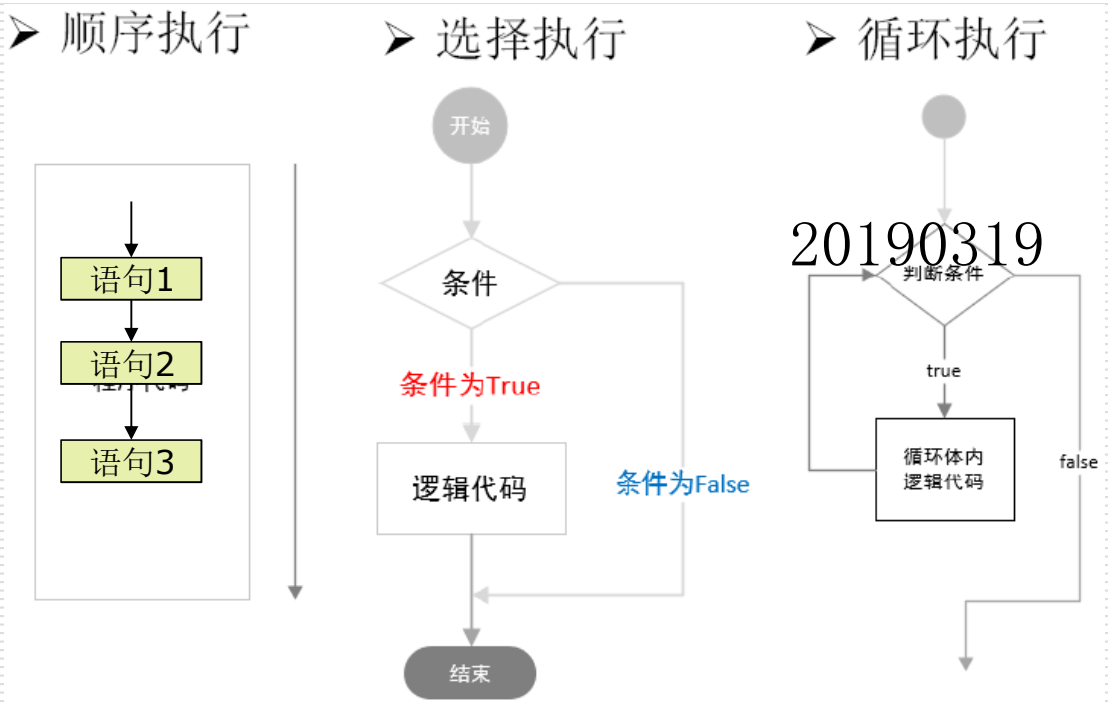
第4讲 Python Basics(2)

- 程序流程控制
- 组合数据类型
- 函数与类
- 文件操作

20190319

程序流程控制


顺序执行
选择执行
循环执行
语句1
语句2
语句3
开始
条件
条件为True
逻辑代码
条件为False
结束
20190319
判断条件
true
循环体内
逻辑代码
false

程序顺序结构

- 例5：圆面积和周长的计算

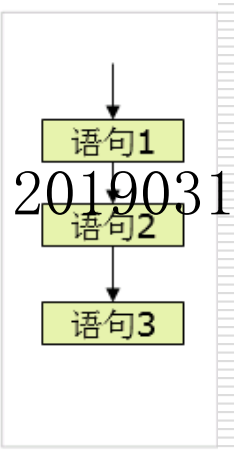
  输入：圆半径R

  处理：

  圆面积：S=π*R*R

  圆周长：L=2*π*R

  输出：圆面积S、周长L



20190319

```
1  R = eval(input("请输入圆半径:"))
2  S = 3.1415*R*R
3  L = 2*3.1415*R
4  print("面积和周长:",S,L)
```

程序分枝结构

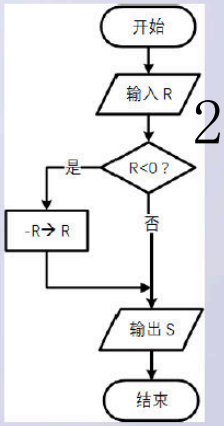
- 例6：实数绝对值的计算

输入：实数R

处理：$|R| = \begin{cases} R & R \geq 0 \\ -R & R < 0 \end{cases}$

输出：输出|R|

(a) 问题IPO描述



(b) 流程图描述

20190319

```
1 R = eval(input("输入实数:"))
2 if (R < 0):
3 R = -R
4 print("绝对值",R)
```

(c) Python代码描述

程序循环结构

- 例7：整数累加

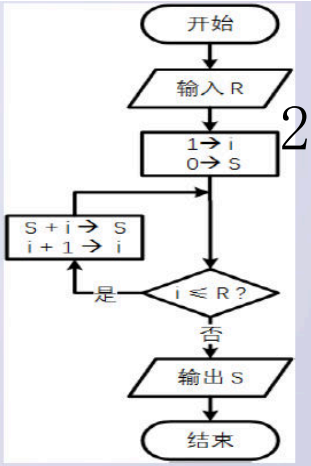
输入：正整数R

处理：

S=1+2+3+…+R

输出：输出S

(a) 问题IPO描述



(b) 流程图描述

20190319

```
1 R = eval(input("请输入正整数:"))
2 i, S = 0, 0
3 while (i<=R):
4 S = S + i
5 i = i + 1
6 print("累加求和",S)
```

(c) Python代码描述

程序描述方式

- IPO描述：主要用于区分程序输入输出关系，重点在于结构划分，算法主要采用自然语言描述
- 流程图描述：侧重于描述算法的具体流程关系，流程图的结构化关系相比自然语言描述更进一步，有助于阐述算法的具体操作过程
- Python代码描述：最终的程序产出，最为细致

