02.万恶之源-运算符和编码

一. 格式化输出

现在有以下需求,让用户输入name, age, job,hobby 然后输出如下所示:

```
------------ info of Alex Li -----------
Name  : Alex Li
Age   : 22
job   : Teacher
Hobbie: girl
------------- end -----------------
```

你怎么实现呢? 你会发现，用字符拼接的方式还难实现这种格式的输出，所以一起来学一下新姿势
只需要把要打印的格式先准备好, 由于里面的 一些信息是需要用户输入的，你没办法预设知道，因此可以先放置个占位符，再把字符串里的占位符与外部的变量做个映射关系就好啦

```
name = input("Name:")
age = input("Age:")
job = input("Job:")
hobby = input("Hobbie:")

info = '''
------------ info of %s ----------- #这里的每个%s就是一个占位符，本行的代表 后面拓号里的 name
Name  : %s  #代表 name
Age   : %s  #代表 age
job   : %s  #代表 job
Hobbie: %s  #代表 hobbie
------------- end -----------------
''' % (name,name,age,job,hobbie)  # 这行的 % 号就是 把前面的字符串 与拓号 后面的 变量 关联起来

print(info)
```

%s就是代表字符串占位符，除此之外，还有%d, 是数字占位符， 如果把上面的age后面的换成%d，就代表你必须只能输入数字啦
这时对应的数据必须是int类型. 否则程序会报错
使用时,需要进行类型转换.

```
int(str)    # 字符串转换成int
str(int)    # int转换成字符串
```

类似这样的操作在后面还有很多
如果, 你头铁. 就不想转换. 觉着转换很麻烦. 也可以全部都用%s. 因为任何东西都可以直接转换成字符串--> 仅限%s
现在又来新问题了. 如果想输出:

```
我叫xxx，今年xx岁了,我们已经学习了2%的python基础了
```

这里的问题出在哪里呢? 没错2%, 在字符串中如果使用了%s这样的占位符. 那么所有的%都将变成占位符. 我们的2%也变成了占位符. 而"%的"是不存在的, 这里我们需要使用%%来表示字符串中的%.

注意: 如果你的字符串中没有使用过%s,%d站位. 那么不需要考虑这么多. 该%就%.没毛病老铁.

```
print("我叫%s，今年22岁了，学习python2%%了" % '王尼玛')     # 有%占位符
print("我叫王尼玛，今年22岁，已经凉凉了100%了")       # 没有占位符
```

二. 基本运算符

计算机可以进行的运算有很多种，可不只加减乘除这么简单，运算按种类可分为:

算数运算、
比较运算、

逻辑运算、
赋值运算、
成员运算、
身份运算、
位运算.

今天我们暂只学习算数运算、比较运算、逻辑运算、赋值运算

2.1 算数运算

以下假设变量：a=10，b=20

运算符	描述	实例
+	加 - 两个对象相加	a + b 输出结果 30
-	减 - 得到负数或是一个数减去另一个数	a - b 输出结果 -10
*	乘 - 两个数相乘或是返回一个被重复若干次的字符串	a * b 输出结果 200
/	除 - x除以y	b / a 输出结果 2
%	取模 - 返回除法的余数	b % a 输出结果 0
**	幂 - 返回x的y次幂	a**b 为10的20次方， 输出结果 100000000000000000000
//	取整除 - 返回商的整数部分	9//2 输出结果 4 , 9.0//2.0 输出结果 4.0

2.2 比较运算

以下假设变量：a=10，b=20

运算符	描述	实例
==	等于 - 比较对象是否相等	(a == b) 返回 False。
!=	不等于 - 比较两个对象是否不相等	(a != b) 返回 true.
<>	不等于 - 比较两个对象是否不相等	(a <> b) 返回 true。这个运算符类似 != 。
>	大于 - 返回x是否大于y	(a > b) 返回 False。
<	小于 - 返回x是否小于y。所有比较运算符返回1表示真，返回0表示假。这分别与特殊的变量True和False等价。注意，这些变量名的大写。	(a < b) 返回 true。
>=	大于等于 - 返回x是否大于等于y。	(a >= b) 返回 False。
<=	小于等于 - 返回x是否小于等于y。	(a <= b) 返回 true。

