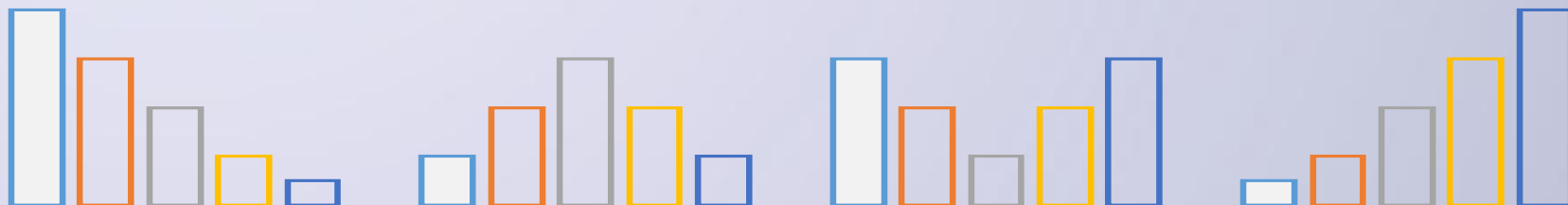


Python语言程序设计

成都信息工程大学区块链产业学院

刘硕

第3章 Python的基本数据类型



目录

3.1 三种数字类型及操作

3.2 math库的应用

3.3 天天向上的力量

3.4 time库的使用

3.5 文本进度条

前情回顾

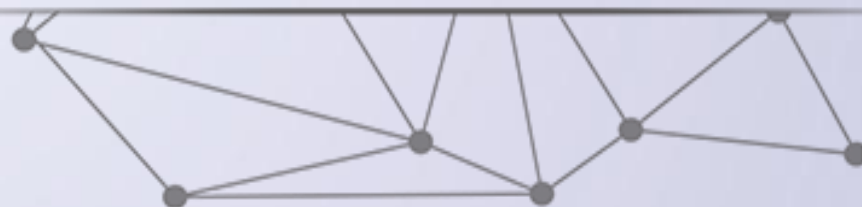
- 温度转换代码（`eval`函数 `input....`）
- 模块的导入和对象
- **Turtle**库的使用

本节内容

- 再（细）谈数字类型和`math`库
- 天天向上的力量-代码实例(`for while`)
- 字符串及其内置函数 格式化(`format`)等
- 文本条进度诠释字符串的格式化(`\r`)



3.1 三种数字类型及操作





数字类型

- 程序元素：010/10，存在多种可能
 - 表示十进制整数值10
 - 类似人名一样的字符串
- 数字类型对Python语言中数字的表示和使用
进行了定义和规范



数字类型

Python语言包括三种数字类型

- 整数类型
- 浮点数类型
- 复数类型



整数类型

- 与数学中的整数概念一致，没有取值范围限制
- `pow(x, y)`函数：计算 x^y
- 打开IDLE
 - 程序1： `pow(2,10)` , `pow(2,15)`
 - 程序2： `pow(2, 1000)`
 - 程序3： `pow(2, pow(2,15))`



整数类型

■ 示例

■ 1010, 99, -217

■ 0x9a, -0X89 (0x, 0X开头表示16进制数)

■ 0b010, -0B101 (0b, 0B开头表示2进制数)

■ 0o123, -0O456 (0o, 0O开头表示8进制数)

进制转换函数

- `bin()`: 转换为2进制; 例: `bin(int("f",16))` 输出: `'0b1111'` .`bin(15)`同样输出`'0b1111'`.
- `oct()`: 转换为8进制;
- `hex()`: 转换为16进制。
- `int()`: 转换为10进制; 语法: `int(字符串,字符串进制)`。例: `int("f",16)` 输出为15; `int('11',2)`输出为3

```
>>> int(0b1111)
15
>>> hex(0b1111)
'0xf'
```

#十进制转换二进制

```
i = 16
j = bin(i)
print(j)
```

#十进制转换十六进制

```
i = 16
j = hex(i)
print(j)
```

#十进制转换八进制

```
i = 16
j = oct(i)
print(j)
```

#二进制转换十进制


```
i = "10"
j = int(i, 2) #在其他进制转换成十进制时，i 的类型需要是字符串类型；
print(j)
```

#八进制转换十进制

```
i = "10"
j = int(i, 8) #在其他进制转换成十进制时，i 的类型需要是字符串类型；
print(j)
```

#十六进制转换十进制

```
i = "10"
j = int(i, 16) #在其他进制转换成十进制时，i 的类型需要是字符串类型；
print(j)
```



浮点数类型

- 带有小数点及小数的数字
- Python语言中浮点数的数值范围存在限制，
小数精度也存在限制。这种限制与在不同计算机系统有关



浮点数类型

```
>>> import sys
>>> sys.float_info
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)
>>>
```

10^{1024}



浮点数类型

■ 示例

■ 0.0, -77., -2.17

■ 96e4, 4.3e-3, 9.6E5 (科学计数法)

■ 科学计数法使用字母 “e” 或者 “E” 作为幂的符号，以10为基数。科学计数法含义如下：

$$\langle a \rangle e \langle b \rangle = a * 10^b$$



复数类型

- 与数学中的复数概念一致, $z = a + bj$, a 是实数部分, b 是虚数部分, a 和 b 都是浮点类型, 虚数部分用 j 或者 J 标识

- 示例:

$12.3 + 4j$, $-5.6 + 7j$



复数类型

- $z = 1.23e-4 + 5.6e+89j$ (实部和虚部是什么?)
- 对于复数 z , 可以用 `z.real` 获得实数部分 ,
`z.imag` 获得虚数部分
- `z.real = 0.000123` `z.imag = 5.6e+89`

- 小练习

既然浮点数可以表示所有整数数值，python 为何还同时提供整数型？

因为 Python 的整数取值不设限，只受配置影响；浮点数却有限制，最大浮点数是 $1.7976931348623157\text{e}+308$ 。

也就是说，如果没有整数数据类型，超过 $1.7976931348623157\text{e}+308$ 的整数将无法计算。

运算符%能不能对浮点数进行求余操作？

可以



数字类型的操作

数字类型的关系

类型间可进行混合运算，生成结果为"最宽"类型

- 三种类型存在一种逐渐"扩展"或"变宽"的关系：

整数 -> 浮点数 -> 复数

- 例如：123 + 4.0 = 127.0 (整数+浮点数 = 浮点数)
- 不同数字类型之间可以进行混合运算，运算后生成结果为最宽类型

```
>>> x=complex(2, 3)
>>> x
(2+3j)
>>> y=1
>>> x+y
(3+3j)
```



内置的数值运算操作符

数字类型之间相互运算所生成的结果是“更宽”的类型，基本规则是：

- 整数之间运算，如果数学意义上的结果是小数，结果是浮点数；
- 整数之间运算，如果数学意义上的结果是整数，结果是整数；
- 整数和浮点数混合运算，输出结果是浮点数；
- 整数或浮点数与复数运算，输出结果是复数。

数值运算操作符

操作符是完成运算的一种符号体系

操作符及使用	描述
$x + y$	加, x与y之和
$x - y$	减, x与y之差
$x * y$	乘, x与y之积
x / y	除, x与y之商 10/3结果是3.3333333333333335
$x // y$	整数除, x与y之整数商 10//3结果是3

数值运算操作符

操作符是完成运算的一种符号体系

操作符及使用	描述
$+ x$	x本身
$- y$	x的负值
$x \% y$	余数，模运算 10%3结果是1
$x ** y$	幂运算，x的y次幂， x^y
	当y是小数时，开方运算10**0.5结果是 $\sqrt{10}$

数值运算操作符

二元操作符有对应的增强赋值操作符

增强操作符及使用	描述
x op = y	即 $x = x \text{ op } y$, 其中, op 为二元操作符
	$x += y$ $x -= y$ $x *= y$ $x /= y$ $x //= y$ $x \% = y$ $x ** = y$
	<pre>>>> x = 3.1415 >>> x **= 3 # 与 x = x **3 等价 31.006276662836743</pre>

数值运算函数

一些以函数形式提供的数值运算功能

函数及使用	描述
abs(x)	绝对值, x的绝对值 abs(-10.01) 结果为 10.01
divmod(x,y)	商余, (x//y, x%y), 同时输出商和余数 divmod(10, 3) 结果为 (3, 1)
pow(x, y[, z])	幂余, (x**y)%z, [...]表示参数z可省略 pow(3, pow(3, 99), 10000) 结果为 4587

pow(3, pow(3, 99), 1000) 587
pow(3, pow(3, 99), 100) 87
pow(3, pow(3, 99), 10) 7

数值运算函数


一些以函数形式提供的数值运算功能

函数及使用	描述
<code>round(x[, d])</code>	四舍五入, d是保留小数位数, 默认值为0 <code>round(-10.123, 2)</code> 结果为 -10.12
<code>max(x₁, x₂, ... , x_n)</code>	最大值, 返回x ₁ , x ₂ , ... , x _n 中的最大值, n不限 <code>max(1, 9, 5, 4, 3)</code> 结果为 9
<code>min(x₁, x₂, ... , x_n)</code>	最小值, 返回x ₁ , x ₂ , ... , x _n 中的最小值, n不限 <code>min(1, 9, 5, 4, 3)</code> 结果为 1

数值运算函数

一些以函数形式提供的数值运算功能

函数及使用	描述
<code>int(x)</code>	将x变成整数，舍弃小数部分 <code>int(123.45)</code> 结果为123; <code>int("123")</code> 结果为123
<code>float(x)</code>	将x变成浮点数，增加小数部分 <code>float(12)</code> 结果为12.0; <code>float("1.23")</code> 结果为1.23
<code>complex(x)</code>	将x变成复数，增加虚数部分 <code>complex(4)</code> 结果为 $4 + 0j$



数字类型的转换

- 示例: $\text{complex}(4.5) = 4.5 + 0j$

```
>>> float(4.5+0j)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    float(4.5+0j)
TypeError: can't convert complex to float
>>>
```

小练习

- 计算下表达式

$30-3^{**}2+8//3^{**}2*10$

$3*4^{**}2/8\%5$

$2^{**}2^{**}3$

$1 \text{ or } 0 \text{ and } 0$

$(1 \text{ or } 0) \text{ and } 0$

```
>>> 30-3**2+8//3**2*10
21
>>> 2**2**3
256
>>> (2**2)**3
64
>>> 2**(2**3)
256
>>> 3*4**2/8%5
...
1.0
```

```
>>> 1 or 0 and 0
1
>>> (1 or 0) and 0
0
...

