

上机

2023.5

Python内置函数

Python解释器提供了68个内置函数

<code>abs()</code>	<code>id()</code>	<code>round()</code>	<code>compile()</code>	<code>locals()</code>
<code>all()</code>	<code>input()</code>	<code>set()</code>	<code>dir()</code>	<code>map()</code>
<code>any()</code>	<code>int()</code>	<code>sorted()</code>	<code>exec()</code>	<code>memoryview()</code>
<code>ascii()</code>	<code>len()</code>	<code>str()</code>	<code>enumerate()</code>	<code>next()</code>
<code>bin()</code>	<code>list()</code>	<code>tuple()</code>	<code>filter()</code>	<code>object()</code>
<code>bool()</code>	<code>max()</code>	<code>type()</code>	<code>format()</code>	<code>property()</code>
<code>chr()</code>	<code>min()</code>	<code>zip()</code>	<code>frozenset()</code>	<code>repr()</code>
<code>complex()</code>	<code>oct()</code>		<code>getattr()</code>	<code>setattr()</code>
<code>dict()</code>	<code>open()</code>		<code>globals()</code>	<code>slice()</code>
<code>divmod()</code>	<code>ord()</code>	<code>bytes()</code>	<code>hasattr()</code>	<code>staticmethod()</code>
<code>eval()</code>	<code>pow()</code>	<code>delattr()</code>	<code>help()</code>	<code>sum()</code>
<code>float()</code>	<code>print()</code>	<code>bytearray()</code>	<code>isinstance()</code>	<code>super()</code>
<code>hash()</code>	<code>range()</code>	<code>callable()</code>	<code>issubclass()</code>	<code>vars()</code>
<code>hex()</code>	<code>reversed()</code>	<code>classmethod()</code>	<code>iter()</code>	<code>__import__</code>

运算符

- 运算符用于在表达式中对一个或多个操作数进行计算并返回结果值
- 表达式计算顺序取决于运算符的结合顺序和优先级
- 可以使用圆括号 “()” 强制改变运算顺序
- **【例】表达式中运算符的优先级示例**

>>> **11 + 22 * 3** #输出: **77**

77

>>> **(11 + 22) * 3** #输出: **99**

99

Python运算符及其优先级

Python 运算符	运算符说明	优先级	结合性
()	小括号	19	无
x[i]或 x[i1: i2 [:i3]]	索引运算符	18	左
x.attribute	属性访问	17	左
**	指数/乘幂	16	右
~	按位取反	15	右
+（正号）、-（负号）	符号	14	右
*, /, //, %	乘法、除法、整除、取余	13	左
+, -	加法、减法	12	左
>>, <<	移位	11	左
&	按位与	10	左
^	按位异或	9	左
	按位或	8	左
==, !=, >, >=, <, <=	比较	7	左
is, is not	同一性测试	6	左
in, not in	成员测试	5	左
not	逻辑非	4	右
and	逻辑与	3	左
or	逻辑或	2	左
exp1, exp2	逗号运算符	1	左

Python语句的书写规则

- (1) 使用换行符分隔，一般情况下，一行一条语句
- (2) 从第一列开始，前面不能有任何空格，否则会产生语法错误。注意，注释语句可以从任意位置开始；复合语句构造体必须缩进

```
>>> #正确
>>> print("abc") #报错。IndentationError: unexpected indent
```

- (3) 反斜杠 (\) 用于一个代码跨越多行的情况。如果语句太长，可以使用续行符 (\)
 - 三引号定义的字符串 ("""..."""或'''...''')、括号（圆括号中的表达式，函数调用的参数（圆括号中）、元组 ((...))、列表 ([...])、字典 ({...})），可以放在多行，而不必使用续行符
- (4) 分号 (;) 用于在一行书写多条语句

```
>>> print("如果语句太长，可以使用续行符 (\)，\
续行内容。")
```

```
>>> a=0; b=0; c=0 #变量a、b和c均指向int对象0。a=b=c=0
>>> s="abc";print(s) #变量s指向值为"abc"的str型实例对象，并输出abc
```

内置函数（1）

- dir()、type()、id()、help()、len()、sum()、max()、min()等
- 输入函数input()和输出函数print() 实现用户交互

【例】内置函数使用示例1

```
>>> s="书山有路勤为径，学海无崖苦作舟"
```

```
>>> len(s)          #返回字符串s的长度。输出：15
```

```
>>> s1 = [1,2,3,4,5]
```

```
>>> max(s1)         #返回列表s1的最大值。输出：5
```

```
>>> min(s1)         #返回列表s1的最小值。输出：1
```

```
>>> sum(s1)/len(s1) #返回列表s1的平均值。输出：3.0
```