赋值运算

以下假设变量：a=10，b=20

运算符	描述	实例
=	简单的赋值运算符	c = a + b 将 a + b 的运算结果赋值为 c
+=	加法赋值运算符	c += a 等效于 c = c + a
-=	减法赋值运算符	c -= a 等效于 c = c - a
*=	乘法赋值运算符	c *= a 等效于 c = c * a
/=	除法赋值运算符	c /= a 等效于 c = c / a
%=	取模赋值运算符	c %= a 等效于 c = c % a
**=	幂赋值运算符	c **= a 等效于 c = c ** a
//=	取整除赋值运算符	c //= a 等效于 c = c // a

算逻辑运

运算符	描述	实例
and	布尔"与" - 如果x为False，x and y返回False，否则它返回y的计算值。	(a and b) 返回 true。
or	布尔"或" - 如果x是True，它返回True，否则它返回y的计算值。	(a or b) 返回 true。
not	布尔"非" - 如果x为True，返回False。如果x为False，它返回True。	not(a and b) 返回 false。

针对逻辑运算的进一步研究：

1, 在没有()的情况下not 优先级高于 and，and优先级高于or，即优先级关系为( )>not>and>or，同一优先级从左往右计算。

() > not > and > or

例题：

判断下列逻辑语句的True，False。

```
3>4 or 4<3 and 1==1
1 < 2 and 3 < 4 or 1>2
2 > 1 and 3 < 4 or 4 > 5 and 2 < 1
1 > 2 and 3 < 4 or 4 > 5 and 2 > 1 or 9 < 8
1 > 1 and 3 < 4 or 4 > 5 and 2 > 1 and 9 > 8 or 7 < 6
not 2 > 1 and 3 < 4 or 4 > 5 and 2 > 1 and 9 > 8 or 7 < 6
```

2, x or y , x为真，值就是x，x为假，值是y；

x and y, x为真，值是y,x为假，值是x。

Operation	Result	Notes
x or y	if *x* is false, then *y*, else *x*	(1)
x and y	if *x* is false, then *x*, else *y*	(2)
not x	if *x* is false, then True, else False	(3)

例题：求出下列逻辑语句的值。

```
8 or 4
```

```
0 and 3
0 or 4 and 3 or 7 or 9 and 6
```

三. 编码的问题

python2解释器在加载 .py 文件中的代码时，会对内容进行编码（默认ascill）,而python3对内容进行编码的默认为utf-8。

计算机:

早期. 计算机是美国发明的. 普及率不高, 一般只是在美国使用. 所以. 最早的编码结构就是按照美国人的习惯来编码的. 对应数字+字母+特殊字符一共也没多少. 所以就形成了最早的编码ASCII码. 直到今天ASCII依然深深的影响着我们.

ASCII（American Standard Code for Information Interchange，美国标准信息交换代码）是基于拉丁字母的一套电脑编码系统，主要用于显示现代英语和其他西欧语言，其最多只能用 8 位来表示（一个字节），即：2**8 = 256，所以，ASCII码最多只能表示 256 个符号。

Bin(二进制)	Oct(八进制)	Dec(十进制)	Hex(十六进制)	缩写/字符	解释
0000 0000	0	0	00	NUL(null)	空字符
0000 0001	1	1	01	SOH(start of headline)	标题开始
0000 0010	2	2	02	STX (start of text)	正文开始
0000 0011	3	3	03	ETX (end of text)	正文结束
0000 0100	4	4	04	EOT (end of transmission)	传输结束
0000 0101	5	5	05	ENQ (enquiry)	请求
0000 0110	6	6	06	ACK (acknowledge)	收到通知
0000 0111	7	7	07	BEL (bell)	响铃
0000 1000	10	8	08	BS (backspace)	退格
0000 1001	11	9	09	HT (horizontal tab)	水平制表符
0000 1010	12	10	0A	LF (NL line feed, new line)	换行键
0000 1011	13	11	0B	VT (vertical tab)	垂直制表符
0000 1100	14	12	0C	FF (NP form feed, new page)	换页键
0000 1101	15	13	0D	CR (carriage return)	回车键

