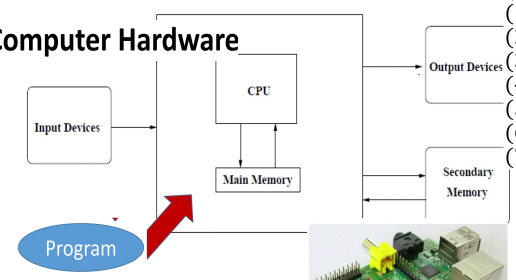
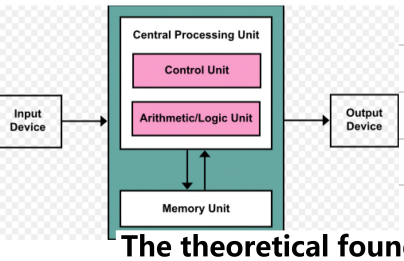
Computer Hardware



Von Neumann Architecture

The modern computer architecture is proposed by John Von Neumann



The theoretical foundation of computer science

- The theoretical foundation of computer science are built by Alan Turing
- Father of theoretical computer science and artificial intelligence
- Computability theory and Turing test

Memory/Storage

- High speed cache
- Internal RAM
- Internal ROM
- External RAM
- Flash
- Hard disk

Key components in a computer

- Central processing unit (CPU): execute your program. Similar to human brain, very fast but not that smart
- Input device: take inputs from users or other devices
- Output device: output information to users or other devices
- Main memory: store data, fast and temporary storage
- Secondary memory: slower but large size, permanent storage

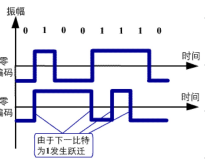
Central Processing Unit (Intel, AMD, ARM, etc)

- A processor contains two units, a control unit (CU) and an arithmetic/logic unit (ALU)
- CU is used to fetch commands from the memory
- ALU contains the electric circuits which can execute commands

Computers use voltage levels to represent 0 and 1

NRZL and NRZI coding

The instructions expressed in binary code is called machine language



Assembly Language → HIGH level (highest efficiency) → object oriented → C++ (code reuse) (platform dependent) → Java (platform independent) (one of the most popular) → Python (open source) (powerful libraries)

A new generation of general-purpose object oriented programming language

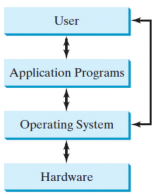
Powerful interfaces to integrate other programming languages (C/C++, Java, and many other languages)

Lower level languages have higher language efficiency (they are faster to run on a computer)

Higher level languages have higher development efficiency (it is easier to write programs in these languages)

Operating Systems

- The operating system (OS) is a low level program, which provides all basic services for managing and controlling a computer's activities
- Applications are programs which are built based upon an OS
- Main functions of an OS:
  - ✓ Controlling and monitoring system activities
  - ✓ Allocating and assigning system resources
  - ✓ Scheduling operations
- Popular OS: Windows, Mac OS, Linux, iOS, Android...



Data Representation and Conversion

- We use positional notation (进位记数法) to represent or encode numbers in a computer
  - binary (二进制)
  - decimal (十进制)
  - octal (八进制)
  - hexadecimal (十六进制)

base (基数) eg. 10

a set of symbols eg. {0,1,2,3,4,5,6,7,8,9}

Every number can be decomposed into the sum of a series of numbers, each is represented by a positional value times a weight

- a_i is the positional value (ranging from 0 to 9), while 10^i represents the weight

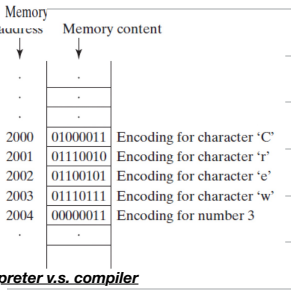
$(0)_8 = (000)_2$	0	0000 8 1000
$(1)_8 = (001)_2$	1	0001 9 1001
$(2)_8 = (010)_2$	2	0010 A 1010
$(3)_8 = (011)_2$	3	0011 B 1011
$(4)_8 = (100)_2$	4	0100 C 1100
$(5)_8 = (101)_2$	5	0101 D 1101
$(6)_8 = (110)_2$	6	0110 E 1110
$(7)_8 = (111)_2$	7	0111 F 1111

The units of information (data)

