第二章　打印

任务：让计算机在屏幕显示我们想要其显示的内容？
输入：任何 Python 可接受的合法输入。
输出：在计算机屏幕显示对应的输出。

一、任务实现

1、显示：你好，世界！

程序(2-1)

```
print('你好，世界！')
```

执行以上代码，程序运行结果预期如下：

输出(2-1)

```
你好，世界！
```

程序(2-1)中：

- ✓ 完成让计算机显示你好，世界！的程序只有一行代码，该行代码也称**语句**，即可以完成特定功能的合法 python 代码。
- ✓ print()是一个让计算机执行打印的**函数**[1]，其功能是：将括号内给定的输入，打印输出（默认标准输出设备为显示器），因此程序的执行结果就是在显示器上显示你好，世界！
- ✓ 函数的输入可以是 0、1 或多个，一般称为**参数**或**参数列表**。
- ✓ '你好，世界！'是一个**字符串**(str)。

字符串是用引号（单引号与双引号均可）括起来的任意字符序列，如：'你好，世界！'，'Hello, world!'，"4078"等。注意，引号并不是字符串的一部分，只是作为字符串的定界标记。此外，一个合法的字符串必须使用成对的相同引号。类似"4078'，'4078"则不是一个合法的字符串。

Python 的程序代码，除在字符串被引号括起来的内容外，所有代码语句必须使用半角字符（注意，作为字符串定界标记的引号，也必须是半角字符），否则均不合法，运行程序则会不成功，python 解释器会报告语法错误。

此外，python 语言是**大小写敏感**的，其含义是，相同字母，但大小写不同的语句，python 认为是不同的。如将程序（2-1）中 print 改为 Print，则程序运行也不会成功，python 解释器也会报告语法错误。

2、打印多行字符串。

程序(2-2)

```
print('你好，世界！')
print('你好, world!')
```

[1] 函数的具体概念及内容将在后续章节详细介绍。

```
print('4078')
print("It's him!")
print('She said:"You fool!"')
```

执行以上代码，程序运行结果预期如下：

输出(2-2)

```
你好，世界！
你好, world!
4078
It's him!
She said:"You fool!"
```

程序(2-2)中：

- ✓ 每一行代码是从上到下依次执行的。
- ✓ 每一行代码都是从头开始的，即代码最前面没有空格。
- ✓ 为打印输出单引号（第 4 行，这个单引号作为输出打印的字符而不是定义字符串边界的符号），可以用双引号包含。
- ✓ 为打印输出双引号（第 5 行），可以用单引号包含。

虽然可以分别打印出单引号或者双引号，但是用程序（2-2）的方法不能同时打印出单引号与双引号，有兴趣的读者可以尝试一下，查看出错信息，分析错误原因。

3、同时打印单双引号。

程序(2-3)

```
print('''三引号是"特殊"的引号，三引号是'特殊'的引号''')
print('She said: "It\'s him!"')
print("She said: \"It's him!\"")
```

执行以上代码，程序运行结果预期如下：

输出(2-3)

```
三引号是"特殊"的引号，三引号是'特殊'的引号
She said: "It's him!"
She said: "It's him!"
```

程序(2-3)中：

- ✓ 一对**三引号**（即三个连续的单引号）括起来的任意字符序列也是 python 字符串。用三引号定义的字符串不但可以同时表示包含单引号与双引号的字符串，还可以表示跨行字符串。
- ✓ 在字符串中，以\开头，在后跟一个或几个字符，在 python 中称为**转义字符**。

转义字符具有特定的含义，不同于原有的字符。例如，\'表示为作为字符串的单引号，而非语句中作为定界符的单引号；\"表示为作为字符串的双引号，而不是 python 语句中作为定界符的双引号。比较常用的转义字符还有：\n，为换行符、\\，为反斜杠。python 常用转义字符及其含义见本章附表 2-1。