类和对象

- 创建类对象
- 实例对象的创建和调用

class 类名:

.... 类体 ↵

anObject = 类名(参数列表) ↵

anObject.对象方法 或 anObject.对象属性

- 【例】类和对象示例 (Person.py) : 定义类Person, 创建其对象, 并调用对象方法

```
class Person:          #定义类Person
    def sayHello(self): #定义类Person的函数sayHello
        print('Hello, how are you?')
p = Person()          #创建对象
p.sayHello()         #调用对象的方法
```

程序运行结果如下

Hello, how are you?

```
class Person:
    def __init__(self,v):
        self.hello='Hello,how are you?{}'.format(v)
    def show(self):
        print(self.hello)
```

```
>>> p=Person('James')
>>> p.show()
... Hello,how are you?James
```

int类型（任意精度整数）（）

- 创建int对象

`int(x=0)`·····#创建 int 对象（十进制）↵

`int(x, base=10)`·····#创建 int 对象，指定进制为 base（2 到 36 之间）

- 【例】int对象示例

```
>>> int                                     #输出: <class 'int'>
>>> int(), int(123), int('456'), int(1.23) #输出: (0, 123, 456, 1)
>>> int('FF', 16), int('100', 2)           #输出: (255, 4)
>>> int('abc')                             #报错。ValueError: invalid literal for int() with base 10: 'abc'
>>> int(100, 2)                             #报错。TypeError: int() can't convert non-string with explicit base
```

- int对象的方法

- 【例】int对象方法示例

`i.bit_length()`: 返回 i 的二进制位数，不包括符号

```
>>> i = -10
>>> bin(i)                                     #数值转换为二进制字符串。输出: '-0b1010'
'-0b1010'
>>> i.bit_length(), int.bit_length(i) #返回i的二进制位数。输出: (4, 4)
(4, 4)
```


float类型（有限精度浮点数）(1)

- 浮点类型常量

- 【例】浮点类型字面量示例

```
>>> 3.14          #输出: 3.14
3.14
>>> type(3.14)    #输出: <class 'float'>
<class 'float'>
```

- 【例】float对象示例

`float(x)`

```
>>> float          #输出: <class 'float'>
>>> float(123), float('3.14')    #输出: (123.0, 3.14)
>>> float('Infinity'), float('-Infinity'), float('NaN') #输出: (inf, -inf, nan)
>>> float('123abc')    #报错。ValueError: could not convert string to float: '123abc'
```

举例 ^⓪	说明 ^⓪
1.23, -24.5, 1.0, 0.2 ^⓪	带小数点的数字字符串 ^⓪
1., .2 ^⓪	小数点的前后0可以省略 ^⓪
3.14e-10, 4E210, 4.0e+210 ^⓪	科学计数法(e或E表示底数10), 如 3.14e-10=3.14*10 ⁻¹⁰

bool数据类型和相关运算符

- bool数据类型包含两个值
 - True（真）或False（假）
- **【例4.8】布尔值字面量示例**

```
>>> True,False          #输出: (True, False)
(True, False)
>>> type(True),type(False) #输出: (<class 'bool'>, <class 'bool'>)
(<class 'bool'>, <class 'bool'>)
```

- **【例4.9】bool对象示例**

```
>>> bool(0)              #输出: False
False
>>> bool(1)              #输出: True
True
>>> bool("abc")          #输出: True
True
```

bool数据类型和相关运算符

(1) Python的任意表达式都可以评价为布尔逻辑值，故均可以参与逻辑运算。例如：

```
>>> not 0          #输出: True
>>> not 'a'        #输出: False
```

(2) $C = A \text{ or } B$ 。如果A不为0或者不为空或者为True，则返回A；否则返回B。仅在必要时才计算第二个操作数，即如果A不为0或者不为空或者为True，则不用计算B。“短路”计算。例如：

```
>>> 1 or 2          #输出: 1
>>> 0 or 2           #输出: 2
>>> False or True   #输出: True
>>> True or False   #输出: True
```

(3) $C = A \text{ and } B$ 。如果A为0或者为空或者为False，则返回A；否则返回B。仅在必要时才计算第二个操作数，即如果A为0或者为空或者为False，则不用计算B。即“短路”计算。例如：

```
>>> 1 and 2          #输出: 2
>>> 0 and 2           #输出: 0
>>> False and 2       #输出: False
>>> True and 2         #输出: 2
```

complex类型（复数）(1)

- 创建complex对象 `complex(real[,imag])` ... #创建 complex 对象（虚部可选）

- 【例4.10】复数字面量示例

```
>>> 1+2j          #输出: (1+2j)
>>> type(1+2j)    #输出: <class 'complex'>
```

