An\_Introduction\_Python\_Computer\_Programming\_c02

p23

Yue Zhang

An Introduction to Python

and Computer Programming

Chapter 2

Using Python as a Calculator

One of the most important tasks that a computer performs is mathematical computation. In fact, the computation of mathematical functions had been the driving motivation for the invention of ‘computing machines’ by pioneer researchers. As other programming languages do, Python provides a direct interface to this fundamental functionality of modern computers. Naturally, an introduction of Python could start by showing how it can be used as a tool for simple mathematical calculations.

—----------------------------------------------

**Una de las tareas más importantes que realiza una computadora es el cálculo matemático**. De hecho, el cálculo de funciones matemáticas había sido la motivación principal para la invención de las "máquinas de computación" por parte de investigadores pioneros. Al igual que otros lenguajes de programación, Python proporciona una interfaz directa a esta funcionalidad fundamental de las computadoras modernas. Naturalmente, una introducción a Python podría comenzar mostrando cómo se puede usar como herramienta para cálculos matemáticos simples.

—----------------------------------------------

2.1 Using Python as a Calculator

The easiest way to perform mathematics calculation using Python is to use IDLE, the interactive development environment of Python, which can be used as a fancy calculator. To begin with the simplest mathematical functions, including integral addition, subtraction, multiplication and division, can be performed in IDLE using the following mathematical expressions 1 :

—---------------------------------------------------

**La forma más sencilla de realizar cálculos matemáticos con Python es utilizar IDLE**, el entorno de desarrollo interactivo de Python, que se puede utilizar como una calculadora elegante. Para comenzar, las funciones matemáticas más simples, incluidas la suma, resta, multiplicación y división integrales, se pueden realizar en IDLE utilizando las siguientes expresiones matemáticas

—---------------------------------------------------

> > > 3+5

# a d d i t i o n

8

> > > 3 -2

# s u b t r a c t i o n

1

> > > 6\*7

# m u l t i p l i c a t i o n

42

> > > 8/4

# d i v i s i o n

2

As can be seen from the examples above, a simple Python expression is similar to a mathematical expression. It consists of some numbers, connected by a mathematical operator. In programming terminology, number constants (e.g. 3, 5, 2) are called literals. An operator (e.g. +, −, ∗) indicates the mathematical function between its operands, and hence the value of the expression (e.g. 3 + 5). The process of deriving the value of an expression is called the evaluation of the expression. When a mathematical expression is entered, IDLE automatically evaluates it and displays its value in the next line.

—-------------------------------------------

Como se puede ver en los ejemplos anteriores, una **expresión simple de Python** es similar a una expresión matemática. Consiste en algunos números, conectados por un operador matemático. En la terminología de programación, **las constantes numéricas (por ejemplo, 3, 5, 2) se denominan literales**. **Un operador (por ejemplo, +, −, ∗) indica la función matemática entre sus operandos** y, por lo tanto, el valor de la expresión (por ejemplo, 3 + 5). El proceso de derivar el valor de una expresión se denomina evaluación de la expresión. Cuando se ingresa una expresión matemática, IDLE la evalúa automáticamente y muestra su valor en la línea siguiente.

—-------------------------------------------

Since expressions themselves represent values, they can be used as operands in

longer composite expressions.

> > > 3+2 -5+1

1

In the example above, the expression 3+2 is evaluated first. Its value 5 is combined with the next literal 5 by the − operator, resulting in the value 0 of the composite expression 3 + 2 − 5. This value is in turn combined with the last literal 1 by the + operator, ending up with the value 1 for the whole expression.

—-----------------------------------------

**En el ejemplo anterior, la expresión 3+2 se evalúa primero. Su valor 5 se combina con el siguiente literal 5 mediante el operador −, dando como resultado el valor 0 de la expresión compuesta 3 + 2 − 5. Este valor a su vez se combina con el último literal 1 mediante el operador +, finalizando con el valor 1 para toda la expresión.**

—-----------------------------------------

In the example, operators are applied from left to right, because + and − have the same priority. In an expression that contains more than one operators, not necessarily all operators have the same priority. For example,

—----------------------------------

**En el ejemplo, los operadores se aplican de izquierda a derecha porque + y − tienen la misma prioridad. En una expresión que contiene más de un operador, no necesariamente todos los operadores tienen la misma prioridad. Por ejemplo,**

—----------------------------------

> > > 3+2\*5 -4

9

The expression above evaluates to 9, because the multiplication (∗) operator has a higher priority compared with the + and − operators. The order in which operators are applied is called operator precedence, and mathematical operators in Python follow the natural precedence by the mathematical function. For example, multiplication (∗) and division (/) have higher priorities than addition (+) and subtraction (−). An operator that has higher priority than multiplication is the power operator (∗∗).

—----------------------------------

La expresión anterior se evalúa como 9, porque el operador de multiplicación **(∗) tiene una prioridad más alta en comparación con los operadores + y −**. El orden en el que se aplican los operadores se denomina **precedencia** de operadores, y los operadores matemáticos en Python siguen la precedencia natural de la función matemática. Por ejemplo, la multiplicación (∗) y la división (/) tienen mayor prioridad que la suma (+) y la resta (−). Un operador que tiene mayor prioridad que la multiplicación es el operador potencia (∗∗).

—----------------------------------

> > > 5 \*\* 2

# p o w e r

25

In general, a ∗∗ b denotes the value of a to the power of b.

Similar to mathematical equations, Python allows the use of brackets (i.e.

‘(’ and ‘)’) to manually specify the order of evaluation. For example,

> > > ( 3 + 2 ) \*(5 -4)

# b r a c k e t s

5

The value of the expression above is 5 because the bracketed expressions 3 + 2

and 5 − 4 are evaluated first, before the ∗ operator is applied.

In the examples above, an operator connects two literal operands, and hence they are called binary operators. An operator can also be unary, taking only a single operand. An example unary operator is −, which takes a single operand and negates the number.

—-------------------------------------

En los ejemplos anteriores, un operador conecta dos operandos literales y, por lo tanto, se denominan **operadores binarios**. Un operador también puede ser unario, tomando solo un operando. Un ejemplo de operador unario es −, que toma un solo operando y niega el número.

—-------------------------------------

> > > -(5\*1)

# n e g a t i o n

-5

There is also a ternary operator in Python, which takes three operands. It will be introduced in a later chapter.

—-------------------------------------

**También hay un operador ternario en Python,** que toma tres operandos. Se introducirá en un capítulo posterior.

—-------------------------------------

Until this point, all the mathematical expressions have been integral, with the values of literal operands and expressions being integers. Take the division operator (/) for example,

—--------------------------------------

**Hasta este punto, todas las expresiones matemáticas han sido integrales, siendo los valores de los operandos literales y las expresiones números enteros. Tome el operador de división (/) por ejemplo,**

—--------------------------------------

> > 5/2

# d i v i s i o n with i n t e g e r o p e r a n d s

2

The result of the integral division operation is the quotient 2, with the fractional part 1 discarded. To find the remainder of integer division, the modulo operator (%) can be used.

