一、函数编程

1.1 引子，函数编程的作用

你是一家公司的IT运维人员， 现在老板让你写一个监控程序，24小时全年无休的监控你们公司网站服务器的系统状况，当cpu \ memory \ disk等指标的使用量超过阀值时即发邮件报警，你掏空了所有的知识量，写出了以下代码

```
while True:
    if cpu利用率 > 90%:
        #发送邮件提醒
        连接邮箱服务器
        发送邮件
        关闭连接
    if 硬盘使用空间 > 90%:
        #发送邮件提醒
        连接邮箱服务器
        发送邮件
        关闭连接
    if 内存占用 > 80%:
        #发送邮件提醒
        连接邮箱服务器
        发送邮件
        关闭连接
```

上面的代码实现了功能，但即使是邻居老王也看出了端倪，老王亲切的摸了下你家儿子的脸蛋，说，你这个重复代码太多了，每次报警都要重写一段发邮件的代码，太low了，这样干存在2个问题：

1. 代码重复过多，一个劲的copy and paste不符合高端程序员的气质
2. 如果日后需要修改发邮件的这段代码，比如加入群发功能，那你就需要在所有用到这段代码的地方都修改一遍

你觉得老王说的对，你也不想写重复代码，但又不知道怎么搞，老王好像看出了你的心思，此时他抱起你儿子，笑着说，其实很简单，**只需要把重复的代码提取出来，放在一个公共的地方，起个名字，以后谁想用这段代码，就通过这个名字调用就行了**，如下

```
def 发送邮件(内容)
    #发送邮件提醒
    连接邮箱服务器
    发送邮件
    关闭连接
while True:
    if cpu利用率 > 90%:
        发送邮件('CPU报警')
    if 硬盘使用空间 > 90%:
        发送邮件('硬盘报警')
    if 内存占用 > 80%:
        发送邮件('内存报警')
```

你看着老王写的代码，气势恢宏、磅礴大气，代码里透露着一股内敛的傲气，心想，老王这个人真是不一般，突然对他的背景更感兴趣了，问老王，这些花式玩法你都是怎么知道的？ 老王亲了一口你儿子，捋了捋不存在的胡子，淡淡的讲，“老夫，年少时，师从京西沙河淫魔银角大王 ”， 你一听“银角大王”这几个字，不由的娇躯一震，心想，真nb,怪不得代码写的这么6, 这“银角大王”当年在江湖上可是数得着的响当当的名字，只可惜后期纵欲过度，卒于公元2019年， 真是可惜了，只留下其哥哥孤守当年兄弟俩一起打下来的江山。 此时你看着的老王离开的身影，感觉你儿子跟他越来越像了。。

1.2 语法定义 & 函数特性

函数是什么?

函数一词来源于数学，但编程中的「函数」概念，与数学中的函数是有很大不同的，具体区别，我们后面会讲，编程中的函数在英文中也有很多不同的叫法。在BASIC中叫做subroutine(子过程或子程序)，在C中只有function，在Java里面叫做method。

定义: 函数是指将一组语句的集合通过一个名字(函数名)封装起来，要想执行这个函数，只需调用其函数名即可

特性:

1. 减少重复代码
2. 使程序变的可扩展
3. 使程序变得易维护

语法定义

```
def sayhi():#函数名
    print("Hello, I'm nobody!")

sayhi() #调用函数
```

可以带参数

```
def calc(x,y):
    res = x**y
    print(res) #函数执行结果

calc(5,22)
```

参数可以让你的函数更灵活，不只能做死的动作，还可以根据调用时传参的不同来决定函数内部的执行流程

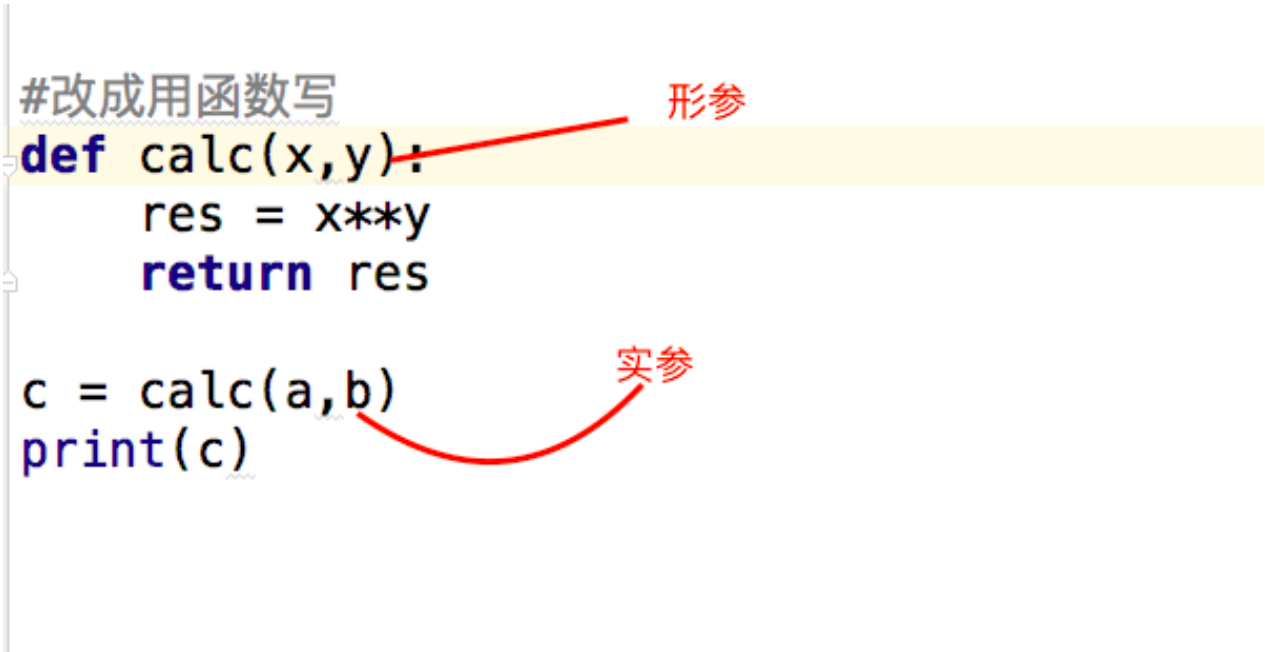
1.3 各种参数

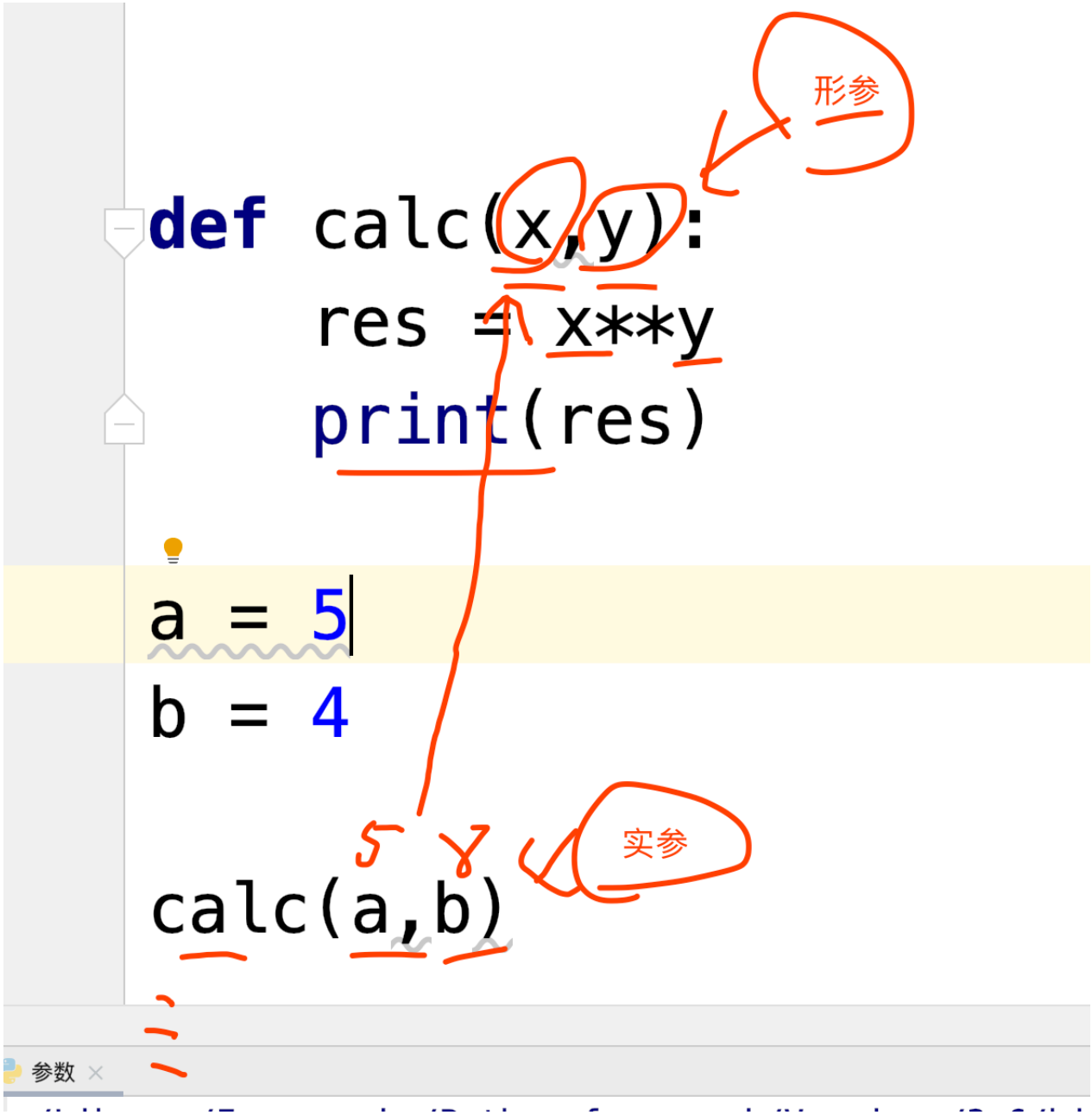
形参变量

只有在被调用时才分配内存单元，在调用结束时，即刻释放所分配的内存单元。因此，形参只在函数内部有效。函数调用结束返回主调用程序代码后则不能再使用该形参变量

实参

可以是变量、任意数据类型、表达式、函数等，无论实参是何种类型的量，在进行函数调用时，它们都必须有确定的值，以便把这些值传送给形参。因此应预先给实参赋值





默认参数

看如下代码

```
def stu_register(name,age,country,course):
    print("----注册学生信息------")
    print("姓名:",name)
    print("age:",age)
    print("国籍:",country)
    print("课程:",course)

stu_register("王山炮",22,"CN","python_devops")
stu_register("张叫春",21,"CN","linux")
stu_register("刘老根",25,"CN","linux")
```

发现 country 这个参数 基本都 是"CN", 就像我们在网站上注册用户，像国籍这种信息，你不填写，默认就会是 中国， 这就是通过默认参数实现的，把country变成默认参数非常简单

```