- 【例4.11】complex对象示例

- complex对象属性和方法

```
>>> complex        #输出: <class 'complex'>
>>> c = complex(4, 5)
>>> c              #输出: (4+5j)
```

属性/方法	说明	示例
real	复数的实部	>>> (1+2j).real #结果: 1.0
imag	复数的虚部	>>> (1+2j).imag #结果: 2.0
conjugate()	共轭复数	>>> (1+2j).conjugate() #结果: (1-2j)

complex类型（复数）(2)

- 复数的运算

表达式	结果	说明
$1+2j$	$(1+2j)$	复数字面量
$(1+2j) + (3+4j)$	$(4+6j)$	加法
$(1+2j) - (3+4j)$	$(-2-2j)$	减法
$(1+2j) * (3+4j)$	$(-5+10j)$	乘法
$(1+2j) / (3+4j)$	$(0.44+0.08j)$	除法
$(1+2j) ** 2.0$	$(-3+4j)$	乘幂
$(1+2j) / 0.0$	运行时错误	除法。除数不能为0
<code>cmath.sqrt(1+2j)</code>	$(1.272019649514069+0.7861513777574233j)$	平方根（调用数学模块函数）
<code>cmath.sqrt(-2.0)</code>	$1.4142135623730951j$	复数的平方根

- 【例4.12】复数运算示例

```
>>> a = 1 + 2j
>>> b = complex(4, 5) #复数4 + 5j
>>> a + b              #复数相加。输出: (5+7j)
(5+7j)
>>> import cmath
>>> cmath.sqrt(b)      #复数的平方根
(2.280693341665298+1.096157889501519j)
```

比较关系运算和条件表达式

- 条件表达式通常用于选择语句中，用于判断是否满足某种条件
- 如果表达式的结果为数值类型（0）、空字符串（""）、空元组（()）、空列表（[]）、空字典（{}），则其bool值为False（假）；否则其bool值为True（真）。例如：123、"abc"、(1,2)均为True
- **【例】条件表达式示例**

```
>>> bool(123),bool("abc"),bool((1,2)),bool([0]),bool([]),bool(0)
(True, True, True, True, False, False)
>>> bool(1>2),bool(1>2 or 3>2),bool(1<=2 and 3>2)
(False, True, True)
```

关系和测试运算符（1）

- 关系运算符用于将两个操作数的大小进行比较。若关系成立，则比较的结果为True，否则为False
- 两个相同类型的对象之间的比较

```
>>> 1 > 2                #输出: False
>>> "ab123" > "ab12"     #输出: True
```

- 数值类型（包括布尔型，True自动转换为1，False自动转换为0）之间可以进行比较

```
>>> 1 > 1.23              #输出: False
>>> 2 > True              #输出: True
>>> 123 > "abc"           #报错。TypeError: unorderable types: int() > str()
```

算术运算符

n=8

运算符	含义	说明	优先级	实例	结果
**	乘幂	操作数的乘幂	1	n**3	512
+	一元+	操作数的值	2	+n	8
-	一元-	操作数的反数	2	-n	-8
*	乘法	操作数的积	3	n*n*2	128
/	除法	第二个操作数除第一个操作数	3	10 / n	1.25
//	整数除法	两个整数相除， 结果为整数	3	10 // n	1
%	模数	第二个操作数除第一个操作数后的余数	3	10 % n	2
+	加法	两个操作数之和	4	10 + n	18
-	减法	从第一个操作数中减去第二个操作数	4	n - 10	-2

位运算符

运算符	用法	含义	优先级	实例	结果
~	~op	按位求补	1	~0x1	-2 (-0x2)
<<	op1<<op2	将op1左移op2位	2	0xf0 << 4	3840 (0xf00)
>>	op1>>op2	将op1右移op2位	2	0xf0 >> 4	15 (0xf)
&	op1&op2	按位逻辑与	3	0xff00 & 0xf0f0	61440 (0xf000)
^	op1^op2	按位逻辑异或	4	0xff00 ^ 0xf0f0	4080 (0xff0)
	op1 op2	按位逻辑或	5	0xff00 0xf0f0	65520 (0xffff0)

2⁴

```
>>> 0xf0
240
>>> bin(0xff00)
'0b1111111100000000'
>>> bin(0xf0f0)
'0b1111000011110000'
>>> 0b1111000000000000
61440
>>> int(0b1111000000000000)
61440
>>>
```

类型转换示例

- 【例】隐式类型转换示例

```
>>> f = 123 + 1.23
>>> f          #输出: 124.23
>>> type(f)    #输出: <class 'float'>
>>> 123 + True  #True转换为1。输出: 124
>>> 123 + False #False转换为0。输出: 123
```

```
>>> int(1.23)   #输出: 1
>>> float(10)   #输出: 10.0
>>> bool("abc") #输出: True
>>> float("123xyz") #报错。 ValueError: could not convert string to float: '123xyz'
```

