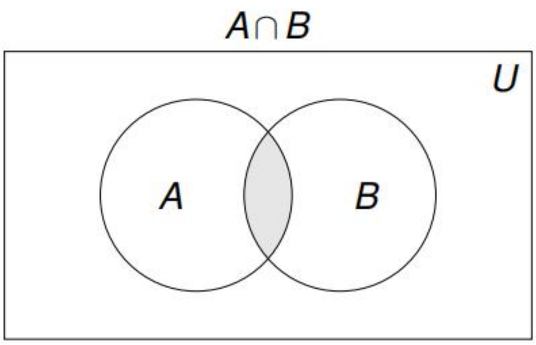
范例：幂集
$A = \{1, 2, 3\}$ $P(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

如果集合 A 的基数为 n，那么 A 的幂集的基数为2^n，即$|P(A)| = 2^n$。

2.2 集合运算

交集(Intersection)

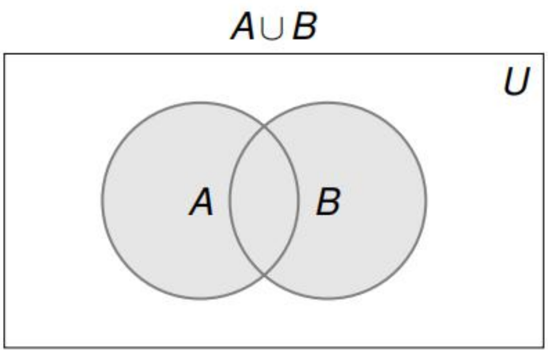
假设 A 和 B 是两个集合，由所有属于 A 并且属于 B 的元素所组成的集合，称为 A 与 B 的交集，表示为“A ∩ B”。



如果两个集合没有公共元素，那么它们的交集为空集。

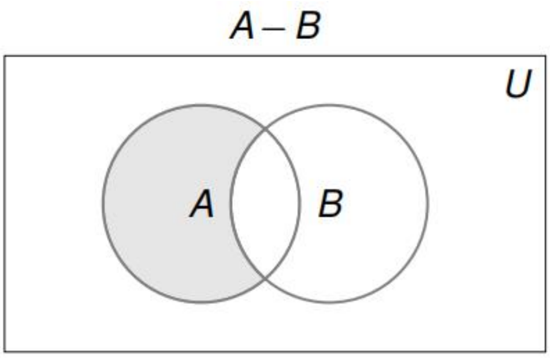
并集(Union)

假设 A 和 B 是两个集合，由它们所有元素合并在一起组成的集合，称为 A 与 B 的并集，表示为“A ∪ B”。



差集(Difference)

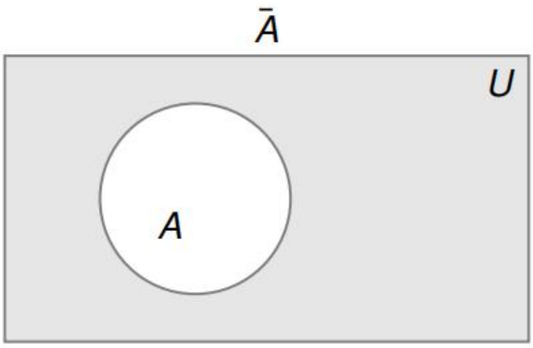
假设 A 和 B 是两个集合，由属于 A 而不属于 B 的元素组成的集合，称为 A 与 B 的差集，表示为“A - B”。



差集运算不满足交换律，即 $A - B \neq B - A$。

补集(Complement)