条件语句

- 单分枝：if 语句

```
if  expression:
    expr_true_suit
```

- 二分枝结构：if–else 语句

```
if  expression:
    expr_true_suit
else:
    expr_false_suit
```

- 多分析结构：if–elif-else 语句

```
if expression1:
    expr_true_suit1
elif expression1:
    expr_true_suit2
…
else:
    expr_true_suitN
```

20190319

```
In [2]: #单分枝if语句
        weight= eval(input("请输入您的体重(kg)："))
        if weight > 90:
            print("胖子，该减肥啦！！！")
请输入您的体重(kg)：50

In [6]: #二分枝if -- else
        weight= eval(input("请输入您的体重(kg)："))
        if weight > 90:
            print("该减肥啦！！！")
        else:
            print("身材不错嘛！！！")
请输入您的体重(kg)：50
身材不错嘛！！！
```

实例8 计算身体质量指数BMI

- 编写一个根据体重和身高计算BMI(Body Mass Index)值的程序，并同时输出国际和国内的BMI指标建议值

分类	国际BMI值（kg/m^2）	国内BMI值（kg/m^2）
偏瘦	< 18.5	< 18.5
正常	18.5 ~ 25	18.5 ~ 24
偏胖	25 ~ 30	24 ~ 28
肥胖	>= 30	>= 28

20190319

- 计算公式：$BMI = weight(kg) / (height)^2\ (m^2)$

实例8 身体质量指数BMI

```
In [2]: #多分枝if -- elif -- else
        #例8：计算身体质量指数BMI
        height, weight= eval(input("请输入身高(米)和体重(千克)[用逗号隔开]："))
        bmi = weight / (height**2)
        print("您的BMI指数为：{:.2f}".format(bmi))
        nat, dom = "",""
        if bmi < 18.5:
            nat,dom = "偏瘦", "偏瘦"
        elif 18.5 <= bmi < 24:
            nat,dom = "正常","正常"
        elif 24 <= bmi < 25:
            nat,dom = "正常","偏胖"
        elif 25 <= bmi < 28:
            nat,dom = "偏胖","偏胖"
        elif 28 <= bmi < 30:
            nat, dom = "偏胖","肥胖"
        else:
            nat,dom = "肥胖","肥胖"
        print("您的国际BMI指标{},国内BMI指标{}".format(nat,dom))
```

```
请输入身高(米)和体重(千克)[用逗号隔开]：1.57,54
您的BMI指数为：21.91
您的国际BMI指标正常,国内BMI指标正常
```

20190319

while 循环语句

- while 语句：条件循环

```
while expression:
        suit_to_repeat1
else:
        suit_to_repeat2
```

20190319

In [8]:
```
'''
while循环,应用场景-对于满足某种条件，重复调用实现相同功能的代码
'''
#实例1：输出10以内的偶数
num = 2
while num <= 10:
    print(num)
    num+= 2
```

```
2
4
6
8
10
```

In [9]:
```
'''
While 条件：
……
else:
……
'''
m=1
while m<10:
    print(m)
    m+=1
else:
    print("finish!")
```

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

for循环语句

- **for 语句：遍历循环**

  for iter_var in sequence:
  suit_to _repeat

```
In [1]: '''
        for循环
        '''
        #例：打印0到9十个数字
        #range(0,10)：0,1,2,3,4,5,6,7,8,9
        for i in range(0,10):
            print(i,end=" ")

        0 1 2 3 4 5 6 7 8 9

In [2]: #例：打印0到9的偶数
        #range(0,10,2)：0,2,4,6,8
        for i in range(0,10,2):
            print(i,end=" ")

        0 2 4 6 8
```

20190319

多重循环

In [45]:
```
# 9*9乘法口诀表

for i in range(1, 10):
    for j in range(1, i + 1):
        result = j * i
        print('%s x %s = %-5s' % (j, i, result), end='' )
    print()
```

20190319

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

循环保留字

- **break：终止循环，在多重循环中仅停止执行本层的循环**
- **continue：逃过当前循环的剩余语句，然后继续执行下一轮循环**
- **pass：空语句，用作占位，为保持程序结构的完整性**

20190319

break语句

- break终止循环语句，在多重循环中仅终止执行本层循环

```
In [39]: # 统计第1个9在2的100次方中出现的位置:

         num = 2**100
         pos = 0
         for digit in str(num):
             pos = pos + 1
             if digit == "9":
                 break
         print("2**100 is: %d \nthe first posion of 9 is Pos.%d" %(num,pos))
```

20190319

```
2**100 is: 1267650600228229401496703205376
the first posion of 9 is Pos.16
```

continue语句

continue 语句用来跳过当前循环的剩余语句，然后继续进行下一轮循环

In [47]:
```
# 求在2的100次方中删除所有的9后的数字:

num = 2**100
without9 = ''
for digit in str(num):
    if digit == "9":
        continue
    without9 += digit
print("2**100 is: %d \nwithout 9 is :%s" %(num,without9))
```

20190319

```
2**100 is: 1267650600228229401496703205376
without 9 is :126765060022822401467032053 76
```

pass语句

pass 语句是空语句，是为了保持程序结构的完整性，一般用做占位语句

In [48]:
```
# 求在2的100次方中删除所有的9后的数字:

num = 2**100
without9 = ''
for digit in str(num):
    if digit == "9":
        pass
    else:
        without9 += digit
print("2**100 is: %d \nwithout 9 is :%s" %(num,without9))
```

20190319

```
2**100 is: 1267650600228229401496703205376
without 9 is :126765060022822401467032053 76
```

