Python 编程基础练习（二）

练习23：字符串、字节串和字符编码

ex23.py

```
1 import sys
2 script, encoding, error = sys.argv
3
4
5 def main(language_file, encoding, errors):
6 line = language_file.readline()
7
8 if line:
9 print_line(line, encoding, errors)
10 return main(language_file, encoding, errors)
11
12
13 def print_line(line, encoding, errors):
14 next_lang = line.strip()
15 raw_bytes = next_lang.encode(encoding, errors=errors)
16 cooked_string = raw_bytes.decode(encoding, errors=errors)
17
18 print(raw_bytes, "<===>", cooked_string)
19
20
21 languages = open("languages.txt", encoding="utf-8")
22
23 main(languages, encoding, error)
```

练习23：字符串、字节串和字符编码

- 点击languages.txt下载，并使用如下命令运行 ex23.py：

python ex23.py utf-8 strict

languages.txt

练习（1）

ex24.py

```
1 print("Let's practice everything.")
2 print('You\'d need to know \'bout escapes with \\ that do:')
3 print('\n newlines and \t tabs.')
4
5 poem = """
6 \tThe lovely world
7 with logic so firmly planted
8 cannot discern \n the needs of love
9 nor comprehend passion from intuition
10 and requires an explanation
11 \n\t\twhere there is none.
12 """
13
14 print("--------------")
15 print(poem)
16 print("--------------")
17
18
19 five = 10 - 2 + 3 - 6
20 print(f"This should be five: {five}")
21
22 def secret_formula(started):
23 jelly_beans = started * 500
24 jars = jelly_beans / 1000
25 crates = jars / 100
26 return jelly_beans, jars, crates
27
28
29 start_point = 10000
30 beans, jars, crates = secret_formula(start_point)
31
32 # remember that this is another way to format a string
33 print("With a starting point of: {}".format(start_point))
34 # it's just like with an f"" string
35  print(f"We'd  have  {beans}  beans,  {jars}  jars,  and  {crates}
```

练习24：更多练习（1）

```
crates.")
36
37 start_point = start_point / 10
38
39 print("We can also do that this way:")
40 formula = secret_formula(start_point)
41 # this is an easy way to apply a list to a format string
42    print("We'd    have    {}    beans,    {}    jars,    and    {}
crates.".format(*formula))
```

练习24：更多练习（1）

```
Exercise 24 Session
```

```
$ python3.6 ex24.py
Let's practice everything.
You'd need to know 'bout escapes with \ that do:

newlines and tabs.
--------------

The lovely world
with logic so firmly planted
cannot discern
the needs of love
nor comprehend passion from intuition
and requires an explanation

where there is none.

--------------
This should be five: 5
With a starting point of: 10000
We'd have 5000000 beans, 5000.0 jars, and 50.0 crates.
We can also do that this way:
We'd have 500000.0 beans, 500.0 jars, and 5.0 crates.
```

练习（2）

ex25.py

```
1 def break_words(stuff):
2 """This function will break up words for us."""
3 words = stuff.split(' ')
4 return words
5
6 def sort_words(words):
7 """Sorts the words."""
8 return sorted(words)
9
10 def print_first_word(words):
11 """Prints the first word after popping it off."""
12 word = words.pop(0)
13 print(word)
14
15 def print_last_word(words):
16 """Prints the last word after popping it off."""
17 word = words.pop(-1)
18 print(word)
19
20 def sort_sentence(sentence):
21 """Takes in a full sentence and returns the sorted words."""
22 words = break_words(sentence)
23 return sort_words(words)
24
25 def print_first_and_last(sentence):
26 """Prints the first and last words of the sentence."""
27 words = break_words(sentence)
28 print_first_word(words)
29 print_last_word(words)
30
31 def print_first_and_last_sorted(sentence):
32 """Sorts the words then prints the first and last one."""
33 words = sort_sentence(sentence)
34 print_first_word(words)

35 print_last_word(words)
```

练习25：更多练习（2）

Exercise 25 Python Session

```
1 import ex25
2 sentence = "All good things come to those who wait."
3 words = ex25.break_words(sentence)
4 words
5 sorted_words = ex25.sort_words(words)
6 sorted_words
7 ex25.print_first_word(words)
8 ex25.print_last_word(words)
9 words
10 ex25.print_first_word(sorted_words)
11 ex25.print_last_word(sorted_words)
12 sorted_words
13 sorted_words = ex25.sort_sentence(sentence)
14 sorted_words
15 ex25.print_first_and_last(sentence)
16 ex25.print_first_and_last_sorted(sentence)
```