假设 A 是一个集合，由全集U中所有不属于 A 的元素组成的集合，称为 A 的补集，表示为“$\bar{A}$”。



2.3 集合恒等式

集合恒等式

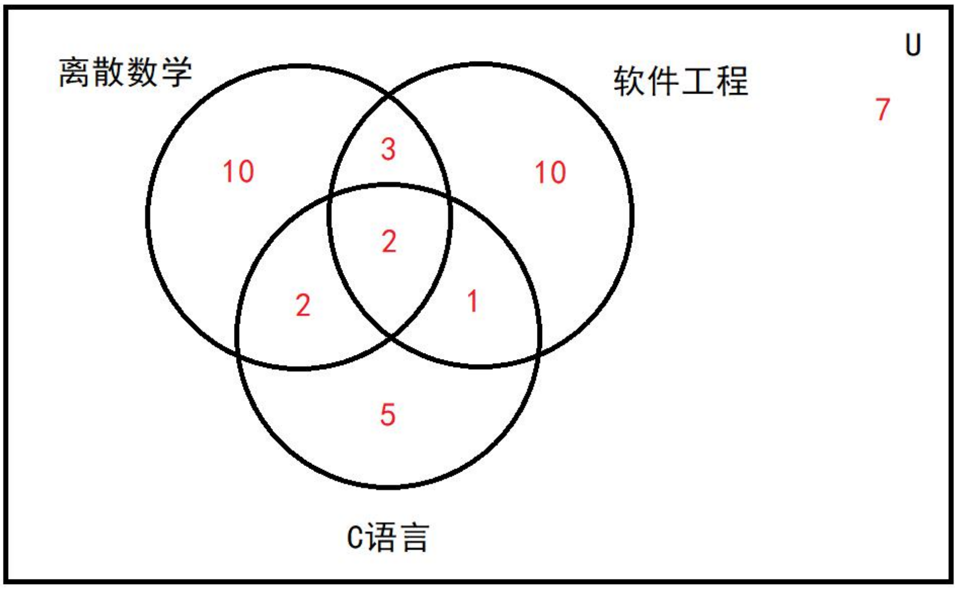
集合恒等式可以直接由对应的逻辑等价式证明。

名称	等价关系
幂等律 **(Idempotent Laws)**	$\mathbf{A} \cap \mathbf{A} = \mathbf{A}$ $\mathbf{A} \cup \mathbf{A} = \mathbf{A}$
恒等律 **(Identity Laws)**	$\mathbf{A} \cap \mathrm{U} = \mathbf{A}$ $\mathbf{A} \cup \emptyset = \mathbf{A}$
支配律 **(Domination Laws)**	$\mathbf{A} \cap \emptyset = \emptyset$ $\mathbf{A} \cup \mathrm{U} = \mathrm{U}$
补律 **(Complement Law)**	$\overline{\overline{\boldsymbol{A}}} = \boldsymbol{A}$
交换律 **(Commutative Laws)**	$\mathbf{A} \cap \mathbf{B} = \mathbf{B} \cap \mathbf{A}$ $\mathbf{A} \cup \mathbf{B} = \mathbf{B} \cup \mathbf{A}$
结合律 **(Associative Laws)**	$\mathbf{A} \cap (\mathbf{B} \cap \mathbf{C}) = (\mathbf{A} \cap \mathbf{B}) \cap \mathbf{C}$ $\mathbf{A} \cup (\mathbf{B} \cup \mathbf{C}) = (\mathbf{A} \cup \mathbf{B}) \cup \mathbf{C}$
分配律 **(Distributive Laws)**	$\mathbf{A} \cap (\mathbf{B} \cup \mathbf{C}) = (\mathbf{A} \cap \mathbf{B}) \cup (\mathbf{A} \cap \mathbf{C})$ $\mathbf{A} \cup (\mathbf{B} \cap \mathbf{C}) = (\mathbf{A} \cup \mathbf{B}) \cap (\mathbf{A} \cup \mathbf{C})$
德摩根律 **(De Morgan's Laws)**	$\overline{\boldsymbol{A} \cap \boldsymbol{B}} = \overline{\boldsymbol{A}} \cup \overline{\boldsymbol{B}}$ $\overline{\boldsymbol{A} \cup \boldsymbol{B}} = \overline{\boldsymbol{A}} \cap \overline{\boldsymbol{B}}$
吸收律	$\mathbf{A} \cap (\mathbf{A} \cup \mathbf{B}) = \boldsymbol{A}$

(Absorption Laws)	$\mathbf{A} \cup (\mathbf{A} \cap \mathbf{B}) = \boldsymbol{A}$

范例：证明$\overline{A \cup (B \cap C)} = (\overline{C} \cup \overline{B}) \cap \overline{A}$
$\overline{A \cup (B \cap C)}$ $= \overline{A} \cap \overline{(B \cap C)}$ $= \overline{A} \cap (\overline{B} \cup \overline{C})$ $= (\overline{B} \cup \overline{C}) \cap \overline{A}$ $= (\overline{C} \cup \overline{B}) \cap \overline{A}$

范例：一共有 40 个学生，有 3 门课程可供学生选择（C 语言、离散数学、软件工程）。
7 人没有选任何课程；
16 人选软件工程；
10 人选 C 语言；
5 人同时选离散数学和软件工程；
4 人同时选离散数学和 C 语言；
3 人同时选软件工程和 C 语言；
2 人同时选离散数学、软件工程和 C 语言。



2.4 笛卡尔积

元组(Tuple)

有时候元素聚集中**次序**是很重要的，由于集合是**无序**的，所以就需要一种不同的结构表示**有序**的聚集，这就是**有序 n 元组**(ordered-n-tuple)。

有序 n 元组$(a_1, a_2, \ldots, a_n)$是以 a_1 为第 1 个元素，a_2 为第 2 个元素，a_n 为第 n 个元素的有序聚集。

只有两个有序 n 元组的**每一对对应的元素都相等**，那么这两个有序 n 元组是相等的，即：

$$(a_1, a_2, \ldots, a_n) = (b_1, b_2, \ldots, b_n) \text{ iff } i = 1, 2, \ldots, n$$

