Jupyter Notebook

Jupyter Notebook 是一款开放源代码的 Web 应用程序，可让我们创建并共享代码和文档。

它提供了一个环境，你可以在其中记录代码，运行代码，查看结果，可视化数据并在查看输出结果。这些特性使其成为一款执行端到端数据科学工作流程的便捷工具，可以用于数据清理，统计建模，构建和训练机器学习模型，可视化数据以及许多其他用途。

安装

如果没有安装Python，可以先安装Anaconda,Anaconda中自带有Python和Jupyter Notebooks,并且还包含了很多数据科学和机器学习中常用的包。

下载地址：https://www.anaconda.com/distribution/#download-section (https://www.anaconda.com/distribution/#download-section)

如果没有用Anaconda，可以直接用pip安装：

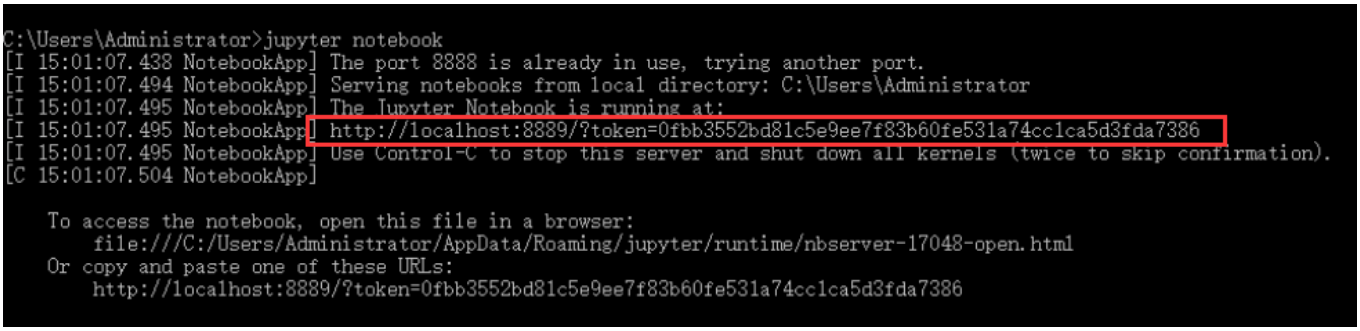
pip install jupyter

操作

要运行Jupyter Notebooks，可以直接在命令行中输入jupyter notebook

执行上面命令之后， Jupyter Notebook 将在你的默认浏览器中打开，网址为：http://localhost:8888/tree (http://localhost:8888/tree)

在某些情况下，它可能无法自动打开。这种情况下，你的终端或者命令提示符中将会生成一个带有令牌密钥（token key ）的网址。要打开 Notebook，你需要将整个 URL（包括令牌密钥）复制粘贴到浏览器中。



Notebook 打开后，你会在顶部看到三个选项卡：Files（文件），Running（运行）和 Clusters（集群）。Files 基本上列出了所有的文件，Running 显示你当前已经打开的终端和Notebooks，Clusters 由 IPython parallel 包提供，用于并行计算。