练习（2）

Exercise 25 Python Session

```
Python 3.6.0 (default, Feb 2 2017, 12:48:29)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> import ex25
>>> sentence = "All good things come to those who wait."
>>> words = ex25.break_words(sentence)
>>> words
['All', 'good', 'things', 'come', 'to', 'those', 'who', 'wait.']
>>> sorted_words = ex25.sort_words(words)
>>> sorted_words
['All', 'come', 'good', 'things', 'those', 'to', 'wait.', 'who']
>>> ex25.print_first_word(words)
All
>>> ex25.print_last_word(words)
wait.
>>> words
['good', 'things', 'come', 'to', 'those', 'who']
>>> ex25.print_first_word(sorted_words)
All
>>> ex25.print_last_word(sorted_words)
who
>>> sorted_words
['come', 'good', 'things', 'those', 'to', 'wait.']
>>> sorted_words = ex25.sort_sentence(sentence)
>>> sorted_words
['All', 'come', 'good', 'things', 'those', 'to', 'wait.', 'who']
>>> ex25.print_first_and_last(sentence)
All
wait.
>>> ex25.print_first_and_last_sorted(sentence)
All
who
```

练习26：恭喜，你可以考试了

- 考试内容：点击下载ex26.py.txt，重命名为ex26.py后修改程序，使之可以正常运行。

ex26.py.txt

- 注意：不可以删除ex26.py中任何一条语句！

练习27：逻辑关系

- and
- or
- not
- != (not equal)
- == (equal)
- >= (greater-than-equal)
- <= (less-than-equal)
- True
- False

练习28：布尔表达式

```
1. True and True
2. False and True
3. 1 == 1 and 2 == 1
4. "test" == "test"
5. 1 == 1 or 2 != 1
6. True and 1 == 1
7. False and 0 != 0
8. True or 1 == 1
9. "test" == "testing"
10. 1 != 0 and 2 == 1
11. "test" != "testing"
12. "test" == 1
13. not (True and False)
14. not (1 == 1 and 0 != 1)
15. not (10 == 1 or 1000 == 1000)
16. not (1 != 10 or 3 == 4)
17. not ("testing" == "testing" and "Zed" == "Cool Guy")
18. 1 == 1 and (not ("testing" == 1 or 1 == 0))
19. "chunky" == "bacon" and (not (3 == 4 or 3 == 3))
20. 3 == 3 and (not ("testing" == "testing" or "Python" == "Fun"))
```

if语句

ex29.py

```
1 people = 20
2 cats = 30
3 dogs = 15
4
5
6 if people < cats:
7 print("Too many cats! The world is doomed!")
8
9 if people > cats:
10 print("Not many cats! The world is saved!")
11
12 if people < dogs:
13 print("The world is drooled on!")
14
15 if people > dogs:
16 print("The world is dry!")
17
18
19 dogs += 5
20
21 if people >= dogs:
22 print("People are greater than or equal to dogs.")
23
24 if people <= dogs:
25 print("People are less than or equal to dogs.")
26
27
28 if people == dogs:
29 print("People are dogs.")
```

练习29：if语句

```
Exercise 29 Session
```

```
$ python3.6 ex29.py
Too many cats! The world is doomed!

The world is dry!
People are greater than or equal to dogs.
People are less than or equal to dogs.
People are dogs.
```

练习30：else和if

ex30.py

```
1 people = 30
2 cars = 40
3 trucks = 15
4
5
6 if cars > people:
7 print("We should take the cars.")
8 elif cars < people:
9 print("We should not take the cars.")
10 else:
11 print("We can't decide.")
12
13 if trucks > cars:
14 print("That's too many trucks.")
15 elif trucks < cars:
16 print("Maybe we could take the trucks.")
17 else:
18 print("We still can't decide.")
19
20 if people > trucks:
21 print("Alright, let's just take the trucks.")
22 else:
23 print("Fine, let's stay home then.")
```

练习30：else和if

```
Exercise 30 Session
```

```
$ python3.6 ex30.py
We should take the cars.
Maybe we could take the trucks.
Alright, let's just take the trucks.
```