需要注意，**(a, b)与(b, a)不相等**，**除非 a = b**。

笛卡尔积(Cartesian Product)

假设有两个集合 A 和 B，A 和 B 的**笛卡尔积**用“A × B”表示，笛卡尔积是**所有序偶(a, b)的集合**，其中 a ∈ A 且 b ∈ B。

$$A \times B = \{(a, b) | a \in A \wedge b \in B\}$$

范例：笛卡尔积的应用
学生集合 A = {s1, s2}
课程集合 B = {c1, c2, c3}
$A \times B = \{(s1, c1), (s1, c2), (s1, c3), (s2, c1), (s2, c2), (s2, c3)\}$
笛卡尔积 A × B 表示学生选课的所有可能情况。

笛卡尔积$A \times B$和$B \times A$是不相等的，除非$A = \emptyset$或$B = \emptyset$或$A = B$。

范例：笛卡尔积$\mathbf{A \times B \times C}$
A = {0, 1} B = {1, 2} C = {0, 1, 2} $\mathrm{A \times B \times C} = \{(0,1,0),(0,1,1),(0,1,2),(0,2,0),(0,2,1),(0,2,2)$ $(1,1,0),(1,1,1),(1,1,2),(1,2,0),(1,2,1),(1,2,2)\}$

一个集合与**自身**的笛卡尔积，如 A×A 可表示为“A^2”。

范例：笛卡尔积A^2
A = {1, 2} $A^2 = \{(1,1),(1,2),(2,1),(2,2)\}$

第 3 章 函数

3.1 函数的概念

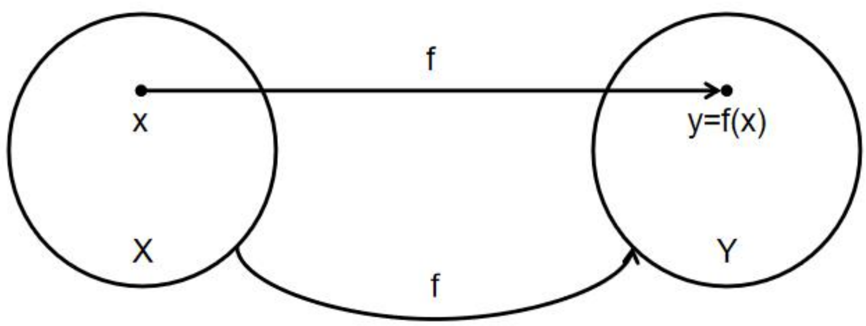
函数(Function)

函数在数学和计算机科学中的概念非常重要，在离散数学中函数用于定义像序列和字符串这样的离散结构。

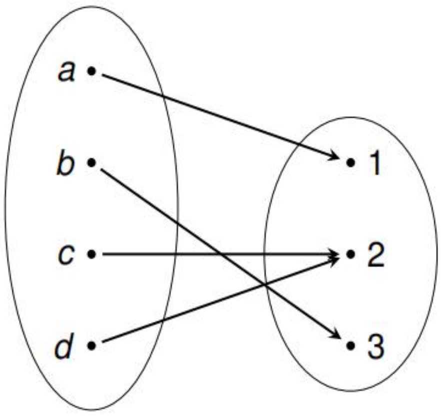
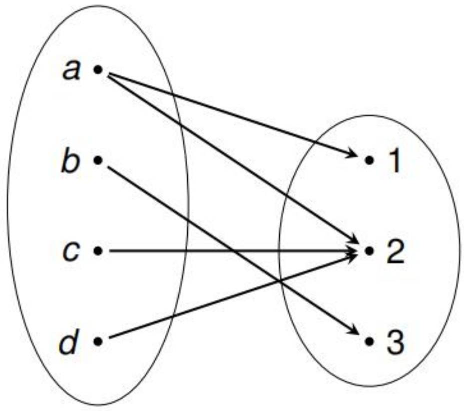
利用一个**函数** f，可以将一个值 $x \in \mathbb{R}$**映射**(mapping)到一个特定的值 $y = f(x)$ 上（$y \in \mathbb{R}$）。

