[Top](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/PYTHON2/DAY02/CASE/01/index.html" \l "page_top_case)

# NSD Python2 DAY02

1. [案例1：简单的加减法数学游戏](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/PYTHON2/DAY02/CASE/01/index.html" \l "case1)
2. [案例2：简单GUI程序](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/PYTHON2/DAY02/CASE/01/index.html" \l "case2)
3. [案例3：快速排序](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/PYTHON2/DAY02/CASE/01/index.html" \l "case3)
4. [案例4：备份程序](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/PYTHON2/DAY02/CASE/01/index.html" \l "case4)

## 1 案例1：简单的加减法数学游戏

### 1.1 问题

编写math\_game.py脚本，实现以下目标：

1. 随机生成两个100以内的数字
2. 随机选择加法或是减法
3. 总是使用大的数字减去小的数字
4. 如果用户答错三次，程序给出正确答案

### 1.2 方案

创建4个函数，分别实现返回两数之和、返回两数之差、判断表达式正确性、是否继续计算四种方法：

1.首先调用main()函数（是否继续计算功能），main函数利用循环无限次调用exam()函数进行计算，计算结束，用户选择是否继续（此过程利用try语句捕获索引错误、ctrl+c（中断）错误、ctrl+d错误），如果选择n即结束循环，不再调用exam()函数，否则循环继续

2.调用exam()函数：

a)输出运算公式：利用列表切片将随机生成的两个数打印（这两个数利用random模块及列表生成式随机生成，并利用sort()方法进行降序排序，确保相减时一直是大的数字减小的数字），利用random模块随机生成“+”“-”号，输出在两数之间

b)用户输入值，利用for循环进行三次判断，如果运算公式结果与用户输入值相同，循环结束，系统输出“你赢了”，exam()函数执行结束，否则系统输出“你答错了”，循环继续，3次都回答错误，利用循环的else分支输出运算公式及结果

c)上诉运算公式结果：利用random模块随机生成“+”“-”值对关系调用（其中“+”“-”号作为字典键，返回和函数add()及返回差函数sub()作为值，调用时将随机生成的两个数字作为参数传递给add()函数及sub()函数）

### 1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

1. [root@localhost day06]# vim math\_game.py
2. #!/usr/bin/env python3
3. import random
4. def add(x, y):
5. return x + y
6. def sub(x, y):
7. return x - y
8. def exam():
9. cmds = {'+': add, '-': sub} # 将函数存入字典
10. nums = [random.randint(1, 100) for i in range(2)] # 生成两个数
11. nums.sort(reverse=True) # 降序排列
12. op = random.choice('+-')
13. result = cmds[op](\*nums) # 调用存入字典的函数，把nums列表拆开，作为参数传入
14. prompt = "%s %s %s = " % (nums[0], op, nums[1])
15. for i in range(3):
16. try:
17. answer = int(input(prompt))
18. except:
19. continue
20. if answer == result:
21. print('你真棒，答对了！')
22. break # 答对了就不要再回答了，结束循环
23. else:
24. print('答错了')
25. else:
26. print("%s%s" % (prompt, result)) # 只有循环不被break才执行
27. def main():
28. while True:
29. exam()
30. try:
31. go\_on = input('Continue(y/n)? ').strip()[0]
32. except IndexError:
33. continue
34. except (KeyboardInterrupt, EOFError):
35. go\_on = 'n'
36. if go\_on in 'nN':
37. print('\nBye-bye.')
38. break
39. if \_\_name\_\_ == '\_\_main\_\_':
40. main()