要打开一个新的 Jupyter Notebook，请单击页面右侧的“New”选项卡。在这里，你有四个选项可供选择：

Python 3： ipynb文件

Text File： 空白的文档

Folder： 创建文件夹

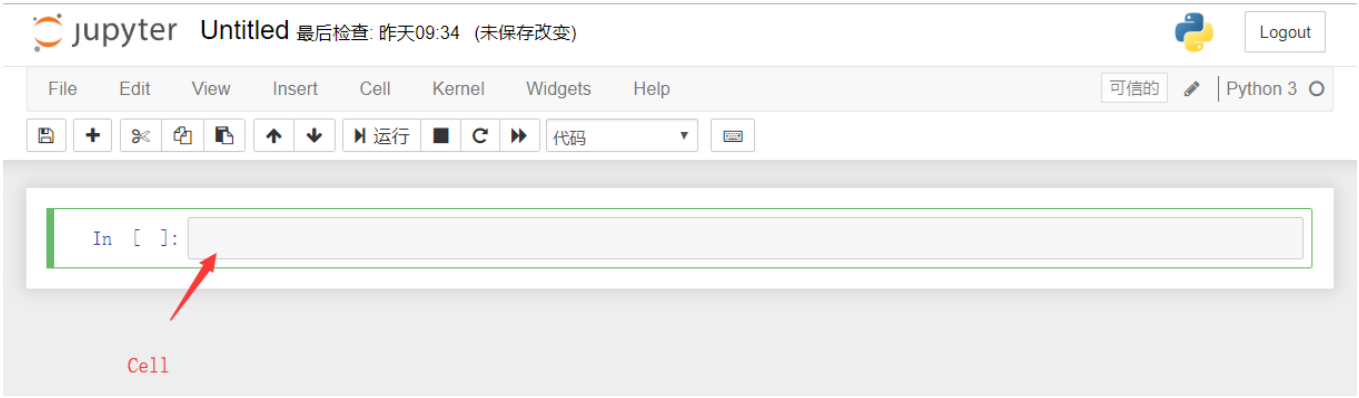
Terminal： 终端（Windows 上的 cmd）

要写python程序的话就New -> Python3

代码上方的菜单栏提供了操作单元格的各种选项：

插入、剪切、复制、粘贴、上下移动单元格、运行、停止以及重启服务等

你的代码是写入到独立的单元中并可以单独执行的，而无需从脚本的开始执行代码。



常用快捷键

Esc 和 Enter 在命令和编辑模式之间跳转。

命令模式下：

1、A 键将在选中单元格上方插入新单元格，B 键将在选中单元格下方插入一个单元格

2、按两次D键，可以删除单元格

3、Z键，撤消已删的但与前各

4、Y键，将选中的单元格变成代码单元格

5、F键、弹出'查找和替换'菜单

编辑模式下：

1、Shift+Enter : 运行本单元，并选中下个单元

2、Ctrl+Enter : 运行本单元

3、Alt+Enter : 运行本单元，在其下插入新单元

Python基础语法

缩进

空白在Python中是重要的。事实上行首的空白是重要的。它称为缩进。在逻辑行首的空白（空格和制表符）用来决定逻辑行的缩进层次，从而用来决定语句的分组。 这意味着同一层次的语句必须有相同的缩进。每一组这样的语句称为一个块。

注意：不要混合使用制表符和空格来缩进，因为这在跨越不同的平台的时候，无法正常工作。

In [1]:

```
if True:
    print ("True") #Tab键或者四个空格,不建议混用
else:
    print ("False")
```

注释

Python中单行注释以 # 开头 多行注释可以用多个 # 号，还有'''和 """，注释中的代码是不会执行的。

In [2]:

```
# 这是一个单行注释print(hello world)
'''
这是一个多行注释
print(hello world)
'''
print('hello world')
```

多行语句

Python 通常是一行写完一条语句，但如果语句很长，我们可以使用反斜杠(\ )来实现多行语句。

In [3]:

```
name = 'zhang \
san '
print(name)
```

在 [], {}, 或 () 中的多行语句，不需要使用反斜杠(\ )

数据类型

Python 中的变量不需要声明。每个变量在使用前都必须赋值，变量赋值以后该变量才会被创建。

在 Python 中，变量就是变量，它没有类型，我们所说的"类型"是变量所指的内存中对象的类型。

等号（=）用来给变量赋值。

等号（=）运算符左边是一个变量名,等号（=）运算符右边是存储在变量中的值。

In [4]:

```
a = 1           #整型
b = 1.1         #浮点型
c = 'zhangsan' #字符串
```

python也允许同时为多个变量赋值

In [5]:

```
a = b = c = 1
print(a,b,c)
a, b, c = 1, 2, 3
print(a,b,c)
```

number（数字类型）

Python3 支持 int、float、bool、complex（复数）。

整数(int):