```

Python运算符及其优先级

表 2-4 Python 运算符优先级

运算符	描述
lambda	Lambda 表达式
or	布尔“或”
and	布尔“与”
not x	布尔“非”
in, not in	成员测试
is, is not	同一性测试
<, <=, >, >=, !=, ==	比较
	按位或
^	按位异或
&	按位与
<<, >>	移位
+, -	加法与减法
*, /, %, //	乘法、除法、取余、整数除法

运算符	描述
+x, -x	正负号
~x	按位翻转
**	指数/幂
x.attribute	属性参考
x[index]	下标
x[index:index]	寻址段
f(arguments...)	函数调用
(expression,...)	绑定或元组显示
[expression...]	列表显示
{key:datum,...}	字典显示
'expression...'	字符串转换

运算符和表达式

- **+**运算符除了用于算术加法以外，还可以用于列表、元组、字符串的连接，但不支持不同类型的对象之间相加或连接。

```
>>> [1, 2, 3] + [4, 5, 6]      #连接两个列表
[1, 2, 3, 4, 5, 6]
>>> (1, 2, 3) + (4,)          #连接两个元组
(1, 2, 3, 4)
>>> 'abcd' + '1234'           #连接两个字符串
'abcd1234'
>>> 'A' + 1                    #不支持字符与数字相加，抛出异常
TypeError: Can't convert 'int' object to str implicitly
>>> True + 3                   #Python内部把True当作1处理
4
>>> False + 3                 #把False当作0处理
3
```

运算符和表达式

- *运算符不仅可以用于数值乘法，还可以用于列表、字符串、元组等类型，当列表、字符串或元组等类型变量与整数进行“*”运算时，表示对内容进行重复并返回重复后的新对象。

>>> 2.0 * 3	#浮点数与整数相乘
6.0	
>>> (3+4j) * 2	#复数与整数相乘
(6+8j)	
>>> (3+4j) * (3-4j)	#复数与复数相乘
(25+0j)	
>>> "a" * 10	#字符串重复
'aaaaaaaaaa'	
>>> [1,2,3] * 3	#列表重复
[1, 2, 3, 1, 2, 3, 1, 2, 3]	
>>> (1,2,3) * 3	#元组重复
(1, 2, 3, 1, 2, 3, 1, 2, 3)	

运算符和表达式

- Python中的除法有两种，“/”和“//”分别表示除法和整除运算，并且Python 2.x和Python 3.x对“/”运算符的解释也略有区别。在Python 3.10中运算结果如下：

```
>>> 3/5
```

```
0.6
```

```
>>> 3//5
```

```
0
```

```
>>> 3.0/5
```

```
0.6
```

```
>>> 3.0//5
```

```
0.0
```

```
>>> 13//10
```

```
1
```

```
>>> -13//10
```

```
-2
```

小于当前小数的最大整数

运算符和表达式

- 上面的表达式在Python 2.7.12中运算结果如下：

```
>>> 3/5
0
>>> 3//5
0
>>> 3.0/5
0.6
>>> 3.0//5
0.0
>>> 13//10
1
>>> -13//10
-2
```

```
>>> 3/5
0.6
>>> 3//5
0
>>> 3.0/5
0.6
>>> 3.0//5
0.0
>>> 13//10
1
>>> -13//10
-2
```

运算符和表达式

- %运算符除去可以用于字符串格式化之外，还可以对整数和浮点数计算余数。但是由于浮点数的精确度影响，计算结果可能略有误差。

```
>>> 3.1%2
```

```
1.1
```

```
>>> 6.3%2.1
```

```
2.0999999999999996
```

```
>>> 6%2
```

```
0
```

```
>>> 6.0%2
```

```
0.0
```

```
>>> 6.0%2.0
```

```
0.0
```

```
>>> 5.7%4.8
```

```
0.90000000000000004
```

Python小练习

❖ 例1-1：用户输入一个三位自然数，计算并输出其百位、十位和个位上的数字。

```
x = input('请输入一个三位数：')
x = int(x)
a = x // 100
b = x // 10 % 10
c = x % 10
print(a, b, c)
```

想一想，还有别的办法吗？

```
321
321//100=3
321%100=21
21//10=2
21%10=1
```

```
x=input("请输入一个三位数:")
for i in x:
    print('对应的第{}位数是{}'.format(x.index(i)+1,i))
```

```
请输入一个三位数:456
对应的第1位数是4
对应的第2位数是5
对应的第3位数是6
```

Python小练习

- 还可以这样写

```
x = input('请输入一个三位数: ')\nx = int(x)\na, b = divmod(x, 100)\nb, c = divmod(b, 10)\nprint(a, b, c)
```

函数及使用	描述
abs(x)	绝对值, x的绝对值 abs(-10.01) 结果为 10.01
divmod(x,y)	商余, (x//y, x%y), 同时输出商和余数 divmod(10, 3) 结果为 (3, 1)
pow(x, y[, z])	幂余, (x**y)%z, [...]表示参数z可省略 pow(3, pow(3, 99), 10000) 结果为 4587

运算符和表达式

- 关系运算符可以连用

```
>>> 1 < 3 < 5                #等价于1 < 3 and 3 < 5
```

```
True
```

```
>>> 'Hello' > 'world'        #比较字符串大小
```

```
False
```

```
>>> [1, 2, 3] < [1, 2, 4]    #比较列表大小
```

```
True
```

```
>>> 'Hello' > 3              #字符串和数字不能比较
```

```
TypeError: unorderable types: str() > int()
```

```
>>> {1, 2, 3} < {1, 2, 3, 4}  #测试是否子集
```

字符比较（character comparison）是指按照字典次序对单个字符或字符串进行比较大小的操作，一般都是以ASCII码值的大小作为字符比较的标准。

0	nul	1	soh	2	stx	3	etx	4	eot	5	enq	6	ack	7	bel
8	bs	9	ht	10	nl	11	vt	12	np	13	cr	14	so	15	si
16	dle	17	dc1	18	dc2	19	dc3	20	dc4	21	nak	22	syn	23	etb
24	can	25	em	26	sub	27	esc	28	fs	29	gs	30	rs	31	us
32	sp	33	!	34	"	35	#	36	\$	37	%	38	&	39	'
40	(41)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[92	\	93]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124		125	}	126	~	127	del

```
>>> ord(' ')
32
>>> ord('s')
115
>>> ord('2')
50
>>> ord('1')
49
>>> chr(49)
'1'
>>> chr(32)
' '
```

```
>>> chr(0)
'\x00'
>>> chr(32)
' '
>>> chr(33)
'!'
>>>
```

ASCII 码是一种电脑编码方式，用于将字符（包括字母、数字、标点符号等）转化为数字。每个字符都有一个对应的 ASCII 码，这个 ASCII 码是一个从 0 到 255 的十进制数。

Python小练习

❖ 例1-2: 任意输入三个英文单词, 按字典顺序输出。

```
s = input('x,y,z=')
x, y, z = s.split(',')
if x > y:
    x, y = y, x
if x > z:
    x, z = z, x
if y > z:
    y, z = z, y
print(x, y, z)
```

■ 或直接写为:

```
s = input('x,y,z=')
x, y, z = s.split(',')
x, y, z = sorted([x, y, z])
print(x, y, z)
```

运算符和表达式

- 成员测试运算符in用于成员测试，即测试一个对象是否为另一个对象的元素。

```
>>> 3 in [1, 2, 3]           #测试3是否存在于列表[1, 2, 3]中
```

```
True
```

```
>>> 5 in range(1, 10, 1)     #range()是用来生成指定范围数字的内置函数
```

```
True
```

```
>>> 'abc' in 'abcdefg'      #子字符串测试
```

```
True
```

```
>>> for i in (3, 5, 7):      #循环，成员遍历
    print(i, end='\t')
```

```
3  5  7
```


运算符和表达式

- 同一性测试运算符（**identity comparison**）**is**用来测试两个对象是否是同一个，如果是则返回**True**，否则返回**False**。如果两个对象是同一个，二者具有相同的内存地址。

```
>>> 3 is 3
```

```
True
```

```
>>> x = [300, 300, 300]
```

```
>>> x[0] is x[1]          #基于值的内存管理，同一个值在内存中只有一份
```

```
True
```

```
>>> x = [1, 2, 3]
```

```
>>> y = [1, 2, 3]
```

```
>>> x is y                #上面形式创建的x和y不是同一个列表对象
```

```
False
```

运算符和表达式

- 集合的交集、并集、对称差集等运算借助于位运算符来实现，而差集则使用减号运算符实现（注意，并集运算符不是加号）。

```
>>> {1, 2, 3} | {3, 4, 5}      #并集，自动去除重复元素
{1, 2, 3, 4, 5}
>>> {1, 2, 3} & {3, 4, 5}      #交集
{3}
>>> {1, 2, 3} ^ {3, 4, 5}      #对称差集
{1, 2, 4, 5}
>>> {1, 2, 3} - {3, 4, 5}      #差集
{1, 2}
```

运算符和表达式

- 逻辑运算符and和or具有**惰性求值**特点。

```
>>> 3>5 and a>3          #注意，此时并没有定义变量a
False
>>> 3>5 or a>3           #3>5的值为False，所以需要计算后面表达式
NameError: name 'a' is not defined
>>> 3<5 or a>3           #3<5的值为True，不需要计算后面表达式
True
>>> 3 and 5               #最后一个计算的表达式的值作为整个表达式的值
5
>>> 3 and 5>2
True
>>> 3 not in [1, 2, 3]    #逻辑非运算not
False
>>> 3 is not 5            #not的计算结果只能是True或False之一
True
```