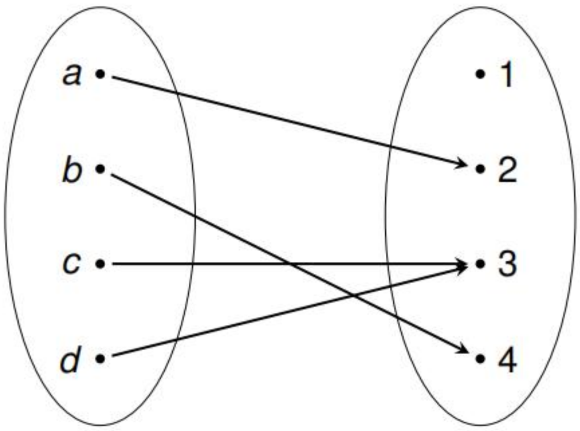
假设有两个非空集合 X 和 Y，从 X 到 Y 的函数 f 是指对于 X 的每个元素恰好都**对应** Y 的一个元素（$f(x) = y, x \in X, y \in Y$），那么就写成 $f: X \rightarrow Y$。

集合 X 被称为函数 f 的**定义域**(domain)，集合 Y 被称为函数 f 的**陪域**(co-domain)。如果 $f(x) = y$，那么 y 是 x 在函数 f 下的**像**(image)，x 是 y 载函数 f 下的**原像**(pre-image)。函数 f 的**值域**(range)是集合 X 中所有像的集合。



范例：判断是否为函数

<table>
<tr><td></td><td></td></tr>
<tr><td>函数</td><td>不是函数</td></tr>
</table>

<table>
<tr><th colspan="2">范例：函数</th></tr>
<tr><td colspan="2"></td></tr>
<tr><td>是否为函数</td><td>是</td></tr>
<tr><td>定义域 domain</td><td>a, b, c, d</td></tr>
<tr><td>陪域 co-domain</td><td>1, 2, 3, 4</td></tr>
<tr><td>值域 range</td><td>2, 3, 4</td></tr>
</table>

当两个函数 f 和 g 有相同的定义域和陪域，并且对于定义域中所有元素 x 都满足 $f(x) = g(x)$，那么函数 f 和 g 相等，表示为“$f = g$”。

3.2 上取整/下取整函数

上取整函数(Ceiling Function)

上取整函数和下取整函数可以将实数**映射到整数**（$\mathbb{R} \to \mathbb{Z}$），它们以不同的方式将实数近似到相邻的整数。

上取整函数将实数 x 向上取到大于或等于 x 的最小整数，表示为"$\lceil x \rceil$"。

范例：上取整函数
$\lceil 3.2 \rceil = 4$ $\lceil 2.6 \rceil = 3$ $\lceil -0.5 \rceil = 0$

下取整函数(Floor Function)

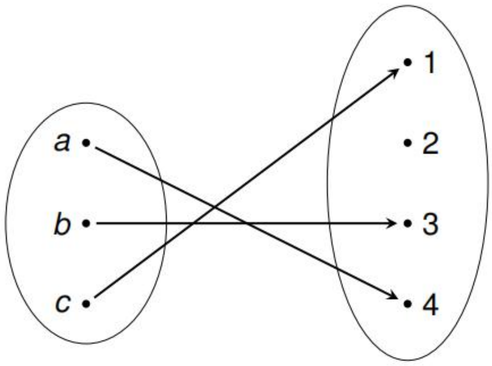
下取整函数将实数 x 向下取到小于或等于 x 的最大整数，表示为"$\lfloor x \rfloor$"。

范例：下取整函数
$\lfloor 3.2 \rfloor = 3$ $\lfloor 35.9 \rfloor = 5$ $\lfloor -0.5 \rfloor =- 1$

3.3 函数的性质

一对一函数(One-to-one)/单射函数(Injection)

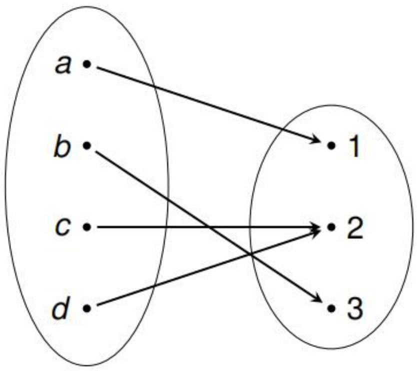
一对一函数/单射函数是指对于函数 f 的定义域中所有的 a 和 b，如果 $a \neq b$，那么 $f(a) \neq f(b)$。



范例：一对一函数/单射函数
$f(x) = x + 1$ 是一对一函数。 $f(x) = x^2$ 不是一对一函数，因为 $f(1) = f(-1) = 1$。

映上函数(Onto)/满射函数(Surjection)

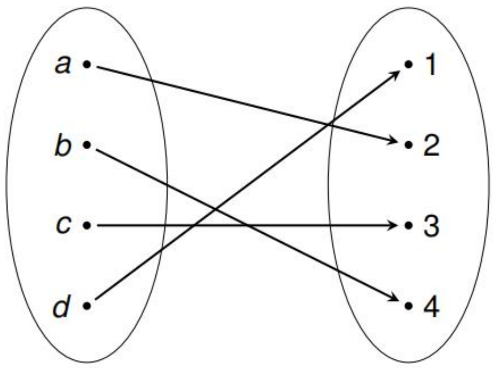
映上函数/满射函数是指对于函数 $f: A \rightarrow B$，每个 $b \in B$ 都有元素 $a \in A$ 使得 $f(a) = b$。