程序异常处理

- 异常：程序所要求执行的操作无法完成
- 异常处理：程序出现错误而在正常控制流外采取的行为
- Python中常见异常
  - NameError: 尝试访问一个未申明的变量
  - ZeroDivisionError: 除数为0
  - SyntaxError: Python解释器语法错误
  - IndexError: 请求的索引超出序列范围
  - KeyError: 请求一个不存在的字典关键字
  - IOError: 输入/输出错误
  - AttributeError: 尝试访问未知的对象属性

程序异常处理

- try-except代码块提供异常处理功能

```
In [20]: #程序的异常处理
         try:
             print(name) #打印变量name的值

           File "<ipython-input-20-cb53aca2a079>", line 3
             print(name) #打印变量name的值

         SyntaxError: unexpected EOF while parsing
```

20190319

```
In [21]: #程序的异常处理
         try:
             print(name) #打印变量name的值
         except Exception: #如果有异常则输出告警 EXception可以捕获python中任意异常，它属于异常基类
             print("Error !")

         Error !
```

程序异常处理

- try-except支持多个except语句捕获多个异常
- try-except与else和finally保留字配合使用

[1]:
```
#程序的异常处理
#定义一个列表，长度为2
namlist = ['test1','test2']
try:
    #namlist[2]很明显下标越界
    print(namlist[2])
except NameError as error:
    print("NameError:",error)
except IndexError as error:
    print("IndexError:",error)
except Exception as error:
    print("Exception:",error)
else:
    print("对try语句块正常执行后的一种追加处理")
finally:  # 不管有没有捕获异常，下属代码段都会执行
    print("无论有无异常产生，此处的代码块都会被执行")
```

```
IndexError: list index out of range
无论有无异常产生，此处的代码块都会被执行
```

20190319

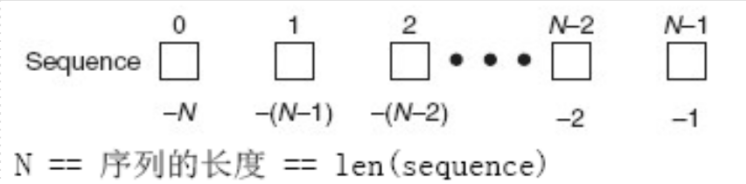
组合类型

- 组合数据类型：能够将多个同类型或不同类型的数据组织起来，通过单一表示，使数据操作更有序更容易
- 根据数据之间的关系，组合数据类型可以分为三类：序列类型、集合类型和字典类型，见图示

20190319

Sequence序列

- Sequence：成员有序排列，可以通过下标访问其成员
- String：’hello’ “hello” ’’’hello’’’
- List：[2, 4, ’python’]
  [2.4, 6.8, ‘python’, [1, 3, “hello world”]]
- Tuple：(3.3, 6, “computer”)
  (4, (“hello”, 9), [1.1, 5.7, 8])
- 序列有相同访问方式：



Sequence序列

- 序列类型：str（字符串）、tuple（元组）、list（列表）
- 元组是包含0个或多个数据项的不可变序列类型。
- 列表则是一个可以修改数据项的序列类型，使用也最灵活

20190319

Sequence序列

- 序列类型有一些通用的操作符和函数

操作符	描述
x in s	如果x是s的元素，返回True，否则返回False
x not in s	如果x不是s的元素，返回True，否则返回False
s + t	连接s和t
s * n 或 n * s	将序列s复制n次
s[i]	索引，返回序列的第i个元素
s[i: j]	分片，返回包含序列s第i到j个元素的子序列（不包含第j个元素）
s[i: j: k]	步骤分片，返回包含序列s第i到j个元素以j为步数的子序列
len(s)	序列s的元素个数（长度）
min(s)	序列s中的最小元素
max(s)	序列s中的最大元素
s.index(x[, i[, j]])	序列s中从i开始到j位置中第一次出现元素x的位置
s.count(x)	序列s中出现x的总次数

20190319

List列表

List（列表） 是 Python 中使用最频繁的数据类型

列表是写在方括号 [ ] 之间、元素之间用逗号分隔开

```
In [101]: list1 = [1,2,3,4,5,6]
          print(list1)
[1, 2, 3, 4, 5, 6]
```

20190319

列表中元素的类型可以不相同，它支持数字，字符串甚至可以包含列表（所谓嵌套）

```
In [102]: list2 = [1,2,3,4,5,6,"hello python",[8,9,10,11,12]]
          print(list2)
[1, 2, 3, 4, 5, 6, 'hello python', [8, 9, 10, 11, 12]]
```

List列表

可以通过索引（下标）来访问列表元素

单个列表元素访问的语法格式为：列表名[下标]

20190319

列表下标从0开始，-1表示倒数第1个

```
In [103]: list1 = [1,2,3,4,5,6]

          list1[0]

Out[103]: 1

In [104]: list1[2]

Out[104]: 3
```

```
In [105]: list1[-1]

Out[105]: 6

In [106]: list1[-3]

Out[106]: 4
```

List列表

列表截取（切片）的语法格式为：列表名[头下标:尾下标]

```
In [108]: list1 = [1,2,3,4,5,6]

          list1[0:3]
          # 列表截取（切片）返回一个包含所需内容的新列表
Out[108]: [1, 2, 3]
```

