2.4.2 输出植树证书

蚂蚁森林是某公司客户端发起的“碳账户”的一款公益活动，用户通过步行、地铁出行、在线消费等行为，可在蚂蚁森林中获取能量，当能量达到一定数值后，用户可以在支付宝中申请种植一棵树，申请成功后会收到某公司发放的一张植树证书。植树证书中包含树苗编号、申请时间、种植地点等信息，具体如图 2-4 所示。

输出植树证书

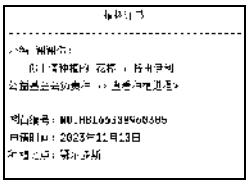

图2-4 植树证书

本案例要求编写代码，根据用户输入的昵称和植物名称实现输出图 2-4 所示植树证书的功能。

2.5 数字类型

2.5.1 整型

整型用于表示整数，例如 100、101、-100、-101 等。Python 中整型数据的大小没有限制，只要计算机的内存足够大，用户就无须考虑内存溢出问题。

整型常用的记数方式有 4 种，分别是二进制、八进制、十进制和十六进制，默认的记数方式为十进制，其中二进制数以“0B”或“0b”开头，八进制数以数字“0o”或“0O”开头，十六进制数以“0x”或“0X”开头。接下来，分别以 4 种记数方式表示整型数据 5，示例代码如下：

```
5                                    # 十进制数
0b101                                # 二进制数
0o5                                  # 八进制数
0x5                                  # 十六进制数
```

为了方便用户使用不同进制的数据，Python 中提供了一些用于转换数据进制的函数，分别是 bin()、oct()、int()、hex()，这些函数都可以将一种进制的整型数据转换为其他进制的整型数据，如表 2-1 所示。

表 2-1 用于整型数据的进制转换的函数

函数	说明
bin(x)	将 x 转换为二进制数
oct(x)	将 x 转换为八进制数
int(x)	将 x 转换为十进制数
hex(x)	将 x 转换为十六进制数

下面演示表 2-1 中各函数的用法，示例代码如下：

```
decimal = 10                         # 十进制数
bin_num = 0b1010                     # 二进制数
print(bin(decimal))                  # 将十进制数 10 转换为二进制数
print(oct(decimal))                  # 将十进制数 10 转换为八进制数
print(int(bin_num))                  # 将二进制数 0b1010 转换为十进制数
print(hex(decimal))                  # 将十进制数 10 转换为十六进制数
```

运行代码，结果如下所示：

```
0b1010
0o12
10
0xa
```

2.5.2　浮点型

浮点型用于表示实数，实数由整数部分、小数点和小数部分组成，如 3.1415926、0.90、-10.0。Python 中一般可以直接使用小数点的形式表示浮点型数据，示例如下：

```
1.09999          1.2021            314.15926
-2.36            -10.0632          -100.03
```

当需要表示较大或较小的实数时，直接使用小数点形式会出现冗长的数字。为了简化这种表示形式，可以使用科学记数法表示浮点型数据。科学记数法会把一个数表示成 a 与 10 的 n 次幂相乘的形式：

$$a\times10^{n}\ (1 \leqslant |a|<10,\ n \in \mathbf{N})$$

Python 使用字母 e 或 E 表示底数 10，示例代码如下：

```
-3.14e10                          # 相当于-3.14×10^10，结果为-31400000000
3.14e-10                          # 相当于 3.14×10^-10，结果为 0.000000000314
```

Python 中的浮点型是双精度的，每个浮点型数据占 8 个字节（即 64 位）的存储空间。这种双精度的浮点型数据遵守 IEEE 754 标准，其中 52 位用于存储尾数，11 位用于存储阶码，剩余 1 位用于存储符号。

Python 中浮点型数据的取值范围为-1.8e308～1.8e308，若超出这个范围，Python 会将数据视为无穷大（inf）或无穷小（-inf）。示例代码如下：

```
print(3.14e500)
print(-3.14e500)
```

运行代码，结果如下所示：

```
inf
-inf
```

2.5.3　复数类型

复数类型用于表示复数，复数的一般形式为 real+imagj 或 real+imagJ，其中 real 是实部，表示复数的实数部分；imag 是虚部，表示复数的虚数部分；j 或 J 是虚数单位。注意，实部可以直接省略；虚部后面必须有虚数单位，虚数单位不能单独存在。示例代码如下：

```
complex_one = 1 + 2j              # 实部为 1，虚部为 2
complex_two = 2j                  # 虚部为 2
```

