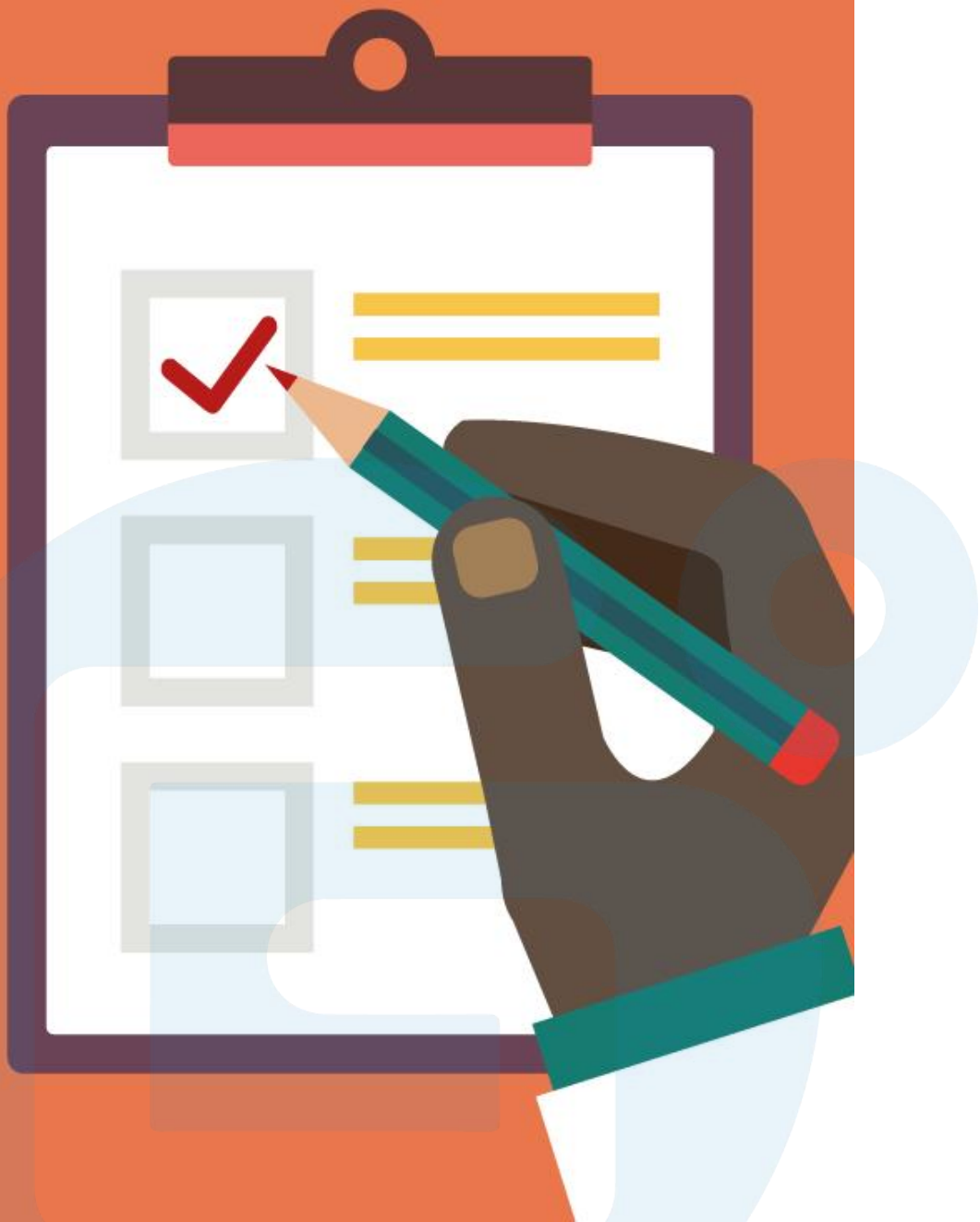


Talk is cheap, show me the code

第五课：Python控制流程

Python初阶入门课程系列



OUTLINE

- 控制流
 - 条件分支
 - 循环
 - 函数
 - 模块
-



一 控制流





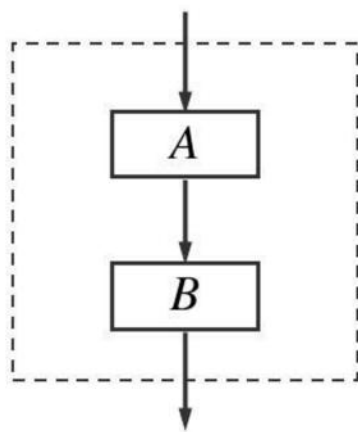
控制流的概念

Python利用内建的若干关键字控制程序的走向，即控制流程。

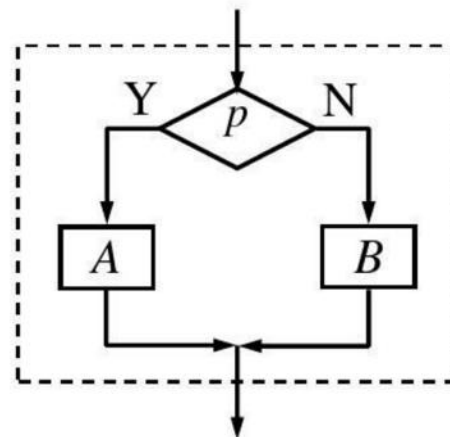
包括条件逻辑控制、循环、顺序、分支等。

- 顺序
- 循环(for,while)
- 条件分支(if,else)
- 函数
- 一些高级特性

i) 顺序结构



ii) 选择结构



顺序

- 平时写的从上到下就是顺序结构，这个最常见，也是默认的结构。
- 代码从上到下顺序执行，打乱顺序则可能导致代码无法运行。

```
7  ###
8  c=a+b
9  a=10
10 b=20
11 print(c)
12 ###
13 a=10
14 b=20
15 c=a+b
16 print(c)
17
```

控制台 1/A

```
In [4]: runfile('C:/Users/Administrator/Desktop/test/aaaa.py', wdir='C:/Users
Traceback (most recent call last):
```

```
File "C:\Users\Administrator\Desktop\test\aaaa.py", line 8, in <module>
    c=a+b
```

```
NameError: name 'a' is not defined
```

```
In [5]: runcell(2, 'C:/Users/Administrator/Desktop/test/aaaa.py')
30
```