—---------------------------------

El resultado de la operación de división integral es el cociente 2, descartando la parte fraccionaria 1. Para encontrar el resto de la división de enteros, se puede usar el operador de **módulo (%).**

—--------------------------------

> > > 5%2

# m o d u l o

1

2.1.1 Floating Point Expressions

So for all the expressions in this chapter are integer expressions, of which all the operands and the value are integers. However, for the expressions 5/2, sometimes the real number 2.5 is a more appropriate value. In computer science, real number are typically called floating point number. To perform floating point arithmetics, at least one floating point number must be put in the expression, which results in a floating point expression. For example,

—--------------------------------

Así pues, **todas las expresiones de este capítulo son expresiones enteras, de las cuales todos los operandos y el valor son enteros**. Sin embargo, para las expresiones 5/2, a veces el número real 2,5 es un valor más apropiado. **En informática, los números reales se denominan típicamente números de coma flotante.** Para realizar aritmética de coma flotante, se debe colocar al menos un número de coma flotante en la expresión, lo que da como resultado una expresión de coma flotante. Por ejemplo,

—--------------------------------

> > > 5 . 0 / 2

# f l o a t i n g p o i n t d i v i s i o n

2.5

> > > 5 / 2 . 0

# f l o a t i n g p o i n t d i v i s i o n

2.5

> > > 25 \*\* 0.5

# f l o a t i n g p o i n t p o w e r

5.0

The last example above calculates the positive square root of 25. Regardless of operands, when all the numbers in a Python expression are integers, the expression is an integer expression, and the value of the expression itself is an integer. However, when there is at least one floating point number in an expression, the expression is a floating point expression, and its value is a floating point number. Below are some more examples, which show that +, − and ∗ operators can all be applied to floating point numbers, resulting in floating point numbers.

—------------------------------------------

El último ejemplo anterior calcula la raíz cuadrada positiva de 25. Independientemente de los operandos, cuando todos los números en una expresión de Python son enteros, la expresión es una expresión entera y el valor de la expresión en sí es un entero. Sin embargo, cuando hay al menos un número de punto flotante en una expresión, la expresión es una expresión de punto flotante y su valor es un número de punto flotante. A continuación hay algunos ejemplos más, que muestran que los operadores +, − y ∗ se pueden aplicar a números de coma flotante, lo que da como resultado números de coma flotante.

> > > 3 . 0 + 5 . 1

# f l o a t i n g p o i n t a d d i t i o n

8.1

> > > 1.0 -2.4

# f l o a t i n g p o i n t s u b t r a c t i o n

-1.4

> > >5.5\*0.3

# f l o a t i n g p o i n t m u l t i p l i c a t i o n

1.65

The observation above leads to an important fact about Python: things have types. Literals have types. The literal 3 indicates an integer, and the literal 3.0 indicates a floating point number. Expressions have types, and their types are the types of their values. The type of a literal or expression can be examined by using the following commands:

—------------------------------------------

La observación anterior lleva a un hecho importante sobre Python: **las cosas tienen tipos**. Los literales tienen tipos. El literal 3 indica un número entero y el literal 3.0 indica un número de coma flotante. Las expresiones tienen tipos, y **sus tipos son los tipos de sus valores**. **El tipo de un literal o expresión se puede examinar mediante los siguientes comandos:**

> > > **type (3)**

< type ‘ int ’ >

> > > type (3.0)

< type ‘ float ’ >

> > > type (3+5)

< type ‘ int ’ >

>>> type ( 3 + 5 . 0 )

< type ‘ float ’ >

**The command type(x) returns the type of x**. This command is a function call in Python, which type is a built-in function of Python. We call a function with specific arguments in order to obtain a specific return value. In the case above, calling the function type with the argument 3 results in the ‘integer type’ return value.

Function calls are also expressions, which are written in a form similar to mathematica function, with a function name followed by a comma-separated list of arguments enclosed in a pair of brackets. The value of a function call expression is the

return value of the function. In the above example, type is the name of a function,

which takes a single argument, and returns the type of the input argument. As a result,

the function call type(3.0) evaluates to the ‘float type’ value.

Intuitively the return value of a function call is decided by both the function itself

and the arguments of the call. To illustrate this, consider two more functions. **The int**

**function takes one argument and converts it into an integer, while the float function**

**takes one argument and converts it into a floating point number.**

> > > f l o a t (3)

3.0

> > > int ( 3 . 0 )

3

> > > f l o a t (3) /2

1.5

>>> 3\* float (3 -2\*5+4) \*\*2

27.0

As can be seen from the examples above, when the function is type, the return values are different when the input argument is 3 and when the input argument is 3.0. On the other hand, when the input argument is 3, the return value of the function type differs from that of the function float. This shows that both the functions and the arguments determine the return value.

Como se puede ver en los ejemplos anteriores, cuando la función es de tipo, los valores devueltos son diferentes cuando el argumento de entrada es 3 y cuando el argumento de entrada es 3.0. Por otro lado, cuando el argumento de entrada es 3, el valor de retorno del tipo de función difiere del de la función float. Esto muestra que tanto las funciones como los argumentos determinan el valor de retorno.

The last two examples above is a composite expression, in which the function call float(3) is evaluated first, before the resulting value 3.0 is combined with the literal 2 by the operator /. Function calls have higher priorities than mathematical operators in operator precedence.

—------------------------------------

Los dos últimos ejemplos anteriores son una expresión compuesta, en la que la llamada de función float(3) se evalúa primero, antes de que el operador / combine el valor resultante 3.0 con el literal 2. **Las llamadas a funciones tienen prioridades más altas que los operadores matemáticos en la precedencia de operadores.**

The int function converts a floating point number into an integer by discarding all

the digits after the floating point. For example,

> > > int ( 3 . 0 )

3

> > > int ( 3 . 1 )

3

> > > int ( 3 . 9 )

3

In the last example, the return value of int(3.9) is 3, even though 3.9 is numerically

closer to the integer 4. For floating-point conversion by rounding up an integer, the

round function can be used

> > > r o u n d ( 3 . 3 )

3

> > > r o u n d ( 3 . 9 )

4

**The round function can round up a number not only to the decimal point, but also to**

**a specific number of digits after the decimal point.** In the latter case, two arguments

must be given to the function all, with the second input argument indicating the

number of digits to keep after the decimal point. The following examples illustrate

this use of the round function with more than one input arguments. Take note of the

comma that separates two input arguments.

> > > round (3.333 , 1)

3.3

> > > round (3.333 , 2)

3.33

A floating point operator that results in an integer value is the integer division

**operator (//)**, which discards any fractional part in the division.

> > > 3 . 0 / / 2

1

> > > 3 . 5 / / 2

1

Correspondingly, the modulo operator can also be applied to floating point divi-

sion.

> > > 3 . 5 **%** 2

1.5

Another useful function is **abs**, which takes one numerical argument and returns

its absolute value.

> > > abs (1)

1

> > > abs ( 1 . 0 )

1.0

> > > abs ( -5)

5

One final note on floating point numbers is that **their literals can be expressed by**

**a scientific notation**. For example,

> > > 3 e1

30.0

> > > **3 e -1**

0.3

> > > 3 E2

3 0 0 . 0

**The notations xey and xEy have the same meaning**. They indicate the value of

x × 10y .

2.1.2 Identifiers, Variables and Assignment

The set of arithmetic expressions introduced above allows simple calculations using IDLE. For example, suppose that the annual interest rate of a savings account is 4 %. To calculate the amount of money in the account after three years, with an initial sum of 3, 000 dollars is put into the account, the following expression can be used.

El conjunto de expresiones aritméticas presentado anteriormente permite cálculos simples usando IDLE. Por ejemplo, suponga que la tasa de interés anual de una cuenta de ahorros es del 4 %. Para calcular la cantidad de dinero en la cuenta después de tres años, con una suma inicial de 3.000 dólares en la cuenta, se puede utilizar la siguiente expresión.