31：做出决定

ex31.py

```
1 print("""You enter a dark room with two doors.
2 Do you go through door #1 or door #2?""")
3
4 door = input("> ")
5
6 if door == "1":
7 print("There's a giant bear here eating a cheese cake.")
8 print("What do you do?")
9 print("1. Take the cake.")
10 print("2. Scream at the bear.")
11
12 bear = input("> ")
13
14 if bear == "1":
15 print("The bear eats your face off. Good job!")
16 elif bear == "2":
17 print("The bear eats your legs off. Good job!")
18 else:
19 print(f"Well, doing {bear} is probably better.")
20 print("Bear runs away.")
21
22 elif door == "2":
23 print("You stare into the endless abyss at Cthulhu's retina.")
24 print("1. Blueberries.")
25 print("2. Yellow jacket clothespins.")
26 print("3. Understanding revolvers yelling melodies.")
27
28 insanity = input("> ")
29
```

练习31：做出决定

```
30 if insanity == "1" or insanity == "2":
31 print("Your body survives powered by a mind of jello.")
32 print("Good job!")
33 else:
34 print("The insanity rots your eyes into a pool of muck.")
35 print("Good job!")
36
37 else:
38  print("You stumble around and fall on a knife and die. Good
job!")
```

练习31：做出决定

Exercise 31 Session

```
$ python3.6 ex31.py
You enter a dark room with two doors.
Do you go through door #1 or door #2?
> 1
There's a giant bear here eating a cheese cake.
What do you do?
1. Take the cake.
2. Scream at the bear.
> 2
The bear eats your legs off. Good job!
```

循环和列表

ex32.py

```
1 the_count = [1, 2, 3, 4, 5]
2 fruits = ['apples', 'oranges', 'pears', 'apricots']
3 change = [1, 'pennies', 2, 'dimes', 3, 'quarters']
4
5 # this first kind of for-loop goes through a list
6 for number in the_count:
7 print(f"This is count {number}")
8
9 # same as above
10 for fruit in fruits:
11 print(f"A fruit of type: {fruit}")
12
13 # also we can go through mixed lists too
14 # notice we have to use {} since we don't know what's in it
15 for i in change:
16 print(f"I got {i}")
17
18 # we can also build lists, first start with an empty one
19 elements = []
20
21 # then use the range function to do 0 to 5 counts
22 for i in range(0, 6):
23 print(f"Adding {i} to the list.")
24 # append is a function that lists understand
25 elements.append(i)
26
27 # now we can print them out too
28 for i in elements:
29 print(f"Element was: {i}")
```

练习32：循环和列表

```
Exercise 32 Session
```

```
$ python3.6 ex32.py
This is count 1
This is count 2
This is count 3
This is count 4
This is count 5
A fruit of type: apples
A fruit of type: oranges
A fruit of type: pears

A fruit of type: apricots
I got 1
I got pennies
I got 2
I got dimes
I got 3
I got quarters
Adding 0 to the list.
Adding 1 to the list.
Adding 2 to the list.
Adding 3 to the list.
Adding 4 to the list.
Adding 5 to the list.
Element was: 0
Element was: 1
Element was: 2
Element was: 3
Element was: 4
Element was: 5
```

练习33：while循环

ex33.py

```
1 i = 0

2 numbers = []
3
4 while i < 6:
5 print(f"At the top i is {i}")
6 numbers.append(i)
7
8 i = i + 1
9 print("Numbers now: ", numbers)
10 print(f"At the bottom i is {i}")
11
12
13 print("The numbers: ")
14
15 for num in numbers:
16 print(num)
```

练习33：while循环

```
Exercise 33 Session
```

```
$ python3.6 ex33.py
At the top i is 0
Numbers now: [0]
At the bottom i is 1
At the top i is 1
Numbers now: [0, 1]
At the bottom i is 2
At the top i is 2
Numbers now: [0, 1, 2]
At the bottom i is 3
At the top i is 3
Numbers now: [0, 1, 2, 3]
At the bottom i is 4
At the top i is 4
Numbers now: [0, 1, 2, 3, 4]
At the bottom i is 5
At the top i is 5
Numbers now: [0, 1, 2, 3, 4, 5]
At the bottom i is 6
The numbers:
0
1
2
3
4
5
```