二 条件分支



- 条件语句if是一类常见的控制语句。在各种语言中都有应用。

1. if

当if后的条件为真时，执行if语句

2. if ... else

当if的条件不为真时，执行else语句

3. if... elif...elif..else

if语句为真，执行if语句，之后的语句不执行。if语句不为真，看elif语句是否为真，为真执行；不为真看下一个elif。都不为真执行else语句。

条件分支

- if条件语句是程序先去判断某个条件是否满足，如果该条件满足，则执行判断语句后的程序。
- If条件后面的程序需要首行缩进。

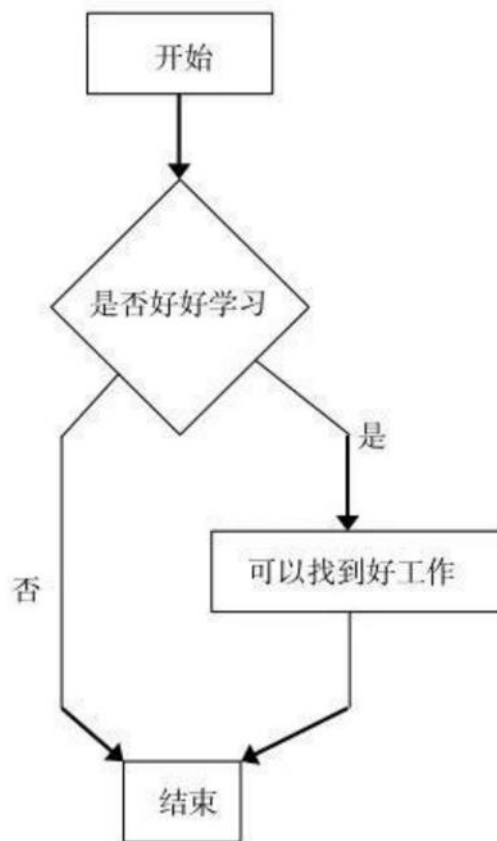
举一个例子

如果你好好学习，那么你就可以找到一份工作，但是如果你不好好学习，那么你很难找到一份工作。