可以通过 complex()函数创建复数，该函数的使用方式为 complex(实部,虚部)，我们需要根据需求传入相应的实部和虚部。若没有传入虚部，则虚部默认为 0。示例代码如下：

```
complex_one = complex(3, 2)     # 创建复数，分别传入实部和虚部
print(complex_one)
complex_two = complex(5)        # 创建复数，只传入实部
print(complex_two)
```

运行代码，结果如下所示：

```
(3+2j)
(5+0j)
```

从上述结果可以看出，控制台中输出了两个复数，即(3+2j)和(5+0j)，可能有的读者会有这样的疑问：为什么输出结果中的复数有小括号呢？实际上这是 print()函数输出复数的默认格式，默认会使用小括号包裹复数，以便清晰地展示出复数是实部和虚部的组合。

通过 real 和 imag 属性可以单独获取复数的实部和虚部，具体使用方式为“复数.real”和“复数.imag”，示例代码如下：

```
complex_one = 1 + 2j
print(complex_one.real)                    # 获取复数的实部
print(complex_one.imag)                    # 获取复数的虚部
```

运行代码，结果如下所示：

```
1.0
2.0
```

2.5.4 布尔类型

布尔类型用于表示逻辑判断的真或假，真对应的取值是 True，假对应的取值是 False。Python 中，对任何数据进行逻辑判断后都可以得到一个布尔值，布尔值为 False 的常见数据如下：

- None；
- 任何值为 0 的数字类型的数据，如 0、0.0、0j；
- 任何空的组合数据类型的数据，如空字符串、空元组、空列表、空集合、空字典。

上述数据中，None 是一个特殊的空值，表示没有值。除了上述列举的数据外，其他数据的布尔值一般都是 True。Python 中可以使用 bool()函数检测数据的布尔值，示例代码如下：

```
print(bool(None))                  # 检测 None 的布尔值
print(bool(0))                     # 检测整型数据 0 的布尔值
print(bool(3.1415))                # 检测浮点型数据 3.1415 的布尔值
print(bool(0j))                    # 检测复数类型数据 0j 的布尔值
print(bool('hello'))               # 检测字符串'hello'的布尔值
print(bool(1))                     # 检测整型数据 1 的布尔值
```

运行代码，结果如下所示：

```
False
False
True
False
True
True
```

从上述结果可以看出，None、0 和 0j 对应的布尔值都是 False，3.1415、'hello'和 1 对应的布尔值都是 True。

2.5.5 数字类型转换

Python 内置了一系列可强制转换数据类型的函数，使用这些函数可将目标数据转换为指定的类型，其中用于转换数字类型的函数有 int()、float()、complex()，如表 2-2 所示。

表 2-2 用于转换数字类型的函数

函数	说明
int(x[, base])	将 x 转换为整型数据，x 的值可以是整型、浮点型、布尔类型的数据，或者符合整型规范的字符串。base 表示进制数，它的取值为 2 到 36 的整数，如果没有使用 base 指定进制数，则会将 x 转换为十进制整数

续表

函数	说明
float(x)	将 x 转换为一个浮点型数据，x 的值可以是整型、浮点型、布尔类型的数据，或者符合浮点型规范的字符串
complex(x)	将 x 转换为复数类型数据，x 的值可以是任意数字类型的数据，或者符合复数类型规范的字符串

需要注意的是，当通过 int()函数将浮点型数据转换为整型数据时只保留整数部分，舍去小数部分。另外，通过 int()函数将布尔类型的数据转换为整型数据时，会将 True 转换成 1，将 False 转换成 0。

下面演示表 2-2 中各函数的用法，示例代码如下：

```
num_one = 2.0
print(int(num_one))                    # 将浮点型数据转换为整型数据
num_two = 5
print(float(num_two))                  # 将整型数据转换为浮点型数据
print(complex(num_one))                # 将浮点型数据转换为复数类型的数据
words_one = '10.01'
print(float(words_one))                # 将字符串类型的数据转换为浮点型数据
words_two = '10'
print(int(words_two))                  # 将字符串类型的数据转换为整型数据
words_three = '1+2j'
print(complex(words_three))            # 将字符串类型的数据转换为复数类型的数据
```