【例】数值数据类型示例：计算复利

```
nb = float(input("请输入本金：")) #输入本金并转换为浮点数
nr = float(input("请输入年利率：")) #输入年利率并转换为浮点数
ny = int(input("请输入年份：")) #输入年份并转换为整数
amount = nb * (1+nr/100) ** ny #计算复利
print('本金利率和为： %0.2f'%amount) #输出复利，保留两位小数
```

程序运行结果如下。↵

请输入本金： 1000 ↵

请输入年利率： 6 ↵

请输入年份： 10 ↵

本金利率和为： 1790.85 ↵

all() 函数用于判断给定的可迭代参数 **iterable** 中的所有元素是否不为 0、"、False 或者 **iterable** 为空，如果是返回 **True**，否则返回 **False**。(iterable 为空也返回 **true**)；
any() 函数用于判断给定的可迭代参数 **iterable** 是否全部为空对象，如果都为空、0、false，则返回 **False**

```
>>> all((1, 2, 3))
True
>>> all((1, 2, ""))
False
>>> all("sfsffs")
True
>>> all("")
True
...
complex(1, 2) #(1 + 2j)
complex(1)    #(1 + 0j)
```

```
>>> any((1, 2, 3.
True
>>> any([])
False
...
|
```

```
divmod(7, 2) #(3,1)
divmod(8, 2) #(4,0)
```

cmp(x,y) 函数用于比较2个对象，如果 $x < y$ 返回 -1，如果 $x == y$ 返回 0，如果 $x > y$ 返回 1。

frozenset() 返回一个冻结的集合，冻结后集合不能再添加或删除任何元素。

hash() 用于获取一个对象（字符串或者数值等）的哈希值 散列值。(可哈希-不可变)

```
>>> some_days = ["Mon", "Tue", "Wed", "Thu"]
>>> some_days[2] = "Fri"
>>> fixed_days = frozenset(some_days)
>>> fixed_days[2] = "Wed"
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    fixed_days[2] = "Wed"
TypeError: 'frozenset' object does not support item assignment
...
|
```

哈希算法

```
import hashlib
data="testing"
result0=hashlib.sha1(data.encode("utf-8")).hexdigest()
result1=hashlib.sha256(data.encode("utf-8")).hexdigest()
result2=hashlib.sha512(data.encode("utf-8")).hexdigest()
for i in range(3):
    print("result"+str(i),'=',eval("result"+str(i)))
```

雪崩效应

```
import hashlib
text1="blockchain with python"
text2="blockchain with python."
hash1=hashlib.sha256(text1.encode("utf-8"))
hash2=hashlib.sha256(text2.encode("utf-8"))
print(hash1.hexdigest())
print(hash2.hexdigest())
print(hash1.hexdigest()!=hash2.hexdigest())
```