我们用1表示好好学习，0表示没有好好学习，并赋初值为1，也就是假设你好好学习了。

分别设置判断条件为**是否好好学习**和**是否没有好好学习**，具体流程如下图所示：

条件分支



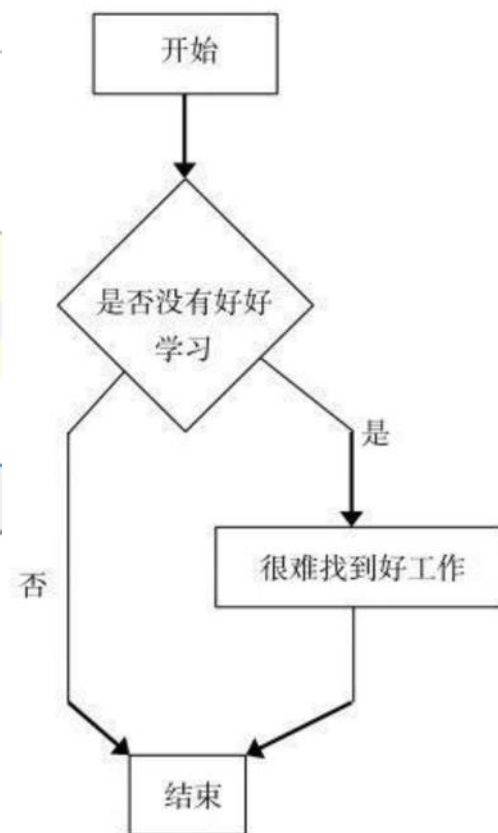
```
7  ###
8  is_study=1
9  if is_study==1:
10     print("找到了工作")
11
12  ###
13  is_study=1
14  if is_study!=0:
15     print("没有找到工作")
16
```

控制台 1/A

In [12]:

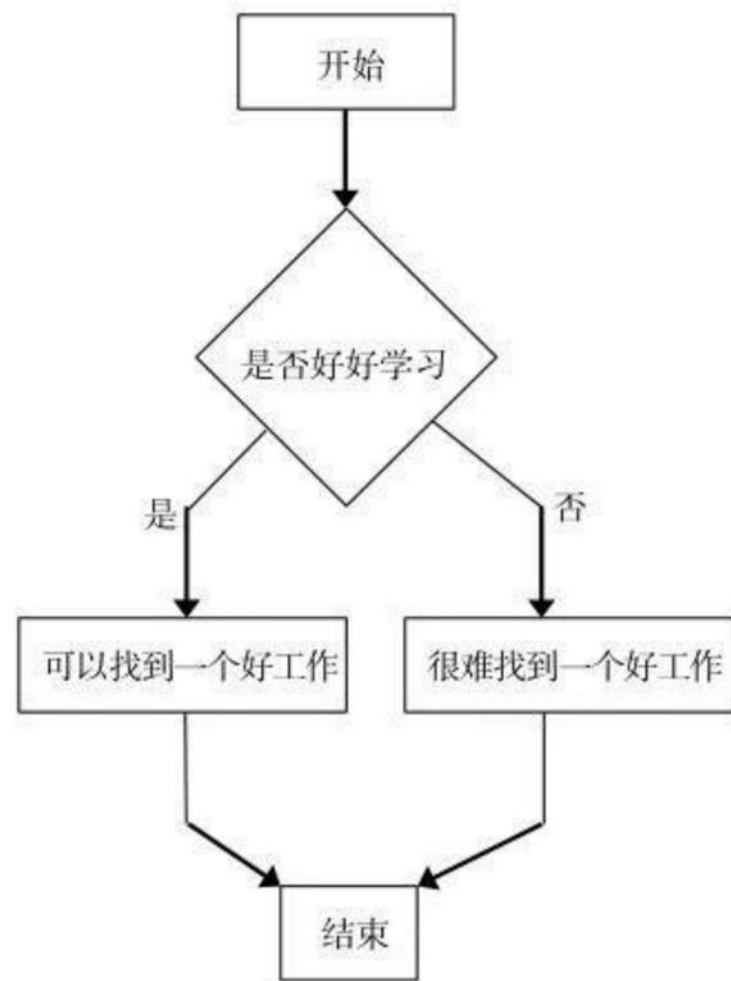
In [13]: `runcell(1, 'C:/Users/Administrator/Desktop/test/aaaa.py')`
找到了工作