> > > 3 0 0 0 \* 1 . 0 4 \* \* 3

3 3 7 4 . 5 9 2

One side note is that brackets can be used to explicitly mark the intended operator precedence, even if they are redundant. In the case above, 3000 ∗ 1.04 ∗ ∗3 canbe written as 3000 ∗ (1.04 ∗ ∗3) to make the operator precedence more obvious. In general, being more explicit can often make the code easier to understand and less likely to contain errors, especially when there are potential ambiguities (e.g.

non-intuitive or infrequently used operator precedence). For a second example, suppose that the area of a square is 10 m2 . The length of each edge can be calculated by:

Una nota al margen es que los corchetes se pueden usar para marcar explícitamente la precedencia prevista del operador, incluso si son redundantes. **En el caso anterior, 3000 ∗ 1.04 ∗ ∗3 se puede escribir como 3000 ∗ (1.04 ∗ ∗3) para que la precedencia del operador sea más obvia.** En general, ser más explícito a menudo puede hacer que el código sea más fácil de entender y menos probable que contenga errores, especialmente cuando existen posibles ambigüedades (p. precedencia de operadores poco intuitiva o de uso poco frecuente). Para un segundo ejemplo, suponga que el área de un cuadrado es de 10 m2. La longitud de cada borde se puede calcular mediante:

> > > 1 0 \* \* 0 . 5

3 . 1 6 2 2 7 7 6 6 0 1 6 8 3 7 9 5

**The result can be rounded up to the second decimal place.**

>>> round (10\*\*0.5 , 2)

3.16

For notational convenience and to make programs easier to maintain, Python allows names to be given to mathematical values. An equivalent way of calculating the edge length is:

**Por conveniencia notacional y para facilitar el mantenimiento de los programas, Python permite dar nombres a los valores matemáticos**. Una forma equivalente de calcular la longitud del borde es:

> > > a =10

> > > w = a \* \* 0 . 5

> > > r o u n d ( w , 2)

3.16

In the example above, **a** denotes the area of the square, and **w** denotes its width.

The use of a and w makes it easier to understand the underlying physical meanings

of the values. **a and w are called identifiers in Python**. Each Python identifiers is

bound to a specific value. In the example, a is bound to 10 and w is bound to √10.

Identifiers can be bound to new value:

> > > x =1

> > > x

1

> > > x =2

> > > x

2

**In the example, the value of x is first 1, and then 2. Because identifiers can change**

**their values, they are also called variables.**

The = sign in the above example is not an operator, and hence the commands

a = 10 and w = round(a ∗ ∗0.5) are not expressions. They bare no values. Instead **the = sign denotes an assignment statement, which binds an identifier to a value.**

**Here a statement is a command to be executed by Python, and statements are the**

**basic execution units in Python.** **There are different types of statements**, as will be

introduced in this book. In an assignment statement, the identifier to which a value is

assigned must be on the left hand side of =, and the value to assign to the identifier,

which can be any expression, should be on the right hand side of =. Python gives a

name to a value by binding the value to an identifier.

**An intuitive difference between identifiers and literals is that the former are names**

**while the latter are values.** **Formally, an identifier must start with a letter or underscore**

**(\_), and contain a sequence of letters, numbers and underscores.** For example, area,

a, a0, area\_of\_square and \_a are all valid identifiers, but 0a, area of square or a1!

are not valid identifiers. **An additional rule is that identifiers must not be keywords**

**in Python, which are a list of reserved words. There are 31 keywords in total**, which

are listed in Table 2.1. Each keyword can be associated with one or more statements,

which will be introduced in the subsequent chapters.

For another example problem, suppose that a ball is tossed up on the edge of a cliff with an initial velocity v0, and that the initial altitude of the ball is 0 m. The question is to find the vertical position of the ball at a certain number of seconds t after the toss. If the initial velocity of 5 m/s and the time is 0.1 s, the altitude can be calculated by:

Para otro problema de ejemplo, suponga que se lanza una pelota al borde de un acantilado con una velocidad inicial v0 y que la altura inicial de la pelota es 0 m. La cuestión es encontrar la posición vertical de la pelota en un cierto número de segundos t después del lanzamiento. Si la velocidad inicial es de 5 m/s y el tiempo es de 0,1 s, la altitud se puede calcular mediante:

> > > v0 =5

> > > g = 9 . 8 1

> > > t =0.1

> > > h = v0 \* t - 0 . 5 \* g \* t \*\*2

> > > r o u n d ( h , 2)

0.45

To further obtain the vertical location of the ball at 1 s, only t and h need to be

modified.

> > > t =1

> > > h = v0 \* t - 0 . 5 \* g \* t \*\*2

> > > r o u n d ( h , 2)

0.09

Note that the value of h must be calculated again after the value of t changes.

This is because an assignment statement binds an identifier to a value, rather than

establishing a mathematical correlation between a set of variables. When h = v0 ∗

t − 0.5 ∗ g ∗ g ∗ t ∗ ∗ 2 is executed, the right hand side of = is first evaluated

according to the current values of v0, g and t, and then the resulting value is bound

to the identifier h. This is different from a mathematical equation, which establishes

factual relations between values. When the value of t changes, the value of h must

be recalculated using h = v0 ∗ t − 0.5 ∗ g ∗ g ∗ t ∗ ∗ 2. For another example,

> > > x =1

> > > x = x +1

> > > x

2

There are three lines of code in this example. The first is an assignment statement,

binding the value 1 to the identifier x. The second is another assignment statement,

which binds the value of x + 1 to the identifier x. When this line is executed, the

right hand side of = is first evaluated, according to the current value of x. The result

is 2. This value is in turn bound to the identifier x, resulting in the new value 2 for

this identifier. The third line is a single expression, of which the value is displayed

by IDLE. Think how absurd it would be if the second line of code is treated as a

mathematical equation rather than an assignment statement!

An equivalent but perhaps less ‘counter-intuitive’ way of doing x = x + 1 is

x+= 1.

> > > x =1

> > > **x +=1**

> > > x

2

The same applies to x = x − 3, x = x ∗ 6, and other arithmetic operators.

> > > x -=3

> > > x

-1

> > > x \*=6

> > > x

-6

**In general, x<op> =y is equivalent to x=x<op>y, where <op> can be +, −, ∗,**

**/, % etc.** The special assignment statements +=, −=, ∗=, /= and %= can be used

as a concise alternative to a = assignment statement when it incrementally changes

the value of one variable.

Given the fact above, it is not difficult to understand the outputs, if the following

commands are entered into IDLE to find the position of the ball after 3 s in the

previous problem.

( c o n t i n u e d from above )

> > > t =3

> > > r o u n d ( h , 2)

0.09

> > > h = v0 \* t - 0 . 5 \* g \* t \*\*2

> > > r o u n d ( h , 2)

-29.15

**The commands above are executed sequentially and individually.** When t is bound

to the new value 3, h is not affected, and remains 0.09. After the new h assignment is

executed, its value changes to −29.15 according to the new t. Cascaded assignments.

Several assignment statements to the same value can be cascaded into a single line.

**For example, a = 1 and b = 1 can be cascaded into a = b = 1.**

> > > a = b =1

> > > a

1

> > > b

1

2.2 The Underlying Mechanism

Floating point arithmatic can be inaccurate in calculators; the same happens in Python.