4、打印多行字符串。

程序(2-4)

```
print('''第一行
第二行
第三行
第四行
''')
print('---')
print('第一行\n 第二行\n 第三行')
```

执行以上代码，程序运行结果预期如下：

输出(2-4)

```
第一行
第二行
第三行
第四行
---
第一行
第二行
第三行
```

程序(2-4)中：

- ✓ 在第一个 print()函数中，使用三引号可以利用一条 print 语句直接输出跨行/多行字符串
- ✓ 最后一行的 print()函数，能够打印多行是利用了转义字符\n，该转义字符的含义是换行。

在不利用转义字符的前提下，采用单引号的字符串或双引号的字符串均无法输出多行字符串，有兴趣的读者可以自行尝试。

5、打印数字。

程序(2-5)

```
print(4078)
print(3.14159)
print(-2.718)
print(1+2j)
```

执行以上代码，程序运行结果预期如下：

输出(2-5)

```
4078
3.14159
-2.718
(1+2j)
```

程序(2-5)中：

- ✓ python 根据数据的不同性质与特点将其分类并进行了定义，统称为**数据类型**。其中，整

数数据类型定义为**整型**(int)，如第 1 行打印的 4078 即为整型。

- ✓ 第 2 行与第 3 行打印的是小数，python 定义其数据类型为**浮点型**(float)。
- ✓ 第 4 行是一个复数，python 定义其数据类型为**复数型**(complex)

整型、浮点型、复数型均被用来表示数值，因此统称为**数值型**数据，如表 2-1 所示。各种类型的数据可以直接作为参数利用 print()函数输出。也可以使用 python 中内置的 type()函数来查看各种数据的类型。

表 2-1 数值型数据

整型(int)	浮点型(float)	复数类型(complex)
10	1.2	5i
-5	3.1415	4+7j
100	-112.3	3-4j
…	…	…

6、查看并打印数据的类型。

程序(2-6)

```
print(type(985))
print(type(3.14))
print(type(3+4j))
print(type('985'))
```

执行以上代码，程序运行结果预期如下：

输出(2-6)

```
<class 'int'>
<class 'float'>
<class 'complex'>
<class 'str'>
```

程序(2-6)中：

- ✓ type()函数接收一个参数，得到该参数的数据类型。
- ✓ class 为类，会在后续章节中介绍。<class 'int'>可暂时理解为：数据类型是整型。

7、进行算术运算并打印结果。

程序(2-7)

```
print(985+211)
print(10+10/5)
print(2.718*(6*5-2))
print(3*(3*(5-2)))
print(3.14*5**2)
print(9%2)
print(9//2)
```

执行以上代码，程序运行结果预期如下：

输出(2-7)

```
1196
12.0
76.104
27
78.5
1
4
```

程序(2-7)中：

- ✓ +、-、*、/、**、%、//为**算术运算符**，分别对应：加、减、乘、除、乘方、求余数和整除运算。算术运算符的含义见下表。
- ✓ 运算符之外，被运算的对象，称为**操作数**。
- ✓ 将操作数利用运算符连接起来组成的式子，称为**表达式**。操作数经运算符运算后得到的值，称为**表达式的值**。每个表达式均有确定的唯一的值，因此可以作为 print()函数的参数直接打印输出。其中，由算术运算符与操作数组成的表达式，称为**算术表达式**或**数值表达式**。只有数值型的数据才能构成算术表达式，进行算术运算。
- ✓ python 表达式中仅使用小括号，而中括号和大括号均由小括号代替。

表 2-2 算术运算符的含义

运算符示例	操作	含义
x + y	加法运算	x 和 y 的和
x - y	减法运算	x 和 y 的差
x * y	乘法运算	x 和 y 的乘积
x / y	除法运算	x 和 y 的商
x // y	整除运算	x 和 y 的商向下取整
x % y	求余运算	x 对 y 求余
x ** y	乘方运算	x 的 y 次方