范例：映上函数/满射函数
f: ℤ → ℤ, f(x) = x + 1 是映上函数。
f: ℤ → ℤ, f(x) = x^2 不是映上函数，因为没有整数 x 使 x^2 = -1。

一一对应函数(One-to-One Correspondance)/双射函数(Bijection)

如果一个函数**既是一对一函数又是映上函数**，那么这个函数就被称为**一一对应函数/双射函数**。



范例：一一对应函数/双射函数
f 是从{a, b, c, d}到{1, 2, 3, 4}的函数， 定义 f(a) = 4， f(b) = 2， f(c) = 1， f(d) = 3。
函数 f 是单射函数，因为没有两个值映射到相同的函数值。
函数 f 是满射函数，因为陪域的个数与值域的个数相同。
因此，函数 f 是双射函数。

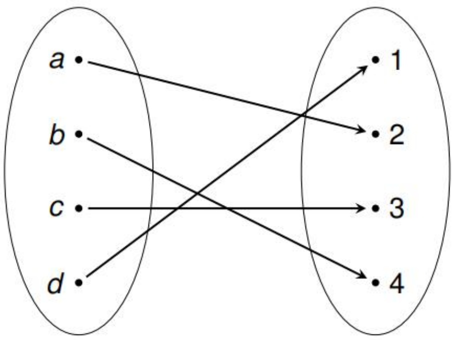
3.4 反函数

反函数(Inverse Function)

假设有一个从集合 A 到集合 B 的双射函数 f。由于 f 是满射函数，所以 B 中的每个元素都是 A 中某些元素的像；又由于 f 还是单射函数，所以 B 的每个元素都是 A 中唯一一个元素的像。

于是，通过把 f 的**对应关系颠倒**，获得的从 B 到 A 的新函数被称为 f 的**反函数**，用“f^{-1}”表示。**当 $f(a) = b$ 时，$f^{-1}(b) = a$**。

需要注意，不要将f^{-1}与$\frac{1}{f}$混淆。

范例：反函数	
	
该函数是否有反函数？	是
$f^{-1}(2)$	a
$f^{-1}(1)$	d

范例：计算 f(x) = x + 3 的反函数
$f^{-1}(x) = x - 3$

3.5 合成函数

合成函数(Composition Function)

假设 g 是从集合 A 到集合 B 的函数，f 是从集合 B 到集合 C 的函数。函数 f 和 g 的**合成**，记作“**f ∘ g**”。

$$(\boldsymbol{f} \circ \boldsymbol{g})(\boldsymbol{a}) = \boldsymbol{f}(\boldsymbol{g}(\boldsymbol{a}))$$

函数合成的顺序很重要，f ∘ g与g ∘ f并不相等。

范例：合成函数
$f: \mathbb{R}^+ \to \mathbb{R}^+, f(x) = x^3$ $g: \mathbb{R}^+ \to \mathbb{R}^+, g(x) = x + 2$ $(f \circ g)(x) = f(g(x)) = (x+2)^3$ $(g \circ f)(x) = g(f(x)) = x^3 + 2$

恒等函数(Identity Function)

如果一个从集合 A 到集合 B 的函数 f 有反函数，那么 f 与f^{-1}的合成函数得到的是**恒等函数**。

如果 f(a) = b，那么$f^{-1}(b) = a$。 $(f^{-1} \circ f)(a) = f^{-1}(f(a)) = f^{-1}(b) = a$

3.6 指数函数/对数函数

指数函数(Exponential Function)

指数函数的定义为$y = a^x (a > 0|a \neq 1)$，其中 a 称为**底数**(base)，x 称为**指数**(exponent)。

指数函数满足以下运算法则：

$$a^m \cdot a^n = a^{m+n}$$

$$(a^m)^n = a^{m+n}$$

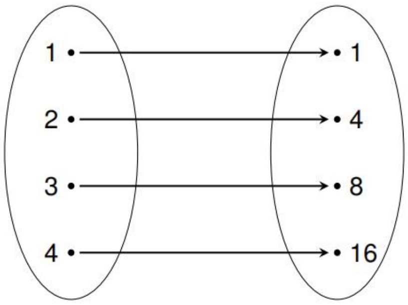
$$\frac{a^x}{a^y} = a^{x-y}$$

$$(a \times b)^x = a^x \times b^x$$

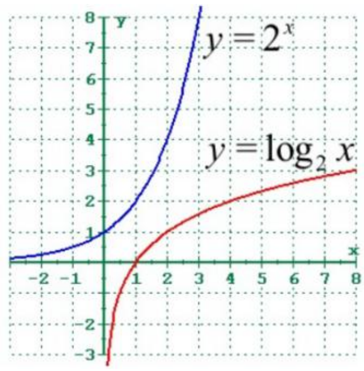
范例：指数函数
$\left(6^{2k}\right)^3 = 6^{6k}$ $6^{k^2} \times 6 = 6^{k^2+1}$ $\frac{3^k}{9} = \frac{3^k}{3^2} = 3^{k-2}$ $3^k \times 27 = 3^k \times 3^3 = 3^{k+3}$

