

NSD Python1 DAY03

1. [案例1：斐波那契数列](#)
2. [案例2：九九乘法表](#)
3. [案例3：模拟cp操作](#)
4. [案例4：生成随机密码](#)

1 案例1：斐波那契数列

1.1 问题

编写fib.py脚本，实现以下目标：

1. 斐波那契数列就是某一个数，总是前两个数之和，比如0, 1, 1, 2, 3, 5, 8
2. 使用for循环和range函数编写一个程序，计算有10个数字的斐波那契数列
3. 改进程序，要求用户输入一个数字，可以生成用户需要长度的斐波那契数列

1.2 方案

本题主要是for循环语句，写法有如下两种：

1. 输入一个变量确定列表长度，for循环用内置函数range确定循环次数，利用切片方法将列表fib最后两数之和追加到列表中，每循环一次追加一个值

2. for循环用内置函数range确定循环次数，每循环一次执行：将变量b的值赋值给变量a，并且将a b之和赋值给b，此时，a的新值是前一个b的值，b的新值是前面a b之和，让a成为数列中的值

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

```
01. [ root@localhost day 03] # vim fib.py
02.
03.  #! /usr/bin/env python3
04.
05.  a, b = 0, 1
06.
07.  for i in range( 10):
08.      print( a)
09.      a, b = b, a + b
```

或将上面的代码改为以下写法：

[Top](#)

```
01. [ root@localhost day 03] # vim fib2.py
```

```
02.
03.  #!/usr/bin/env python3
04.
05.  fib = [ 0, 1]
06.
07.  l = int( input( "数列长度: " ) )
08.  for i in range( l - 2 ):
09.      fib.append( fib[ - 1] + fib[ - 2] )
10.
11.  print( fib)
```

或将上面的代码改为以下写法：

```
01.  [ root@localhost day03] # vim fib_func.py
02.
03.  #!/usr/bin/env python3
04.
05.  def gen_fib( l ):
06.      fib = [ 0, 1]
07.
08.      for i in range( l - len( fib ) ):
09.          fib.append( fib[ - 1] + fib[ - 2] )
10.
11.      return fib # 返回列表，不返回变量fib
12.
13.  a = gen_fib( 10)
14.  print( a)
15.  print( '-' * 50)
16.  n = int( input( "length: " ) )
17.  print( gen_fib( n) ) # 不会把变量n传入，是把n代表的值赋值给形参
```

步骤二：测试脚本执行

```
01.  [ root@localhost day03] # python3 fib.py
02.  0
03.  1
04.  1
05.  2
06.  3
```

[Top](#)

```

07. 5
08. 8
09. 13
10. 21
11. 34
12. [root@localhost day03] # python3 fib2.py
13. 数列长度: 9
14. [0, 1, 1, 2, 3, 5, 8, 13, 21]
15. [root@localhost day03] # python3 fib_func.py
16. [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
17. -----
18. length: 9
19. [0, 1, 1, 2, 3, 5, 8, 13, 21]

```

2 案例2：九九乘法表

2.1 问题

创建mtable.py脚本，要求如下：

1. 程序运行后，可以在屏幕上打印出九九乘法表
2. 修改程序，由用户输入数字，可打印任意数字的乘法表

2.2 方案

本题主要用for循环双层嵌套方式编写脚本，需要注意的是：

1. 外层for循环用内置函数range，将1~9范围内的每个数字，依次装入自定义变量i中，此时，变量i被循环赋值9次
2. 内层for循环将1~变量i范围内的每个数字，依次装入变量j中，此时变量j被循环赋值i次，此时外层for循环每循环一次，内层for循环i次
3. 内层for循环range取值节点应是外层变量i加1，这样内层变量j可以取到i的值
4. 程序最后print()相当于回车，每完成一次外部循环，执行回车，作用在于美化执行结果

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

```

01. [root@localhost day03] # vim mtable.py
02.
03. #!/usr/bin/env python3
04.
05. for i in range(1, 10):          # [0, 1, 2]
06.     for j in range(1, i+1):    # i>0: [0], i>1: [0, 1], i>2: [0, 1, 2]