运算符和表达式

- Python 3.5增加了一个新的矩阵相乘运算符@

```
>>> import numpy          #numpy是用于科学计算的Python扩展库
>>> x = numpy.ones(3)      #ones()函数用于生成全1矩阵
>>> m = numpy.eye(3)*3    #eye()函数用于生成单位矩阵
>>> m[0,2] = 5            #设置矩阵指定位置上元素的值
>>> m[2, 0] =3
>>> x @ m                  #矩阵相乘
array([ 6.,  3.,  8.])
```

运算符和表达式

- 逗号并不是运算符，只是一个普通分隔符。

```
>>> 'a' in 'b', 'a'
(False, 'a')
>>> 'a' in ('b', 'a')
True
>>> x = 3, 5
>>> x
(3, 5)
>>> 3 == 3, 5
(True, 5)
>>> x = 3+5, 7
>>> x
(8, 7)
```

运算符和表达式

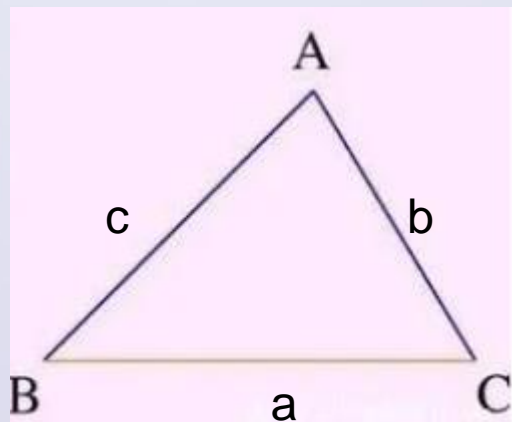
- 在Python中，单个任何类型的对象或常数属于合法表达式，使用运算符连接的变量和常量以及函数调用的任意组合也属于合法的表达式。

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> c
[1, 2, 3, 4, 5, 6]
>>> d = list(map(str, c))
>>> d
['1', '2', '3', '4', '5', '6']
>>> import math
>>> list(map(math.sin, c))
[0.8414709848078965, 0.9092974268256817, 0.1411200080598672, -0.7568024953079282,
 -0.9589242746631385, -0.27941549819892586]
>>> 'Hello' + ' ' + 'world'
'Hello world'
>>> 'welcome' * 3
'welcome welcome welcome'
>>> ('welcome'*3).rstrip(',')+'!'
'welcome, welcome, welcome!'
```

Python小练习

❖ 例1-3：已知三角形的两边长及其夹角，求第三边长。

```
import math
x = input('输入两边长及夹角（度）：')
a, b, theta = map(float, x.split())
c = math.sqrt(a**2 + b**2 - 2*a*b*math.cos(theta*math.pi/180))
print('c=', c)
```



正弦定理

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R$$

余弦定理

$$c^2 = a^2 + b^2 - 2ab \cos C$$

小程序

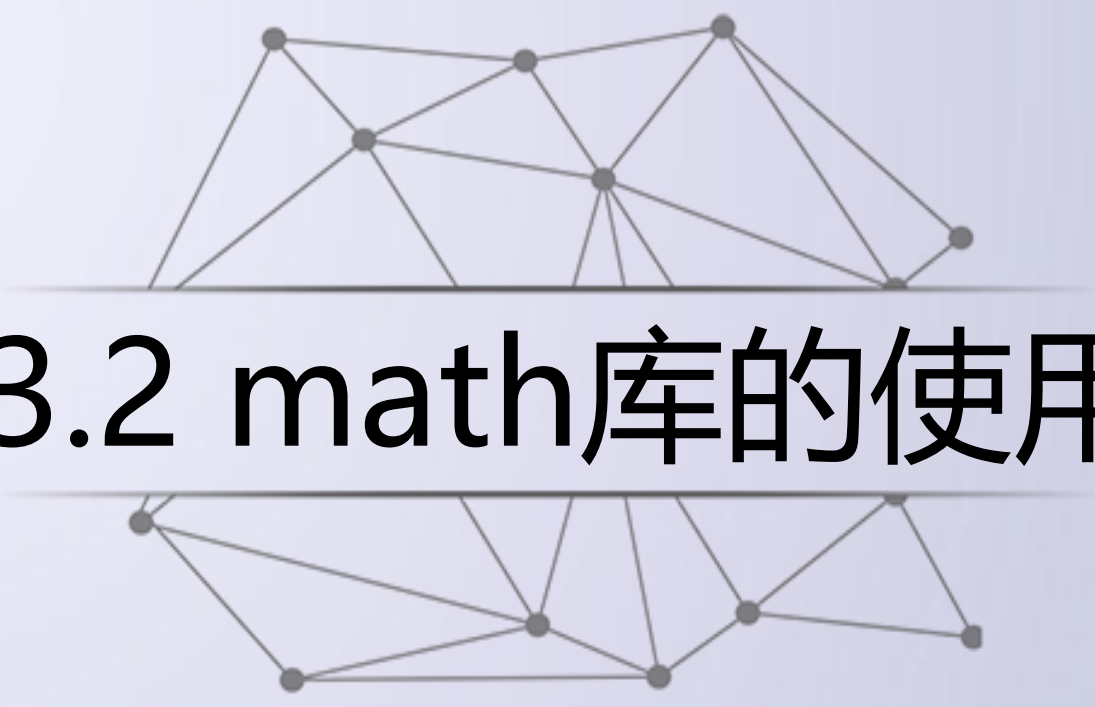
- 提供一个字符串和一个数字即偏移数，把字符串从左向右旋转字符串。(abcd 3 bcda)

```
rotatestr.py - E:/liushuo/rotatestr.py (3.10.7)
File Edit Format Run Options Window Help
def rotatestr(A,moveposition):
    if len(A)>0:
        moveposition = moveposition % len(A)
        newstr = (A+A)[(len(A)-moveposition) : 2*len(A)-moveposition]
        return newstr
x= eval(input("输入: A="))
y= eval(input("输入: moveposition="))
print("偏移之后的字符串是", rotatestr(x,y))
```

```
输入: A="2wewe"
输入: moveposition=3
偏移之后的字符串是 ewe2w
```

```
输入: A="abcd"
输入: moveposition=3
偏移之后的字符串是 bcda
```

a	b	c	d	a	b	c	d
0	1	2	3	4	5	6	7



3.2 math库的使用



math库概述

- math库是Python提供的内置数学类函数库
- math库不支持复数类型
- math库一共提供了4个数学常数和44个函数。
 - 44个函数共分为4类，包括：16个数值表示函数、8个幂对数函数、16个三角对数函数和4个高等特殊函数



math库概述

首先使用保留字import引用该库

- 第一种: import math

对math库中函数采用math.()形式使用

```
>>>import math
>>>math.ceil(10.2)
11
```

- 第二种, from math import <函数名>

对math库中函数可以直接采用<函数名>()形式使用

```
>>>from math import floor
>>>floor(10.2)
10
```



math库解析

■ math库包括4个数学常数

常数 ↵	数学表示 ↵	描述 ↵
<u>math.pi</u> ↵	Π ↵	圆周率，值为 3.141592653589793 ↵
<u>math.e</u> ↵	e ↵	自然对数，值为 2.718281828459045 ↵
<u>math.inf</u> ↵	∞ ↵	正无穷大，负无穷大为 -math.inf ↵
<u>math.nan</u> ↵	↵	非浮点数标记， <u>NaN</u> (Not a Number) ↵

math库解析

math库包括16个数值表示函数

函数	数学表示	描述
<code>math.fabs(x)</code>	$ x $	返回x的绝对值
<code>math.fmod(x, y)</code>	$x\%y$	返回x与y的模
<code>math.fsum([x,y,...])</code>	$x+y+....$	浮点数精确求和
<code>math.ceil(x)</code>		向上取整，返回不小于x的最小整数 <code>math.ceil(3.2)=4</code>
<code>math.floor(x)</code>		向下取整，返回不大于x的最大整数 <code>math.floor(3.2)=3</code>
<code>math.factorial(x)</code>	$X!$	返回x的阶乘，如果x是小数或负数，返回ValueError
<code>math.gcd(a, b)</code>		返回a与b的最大公约数 <code>math.gcd(16,24)=8</code>
<code>math.frexp(x)</code>	$X=m*2^e$	返回(m, e)，当x=0，返回(0.0, 0)
<code>math.ldexp(x, i)</code>	$X*2^i$	返回 $x * 2^i$ 运算值， <code>math.frexp(x)</code> 函数的反运算
<code>math.modf(x)</code>		返回x的小数和整数部分
<code>math.trunc(x)</code>		返回x的整数部分
<code>math.copysign(x, y)</code>	$ x * y /y$	用数值y的正负号替换数值x的正负号
<code>math.isclose(a,b)</code>		比较a和b的相似性，返回True或False
<code>math.isfinite(x)</code>		当x为无穷大，返回True；否则，返回False
<code>math.isinf(x)</code>		当x为正数或负数无穷大，返回True；否则，返回False
<code>math.isnan(x)</code>		当x是NaN，返回True；否则，返回False

• 小程序-计算内积空间两矢量的余弦相似度

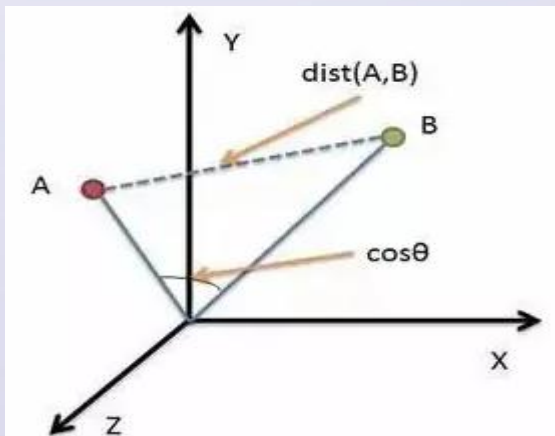
两个向量间的余弦值可以通过使用欧几里得点积公式求出：

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos \theta.$$

给定两个属性向量， A 和 B ，其余弦相似性 θ 由点积和向量长度给出，如下所示：

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}.$$

A B 向量维度不一样，
结果为2；
分母为0，结果也为2



• 余弦相似度代码