Python可以处理任意大小的整数，当然包括负整数，在程序中的表示方法和数学上的写法一模一样，例如：100，-100，0，等等。

In [6]:

```
a = 1
b = -100
c = 0
print(type(a),type(b),type(c))
```

浮点数(float):

浮点数也就是小数，之所以称为浮点数，是因为按照科学记数法表示时，一个浮点数的小数点位置是可变的，比如，1.23x109和12.3x108是完全相等的。浮点数可以用数学写法，如 3.14,-9.9等，也可以用科学计数法来写，用e代替10，比如3.14e3。

In [7]:

```
a = 3.14
b = - 9.9
c = 3.14e3
print(type(a),type(b),type(c))
```

复数(complex):

复数很少用到，与数学中的表达式一样，比如：4+3j

In [8]:

```
a = 4 + 3j
print(type(a))
```

布尔值:

布尔值和布尔代数的表示完全一致，一个布尔值只有True、False两种值，要么是True，要么是False，在Python中，可以直接用True、False表示布尔值（请注意大小写）.

In [9]:

```
a = True
b = False
print(type(a),type(b))
```

算术运算符

In [10]:

```
a = 10
b = 8
#加法运算：+
c = a + b
print('a + b的值为：',c)
#减法：-
c = a - b
print('a - b的值为：',c)
#乘法：*
c = a * b
print('a * b的值为：',c)
#除法：/
c = a / b
print('a / b的值为：',c)
#取余数：%
c = a % b
print('a % b的值为：',c)
#取整数：//
c = a // b
print('a // b的值为：',c)
#幂运算：**
c = a ** 3
print('a 的三次方的值为：',c)
```

比较运算符

In [11]:

```
a = 10
b = 8
#等于：==
#注意是两个等号，一个=号是赋值
print(a == b)
```

In [12]:

```
#!=：不等于
print(a != b)
```

In [13]:

```
#>：大于
print(a > b)
#<：小于
print(a < b)
```

In [14]:

```
#>=：大于等于
print(a >= b)
#<=：小于等于
print(a <= b)
```

赋值运算符

In [15]:

```
a = 10
b = 8
a += b   #等效于 a = a + b
print('a += b的值为：',a)
a -= b   #等效于 a = a - b
print('a -= b的值为：',a)
a *= b   #等效于 a = a * b
print('a *= b的值为：',a)
a /= b   #等效于 a = a / b
print('a /= b的值为：',a)
a %= b   #等效于 a = a % b
print('a %= b的值为：',a)
a //= b   #等效于 a = a // b
print('a //= b的值为：',a)
a **= b   #等效于 a = a ** b
print('a **= b的值为：',a)
```

string(字符串)

字符串是 Python 中最常用的数据类型。我们可以使用引号(' 或 ")来创建字符串。

创建字符串很简单，只要为变量分配一个值即可

In [16]:

```
a = 'hello world'
print(a)
```

字符串截取

In [17]:

```
print(a[1])      #取第二个值
print(a[-1])     #取最后一个值
print(a[0:5])    #a中的第一个到第五个，索引是从0-4
print(a[:5])     #截取到第五个
print(a[::2])    #第二个冒号后面为步长，2表示没间隔一个就区一个值
print(a[::-1])   #反转字符串
```

字符串运算符

In [18]:

```
# +,字符串相加，实际上是字符串进行拼接
a = 'hello '
b = 'world'
print(a + b)
# print(a + 1)  当字符串与数字类型相加时，会出现错误
#TypeError: can only concatenate str (not "int") to str

# *, *多少就重复多少次
print(a * 3)   #重复三次
```

list(列表)