实现此案例还可利用while循环：

1. [root@localhost day06]# vim mygui.py
2. #!/usr/bin/env python3
3. import random
4. def add(x, y):
5. return x + y
6. def sub(x, y):
7. return x - y
8. def exam():
9. cmds = {'+': add, '-': sub} # 将函数存入字典
10. nums = [random.randint(1, 100) for i in range(2)] # 生成两个数
11. nums.sort(reverse=True) # 降序排列
12. op = random.choice('+-')
13. result = cmds[op](\*nums) # 调用存入字典的函数，把nums列表拆开，作为参数传入
14. prompt = "%s %s %s = " % (nums[0], op, nums[1])
15. tries = 0
16. while tries < 3:
17. try:
18. answer = int(input(prompt))
19. except:
20. continue
21. if answer == result:
22. print('你真棒，答对了！')
23. break # 答对了就不要再回答了，结束循环
24. else:
25. print('答错了')
26. tries += 1
27. else:
28. print("%s%s" % (prompt, result)) # 只有循环不被break才执行
29. def main():
30. while True:
31. exam()
32. try:
33. go\_on = input('Continue(y/n)? ').strip()[0]
34. except IndexError:
35. continue
36. except (KeyboardInterrupt, EOFError):
37. go\_on = 'n'
38. if go\_on in 'nN':
39. print('\nBye-bye.')
40. break
41. if \_\_name\_\_ == '\_\_main\_\_':
42. main()

步骤二：测试脚本执行

1. [root@localhost day06]# python3 math\_game.py
2. 54 + 19 =
3. 54 + 19 =
4. 54 + 19 = 73
5. 你真棒，答对了！
6. Continue(y/n)? y
7. 60 + 39 = 99
8. 你真棒，答对了！
9. Continue(y/n)? y
10. 18 + 15 = 33
11. 你真棒，答对了！
12. Continue(y/n)? y
13. 35 + 20 = 55
14. 你真棒，答对了！
15. Continue(y/n)? y
16. 37 + 35 = 72
17. 你真棒，答对了！
18. Continue(y/n)? y
19. 77 - 57 = 20
20. 你真棒，答对了！
21. Continue(y/n)? y
22. 35 + 23 = 5
23. 答错了
24. 35 + 23 = 6
25. 答错了
26. 35 + 23 = 7
27. 答错了
28. 35 + 23 = 58
29. Continue(y/n)? y
30. 75 + 47 = 122
31. 你真棒，答对了！
32. Continue(y/n)? ^C
33. Bye-bye.

## 2 案例2：简单GUI程序

### 2.1 问题

创建mygui.py脚本，要求如下：

1. 窗口程序提供三个按钮
2. 其中两个按钮的前景色均为白色，背景色为蓝色
3. 第三个按钮前景色为红色，背景色为红色
4. 按下第三个按钮后，程序退出

### 2.2 方案

1.导入tkinter模块、创建顶层窗口，顶层窗口只应该创建一次

2.添加窗口部件：用Label控件创建标签、用Butten控件来创建按钮

3.引入偏函数partial把tkinter.Button的某些参数给固定住（也就是设置默认值），返回一个新的函数，调用这个新函数重复创建按钮会更简单。对于有很多可调用对象，并且许多调用都反复使用相同参数的情况，使用偏函数比较合适。

4.创建第三个按钮需command绑定退出命令

5.最后将按钮及标签填充到界面

6.运行这个GUI应用

### 2.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

1. [root@localhost day06]# vim mygui.py
2. #!/usr/bin/env python3
3. import tkinter
4. from functools import partial
5. root = tkinter.Tk()    #创建顶层窗口
6. lb1 = tkinter.Label(root, text="hello world!", font = "Aria 16 bold")#创建标签
7. b1 = tkinter.Button(root, bg='blue', fg='white', text="Button 1")#创建按钮
8. mybutton = partial(tkinter.Button, root, bg='blue', fg='white')
9. #调用新的函数时，给出改变的参数即可
10. b2 = mybutton(text='Button 2')
11. b3 = tkinter.Button(root, bg='red', fg='red', text='QUIT', command=root.quit)    #创建按钮，绑定了root.quit命令
12. lb1.pack()    #填充到界面
13. b1.pack()
14. b2.pack()
15. b3.pack()
16. root.mainloop()    #运行这个GUI应用