```
输入: A=[1, 2, 34]
输入: B=[1, 3, 2, 34]
输出: 2
>>> |
```

```
输入: A=[1, 2, 3, 4]
输入: B=[434, 3, 4, 6]
输出: 0.20021023853018563
```

cosinesim.py - E:/liushuo/cosinesim.py (3.9.5)

File Edit Format Run Options Window Help

```
import math
def cosinesim(A,B):
    if len(A)!=len(B):
        return 2
    n,numerator=len(A),0
    for i in range(n):
        numerator+=A[i]*B[i]
    denominator = sum(a*a for a in A)*sum(b*b for b in B)
    if denominator == 0:
        return 2
    return numerator/math.sqrt(denominator)

x = eval(input("输入: A="))
y = eval(input("输入: B="))
print("输出: ",cosinesim(x,y))
```

输入: A=[1, 2, 34]
输入: B=[2, 3, 45]
输出: 0.9998575624704967



3.3 天天向上的力量



实例代码3.1: 天天向上

基本问题: 持续的价值

一年365天, 以第1天的能力值为基数, 记为1.0, 当好好学习时能力值相比前一天提高1‰, 当没有学习时由于遗忘等原因能力值相比前一天下降1‰。每天努力和每天放任, 一年下来的能力值相差多少呢?

天天向上的力量



问题1： 1‰的力量

—一年365天，每天进步1‰，
累计进步多少呢？

1.001^{365}

—一年365天，每天退步1‰，
累计剩下多少呢？

0.999^{365}

天天向上的力量

问题1： 1‰的力量

```
#DayDayUpQ1.py
import math
dayup=math.pow(1.0+0.001,365)
daydown=math.pow(1.0-0.001,365)
print("向上: {:.2f}, 向下: {:.2f}".format(dayup, daydown))
```

编写上述代码，并保存为DayDayUpQ1.py文件

天天向上的力量

问题1：1‰的力量

>>> (运行结果)

向上：1.44，向下：0.69

$$1.001^{365} = 1.44 \quad 0.999^{365} = 0.69$$

1‰的力量，接近2倍，不可小觑哦

天天向上的力量

问题2： 5‰和1%的力量

—一年365天，每天进步5‰或1%，
累计进步多少呢？

$$1.005^{365}$$

$$1.01^{365}$$

—一年365天，每天退步5‰或1%，
累计剩下多少呢？

$$0.995^{365}$$

$$0.99^{365}$$

天天向上的力量

问题2： 5‰和1%的力量

```
#DayDayUpQ2.py
```

```
Import math
```

```
dayfactor = 0.005
```

使用变量的好处：一处修改即可

```
dayup=math.pow(1+dayfactor, 365)
```

```
daydown=math.pow(1-dayfactor, 365)
```

```
print("向上: {:.2f}, 向下: {:.2f}".format(dayup, daydown))
```

编写上述代码，并保存为DayDayUpQ2.py文件

天天向上的力量

问题2： 5‰和1%的力量

>>> (5‰运行结果)

向上： 6.17, 向下： 0.16

$$1.005^{365} = 6.17$$

$$0.995^{365} = 0.16$$

5‰的力量，惊讶！

>>> (1%运行结果)

向上： 37.78, 向下： 0.03

$$1.01^{365} = 37.78$$

$$0.99^{365} = 0.03$$

1%的力量，惊人！

天天向上的力量

问题3：工作日的力量

- 一年365天，一周5个工作日，每天进步1%
- 一年365天，一周2个休息日，每天退步1%
- 这种工作日的力量，如何呢？

1.01^{365} (数学思维) ➡ for..in.. (计算思维)

天天向上的力量

#DayDayUpQ3.py

采用循环模拟365天的过程

dayup = 1.0

抽象 + 自动化

dayfactor = 0.01

for i **in** range(365):

if i % 7 **in** [6,0]:

 dayup = dayup*(1-dayfactor)

else:

 dayup = dayup*(1+dayfactor)

print(“工作5天休息2天的力量: {:.2f}”.format(dayup))

天天向上的力量

问题3： 工作日的力量

>>> (运行结果)

工作5天休息2天的力量的力量： 4.63

$$1.001^{365} = 1.44 \quad 1.005^{365} = 6.17 \quad 1.01^{365} = 37.78$$

尽管工作日提高1%，但总体效果介于1‰和5‰的力量之间

天天向上的力量

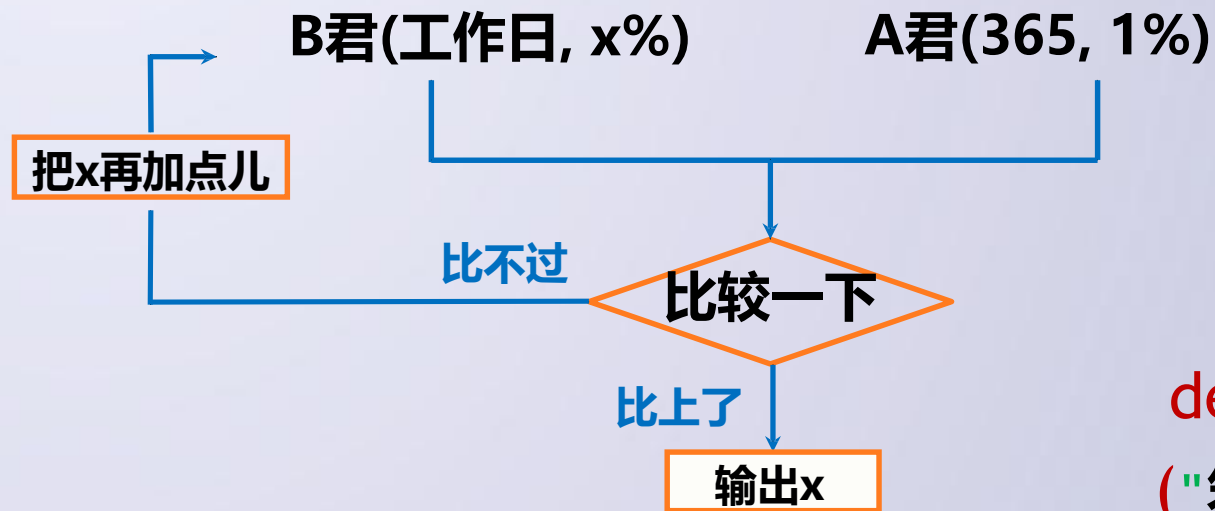
问题4：工作日的努力

- 工作日模式要努力到什么水平，才能与每天努力1%一样？
- A君：一年365天，每天进步1%，不停歇
- B君：一年365天，每周工作5天休息2天，休息日下降1%，要多努力呢？

for..in.. (计算思维) ➡ def..while.. ("笨办法"试错)

天天向上的力量

问题4：工作日的努力



def..while..
("笨办法"试错)

天天向上的力量

```
#DayDayUpQ4.py
```

```
def dayUP(df):
```

```
    dayup = 1
```

```
    for i in range(365):
```

```
        if i % 7 in [6,0]:
```

```
            dayup = dayup*(1 - 0.01)
```

```
        else:
```

```
            dayup = dayup*(1 + df)
```

```
    return dayup
```

```
dayfactor = 0.01
```

```
while dayUP(dayfactor) < 37.78:
```

```
    dayfactor += 0.001
```

```
print("工作日的努力参数是: {:.3f} ".format(dayfactor))
```

根据df参数计算工作日力量的函数

参数不同，这段代码可共用

def保留字用于定义函数

while保留字判断条件是否成立

条件成立时循环执行

天天向上的力量

问题4：工作日的努力

>>> (运行结果)

工作日的努力参数是：0.019

$$1.01^{365} = 37.78$$

$$1.019^{365} = 962.89$$

工作日模式，每天要努力到1.9%，相当于365模式每天1%的效果！


天天向上的力量

GRIT: perseverance and passion for long-term goals

$$1.01^{365} = 37.78$$

$$1.019^{365} = 962.89$$

- **GRIT, 坚毅, 对长期目标的持续激情及持久耐力**
- **GRIT是获得成功最重要的因素之一, 牢记天天向上的力量**



"天天向上的力量"举一反三

举一反三

天天向上的力量

- 实例虽然仅包含8-12行代码，但包含很多语法元素
- 条件循环、计数循环、分支、函数、计算思维
- 清楚理解这些代码能够快速入门Python语言

举一反三

问题的变化和扩展

- 工作日模式中，如果休息日不下降呢？
- 如果努力每天提高1%，休息时每天下降1‰呢？
- 如果工作3天休息1天呢？

字符串类型及操作

字符串类型及操作

- **字符串类型的表示**
- **字符串操作符**
- **字符串处理函数**
- **字符串处理方法**
- **字符串类型的格式化**





字符串类型的表示

字符串

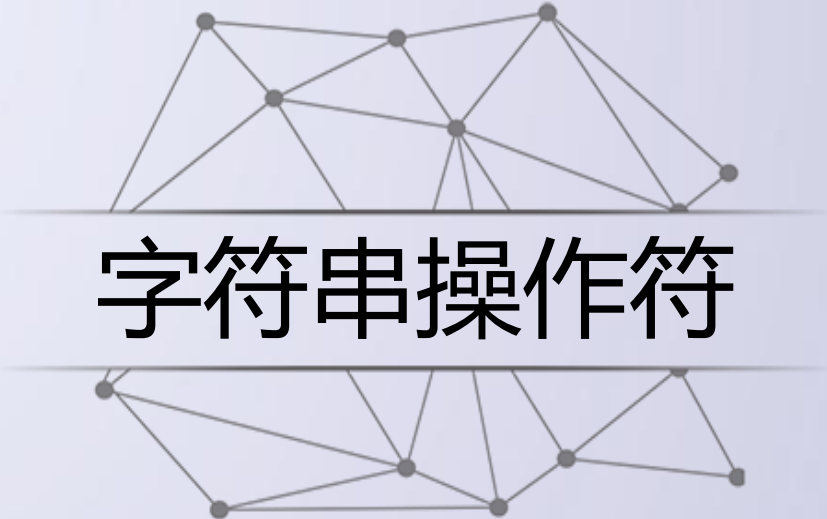
由0个或多个字符组成的有序字符序列

—字符串由一对单引号或一对双引号表示

"请输入带有符号的温度值:"或者'c'

—字符串是字符的有序序列，可以对其中的字符进行索引

"请" 是"请输入带有符号的温度值:" 的第0个字符



字符串操作符

字符串操作符

由0个或多个字符组成的有序字符序列

操作符及使用	描述
$x + y$	连接两个字符串x和y
$n * x$ 或 $x * n$	复制n次字符串x
$x \text{ in } s$	如果x是s的子串，返回True，否则返回False

字符串操作符

获取星期字符串

- **输入：1-7的整数，表示星期几**
- **输出：输入整数对应的星期字符串**
- **例如：输入3，输出 星期三**

字符串操作符

获取星期字符串