def stu_register(name,age,course,country="CN"):
```

这样，这个参数在调用时不指定，那默认就是CN，指定了的话，就用你指定的值。

另外，你可能注意到了，在把country变成默认参数后，我同时把它的位置移到了最后面，为什么呢?

这是语法强制的，默认参数放在其他参数后边，为啥呢？ 假设允许这样：

```
def stu_register(name,age,country="CN",course):
```

那调用时

```
stu_register("Mack",22,"Python","US")
```

你告诉我，第3个参数`python`到底应该给到country还是course呢？ 无论给哪个，都会出现歧义，所以Python语法干脆就让你把默认参数放最后， 解释器在处理函数时参数时，按优先级，**位置参数>默认参数**

关键参数(指定参数)

正常情况下，给函数传参数要按顺序，不想按顺序就可以用关键参数，只需指定参数名即可(指定了参数名的参数就叫关键参数)，***但记住一个要求就是，关键参数必须放在位置参数(以位置顺序确定对应关系的参数)之后***

```
def stu_register(name, age, course='PY' ,country='CN'):
    print("----注册学生信息------")
    print("姓名:", name)
    print("age:", age)
    print("国籍:", country)
    print("课程:", course)
```

调用可以这样

```
stu_register("王山炮",course='PY', age=22,country='JP' )
```

但绝不可以这样

```
stu_register("王山炮",course='PY',22,country='JP' )
```

当然这样也不行

```
stu_register("王山炮",22,age=25,country='JP' )
```

这样相当于给age赋值2次，会报错!