- Bit (比特/位): a binary digit which takes either 0 or 1
- Bit is the smallest information unit in computer programming
- Byte (字节): 1 byte = 8 bits, every English character is represented by 1 byte
- KB (千字节):1 KB = 2^10 B = 1024 B
- MB (兆字节):1MB = 2^20 B = 1024 KB
- GB (千兆字节):1GB = 2^30 B = 1024 MB
- TB (兆兆字节):1TB = 2^40 B = 1024 GB

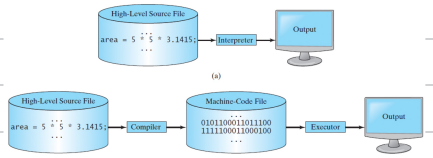
Memory and addressing

- A computer's memory consists of an ordered sequence of bytes for storing data
- Every location in the memory has a unique address
- The key difference between high and low level programming languages is whether programmer has to deal with memory addressing directly



Interpreter v.s. compiler

- Interpreter (解释器) is a computer program that directly executes, i.e. performs, instructions written in a programming or scripting language, without previously compiling them into a machine language program
- A compiler (编译器) is a computer program (or a set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language), with the latter often having a binary form known as object code



Elements of Python Language

- Vocabulary/words – Variables and Reserved words

You cannot use the following words as variables

False	None	True	and	as	assert	break	
class	continue	def	del	elif	else	except	
finally	for	from	global	if	import	in	
is	lambda	nonlocal	not	or	pass	raise	
return	try	while	with	yield			

Rules for defining variables in Python

- Must start with a letter or underscore _
- Can only contain letters, numbers and underscore
- Case sensitive
- Good: apple, car, myNumber123, _light
- Bad: 456aaa, #ab, var.12
- Different: apple, Apple, APPLE

Personal tips

- Use meaningful words as variable names
- Start with a lower letter
- Capitalize the first letter of each word
- Example: myBankAccountID, numOfCards, salaryAtYear1995...

Cascaded assignmen Example

```
>>> z = y = x = 2 + 7 + 2
>>> x, y, z
(11, 11, 11)
```

Simultaneous assignment Example

```
>>> c = "deepSecret"          # Set current password.
>>> c, o = "you'll never guess" # Set old password.
>>> c, o                      # See what passwords are.
('deepSecret', "you'll never guess")
>>> c, o = o, c               # Exchange the passwords.
```

Floor division

```
>>> time = 257          # Time in seconds.
>>> minutes = time // 60 # Number of complete minutes in time.
>>> print("There are", minutes, "complete minutes in", time, "seconds.")
There are 4 complete minutes in 257 seconds.
>>> 143 // 25
5
>>> 143.4 // 25
5.0
>>> 9 // 2.5
3.0
```

divmod()

```
>>> time = 257          # Initialize time.
>>> SEC_PER_MIN = 60    # Use a "named constant" for 60.
>>> divmod(time, SEC_PER_MIN)  # See what divmod() returns.
(4, 17)
>>> # Use simultaneous assignment to obtain minutes and seconds.
>>> minutes, seconds = divmod(time, SEC_PER_MIN)
>>> # Attempt to display the minutes and seconds in "standard" form.
>>> print(minutes, ":", seconds)
4 : 17
>>> # Successful attempt to display time "standard" form.
>>> print(minutes, ":", seconds, sep="")
4:17
>>> # Obtain number of quarters and leftover change in 143 pennies.
>>> quarters, cents = divmod(143, 25)
>>> quarters, cents
(5, 18)
```

Why use binary number?

- Easy to implement physically
- Simple calculation rules
- Easy to combine arithmetic and logic operations

整数 小数

Augmented assignment

Example

```
>>> x = 22        # Initialize x to 22.
>>> x += 7        # Equivalent to: x = x + 7
>>> x
29
>>> x -= 2 * 7    # Equivalent to: x = x - (2 * 7)
>>> x
15
```

Comments

- Anything after a "#" is ignored by Python
- Why comment?
  - ✓Describe what is going to happen in a sequence of code
  - ✓Document who wrote the code and other important information
  - ✓Turn off a line of code – usually temporarily

```
print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

String operations

- Some operators apply to strings
  - ✓"+": concatenation
  - ✓"*": multiple concatenation

"d" in print("%d"%v) means the variable v is integer type;
"f" in print("%f"%v) means the variable v is a floating point number;
"s" in print("%s"%v) means the variable v is a string.

Determine the space for the variable to be shown:
the number('8') before 'd' in print("%8d"%v) means 8 spaces for the integer v;
the number('7.2') before 'f' in print("%7.2f"%v) means 7 spaces for the floating point number v while keeping the 2 digits after decimal point '.'.

To align the result leftward, add the symbol '-': print("%-8d"%v) will make the integer v printed from the left of the 8 spaces. Without '-' will align the result rightward.