```
#WeekNamePrintV1.py
```

```
weekStr = "星期一星期二星期三星期四星期五星期六星期日"
```

```
weekId = eval(input("请输入星期数字(1-7): "))
```

```
pos = (weekId - 1)*3
```

```
print(weekStr[pos: pos+3])
```

字符串操作符

获取星期字符串

```
#WeekNamePrintV2.py
```

```
weekStr = "一二三四五六日"
```

```
weekId = eval(input("请输入星期数字(1-7): "))
```

```
print("星期" + weekStr[weekId-1])
```

字符串


■ 常用转义字符

转义字符	含义	转义字符	含义
\b	退格，把光标移动到前一列位置	\\	一个斜线\
\f	换页符	\a	蜂鸣，响铃
\n	换行符	\0	NULL，啥都不做
\r	回车	\ooo	3位八进制数对应的字符
\t	水平制表符	\xhh	2位十六进制数对应的字符
\v	垂直制表符	\uhhhh	4位十六进制数表示的Unicode字符

小练习：如果我需要在一个字符串中嵌入一个双引号，正确的做法是？

```
>>> print("I like "python".")
SyntaxError: invalid syntax. Perhaps you forgot a comma?
>>> |
```

```
>>> print("I like \"python\".")
...
I like "python".
>>> print('I like "python".')
...
I like "python".
```



字符串处理函数

字符串处理函数

一些以函数形式提供的字符串处理功能

函数及使用	描述
len(x)	长度，返回字符串x的长度 len("一二三456") 结果为 6
str(x)	任意类型x所对应的字符串形式 str(1.23)结果为"1.23" str([1,2])结果为"[1,2]"
hex(x) 或 oct(x)	整数x的十六进制或八进制小写形式字符串 hex(425)结果为"0x1a9" oct(425)结果为"0o651"

字符串处理函数

一些以函数形式提供的字符串处理功能

函数及使用	描述
chr(u)	x为Unicode编码，返回其对应的字符
ord(x)	x为字符，返回其对应的Unicode编码

Unicode

chr(u)



单字符

ord(x)

英文字母对应的Unicode编码


A~Z : 65~90

a~z : 97~122

0~9 : 48~57

```
>>> ord("a")
97
>>> ord("d")
100
>>> ord("A")
65
```

```
>>> chr(97)
'a'
>>> chr(100)
'd'
```



内置的字符串处理函数

操作	含义
+	连接
*	重复
<string>[]	索引
<string>[:]	剪切
len(<string>)	长度
<string>.upper()	字符串中字母大写
<string>.lower()	字符串中字母小写
<string>.strip()	去两边空格及去指定字符
<string>.split()	按指定字符分割字符串为数组
<string>.join()	连接两个字符串序列
<string>.find()	搜索指定字符串
<string>.replace()	字符串替换
for <var> in <string>	字符串迭代

内置的字符串处理函数

微实例3.2：恺撒密码。

凯撒密码是古罗马凯撒大帝用来对军事情报进行加密的算法，它采用了替换方法对信息中的每一个英文字符循环替换为字母表序列该字符后面第三个字符，对应关系如下：

原文：A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

密文：D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

原文字符P，其密文字符C满足如下条件：

$$C = (P + 3) \bmod 26$$

解密方法反之，满足：

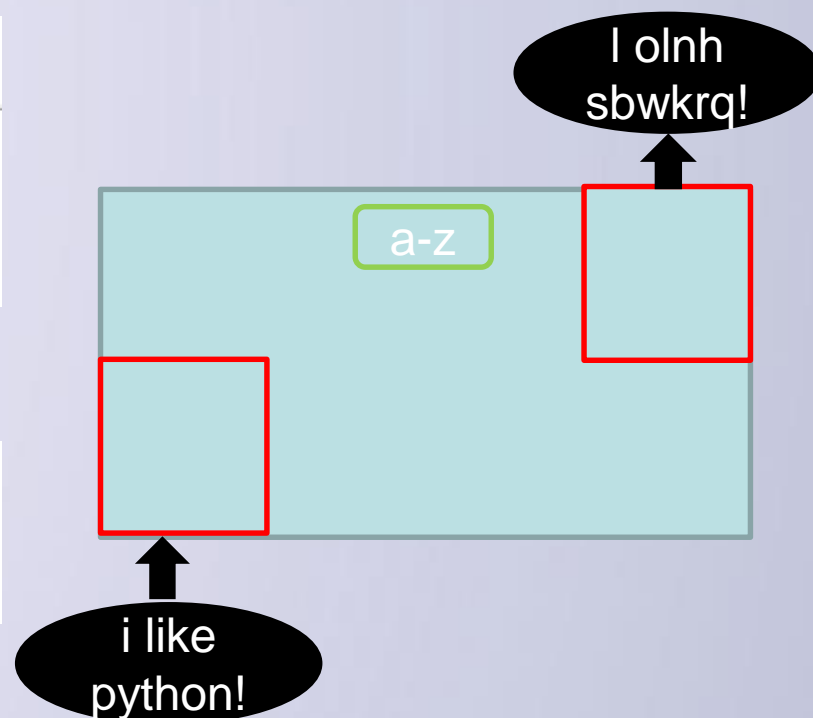
$$P = (C - 3) \bmod 26$$

明文的字母由其它字母或数字或符号代替

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

```
CaesarCode.py - E:\liushuo\CaesarCode.py (3.10.7)
File Edit Format Run Options Window Help
plaincode = input("请输入明文: ")
for p in plaincode:
    if ord("a") <= ord(p) <= ord("z"):
        print(chr(ord("a") + (ord(p)-ord("a") + 3)%26), end="")
    else:
        print(p, end="")
```

请输入明文: i like python!
l olnh sbwkrq!



Unicode编码

Python字符串的编码方式

- 统一字符编码，即覆盖几乎所有字符的编码方式
- 从0到1114111 (0x10FFFF)空间，每个编码对应一个字符
- Python字符串中每个字符都是Unicode编码字符

Unicode编码

一些有趣的例子

```
>>> "1" + "1" = "2" + chr(10004)
```

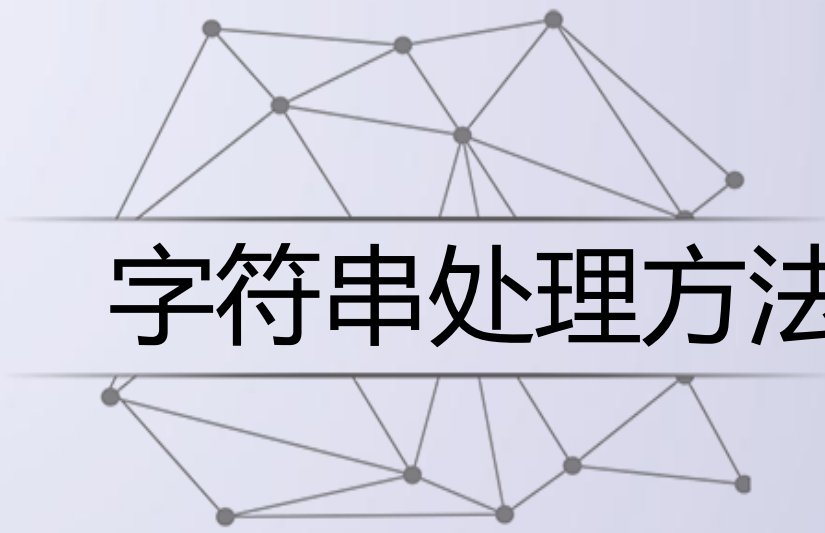
```
'1 + 1 = 2 ✓'
```

```
>>> "这个字符𐄀的Unicode值是: " +  
str(ord("𐄀"))
```

```
'这个字符𐄀的Unicode值是: 9801'
```

```
>>> for i in range(12):  
    print(chr(9800 + i), end="")
```

𐄀𐄁𐄂𐄃𐄄𐄅𐄆𐄇𐄈𐄉𐄊𐄋



字符串处理方法

字符串处理方法

"方法"在编程中是一个专有名词

"方法"特指<a>.()风格中的函数()

方法本身也是函数，但与<a>有关，<a>.()风格使用

字符串或字符串变量是<a>，存在一些可用方法

字符串处理方法

一些以方法形式提供的字符串处理功能

方法及使用 1/3	描述
<code>str.lower()</code> 或 <code>str.upper()</code>	返回字符串的副本，全部字符小写/大写 <code>"AbCdEfGh".lower()</code> 结果为 <code>"abcdefgh"</code>
<code>str.split(sep=None)</code>	返回一个列表，由str根据sep被分隔的部分组成 <code>"A,B,C".split(",")</code> 结果为 <code>['A', 'B', 'C']</code>
<code>str.count(sub)</code>	返回子串sub在str中出现的次数 <code>"an apple a day".count("a")</code> 结果为 4

```
>>> "A,B,C".split(",")
['A', 'B', 'C']
>>> "A,B,C".split()
['A,B,C']
>>> "A B C".split()
['A', 'B', 'C']
>>>
```

字符串处理方法

一些以方法形式提供的字符串处理功能

方法及使用 2/3	描述
<code>str.replace(old, new)</code>	返回字符串str副本，所有old子串被替换为new <code>"python".replace("n","n123.io")</code> 结果为 <code>"python123.io"</code>
<code>str.center(width[,fillchar])</code>	字符串str根据宽度width居中，fillchar可选 <code>"python".center(20,"=")</code> 结果为 <code>'=====python====='</code>