运行代码，结果如下所示：

```
2
5.0
(2+0j)
10.01
10
(1+2j)
```

注意，如果字符串中的所有内容是除十进制以外的其他进制的数据，那么使用 int()函数将该字符串转换成整型数据时，需要使用 base 指定要转换的进制数。示例代码如下：

```
words_four = '0b1010'          # 字符串中包含二进制数
print(int(words_four, base=2)) # 将字符串转换为整型数据时，通过 base 指定要转换为二进制数据
```

运行代码，结果如下所示：

```
10
```

2.6　运算符

Python 运算符是一种特殊的符号，主要用于在表达式中对操作数执行特定操作或者计算。根据操作数数量的不同，运算符可以分为单目运算符和双目运算符；根据运算符功能的不同，运算符可以分为算术运算符、赋值运算符、比较运算符、逻辑运算符、成员运算符和位运算符。当一个表达式中包含多个运算符时，Python 将根据运算符的优先级确定它们的运算顺序。本节将按功能详细介绍 Python 中的运算符，并介绍运算符的优先级。

2.6.1 算术运算符

算术运算符主要用于执行基本的数学运算，比如加法、减法、乘法、除法等。Python 中的算术运算符包括+、-、*、/、//、%和**，它们都是双目运算符，能够对两个操作数进行相应的数学运算。

下面以 a = 2，b = 8 为例，介绍算术运算符的功能及示例，具体如表 2-3 所示。

表 2-3 算术运算符

运算符	功能说明	示例
+	加法运算符，用于计算两个操作数相加后的和	a + b，结果为 10
-	减法运算符，用于计算两个操作数相减后的差	a -b，结果为-6
*	乘法运算符，用于计算两个操作数相乘后的积	a * b，结果为 16
/	除法运算符，用于计算两个操作数相除后的商，结果是浮点数类型	a / b，结果为 0.25
//	整除运算符，用于计算两个操作数相除后的商，结果取商的整数部分	a // b，结果为 0
%	取模运算符，用于计算两个操作数相除后的余数	a % b，结果为 2
**	幂运算符，用于对数值进行幂运算	a ** b，结果为 256

Python 中的算术运算符既支持对相同类型的数值进行运算，也支持对不同类型的数值进行混合运算。在进行混合运算时 Python 会强制将数值的类型进行类型转换，类型转换遵循如下原则。

（1）整型与浮点型进行混合运算时，将整型转化为浮点型。

（2）其他类型与复数类型进行运算时，将其他类型转换为复数类型。

接下来，使用整型数据分别与浮点型和复数类型的数据进行运算，示例代码如下：

```
print(10 / 2.0)                    # 整型数据 10 会转换为浮点型的数据 10.0
print(10 - (3 + 5j))               # 整型数据 10 会转换为复数类型的数据 10+0j
```

运行代码，结果如下所示：

```
5.0
(7-5j)
```

2.6.2 赋值运算符

赋值运算符用于将一个表达式或值赋给变量。最简单的赋值运算符是=，例如，使用赋值运算符=将整数 3 赋给变量 num，示例代码如下：

```
num = 3
```

赋值运算符=允许同时为多个变量赋值，既可以为多个变量赋同一个值，也可以为多个变量赋不同的值，示例代码如下：

```
x = y = z = 1                      # 变量 x、y、z 均赋为 1
a, b = 1, 2                        # 变量 a 赋为 1，变量 b 赋为 2
```

注意，当使用赋值运算符=同时为多个变量赋不同的值时，赋值运算符=左右两边的变量和值的数量必须保持一致，否则运行时会出现错误。

除此之外，Python 还有一种复合赋值运算符，复合赋值运算符同时具备算术运算和赋值两项功能。下面以变量 num 为例，介绍 Python 复合赋值运算符的功能及示例，具体如表 2-4 所示。

表 2-4　复合赋值运算符

运算符	功能说明	示例
+=	将右边的值与左边的变量相加，并将结果赋给左边的变量	num += 2 等价于 num = num+2
-=	将右边的值与左边的变量相减，并将结果赋给左边的变量	num -= 2 等价于 num = num - 2
*=	将右边的值与左边的变量相乘，并将结果赋给左边的变量	num *= 2 等价于 num = num* 2
/=	将左边的变量除以右边的值，并将结果赋给左边的变量	num /= 2 等价于 num = num/2
//=	将左边的变量整除右边的值，并将结果赋给左边的变量	num //= 2 等价于 num = num//2
%=	将左边的变量对右边的值取模，并将结果赋给左边的变量	num %= 2 等价于 num = num%2
**=	将左边的变量提升至右边值的次幂，并将结果赋给左边的变量	num **= 2 等价于 num = num**2