实现此案例还可用以下方法，需要注意的是：

Command绑定闭包函数返回函数块，上传多个参数调用闭包函数时，内层函数利用config方法替换标签内容

1. [root@localhost day06]# vim mygui.py
2. #!/usr/bin/env python3
3. import tkinter
4. from functools import partial
5. def say\_hi(word):
6. def welcome():
7. lb1.config(text='hello %s!' % word)
8. return welcome
9. root = tkinter.Tk()
10. lb1 = tkinter.Label(root, text="hello world!", font = "Aria 16 bold")
11. mybutton = partial(tkinter.Button, root, bg='blue', fg='white')
12. b1 = mybutton(text='Button 1', command=say\_hi('tedu'))
13. b2 = mybutton(text='Button 2', command=say\_hi('达内'))
14. b3 = tkinter.Button(root, bg='red', fg='red', text='QUIT', command=root.quit)
15. lb1.pack()
16. b1.pack()
17. b2.pack()
18. b3.pack()
19. root.mainloop()

步骤二：测试脚本执行，结果如图-1、图-2、图-3所示：

1. [root@localhost day05]# python3 mygui.py



图-1

1. [root@localhost day06]# python3 mygui2.py



图-2



图-3

## 3 案例3：快速排序

### 3.1 问题

创建qsort.py文件，实现以下目标：

1. 随机生成10个数字
2. 利用递归，实现快速排序

### 3.2 方案

将要排序的数据分割成独立的三部分，任意选取一个数据作为关键数据，然后将所有比它小的数都放到它前面，所有比它大的数都放到它后面，这个过程称为一趟快速排序，整个排序过程通过递归进行，以此达到整个数据变成有序序列。

一趟快速排序的算法是：

1.创建两个空列表分别用于存放比关键数小的数据和比关键数大的数据smaller和larger

2.For循环遍历将要排序的数据，将数据与关键数对比，比关键数小的放入smaller列表中，比关键数大的放入larger列表中

3.函数返回值为，以smaller列表为参数调用自身函数、关键数、以larger列表为参数调用自身函数：此时，函数每一次调用都会基于上一次的调用进行，会持续调用自身函数，参数列表数据会越来越少，我们规定，参数列表长度为0或1，递归结束，输出最终数据

4.注意：在调用qsort函数时，根据上传数据类型不同，一定要注意数据类型转化

### 3.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

1. [root@localhost day05] # vim qsort.py
2. #!/usr/bin/env python3
3. from random import randint
4. def qsort(data):
5. data = list(data)
6. if len(data) == 0 or len(data) == 1: # 长度为0或1，直接返回
7. return data
8. middle = data.pop() # 假设最后一项是中间值
9. smaller = []
10. larger = []
11. for item in data:
12. if item > middle: # 比middle大的放到larger，否则放到smaller
13. larger.append(item)
14. else:
15. smaller.append(item)
16. return qsort(smaller) + [middle] + qsort(larger)
17. if \_\_name\_\_ == '\_\_main\_\_':
18. nums = [randint(1, 100) for i in range(10)]
19. print(nums)
20. print(qsort(nums))
21. astr = 'qwertyuio'
22. print(''.join(qsort(astr)))
23. atuple = (10, 2, 34, 234, 1, 66, 88, 77)
24. print(tuple(qsort(atuple)))

步骤二：测试脚本执行

1. [root@localhost day06]# python3 qsort.py
2. [31, 87, 88, 22, 26, 91, 99, 6, 7, 44]
3. [6, 7, 22, 26, 31, 44, 87, 88, 91, 99]
4. eioqrtuwy
5. (1, 2, 10, 34, 66, 77, 88, 234)

## 4 案例4：备份程序

### 4.1 问题

编写backup.py脚本，实现以下目标：

1. 需要支持完全和增量备份
2. 周一执行完全备份
3. 其他时间执行增量备份
4. 备份文件需要打包为tar文件并使用gzip格式压缩

### 4.2 方案

整体框架创建3个函数，分别实现完全备份、增量备份、文件加密3种功能：

1.首先导入time模块，利用if进行判断，如果当地时间是星期一，执行完全备份函数，否则执行增量备份函数，其中，通配符%a代表时间星期几缩写，上传参数分别为要备份的原目录、目标目录、md5字典存放目录