```
>>> "python".center(20,"=")
'=====python====='
>>> "python".center(20)
python
... 
```


字符串处理方法

一些以方法形式提供的字符串处理功能

方法及使用 3/3	描述
<code>str.strip(chars)</code>	从str中去掉在其左侧和右侧chars中列出的字符 <code>"= python= ".strip(" =np")</code> 结果为 <code>"ytho"</code>
<code>str.join(iter)</code>	在iter变量除最后元素外每个元素后增加一个str <code>",".join("12345")</code> 结果为 <code>"1,2,3,4,5"</code> #主要用于字符串分隔等

```
>>> "123".join('wr')
'w123r'
```

内置的字符串处理方法

```
>>> "A,B,C".split(",",maxsplit=2)
['A', 'B', 'C']
>>> "A,B,C".split(",",maxsplit=1)
['A', 'B,C']
>>> "A,B,C,E,F,G".split(",",maxsplit=-1)
['A', 'B', 'C', 'E', 'F', 'G']
>>> "A,B,C,E,F,G".split(",",maxsplit=3)
['A', 'B', 'C', 'E,F,G']
```

方法	描述
str.lower()	返回字符串str的副本，全部字符小写
str.upper()	返回字符串str的副本，全部字符大写
str.islower()	当str所有字符都是小写时，返回True，否则False
str.isprintable()	当str所有字符都是可打印的，返回True，否则False
str.isnumeric()	当str所有字符都是字符时，返回True，否则False
str.isspace()	当str所有字符都是空格，返回True，否则False
str.endswith(suffix[,start[,end]])	str[start: end] 以suffix结尾返回True，否则返回False
str.startswith(prefix[, start[, end]])	str[start: end] 以suffix开始返回True，否则返回False
str.split(sep=None, maxsplit=-1)	返回一个列表，由str根据sep被分割的部分构成
str.count(sub[,start[,end]])	返回str[start: end]中sub子串出现的次数
str.replace(old, new[, count])	返回字符串str的副本，所有old子串被替换为new，如果count给出，则前count次old出现被替换
str.center(width[, fillchar])	字符串居中函数，详见函数定义
str.strip([chars])	返回字符串str的副本，在其左侧和右侧去掉chars中列出的字符
str.zfill(width)	返回字符串str的副本，长度为width，不足部分在左侧添0
str.format()	返回字符串str的一种排版格式，3.6节将详细介绍
str.join(iterable)	返回一个新字符串，由组合数据类型（见第6章）iterable变量的每个元素组成，元素间用str分割

```
>>> "I like python!".split(maxsplit=1)
['I', 'like python!']
>>> "I like python since it is very powerful!".split(maxsplit=3)
['I', 'like', 'python', 'since it is very powerful!']
>>> "I like python since it is very powerful!".split(maxsplit=-1)
['I', 'like', 'python', 'since', 'it', 'is', 'very', 'powerful!']
>>> "I like python since it is very powerful!".split()
['I', 'like', 'python', 'since', 'it', 'is', 'very', 'powerful!']
>>>
```

```
>>> "python".center(40,"=")
'=====python=====
>>> "123".zfill(20)
'000000000000000000123'
```

- 小程序

思考如何写一个函数实现转换小写字母为大写。

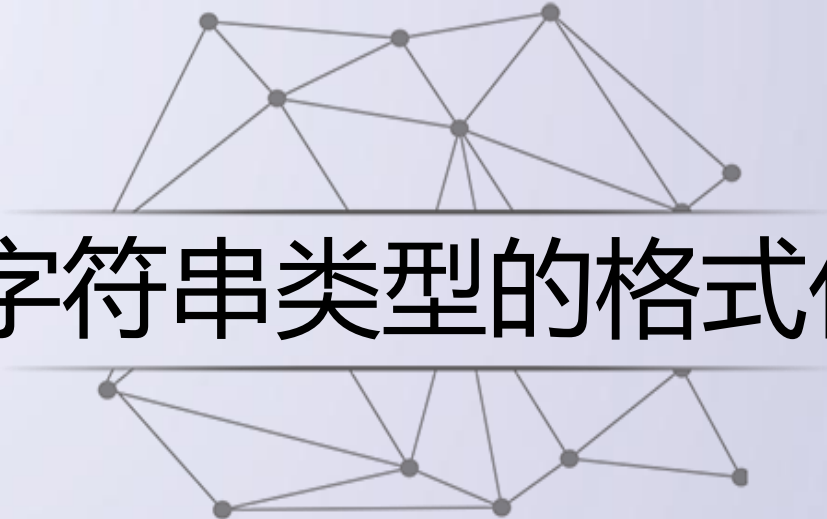
英文字母对应的Unicode编码

A~Z : 65~90

a~z : 97~122

0~9 : 48~57

```
>>> def lower(a):  
        return chr(ord(a)-32)  
  
>>> lower('a')  
'A'  
>>> lower('b')  
'B'  
>>> print(chr(ord('c')-32))  
C
```



字符串类型的格式化

字符串

- 字符串合并

```
>>> a = 'abc' + '123'      #生成新对象
```

- 字符串格式化

```
>>> a = 3.6674
```

```
>>> '%7.3f' % a
```

```
' 3.667'
```

```
>>> "%d:%c"%(65,65)
```

```
'65:A'
```

```
>>> """My name is %s, and my age is %d""" % ('python',33)
```

```
'My name is python, and my age is 33'
```

字符串

- 字符串界定符前面加字母r表示原始字符串，其中的特殊字符不进行转义，但字符串的最后一个字符不能是\。原始字符串主要用于正则表达式、文件路径或者URL的场合。

```
>>> path = 'C:\Windows\notepad.exe'
```

```
>>> print(path)          #字符\n被转义为换行符
```

```
C:\Windows
```

```
otepad.exe
```

```
>>> path = r'C:\Windows\notepad.exe' #原始字符串，任何字符都不转义
```

```
>>> print(path)
```

```
C:\Windows\notepad.exe
```

```
>>> path = 'C:\Windows\notepad.exe'
>>> print(path)
C:\Windows\notepad.exe
```

字符串类型的格式化

格式化是对字符串进行格式表达的方式

- **字符串格式化使用.format()方法，用法如下：**

<模板字符串>.format(<逗号分隔的参数>)

字符串类型的格式化

槽

"{ } : 计算机{ } 的CPU占用率为{ }%".format("2018-10-10", "C", 10)



0



1



2



0



1



2

字符串中槽{}的默认顺序

format()中参数的顺序

槽的内部样式: {<参数序号>:<格式控制标记>}

字符串类型的格式化

槽

"{1}:计算机{0}的CPU占用率为{2}%" .format("2018-10-10", "C", 10)

```
graph TD; S["\"{1}:计算机{0}的CPU占用率为{2}%\" .format(\"2018-10-10\", \"C\", 10)"]; S --> A["\"2018-10-10\""]; S --> B["\"C\""]; S --> C["10"];
```

The diagram illustrates the mapping of format specifiers to arguments in a Python string formatting statement. A blue line connects the opening quote of the format string to the first argument, "2018-10-10". Another blue line connects the {0} specifier to the second argument, "C". A third blue line connects the {1} specifier to the third argument, 10. A fourth blue line connects the {2} specifier to the closing quote of the format string.

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 符号	用于填 充的单 个字符	< 左对齐 > 右对齐 ^ 居中对齐	槽的设定输 出宽度	数字的千位 分隔符 适用于整数 和浮点数	浮点数小数 精度 或 字 符串最大输 出长度	整数类型 b, c, d, o, x, X 浮点数类型 e, E, f, %

format()方法的格式控制


:	<填充>	<对齐>	<宽度>	<,>	<.精度>	<类型>
<pre>S= "PYTHON" >>> "{0:30}".format(S) 'PYTHON 默认左对齐</pre>				<pre>>>> "{0:=^20}".format("PYTHON") '=====PYTHON====='</pre>		
				<pre>>>> "{0:*>20}".format("BIT") '*****BIT'</pre>		
				<pre>>>> "{:10}".format("BIT") 'BIT '</pre>		

format()方法的格式控制

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

```
>>> bin(425)
'0b110101001'
>>> chr(425)
'Σ'
>>> oct(425)
'0o651'
>>> hex(425)
'0x1a9'
```

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



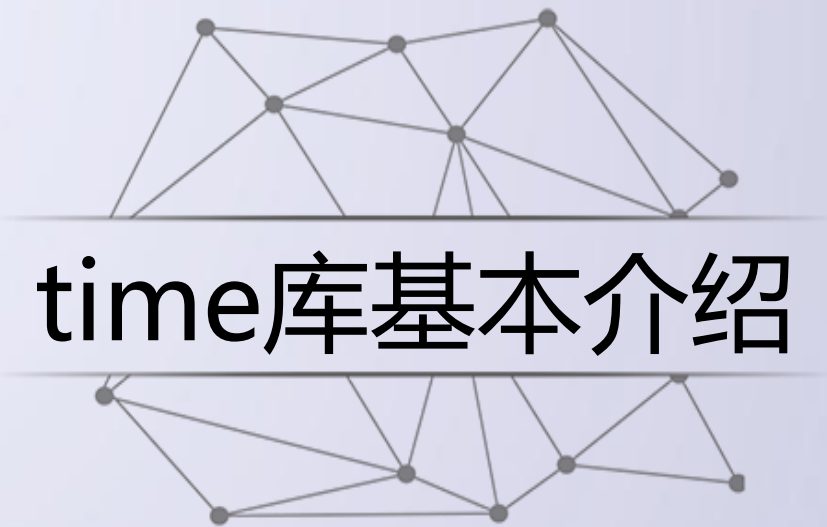
单元小结

字符串类型及操作

- 正向递增序号、反向递减序号、<字符串>[M:N:K]
- +、*、len()、str()、hex()、oct()、ord()、chr()
- .lower()、.upper()、.split()、.count()、.replace()
- .center()、.strip()、.join() 、.format()格式化



3.4 time库的使用



time库基本介绍

time库概述

time库是Python中处理时间的标准库

- 计算机时间的表达