0000 1110	16	14	0E	SO (shift out)	不用切换
0000 1111	17	15	0F	SI (shift in)	启用切换
0001 0000	20	16	10	DLE (data link escape)	数据链路转义
0001 0001	21	17	11	DC1 (device control 1)	设备控制1
0001 0010	22	18	12	DC2 (device control 2)	设备控制2
0001 0011	23	19	13	DC3 (device control 3)	设备控制3
0001 0100	24	20	14	DC4 (device control 4)	设备控制4
0001 0101	25	21	15	NAK (negative acknowledge)	拒绝接收
0001 0110	26	22	16	SYN (synchronous idle)	同步空闲
0001 0111	27	23	17	ETB (end of trans. block)	结束传输块
0001 1000	30	24	18	CAN (cancel)	取消
0001 1001	31	25	19	EM (end of medium)	媒介结束
0001 1010	32	26	1A	SUB (substitute)	代替
0001 1011	33	27	1B	ESC (escape)	换码(溢出)
0001 1100	34	28	1C	FS (file separator)	文件分隔符
0001 1101	35	29	1D	GS (group separator)	分组符
0001 1110	36	30	1E	RS (record separator)	记录分隔符
0001 1111	37	31	1F	US (unit separator)	单元分隔符
0010 0000	40	32	20	(space)	空格
0010 0001	41	33	21	!	叹号
0010 0010	42	34	22	"	双引号

0010 0011	43	35	23	#	井号
0010 0100	44	36	24	$	美元符
0010 0101	45	37	25	%	百分号
0010 0110	46	38	26	&	和号
0010 0111	47	39	27	'	闭单引号
0010 1000	50	40	28	(	开括号
0010 1001	51	41	29	)	闭括号
0010 1010	52	42	2A	*	星号
0010 1011	53	43	2B	+	加号
0010 1100	54	44	2C	,	逗号
0010 1101	55	45	2D	–	减号/破折号
0010 1110	56	46	2E	.	句号
00101111	57	47	2F	/	斜杠
00110000	60	48	30	0	数字0
00110001	61	49	31	1	数字1
00110010	62	50	32	2	数字2
00110011	63	51	33	3	数字3
00110100	64	52	34	4	数字4
00110101	65	53	35	5	数字5
00110110	66	54	36	6	数字6
00110111	67	55	37	7	数字7
00111000	70	56	38	8	数字8
00111001	71	57	39	9	数字9
00111010	72	58	3A	:	冒号
00111011	73	59	3B	;	分号
00111100	74	60	3C	<	小于
00111101	75	61	3D	=	等号
00111110	76	62	3E	>	大于
00111111	77	63	3F	?	问号

01000000	100	64	40	@	电子邮件符号
01000001	101	65	41	A	大写字母A
01000010	102	66	42	B	大写字母B
01000011	103	67	43	C	大写字母C
01000100	104	68	44	D	大写字母D
01000101	105	69	45	E	大写字母E
01000110	106	70	46	F	大写字母F
01000111	107	71	47	G	大写字母G
01001000	110	72	48	H	大写字母H
01001001	111	73	49	I	大写字母I
01001010	112	74	4A	J	大写字母J
01001011	113	75	4B	K	大写字母K
01001100	114	76	4C	L	大写字母L
01001101	115	77	4D	M	大写字母M
01001110	116	78	4E	N	大写字母N
01001111	117	79	4F	O	大写字母O
01010000	120	80	50	P	大写字母P
01010001	121	81	51	Q	大写字母Q
01010010	122	82	52	R	大写字母R
01010011	123	83	53	S	大写字母S
01010100	124	84	54	T	大写字母T
01010101	125	85	55	U	大写字母U
01010110	126	86	56	V	大写字母V
01010111	127	87	57	W	大写字母W
01011000	130	88	58	X	大写字母X
01011001	131	89	59	Y	大写字母Y
01011010	132	90	5A	Z	大写字母Z
01011011	133	91	5B	[	开方括号
01011100	134	92	5C	\	反斜杠