除了以上赋值运算符外，Python 3.8 中还新增了一个赋值运算符——海象运算符“:=”，该运算符用于在表达式内部同时为变量赋值和使用变量，因其形似海象的眼睛和长牙而得名。海象运算符的用法示例如下：

```
num_one = 1
result = num_one + (num_two:=2)   # 使用海象运算符为变量 num_two 赋值
print(result)
```

运行代码，结果如下所示：

```
3
```

2.6.3　比较运算符

比较运算符也叫关系运算符，用于比较其两边操作数。Python 中的比较运算符包括==、!=、>、<、>=、<=，它们通常用于布尔测试，测试的结果只能是 True 或 False。

下面以 x=2，y=3 为例，介绍比较运算符的功能及示例，具体如表 2-5 所示。

表 2-5　比较运算符

运算符	功能说明	示例
==	比较两个操作数是否相等，如果相等返回 True	x==y，返回 False
!=	比较两个操作数是否不相等，如果不相等返回 True	x!=y，返回 True
>	比较左操作数是否大于右操作数，如果大于返回 True	x>y，返回 False
<	比较左操作数是否小于右操作数，如果小于返回 True	x<y，返回 True
>=	比较左操作数是否大于或等于右操作数，如果大于或等于返回 True	x>=y，返回 False
<=	比较左操作数是否小于或等于右操作数，如果小于或等于返回 True	x<=y，返回 True

2.6.4　逻辑运算符

逻辑运算符可以把多个条件按照逻辑进行连接，变成更复杂的条件。Python 中使用 and、or、not 这 3 个关键字作为逻辑运算符，其中 and 与 or 为双目运算符，分别用于对两个操作数进行逻辑与、逻辑或运算；not 为单目运算符，用于对一个操作数进行逻辑非运算。

下面以 x=10，y=20 为例，介绍逻辑运算符的功能以及示例，具体如表 2-6 所示。

表 2-6　逻辑运算符

运算符	功能说明	示例
and	若两个操作数的布尔值均为 True，则返回右操作数，运算结果的布尔值为 True；若两个操作数的布尔值均为 False，则会返回左操作数，运算结果的布尔值为 False；若两个操作数的布尔值有一个为 False，则会返回布尔值为 False 的操作数	x and y 的结果为 20

续表

运算符	功能说明	示例
or	若左操作数的布尔值为 True，则返回左操作数，否则返回右操作数	x or y 的结果为 10
not	若操作数的布尔值为 True，则结果为 False	not x 的结果为 False

2.6.5 成员运算符

成员运算符用于检测给定数据是否存在于字符串、列表、元组、集合、字典等之中，关于它们的介绍如下。

- in：如果指定数据在上述数据类型中返回 True，否则返回 False。
- not in：如果指定数据不在上述数据类型中返回 True，否则返回 False。

成员运算符的用法示例如下：

```
x = 'Python'
y = 'P'
print(y in x)
print(y not in x)
```

运行代码，结果如下所示：

```
True
False
```

2.6.6 位运算符

位运算符用于对操作数的二进制位进行逻辑运算，操作数必须为整数。Python 中一共有 6 个位运算符，分别是<<、>>、&、|、^、 ~ 。下面通过一张表罗列这些位运算符的功能，具体如表 2-7 所示。

表 2-7 位运算符

运算符	功能说明
<<	操作数按位左移
>>	操作数按位右移
&	左操作数与右操作数执行按位与运算
|	左操作数与右操作数执行按位或运算
^	左操作数与右操作数执行按位异或运算
~	操作数按位取反

表 2-7 中每个位运算符的功能可能不容易理解，接下来，逐一介绍表 2-7 中罗列的位运算符，并通过示例进行演示，具体内容如下。

1. 按位左移运算符（<<）

按位左移是指将二进制形式操作数的所有位向左移动指定的位数，移出位被丢弃，移入位补 0。以十进制的整数 9 为例，当执行 9<<4 运算时，首先会将 9 自动转换为二进制数 00001001，然后将二进制数 00001001 的所有位向左移动 4 位，其运算过程和结果如图 2-5 所示。