```
import time
```

- 提供获取系统时间并格式化输出功能

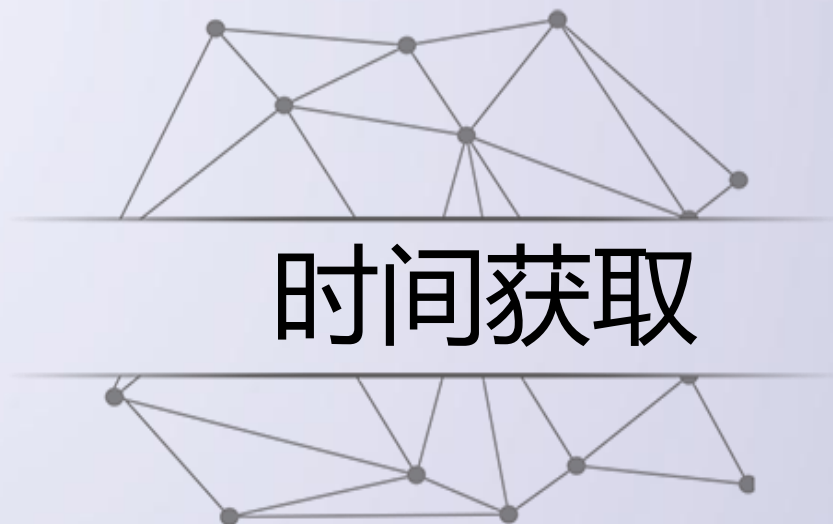
```
time.<b>()
```

- 提供系统级精确计时功能，用于程序性能分析

time库概述

time库包括三类函数

- 时间获取: `time()`
`ctime()`
`gmtime()`
- 时间格式化: `strftime()`
`strptime()`
- 程序计时: `sleep()`,
`perf_counter()`



时间获取

时间获取

函数	描述
time()	<p>获取当前时间戳，即计算机内部时间值，浮点数</p> <pre>>>>time.time()</pre> <pre>1683621906.4174783</pre> <p>How many seconds from 1970.Jan.1st.00: 00 to now</p>
ctime()	<p>获取当前时间并以易读方式表示，返回字符串</p> <pre>>>>time.ctime()</pre> <pre>'Tue May 9 16:45:14 2023'</pre>

时间获取

函数	描述
gmtime()	<p>获取当前时间，表示为计算机可处理的时间格式</p> <pre>>>>time.gmtime() time.struct_time(tm_year=2023, tm_mon=5, tm_mday=9, tm_hour=8, tm_min=46, tm_sec=59, tm_wday=1, tm_yday=129, tm_isdst=0)</pre>



时间格式化

时间格式化

将时间以合理的方式展示出来

- 格式化：类似字符串格式化，需要有展示模板
- 展示模板由特定的格式化控制符组成
 - `strftime()`方法

时间格式化

函数	描述
strftime(tpl, ts)	<p>tpl是格式化模板字符串，用来定义输出效果</p> <p>ts是计算机内部时间类型变量</p> <pre>>>>t = time.gmtime() >>>time.strftime("%Y-%m-%d %H:%M:%S",t) '2023-05-09 08:47:50'</pre>

```
>>> import time
>>> t=time.gmtime()
>>> t
time.struct_time(tm_year=2023, tm_mon=5, tm_mday=9, tm_hour=8, tm_min=47, tm_sec=50, tm_wday=1, tm_yday=129, tm_isdst=0)
>>> time.strftime("%Y-%m-%d %H:%M:%S",t)
'2023-05-09 08:47:50'
```


格式化控制符

格式化字符串	日期/时间说明	值范围和实例
%Y	年份	0000~9999, 例如: 2023
%m	月份	01~12, 例如: 05
%B	月份名称	January~December, 例如: May
%b	月份名称缩写	Jan~Dec, 例如: Oct
%d	日期	01~31, 例如: 10
%A	星期	Monday~Sunday, 例如: Wednesday

格式化控制符

格式化字符串	日期/时间说明	值范围和实例
%a	星期缩写	Mon~Sun, 例如: Wed
%H	小时 (24h制)	00~23, 例如: 12
%h	小时 (12h制)	01~12, 例如: 7
%p	上/下午	AM, PM, 例如: PM
%M	分钟	00~59, 例如: 26
%S	秒	00~59, 例如: 26

时间格式化

```
>>>t = time.gmtime()
```

```
>>>time.strftime("%Y-%m-%d %H:%M:%S",t)
```


'2023-05-09 12:50:21'


>>>timeStr = '2023-05-09 12:50:21'

```
>>>time.strptime(timeStr, "%Y-%m-%d %H:%M:%S")
```

一对逆运算

时间格式化

函数	描述
strptime(str, tpl)	<p>str是字符串形式的时间值</p> <p>tpl是格式化模板字符串，用来定义输入效果</p> <pre>>>>timeStr = '2023-05-09 12:50:21' >>>time.strptime(timeStr, "%Y-%m-%d %H:%M:%S") time.struct_time(tm_year=2023, tm_mon=5, tm_mday=9, tm_hour=12, tm_min=50, tm_sec=21, tm_wday=1, tm_yday=129, tm_isdst=-1)</pre>



程序计时应用

程序计时

程序计时应用广泛

—程序计时指测量起止动作所经历时间的过程

—测量时间：
`perf_counter()`

—产生时间：`sleep()`

程序计时

函数	描述
	<p>返回一个CPU级别的精确时间计数值，单位为秒</p> <p>由于这个计数值起点不确定，连续调用差值才有意义</p> <pre>>>>start = time.perf_counter()</pre>
perf_counter()	<pre>283327.5088477</pre> <pre>>>>end = time.perf_counter()</pre>
	<pre>283370.58136</pre> <pre>>>>end - start</pre> <pre>43.072512299986556</pre>

程序计时

函数	描述
sleep(s)	<p>s拟休眠的时间，单位是秒，可以是浮点数</p> <pre>>>>def wait(): time.sleep(5.5) >>>wait() #程序将等待5.5秒后再退出</pre>

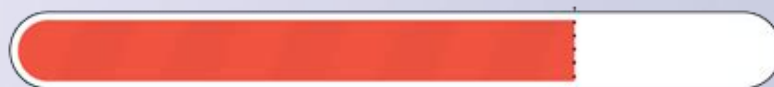
3.5 文本进度条



"文本进度条"问题分析

文本进度条

用过计算机的都见过



75%

- 进度条什么原理呢?



需求分析

文本进度条

- 采用字符串方式打印可以动态变化的文本进度条
- 进度条需要能在一行中逐渐变化

问题分析

如何获得文本进度条的变化时间？

- 采用sleep()模拟一个持续的进度**
- 似乎不那么难**



"文本进度条"简单的开始

简单的开始

```
import time
scale = 10

print("-----执行开始-----")
for i in range(scale+1):
    a = '*' * i
    b = '.' * (scale - i)
    c = (i/scale)*100
    print("{:^3.0f}%[{}->{}]".format(c,a,b))
time.sleep(0.1)
print("-----执行结束-----")
```

```
-----执行开始-----
 0 %[->.....]
10 %[*->.....]
20 %[*->.....]
30 %[*->.....]
40 %[*->.....]
50 %[*->.....]
60 %[*->.....]
70 %[*->.....]
80 %[*->.....]
90 %[*->.....]
100%[*->.....]
-----执行结束-----
```



"文本进度条"单行动态刷新



单行动态刷新

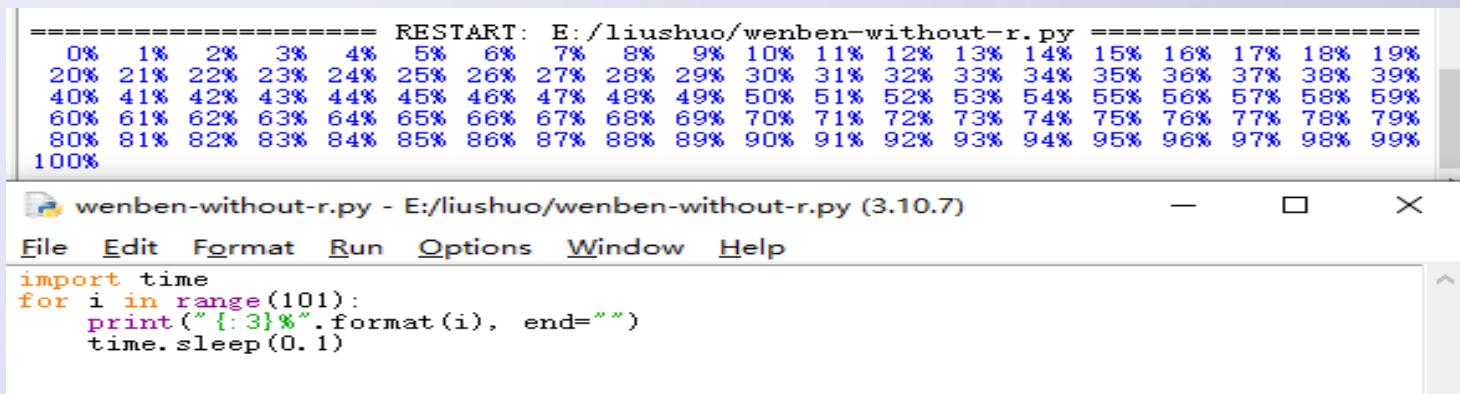
刷新的关键是 `\r`

- 刷新的本质是：用之后打印的字符覆盖之前的字符
- 不能换行： `print()` 需要被控制
- 要能回退：打印后光标退回到之前的位置 `\r`

单行动态刷新

```
import time
for i in range(101):
    print("\r{:3}%".format(i), end="")
    time.sleep(0.1)
```