```
>>> string = "5 + 12" # Create a string.
>>> print(string)     # Print the string.
5 + 12
>>> eval(string)      # Evaluate the string.
17
>>> print(string, "=", eval(string))
5 + 12 = 17
>>> eval("print('Hello World!')") # Can call functions from eval()
Hello World!
>>> # Using eval(), we can accept all kinds of input...
>>> age = eval(input("Enter your age: "))
Enter your age: 57.5
>>> age
57.5
>>> age = eval(input("Enter your age: "))
Enter your age: 57
>>> age
57
>>> age = eval(input("Enter your age: "))
Enter your age: 40 + 17 + 0.5
>>> age
57.5
```

```
>>> eval("10, 32")          # String with comma-separated values.
(10, 32)
>>> x, y = eval("10, 20 + 12") # Use simultaneous assignment.
>>> x, y
(10, 32)
>>> # Prompt for multiple values.  Must separate values with a comma.
>>> x, y = eval(input("Enter x and y: "))
Enter x and y: 5 * 2, 32
>>> x, y
(10, 32)
```


```
>>> 7 == 7.0
True
>>> x = 0.1
>>> 1 == 10 * x
True
>>> 1 == x + x + x + x + x + x + x + x + x + x
False
>>> x + x + x + x + x + x + x + x + x + x
0.9999999999999999
>>> 7 != "7"
True
>>> 'A' == 65
False
```

Numeric Error 数值类错

```
>>> x = 0; y = 0.0; z = 0 + 0j
>>> bool(x), bool(y), bool(z)
(False, False, False)
>>> x = -1; y = 1.e-10; z = 0 + 1j
>>> bool(x), bool(y), bool(z)
(True, True, True)
>>> x = []; y = [0]; z = "0"
>>> bool(x), bool(y), bool(z)
(False, True, True)
```

Logical operators

not T = F T and T = T T or T = T
not F = T T and F = F T or F = T
F and F = F F or F = F

```
while True:
    line = input('Input a word:')
    if line[0] == '#':
        continue
    if line == 'done':
        break
    print(line)
print('Done')
```

Finding the smallest number

```
smallest_so_far = None
print('Before', smallest_so_far)

for num in [9, 39, 21, 98, 4, 5, 100, 65]:
    if smallest_so_far == None:
        smallest_so_far = num
    elif num < smallest_so_far:
        smallest_so_far = num
    print(smallest_so_far, num)

print('After', smallest_so_far)
```

Use try/except to capture errors

```
rawstr = input('Enter a number:')

try:
    ival = int(rawstr)
except:
    ival = -1

if ival>0:
    print('Nice work')
else:
    print('Invalid number')
```

== / != : Value is / isnot : Address

```
def greet(lang):
    if lang=='es':
        return 'Hola'
    elif lang=='fr':
        return 'Bonjour'
    else:
        return 'Hello'

>>> print(greet('en'),'Glenn')
Hello Glenn
>>> print(greet('es'),'Sally')
Hola Sally
>>> print(greet('fr'),'Michael')
Bonjour Michael
```

parameter (variable) (value) Argument

Void functions

- When a function does not return a value, it is called a "void" function
- Functions that return values are "fruitful" functions
- Void functions are "not fruitful"
- When a function has no return statement, it will return None

Global variable

- In a function, you can use keyword global to specify that a variable is a global variable
- Be very careful when define and use global variable

```
x = 1
def increase():
    global x
    x = x + 1
    print(x) # Displays 2

increase()
print(x) # Displays 2
```

Return multiple values

• Python allows a function to return multiple values
• The sort function returns two values;
when it is invoked, you need to
pass the returned values in a simultaneous assignment