练习34：访问列表元素

Python列表脚本操作符

列表对 + 和 * 的操作符与字符串相似。+ 号用于组合列表，* 号用于重复列表。

如下所示：

Python 表达式	结果	描述
len([1, 2, 3])	*3*	长度
[1, 2, 3] + [4, 5, 6]	*[1, 2, 3, 4, 5, 6]*	组合
*['Hi!'] * 4*	*['Hi!', 'Hi!', 'Hi!', 'Hi!']*	重复
3 in [1, 2, 3]	*True*	元素是否存在于列表中
for x in [1, 2, 3]: print(x, end=" ")	*1 2 3*	迭代

练习34：访问列表元素

Python列表截取与拼接

*Python*的列表截取与字符串操作类型，如下所示：

```
L=['Google', 'Runoob', 'Taobao']
```

操作：

Python 表达式	结果	描述
L[2]	*'Taobao'*	读取第三个元素
L[-2]	*'Runoob'*	从右侧开始读取倒数第二个元素: *count from the right*
L[1:]	*['Runoob', 'Taobao']*	输出从第二个元素开始后的所有元素

练习34：访问列表元素

Python包含以下函数:

序号	函数
1	len(list) 列表元素个数
2	max(list) 返回列表元素最大值
3	min(list) 返回列表元素最小值
4	list(seq) 将元组转换为列表

ex35.py

```
1 from sys import exit
2
3 def gold_room():
4 print("This room is full of gold. How much do you take?")
5
6 choice = input("> ")
7 if "0" in choice or "1" in choice:
8 how_much = int(choice)
9 else:
10 dead("Man, learn to type a number.")
11
12 if how_much < 50:
13 print("Nice, you're not greedy, you win!")
14 exit(0)
15 else:
16 dead("You greedy bastard!")
17
18
19 def bear_room():
20 print("There is a bear here.")
21 print("The bear has a bunch of honey.")
22 print("The fat bear is in front of another door.")
23 print("How are you going to move the bear?")
24 bear_moved = False
25
26 while True:
27 choice = input("> ")
28
29 if choice == "take honey":
30 dead("The bear looks at you then slaps your face off.")
31 elif choice == "taunt bear" and not bear_moved:
32 print("The bear has moved from the door.")
33 print("You can go through it now.")
34 bear_moved = True
35 elif choice == "taunt bear" and bear_moved:
36 dead("The bear gets pissed off and chews your leg off.")
37 elif choice == "open door" and bear_moved:
38 gold_room()
39 else:
```

分支和函数

```
40 print("I got no idea what that means.")
41
42
43 def cthulhu_room():
44 print("Here you see the great evil Cthulhu.")
45 print("He, it, whatever stares at you and you go insane.")
46 print("Do you flee for your life or eat your head?")
47
48 choice = input("> ")
49
50 if "flee" in choice:
51 start()
52 elif "head" in choice:
53 dead("Well that was tasty!")
54 else:
55 cthulhu_room()
56
57
58 def dead(why):
59 print(why, "Good job!")
60 exit(0)
61
62 def start():
63 print("You are in a dark room.")
64 print("There is a door to your right and left.")
65 print("Which one do you take?")
66
67 choice = input("> ")
68
69 if choice == "left":
70 bear_room()
71 elif choice == "right":
72 cthulhu_room()
73 else:
74 dead("You stumble around the room until you starve.")
75
76
77 start()
```

练习35：分支和函数

```
Exercise 35 Session
```

```
$ python3.6 ex35.py
You are in a dark room.
There is a door to your right and left.

Which one do you take?
> left
There is a bear here.
The bear has a bunch of honey.
The fat bear is in front of another door.
How are you going to move the bear?
> taunt bear
The bear has moved from the door.
You can go through it now.
> open door
This room is full of gold. How much do you take?
> 1000
You greedy bastard! Good job!
```

练习36：设计和调试

- 略

练习37：复习各种符合

Python 保留字符

下面的列表显示了在*Python*中的保留字。这些保留字不能用作常数或变数，或任何其他标识符名称。

所有 *Python* 的关键字只包含小写字母。

and	*exec*	*not*
assert	*finally*	*or*
break	*for*	*pass*
class	*from*	*print*
continue	*global*	*raise*
def	*if*	*return*
del	*import*	*try*
elif	*in*	*while*
else	*is*	*with*
except	*lambda*	*yield*

学以精工

练习37：复习各种符合

Python算术运算符

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（**向下取整**）	`>>> 9//2`
		`4`
		`>>> -9//2`
		`-5`