20190319

结果不包含尾下标那个元素！

```
In [109]: list1[-3:-1]
Out[109]: [4, 5]
```

切片步长

```
In [110]: list1[::2]
Out[110]: [1, 3, 5]
```

列表操作

In [2]:
```
#list用方括号[]表示，元素之间用逗号分隔开。也可以通过list函数将元组或字符串转化成列表
#list包含0个或多个对象索引的有序序列，没有长度限制，可自由增删元素
ls = [1, 'a', 2, 'b']
print(type(ls))

#访问区间为左闭后开，即不包含尾下标元素
print(ls[0:2])
print('修改前：', ls)

# 修改list的内容
ls[0] = 3
print('修改后：', ls)

# 末尾添加元素
ls.append(4)
print('添加后：', ls)

# 指定位置添加元素
ls.insert(1,'c')
print('添加后：', ls)

#删除元素
del ls[1]
print('删除后：', ls)
```

20190319

```
# 遍历list
print('遍历list(for循环)：')
for item in ls:
    print(item)

# 通过索引遍历list
print('遍历list(while循环)：')
i = 0
while i != len(ls):
    print(ls[i])
    i += 1

# 列表合并
print('列表合并(+)：', [1, 2] + [3, 4])

# 列表重复
print('列表重复(*)：', [1, 2] * 5)

# 判断元素是否在列表中
print('判断元素存在(in)：', 1 in [1, 2])
```

列表方法

- append()/insert()添加元素
- “+”组合两个列表生成新的列表
- extend()向调用它的列表中添加另外一个列表的元素
- del()/pop()/remove()删除元素　　20190319
- 切片
- in/not in判断元素在列表中是否存在
- sort()列表内元素重排序
- reverse()列表内容倒置
- count()统计列表内指定元素个数

列表推导式

- 列表推导式(List Comprehension)提供了一个创建和操作列表的有力工具
- 列表推导式由一 个表达式以及紧跟着这个表达式的for语句构成，for语句还可以跟0个或多个if或for语句

20190319

```
In [153]: lst1 = [1, 2, 3]
          lst2 = [3, 4, 5]
          [x * y for x in lst1 for y in lst2]

Out[153]: [3, 4, 5, 6, 8, 10, 9, 12, 15]
```

➢ 尝试用循环完成此功能

列表推导式

- 数值判断可以链接使用，例如 1<x<3 能够判断变量 x是否在1和3之间

20190319

```
In [154]: [x for x in lst1 if 4 > x > 1]

Out[154]: [2, 3]
```

列表推导式

```
In [1]: print('利用for循环找出20内的偶数：')
        l1 = []
        for i in range(20):
            if i % 2 == 0:
                l1.append(i)
        print(l1)
```

20190319

```
利用for循环找出20内的偶数：
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```
In [2]: print('利用列表推导式找出200内的偶数：')
        l2 = [i for i in range(20) if i % 2 == 0]
        print(l2)
```

```
利用列表推导式找出200内的偶数：
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Tuple元组

元组（tuple）与列表类似，不同之处在于元组的元素不能修改

元组写在小括号 ( ) 里，元素之间用逗号隔开

元组中元素的类型可以不相同，和列表类似，也支持嵌套

20190319

```
In [114]: tuple1 = (1,2,3,4,5,6,"hello python",[8,9,10,11,12],(13,14))

          print(tuple1)

          (1, 2, 3, 4, 5, 6, 'hello python', [8, 9, 10, 11, 12], (13, 14))
```

Tuple元组

```
In [3]: #tuple
        t = (1, 'a', 2, 'b')
        print(type(t))

        #元组的内容不能修改，否则会报错
        # t[0] = 3

        # 遍历tuple
        print('遍历list(for循环)：')
        for item in t:
            print(item)

        # 通过索引遍历tuple
        print('遍历tuple(while循环)：')
        i = 0
        while i != len(t):
            print(t[i])
            i += 1
```

20190319

Set集合

- 无序存储不同数据类型不重复元素的序列
- 集合定义：name_set={"xiaoming","xiaoqiang","xiaobai","lisi"}
- 使set(序列)对序列中元素去重，同时创建集合
  例如：name_set = set(["xiaoming“, "zhansan"])
- 创建空集合：none_set = set()
- 使用in和not in判断一个元素在集合中是否存
- 使用add(元素)方法添加一个元素到集合中
- 使用update(序列)方法将一个序列中的元素添加到集合中，同时对元素去重

20190319

- remove(元素)根据元素值删除集合中指定元素，如果元素不存在，则报错
- discard(元素)根据元素值删除集合中指定元素，如果元素不存在，不会引发错误
- pop()随机删除集合中的某个元素，并且返回被删除的元素
- clear()清空集合
- 集合操作
  - 交集intersection（&）
  - 并集union（|）
  - 差集difference（-）
  - 对称差集（^）

集合操作

```
In [9]: #set是一个无序，且不含重复元素的序列
        #创建set
        my_set = {1, 2, 3}
        print(my_set)

        #自动去重
        my_set = set([1, 2, 3, 2])
        print(my_set)

        #添加元素
        my_set.add(3)
        print('添加3', my_set)

        #添加多个元素
        my_set.update([4, 5, 6])
        print(my_set)

        #成员关系测试
        4 in my_set

        {1, 2, 3}
        {1, 2, 3}
        添加3 {1, 2, 3}
        {1, 2, 3, 4, 5, 6}

Out[9]: True
```

集合的操作：交、并、差、补

```
In [120]: set1 = {1,2,3}
          set2 = {2,4,5}

In [121]: #集合的并
          set1 | set2

Out[121]: {1, 2, 3, 4, 5}

In [122]: #集合的交
          set1 & set2

Out[122]: {2}
```

```
In [123]: #集合的差
          set1 - set2

Out[123]: {1, 3}

In [124]: #集合的补，两个集合中不同时存在的元素集合
          set1 ^ set2

Out[124]: {1, 3, 4, 5}

In [125]: (set1|set2)-(set1&set2)