```

[Top](#)

```

07.         print( '%sX%s=%s' %(j, i, i*j), end=' ')
08.     print()
09.
10. [ root@localhost day 03] # vim mtable.py
11.
12.     #! /usr/bin/env python3
13.
14.     i=1
15.     while i<=9:
16.         j=1
17.         while j<=i:
18.             print( "%d*%d=%d" %(j,i,j*i),end=" ")
19.             j+=1
20.         print( "" )
21.         i+=1

```

步骤二：测试脚本执行

```

01. [ root@localhost day 03] # python3 mtable.py
02. 1X1=1
03. 1X2=2 2X2=4
04. 1X3=3 2X3=6 3X3=9
05. 1X4=4 2X4=8 3X4=12 4X4=16
06. 1X5=5 2X5=10 3X5=15 4X5=20 5X5=25
07. 1X6=6 2X6=12 3X6=18 4X6=24 5X6=30 6X6=36
08. 1X7=7 2X7=14 3X7=21 4X7=28 5X7=35 6X7=42 7X7=49
09. 1X8=8 2X8=16 3X8=24 4X8=32 5X8=40 6X8=48 7X8=56 8X8=64
10. 1X9=9 2X9=18 3X9=27 4X9=36 5X9=45 6X9=54 7X9=63 8X9=72 9X9=81

```

3 案例3：模拟cp操作

3.1 问题

创建cp.py文件，实现以下目标：

1. 将/bin/ls “拷贝” 到/root/目录下
2. 不要修改原始文件

3.2 方案

获取用户原文件名和新文件名，打开原文件，打开新文件，从打开的原文件中读取数据，写入到打开的新文件中，关闭两个文件

cp代码的过程中，需要注意的部分在于：

[Top](#)

如果一个文件过大，你将无法直接读取数据到内存，此时，使用while循环语句，分次读取数据，每次读4096字节，读取数据为空时，结束循环，将数据写入到目标文件

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

```
01. [ root@localhost day03] # vim cp.py
02.  #!/usr/bin/env python3
03.
04.  f1 = open( '/bin/ls', 'rb')
05.  f2 = open( '/root/ls', 'wb')
06.
07.  data = f1.read()
08.  f2.write( data)
09.
10.  f1.close()
11.  f2.close()
```

或将上面的代码改为以下写法：

循环读取文件中数据，避免读取数据过大

```
01. [ root@localhost day03] # vim cp2.py
02.  #!/usr/bin/env python3
03.
04.  src_fname = '/bin/ls'
05.  dst_fname = '/tmp/ls'
06.
07.  src_fobj = open( src_fname, 'rb')
08.  dst_fobj = open( dst_fname, 'wb')
09.
10.  while True:
11.      data = src_fobj.read( 4096) # 每次读4096字节
12.      if data == b'':             # 读不到数据意味着读写完毕，中断循环
13.          break
14.      dst_fobj.write( data)       # 将数据写到目标文件
15.
16.  src_fobj.close()
17.  dst_fobj.close()
```

[Top](#)

或将上面的代码改为以下写法：

With打开文件读取数据或写入数据后，文件会直接关闭

```
01. [ root@localhost day 03] # vim cp3.py
02.  #! /usr/bin/env python3
03.
04.  src_fname = '/bin/ls'
05.  dst_fname = '/root/ls'
06.
07.  with open( src_fname, 'rb') as src_fobj:
08.      with open( dst_fname, 'wb') as dst_fobj:
09.          while True:
10.              data = src_fobj.read( 4096)
11.              if not data:
12.                  break
13.              dst_fobj.write( data)
```