表 2-2 中：

- ✓ 除法运算，其运算结果是一个浮点数。
- ✓ 整除运算，是做除法运算后返回商的整数部分的值，该值根据操作数类型的不同，可能返回整型或浮点型。此外，当除法运算结果为负数时，整除运算的结果为商的整数部分的值减一。
- ✓ 各种算术运算符的使用与数学中的规定基本一致。各种算术运算符具有各自的优先级，其优先级顺序为：乘方>乘、除、求余、整除>加、减。运算优先级高的运算符先进行运算，运算优先级相同时，运算会从左到右依次进行。

表达式中若由相同类型的数据组成，则运算结果的类型默认不变。同时，Python 允许在同一个表达式中不同类型的数据参与运算,此时运算后的结果,会自动进行**数据类型转换**。如整型与浮点型数据进行运算，则运算结果默认为浮点型。python 可以将一种数据**类型强制转换**为另一种数据类型，这就需要使用内置的类型转换函数。

8、打印强制转换的数字。

程序(2-8)

```
print(int(3.14))
print(float(3))
print(complex(3))
print(complex(3,5))
```

执行以上代码，程序运行结果预期如下：

输出(2-8)

```
3
3.0
3+0j
3+5j
```

程序(2-8)中：

- ✓ int()、float()及 complex()均为类型转换函数，功能分别为将函数括号内给定的参数转换为整型、浮点型及复数型。
- ✓ 对 complex()函数，如果仅给定一个参数，则该参数作为实部，虚部为零。如果给定两个参数，两个参数之间用逗号分隔，依次作为一个复数的实部和虚部。

除了不同数值型之间可以进行强制转换外，字符串类型与数值型之间也可以进行强制类型转换。

9、字符串与的数字相互转换的打印。

程序(2-9)

```
print(int('4078'))
print(type(int('4078')))
print(str(4078))
print(type(str(4078)))
print(float('3.14'))
print(type(float('3.14')))
print(str(3.14))
print(type(str(3.14)))
```

执行以上代码，程序运行结果预期如下：

输出(2-9)

```
4078
<class 'int'>
4078
<class 'str'>
3.14
<class 'float'>
3.14
<class 'str'>
```

(2-9)中：

- ✓ 第 1 行 int('4078')将字符串'4078'强制转换为整型数 4078，然后 type()函数得到其类型为 int，并由 print()函数打印到显示器。虽然第 1 行与第 3 行，第 5 与第 7 行打印的结果相同，但实际上打印的是类型不同的数据。

数值型的数据可以直接进行算术运算，字符串类型的数据不能直接进行算术运算，必须进行转换。数值型数据均可以转换为字符串，但只有类似数值形式的字符串才能够转换为数值型数据。以上两点，感兴趣的读者可以自行尝试。

10、打印关系表达式的值。

程序(2-10)

```
print(1 < 2)
prnt(3 > 3.14)
print(1.0 == 1)
print(complex(1) == 1.0)
print(3.14 == '3.14')
print(2.2-1 == 1.2)
```

执行以上代码，程序运行结果预期如下：

输出(2-10)

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

程序(2-10)中：

- ✓ 符号 < 、> 和 == 都是 Python 中内置的**关系运算符**。Python 中一共有 8 个关系运算符，分别是<、<=、>、>=、==、!=、is、is not。他们的运算优先级相同，各运算符的具体含义见表 2-3。
- ✓ 由关系运算符和操作数组成的表达式称为**关系表达式**。
- ✓ 每一个关系表达式有且仅有唯一确定的值，这个值是一个**逻辑值**（或**布尔值**）：True（真）或 False（假）。True 与 False 均属于 python 中**布尔类型**。**布尔类型**的数据仅可能取这两个值。
- ✓ 除了数值类型目标之间，其他的数据类型之间==运算的值都是 False。
- ✓ 最后一行打印出结果是 False，读者理应意外。这是由于计算机中各种数据均由二进制表示，因此 python 中能够表示浮点数的有效数字是有限的，在实际运算中会带来精度损失或者说误差。实际上，2.2-1 这个算术表达式的值[2]，在 python 中不精确等于 1.2，而是等于一个非常接近 1.2 的浮点数。因此，在实际编程中，应该尽量杜绝使用类似的关系比较，而是应该使用可表达类似关系的表达式如：2.2-1-1.2 < 0.00001 来替代。