Out[125]: {1, 3, 4, 5}
```

20190319

Dictionary字典

字典是一种映射类型，用"{ }"标识，它是一个无序的 键(key) : 值(value)对 集合

键(key)必须使用不可变类型，在同一个字典中，键(key)是唯一的

字典当中的元素是通过键来存取的

```
In [126]: dict1 = {"name":"giggle","height":176,"weight":72}

In [127]: dict1["height"]
Out[127]: 176
```

20190319

存储Key-Value键值对类型的数据

字典定义：{key1:value1,key2:value2,…}

- 查询：根据Key查找Value
- 字典具有添加、修改、删除操作
- 内置方法get、keys、values、items、clear
- 循环遍历字典

字典内建方法

字典类型也有一些内置的函数，例如clear()、keys()、values()

```
In [132]: dict2.keys()
Out[132]: dict_keys(['name', 'height'])

In [133]: dict2.values()
Out[133]: dict_values(['giggle', 176])

In [134]: dict2.clear()
          print(dict2)
          {}
```

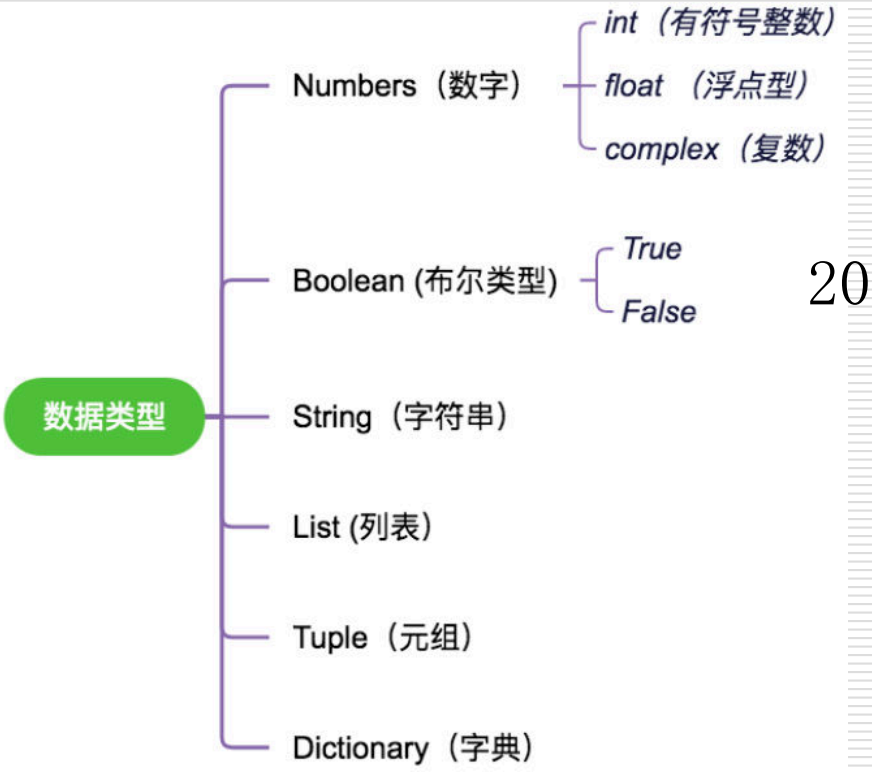
20190319

Dictionary字典

```
In [2]: #dictionary
d = {'百度': 'https://www.baidu.com/',
     '阿里巴巴': 'https://www.alibaba.com/',
     '腾讯': 'https://www.tencent.com/'}

print('通过key获取value: ', d['百度'])

# 遍历key
print('遍历key: ')
for key in d.keys():
    print(key)

# 遍历value
print('遍历value: ')
for value in d.values():
    print(value)

# 遍历item
print('遍历item: ')
for key, value in d.items():
    print(key + ': ' + value)

# format输出格式
print('format输出格式: ')
for key, value in d.items():
    print('{}的网址是{}'.format(key, value))
```

20190319

数据类型



20190310

不可变类型： String, Tuple, Set
对象创建之后，其值不能更新

可变类型：List, Dictionary
对象创建之后，其值可以更新-

类型转换函数

函　　数	描　　述
int(x [,base])	将字符串 x 转换为一个整数
float(x)	将字符串 x 转换为一个浮点数
complex(real [,imag])	根据 real 和 imag 创建一个浮点数
str(x)	将对象 x 转换为字符串
repr(obj)	将对象 obj 当作 Python 语句执行，返回结果的字符串形式
eval(str)	计算字符串中的有效Python表达式，返回结果
tuple(s)	将序列 s 转换为一个元组
list(s)	将序列 s 转换为一个列表
chr(x)	将一个整数转换为一个字符
unichr(x)	将一个整数转换为 Unicode 字符
ord(x)	将一个字符转换为它的整数值
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串

- **eval函数实现list、dict、tuple与str之间转化**
  str函数把list，dict，tuple转为为字符串

eval()函数

- eval(<字符串>)函数能够以表达式的方式解析并执行字符串，将返回结果输出
- eval(<字符串>)函数的作用是将输入的字符串转变成表达式

20190319

```
#eval()函数
#一定要确保第一个参数expression满足表达式的要求，它是可以被解析然后计算的
a=10;
print(eval("a+1"))
```

```
11
```

Function函数

- 定义函数：def fun_name()
- 调用函数：fun_name( )
- 函数参数：形参、实参
- 缺失参数和命名参数
- 不定长参数和拆包
- 局部变量和全局变量
- 递归函数和Lamda函数

20190319

函数定义

- 可以理解为对实现某一功能的封装
- 函数定义