**La aritmética de punto flotante puede ser imprecisa en las calculadoras; lo mismo sucede en Python.**

> > > x = 1 . 0 / 7

> > > x + x + x + x + x + x + x

0 . 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 8

> > > 3 . 3 % 2

1 . 2 9 9 9 9 9 9 9 9 9 9 9 9 9 9 8

In both cases, the expression is evaluated to an imprecise number. The main reason is limitation of memory space. A floating point number can contain an infinite amount of information, if there is an infinite number of digits after the decimal point. However, the amount of information that can be processed or stored by a calculator or a computer is finite. As a result, floating point numbers cannot be stored to an arbitrary precision, and floating point operators cannot be infinitely precise. The error that results from the imprecise operations is called rounding off error.

**En ambos casos, la expresión se evalúa a un número impreciso.** La razón principal es la limitación del espacio de memoria. Un número de punto flotante puede contener una cantidad infinita de información, si hay un número infinito de dígitos después del punto decimal. Sin embargo, la cantidad de información que una calculadora o una computadora puede procesar o almacenar es finita. Como resultado, los números de punto flotante no se pueden almacenar con una precisión arbitraria y los operadores de punto flotante no pueden ser infinitamente precisos. El error que resulta de las operaciones imprecisas se llama error de redondeo.

At this point it is useful to know a little about the underlying mechanism of Python, so that deeper understanding can be gained on facts such as rounding off errors, which enables more solid programs to be developed. This section discusses the prepresentation and storage of numbers, the way in which arithmetic operations are carried out by Python, and the underlying mechanisms of identifiers and assignment statements.

En este punto, es útil saber un poco sobre el mecanismo subyacente de Python, de modo que se pueda obtener una comprensión más profunda de hechos como el redondeo de errores, lo que permite desarrollar programas más sólidos. Esta sección analiza la presentación previa y el almacenamiento de números, la forma en que Python lleva a cabo las operaciones aritméticas y los mecanismos subyacentes de los identificadores y las sentencias de asignación.

The basic architecture of a computer is shown in Fig. 2.2 in the previous chapter. On the bottom of the figure, the main hardware components are shown, which include the CPU, memory and devices. Among the three main components, the CPU is the most important; it carries out computer instructions, including arithmetic operations. The typical way in which arithmetic operations are executed is: the CPU takes the operands from the memory, evaluates the result by applying the operator on them, and then stores the result back into the memory. The memory can be regarded as a long array of information storage units. Each unit is capable of storing a certain amount of information, and the index of its location in the long array if also referred to as its memory address. Devices are the channel through which computers are connected to the physical world. Keyboards, mouses, displays, speakers, microphones are a few commonly-used devices. An important device is the hard disk, on which the OS defines a file system for external storage of information.

La arquitectura básica de una computadora se muestra en la figura 2.2 del capítulo anterior. En la parte inferior de la figura, se muestran los principales componentes de hardware, que incluyen la CPU, la memoria y los dispositivos. Entre los tres componentes principales, la CPU es el más importante; lleva a cabo las instrucciones de la computadora, incluidas las operaciones aritméticas. La forma típica en que se ejecutan las operaciones aritméticas es: **la CPU toma los operandos de la memoria, evalúa el resultado aplicando el operador sobre ellos y luego almacena el resultado nuevamente en la memoria.** La memoria puede considerarse como una gran variedad de unidades de almacenamiento de información. Cada unidad es capaz de almacenar una cierta cantidad de información, y el índice de su ubicación en la matriz larga también se conoce como su dirección de memoria. Los dispositivos son el canal a través del cual las computadoras se conectan al mundo físico. Teclados, ratones, pantallas, altavoces, micrófonos son algunos de los dispositivos de uso común. Un dispositivo importante es el disco duro, en el que el sistema operativo define un sistema de archivos para el almacenamiento externo de información.

2.2.1 Information

Computers are information-processing machines. The rounding-off error examples show that the basic unit of storage and computation can only accommodate a limited amount of information. But what is information, and how can one store information? A short answer to the first question is that, information is represented in computers as discrete numbers, or integers. It is an abstraction of real physical quantities, such as the pitch of sound, the colour, and the alphabet. Signals from input devices are transformed into discrete numbers before being processed by a computer, and output devices turn discrete values back into physical signals.

**Las computadoras son máquinas de procesamiento de información.** Los ejemplos de error de redondeo muestran que la unidad básica de almacenamiento y computación solo puede acomodar una cantidad limitada de información. Pero, ¿qué es la información y cómo se puede almacenar información? Una respuesta breve a la primera pregunta es que la información se representa en las computadoras como números discretos o enteros. Es una abstracción de cantidades físicas reales, como el tono del sonido, el color y el alfabeto. Las señales de los dispositivos de entrada se transforman en números discretos antes de ser procesadas por una computadora, y los dispositivos de salida vuelven a convertir los valores discretos en señales físicas.

For example, letters typed on a keyboard are mapped into discrete numbers (e.g. ‘A’ → 64) before being stored into the memory. Sound waves received by a microphone are sampled at a certain rate (e.g. 256,000 times a second), and then turned into an array of discrete values. Such type of sound information can be processed (e.g. denoised) or transformed (e.g. enlarged), and than passed to a sound output device and transformed back into sound waves. A black and white display transforms a grid of numbers into a gird of pixels, each number depicting the brightness of a pixel on the display. A robot can move according to input numbers that indicate desired velocity and direction. In all these cases, devices act as a channel between computers and the physical world.

Por ejemplo, las letras escritas en un teclado se asignan a números discretos (por ejemplo, 'A' → 64) antes de almacenarse en la memoria. Las ondas de sonido recibidas por un micrófono se muestrean a una determinada velocidad (por ejemplo, 256 000 veces por segundo) y luego se convierten en una matriz de valores discretos. Este tipo de información de sonido puede procesarse (p. ej., eliminar el ruido) o transformarse (p. ej., ampliarse), y luego pasar a un dispositivo de salida de sonido y volver a transformarse en ondas de sonido. Una pantalla en blanco y negro transforma una cuadrícula de números en una red de píxeles, cada número representa el brillo de un píxel en la pantalla. Un robot puede moverse de acuerdo con los números de entrada que indican la velocidad y la dirección deseadas. En todos estos casos, los dispositivos actúan como un canal entre las computadoras y el mundo físico.

To answer the second question above, the easiest data storage medium that can be found is probably some material that can have two states (e.g. hight-voltage vs. low-voltage, solid vs. liquid, hot vs. cold). A basic storage unit made of such material can store a binary value, denoted as 0 or 1, each representing a distinct state of the material. Larger integers can be stored by using a combination of multiple basic storage units. For example, the combination of two basic storage units can store four distinct values: 00, 01, 10 and 11. An illustration is shown in Fig. 2.1. In general, the total number of possible states by combining n basic storage units is 2n . On the other hand, at any time, the combined units can only have one actual combined state, which can be represented by a unique array of 0s and 1s.

Para responder a la segunda pregunta anterior, el medio de almacenamiento de datos más fácil que se puede encontrar es probablemente algún material que pueda tener dos estados (por ejemplo, alto voltaje frente a bajo voltaje, sólido frente a líquido, caliente frente a frío). Una unidad de almacenamiento básica hecha de dicho material puede almacenar un valor binario, indicado como 0 o 1, cada uno de los cuales representa un estado distinto del material. Los enteros más grandes se pueden almacenar mediante una combinación de varias unidades básicas de almacenamiento. Por ejemplo, la combinación de dos unidades básicas de almacenamiento puede almacenar cuatro valores distintos: 00, 01, 10 y 11. En la Fig. 2.1 se muestra una ilustración. En general, el número total de estados posibles al combinar n unidades básicas de almacenamiento es 2n. Por otro lado, en cualquier momento, las unidades combinadas solo pueden tener un estado combinado real, que puede representarse mediante una matriz única de 0 y 1.