```
def sort(number1, number2):
    if number1 < number2:
        return number1, number2
    else:
        return number2, number1

n1, n2 = sort(3, 2)
print("n1 is", n1)
print("n2 is", n2)
```

String library

• Python has a number of string functions which are in the string library
• These functions are built-into every string,
we invoke them by appending the function to the string variable

```
>>> greet = 'Hello, President Xu'
>>> zap = greet. lower()
>>> print(zap)
hello, president xu
>>> print(greet)
Hello, President Xu
>>> print('Hello, Junhua'.lower())
hello, junhua
```

upper() lower()

str.center(width[, fillchar])

width: 指定字符串的总宽度。
fillchar（可选参数）：指定用于填充字符串左右的字符。如果未提供 fillchar，则默认使用空格进行填充。
这个方法会在原始字符串的左右两侧填充字符，使得最终字符串的长度达到指定的宽度。如果需要居中对齐，填充字符将会平均地分布在字符串的左右两侧。
示例：

```
original_str = "Hello"
width = 10

# 使用空格填充居中对齐
centered_str = original_str.center(width)
print(centered_str)
# 输出: '  Hello   '

# 使用指定字符填充居中对齐
centered_str_custom_fill = original_str.center(width, '*')
print(centered_str_custom_fill)
# 输出: '**Hello***'
```

str.count(sub[, start[, end]])

sub: 要计数的子字符串。
start（可选参数）：指定开始搜索的起始位置（默认为0）。
end（可选参数）：指定搜索的结束位置（默认为字符串的末尾）。
这个方法返回一个整数，表示子字符串在指定范围内出现的次数。

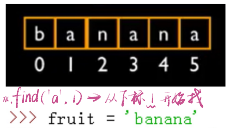
```
python
s = "abababababab"
sub = "ab"

# 计算子字符串 "ab" 在整个字符串中出现的次数
count = s.count(sub)
print(count)
# 输出: 5

# 从索引2到索引7的范围内计算子字符串 "ab" 的出现次数
count_within_range = s.count(sub, 2, 7)
print(count_within_range)
# 输出: 2
```

Searching a string

• We can use the find() function to search for a substring in a string
• find() finds the first occurrence of the target sub-string
• If the sub-string is not found, it returns -1
• Important: the string position starts from 0



find('a',i)→从下标i开始找a

```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print(pos)
2
>>> aa = fruit.find('z')
>>> print(aa)
-1
```

Prefixes

```
>>> line = 'Please submit your application'
>>> line.startswith('Please')
True
>>> line.startswith('p')
False
```

Opening files

• Before we can read the contents of a file, we must
tell Python which file we are going to work with and
what we will do with that file
• This is done with the open() function
• Open() returns a "file handle" - a variable used to
perform operations on files
• Kind of like "File -> Open" in a word processor

a special datatype

Using open()

• handle = open(filename, mode)
• Returns a handle used to manipulate the file
• Filename is a string
• Mode is optional, use 'r' if we want to read
the file, and 'w' if we want to write to the file

```
fhand = open('c:\Python35\myhost.txt','r')
print(fhand)
```

fhand.close()

Writing to a file

• To write a file, use the open() function with 'w' argument
• Use the write() method to write to the file

```
fhand = open('test.txt','w')
fhand.write('The first line\n')
fhand.write('The second line\n')
fhand.write('The third line\n')
fhand.close()
```

Range() function

• The range() function returns
a list of numbers
• We can construct an index loop
using for and an integer iterator

```
>>> x=range(4)
>>> x
range(0, 4)
>>> x[0]
0
>>> x[1]
1
>>> x[2]
2
>>> x[3]
3
>>> x=range(2, 10, 2)
>>> x[0]
2
>>> x[3]
8
>>> x[4]
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    x[4]
IndexError: range object index out of range
```

Concatenating lists using +

```
>>> a=[1, 2, 3]
>>> b=[4, 5, 6]
>>> c=a+b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]
```

Building a list from scratch

• We can create an empty list
using list(), and then add
elements using append()
method
• The list stays in order, and
new elements are added at
the end of the list

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print(stuff)
['book', 99]
>>> stuff.append('cookie')
>>> print(stuff)
['book', 99, 'cookie']
```