```
def 函数名称（参数）：
    函数体代码
    return 返回值
```

- 函数调用：函数名(参数)
- 函数参数

形参：定义函数时设置的参数
实参：调用函数时传入的参数

20190319

函数定义

- 函数通过“def”关键字进行声明，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
- 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

20190319

```
In [7]: # 定义一个求 n!的函数
        def fact(n):
            result=1
            for i in range(1,n+1):
                result=result*i
            return result

In [8]: fact(5)

Out[8]: 120
```

函数调用

```
#函数调用
#调用函数时传入的参数个数与函数定义参数个数相同
#参数顺序要相同
def print_user_info2(name,age,gender):
    print("name:%s"%name)
    print("age:%d"%age)
    print("gender:%s"%gender)

print_user_info2("悟空",20,"male")
print_user_info2("八戒",18,"male")
```

```
name:悟空
age:20
gender:male
name:八戒
age:18
gender:male
```

20190319

函数参数

- 缺省参数
  - 函数定义带有初始值的形参
  - 函数调用时，缺省参数可传，也可不传
  - 缺省参数一定要位于参数列表的最后
  - 缺省参数数量没有限制
- 命名参数
  - 调用带有参数的函数时，通过指定参数名称传入参数的值
  - 可以不按函数定义的参数顺序传入

20190319

函数参数传递

```
#函数调用：参数的位置和名称传递
def x_y_sum(x, y=20):
    print("x=%d" %x)
    print("y=%d" %y)
    return  x + y

sum1 = x_y_sum(2,5)
print('sum1 = ',sum1)
sum2 = x_y_sum(2,)
print('sum2 = ',sum2)
```

```
x=2
y=5
sum1 =  7
x=2
y=20
sum2 =  22
```

```
#错误缺省参数使用方法
def x_y_sum2(y=10,x):
    return x + y

rs1 = x_y_sum(10,30)
print('rs1 = ',rs1)
```

```
  File "<ipython-input-3-3b750423b03f>", line 2
    def x_y_sum2(y=10,x):

SyntaxError: non-default argument follows default argument
```

In [7]:

```
#命名参数
def x_y_sum(x=10, y=20):
    return  x + y

#函数调用时命名参数的名称与函数定义时的形参名称相同，但是顺序可以不同
rs1 = x_y_sum(y=30,x=15)
rs2 = x_y_sum(x=15)
rs3 = x_y_sum()

print("rs1=%d" %rs1)
print("rs2=%d" %rs2)
print("rs3=%d" %rs3)
```

```
5
5
0
```

20190319

函数不定长参数

- 不定长参数
  - 函数可以接收不定个数的参数传入
  - def fuction([formal_args,]*args)函数调用时，传入的不定参数会被封装成元组

20190319

  - def fuction([formal_args,]**args)函数调用时，如果传入key=value形式的不定长参数，会被封装成字典
- 拆包
  - 对于定义了不定长参数的函数，在函数调用时需要把已定义好的元组或者列表传入到函数中，需要使用拆包方法

函数不定长参数

```
#工资计算器
def salary_comp(basic_money,*other_money,**proportion):
    print("缴费基数:{}".format(basic_money))
    print("其他工资:{}".format(other_money))
    print("比例:{}".format(proportion))

other_money = (500,200,100,1000)
proportion_dict = {"e":0.2,"m":0.1,"a":0.12}

salary_comp(8000,other_money,proportion_dict)

#拆包方法
salary_comp(8000,*other_money,**proportion_dict)
```

20190319

```
缴费基数:8000
其他工资:((500, 200, 100, 1000), {'e': 0.2, 'a': 0.12, 'm': 0.1})
比例:{}
缴费基数:8000
其他工资:(500, 200, 100, 1000)
比例:{'e': 0.2, 'a': 0.12, 'm': 0.1}
```

局部变量与全局变量

- 局部变量
  - 函数内部定义的变量
  - 不同函数内的局部变量可以定义相同的名字，互不影响
  - 作用范围：函数体内有效，其他函数不能直接使用
- 全局变量
  - 函数外部定义的变量
  - 作用范围：可以在不同函数中使用
  - 在函数内使用global关键字实现修改全局变量的值
  - 全局变量命名建议以g_开头，如：g_name

20190319

```
In [1]: #局部变量
        def set_name():
            name = "zhangsan"
            return name
        nm = set_name()
        print(nm)

zhangsan

In [2]: def get_name(name):
            name = "lisi"
            print(name)
        get_name(nm)

lisi

In [3]: #全局变量
        #在不同的函数中使用
        name = "zhangsan"
        def get_name():
            print(name)
        def get_name2():
            print(name)
        get_name()
        get_name2()
        print(name)

zhangsan
zhangsan
zhangsan
```

局部变量与全局变量

In [8]:
```
#全局变量定义的位置
g_num1 = 100
g_num2 = 200
g_num3 = 300
def print_global_num():
    print("g_num1:%d"%g_num1)
    print("g_num2:%d"%g_num2)
    print("g_num3:%d"%g_num3)
print_global_num()
```

```
g_num1:100
g_num2:200
g_num3:300
```

In [9]:
```
#修改全局变量的值
age = 20
def change_age():
    global age
    age = 25
    print("函数体内age=%d"%age)
change_age()
print("函数体外age=%d"%age)
```