In [14]: `runcell(2, 'C:/Users/Administrator/Desktop/test/aaaa.py')`



条件分支

`else`语句是`if`语句的补充，`if`条件只说明了当条件满足时程序做什么，没有说当条件不满足时程序做什么。而`else`语句正好是用来说明当条件不满足时，程序做什么。



条件分支

```
6 """
7 #%%
8 is_study=1
9 if is_study==1:
10     print("找到了工作")
11 else:
12     print("没有找到工作")
```

控制台 1/A

In [16]: runfile('C:/Users/Administrator/Desktop/
找到了工作

```
7 #%%
8 is_study=0
9 if is_study==1:
10     print("找到了工作")
11 else:
12     print("没有找到工作")
```

控制台 1/A

In [18]: runfile('C:/Users/Administrator/Des
没有找到工作

- `elif`语句可以近似理解成`else_if`，前面提到的`if`语句、`else`语句都只能对一条语句进行判断，但是当你需要读多条语句进行判断时，就可以用`elif`语句判断。
- `elif`中可以有`else`语句，也可以没有，但是必须有`if`语句，具体执行顺序是先判断`if`后面的条件是否满足，如果满足则运行`if`为真时的程序，结束循环；如果`if`条件不满足时就判断`elif`语句。可以有多个`elif`语句，但是只有0个或者1个`elif`语句会被执行。

条件分支

```
7  #%%  
8  number = 23  
9  guees = int(input("请输入一个整数: "))  
10 if number == guees:  
11     print("恭喜你猜对啦")  
12 elif number > guees:  
13     print("抱歉, 猜小了")  
14 else:  
15     print("抱歉, 猜大了")  
16  
17
```

控制台 1/A

```
In [10]: runfile('C:/Users/Administrator/Desktop/test/aaaa.py',
```

```
请输入一个整数: 50  
抱歉, 猜大了
```

```
In [11]: runfile('C:/Users/Administrator/Desktop/test/aaaa.py',
```

```
请输入一个整数: 20  
抱歉, 猜小了
```

```
In [12]: runfile('C:/Users/Administrator/Desktop/test/aaaa.py',
```

```
请输入一个整数: 23  
恭喜你猜对啦
```

条件分支

分类	国际BMI值	国内BMI值
偏瘦	<18.5	<18.5
正常	18.5 ~ 25	18.5 ~ 24
偏胖	25 ~ 30	24 ~ 28
肥胖	≥30	≥28

```
1  ###
2  height, weight = eval(input("请输入身高(米)和体重(公斤)[逗号隔开]: "))
3  bmi = weight / pow(height, 2)
4  print("BMI 数值为: {:.2f}".format(bmi))
5  who, nat = "", ""
6  if bmi < 18.5:
7      who, nat = "偏瘦", "偏瘦"
8  elif 18.5 <= bmi < 24:
9      who, nat = "正常", "正常"
10 elif 24 <= bmi < 25:
11     who, nat = "正常", "偏胖"
12 elif 25 <= bmi < 28:
13     who, nat = "偏胖", "偏胖"
14 elif 28 <= bmi < 30:
15     who, nat = "偏胖", "肥胖"
16 else:
17     who, nat = "肥胖", "肥胖"
18 print("BMI 指标为:国际'{0}', 国内'{1}'".format(who, nat))
```