```
0% 1% 2% 3% 4% 5% 6% 7% 8% 9% 10% 11% 12% 13% 14% 15% 16% 17% 18% 19%
20% 21% 22% 23% 24% 25% 26% 27% 28% 29% 30% 31% 32% 33% 34% 35% 36% 37% 38% 39%
40% 41% 42% 43% 44% 45% 46% 47% 48% 49% 50% 51% 52% 53% 54% 55% 56% 57% 58% 59%
60% 61% 62% 63% 64% 65% 66% 67% 68% 69% 70% 71% 72% 73% 74% 75% 76% 77% 78% 79%
80% 81% 82% 83% 84% 85% 86% 87% 88% 89% 90% 91% 92% 93% 94% 95% 96% 97% 98% 99%
```



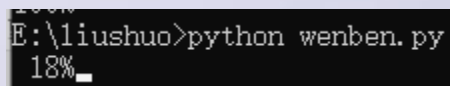
The screenshot shows a Python IDLE window titled "wenben-without-r.py - E:/liushuo/wenben-without-r.py (3.10.7)". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code editor contains the following Python code:

```
import time
for i in range(101):
    print("\r{:3}%".format(i), end="")
    time.sleep(0.1)
```

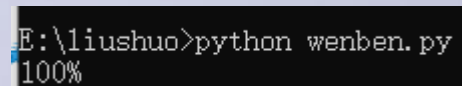
The output window at the bottom shows the execution progress, with the text "100%" displayed on a single line, indicating that the script has completed its execution.

IDLE屏蔽了\r功能

使用cmd命令行



```
E:\liushuo>python wenben.py
18%
```



```
E:\liushuo>python wenben.py
100%
```



"文本进度条"实例完整效果

完整效果

```
import time
scale = 50

print("执行开始".center(scale//2, "-"))
start = time.perf_counter()
for i in range(scale+1):
    a = '*' * i
    b = '.' * (scale - i)
    c = (i/scale)*100
dur = time.perf_counter() - start
print("\r{:^3.0f}%[{}->{}][:.2f]s".format(c,a,b,dur),end=' ')
time.sleep(0.1)
print("\n"+"执行结束".center(scale//2, '-'))
```

```
E:\liushuo>python wenbenjindutiao.py
-----执行开始-----
98 %[*****->.]5.41s
```

```
E:\liushuo>python wenbenjindutiao.py
-----执行开始-----
100%[*****->]5.52s
-----执行结束-----
```



"文本进度条"举一反三

#TextProBarV3.py

import time

scale = 50

print("执行开始".center(scale//2, "-"))

start = time.perf_counter()

for i **in** range(scale+1):

 a = '*' * i

 b = '.' * (scale - i)

 c = (i/scale)*100

 dur = time.perf_counter() - start

print("\\r{:^3.0f}%[{}->{}]{:.2f}s".format(c,a,b,dur),end='')

 time.sleep(0.1)

print("\\n"+"执行结束".center(scale//2, '-'))

```
E:\\liushuo>python TextProBarV3.py
```

```
-----执行开始-----
```

```
100%[*****->]5.43s
```

```
-----执行结束-----
```

#TextProBarV4.py

```
import time
scale = 50
print("执行开始".center(scale//2, "-"))
start = time.perf_counter()
for i in range(scale+1):
    a = '*' * i
    b = '.' * (scale - i)
    c = (i/scale)*100
    dur = time.perf_counter() - start
    print( " {:^3.0f}%[{}->{}][{: .2f}s" .format(c,a,b,dur),end= '\n')
    time.sleep(0.1)
print("\n"+"执行结束".center(scale//2, '-'))
```

```
-----执行开始-----
0 %[->.....]0.00s
2 %[*->.....]0.45s
4 %[*->.....]0.76s
6 %[*->.....]1.06s
8 %[*->.....]1.38s
10 %[*->.....]1.69s
12 %[*->.....]1.99s
14 %[*->.....]2.30s
16 %[*->.....]2.58s
18 %[*->.....]2.86s
20 %[*->.....]3.16s
22 %[*->.....]3.46s
24 %[*->.....]3.76s
26 %[*->.....]4.07s
28 %[*->.....]4.37s
30 %[*->.....]4.68s
32 %[*->.....]5.00s
34 %[*->.....]5.31s
36 %[*->.....]5.63s
38 %[*->.....]5.94s
40 %[*->.....]6.26s
42 %[*->.....]6.53s
44 %[*->.....]6.78s
46 %[*->.....]7.03s
48 %[*->.....]7.28s
50 %[*->.....]7.54s
52 %[*->.....]7.79s
54 %[*->.....]8.02s
56 %[*->.....]8.28s
58 %[*->.....]8.52s
60 %[*->.....]8.79s
62 %[*->.....]9.02s
64 %[*->.....]9.27s
66 %[*->.....]9.53s
68 %[*->.....]9.78s
70 %[*->.....]10.04s
72 %[*->.....]10.27s
74 %[*->.....]10.52s
76 %[*->.....]10.77s
78 %[*->.....]11.02s
80 %[*->.....]11.27s
82 %[*->.....]11.52s
84 %[*->.....]11.77s
86 %[*->.....]12.02s
88 %[*->.....]12.31s
90 %[*->.....]12.60s
92 %[*->.....]12.90s
94 %[*->.....]13.21s
96 %[*->.....]13.55s
98 %[*->.....]13.87s
100%[*->.....]14.22s
-----执行结束-----
```

删除\r，换为\n

举一反三

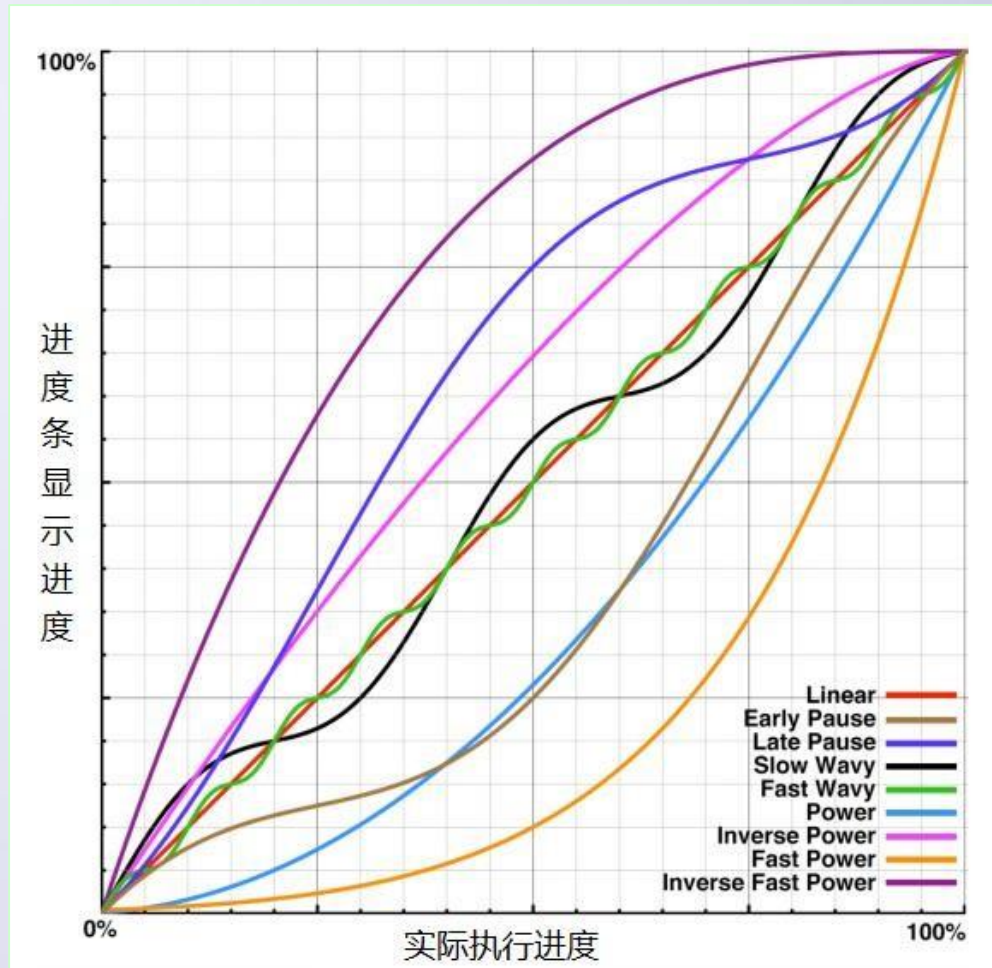
计算问题扩展

- 文本进度条程序使用了perf_counter()计时
- 计时方法适合各类需要统计时间的计算问题
- 例如：比较不同算法时间、统计程序运行时间

举一反三

进度条应用

- 在任何运行时间需要较长的程序中增加进度条
- 在任何希望提高用户体验的应用中增加进度条
- 进度条是人机交互的纽带之一



Harrison C. et al. Rethinking the Progress Bar. In ACM Symposium on User Interface Software and Technology, 2007

举一反三

文本进度条的不同设计函数

设计名称	趋势	设计函数
Linear	Constant	$f(x) = x$
Early Pause	Speeds up	$f(x) = x + (1 - \sin(x * \pi * 2 + \pi / 2)) / -8$
Late Pause	Slows down	$f(x) = x + (1 - \sin(x * \pi * 2 + \pi / 2)) / 8$
Slow Wavy	Constant	$f(x) = x + \sin(x * \pi * 5) / 20$
Fast Wavy	Constant	$f(x) = x + \sin(x * \pi * 20) / 80$

举一反三

文本进度条的不同设计函数

设计名称	趋势	设计函数
Power	Speeds up	$f(x) = (x + (1-x) * 0.03)^2$
Inverse Power	Slows down	$f(x) = 1 + (1-x)^{1.5} * -1$
Fast Power	Speeds up	$f(x) = (x + (1-x)/2)^8$
Inverse Fast Power	Slows down	$f(x) = 1 + (1-x)^3 * -1$

课堂测试

- 下列表达式错误的是（）

A. 'abcd' < 'ad' B. '' < 'a' C. 'abc' < 'abcd'
D. 'Hello' > 'hello'

- 请写出以下运算的结果

```
print(3**2**2)
```

```
print(30-2*3**2)
```

- 第二次平时作业

当从键盘输入一个任意位数的数字时，请编写程序判断这个数字是不是回文偶数。