```
函数体内age=25
函数体外age=25
```

In [6]:
```
#在函数内修改字典、列表类型的全局变量中的元素，不需要关键字global
g_num_list = [1,2,3]
g_info_dict = {"name":"zhangsan","age":20}
def update_info():
    g_num_list.append(4)
    g_info_dict["gender"] = "male"

update_info()
print(g_num_list)
print(g_info_dict)
```

```
[1, 2, 3, 4]
{'name': 'zhangsan', 'age': 20, 'gender': 'male'}
```

20190319

递归函数

- 递归函数
  - 函数调用自身
  - 注意：递归过程中要有用于结束递归的判断

In [16]:
```
#递归函数，自身调用自身
#利用递归函数实现求n!
def recursive(num):
    if num > 1:
        return num * recursive(num -1)
    else:
        return num

#3 * 2!
#2 * 1!

print(recursive(4))
```

24

20190319

[14]:
```
#for循环实现求n!
'''
1! = 1
2! = 2 * 1
3! = 3 * 2 * 1
...
n! = n * (n-1)!
'''

def recursive_for(num):
    rs = num
    for i in range(1,num): # (1, 4) = [1, 2, 3)
        rs *= i
    return rs
rs = recursive_for(5)
print(rs)
```

120

匿名函数

- 匿名函数
  - 用lambda关键字创建匿名函数
  - 定义：lambda［参数列表]:表达式
  - 匿名函数可以作为参数被传入其他函数

20190319

```
# 匿名函数：定义单行函数
sum = lambda x,y: x+y
print(sum(10,20))

#等价函数
def sum(x,y):
    return x + y
```

```
30
```

```
#匿名函数作为参数传入自定义函数中
def x_y_comp(x,y,func):
    rs = func(x,y)
    print(rs)
x_y_comp(3,5,lambda x,y: x+y)
print("----------------")
x_y_comp(4,7,lambda x,y: x*y)
```

```
8
----------------
28
```

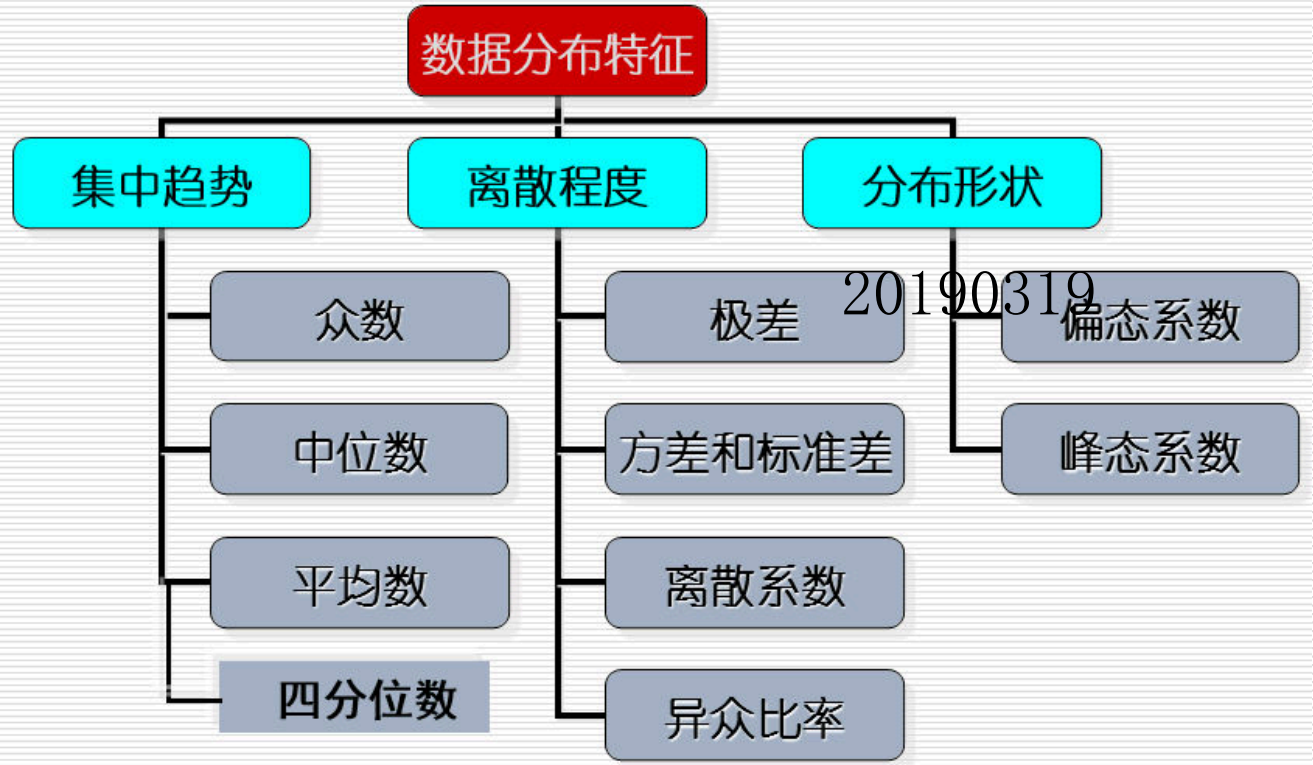
Python内置函数

- Python提供的内置函数，前36个需要掌握

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

20190319

Python计算基本统计值



20190319

Python计算基本统计值

In [15]:
```
#Python 基本统计值
#输入多个数，计算出平均值， 方差， 中位数

def getNum():
    nums = []
    iNumStr = input("请输入数字（回车退出）：")
    while iNumStr != "":
        nums.append(eval(iNumStr))
        iNumStr = input("请输入数字（回车退出）：")
    return nums

def mean(numbers):
    s = 0.0
    for num in numbers:
        s += num
    return s / len(numbers)

def dev(numbers, mean):
    sdev = 0.0
    for num in numbers:
        sdev += (num - mean)**2
    return pow(sdev / (len(numbers)-1), 0.5)

def median(numbers):
    sorted(numbers)
    size = len(numbers)
    if size % 2 == 0:
        med = (numbers[size//2-1] + numbers[size//2])/2
    else:
        med = numbers[size//2]
    return med

n = getNum()
m = mean(n)
print("平均值:{}, 方差:{:.2}, 中位数:{}.".format(m, dev(n, m), median(n)))
```

20190319