或将上面的代码改为以下写法：

sys.argv方法表示空列表，执行脚本时输入命令：python3 cp_func.py /bin/ls /root/ls，表示sys.argv=[cp_func.py, '/bin/ls', '/root/ls']，所以，调用copy函数时，列表切片方式获取实参为（ '/bin/ls', '/root/ls' ）

```
01. [ root@localhost day 03] # vim cp_func.py
02.  #! /usr/bin/env python3
03.
04.  import sys
05.
06.  def copy( src_fname, dst_fname):
07.      src_fobj = open( src_fname, 'rb')
08.      dst_fobj = open( dst_fname, 'wb')
09.
10.      while True:
11.          data = src_fobj.read( 4096)
12.          if not data:
13.              break
14.          dst_fobj.write( data)
15.
16.      src_fobj.close()
17.      dst_fobj.close()
```

[Top](#)

- 18.
19. `copy(sys.argv[1], sys.argv[2])`

步骤二：测试脚本执行

01. `[root@localhost day 03] # python3 cp.py`
02. `[root@localhost day 03] # cd /root`
03. `[root@localhost ~] # ls`
04. `core ls`
- 05.
06. `[root@localhost day 03] # python3 cp2.py`
07. `[root@localhost day 03] # cd /root`
08. `[root@localhost ~] # ls`
09. `core ls`
- 10.
11. `[root@localhost day 03] # python3 cp3.py`
12. `[root@localhost day 03] # cd /root`
13. `[root@localhost ~] # ls`
14. `core ls`
- 15.
16. `[root@localhost day 03] # python3 cp_func.py /bin/ls /root/ls`
17. `[root@localhost day 03] # cd /root`
18. `[root@localhost ~] # ls`
19. `core ls`

4 案例4：生成随机密码

4.1 问题

创建randpass.py脚本，要求如下：

1. 编写一个能生成8位随机密码的程序
2. 使用random的choice函数随机取出字符
3. 改进程序，用户可以自己决定生成多少位的密码

4.2 方案

导入random模块，通过random静态对象调用choice()方法，从自定义字符串all_chs中获取随机项，将获取到的随机字符ch与原result值进行拼接，将最终字符串结果返回给函数，for循环每循环一次获取一个随机字符，密码位数由循环次数决定，循环次数由传递参数值决定。

此程序需要注意的部分在于：

[Top](#)

1. 导入String模块，其中ascii_letters是生成所有字母，从a-z和A-Z，digits是生成所有数字0-

2.将整个生成随机密码的代码封装进gen_pass()函数中，当模块文件直接执行时，调用函数即可输出结果

3.参数传递问题：调用函数无实参时，函数调用默认参数，有实参时，函数调用实际参数

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

```
01. [ root@localhost day 03] # vim randpass.py
02. #!/usr/bin/env python3
03. import random
04. import string
05.
06. all_chs = string.digits + string.ascii_letters
07.
08. def gen_pass( n=8 ):
09.     result = ''
10.
11.     for i in range( n ):
12.         ch = random.choice( all_chs )
13.         result += ch
14.     return result
15.
16. if __name__ == '__main__':
17.     print( gen_pass( ) )
18.     print( gen_pass( 4 ) )
```

或将上面的代码改为以下写法：

利用列表推导式更简洁输出数据

```
01. [ root@localhost day 03] # vim randpass2.py
02. #!/usr/bin/env python3
03.
04. from random import choice
05. from string import ascii_letters, digits
06.
07. all_chs = ascii_letters + digits
08.
09.
```

[Top](#)


```
10. def randpass( n=8 ):
11.     result = [ choice( all_chs) for i in range( n) ]
12.     return ''.join( result) # 将列表的字符拼接起来
13.
14.
15. if __name__ == '__main__':
16.     print( randpass( ) )
17.     print( randpass( 4) )
```

步骤二：测试脚本执行

```
01. [ root@localhost day 03] # py thon3 randpass.py
02. 82wi2gOP
03. XzM
04. [ root@localhost day 03] # py thon3 randpass.py
05. 5wMbDEgC
06. BDpc
07. [ root@localhost day 03] # py thon3 randpass.py
08. lge2VGod
09. AzOz
10. [ root@localhost day 03] # py thon3 randpass2.py
11. eajAocMH
12. edW1
```

[Top](#)