[2] 可引入 decimal 模块进行精确的计算。

表 2-3 关系运算符

关系运算符	含义
x < y	x 严格小于 y
x <= y	x 小于或等于 y
x > y	x 严格大于 y
x >= y	x 大于或等于 y
x == y	x 等于 y
x != y	x 不等于 y

11、打印基本逻辑表达式的值。

程序(2-11)

```
print(True and True)
print(True and False)
print(True or False)
print(False or False)
print(not True)
```

执行以上代码，程序运行结果预期如下：

输出(2-11)

```
True
False
True
False
False
```

程序(2-11)中：

- ✓ and、or 和 not 是 Python 内置的三个**逻辑运算符**或称**布尔运算符**，分别表示与、或、非。布尔运算符在使用时与操作数之间须以空格符分隔。各个逻辑运算符的具体含义见表 2-4，
- ✓ 用逻辑运算符将关系表达式或逻辑量连接起来的有意义的式子称为**逻辑表达式**。
- ✓ 逻辑表达式的值是一个布尔值。逻辑运算符的真值表见表 2.5。
- ✓ 逻辑运算符的运算优先级从高到低依次是 not、and、or。逻辑运算符的运算优先级都低于关系运算符。

表 2-4 逻辑运算符

逻辑运算符	结果
x or y	如果 x 的值为假，那么结果就是 y 的值，否则结果是 x 的值
x and y	如果 x 的值为假，那么结果就是 x 的值，否则结果是 y 的值
not x	如果 x 的值为假，那么结果为真，否则结果为假

表 2-4 中：

- ✓ or 运算是一个短路操作，即如果 x 的值是假，它只判断 y 的值。
- ✓ and 运算也是一个短路操作，即如果 x 的值为真，它只判断 y 的值。

表 2-5 真值表

x	y	x or y	x and y	not x
假	假	假	假	真
假	真	真	假	真
真	假	真	假	假
真	真	真	真	假

12、打印各种逻辑表达式的值。

程序(2-12)

```
print(1 > 2 and 3.14 > 3)
print(1 < 2 or 3.14 != 3)
print(not 'Python' == 'python')
```

执行以上代码，程序运行结果预期如下：

输出(2-12)

```
False
True
True
```

程序(2-12)中：

- ✓ 可由关系运算符与逻辑运算符联合组成各种逻辑表达式。
- ✓ 逻辑运算符的优先级低于关系运算符，所以 not a == b 可以理解为 not (a == b)。此外，a == not b 在语法上是错误的，错误原因请读者自行分析思考。
- ✓ 实际编程中，一般建议以显式的小括号来直接确定运算的优先顺序。

13、打印多个参数。

程序(2-13)

```
print('a','b','c')
print('a=',3.14)
```

执行以上代码，程序运行结果预期如下：

输出(2-13)

```
a b
a= 3.14
```

程序(2-13)中：

- ✓ 打印多个参数的最简单方法就是以逗号隔开各个参数。
- ✓ 打印时，各个参数之间默认由一个空格符隔开。

二、拓展与总结

1、Python 基本数据类型

Python 的基本数据类型可分为数值类型、字符串类型、布尔类型。其中数值类型分为整型 (int)，浮点型 (float)，复数类型 (complex)。

2、运算符及优先级

Python 的运算符十分丰富，包括：算术运算符、关系运算符/比较运算符、逻辑运算符、赋值运算符、成员运算符、身份运算符、位运算符等，可组成各种表达式进行相应的运算。本章仅介绍了算术运算符、关系运算符及逻辑运算符。各种运算符的优先级见下表。