练习37：复习各种符合

Python比较运算符

以下假设变量a为10，变量b为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。python3 已废弃。	(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。	(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。

练习37：复习各种符合

Python赋值运算符

以下假设变量a为10，变量b为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

练习37：复习各种符合

Python位运算符

按位运算符是把数字看作二进制来进行计算的。*Python*中的按位运算法则如下：

下表中变量 *a* 为 *60*，*b* 为 *13*，二进制格式如下：

```
a = 0011 1100

b = 0000 1101

-----------------

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a  = 1100 0011
```

练习37：复习各种符合

运算符	描述	实例
&	按位与运算符：参与运算的两个值,如果两个相应位都为1,则该位的结果为1,否则为0	(a & b) 输出结果 12 ，二进制解释： 0000 1100
|	按位或运算符：只要对应的二个二进位有一个为1时，结果位就为1。	(a | b) 输出结果 61 ，二进制解释： 0011 1101
^	按位异或运算符：当两对应的二进位相异时，结果为1	(a ^ b) 输出结果 49 ，二进制解释： 0011 0001
~	按位取反运算符：对数据的每个二进制位取反,即把1变为0,把0变为1 。`~x` 类似于 `-x-1`	(~a) 输出结果 -61 ，二进制解释： 1100 0011，在一个有符号二进制数的补码形式。
<<	左移动运算符：运算数的各二进位全部左移若干位，由 `<<` 右边的数字指定了移动的位数，高位丢弃，低位补0。	a << 2 输出结果 240 ，二进制解释： 1111 0000
>>	右移动运算符：把">>"左边的运算数的各二进位全部右移若干位，`>>` 右边的数字指定了移动的位数	a >> 2 输出结果 15 ，二进制解释： 0000 1111

练习37：复习各种符合

Python逻辑运算符

*Python*语言支持逻辑运算符，以下假设变量 *a* 为 *10*, *b*为 *20*:

运算符	逻辑表达式	描述	实例
and	*x and y*	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 *20*。
or	*x or y*	布尔"或" - 如果 *x* 是非 *0*，它返回 *x* 的值，否则它返回 *y* 的计算值。	*(a or b)* 返回 *10*。
not	*not x*	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*

练习37：复习各种符合

Python成员运算符

除了以上的一些运算符之外，*Python*还支持成员运算符，测试实例中包含了一系列的成员，包括字符串，列表或元组。

运算符	描述	实例
in	如果在指定的序列中找到值返回 *True*，否则返回 *False*。	*x* 在 *y* 序列中 , 如果 *x* 在 *y* 序列中返回 *True*。
not in	如果在指定的序列中没有找到值返回 *True*，否则返回 *False*。	*x* 不在 *y* 序列中 , 如果 *x* 不在 *y* 序列中返回 *True*。

练习37：复习各种符合

Python身份运算符

身份运算符用于比较两个对象的存储单元

运算符	描述	实例
is	*is* 是判断两个标识符是不是引用自一个对象	**x is y**, 类似 **id(x) == id(y)** , 如果引用的是同一个对象则返回 *True*，否则返回 *False*
is not	*is not* 是判断两个标识符是不是引用自不同对象	**x is not y** ， 类似 **id(a) != id(b)**。如果引用的不是同一个对象则返回结果 *True*，否则返回 *False*。

练习37：复习各种符合

Python运算符优先级

以下表格列出了从最高到最低优先级的所有运算符：

运算符	描述
**	指数 (最高优先级)
~ + -	按位翻转, 一元加号和减号 (最后两个的方法名为 +@ 和 -@)
* / % //	乘，除，取模和取整除
+ -	加法减法
>> <<	右移，左移运算符
&	位 'AND'
^ |	位运算符
<= < > >=	比较运算符
<> == !=	等于运算符
= %= /= //= -= += *= **=	赋值运算符
is is not	身份运算符
in not in	成员运算符
not and or	逻辑运算符

练习37：复习各种符合

Python 转义字符

在需要在字符中使用特殊字符时，*python* 用反斜杠 `\` 转义字符。如下表：

转义字符	描述
\(在行尾时)	续行符
**	反斜杠符号
\'	单引号
\"	双引号
\a	响铃
\b	退格*(Backspace)*
\e	转义
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\oyy	八进制数，*yy*代表的字符，例如：*\o12*代表换行
\xyy	十六进制数，*yy*代表的字符，例如：*\x0a*代表换行
\other	其它的字符以普通格式输出