对数函数(Logarithm Function)

对于函数 f: {1, 2, 3, 4} → {1, 4, 8, 16}，$f(x) = 2^x$，指数函数是**双射函数**，因此它是有**反函数**的。



对数函数是指数函数的反函数，对数函数的定义为 $y = \log_a x(a > 0|a \neq 1)$，其中 a 称为**底数**。



对数函数满足以下运算法则：

$$\boldsymbol{log_a x \times y = log_a x + log_a y}$$

$$\boldsymbol{log_a \frac{x}{y} = log_a x - log_a y}$$

$$\boldsymbol{log_a x^y = y \times log_b x}$$

$$\boldsymbol{log_b x = \frac{log_a x}{log_a b}}$$

范例：对数函数
$log_5 k + log_5 2 = log_5(2k)$ $log_2 5^2 = 2 \times log_2 5$ $\frac{log_3(k^2)}{log_3 25} = \frac{2 \times log_3 k}{log_3 5^2} = \frac{2 \times log_3 k}{2 \times log_3 5} = log_5 k$

第 4 章 数论

4.1 进制转换

进制

日常生活中都用**十进制(decimal)**来表示整数，十进制数由 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 这 10 个字符组成。一个十进制整数的第 k 位的值可以由10^{k-1}计算得到。

范例：十进制
$256 = 200 + 50 + 6$ $= 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0$

二进制(binary)、八进制(octal)、十六进制(hexadecimal)也是非常常用的表示法，例如**计算机通常用二进制来做算术运算**，而**用八进制或十六进制来表示字符**。

十进制	二进制	八进制	十六进制
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7

8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

<u>进制转换</u>

一个 b 进制的正整数 n 可以唯一地构造**展开式**：

$$\mathbf{n = a_k \times b^k + a_{k-1} \times b^{k-1} + \ldots + a_1 \times b^1 + a_0 \times b^0}$$

范例：b 进制展开式
$11 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = (1011)_2$ $197 = 2 \times 3^4 + 1 \times 3^3 + 0 \times 3^2 + 2 \times 3^1 + 2 \times 3^0 = (21022)_3$

范例：b 进制转十进制
$(21331)_4 = 2 \times 4^4 + 1 \times 4^3 + 3 \times 4^2 + 3 \times 4^1 + 1 \times 4^0$ $= 512 + 64 + 48 + 12 + 1$ $= 637$

十进制转 b 进制还可以使用**短除法**的方式。

范例：十进制转 b 进制

4	637	1
4	159	3
4	39	3
4	9	1
4	2	2

$(21331)_4$

2	637	1
2	318	0
2	159	1
2	79	1
2	39	1
2	19	1
2	9	1
2	4	0
2	2	0
2	1	1

$(1001111101)_2$

8	1000	0
8	125	5
8	15	7
8	1	1

$(1750)_8$

4.2 素数

素数(Prime Number)

基于**整除性**的一个重要概念就是素数，**素数是大于 1 的且不能被 1 和它自身以外的正整数整除的整数**。素数是**现代密码学**中必不可少的一部分，密码学中的大素数就用在**信息加密**的某些方法中。

范例：素数
7 是素数，因子是 1 和 7。 9 是合数，9 能被 3 整除。

每个大于 1 的整数都可以唯一地写成多个素数的乘积。

范例：素因子分解式
$100 = 2 * 2 * 5 * 5 = 2^2 * 5^2$ $999 = 3 * 3 * 3 * 37 = 3^3 * 37$ $1024 = 2^{10}$

如果 n 是一个合数，那么 n 必有一个素因子小于或等于$\sqrt{n}$。

范例：证明 101 是素数
不超过$\sqrt{101}$的素数只有 2, 3, 5, 7，因为 101 不能被 2, 3, 5, 7 整除，所以 101 是素数。

范例：判断素数

```
import math

def is_prime(num):
    for i in range(2, int(math.sqrt(num)) + 1):
```

```python
        if num % i == 0:
            return False
    return True

def main():
    print(is_prime(13))
    print(is_prime(18))

if __name__ == "__main__":
    main()
```