Python内置的一种数据类型是列表：list。list是一种有序的集合，可以随时添加和删除其中的元素。

创建一个列表，只要把逗号分隔的不同的数据项使用方括号括起来即可。

In [19]:

```
a = ['zhangsan','lisi','wangwu']
b = [1, 2, 3]
```

In [20]:

```
#列表也可以进行嵌套
c = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(c)
```

列表取值

与字符串的索引一样，列表索引从0开始。列表可以进行截取、组合等。

In [21]:

```
print(a[0])      #取出列表中的第一个值
print(a[-1])     #取出列表中的最后一个值
print(a[::-1])   #反正列表

#对于嵌套的列表的取值
print(c[1][1])
print(c[0][2])
```

更新列表

In [22]:

```
#append方法：向列表末尾加入一个元素
a.append('xiaoming')
print(a)
```

In [23]:

```
#insert方法：向列表指定位置插入一个元素
#list.insert(index, obj)
b.insert(1,'123')     #索引为1的位置插入123
print(b)
```

In [24]:

```
#pop方法：删除list末尾的元素
b.pop()
print(b)
```

```
[1, '123', 2]
```

In [25]:

```
#remove方法：用于移除列表中某个值的第一个匹配项。
#list.remove(obj)

b.remove(1)
print(b)
```

```
['123', 2]
```

tuple(元组)

tuple和list非常类似，但是tuple一旦初始化就不能修改。

元组创建只需要在括号中添加元素，并使用逗号隔开即可。

In [26]:

```
a = ('zhangsan', 'lisi', 'wangwu')
#获取元素的方法和list是一样
print(a[0])
print(a[-1])
print(a[::-1])
```

```
zhangsan
wangwu
('wangwu', 'lisi', 'zhangsan')
```

In [27]:

```
#注意：创建元组时，当元组内只有一个元素时，需要写一个（，），否则会被当做运算符使用。
a = (50)
b = (50,)
print(type(a))
print(type(b))
```

```
<class 'int'>
<class 'tuple'>
```

删除元组

元组中的元素值是不允许删除的，但我们可以使用del语句来删除整个元组。

In [28]:

```
a = (1, 2, 3)
print(a)
del a
# print(a)   #删除元组后在访问它，程序会报错
```

```
(1, 2, 3)
```

dict(字典)

字典是另一种可变容器模型，且可存储任意类型对象。

字典的每个键值对用冒号(:)分割，每个对之间用逗号(,)分割，整个字典包括在花括号({})中 。

In [29]:

```
a = {'name':'zhangsan', 'age' : 18}
print(a['name'])
print(a['age'])
```

```
zhangsan
18
```

修改字典

In [30]:

```
#直接重新赋值
a['age'] = 12
print(a)
```

```
{'name': 'zhangsan', 'age': 12}
```

删除字典

In [31]:

```
del a['name']
print(a)

#clear(方法):清空字典
a.clear()
print(a)
```

```
{'age': 12}
{}
```

注意：由于一个key只能对应一个value，所以，多次对一个key放入value，后面的值会把前面的值覆盖掉

In [32]:

```
a = {'name':'zhangsan', 'age':18, 'name':'lisi'}
print(a)
```

```
{'name': 'lisi', 'age': 18}
```

与list相比较：

字典的查找速度和插入速度快，不会随着数据量的增多而减慢。 但是，字典需要占用较多的内存，是一种以空间换时间的方法。

set(集合)

集合（set）是一个无序的不重复元素序列。

可以使用大括号 { } 或者 set() 函数创建集合。

注意：创建一个空集合必须用 set() 而不是 { }，因为 { } 是用来创建一个空字典。

In [33]:

```
a = {'a', 'b', 'c', 'd'}
print(a)
b = set([1, 2, 3, 4])
print(type(b))
```

```
{'c', 'a', 'b', 'd'}
<class 'set'>
```