注意，参数优先级顺序是 位置参数>关键参数

非固定参数

若你的函数在定义时不确定用户想传入多少个参数，就可以使用非固定参数

```
def stu_register(name,age,*args): # *args 会把多传入的参数变成一个元组形式
    print(name,age,args)

stu_register("Alex",22)
#输出
#Alex 22 () #后面这个()就是args,只是因为没传值,所以为空

stu_register("Jack",32,"CN","Python")
#输出
# Jack 32 ('CN', 'Python')
```

还可以有一个`**kwargs`

```
def stu_register(name,age,*args,**kwargs): # *kwargs 会把多传入的参数变成一个dict形式
    print(name,age,args,kwargs)
stu_register("Alex",22)
#输出
#Alex 22 () {}#后面这个{}就是kwargs,只是因为没传值,所以为空

stu_register("Jack",32,"CN","Python",sex="Male",province="ShanDong")
#输出
# Jack 32 ('CN', 'Python') {'province': 'ShanDong', 'sex': 'Male'}
```

练习题

1.根据下图所示，对print_info里的代码进行实现

函数参数.py

```
1
2   def print_info(*args,**kwargs):...
12
13
14  print_info(name="Alex",age=22,sex="M")
15  print_info(name="Jack",age=26,sex="M",hobbie="学习")
16
17
```

Run: 函数参数

```
/Users/alex/PycharmProjects/apeland_py_learn/venv/bin/python /Users/alex/Py
--------info-------
Name: Alex
Age: 22
Sex: M
Hobbie: 大保健
--------info-------
Name: Jack
Age: 26
Sex: M
Hobbie: 学习
```

1.4 返回值return

函数外部的代码要想获取函数的执行结果，就可以在函数里用return语句把结果返回

```
def stu_register(name, age, course='PY' ,country='CN'):
    print("----注册学生信息------")
    print("姓名:", name)
    print("age:", age)
    print("国籍:", country)
    print("课程:", course)
    if age > 22:
        return False
    else:
        return True

registriation_status = stu_register("王山炮",22,course="PY全栈开发",country='JP')
if registriation_status:
    print("注册成功")
else:
    print("too old to be a student.")
```

注意

- 函数在执行过程中只要遇到return语句，就会停止执行并返回结果，so 也可以理解为 return 语句代表着函数的结束
- 如果未在函数中指定return,那这个函数的返回值为None

1.5 局部变量与全局变量

看如下代码

```
name = "Alex Li"
def change_name():
    name = "金角大王，   一个有Tesla的高级屌丝"
    print("after change", name)

change_name()
print("在外面看看name改了么?",name)
```

输出

```
after change 金角大王,一个有Tesla的高级屌丝
在外面看看name改了么? Alex Li
```

为什么在函数内部改了name的值后， 在外面print的时候却没有改呢？ 因为这两个name根本不是一回事

- 在函数中定义的变量称为局部变量，在程序的一开始定义的变量称为全局变量。
- 全局变量作用域(即有效范围)是整个程序，局部变量作用域是定义该变量的函数。
- 变量的查找顺序是**局部变量>全局变量**
- 当全局变量与局部变量同名时，在定义局部变量的函数内，局部变量起作用；在其它地方全局变量起作用。
- 在函数里是不能直接修改全局变量的

就是想在函数里修改全局变量怎么办?

```
name = "Alex Li"
def change_name():
    global name #声明一个全局变量
    name = "Alex 又名金角大王,爱生活、爱自由、爱姑娘"
    print("after change", name)

change_name()
print("在外面看看name改了么?", name)
```

`global name`**的作用就是要在函数里声明全局变量name，意味着最上面的**`name = “Alex Li”`**即使不写，程序最后面的print也可以打印name**