控制台 1/A

Python 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Administrator/Desktop/test/aaaa.py', wdir='C:/Users/Admin

请输入身高(米)和体重(公斤)[逗号隔开]: 1.8,90

BMI 数值为: 27.78

BMI 指标为:国际'偏胖', 国内'偏胖'



三 循环



3.1. for循环

for循环可以用来遍历

- 字符串
- 元组
- 列表
- 字典的key
- value

```
7  """
8  str1 = "python is the best"
9  # list1 = [1, 1.36, 'a', 'python', (1,), [1, 2], {'age': 18}]
10 # tuple1 = (1, 1.36, 'a', 'python', (1,), [1, 2], {'age': 18})
11 # dict1 = {"Name": "yosef",
12 # "Sex": "man",
13 # "Age": 22,
14 # "City": "Shanghai"}
15 for a in str1:
16     print(a, end=" ")
17     print("\n")
18 # for b in list1:
19 #     print(b, end=" ")
20 #     print("\n")
21 # for c in tuple1:
22 #     print(c, end=" ")
23 #     print("\n")
24 # for d in dict1.values():
25 #     print(d, end=" ")
26 #     print("\n")
27 # for e in dict1.keys():
```

控制台 1/A

```
In [22]: runfile('C:/Users/Administrator/Desktop/test/aaaa.py', wdir='C:/Users/Administrator/Desktop/test')
p
y
t
h
o
n

i
s

t
h
e
```


3.2. range循环

- range函数的结果是序列
 - range(m,n,k) 在[m,n)中，步长为k的整数序列
 - range(m,n) 在[m,n)中，步长为1的整数序列
 - range(m) 在[0, m)中，步长为1的整数序列
-

3.2. range循环

```
7  ###
8  for a in range(2, 8, 2):
9      print(a)
10 print("\n")
11 for i in range(1,9):
12     print(i)
13     print("\n")
14 for j in range(10):
15     print(j)
16 ###
17 for a in range(2, 8, 2):
18     print(a)
19     print("\n")
20 for i in range(1,9):
21     print(i)
22     print("\n")
23 for j in range(10):
24     print(j)
```

```
控制台 1/A
In [31]: runcell(1, 'C:/Users/Administrator
2
4
6
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
9
```

```
In [32]: runcell(2, '
2
4
6
1
2
3
4
5
6
7
8
```

3.3. 嵌套for循环

```
7  #%%
8  list1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
9  for i in range(len(list1)):
10     for j in range(len(list1[i])):
11         print(list1[i][j])
```

控制台 1/A

In [35]: runfile('C:/Users/Administrator/Desktop/tes

```
1
2
3
4
5
6
7
8
9
```

```
5  @author: Administrator
6  """
7  #%%
8  list1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
9  for i in range(len(list1)):
10     for j in range(len(list1[i])):
11         print(list1[j][i])
```

控制台 1/A

In [38]: runfile('C:/Users/Administrator/Desktop/tes

```
1
4
7
2
5
8
3
6
9
```

3.4. while循环

while循环当条件为真时，会执行代码，为假时退出代码循环。

```
7  ###
8  a=1
9  while a<10:
10     print('a=',a,'a小于10，继续循环')
11     a=a+1
12     # a+=1
13     print('a足够大了，跳出循环')
```