In [34]:

```
#重复的元素在集合中会被自动过滤掉
a = {'a', 'b', 'c', 'd', 'b', 'b'}
print(a)
```

{'c', 'a', 'b', 'd'}

添加元素

In [35]:

```
#add方法
a = {'a', 'b', 'c', 'd'}
a.add(1)
print(a)
```

{1, 'c', 'a', 'b', 'd'}

删除元素

In [36]:

```
#remove方法
a.remove('a')
print(a)
```

{1, 'c', 'b', 'd'}

数据类型转换

python内置的数据类型进行转换时，可以使用内置函数。

In [37]:

```
#字符串转数字
int("8")
float('2')
#文字不能转数字
# int('a')

#数字转字符串
str(1)

#tuple()参数可以是元组、列表或者字典
tuple([1,2,3])

# list()将序列转变成一个列表，参数可为元组、字典、列表
list((1,2,3))

# set()将一个可以迭代对象转变为可变集合，并且去重复
set([1,1,2,3])
#set,可以用来去重
```

Out[37]:

```
{1, 2, 3}
```

回顾一下

Python中的标准数据类型：

Number（数字）

String（字符串）

List（列表）

Tuple（元组）

Set（集合）

Dictionary（字典）

不可变数据（3 个）：Number（数字）、String（字符串）、Tuple（元组）；

可变数据（3 个）：List（列表）、Dictionary（字典）、Set（集合）。

In [38]:

```
#id() 函数用于获取对象的内存地址。
#不可变类型:
#对不可变类型的变量重新赋值，实际上是重新创建一个不可变类型的对象，并将原来的变量重新指向新创建的对象。
a = 1
print(id(a))
a += 1
print(id(a))
```

```
1368293488
1368293520
```

In [39]:

```
#可变类型
b = [1, 2, 3]
print(id(b))
b.append(4)
print(id(b))
```

```
2317878785160
2317878785160
```

注意：在写程序的过程当中，不能把关键字用作任何标识符名称。

In [40]:

```
#查看当前版本有哪些关键字
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'de
f', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'im
port', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield']
```

条件控制语句

Python 条件语句是通过一条或多条语句的执行结果（True 或者 False）来决定执行的代码块。

if语句

In [41]:

```
'''
if 条件1:
    代码  #满足条件则执行的代码
else:
    代码  #不满足条件时的代码
'''
age = 15
if age >= 18:
    print('成年了')
else:
    print('未成年')
```

```
未成年
```

In [42]:

```
'''
if 条件1:
    代码  #满足条件1则执行的代码
elif 条件2:
    代码  #满足条件2时执行的代码
else:
    代码  #条件1，条件2都不满足时的代码
'''
score = 51
if score >= 80:
    print('优秀')
elif score >= 60:
    print('良好')
else:
    print('不及格')
```

不及格

练习：有三个整数x、y、z，请把这三个整数由小到大输出

思路：做一个简单的排序，将最小值放在x,中间的值放在y,最大值放在z。

先比较x与y的值，如果x的值较大，则互换x和y的位置。

再比较x与z的值，如果还是x比较大，则互换位置，此时x已经是最小的值

接着在比较y与z的值，则完成了一次简单的排序。

In [43]:

```
#假如x=5 y=6 z=2
x = 5
y = 6
z = 2
if x > y:
    x, y = y, x     #如果x比y大，则x，y互换位置,如果x比y小,则保持原样不动
if x > z:
    x, z = z, x
if y > z:
    y, z = z, y
print(x,y,z)
```

2 5 6

循环语句

Python中的循环语句有 for 和 while。 当我们要计算1+2+3...+99+100时，可以使用等差数列的求和公式计算，也可以使用循环语句进行计算。

for循环

In [44]:

```
num = 0
for i in range(1,101):
    num += i
print(num)
#range(1, 101)就是1,2,3,4,5.....101,i第一次是1，执行完一次后，i就赋值为2，重新执行代码，一直到100
#注意，range(1,101)，i取不到101，只取得到100
```