练习37：复习各种符合

Python字符串运算符

下表实例变量 *a* 值为字符串 *"Hello"*，*b* 变量值为 *"Python"*:

操作符	描述	实例
+	字符串连接	>>>a + b 'HelloPython'
*	重复输出字符串	>>>a * 2 'HelloHello'
[]	通过索引获取字符串中字符	>>>a[1] 'e'
[:]	截取字符串中的一部分	>>>a[1:4] 'ell'
in	成员运算符 - 如果字符串中包含给定的字符返回 *True*	>>>"H" in a True
not in	成员运算符 - 如果字符串中不包含给定的字符返回 *True*	>>>"M" not in a True
r/R	原始字符串 - 原始字符串：所有的字符串都是直接按照字面的意思来使用，没有转义特殊或不能打印的字符。 原始字符串除在字符串的第一个引号前加上字母"r"（可以大小写）以外，与普通字符串有着几乎完全相同的语法。	>>>print r'\n' \n >>> print R'\n' \n

练习37：复习各种符合

python 字符串格式化符号:

符 号	描述
%c	格式化字符及其ASCII码
%s	格式化字符串
%d	格式化整数
%u	格式化无符号整型
%o	格式化无符号八进制数
%x	格式化无符号十六进制数
%X	格式化无符号十六进制数（大写）
%f	格式化浮点数字，可指定小数点后的精度
%e	用科学计数法格式化浮点数
%E	作用同%e，用科学计数法格式化浮点数
%g	%f和%e的简写
%G	%F 和 %E 的简写
%p	用十六进制数格式化变量的地址

练习38：列表的操作

ex38.py

```
1 ten_things = "Apples Oranges Crows Telephone Light Sugar"
2
3 print("Wait there are not 10 things in that list. Let's fix
that.")
4
5 stuff = ten_things.split(' ')
6 more_stuff = ["Day", "Night", "Song", "Frisbee",
7 "Corn", "Banana", "Girl", "Boy"]
8
9 while len(stuff) != 10:
10 next_one = more_stuff.pop()
11 print("Adding: ", next_one)
12 stuff.append(next_one)
13 print(f"There are {len(stuff)} items now.")
14
15 print("There we go: ", stuff)
16
17 print("Let's do some things with stuff.")
18
19 print(stuff[1])
20 print(stuff[-1]) # whoa! fancy

21 print(stuff.pop())
22 print(' '.join(stuff)) # what? cool!
23 print('#'.join(stuff[3:5])) # super stellar!
```

练习38：列表的操作

```
Wait there are not 10 things in that list. Let's fix that.
Adding: Boy
There are 7 items now.
Adding: Girl
There are 8 items now.
Adding: Banana
There are 9 items now.
Adding: Corn
There are 10 items now.
There we go: ['Apples', 'Oranges', 'Crows', 'Telephone', 'Light',
'Sugar', 'Boy', 'Girl', 'Banana', 'Corn']
Let's do some things with stuff.
Oranges
Corn
Corn
Apples Oranges Crows Telephone Light Sugar Boy Girl Banana
Telephone#Light
```

39：字典

ex39.py

```
1 # create a mapping of state to abbreviation
2 states = {
3 'Oregon': 'OR',
4 'Florida': 'FL',
5 'California': 'CA',
6 'New York': 'NY',
7 'Michigan': 'MI'
8 }
9
10 # create a basic set of states and some cities in them
11 cities = {
12 'CA': 'San Francisco',
13 'MI': 'Detroit',
14 'FL': 'Jacksonville'
15 }
16
17 # add some more cities
18 cities['NY'] = 'New York'
19 cities['OR'] = 'Portland'
20
21 # print out some cities
22 print('-' * 10)
23 print("NY State has: ", cities['NY'])
24 print("OR State has: ", cities['OR'])
25
26 # print some states
27 print('-' * 10)
28 print("Michigan's abbreviation is: ", states['Michigan'])
29 print("Florida's abbreviation is: ", states['Florida'])
30
31 # do it by using the state then cities dict
32 print('-' * 10)
33 print("Michigan has: ", cities[states['Michigan']])
34 print("Florida has: ", cities[states['Florida']])
35
```