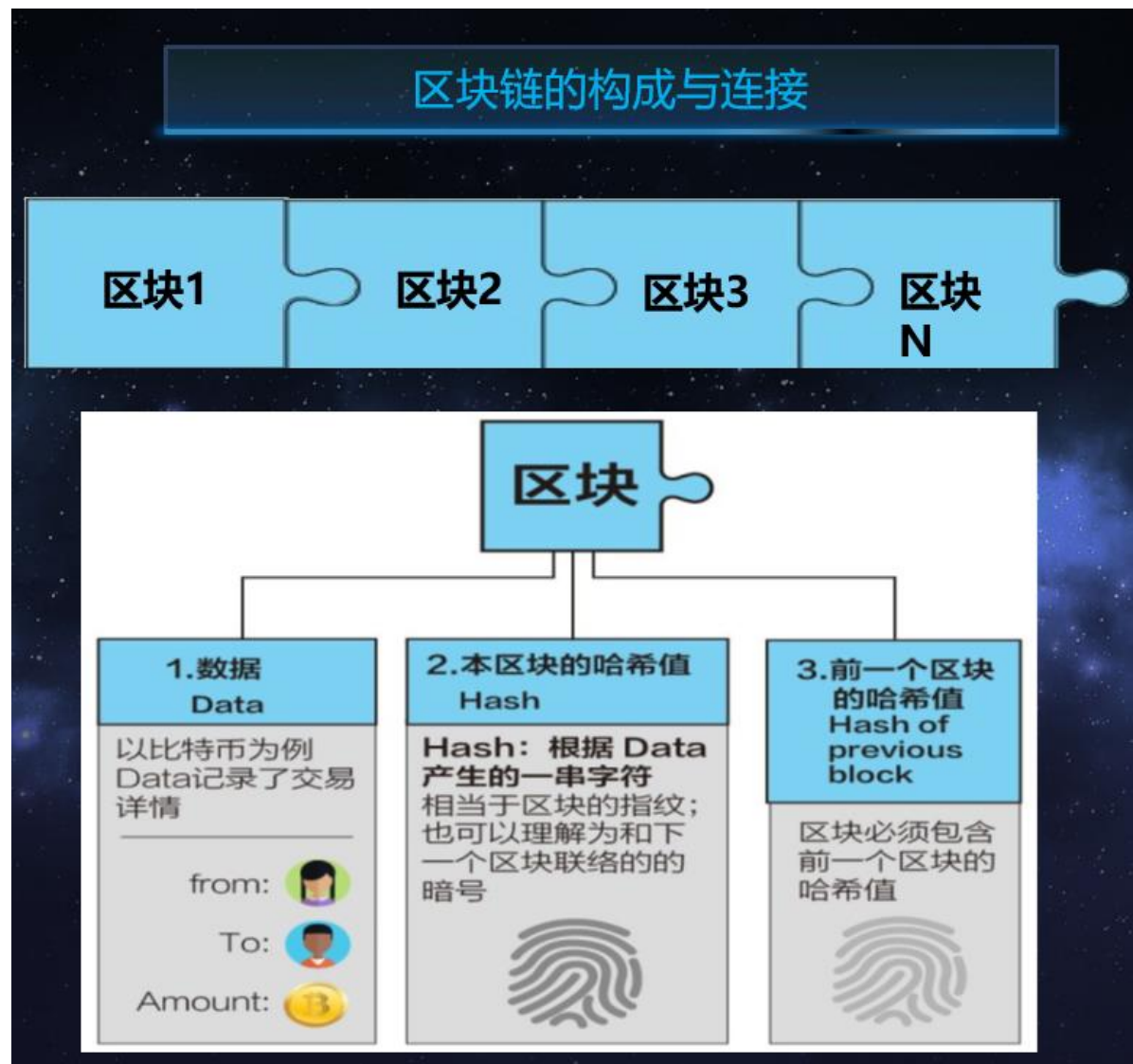
```

from datetime import datetime
import hashlib# 导入hashlib包
# 定义区块类
class Block:
    # 构造函数，包含4大属性
    def __init__(self,index,timestamp,data,previous_hash):
        # 前一个区块的哈希值
        self.previous_hash = previous_hash
        # 本区块所包含的数据
        self.data = data
        # 本区块所持有的时间戳
        self.timestamp = timestamp
        # 本区块的索引
        self.index = index
        # 本区块的哈希值
        self.hash = self.hash_block()

    # 求区块的哈希值(当前的哈希值)
    def hash_block(self):
        sha = hashlib.sha256()
        sha.update((
            str(self.index)+
            str(self.timestamp)+
            str(self.data)+
            str(self.previous_hash)).encode("utf8"))
        return sha.hexdigest()

    def create_genesis_block(): #定义创建创世区块的方法
        #调用区块链类的构造函数返回创世块
        return Block(0, datetime.now(), "Genesis Block", "0")

```



```
def next_block(last_block):#后续区块定义
    this_index = last_block.index + 1 #后续区块序号在前一个序号基础上递增
    this_timestamp = datetime.now() #当前时间作为时间戳
    this_data = " Transaction Data " + str(this_index) #后续区块数据设为当前数据
    与其序号的连接
    this_hash = last_block.hash #后续区块散列值等于前序区块生成的散列值
    #调用区块类的构造方法返回后续区块
    return Block(this_index, this_timestamp, this_data, this_hash)
```

```
#创建区块链并添加创世块
blockchain = [create_genesis_block()] #添加创世块作为列表第一个元素
previous_block = blockchain[0]#此处的0对应第一个index
print("创世区块的创造时间:{}".format(previous_block.timestamp))
print("Block Data:{}".format(previous_block.data))
print("previous Hash:{}".format(previous_block.previous_hash))
print("Hash: {}".format(previous_block.hash))
```

```
#创世块后需要继续添加的区块个数
num_of_blocks = 20
```