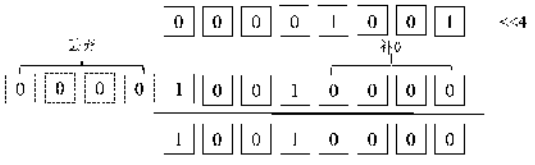

图2-5 按位左移的运算过程和结果

从图 2-5 中可以看出，二进制数 00001001 按位左移 4 位后的结果为 10010000。接下来，通过代码实现 9<<4 运算，示例代码如下：

```
a = 9
print(bin(a<<4))                    # 使用<<将变量 a 的值按位左移 4 位
```

运行代码，结果如下所示：

```
0b10010000
```

按位左移 n 位相当于操作数乘 2 的 n 次方，根据此原理可借助乘法运算符实现按位左移，例如，将 10 按位左移 3 位，利用乘法运算符进行计算即 10×2^3。

2. 按位右移运算符（>>）

按位右移是指将二进制形式操作数的所有位向右移动指定的位数，移出位被丢弃，移入位补 0。以十进制的整数 8 为例，当执行 8>>2 运算时，首先会将 8 自动转换为二进制数 00001000，然后将二进制数 00001000 的所有位向右移动 2 位，其运算过程和结果如图 2-6 所示。

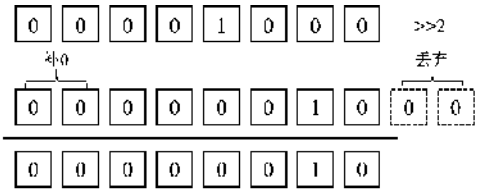

图2-6　按位右移的运算过程和结果

从图 2-6 中可以看出，二进制数 00001000 按位右移 2 位后的结果为 00000010。接下来，通过代码实现 8<<2 运算，示例代码如下：

```
a = 8
print(bin(a>>2))                    # 使用>>将变量 a 的值按位右移 2 位
```

运行代码，结果如下所示：

```
0b10
```

从上述结果可以看出，程序输出的结果为 0b10，之所以不是 0b00000010，是因为 Python 会自动省略没有实际意义的 0，并不改变数值。

按位右移 n 位相当于操作数除以 2 的 n 次方并向下取整，根据此原理可借助除法运算符实现按位右移运算，例如，将 10 按位右移 3 位，利用除法运算符进行计算即 $10\div2^3$，其结果等于 1.25，向下取整后为 1。

3. 按位与运算符（&）

按位与是指将参与运算的两个操作数对应的二进制位进行逻辑与运算。当对应的两个二进制位均为 1 时，结果位就为 1，否则为 0。以十进制的整数 9 和 3 为例，当执行 9&3 运算时，首先会将 9 和 3 分别自动转换为二进制数 00001001 和 00000011，然后将二进制数 00001001 和 00000011 的对应位进行逻辑与运算，其运算过程和结果如图 2-7 所示。

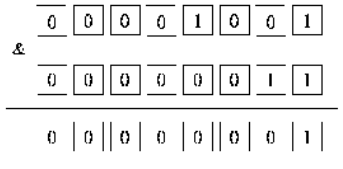

图2-7　按位与的运算过程和结果

从图 2-7 中可以看出，二进制数 00001001 和 00000011 进行按位与运算后的结果为 00000001。接下来，通过代码实现 9&3 运算，示例代码如下：

```
a = 9
b = 3
print(bin(a&b))                     # 使用&将变量 a 和 b 的值进行按位与运算
```

运行代码，结果如下所示：

```
0b1
```

4. 按位或运算符（|）

按位或是指将参与运算的两个操作数对应的二进制位进行逻辑或运算。若对应的两个二进制位有一个为 1 时，结果位就为 1，否则为 0。若参与运算的操作数为负数，则参与运算的两个操作数均以补码出现。以十进制整数 8 和 3 为例，当执行 8|3 运算时，首先会将 8 和 3 分别自动转换为二进制数 00001000 和 00000011，然后将二进制数 00001000 和 00000011 的对应位进行逻辑或运算，其运算过程和结果如图 2-8 所示。