运行结果	True False

埃拉托斯特尼筛法(Sieve of Eratosthenes)可以用来寻找不超过一个给定整数的所有素数。步骤如下：

1. 建立包含所有给定整数以内的表格。
2. 从 i = 2 开始。
3. 移除所有整数 n % i == 0（除 i 以外）。
4. i = i + 1。
5. 重复第 3 步和第 4 步。

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~1~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~

4.3 序列

序列(Sequence)

序列是一种用来表示有序列表的离散结构。例如 1, 2, 3, 5, 8 是一个含有五项的序列，而 1, 3, 9, 27, 81, ..., 3^n, ...是一个无穷序列。序列可以用记号“$\{a_n\}$”表示。

范例：序列
序列$\{a_n\}$，其中$a_n = \frac{1}{n}$：$1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots$ 序列$\{a_n\}$，其中$a_n = 2^n$：1, 2, 4, 8, 16, 32, ...

如果一个序列任意相邻的两项满足$a_k < a_{k+1}$，那么这个序列被称为递增序列(increasing sequence)。

如果一个序列任意相邻的两项满足$a_k \leq a_{k+1}$，那么这个序列被称为非递减序列(non-decreasing sequence)。

如果一个序列任意相邻的两项满足$a_k > a_{k+1}$，那么这个序列被称为递减序列(decreasing sequence)。

如果一个序列任意相邻的两项满足$a_k \geq a_{k+1}$，那么这个序列被称为非递增序列(non-increasing sequence)。

算术级数(Arithmetic Sequence)

算术级数也称“等差级数”，序列形式如下：

$$\mathbf{a, a+d, a+2d, \ldots, a+nd, \ldots\ (a, d \in \mathbb{R})}$$

范例：算术级数

序列$\{a_n\}$，其中$a_n = -1 + 4n$：-1, 3, 7, 11, ...
序列$\{b_n\}$，其中$b_n = 7 - 3n$：7, 4, 1, -2, ...

几何级数(Geometric Sequence)

几何级数也称“**等比级数**”，序列形式如下：

$$\mathbf{a, ar, ar^2, \dots, ar^n, \dots (a, r \in \mathbb{R})}$$

范例：几何级数
序列$\{a_n\}$，其中$a_n = (-1)^n$：1, -1, 1, -1, 1, ...
序列$\{b_n\}$，其中$b_n = 2 \times 5^n$：2, 10, 50, 250, 1250, ...
序列$\{c_n\}$，其中$c_n = 6 \times \left(\frac{1}{3}\right)^n$：$6, 2, \frac{2}{3}, \frac{2}{9}, \frac{2}{27}, \dots$

4.4 递推关系

递推关系(Recurrence Relation)

如果数列$\{a_n\}$的第 n 项与它前一项的关系可以用一个式子来表示，那么这个公式就叫做这个数列的**递推公式**。

算术级数的递推关系如下：

$$\boldsymbol{a_0 = a}$$

$$\boldsymbol{a_n = a_{n-1} + d}$$

几何级数的递推关系如下：

$$\boldsymbol{a_0 = a}$$

$$\boldsymbol{a_n = a_{n-1} \times r}$$

范例：递推关系
$\{a_n\}$是一个序列，满足递归关系$a_n = a_{n-1} + 3, n = 1, 2, 3, \ldots$ $a_0 = 2$ $a_1 = a_0 + 3 = 2 + 3 = 5$ $a_2 = a_1 + 3 = 5 + 3 = 8$ $a_3 = a_2 + 3 = 8 + 3 = 11$

斐波那契数列(Fibonacci Sequence)

斐波那契数列$f_0, f_1, f_2, \ldots$，由初始条件$f_0 = 0, f_1 = 1$ 递推关系$f_n = f_{n-1} + f_{n-2}$定义。