虽然可以改，但不建议用这个global语法，随着代码增多，会造成代码调试困难

传递列表、字典产生的现象

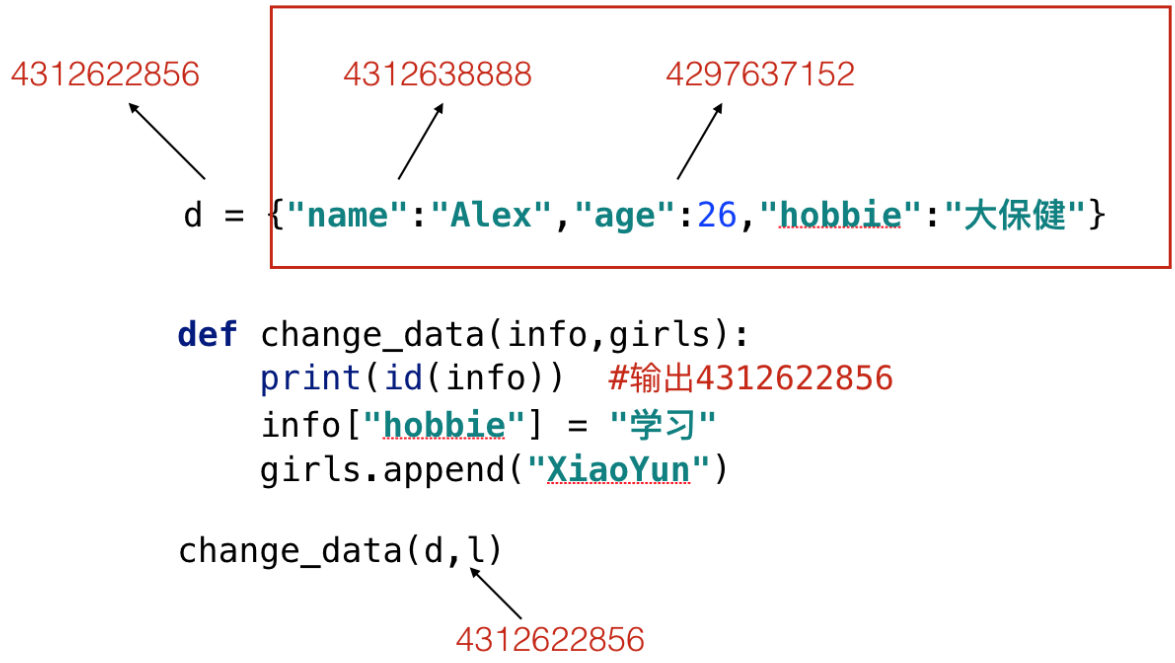
```
d = {"name":"Alex","age":26,"hobbie":"大保健"}
l = ["Rebeeca","Katrina","Rachel"]

def change_data(info,girls):
    info["hobbie"] = "学习"
    girls.append("XiaoYun")

change_data(d,l)
print(d,l)
```

执行结果

{'name': 'Alex', 'age': 26, 'hobbie': '学习'}

['Rebeeca', 'Katrina', 'Rachel', 'XiaoYun']

不是说不能在函数里改全局变量么，怎么改了呀?



根据上图我们能看出， 程序只是把d这个dict的内存地址传给了change_data函数，把dict比作鱼缸，里面的k,v比作缸里装的鱼。现在只是把鱼缸丢给了函数，这个鱼缸本身你不能改，但是里面的鱼可以。相当于只是传了一个对这个d的引用关系给到函数的形参。这样是为了减少内存的浪费，因为如果这个dict比较大，传一次到函数里就要copy一份新的值的话，效率太低了。