表 2-6 运算符及优先级

运算符	描述
**	指数 (最高优先级)
*、/、%、//	乘、除、取模和取整除
+、-	加、减
>>、<<	左移、右移
&	位与
^、|	位运算符
<=、<、>、>= 、== 、!=	比较运算符
= %= /= //= -= += *= **=	赋值运算符
Is、 is not	身份运算符
In、 not in	成员运算符
not 、or、 and	逻辑运算符

3、数据在内存中的存储

程序需要被装入计算机**内存**中才能够运行。程序执行时，其中的各种数据当然也就存储在内存中。内存是以**字节**为存储单位的一片连续的存储空间，为便于访问，计算机系统给每个字节单元一个唯一的从 0 开始的编号，第一个字节的单元编号为 0，后面的各个单元按顺序连续编号，这些编号就被称为**内存单元的地址**。实际上数据是存储在各个内存单元中，这样，通过内存单元的地址，就可以找到数据。

常量(constant)是在程序运行中，不能被改变值的数据对象。本章各个程序代码中的数据对象均为常量。常量的数据类型按其值的表示形式区分，例如：78 为整型常量、3.14 为浮点型常量、'你好'为字符串常量等。

程序(2-14)

```
print(3.14)
print(256)
print(id(3.14))
print(id(4078))
```

执行以上代码，程序运行结果预期如下：

输出(2-14)

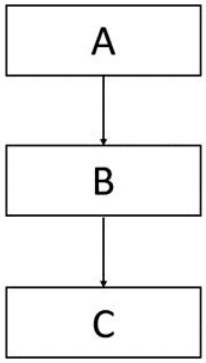
```
3.14
256
1527021192608
1527021500784
```

程序(2-14)中：

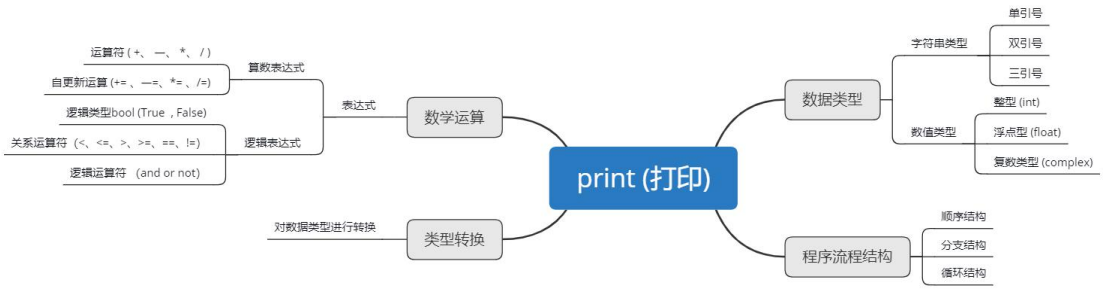
- ✓ id()函数接受一个数据对象参数(本例中均为常量)，返回该数据对象的编号，该编号与该数据对象的内存地址对应。不同数据对象的内存地址不同，因此编号也不同。也即是说，每一个编号对应唯一的数据对象。
- ✓ 第 3,4 行的输出可能与实际输出不尽相同

4、顺序结构

程序基本结构有三种：顺序结构、分支结构和循环结构。本章所有程序语句均为从上到下一条一条按照顺序执行，这样的程序结构被称为**顺序结构**。其流程图如下图所示。



三、本章思维导图



四、附表

2-1 python 常用转义字符及含义

符号	描述	符号	描述
\\	反斜线	\v	纵向制表符
\’	单引号	\r	回车符
\”	双引号	\f	换页符
\a	发出系统响铃声	\o	八进制数代表的字符
\b	退格符	\x	十六进制代表的字符
\n	换行符	\000	终止符，其后的字符串全部忽略
\t	横向制表符		