2.调用完全备份函数：

a)首先获取新文件名，将新文件名放入目标目录下，目的是定义备份文件的绝对路径，以写压缩方式打开目标目录下新文件，将原目录写入新文件中，完成完全备份，其中os.path.join作用是将目录名和文件的基名拼接成一个完整的路径

b)了解os.walk()目录遍历器输出文件结构，利用for循环将要备份原目录中文件遍历出来作为字典键值对键， md5加密结果作为字典键值对的值（此时将原目录中文件作为上传参数调用文件加密函数），存入空字典中，字典中每个文件对应一个md5值，最后将字典写入到md5字典存放目录中

3.调用文件加密函数：将原目录文件循环读取逐一加密，返回加密结果

4.调用增量备份函数：

a)增量备份函数代码与完全备份函数基本一致

b)区别在于，备份前要先以二进制读方式打开md5字典存放目录，读取旧数据，判断旧数据中键对应的加密值与新加密值是否相同，如果不相同，则将新增内容写入到目标文件中（即只备份新数据）

5.注意：md5主要用于原文件与新文件判断

### 4.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

1. [root@localhost day06]# vim backup.py
2. #!/usr/bin/env python3
3. import time
4. import os
5. import tarfile
6. import hashlib
7. import pickle
8. #用于判断两个文件是否相同，提取每个文件中的前4字节的内容然后输出md5码进行比较
9. def check\_md5(fname):
10. m = hashlib.md5()
11. with open(fname, 'rb') as fobj:
12. while True:
13. data = fobj.read(4096)
14. if not data:
15. break
16. m.update(data)
17. return m.hexdigest()
18. def full\_backup(src\_dir, dst\_dir, md5file):
19. fname = os.path.basename(src\_dir.rstrip('/'))
20. fname = '%s\_full\_%s.tar.gz' % (fname, time.strftime('%Y%m%d'))
21. fname = os.path.join(dst\_dir, fname)
22. md5dict = {}
23. tar = tarfile.open(fname, 'w:gz')
24. tar.add(src\_dir)
25. tar.close()
26. for path, folders, files in os.walk(src\_dir):
27. for each\_file in files:
28. key = os.path.join(path, each\_file)
29. md5dict[key] = check\_md5(key)
30. with open(md5file, 'wb') as fobj:
31. pickle.dump(md5dict, fobj)
32. def incr\_backup(src\_dir, dst\_dir, md5file):
33. fname = os.path.basename(src\_dir.rstrip('/'))
34. fname = '%s\_incr\_%s.tar.gz' % (fname, time.strftime('%Y%m%d'))
35. fname = os.path.join(dst\_dir, fname)
36. md5dict = {}
37. with open(md5file, 'rb') as fobj:
38. oldmd5 = pickle.load(fobj)
39. for path, folders, files in os.walk(src\_dir):
40. for each\_file in files:
41. key = os.path.join(path, each\_file)
42. md5dict[key] = check\_md5(key)
43. with open(md5file, 'wb') as fobj:
44. pickle.dump(md5dict, fobj)
45. tar = tarfile.open(fname, 'w:gz')
46. for key in md5dict:
47. if oldmd5.get(key) != md5dict[key]:
48. tar.add(key)
49. tar.close()
50. if \_\_name\_\_ == '\_\_main\_\_':
51. # mkdir /tmp/demo; cp -r /etc/security /tmp/demo
52. src\_dir = '/tmp/demo/security'
53. dst\_dir = '/var/tmp/backup' # mkdir /var/tmp/backup
54. md5file = '/var/tmp/backup/md5.data'
55. if time.strftime('%a') == 'Mon':
56. full\_backup(src\_dir, dst\_dir, md5file)
57. else:
58. incr\_backup(src\_dir, dst\_dir, md5file)

步骤二：测试脚本执行

1. [root@localhost day07]# python3 backup.py
2. [root@localhost day07]# cd /var/tmp/backup/
3. [root@localhost backup]# ls
4. md5.data security\_full\_20180502.tar.gz security\_incr\_20180502.tar.gz