练习39：字典

```
36 # print every state abbreviation
37 print('-' * 10)
38 for state, abbrev in list(states.items()):
39 print(f"{state} is abbreviated {abbrev}")
40
41 # print every city in state
42 print('-' * 10)
43 for abbrev, city in list(cities.items()):
44 print(f"{abbrev} has the city {city}")
45
46 # now do both at the same time
47 print('-' * 10)
48 for state, abbrev in list(states.items()):
49 print(f"{state} state is abbreviated {abbrev}")
50 print(f"and has city {cities[abbrev]}")
51
52 print('-' * 10)
53 # safely get a abbreviation by state that might not be there
54 state = states.get('Texas')
55
56 if not state:
57 print("Sorry, no Texas.")
58
59 # get a city with a default value
60 city = cities.get('TX', 'Does Not Exist')
61 print(f"The city for the state 'TX' is: {city}")
```

39：字典

```
Exercise 39 Session
----------

$ python3.6 ex39.py
----------
NY State has: New York
OR State has: Portland
----------
Michigan's abbreviation is: MI
Florida's abbreviation is: FL
----------
Michigan has: Detroit
Florida has: Jacksonville
----------
Oregon is abbreviated OR
Florida is abbreviated FL
California is abbreviated CA
New York is abbreviated NY
Michigan is abbreviated MI
----------
CA has the city San Francisco
MI has the city Detroit
FL has the city Jacksonville
NY has the city New York
OR has the city Portland
----------
Oregon state is abbreviated OR
and has city Portland
Florida state is abbreviated FL
and has city Jacksonville
California state is abbreviated CA
and has city San Francisco
New York state is abbreviated NY
and has city New York
Michigan state is abbreviated MI
and has city Detroit
----------
Sorry, no Texas.
The city for the state 'TX' is: Does Not Exist
```

练习40：模块、类和对象

ex40.py

```
1 class Song(object):
2
3 def __init__(self, lyrics):
4 self.lyrics = lyrics
5
6 def sing_me_a_song(self):
7 for line in self.lyrics:
8 print(line)
9
10 happy_bday = Song(["Happy birthday to you",
11 "I don't want to get sued",
12 "So I'll stop right there"])
13
14 bulls_on_parade = Song(["They rally around tha family",
15 "With pockets full of shells"])
16
17 happy_bday.sing_me_a_song()
18
19 bulls_on_parade.sing_me_a_song()
```

练习40：模块、类和对象

```
Exercise 40 Session
```

```
$ python3.6 ex40.py
Happy birthday to you
I don't want to get sued
So I'll stop right there
They rally around tha family
With pockets full of shells
```

练习40：模块、类和对象

单下划线、双下划线、头尾双下划线说明：

- __foo__: 定义的是特殊方法，一般是系统定义名字，类似 `__init__()` 之类的。
- _foo: 以单下划线开头的表示的是 *protected* 类型的变量，即保护类型只能允许其本身与子类进行访问，不能用于 `from module import *`
- __foo: 双下划线的表示的是私有类型(*private*)的变量, 只能是允许这个类本身进行访问了。

练习41：学习面向对象术语

面向对象技术简介

- **类(Class)**: 用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例。
- **类变量**：类变量在整个实例化的对象中是公用的。类变量定义在类中且在函数体之外。类变量通常不作为实例变量使用。
- **数据成员**：类变量或者实例变量, 用于处理类及其实例对象的相关的数据。
- **方法重写**：如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖（*override*），也称为方法的重写。
- **局部变量**：定义在方法中的变量，只作用于当前实例的类。
- **实例变量**：在类的声明中，属性是用变量来表示的。这种变量就称为实例变量，是在类声明的内部但是在类的其他成员方法之外声明的。
- **继承**：即一个派生类（*derived class*）继承基类（*base class*）的字段和方法。继承也允许把一个派生类的对象作为一个基类对象对待。例如，有这样一个设计：一个*Dog*类型的对象派生自*Animal*类，这是模拟"是一个（*is-a*）"关系（例图，*Dog*是一个*Animal*）。
- **实例化**：创建一个类的实例，类的具体对象。
- **方法**：类中定义的函数。
- **对象**：通过类定义的数据结构实例。对象包括两个数据成员（类变量和实例变量）和方法。

练习42：对象、类和从属关系

```
2 class Animal(object):
3 pass
4
5 ## ??
6 class Dog(Animal):
7
8 def __init__(self, name):
9 ## ??
10 self.name = name
11
12 ## ??
13 class Cat(Animal):
14
15 def __init__(self, name):
16 ## ??
17 self.name = name
18
19 ## ??
20 class Person(object):
21
22 def __init__(self, name):
23 ## ??
24 self.name = name
25
```