控制台 1/A

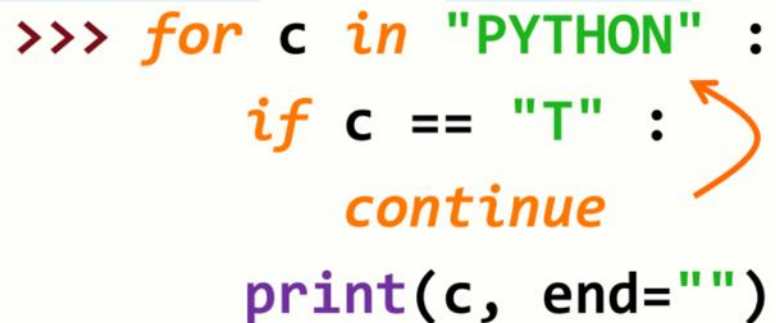
```
In [41]: runfile('C:/Users/Administrator/Desktop/test/aaaa
a= 1 a小于10，继续循环
a= 2 a小于10，继续循环
a= 3 a小于10，继续循环
a= 4 a小于10，继续循环
a= 5 a小于10，继续循环
a= 6 a小于10，继续循环
a= 7 a小于10，继续循环
a= 8 a小于10，继续循环
a= 9 a小于10，继续循环
a足够大了，跳出循环
```

3.4. while循环

while循环结合continue和break

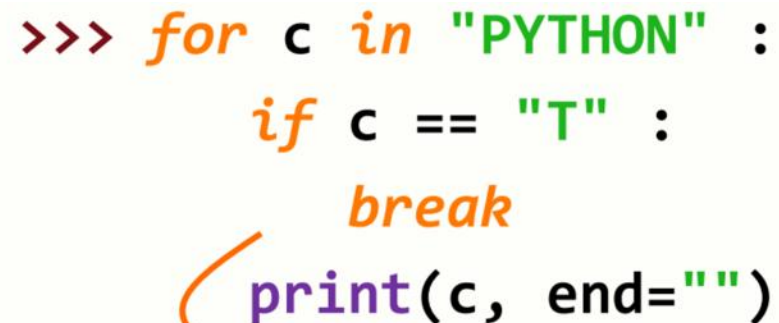
- break是直接跳出该层循环
- continue是直接进入该层循环的下一迭代

```
>>> for c in "PYTHON" :  
    if c == "T" :  
        continue  
    print(c, end="")
```



PYHON

```
>>> for c in "PYTHON" :  
    if c == "T" :  
        break  
    print(c, end="")
```



PY

3.4. while循环

while循环结合continue和break

- break是直接跳出该层循环
- continue是直接进入该层循环的下一次迭代

```
7  #%%
8  number = 56
9  while True:
10     guess = int(input("请输入一个整数: "))
11     if number == guess:
12         print("恭喜你猜对啦")
13         break
14     elif number > guess:
15         print("抱歉, 猜小了")
16         continue
17     else:
18         print("抱歉, 猜大了")
19         continue
```

控制台 1/A

```
In [45]: runfile('C:/Users/Administrator/Desktop/test/a
```

请输入一个整数: 60
抱歉, 猜大了

请输入一个整数: 40
抱歉, 猜小了

请输入一个整数: 50
抱歉, 猜小了

请输入一个整数: 55
抱歉, 猜小了

请输入一个整数: 56
恭喜你猜对啦



四 函数



函数

函数是一个程序中可以重复使用的一段代码。这段代码是由一块语句和一个名称组成的，只要函数定义好以后，你就可以在程序中通过该名字调用执行这段程序。实际上函数就是一个黑箱，用于实现某个功能的黑盒子。

4.1. 普通函数

普通函数一般由函数名（必需）、参数、语句块（必需）、`return`、变量这几部分组成。

```
def 函数名(参数):  
    语句块
```


4.1. 普通函数

定义函数使用的关键词是`def`，函数名后面的括号里面放参数（参数可以为空），参数后面要以冒号结尾，语句块要缩进四个空格，语句块是函数具体要做的事情。

函数在解释型语言里需要先声明后调用，不能更改顺序。

```
def 函数名(参数):  
    语句块
```

```
7  """  
8  def area(r):  
9      area=3.1415926*r**2  
10     return area  #带返回值的函数  
11  
12     def show(a,b):  
13         print('半径为',a,'m圆的面积为',b,'m^2')  
14         #无返回值的函数  
15  
16     A=area(5)  
17     print(A)  
18     show(5,A)
```