5050

while循环

In [45]:

```
'''
while 判断条件：

    语句
'''
num = 0
i = 1
while i < 101:
    num += i
    i += 1      #注意跳出循环的条件，不要写死循环
print(num)
```

5050

当程序错写成死循环时，可以按CTRL+C来退出。

break和continue语句

In [46]:

```
#break 语句可以跳出 for 和 while 的循环体。
num = 0
for i in range(1,11):
    num += i
    if i == 5:  #加到5时，跳出循环
        break
print(num)

num = 0
i = 1
while i < 11:
    num += i
    if i == 5:
        break
    i += 1
print(num)
```

15
15

In [47]:

```
#continue语句被用来告诉Python跳过当前循环块中的剩余语句，然后继续进行下一轮循环。
num = 0
for i in range(1,11):
    if i % 2 == 0:    #如果i为偶数，则跳过
        continue
    else:
        num += i
print(num)

num = 0
i = 1
while i < 11:
    if i % 2 == 0:
        i += 1      #注意改变条件，如果没有改变i，则会进入死循环
        continue
    else:
        num += i
        i += 1
print(num)
```

```
25
25
```

pass 语句

pass是空语句，是为了保持程序结构的完整性。

pass 不做任何事情，一般用做占位语句。

In [48]:

```
students = ['zhangsan', 'lisi', 'wangwu', 'xiaohong', 'xiaoming']
for student in students:
    if student == 'xiaohong':
        pass                #如果名字为xiaohong，则不执行语句
    else:
        print(student)
```

```
zhangsan
lisi
wangwu
xiaoming
```

练习：输出99乘法表

思路：分行和列考虑，使用循环嵌套，用i控制行，j来控制列

In [49]:

```
for i in range(1, 10):
    print()           #这个print用来换行
    for j in range(1, i + 1):
        print('{}*{}={}   '.format(j, i, i*j), end = '')   #end='', 让print不换行
```

```
1*1=1
1*2=2   2*2=4
1*3=3   2*3=6   3*3=9
1*4=4   2*4=8   3*4=12   4*4=16
1*5=5   2*5=10   3*5=15   4*5=20   5*5=25
1*6=6   2*6=12   3*6=18   4*6=24   5*6=30   6*6=36
1*7=7   2*7=14   3*7=21   4*7=28   5*7=35   6*7=42   7*7=49
1*8=8   2*8=16   3*8=24   4*8=32   5*8=40   6*8=48   7*8=56   8*8=64
1*9=9   2*9=18   3*9=27   4*9=36   5*9=45   6*9=54   7*9=63   8*9=72   9*9=81
```

函数

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。

函数能提高应用的模块性，和代码的重复利用率。你已经知道Python提供了许多内建函数，比如print()。但你也可以自己创建函数，这被叫做用户自定义函数。

你可以定义一个由自己想要功能的函数，以下是简单的规则：

函数代码块以 def 关键词开头，后接函数标识符名称和圆括号 ()。

任何传入参数和自变量必须放在圆括号中间，圆括号之间可以用于定义参数。

函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。

函数内容以冒号起始，并且缩进。

return [表达式] 结束函数，选择性地返回一个值给调用方。不带表达式的return相当于返回 None。

In [50]:

```
'''
def 函数名（参数列表）：
    函数体
'''


def hello():
    print('hello world')
    #没有return，默认为return None
hello()      #调用函数
```

```
hello world
```

In [51]:

```
#带参数的函数
def area(x,y):
    '''
    这是一个计算矩形面积的函数,x，y分别为矩形的边长。
    '''
    return x*y      #返回x*y的值

print(area(3,4))    #计算一个边长为3和4的矩形的面积
print(area(5,5))    #计算一个边长为5的正方形的面积
#一个函数可以多次调用，根据传入值得不同，返回不同的结果。
```

```
12
25
```