区块链的连接与断链

genesis_block	previous_hash	0
	data	this is the genesis block
	hash:	bb8d6f67262cdda0872593280a5b0642a07a1196bbcc3042379811cf25b8c9d1
block_1	previous_hash	bb8d6f67262cdda0872593280a5b0642a07a1196bbcc3042379811cf25b8c9d1
	data	this is block 2
	hash:	0906977a66813074c24013ae6ddcbdbf6e437face835eae43d1a45eb663ea5ec
block_2	previous_hash	eef23e3fd8de411b5138dfc7c936f343fcb11cfce11757e6118b172a192307d5
	data	this is block 2
	hash:	802631e0a0a6efcace29b7d826b64b9bc59a29ec4b220afd562f1ccd66641961
block_3	previous_hash	802631e0a0a6efcace29b7d826b64b9bc59a29ec4b220afd562f1ccd66641961
	data	this is block 3
	hash:	ea4991a6b820214c68527d752b409d8a7a6e9cbd837373d737903e0f4d988d80

#通过循环将区块添加进区块链

for i in range(0, num_of_blocks):

block_to_add = next_block(previous_block) #创建后续区块

blockchain.append(block_to_add) #添加后续区块至区块链

previous_block = block_to_add #上一个区块的内容

#向所有区块链节点发布信息

print("Block #{} has been added to the blockchain!".format(block_to_add.index))

print("Block Timestamp:{}".format(block_to_add.timestamp))

print("Block Data:{}".format(block_to_add.data))

print("previous Hash:{}".format(block_to_add.previous_hash))

print("Hash: {}\n".format(block_to_add.hash))

```
class Block:
    # 构造函数, 包含4大属性
    def __init__(self, index, timestamp, data, previous_hash):
        # 前一个区块的哈希值
        self.previous_hash = previous_hash
        # 本区块所包含的数据
        self.data = data
        # 本区块所持有的时间戳
        self.timestamp = timestamp
        # 本区块的索引
        self.index = index
        # 本区块的哈希值
        self.hash = self.hash_block()
```



create_genesis_block

next_block

```
def create_genesis_block(): #定义创建创世块的方法
    #调用区块链类的构造函数返回创世块
    return Block(0, datetime.now(), "Genesis Block", "0")
```

```
#创建区块链并添加创世块
blockchain = [create_genesis_block()] #添加创世块作为列表第一个元素
previous_block = blockchain[0]
print("创世区块的创造时间:{}".format(previous_block.timestamp))
print("Block Data:{}".format(previous_block.data))
print("previous Hash:{}".format(previous_block.previous_hash))
print("Hash: {}\n".format(previous_block.hash))
```


内置map()函数(1)

- map()函数实现为内置的map(f, iterable, ...)可迭代对象，将函数f应用于可迭代对象，返回结果为可迭代对象
- **【例8】 map()函数示例1：自定义函数is_odd，应用该函数到可迭代对象的每一个元素，返回是否为奇数的可迭代对象结果**

```
>>> def is_odd(x):  
    return x % 2 == 1
```

- ```
>>> list(map(is_odd, range(5)))
```

 #输出: **[False, True, False, True, False]**
- **[False, True, False, True, False]**

```
>>> list(map(abs, [1,-3, 5, 6, -2, 4]))
```

 #输出: **[1, 3, 5, 6, 2, 4]**  
**[1, 3, 5, 6, 2, 4]**

# 内置filter()函数

- filter()函数实现为内置的filter(f, iterable)可迭代对象（参见第9章），将函数f应用于每个元素，然后根据返回值是True还是False决定保留还是丢弃该元素，返回结果为可迭代对象

- 【例8】filter()函数示例1：返回奇数的可迭代对象

```
>>> def is_odd(x):
 return x % 2 == 1
>>> list(filter(is_odd, range(10))) #输出: [1, 3, 5, 7, 9]
```

- 【例8】filter()函数示例2：返回三位数的回文数（正序和反序相同）可迭代对象

```
>>> def is_palindrome(x):
 if str(x) == str(x)[::-1]:
 return x
```

```
>>> list(filter(is_palindrome, range(100,1000)))
```

```
[101, 111, 121, 131, 141, 151, 161, 171, 181, 191, 202, 212, 222, 232, 242, 252, 262, 272, 282, 292, 303,
313, 323, 333, 343, 353, 363, 373, 383, 393, 404, 414, 424, 434, 444, 454, 464, 474, 484, 494, 505, 515,
525, 535, 545, 555, 565, 575, 585, 595, 606, 616, 626, 636, 646, 656, 666, 676, 686, 696, 707, 717, 727,
737, 747, 757, 767, 777, 787, 797, 808, 818, 828, 838, 848, 858, 868, 878, 888, 898, 909, 919, 929, 939,
949, 959, 969, 979, 989, 999]
```

# Lambda表达式和匿名函数

- lambda是一种简便的、在同一行中定义函数的方法。lambda实际上生成一个函数对象，即匿名函数
- Lambda表达式的基本格式为
- **【例8.38】 匿名函数示例1**