控制台 1/A

```
In [50]: runfile('C:/Users/Administrator/Desktop/test/aaa  
78.539815  
半径为 5 m圆的面积为 78.539815 m^2
```

4.1. 普通函数

需要注意的一点，全局变量和局部变量的关系。

- 局部变量是函数内部的占位符，与全局变量可能重名但不同
- 函数运算结束后，局部变量被释放
- 可以使用global保留函数内部使用全局变量

```
1  #%%1
2  n, s = 10, 100
3  def fact(n):
4      s = 1
5      for i in range(1, n+1):
6          s=s*i
7      return s
8  print(fact(n), s)
```

控制台 1/A

```
In [13]: runfile('C:/Users/Administrat
3628800 100
```

```
1  #%%1
2  n, s = 10, 100
3  def fact(n):
4      global s
5      s = 1
6      for i in range(1, n+1):
7          s=s*i
8      return s
9  print(fact(n), s)
```

控制台 1/A

```
In [16]: runfile('C:/Users/Administra
3628800 3628800
```

4.2. 匿名函数

匿名函数，顾名思义就是没有名字的函数，也就是省略了def定义函数的过程。

- 定义：使用【lambda + 参数 +表达式】的方式

lambda [arg1 [,arg2, ... argN]] : expression

- lambda用来表示匿名函数，可以传入多个参数，但只能有一个表达式。
-

4.2. 匿名函数

匿名函数，顾名思义就是没有名字的函数，也就是省略了def定义函数的过程。

lambda [arg1 [,arg2, ... argN]] : expression

```
6      """
7      #%%
8      # def area(r):
9      #     area=3.1415926*r**2
10     #     return area #带返回值的函数
11
12     # def show(a,b):
13     #     print('半径为',a,'m圆的面积为',b,'m^2')
14     #     #无返回值的函数
15
16     area=lambda r:3.1415926*r**2
17     A=area(5)
18     print(A)
19     |
```

控制台 1/A

```
In [54]: runfile('C:/Users/Administrator/Desktop/test/
78.539815
```

4.2. 匿名函数

优点:

- 不用取名称，因为给函数取名是比较头疼的一件事，特别是函数比较多的时候
- 可以直接在使用的地方定义，如果需要修改，直接找到修改即可，方便以后代码的维护工作
- 语法结构简单，不使用def 函数名(参数名):这种方式定义，直接使用lambda 参数:返回值 定义即可



五 模块



- 模块是升级版的函数，前面说过，在一段程序中可以通过函数名多次调用函数，但是必须在定义函数的这段程序里面调用，如果换到其他程序里该函数就不起作用了。
- 模块之所以是升级版的函数，是因为在任意程序中都可以通过模块名去调用该模块对应的程序。
- 你要调用函数首先需要定义一个函数，同理，你要调用模块，首先需要导入模块，导入模块的方法主要有两种。我们上节课已经说过了。

```
import module_name #直接 import 具体的模块名
```

```
from module1 import module2 #从一个较大的模块中 import 较小的一个模块
```

模块

```
aaaa.py x area.py x
1  # -*- coding: utf-8 -*-
2  """
3  Created on Thu Apr 29 18:31:04 2021
4
5  @author: Administrator
6  """
7  def area(r):
8      area=3.1415926*r**2
9      return area #带返回值的函数
```

```
15
16 import area
17 A=area.area(5)
18 print(A)
19
```

```
控制台 1/A x
In [58]: runfile('C:/Users/Admini
Reloaded modules: area
78.539815
```

```
15
16 import area1
17 A=area1.area(5)
18 print(A)
19
```

```
控制台 1/A x
In [62]: runfile('C:/Users/Administr
Reloaded modules: area1
78.539815
```




感谢参与 下堂课见