参数

位置参数

必需参数须以正确的顺序传入函数。调用时的数量必须和声明时的一样。

In [52]:

```
def information(name,age):
    print('名字:',name)
    print('年龄:',age)
information('张三',12)
# information('张三')   #只传一个参数时，程序会报错
```

```
名字: 张三
年龄: 12
```

Python 解释器能够用参数名匹配参数值。

In [53]:

```
def information(name,age):
    print('名字:',name)
    print('年龄:',age)
information(age = 12, name = '张三')
```

```
名字: 张三
年龄: 12
```

默认参数

在创建函数时，定义好默认参数，在调用时如果没有传入，则使用默认的参数进行运算。

In [54]:

```
def information(name,age = 18):
    print('名字:',name)
    print('年龄:',age)
information(age = 12, name = '张三')
information(name = '李四')
```

```
名字: 张三
年龄: 12
名字: 李四
年龄: 18
```

可变参数

可变参数就是传入的参数个数是可变的，可以是1个、2个到任意个，还可以是0个。

加了星号 * 的参数会以元组(tuple)的形式导入，存放所有未命名的变量参数。

In [55]:

```
def calc(*args):
    sum = 0
    for n in args:
        sum = sum + n
    return sum
print(calc(1,2,3))
print(calc())
```

```
6
0
```

还有一种就是参数带两个星号 ,**加了两个星号** 的参数会以字典的形式导入。

In [56]:

```
def info(a, **kw):
    print(a)
    print(kw)
info(1, name = '张三',age = 12)
```

```
1
{'name': '张三', 'age': 12}
```

匿名函数

python 使用 lambda 来创建匿名函数。

所谓匿名，意即不再使用 def 语句这样标准的形式定义一个函数。

In [57]:

```
sum1 = lambda x,y : x + y
print(sum1(1,2))
```

```
3
```

全局变量和局部变量

定义在函数内部的变量拥有一个局部作用域，定义在函数外的拥有全局作用域。

局部变量只能在其被声明的函数内部访问，而全局变量可以在整个程序范围内访问。调用函数时，所有在函数内声明的变量名称都将被加入到作用域中。

In [58]:

```
num = 0
def sum1(x, y):
    num = x + y
    print('函数内的局部变量为:',num)
sum1(2,3)
print('函数外的全局变量为:',num)
```

函数内的局部变量为: 5
函数外的全局变量为: 0

global关键字

当内部作用域想修改外部作用域的变量时，就要用到global关键字。

In [59]:

```
num = 0
def sum1(x, y):
    global num
    print(num)
    num = x + y
    print(num)
sum1(2,3)
print('函数外输出num为:',num)
```

0
5
函数外输出num为: 5

练习：编写一个函数isPrimer()，用于判断传入的参数是否为素数。

素数定义为在大于1的自然数中，除了1和它本身以外不再有其他因数。

In [60]:

```
def isPrimer(num):
    for i in range(2, num):
        if num % i == 0:
            print('不是素数')
            return None        #判断出是素数后，要终止循环，使用return None可以直接退出函数
    print('是素数')
    return None
isPrimer(11)
```

是素数

递归函数

在函数内部，可以调用其他函数。如果一个函数在内部调用自身本身，这个函数就是递归函数。

例子：用递归函数的方式来计算阶乘n！=1 2 3 *...n,用函数fact(n)表示。

可以看出:fact(n) = n! = n * (n-1)!