范例：斐波那契数列

$$f_2 = f_1 + f_0 = 1 + 0 = 1$$

$$f_3 = f_2 + f_1 = 1 + 1 = 2$$

$$f_4 = f_3 + f_2 = 2 + 1 = 3$$

$$f_5 = f_4 + f_3 = 3 + 2 = 5$$

$$f_6 = f_5 + f_4 = 5 + 3 = 8$$

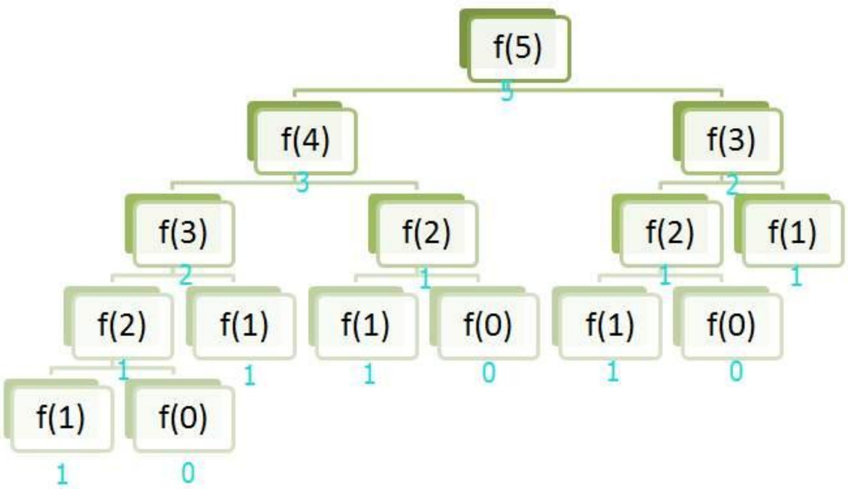
范例：斐波那契数列（递归）

```
def fibonacci(n):
    if n == 1 or n == 2:
        return 1
    return fibonacci(n-2) + fibonacci(n-1)

def main():
    n = 7
    print("斐波那契数列第%d 位：%d" % (n, fibonacci(n)))

if __name__ == "__main__":
    main()
```

运行结果	斐波那契数列第 7 位：13



范例：斐波那契数列（迭代）

```
def fibonacci(n):
```

```
    f = [0] * n
    f[0] = f[1] = 1
    for i in range(2, n):
        f[i] = f[i-2] + f[i-1]
    return f[n-1]

def main():
    n = 7
    print("斐波那契数列第%d 位：%d" % (n, fibonacci(n)))

if __name__ == "__main__":
    main()
```

运行结果	斐波那契数列第 7 位：13

范例：银行储蓄账户上有 10000 元，年利率为 5.8%，7 年后账户中将有多少钱？

$P_n = P_{n-1} + 0.058P_{n-1} = (1.058)P_{n-1}$

$P_0 = 10000$

$P_1 = (1.058)P_0$

$P_2 = (1.058)P_1 = (1.058)^2P_0$

$\ldots$

$P_7 = (1.058)P_6 = (1.058)^7P_0 \approx 14838.83$

4.5 求和

求和(Summation)

求和记号“**Σ**”可以用于表示序列中所有项的累加和。

在$\sum_{i=lower}^{upper} a_i$中，i 用于表示求和**下标**，lower 为**下限**，upper 为**上限**。

范例：求和
$\sum_{i=1}^{100} i = 1 + 2 + 3 + \ldots + 99 + 100 = 5050$
$\sum_{j=1}^{5} j^2 = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 55$
$\sum_{k=4}^{6} (-1)^k = (-1)^4 + (-1)^5 + (-1)^6 = 1 - 1 + 1 = 1$

双重求和

很多情况下需要使用双重求和，比如在计算机程序中嵌套循环的分析中。

计算双重求和的方法是先展开内层求和，再继续计算外层求和。

范例：双重求和
$\sum_{i=1}^{4}\sum_{j=1}^{3} ij = \sum_{i=1}^{4}(i + 2i + 3i)$
$= \sum_{i=1}^{4} 6i$
$= 6 + 12 + 18 + 24$
$= 60$

4.6 数学归纳法

数学归纳法(Mathematical Induction)

数学归纳法是一种数学证明方法，通常被用于证明某个给定命题在一个给定范围内成立。

数学归纳法分为三个步骤：**归纳基础**、**归纳假设**、**归纳递推**。

范例：证明$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, n \in \mathbb{Z}^+$

1. 归纳基础：假设 n = 1。

$$\sum_{i=1}^{1} i = \frac{1(1+1)}{2} = 1$$

2. 归纳假设：假设 n = k 时，

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$$

成立。

3. 归纳递推：证明 n = k+1 时，

$$\sum_{i=1}^{k+1} i = \frac{(k+1)(k+2)}{2}$$

成立。

$$\begin{aligned}\sum_{i=1}^{k+1} i &= \sum_{i=1}^{k} i + k + 1 \\ &= \frac{k(k+1)}{2} + k + 1 \\ &= \frac{k(k+1) + 2(k+1)}{2} \\ &= \frac{(\mathrm{k}+1)(\mathrm{k}+2)}{2}\end{aligned}$$

范例：证明$2^n \geq 3n, n \geq 4$
1. 归纳基础：假设 n = 4。 $$2^4 \geq 3 \times 4$$ 2. 归纳假设：假设 n = k (k ≥ 4)时， $$2^k \geq 3k$$ 成立。 3. 归纳递推：证明 n = k+1 时， $$2^{k+1} \geq 3(k+1)$$ 成立。 $2^{k+1} = 2 \times 2^k$ $\geq 2 \times 3\mathrm{k}$ $= 3\mathrm{k} + 3\mathrm{k}$ $\geq 3\mathrm{k} + 3$ $\geq 3(\mathrm{k} + 1)$