1.6 内置函数

Python的`len`为什么你可以直接用？肯定是解释器启动时就定义好了

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

画红线的是咱们这个集训营必须要学的，其它的有兴趣可以自己探索。

内置参数详解 https://docs.python.org/3/library/functions.html?highlight=built#ascii

每个函数的作用我都帮你标好了

1. abs # 求绝对值
2. all #Return True if bool(x) is True for all values x in the iterable.If the iterable is empty, return True.
3. any #Return True if bool(x) is True for any x in the iterable.If the iterable is empty, return False.
4. ascii #Return an ASCII-only representation of an object,ascii(“中国”) 返回’’\u4e2d\u56fd’’
5. bin #返回整数的2进制格式
6. bool # 判断一个数据结构是True or False, bool({}) 返回就是False, 因为是空dict
7. bytearray # 把byte变成 bytearray, 可修改的数组
8. bytes # bytes(“中国”,”gbk”)
9. callable # 判断一个对象是否可调用
10. chr # 返回一个数字对应的ascii字符 ， 比如chr(90)返回ascii里的’Z’
11. classmethod #面向对象时用，现在忽略
12. compile #py解释器自己用的东西，忽略
13. complex #求复数，一般人用不到
14. copyright #没用
15. credits #没用
16. delattr #面向对象时用，现在忽略
17. dict #生成一个空dict
18. dir #返回对象的可调用属性

19. divmod #返回除法的商和余数，比如divmod(4,2)，结果(2, 0)
20. enumerate #返回列表的索引和元素，比如 d = [“alex”,”jack”]，enumerate(d)后，得到(0, ‘alex’) (1, ‘jack’)
21. eval #可以把字符串形式的list,dict,set,tuple,再转换成其原有的数据类型。
22. exec #把字符串格式的代码，进行解义并执行，比如exec(“print(‘hellworld’)”)，会解义里面的字符串并执行
23. exit #退出程序
24. filter #对list、dict、set、tuple等可迭代对象进行过滤， filter(lambda x:x>10, [0,1,23,3,4,4,5,6,67,7])过滤出所有大于10的值
25. float #转成浮点
26. format #没用
27. frozenset #把一个集合变成不可修改的
28. getattr #面向对象时用，现在忽略
29. globals #打印全局作用域里的值
30. hasattr #面向对象时用，现在忽略
31. hash #hash函数
32. help
33. hex #返回一个10进制的16进制表示形式,hex(10) 返回’0xa’
34. id #查看对象内存地址
35. input
36. int
37. isinstance #判断一个数据结构的类型，比如判断a是不是fronzenset, isinstance(a,frozenset) 返回 True or False
38. issubclass #面向对象时用，现在忽略
39. iter #把一个数据结构变成迭代器，讲了迭代器就明白了
40. len
41. list
42. locals
43. map # map(lambda x:x**2,[1,2,3,43,45,5,6,]) 输出 [1, 4, 9, 1849, 2025, 25, 36]
44. max # 求最大值
45. memoryview # 一般人不用，忽略
46. min # 求最小值
47. next # 生成器会用到，现在忽略
48. object #面向对象时用，现在忽略
49. oct # 返回10进制数的8进制表示
50. open
51. ord # 返回ascii的字符对应的10进制数 ord(‘a’) 返回97，
52. print
53. property #面向对象时用，现在忽略
54. quit
55. range
56. repr #没什么用
57. reversed # 可以把一个列表反转
58. round #可以把小数4舍5入成整数，round(10.15,1) 得10.2
59. set
60. setattr #面向对象时用，现在忽略
61. slice # 没用

62. sorted
63. staticmethod #面向对象时用，现在忽略
64. str
65. sum #求和,a=[1, 4, 9, 1849, 2025, 25, 36],sum(a) 得3949
66. super #面向对象时用，现在忽略
67. tuple
68. type
69. vars #返回一个对象的属性，面向对象时就明白了
70. zip #可以把2个或多个列表拼成一个， a=[1, 4, 9, 1849, 2025, 25, 36]，b = [“a”,”b”,”c”,”d”],

```
list(zip(a,b)) #得结果 [(1, 'a'), (4, 'b'), (9, 'c'), (1849, 'd')]
```

1.7 练习题：学籍注册程序

需求：

1. 要求用户输入姓名、年龄、手机号、身份证号、所选课程，然后为学员完成注册
2. 手机号、身份证号唯一
3. 可选的课程只能从Python、Linux、网络安全、前端、数据分析 这几门里选
4. 学员信息存入文件

1.8 练习题：炸金花棋牌游戏

需求：

1. 允许用户一次性输入多个玩家姓名，不限个数，然后为每个玩家随机生成3张牌
2. 你只有一付扑克牌，确保发出去的每张牌不重样
3. 牌需要有黑桃、红桃、方片、梅花之分