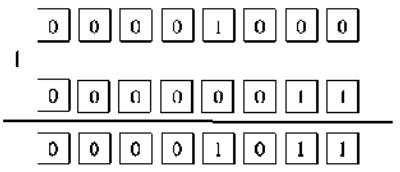

图2-8 按位或的运算过程和运算

从图 2-8 中可以看出，二进制数 00001000 和 00000011 进行按位或运算后的结果为 00001011。接下来，通过代码实现 8 | 3 运算，示例代码如下：

```
a = 8
b = 3
print(bin(a|b))                    # 使用|将变量 a 和 b 的值进行按位或运算
```

运行代码，结果如下所示：

```
0b1011
```

5. 按位异或运算符（^）

按位异或是指将参与运算的两个操作数对应的二进制位进行异或运算。当对应的两个二进制位中有一个为 1、另一个为 0 时，结果位为 1，否则结果位为 0。以十进制整数 8 和 4 为例，当执行 8^4 运算时，首先会将 8 和 4 分别自动转换为二进制数 00001000 和 00000100，然后将二进制数 00001000 和 00000100 的对应位进行异或运算，其运算过程和结果如图 2-9 所示。

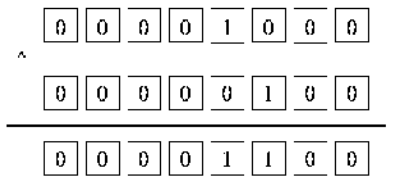

图2-9 按位异或的运算过程和运算

从图 2-9 中可以看出，二进制数 00001000 和 00000100 进行按位异或运算后的结果为 00001100。接下来，通过代码实现 8^4 运算，示例代码如下：

```
a = 8
b = 4
print(bin(a^b))                    # 使用^将变量 a 和 b 的值进行按位异或运算
```

运行代码，结果如下所示：

```
0b1100
```

6. 按位取反运算符（~）

按位取反是指将二进制的每一位进行取反，0 取反为 1，1 取反为 0。进行按位取反运算时，首先会获取操作数的补码，然后对补码进行取反，最后将取反结果转换为原码。例如，对 9 进行按位取反的运算过程如下。

（1）获取补码。9 的二进制数是 00001001，因为 9 是正数，计算机中正数的原码、反码和补码相等，所以 9 的补码仍然为 00001001。

（2）进行取反操作。对 9 的补码 00001001 进行取反操作，即将 0 变为 1，将 1 变为 0，取反后结果为 11110110。

（3）转换为原码。将 11110110 转换为原码时，转换规则为符号位不变，其他位取反，末位加 1。因此，11110110 除符号位之外再次取反后的结果为 10001001，末位加 1 后得到的结果为 10001010，即-10。

正数 9 按位取反的计算过程如图 2-10 所示。

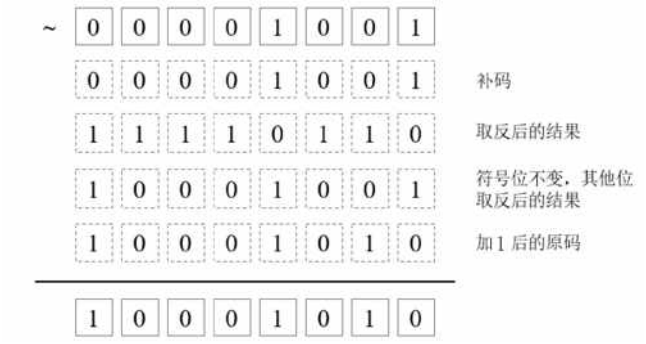

图2-10　正数9按位取反的计算过程

从图 2-10 中可以看出，二进制数 00001001 按位取反后的结果为 10001010。接下来，通过代码实现 ~9 运算，示例代码如下：

```
a = 9
print(bin(~a))                    # 使用~将变量 a 的值进行按位取反运算
```

运行代码，结果如下所示：

```
-0b1010
```

2.6.7　运算符优先级

Python 支持使用多个不同的运算符连接简单表达式实现相对复杂的功能，为了避免含有多个运算符的表达式出现歧义，Python 为每种运算符都设定了优先级。Python 中运算符的优先级如表 2-8 所示。