01011101	135	93	5D	]	闭方括号
01011110	136	94	5E	^	脱字符
01011111	137	95	5F	_	下划线
01100000	140	96	60	`	开单引号
01100001	141	97	61	a	小写字母a
01100010	142	98	62	b	小写字母b
01100011	143	99	63	c	小写字母c
01100100	144	100	64	d	小写字母d
01100101	145	101	65	e	小写字母e
01100110	146	102	66	f	小写字母f
01100111	147	103	67	g	小写字母g
01101000	150	104	68	h	小写字母h
01101001	151	105	69	i	小写字母i
01101010	152	106	6A	j	小写字母j
01101011	153	107	6B	k	小写字母k
01101100	154	108	6C	l	小写字母l
01101101	155	109	6D	m	小写字母m
01101110	156	110	6E	n	小写字母n
01101111	157	111	6F	o	小写字母o
01110000	160	112	70	p	小写字母p
01110001	161	113	71	q	小写字母q
01110010	162	114	72	r	小写字母r
01110011	163	115	73	s	小写字母s
01110100	164	116	74	t	小写字母t
01110101	165	117	75	u	小写字母u
01110110	166	118	76	v	小写字母v
01110111	167	119	77	w	小写字母w
01111000	170	120	78	x	小写字母x
01111001	171	121	79	y	小写字母y

01111010	172	122	7A	z	小写字母z
01111011	173	123	7B	{	开花括号
01111100	174	124	7C	|	垂线
01111101	175	125	7D	}	闭花括号
01111110	176	126	7E	~	波浪号
01111111	177	127	7F	DEL (delete)	删除

随着计算机的发展. 以及普及率的提高. 流行到欧洲和亚洲. 这时ASCII码就不合适了. 比如: 中文汉字有几万个. 而ASCII最多也就256个位置. 所以ASCII不行了. 怎么办呢? 这时, 不同的国家就提出了不同的编码用来适用于各自的语言环境. 比如, 中国的GBK, GB2312, BIG5, ISO-8859-1等等. 这时各个国家都可以使用计算机了.

GBK, 国标码占用2个字节. 对应ASCII码 GBK直接兼容. 因为计算机底层是用英文写的. 你不支持英文肯定不行. 而英文已经使用了ASCII码. 所以GBK要兼容ASCII.

这里GBK国标码. 前面的ASCII码部分. 由于使用两个字节. 所以对于ASCII码而言. 前9位都是0

```
字母A:0100 0001        # ASCII
字母A:0000 0000 0100 0001 # 国标码
```

国标码的弊端: 只能中国用. 日本就垮了. 所以国标码不满足我们的使用. 这时提出了一个万国码Unicode. unicode一开始设计是每个字符两个字节. 设计完了. 发现我大中国汉字依然无法进行编码. 只能进行扩充. 扩充成32位也就是4个字节. 这回够了. 但是. 问题来了. 中国字9万多. 而unicode可以表示40多亿. 根本用不了. 太浪费了. 于是乎, 就提出了新的UTF编码.可变长度编码

UTF-8: 每个字符最少占8位. 每个字符占用的字节数不定.根据文字内容进行具体编码. 比如. 英文. 就一个字节就够了. 汉字占3个字节. 这时即满足了中文. 也满足了节约. 也是目前使用频率最高的一种编码

UTF-16: 每个字符最少占16位.

GBK: 每个字符占2个字节, 16位.

单位转换:

8bit = 1byte

1024byte = 1KB

1024KB = 1MB

1024MB = 1GB

1024GB = 1TB

1024TB = 1PB

1024TB = 1EB

1024EB = 1ZB

1024ZB = 1YB

1024YB = 1NB

1024NB = 1DB

常用到TB就够了

补充1: while循环.

while 条件:

循环体

else: 循环在正常情况跳出之后会执行这里

```
index = 1
while index < 11:
    if index == 8:
        # break
        pass
    else:
        print(index)
    index = index+1
else:print("你好")
```

注意: 如果循环是通过break退出的. 那么while后面的else将不会被执行, 只有在while条件判断是假的时候才会执行这个else

pass: 不表示任何内容. 为了代码的完整性. 占位而已

补充2: in和not in

可以判断xxx字符串是否出现在xxxxx字符串中

```
content = input("请输入你的评论")
if "苍老师" in content or '邱老师' in content:
    print('你输入的内容不合法')
else:
    print("评论成功")
```