A list is an ordered sequence

• A list can hold many items and keeps
them in the order until we do something to
change the order
• A list can be sorted (i.e. change the order)
• The sort() method means "sort yourself"

```
>>> friend = ['Tom','Jerry','Bat']
>>> friends.sort()
>>> print(friends)
['Bat', 'Jerry', 'Tom']
>>> print(friends[1])
Jerry
>>>
>>> numbers = [1,2,5,100,32,7,97,1001]
>>> numbers.sort()
>>> print(numbers)
[1, 2, 5, 7, 32, 97, 100, 1001]
```

Best friends: strings and lists

• Use the split() method to break
up a string into a list of strings
• We think of these as words
• We can access a particular word
or loop through all the words

```
>>> line = 'A lot of spaces'
>>> etc = line.split()
>>> print(etc)
['A', 'lot', 'of', 'spaces']
>>>
>>> line = 'first,second,third'
>>> thing = line.split()
>>> print(thing)
['first,second,third']
>>> len(thing)
1
>>>
>>> thing = line.split(',')
>>> print(thing)
['first', 'second', 'third']
>>> print(len(thing))
3
```

```
counts = {'chuck':1,'fred':42,'jan':100}
for key,value in counts.items():
    print(key,value)
```

chuck 1
fred 42
jan 100

Built-in functions and lists

```
>>> numbers = [3,41,12,9,74,15]
>>> print(len(numbers))
6
>>> print(max(numbers))
74
>>> print(min(numbers))
3
>>> print(sum(numbers))
154
>>> print(sum(numbers)/len(numbers))
25.666666666666668
```

```
>>> myStr = 'Catch me if you can'
>>> words = myStr.split()
>>> print(words)
['Catch', 'me', 'if', 'you', 'can']
>>> print(len(words))
5
>>> print(words[0])
Catch
>>> for w in words: print(w)
Catch
me
if
you
can
```

The get() method

```
>>> counts = {'aaa':1,'bbb':2,'ccc':5}
>>> print(counts.get('eee',0))
0
```

Dictionary

Retrieving lists of keys and values

```
>>> jjj = {'chuck':1,'fred':42,'jan':100}
>>> print(list(jjj))
['jan', 'fred', 'chuck']

>>> print(list(jjj.keys()))
['jan', 'fred', 'chuck']

>>> print(list(jjj.values()))
[100, 42, 1]
>>> print(list(jjj.items()))
[('jan', 100), ('fred', 42), ('chuck', 1)]
```

Tuples and dictionaries

• The item() method in dictionaries
returns a list of (key, value) tuples

```
csev 2
cwen 4
>>> tups = d.items()
>>> print(tups)
dict_items([('csev', 2), ('cwen', 4)])
>>> print(list(tups))
[('csev', 2), ('cwen', 4)]

>>> tups = list(tups)
>>> tups[1]
('cwen', 4)
```

Tuples are comparable

```
>>> (0, 1, 2)<(5, 1, 2)
True
>>> (0, 1, 200000)<(0, 3, 4)
True
>>> ('Jones','Sally')<('Jones','Fred')
False
>>> ('Jones','Sally')>('Adams','Sam')
True
```

Sorting lists of tuples

```
>>> d={'a':10,'b':1,'c':22}
>>> t=d.items()
>>> t=list(t)
>>> t
[('c', 22), ('b', 1), ('a', 10)]
>>> t.sort()
>>> t
[('a', 10), ('b', 1), ('c', 22)]
```

Using sorted()

```
>>> d={'a':10,'b':1,'c':22}
>>> d.items()
dict_items([('c', 22), ('b', 1), ('a', 10)])
>>> t=sorted(list(d.items()))
>>> t
[('a', 10), ('b', 1), ('c', 22)]

>>> for k,v in t:
        print(k,v)

a 10
b 1
c 22
```

Sort by values instead of key

```
>>> d={'a':10,'b':1,'c':22}
>>> tmp = list()
>>> for k,v in d.items():
        tmp.append((v,k))

>>> print(tmp)
[(22, 'c'), (1, 'b'), (10, 'a')]
>>> tmp.sort(reverse=True)
>>> print(tmp)
[(22, 'c'), (10, 'a'), (1, 'b')]
```

Stripping whitespace

```
>>> greet = '   Hello Bob   '
>>> greet.lstrip()
'Hello Bob   '
>>> greet.rstrip()
'   Hello Bob'
>>> greet.strip()
'Hello Bob'
```

endwith()