It is natural to associate an array of 0 s and 1s with a discrete number (integer). One

way to number distinct states of N basic storage units s0, s1, . . . s N , si ∈ {0, 1} is to

interpret each distinct state as a non-negative binary number, treating the value of

s N s N −1 . . . s0 as 2N ∗s N +2N −1 ∗s N −1 +. . .+2 0 ∗s0 = ∑N

i=0 si ·2i . For example, the

state 101 corresponds to the integer 1 ∗ 2 2 + 0 ∗ 2 1 + 1 ∗ 2 0 = 1 ∗ 4 + 0 ∗ 2 + 1 ∗ 1 =

5, and 1101 corresponds to the integer 1 ∗ 2 3 + 1 ∗ 2 2 + 0 ∗ 2 1 + 1 ∗ 2 0 =

1 ∗ 8+1 ∗ 4+0 ∗ +1∗1 = 13. In this way, a natural connection is established between

the states of data storage materials and integers, which represent information. In

computer science, each binary-valued digit is called a bit, and a combination of

8 bits is a byte. A byte storage unit can store 2 8 = 256 distinct numbers, which can

be index by the integers 0–255. Most modern computers treat 8 bytes, or 64 bits, as

a word, which serves as the basic units for storage.

As mentioned earlier, a digital number represents abstract information. Numbers,

alphabets, sounds and all other types of information are abstracted and represented by

binary numbers in computers. In information theory, the amount of information is

measured by bits. A 64-bit word can represent 64 bits of information, which translates

to 2 64 distinct integers. The abstract information, however, can be interpreted in

different ways. The aforementioned interpretation of information as non-negative

integer values is just one example, which can be used to represent the brightness of

a pixel on a black and white monitor.

Another example is the 2’s complement interpretation of signed integers, which

uses 64 bits to represent an integer that ranges from −2 63 to 2 63 − 1. In this repre-

sentation, the first bit always indicates the sign: when it is 0, the number is positive;

when it is 1, the number is negative. When the number is positive, the remaining 63

bits indicate the absolute value of the number. Negative numbers are represented in

a slightly more complicated way. Rather than concatenating the sign bit (i.e. 1) with

a 63-bit representation of the absolute value, a negative number is represented by

first inversing its absolute value bit by bit, and then adding 1 to the result. For the

convenience of illustration, take a byte-sized number for example. To find the repre-

sentation of –13, two steps are necessary. First, the absolute value, 13, or 00001101 in

binary form, is inversed bit by bit into 11110010. Then 1 is added to the back of the

number:

1 1 1 1 0 0 1 0

+) 0 0 0 0 0 0 0 1

- - - - - - - - - - -

1 1 1 1 0 0 1 1

The resulting number, 1110011, is the binary representation of –13 in 2’s com-

plement form. Note that the first bit is 1, indicating that it is a negative number. For another example, to represent –16, its absolute value 00010000 is first inversed bit

by bit into 11101111, and then 1 is added to the number,

1 1 1 0 1 1 1 1

+) 0 0 0 0 0 0 0 1

- - - - - - - - - - -

1 1 1 1 0 0 0 0

As a result, –13 and –16 are 11110011 and 11110000, respectively, according to

2’s complement representation. The same process of negative number interpretation

applies to 64-bit words. The advantage of 2’s complement representations is that the

addition operation between numbers can be performed in the same way regardless

of whether negative numbers are involved or not. For example, −13 + 13 can be

performed by

1 1 1 1 0 0 1 1

+) 0 0 0 0 1 1 0 1

- - - - - - - - - - -

1 0 0 0 0 0 0 0 0

With the first bit being discarded (it runs out of the 8-bit boundary, and hence

cannot be recorded by 8-bit physical media), the result is 0, the correct answer.

A third useful interpretation of information is floating point numbers. When

sound is concerned, floating point numbers give a more convenient model of the

pitches in sound wave samples. As discussed earlier, all floating point numbers

cannot be represented using a finite amount of information, and therefore some have

to be truncated when represented using a computer word. A standard approach of

representing floating point numbers in a finite number of bits is to split the total

number of bits into two parts, one representing a base number (also referred to as

the significant) and the other representing the exponent. For example, from a 64-

bit word, 11 bits can be used to denote the exponent (e) and 53 bit the base (b).

This type of representation naturally corresponds to the scientific notation of float

literals in Python, where bEe = b × 10e. Of course, abstract information can also

be interpreted as letters and other quantities in the physical world, which are out of

the scope of this book.

Words are used not only as the basic units of data storage, but also as the basic

units of computation. In digital circuits, of which all modern computers are made,

electric signals are represented by binary values, with a high voltage in a wire

denoting the value 1, and a low voltage denoting 0. Digital chips, such as CPUs,

takes a fixed number of binary signals as input, and have a fixed number of output.

64-bit CPUs perform arithmetic operations on 64-bit operands, yielding 64-bit results

by hardware computation. As a consequence, floating point numbers can contain

only 64 bits of information, and floating point arithmetics have rounding off errors.

In fact, integers are also represented by words on computer hardware, typically in

2’s complement form. However, Python provides a new type, long, which repre-

sents numbers that exceeds the range of 64 bits. Python converts large integers into

the long type automatically, and performs arithmetic operations between long type

numbers implicitly by using a sequence of 64-bit integer arithmetic operations, so

that programmers can use a large integer in Python without noticing the difference

between int and long. Long type numbers can be identified by examining their types

explicitly.

> > > type (111)

< type ‘ int ’ >

>>> type ( 2 \* \* 6 4 )

< type ‘ long ’ >

A long type number can also be specified explicitly using long literals, which are

integer literals with a ‘l’ or ‘L’ added to the end

> > > type (111 L )

< type ‘ long ’ >

In summary, information that a computer stores and processes is ultimately rep-

resented by a finite number of 0s and 1s (i.e. bits), organized in basic unites (e.g.

words). They are interpreted in different ways when turned into specific types.

2.2.2 Python Memory Management

When evaluating an arithmetic expression, Python first constructs an object in the memory for each literal in the expression, and then applies the operators one by one to obtain intermediate and final values. All the intermediate and final values are stored in the memory as objects. Python objects are one of the most important concepts in understanding the underlying mechanism of Python. Integers, floating point numbers and instances of many more types to be introduced in this book, are maintained in the memory as Python objects.

Al evaluar una expresión aritmética, Python primero construye un objeto en la memoria para cada literal de la expresión y luego aplica los operadores uno por uno para obtener valores intermedios y finales. Todos los valores intermedios y finales se almacenan en la memoria como objetos. Los objetos de Python son uno de los conceptos más importantes para comprender el mecanismo subyacente de Python. **Los números enteros, los números de coma flotante y las instancias de muchos más tipos que se presentarán en este libro se mantienen en la memoria como objetos de Python.**

Fig. 2.2 illustrates how the memory changes when some arithmetic expressions are evaluated. After IDLE starts, the memory contains some default objects, which are not under concern at this stage, and therefore not shown in the figure. When the expression 3.0 is evaluated, a new float object is constructed in the memory. Python always creates a new object when it evaluates a literal. When the expression 3 + 5 ∗ 2 is evaluated, the integer constants 3, 5 and 2 are constructed in the memory, before the operators ∗ and + are executed in their precedence. When ∗ is executed, Python passes the values of the objects 5 and 2 to the CPU, together with the ∗ operator, and stores the result 10 as a new integer object in the memory. When + is executed, Python invokes the CPU addition operation with the values of the objects 3 and 10, storing the result 13 as a new object.