表 2-8　Python 中运算符的优先级

运算符	描述
**	幂运算符
*、/、%、//	乘法运算符、除法运算符、取模运算符、整除运算符
+、-	加法运算符、减法运算符
>>、<<	按位右移运算符、按位左移运算符
&	按位与运算符
^、|	按位异或运算符、按位或运算符
==、!=、>=、>、<=、<	比较运算符
in、not in	成员运算符
not、and、or	逻辑运算符
+=、-=、*=、/=、//=、%=、**=	复合赋值运算符
=	赋值运算符

需要说明的是，如果表达式中包含小括号，那么解释器会先执行小括号包裹的子表达式。接下来，通过一些示例来验证运算符的优先级，代码如下：

```
a = 20
b = 2
c = 15
```

```
result_01 = (a - b) + c                    # 先执行小括号中的表达式，再执行加法运算
result_02 = a / b % c                      # 先执行除法运算，再执行取余运算
result_03 = c ** b * a                     # 先执行幂运算，再执行乘法运算
print(result_01)
print(result_02)
print(result_03)
```

运行代码，结果如下所示：

```
33
10.0
4500
```

2.7 实训案例

2.7.1 间隔时间计算器

间隔时间计算器

间隔时间计算器是一款精确的时间计算工具，旨在帮助用户计算他们所输入的起始时间和结束时间之间的时间间隔。该计算器可以准确地计算时间间隔，并显示小时数、分钟数和秒数。

本案例要求编写代码，根据用户输入的起始时间和结束时间的小时数和分钟数，计算这两个时间之间的时间间隔。例如，用户输入的起始时间的小时数和分钟数分别为 17 和 10，结束时间的小时数和分钟数分别为 18 和 15，则计算器最终计算的时间间隔为 1 小时 5 分钟。

2.7.2 身体质量指数

身体质量指数

BMI（Body Mass Index，身体质量指数）与人的体重和身高相关，是目前国际常用的衡量人体胖瘦程度以及是否健康的一个指标。已知 BMI 的计算公式如下：

```
BMI= 体重（kg）÷身高（m）÷身高（m）
```

本案例要求编写代码，实现根据用户输入的身高、体重计算 BMI 的功能。

2.8 本章小结

本章主要介绍了 Python 基础知识，包括代码格式、标识符和关键字、变量和数据类型、数字类型以及运算符。本章内容简单易学，希望读者在初学 Python 时结合实训案例对本章内容多加应用，为后期深入学习 Python 打好基础。

2.9 习题

一、填空题

1. Python 中建议使用_____个空格表示一级缩进。
2. 布尔类型的取值包括_____和_____。

3. 使用_____函数可查看数据的类型。
4. float()函数用于将数据转换为_____类型的数据。
5. 若 a=3，b=−2，则 a+=b 执行后 a 的结果为_____。

二、判断题

1. Python 中可以使用关键字作为变量名。（　　）
2. 变量名可以以数字开头。（　　）
3. Python 标识符不区分大小写。（　　）
4. 布尔类型是特殊的浮点型。（　　）
5. 复数类型数据的实数部分可以为 0。（　　）

三、选择题

1. 下列选项中，用于在 Python 代码中添加单行注释的符号是（　　）。

A. #　　B. /　　C. //　　D. <!-- -->

2. 下列选项中，不属于 Python 关键字的是（　　）。

A. name　　B. if　　C. is　　D. and

3. 下列选项中，哪个数据的类型是属于数字类型的？（　　）

A. 0　　B. 1.0　　C. 1+2j　　D. 以上全部

4. 若想要将 2 转换为二进制数 0b10，应该使用下列哪个函数？（　　）

A. oct()　　B. bin()　　C. hex()　　D. int()

5. 下列选项中，不属于 Python 数据类型的是（　　）。

A. bool　　B. dict　　C. string　　D. set

四、简答题

1. 请简单介绍 Python 中的数据类型和数字类型。
2. 请简述 Python 变量的命名规范。
3. 请简单介绍 Python 中的运算符。

五、编程题

1. 编写程序，要求程序能根据用户输入的数据计算圆的面积，并分别输出圆的直径和面积。提示：圆的面积计算公式为 $S = \pi r^2$，π 的取值为 3.14。

2. 已知某煤场有 29.5 吨（t）煤，先用一辆载重 4t 的汽车运送 3 次，剩下的用一辆载重为 2.5t 的汽车运送，请计算还需要运送几次才能运送完？编写程序，解答此问题。