Search and replace

```
>>> greet = 'Hello, Bob'
>>> newStr = greet.replace('Bob', 'Jane')
>>> print(newStr)
Hello, Jane
>>> newStr = greet.replace('o','X')
>>> print(newStr)
HellX, BXb
>>> newStr = greet.replace('z','X')
>>> newStr
'Hello, Bob'
```

The newline character

```
>>> stuff = 'Hello\nWorld'
>>> stuff
'Hello\nWorld'
>>> print(stuff)
Hello
World
>>> stuff = 'X\nY'
>>> print(stuff)
X
Y
>>> len(stuff)
3
```

Example: Finding the 10 most common words in a file

```
fhand = open('myhost.txt','r')
counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0)+1

lst = list()
for key,val in counts.items():
    lst.append((val,key))

lst.sort(reverse = True)

for val,key in lst[:10]:
    print(key,val)
```

s.join()

```
print('-'.join('new spam'))
>>> n-e-w- -s-p-a-m

print('-'.join(['apple','orange','grape']))
>>> apple-orange-grape
```

s.isdigit()

```
s = '12345'
result = s.isdigit()    # true or false
print(result)
>>> True
```

```
>>>"{} {}".format("hello", "world")    # 不设置指定位置，按默认顺序
'hello world'

>>> "{0} {1}".format("hello", "world")  # 设置指定位置
'hello world'

>>> "{1} {0} {1}".format("hello", "world")  # 设置指定位置
'world hello world'
```

```
number = '1'
sentence = "This number is %s" % number
print(sentence)

number = '1'
sentence = "This number is {}".format(number)
print(sentence)

>>> This number is 1
>>> This number is 1
```

```
j = 0
while j<3:
    print ("j = %d" % j)
    j += 1

>>> j = 0
>>> j = 1
>>> j = 2
```

```
print(ord('A')) # 输出 65，因为大写字母 A 的 Unicode 码点是 65
print(ord('a')) # 输出 97，因为小写字母 a 的 Unicode 码点是 97
print(ord('0')) # 输出 48，因为数字 0 的 Unicode 码点是 48

print(chr(65)) # 输出 'A'
print(chr(97)) # 输出 'a'
print(chr(48)) # 输出 '0'
```

```
l = [1,2,3]
print(l)

l.insert(0,[9]) # insert sth in the list
print(l)
l.insert(0,9)
print(l)

>>> [1, 2, 3]
>>> [[9], 1, 2, 3]
>>> [9, [9], 1, 2, 3]
```

All programs could be written with only three control structur

- sequence structure: sequential flow
- selection structure : conditional flow
- repetition structure: repeated flow

Sequence structure

Sequence structure is the default structure in Python
Python provides following selection structures

- if, if/else, if/elif

Python provides following repetition structures

- while, for

```
a=0.1
print(a)
print(a+a)
print(a+a+a)
print(a+a+a+a)
print(a+a+a+a+a)
print(a+a+a+a+a+a)
print(a+a+a+a+a+a+a)
print(a+a+a+a+a+a+a+a)
print(a+a+a+a+a+a+a+a+a)
```

```
0.1
0.2
0.30000000000000004
0.4
0.5
0.6
0.7
0.7999999999999999
0.8999999999999999
0.9999999999999999
```

round()函数的格式：round(x,d)
其中x为需要被处理的数据，d为需要保留的小数位数，d=0表示取整，d=1表示保留一位小数，以此类推。另外，round()会自动四舍五入。

函数注意，全局变量和参数的名称不能相同

形参（Formal Parameters）：
形参是函数定义中声明的参数，用于接收调用函数时传递的值。
形参是函数签名的一部分，它们是函数定义时列出的变量名，用于标识函数在执行时可以接受的输入。
形参在函数体内被当作局部变量使用。
例如，在下面的函数定义中，x 和 y 是形式参数：

```
def add(x, y):
    result = x + y
    return result
```

实参（Actual Parameters）：
实参是函数调用时提供给函数的值或变量。
实参是函数调用时传递给形参的具体数值或对象。
实参的值被传递给对应的形参，从而在函数体内使用。
在下面的函数调用中，2 和 3 是实际参数：

```
sum_result = add(2, 3)
```

在函数调用过程中，实参的值被传递给形参，使得函数能够使用这些值执行操作。在上述的例子中，x 接收了值 2，而 y 接收了值 3。