La figura 2.2 ilustra cómo cambia la memoria cuando se evalúan algunas expresiones aritméticas. Después de que se inicia IDLE, la memoria contiene algunos objetos predeterminados, que no son motivo de preocupación en esta etapa y, por lo tanto, no se muestran en la figura. Cuando se evalúa la expresión 3.0, se construye un nuevo objeto flotante en la memoria. Python siempre crea un nuevo objeto cuando evalúa un literal. Cuando se evalúa la expresión 3 + 5 ∗ 2, se construyen en memoria las constantes enteras 3, 5 y 2, antes de que se ejecuten los operadores ∗ y + en su precedencia. Cuando se ejecuta ∗, Python pasa los valores de los objetos 5 y 2 a la CPU, junto con el operador ∗, y almacena el resultado 10 como un nuevo objeto entero en la memoria. Cuando se ejecuta +, Python invoca la operación de suma de CPU con los valores de los objetos 3 y 10, almacenando el resultado 13 como un nuevo objeto.

Identifiers are names of objects in memory, used by Python to access the corresponding objects. Python associates identifiers to their corresponding values, or objects, by using a binding table, which is a lookup table.

**Los identificadores son nombres de objetos en la memoria**, utilizados por Python para acceder a los objetos correspondientes. Python asocia identificadores a sus valores u objetos correspondientes mediante el uso de una tabla de vinculación, que es una tabla de búsqueda.

Figure 2.3 shows an example of the binding table, and how it changes as Python executes assignment statements. After IDLE starts, some default entries are put into the binding table, which are ignored in this figure. When x = 6 is executed, the expression 6 is first evaluated, resulting in the integer object 6 in the memory. Then Python adds an entry in the binding table, associating the name x with the object 6.

La figura 2.3 muestra un ejemplo de la tabla de vinculación y cómo cambia a medida que Python ejecuta sentencias de asignación. Después de que se inicia IDLE, algunas entradas predeterminadas se colocan en la tabla de vinculación, que se ignoran en esta figura. Cuando se ejecuta x = 6, primero se evalúa la expresión 6, lo que da como resultado el objeto entero 6 en la memoria. Luego, Python agrega una entrada en la tabla de vinculación, asociando el nombre x con el objeto 6.

When y = x ∗ ∗2 is executed, the value of the expression x ∗ ∗2 is first evaluated by evaluating x, and 2, and then calculating x ∗ ∗2. When Python evaluates the literal 2, it creates a new object in the memory; then when it evaluates the identifier x, it looks up the binding table for an entry named x, which is bound to the object 6. The final value of the expression x ∗ ∗2 is saved to a memory object 36, and bound to the identifier y.

Cuando se ejecuta y = x ∗ ∗2, el valor de la expresión x ∗ ∗2 se evalúa primero evaluando x, y 2, y luego calculando x ∗ ∗2. Cuando Python evalúa el literal 2, crea un nuevo objeto en la memoria; luego, cuando evalúa el identificador x, busca en la tabla de vinculación una entrada llamada x, que está vinculada al objeto 6. El valor final de la expresión x ∗ ∗2 se guarda en un objeto de memoria 36 y se vincula al identificador y.

When the statement x = 3 is executed next, the expression 3 is first evaluated, resulting in a new object 3 in the memory, which is bound to the name x in the binding table. The old association between the name x and the object 6 is deleted, since one name can be bound to only one object. Note that the assignment statement always binds a name in the binding table with an object in the memory. Like all other Python statements, the execution is rather mechanic. Given this fact, it is easy to understand the reason why the value of y does not change to 9 automatically when x = 3 is executed.

Cuando se ejecuta a continuación la declaración x = 3, primero se evalúa la expresión 3, lo que da como resultado un nuevo objeto 3 en la memoria, que está vinculado al nombre x en la tabla de vinculación. Se elimina la antigua asociación entre el nombre x y el objeto 6, ya que un nombre puede vincularse a un solo objeto. Tenga en cuenta que la declaración de asignación siempre vincula un nombre en la tabla de vinculación con un objeto en la memoria. Como todas las demás declaraciones de Python, la ejecución es bastante mecánica. Dado este hecho, es fácil entender la razón por la cual el valor de y no cambia a 9 automáticamente cuando se ejecuta x = 3.

At this stage, the objects 2 and 6 are still in the memory, although they are not

bound to any identifiers. As the number of statements increases, the memory can

be filled with many such objects. They are no longer used, but still occupy memory

space. Python has a garbage collector that periodically removes unused objects from

the memory, so that more memory space can be available. Note also that Fig. 2.2 does

not show the binding table, although it exists in the memory, containing identifiers

that are irrelevant to the example.

For readers who know C++ and Java variables, it is worth nothing that Python

variables are not exactly the same as their C++ and java counterparts. Specifically,

the assignment statement in Python changes the value of a variable by changing

the binding (i.e. associating the identifier to a different object in the binding table),

while the assignment statement of C++ and Java directly changes the value of the

object that the identifier is associated with, without changing the binding between

identifiers and memory objects. Although in many cases, Python variable can be used

in the same way as C++ and Java variables from the programming perspective, an

understanding of this difference could be useful in a voiding subtle errors, especially

when mutability (Chap. 7) is involved.

Python provides a special statement, the del statement, for deleting an identifier from the binding table.

> > > x =1

> > > y =2

> > > x

1

> > > y

2

> > > del x

> > > x

T r a c e b a c k ( m o s t r e c e n t call last ) :

File " < p y s h e l l #5 > " , line 1 , in < module >

x

N a m e E r r o r : name ’x ’ is not d e f i n e d

> > > y

2

As can be seen from the example above, the del statement begins with the del

key word, followed by an identifier. It deletes the identifier from the binding table.

As the second last command shows, Python reports a name error when the value of

x is requested after the identifier x has been deleted from the binding table. After

an identifier is deleted, the objected that it is bounded to is not deleted immediately.

Instead, the garbage collector will remove it later when no other identifiers are bound

to it.

Each Python object has a unique identify, which is typically its memory address.

Python provides a function, id, which takes a Python object as its input argument,

and returns the identify of the object. For example, 2

> > > x = 1 2 3 4 5

> > > id ( x )

4 5 3 5 2 4 5 7 2 0

> > > y = x

> > > id ( y )

4 5 3 5 2 4 5 7 2 0

> > > y = 2 3 4 5 6

> > > id ( x )

4 5 3 5 2 4 5 7 2 0

> > > id ( y )

4 5 3 5 2 4 5 6 7 2

> > > id ( 1 2 3 4 5 )

4 5 3 5 2 4 5 9 8 4

When x is assigned to the value 12345, it is bound to a new object 12345 in the

memory. When y is assigned to the value of x, it is bound to the same object 12345.

Hence the identifies of x and y are the same. When y is reassigned to the value 23456,

a new object 23456 is constructed in the memory, occupying a new memory address,

and y is bound to his object. Hence the identify of y changes, while the identify of x

remains the same. The last command shows the identify of a new object, constructed

by the evaluation of the expression 12345. It is different from that of x, because every

time a literal is evaluated, a new object is constructed in the memory. Here is another

example.