In [61]:

```
def fact(n):
    if n == 1:
        return 1              #注意递归结束的条件
    return n * fact(n - 1)
fact(5)
# 5 * fact(4)
# 5 * 4 * fact(3)
# 5 * 4 * 3 * fact(2)
# 5 * 4 * 3 * 2 * fact(1)
# 5 * 4 * 3 * 2 * 1
```

Out[61]:

120

Python异常处理

异常即是一个事件，该事件会在程序执行过程中发生，影响了程序的正常执行。

一般情况下，在Python无法正常处理程序时就会发生一个异常。

异常是Python对象，表示一个错误。

当Python脚本发生异常时我们需要捕获处理它，否则程序会终止执行。

捕捉异常可以使用try/except语句。

In [1]:

```
#语法
'''
try:
    代码  #要执行的代码
except:
    代码  #如果try中的代码出现错误，则执行except中的代码，可以在except后面加上错误的类型，如果引发此类异常，则执行except
finally:
    代码  #不论程序出没出错都要执行
'''


a = 1
b = '1'
try:
    print(a+b)
except TypeError:
    print('类型错误')
finally:
    print('结束')

a = 1
b = '1'
try:
    print(a+b)
except:
    print('错误')
finally:
    print('结束')
```

```
类型错误
结束
错误
结束
```

Python文件读写

读写文件是最常见的IO操作。Python内置了读写文件的函数，用法和C是兼容的。

读写文件前，我们先必须了解一下，在磁盘上读写文件的功能都是由操作系统提供的，现代操作系统不允许普通的程序直接操作磁盘，所以，读写文件就是请求操作系统打开一个文件对象（通常称为文件描述符），然后，通过操作系统提供的接口从这个文件对象中读取数据（读文件），或者把数据写入这个文件对象（写文件）。

读文件

Python内置的open()函数可以打开一个文件。

In [2]:

```
#打开文件，'r'表示只读模式
#注意：要读的文件与py文件不再同一目录下时，路径要写绝对路径，在同一目录下时，可以只写名字
f = open('1.txt', 'r')

#读取内容，read()
print(f.read())

#关闭文件
f.close()

#f = open('2.txt', 'r')
#如果文件不存在，抛出一个IOError的错误
#FileNotFoundError                         Traceback (most recent call last)
#<ipython-input-2-fc64edc67529> in <module>
#----> f = open('2.txt', 'r')
#
#FileNotFoundError: [Errno 2] No such file or directory: '2.txt'
```

```
nice to meet youhello
```

In [3]:

```
#由于文件的读写可能产生错误，导致文件不能正常关闭，为了保证不出现文件没关闭的情况，可以使用try...except来处理。
try:
    f = open('1.txt', 'r')
    print(f.read())
except:
    print('文件读取出错')
finally:
    if f:
        f.close()
```

```
nice to meet youhello
```

In [4]:

```
#每次都这样写可能会有点繁琐，python引入了with语句来自动帮我们调用close()方法
#as用来取别名
with open('1.txt', 'r') as f:
    print(f.read())
```

```
nice to meet youhello
```

调用read()会一次性读取文件的全部内容，如果文件较大的话，会出现内存不够的情况，所以保险起见，可以使用read(size)的方法，每次读取一定大小的数据。也可以使用readline()，每次读取一行内容。

In [ ]:

```
with open('1.txt', 'r') as f:
    for line in f.readlines():
        print(line)
```

其他模式：

r :以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。

rb :以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。

r+ :打开一个文件用于读写。文件指针将会放在文件的开头。 rb+ :以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。

w :打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。 如果该文件不存在，创建新文件。

wb :以二进制格式打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。 如果该文件不存在，创建新文件。

w+ :打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。 如果该文件不存在，创建新文件。

wb+ :以二进制格式打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。 如果该文件不存在，创建新文件。

a :打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。 也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。

ab :以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。 也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。

a+ :打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。

ab+ :以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。 如果该文件不存在，创建新文件用于读写。

写文件

写文件和读文件是一样的，唯一区别是调用open()函数时，传入标识符'w'或者'wb'表示写文本文件或写二进制文件。

In [8]:

```
#w表示从开头开始编辑，即原有内容会被删除
f = open('1.txt', 'w')
f.write('nice to meet you')
f.close()
```

In [9]:

```
#a:追加
with open('1.txt', 'a') as f:
    f.write('hello')
```

In [ ]:

```

```