`lambda arg1,arg2... : <expression>+'`

```
>>> f = lambda x,y: x + y
```

```
>>> type(f) #输出: <class 'function'>
```

```
<class 'function'>
```

```
>>> f(12, 34) #计算两数之和。输出: 46
```

```
46
```

# 匿名函数示例

**【例】匿名函数应用示例1：过滤列表，返回元素为奇数的可迭代对象**

```
>>> list(filter(lambda x:x%2==1, range(10))) #输出: [1, 3, 5, 7, 9]
```

**【例】匿名函数应用示例2：过滤列表，返回元素大于零的可迭代对象**

```
>>> list(filter(lambda x:x>0, [1,0,-2,8,5])) #输出: [1, 8, 5]
```

**【例】匿名函数应用示例3：过滤列表，返回元素平方的可迭代对象**

```
>>> list(map(lambda x:x*x, range(10)))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```

>>> i=[1,2,3]
>>> j=reversed(i)
>>> list(j)
[3, 2, 1]
>>> type(j)
<class 'list_reverseiterator'>
>>> j
<list_reverseiterator object at 0x000001F9A8DF3D00>
>>>

```

```

x=[4, 6, 2, 3, 9, 7]
y=x
x.sort()
print(x)
print(y)

```

```

x=[4, 6, 2, 3, 9, 7]
y=x[:]
x.sort()
print(x)
print(y)

```

a=[3,4,1,9,0,4]

sorted(a)=?

a=?

```

>>> sorted(a)
... [0, 1, 3, 4, 4, 9]

```

```

>>> a
... [3, 4, 1, 9, 0, 4]

```

```

>>> a=[3, 4, 1, 9, 0, 4]
>>> a.sort()
>>> a

```

```

>>> a=[3, 4, 1, 9, 0, 4]
>>> a.sort()
>>> a
... [0, 1, 3, 4, 4, 9]

```

# 内置sorted()函数

- 内置sorted()函数把一个可迭代对象进行排序，返回结果列表
- **【例】sorted()函数示例**

```
>>> sorted([1,6,4,-2,9]) #按数值自然排序: [-2, 1, 4, 6, 9]
```

```
>>> sorted([1,6,4,-2,9], reverse=True) #按数值逆序排序: [9, 6, 4, 1, -2]
```

```
>>> sorted([1,6,4,-2,9], key=abs) #按绝对值排序: [1, -2, 4, 6, 9]
```

```
>>> sorted(['Dog','cat','Rabbit']) #按字符串字典序排序: ['Dog', 'Rabbit', 'cat']
```

```
>>> sorted(['Dog','cat','Rabbit'], key=str.lower) #字符串排序不区分大小写
['cat', 'Dog', 'Rabbit']
```

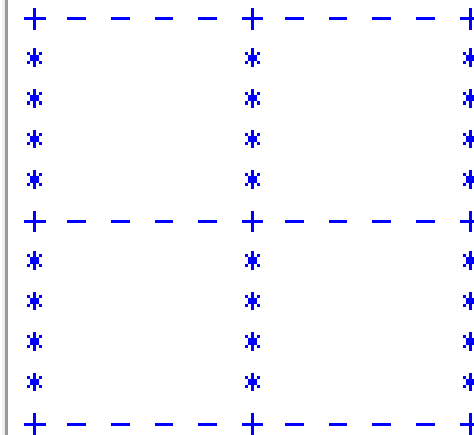
```
>>> sorted(['Dog','cat','Rabbit'], key=len) #按字符串长度排序
['Dog', 'cat', 'Rabbit']
```

```
>>> sorted([('Bob',75),('Adam',92),('Lisa',88)]) #默认按元组第一个元素排序
[('Adam', 92), ('Bob', 75), ('Lisa', 88)]
```

```
>>> sorted([('Bob',75),('Adam',92),('Lisa',88)],key=lambda t:t[1]) #按元组第二个元素排序
[('Bob', 75), ('Lisa', 88), ('Adam', 92)]
```

# 特殊田字格的打印

- for i in range(11):
- if i in [0,5,10]:
- print("+ - - - - + - - - - +")
- else:
- print("\*         \*         \*")



```
+ - - - - + - - - - +
* * *
* * *
* * *
* * *
+ - - - - + - - - - +
* * *
* * *
* * *
* * *
+ - - - - + - - - - +
```

以天天向上为例，假设以7天为周期，连续学习3天能力不变，从第4天开始至第7天每天能力增长为前一天的1%，请编写程序求假如初始能力值为1，连续学习365天后能力值是多少？

- `dayup = 1`
- `dayfactor = 0.01`
- `for j in range(1,366):`
- `if j % 7 in [4,5,6,0]:`
- `dayup = dayup * (1 + dayfactor)`
- `print('%f, the result is %.2f' %(dayfactor, dayup))`