> > > x = 1 2 3 4 5

> > > y = x

> > > id ( x

4 4 6 2 8 9 3 9 7 6

> > > id ( y )

4 4 6 2 8 9 3 9 7 6

> > > x = 1 2 3 4 5

> > > id ( x )

4 4 6 2 8 9 4 0 7 2

> > > id ( y )

4 4 6 2 8 9 3 9 7 6

When x is assigned to the value 12345 the second time, the right hand side of

the assignment statement is evaluated first, which leads to a new object having the

value 12345 in the memory. x is bound to this new object, while y remains the same.

Although the values of x and y are the same, their memory addresses, or identifies

are different, because they are bound to two different objects.

Note that the observations above may not hold for small numbers (e.g. 3 instead

of 12345). This is because to avoid frequent construction of new objects, Python

constructs at initialization a set of frequently used objects, including small integers,

so that they are reused rather than constructed afresh when their literals are evaluated.

> > > x =10

> > > y =10

> > > id ( x )

1 4 0 6 2 1 6 9 6 2 8 2 7 5 2

> > > id ( y )

1 4 0 6 2 1 6 9 6 2 8 2 7 5 2

In the example above, y is assigned the value 10 after x is assigned the value 10.

In each case, no new object is created when the literal 10 is evaluated, because an

object with the value 10 has been created in the memory location 140621696282752

to represent all objects with this value. 3

In summary, in addition to a value, a Python object also has an identify and a type.

Once constructed, the identify and type of an object cannot be changed. For number

objects, the value also cannot be changed after the object is constructed.

2.3 More Mathematical Functions Using the math and cmath Modules

Several mathematical functions have been introduced so far, which include addition, subtraction, multiplication, division, modulo and power. There are more mathematical functions that a typical calculator can do, such as factorial, logarithm and trigonometric functions. These functions are supported by Python through **a special module called math**.

A Python module is a set of Python code that typically includes the definition of

specific variables and functions. The next chapter will show that a Python module can

be nothing but a normal Python program. In order to use the variables and functions

defined in a Python module, the module must be imported. For example,

> > > **i m p o r t math**

> > > math . pi

3 . 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3

> > > math . e

2 . 7 1 8 2 8 1 8 2 8 4 5 9 0 4 5

In the example above, the math module is imported by the statement import math.

The import statement is the third type of statement introduced in this chapter, with the previous two being the assignment statement and the del statement. The import statement loads the content of a specific module, and adds the name of the module in the binding table, so that content of the module can be accessed by using the name, followed by a dot (.).

**La declaración de importación es el tercer tipo de declaración que se presenta en este capítulo, siendo las dos anteriores la declaración de asignación y la declaración del**. La declaración de importación carga el contenido de un módulo específico y agrega el nombre del módulo en la tabla de enlace, de modo que se pueda acceder al contenido del módulo usando el nombre, seguido de un punto (.).

Two mathematical constants **pi** and **e**, are defined in the math module, and accessed

by using ‘math.’ in the above example. Both pi (π) and e are defined as floating point

numbers, **up to the precision supported by a computer word.**

Mathematical functions can be accessed in the same way as constants, by using

‘math.’. For example, the factorial function returns the factorial of the input argument.

> > > i m p o r t math

> > > math . f a c t o r i a l (3)

6

> > > math . f a c t o r i a l (8)

4 0 3 2 0

The math module provides several classes of functions, including power and

logarithmic functions, trigonometric functions and hyperbolic functions. The two

basic power and logarithmic functions are math.pow(x, y) and math.log(x, y), which

take two floating point arguments x, and y, and return x y and logy x, respectively.

> > > i m p o r t math

> > > math . pow (2 , 5)

32.0

> > > math . pow (25 ,0.5)

5.0

> > > math . log (1000 ,10)

2 . 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 6

Note the rounding off error in the last example. The functions math.pow and math.log always return floating point numbers. The function math.log can also take one argument only, in which case it returns the natural logarithm of the input argument.

Tenga en cuenta el error de redondeo en el último ejemplo. Las funciones math.pow y math.log siempre devuelven números de coma flotante. La función math.log también puede tomar un solo argumento, en cuyo caso devuelve el logaritmo natural del argumento de entrada.

> > > i m p o r t math

> > > math . log (1)

0.0

> > > math . log ( math . e )

1.0

> > > math . log (10)

2 . 3 0 2 5 8 5 0 9 2 9 9 4 0 4

There is also a handy function to calculate the base-10 logarithm of an input float-

ing point number: math.log10(x). The function take a single floating point argument.

As two other special power functions, math.exp(x) can be used to calculate the value

of e x , and math.sqrt(x) can be used to calculate the square root of x.

The set of trigonometric functions that the math module provide include math.sin

(x), math.cos(x), math.tan(x), math.asin(x), math.acos(x) and math.atan(x), which

calculate the sine, the cosine, the tangent, the arc sine, the arc cosine, the arc tangent

of x, respectively.

> > > i m p o r t math

> > > math . sin (3)

0 . 1 4 1 1 2 0 0 0 8 0 5 9 8 6 7 2

> > > math . cos ( math . pi )

-1.0

> > > math . tan (3\* math . pi )

- 3 . 6 7 3 9 4 0 3 9 7 4 4 2 0 5 9 4 e -16

> > > math . asin (1)

1 . 5 7 0 7 9 6 3 2 6 7 9 4 8 9 6 6

> > > math . acos (1)

0.0

> > > math . atan (100)

1 . 5 6 0 7 9 6 6 6 0 1 0 8 2 3 1 5

Note the rounding off errors in some of the examples above. All angles in the

functions above are represented by radians. The math module provides two functions

to convert between radians and degrees: the math.degrees function takes a single

argument x, and converts x from radians to degrees; the math.radians function takes

a single argument x, and converts x from degrees to radians.

The set of hyperbolic functions include math.sinh(x), math.cosh(x), math.tanh(x),

math.asinh(x), math.acosh(x) and math.atanh(x), which calculate the hyperbolic

sine, the hyperbolic cosine, the hyperbolic tangent, the inverse hyperbolic sine, the

inverse hyperbolic cosine, and the inverse hyperbolic tangent of x, respectively.

There are more functions that the math module provides, including math.ceil(x),

which returns the smallest integer that is greater than or equal to x, and math.floor(x),

which returns the largest integer that is less than or equal to x. It does not make sense

to remember all the functions that Python provides for the purpose of programming,

although remembering a few commonly-used functions would be useful for the effi-

ciency of programming. A good practice is to keep the Python documentation at

hand, which is also easily accessible online. For example, searching for the key words

‘Python math’ using a search engine can lead to the Python documentation for the

math module.

2.3.1 Complex Numbers and the cmath Module

Complex literals. In some engineering disciplines, complex numbers are commonly

useful. A complex number consists of a real part and an imaginary part. While the

real part of a complex number is an arbitrary real number, represented by a floating

point number in Python, the imaginary part is based on **j**, the imaginary square root

of −1. Python represents the imaginary part of a complex literal by a floating point

number, followed by the special character j.

> > > **c =1 j**

> > > type ( c )

< type ‘ c o m p l e x ’ >

> > > c \* c

( -1+0 j )

In the example above, c is a complex number that has only the imaginary part,

1 j. The square of c is 1 j × 1 j = −1. In Python, if a complex number co-exists

in an expression with floating point numbers and integers, the type of the whole

expression becomes a complex number. Therefore, the value of the expression c ∗ c

is the complex number (−1 + 0 j).

1 j is a special complex number, with the real part being 0. In general, a complex

number is specified by the sum of its real part and imaginary part, as shown by IDLE

in the example above.