练习42：对象、类和从属关系

```
25
26 ## Person has-a pet of some kind
27 self.pet = None
28
29 ## ??
30 class Employee(Person):
31
32 def __init__(self, name, salary):
33 ## ?? hmm what is this strange magic?
34 super(Employee, self).__init__(name)
35 ## ??
36 self.salary = salary
37
38 ## ??
39 class Fish(object):
40 pass
41
42 ## ??
43 class Salmon(Fish):
44 pass
45
46 ## ??
47 class Halibut(Fish):
48 pass
49
```

练习42：对象、类和从属关系

```
51 ## rover is-a Dog
52 rover = Dog("Rover")
53
54 ## ??
55 satan = Cat("Satan")
56
57 ## ??
58 mary = Person("Mary")
59
60 ## ??
61 mary.pet = satan
62
63 ## ??
64 frank = Employee("Frank", 120000)
65
66 ## ??
67 frank.pet = rover
68
69 ## ??
70 flipper = Fish()
71
72 ## ??
73 crouse = Salmon()
74
75 ## ??
76 harry = Halibut()
```

练习43：基本的面向对象分析设计

- 略

练习44：继承与组合

ex44d.py

```
1 class Parent(object):
2
3 def override(self):
4 print("PARENT override()")
5
6 def implicit(self):

7 print("PARENT implicit()")
8
9 def altered(self):
10 print("PARENT altered()")
11
12 class Child(Parent):
13
14 def override(self):
15 print("CHILD override()")
16
17 def altered(self):
18 print("CHILD, BEFORE PARENT altered()")
19 super(Child, self).altered()
20 print("CHILD, AFTER PARENT altered()")
21
22 dad = Parent()
23 son = Child()
24
25 dad.implicit()
26 son.implicit()
27
28 dad.override()
29 son.override()
30
31 dad.altered()
32 son.altered()
```

练习44：继承与组合

```
Exercise 44d Session
==========

$ python3.6 ex44d.py
PARENT implicit()
PARENT implicit()
PARENT override()
CHILD override()
PARENT altered()
CHILD, BEFORE PARENT altered()
PARENT altered()
CHILD, AFTER PARENT altered()
```

练习44：继承与组合

ex44e.py

```
1 class Other(object):
2
3 def override(self):
4 print("OTHER override()")
5
6 def implicit(self):
7 print("OTHER implicit()")
8
9 def altered(self):
10 print("OTHER altered()")
11
12 class Child(object):
13
14 def __init__(self):
15 self.other = Other()
16
17 def implicit(self):
18 self.other.implicit()
19
20 def override(self):
21 print("CHILD override()")
22
23 def altered(self):
24 print("CHILD, BEFORE OTHER altered()")
25 self.other.altered()
26 print("CHILD, AFTER OTHER altered()")
27
28 son = Child()
29
30 son.implicit()
31 son.override()
32 son.altered()
```

练习44：继承与组合

```
Exercise 44e Session
```

```
$ python3.6 ex44e.py

OTHER implicit()
CHILD override()
CHILD, BEFORE OTHER altered()
OTHER altered()
CHILD, AFTER OTHER altered()
```

练习44：继承与组合

类的继承

面向对象的编程带来的主要好处之一是代码的重用，实现这种重用的方法之一是通过继承机制。

通过继承创建的新类称为**子类**或**派生类**，被继承的类称为**基类**、**父类**或**超类**。

继承语法

```
class 派生类名(基类名)
    ...
```

在*python*中继承中的一些特点：

- *1*、如果在子类中需要父类的构造方法就需要显示的调用父类的构造方法，或者不重写父类的构造方法。详细说明可查看：*python* 子类继承父类构造函数说明。
- *2*、在调用基类的方法时，需要加上基类的类名前缀，且需要带上 *self* 参数变量。区别在于类中调用普通函数时并不需要带上 *self* 参数
- *3*、*Python* 总是首先查找对应类型的方法，如果它不能在派生类中找到对应的方法，它才开始到基类中逐个查找。（先在本类中查找调用的方法，找不到才去基类中找）。

如果在继承元组中列了一个以上的类，那么它就被称作"多重继承"。