- — a和b是两个列表变量，列表a为[2,5,8] 已给定，键盘输入列表b ,计算a中元素与b中对应元素乘积的累加和。

- 二 梁朝伟被香港金像奖提名为最佳男主角，但是要想最终当选，他需要获得下边所有的团体的一半以上的投票，而评价一个团体是否投票支持他的标准是该团体的所有人的一半以上都支持他，请写出一个程序，使得输入团体个数 各个团体的人数时，可以得到至少需要多少人支持他才可以当选。（注意：输入的团体的人数是没有按照顺序排列的）

`split(sep, num)`

sep为分隔符，不写sep时，默认表示用 空格，\n，\t 分隔字符串

```
string = "abc abc\ndef\t"
print(string.split())
打印结果：['abc', 'abc', 'def']
```

```
请输入团体数目 各团体的人数（以空格隔开）： 2 4 3
4
```

```
请输入团体数目 各团体的人数（以空格隔开）： 3 9 6 10
11
```

## format()方法的格式控制

### 槽内部对格式化的配置方式

{ <参数序号> : <格式控制标记> }

| :        | <填充>          | <对齐>                     | <宽度>         | < , >        | <. 精度>                      | <类型>                                            |
|----------|---------------|--------------------------|--------------|--------------|-----------------------------|-------------------------------------------------|
| 引导<br>符号 | 用于填充的<br>单个字符 | < 左对齐<br>> 右对齐<br>^ 居中对齐 | 槽设定的<br>输出宽度 | 数字的<br>千位分隔符 | 浮点数小数<br>精度 或 字符串<br>最大输出长度 | 整数类型<br>b, c, d, o, x, X<br>浮点数类型<br>e, E, f, % |

键盘输入正整数n，按要求把n输出到屏幕，格式要求：宽度为20个字符，减号字符-填充，右对齐，带千位分隔符。如果输入正整数超过20位，则按照真实长度输出

# format()方法的格式控制

| :                                                                                             | <填充> | <对齐> | <宽度> | <,>          | <.精度>                           | <类型>                                            |
|-----------------------------------------------------------------------------------------------|------|------|------|--------------|---------------------------------|-------------------------------------------------|
| >>>"{0:,.2f}".format(12345.6789)<br>'12,345.68'                                               |      |      |      | 数字的千位<br>分隔符 | 浮点数小数<br>精度 或 字<br>符串最大输<br>出长度 | 整数类型<br>b, c, d, o, x, X<br>浮点数类型<br>e, E, f, % |
|                                                                                               |      |      |      |              |                                 |                                                 |
|                                                                                               |      |      |      |              |                                 |                                                 |
| >>>"{0:b},{0:c},{0:d},{0:o},{0:x},{0:X}".format(425)<br>'110101001,Σ,425,651,1a9,1A9'         |      |      |      |              |                                 |                                                 |
| >>>"{0:e},{0:E},{0:f},{0:%}".format(3.14)<br>'3.140000e+00,3.140000E+00,3.140000,314.000000%' |      |      |      |              |                                 |                                                 |

b-二进制 c-Unicode d-十进制 o-八进制 x-小写十六进制 X-大写十六进制



# format()方法的格式控制

| :        | <填充>          | <对齐>                     | <宽度>         | <, >                                                                       | <. 精度> | <类型> |
|----------|---------------|--------------------------|--------------|----------------------------------------------------------------------------|--------|------|
| 引导<br>符号 | 用于填充的<br>单个字符 | < 左对齐<br>> 右对齐<br>^ 居中对齐 | 槽设定的<br>输出宽度 | <pre>&gt;&gt;&gt; "{0:=^20}".format("PYTHON")<br/>'=====PYTHON====='</pre> |        |      |

$$x = \sqrt{\frac{(3^4 + 5 \times 6^7)}{8}}$$

计算下列数学表达式的结果并输出，小数后保留三位

```
print('表达式的值为{:.3f}'.format(x))
```

- 三 接收用户输入的一个小于 20 的正整数，在屏幕上逐行递增显示从 001 到该正整数，数字显示的宽度为 3，不足位置补 0，后面追加一个空格，然后显示 '\*' 号， '\*' 号的个数等于行首数字。

```
请输入一个小于20的正整数: 12
001 *
002 **
003 ***
004 ****
005 *****
006 ****
007 *****
008 *****
009 *****
010 *****
011 *****
012 *****
```

- 四 `a=[1,34,2,1,9,True,False,1,"True",3,1]`  
请统计其中有多少个1?

- 五 神奇的四位数：请找出在所有的四位数中满足以下几个条件的数字：所有的位数均不相同；所有的位数加起来和为6；该四位数是11的倍数