> > > a =1+2 j

> > > a

(1+2 j )

> > > type ( a )

< type ‘ c o m p l e x ’ >

> > > b =3+4 j

> > > b

(3+4 j )

> > > type ( b )

< type ‘ c o m p l e x ’ >

Type conversion from integers and floating point number into complex num-

bers. Similar to the construction of integer and floating number objects using the

functions int and float introduced earlier, a complex number can be constructed from

integers and floating point numbers by using the function complex.

> > > **c o m p l e x (1 ,2)**

(1+2 j )

> > > a = c o m p l e x ( -1 , 0.5)

> > > a

( -1+0.5 j )

As shown by the example above, the function complex takes two numeric argu-

ments specifying the real and imaginary components, respectively, and returns a

complex object.

Complex operators. Similar to integers and floating point numbers, complex

numbers also support arithmetic operations by using operators. The +, −, ∗, / and

∗∗ operators for integers and floating point numbers also apply to complex numbers.

> > > a =1+2 j

> > > b = -1+3 j

> > > a + b

5 j

> > > a - b

(2 -1 j )

> > > a \* b

( -7+1 j )

> > > a / b

( 0 . 5 - 0 . 4 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 4 j )

> > > a \*\*2

( -3+4 j )

Functions for complex numbers. The abs function, when applied to complex numbers, returns the magnitude of the number.

> > > a =3+4 j

> > > abs ( a )

5.0

In the example above, the input argument to the function call abs(a) is a complex

number, and the return value is a floating point number. However, no built-in function

or operator takes floating point numbers but results in a complex number. In other

words, the default domain in which Python handles mathematical expressions is real

numbers. For example, trying to obtain the square root of −1 by the ∗∗ operator, or

using the math.sqrt function, will result in an error.

> > > ( -1) \* \* 0 . 5

T r a c e b a c k ( m o s t r e c e n t call last ) :

File " < stdin > " , line 1 , in < module >

V a l u e E r r o r : n e g a t i v e n u m b e r c a n n o t be r a i s e d to a

f r a c t i o n a l p o w e r

> > > i m p o r t math

> > > math . sqrt ( -1)

T r a c e b a c k ( m o s t r e c e n t call last ) :

File " < stdin > " , line 1 , in < module >

V a l u e E r r o r : m a t h d o m a i n e r r o r

This choice of the default mathematical domain for Python is based on the fact

that real numbers are the most widely used, while complex numbers are common

only in specific fields. Some users of Python might not even know the existence of

complex numbers. As a result, the most natural choice is to leave their processing

only to specific modules. Python provides a module, cmath, for complex numbers.

The cmath module. The power and logarithmic functions that the cmath module

provides include cmath.exp, cmath.log, cmath.log10 and cmath.sqrt. They bare the

same names as their counterparts in the math module, with the difference being

that they can be applied to complex numbers, and can return complex numbers. For

example, to get the square root of −1 in the complex domain, the cmath.sqrt function

should be used.

> > > **i m p o r t c m a t h**

> > > cmath . sqrt ( -1)

1 j

The cmath module also provides the trigonometric functions cmath.sin, cmath.cos,

cmath.tan, cmath.asin, cmath.acos and cmath.atan, and the hyperbolic functions

cmath.sinh, cmath.cosh, cmath.tanh, cmath.asinh, cmath.acosh and cmath.atanh,

with exactly the same use as their math counterparts except of the domain.

While the abs(x) function returns the magnitude of a complex number x, the

function cmath.phase(x) returns the phase of x. The cmath.polar(x) function returns

the representation of a complex number x in polar coordinates, while the function

cmath.rect(r , p) returns the complex number x given it polar coordinates (r , p).

2.3.2 Random Numbers and the random Module

For one last example of modules in this chapter, **the random module provides functions for generating random numbers**. It is useful to a range of mathematical problems,

including a branch of numerical simulation methods that will be introduced in this

book.

Two important functions provided by the random module include **random.random** and **random.randint**. random.random() takes no input arguments, and returns a random floating point number in the range [0.0, 1.0). random.randint(a, b) takes two integer input arguments a and b, and returns a random number between a and b, inclusive.

> > > i m p o r t r a n d o m

> > > r a n d o m . r a n d o m ()

0 . 1 2 6 5 6 4 6 1 4 1 8 1 2 9 1 5

> > > r a n d o m . r a n d o m ()

0 . 5 7 0 1 2 9 4 6 3 7 3 9 0 3 6 2

> > > r a n d o m . r a n d o m ()

0 . 8 8 4 2 0 2 7 5 8 1 9 7 0 5 7 6

> > > r a n d o m . r a n d i n t (10 ,20)

12

> > > r a n d o m . r a n d i n t (10 ,20)

17

> > > r a n d o m . r a n d i n t (10 ,20)

17

> > > r a n d o m . r a n d i n t (10 ,20)

19

In general, many commonly-used mathematical functions are provided by Python,

and it would always be useful to look for a readily-available implementation via the

Python documentation and other resources. However, there are also cases where a

customized function is needed. The following chapters will introduce step by step

how complex functionalities can be achieved by the powerful Python language.

Exercises

1. What are the values of the following expressions?

(a) 1 + 3 ∗ 2 − 5 + 4

(b) 1 + 3 ∗ (2 − 5) + 4

(c) 5\*\*2\*\*2\*3+1

(d) 5\*\*(2\*\*2)\*3+1

(e) 1 + 3/2

(f) 1 + 3.0/2

(g) – 2 – 1

(h) –(2 – 1)

(i) 3.0 + 3/2

(j) 3 + 3/2.0

(k) – 1\*\*0.5

2. Use IDLE to calculate the following mathematical values.

(a) 10 5

(b) √10

(c) the roots of x2 − 7x + 10 = 0

(d) lg(2 + √5)

(e) the area of a circle with a radius of 5.5

(f) sin2.5

(g) the complex roots of x2 − 2x + 10 = 0

(h) 4!

(i) ∑128

k=32 k

(j) ∏17

k=3 k

3. What are the values of the following binary numbers if they are (a) non-negative;

or (b) 2’s Complements?

(a) 01001

(b) 100

(c) 1100

(d) 11111

(e) 11111111

4. Use IDLE to solve the following mathematical problems.

(a) A car runs at a constant speed of 20 km/h. When it passes another case, the

latter starts to accelerate in order to catch up. Assuming that the first case

keeps a constant speed, and the second car keeps a constant acceleration of

2 m/s 2 . After how many seconds will the second car catch up with the first

one?

(b) The annual interest rate of a savings account is 4.1 %. John has $10,000 in

his account, and aims at saving $50,000 within 5 years by depositing a fixed

amount of money to his account in the beginning of each year, including

this year. How much money does John need to save each year in order to

achieve his goal?

(c) In a shooting exercise, a coach stands 5 m away from a trainee, and throws

a target up vertically at 5 m/s. If the trainee must fire her gun exactly 0.5 s

after the throwing of the ball, then at which angle should she aim? If she

must fire the gun exactly 1 s after the throwing, then at which angle should

she aim?

(d) John deposited an initial sum of $3000 in his account. After 3 years, the

balance reaches $3335.8 due to composite interest. What is the interest rate

per annual?

(e) The three sides of a triangle are 3, 4 and 6m, respectively. What is its area?

5. What are the three most important properties of a Python object? Which of them

can change after the object is constructed, if the object is a number?

6. State the main differences between identifiers and literals. Given a token in a

program, how does Python know whether it is an identifier or a literal?