```
请输入数字（回车退出）：1
请输入数字（回车退出）：2
请输入数字（回车退出）：9
请输入数字（回车退出）：10
请输入数字（回车退出）：
平均值:5.5, 方差:4.7, 中位数:5.5.
```

Class类

- 类（Class）用来描述具有相同的属性和方法的对象的集合
- 它定义了该集合中每个对象所共有的属性和方法
- 对象是类的实例

使用 class 语句来创建一个新类，class 之后为类的名称并以冒号结尾:

```
class ClassName:
        '类的帮助信息'   #类文档字符串
        class_suite      #类体
```

class_suite 由类成员，方法，数据属性组成

Class类

- 类的构成
  - 类的名称
  - 类的属性
  - 类的方法
- 类的定义

20190319

```
class 类名:
    def 方法名(self[,参数列表])
```

- 类名的命名规则按照“大驼峰”
- 定义的方法默认要传入一个self参数，表示自己，self参数必须是第一个参数
- 创建对象：对象变量名 = 类名()

定义类

```
#定义类
class Dog:
    def eat(self):
        print("小狗正在啃骨头...")
    def drink(self):
        print("小狗正在喝水...")
```

```
#创建对象
wang_cai = Dog()
print(id(wang_cai))
wang_cai.eat()
wang_cai.drink()
```

```
71143152
小狗正在啃骨头...
小狗正在喝水...
```

```
In [2]: #查看类的属性
        dir(Dog)

Out[2]: ['__class__',
         '__delattr__',
         '__dict__',
         '__dir__',
         '__doc__',
         '__eq__',
         '__format__',
         '__ge__',
         '__getattribute__',
         '__gt__',
         '__hash__',
         '__init__',
         '__le__',
         '__lt__',
         '__module__',
         '__ne__',
         '__new__',
         '__reduce__',
         '__reduce_ex__',
         '__repr__',
         '__setattr__',
         '__sizeof__',
         '__str__',
         '__subclasshook__',
         '__weakref__',
         'drink',
         'eat']
```

```
wang_cai = Dog()
dir(wang_cai)

['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'drink',
 'eat']
```

20190319

构造函数

- 构造函数：__init__，也称为类的初始化方法，当创建类的实例时会自动调用该方法
- 类方法与普通函数区别：前者必须有第一个额外参数self，但在调用时不必传入相应的参数

```
class Dog:
    "这是Dog类"
    def __init__(self,gender,variety,name,age):
        #print("构造方法：在创建对象时自动调用")
        self.gender = gender
        self.variety = variety
        self.name = name
        self.__age = age   #__age私有变量

    #获取对象属性并打印出来
    def get_pro(self):
        print("gender:{},variety:{},name:{},age:{}".format(self.gender,self.variety,self.name,self.__age))

    #设置对象内部属性
    def set_pro(self,**kwargs):
        if "gender" in kwargs:
            self.gender = kwargs["gender"]
        elif "age" in kwargs:
            if kwargs["age"] < 0 or kwargs["age"]>20:
                print("非法年龄")
            else:
                self.__age = kwargs["age"]
    def eat(self):
        print("正在吃骨头...")

    def drink(self):
        print("正在喝水....")
```

20190319

```
#创建对象
wangcai = Dog("male","golden","wangcai",1)
print(type(wang_cai))
wangcai.eat()
wangcai.drink()
wangcai.get_pro()
print(Dog.__doc__)
```

```
<class '__main__.Dog'>
正在吃骨头...
正在喝水....
gender:male,variety:golden,name:wangcai,age:1
这是Dog类
```

dir()

调用dir()来显示该对象的所有方法

```
In [10]: dir(1)

Out[10]: ['__abs__',
          '__add__',
          '__and__',
          '__bool__',
          '__ceil__',
          '__class__',
          '__delattr__',
          '__dir__',
          '__divmod__',
          '__doc__',
          '__eq__',
          '__float__',
          '__floor__',
          '__floordiv__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getnewargs__',
```

```
In [29]: dir(wangcai)

Out[29]: ['_Dog__age',
          '__class__',
          '__delattr__',
          '__dict__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__gt__',
          '__hash__',
          '__init__',
          '__le__',
          '__lt__',
          '__module__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__setattr__',
          '__sizeof__',
          '__str__',
          '__subclasshook__',
          '__weakref__',
          'drink',
          'eat',
          'gender',
          'get_pro',
          'name',
          'set_pro',
          'variety']
```

20190319

help()

- help()显示类的帮助信息

```
In [33]: help(Dog)

Help on class Dog in module __main__:

class Dog(builtins.object)
 |  这是Dog类
 |
 |  Methods defined here:
 |
 |  __init__(self, gender, variety, name, age)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  drink(self)
 |
 |  eat(self)
 |
 |  get_pro(self)
 |      # 获取对象属性并打印出来
 |
 |  set_pro(self, **kwargs)
 |      #设置对象内部属性
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

20190319

File文件

- 文件：一个存储在辅助存储器上的数据序列，可包含任何数据内容
- 两种类型：文本文件和二进制文件
- 主要区别：在于是否有统一的字符编码
- 无论文件创建为文本文件或者二进制文件，都可以用“文本文件方式”和“二进制文件方式”打开，打开后的操作不同。

File文件

- 理解文本文件和二进制文件的区别

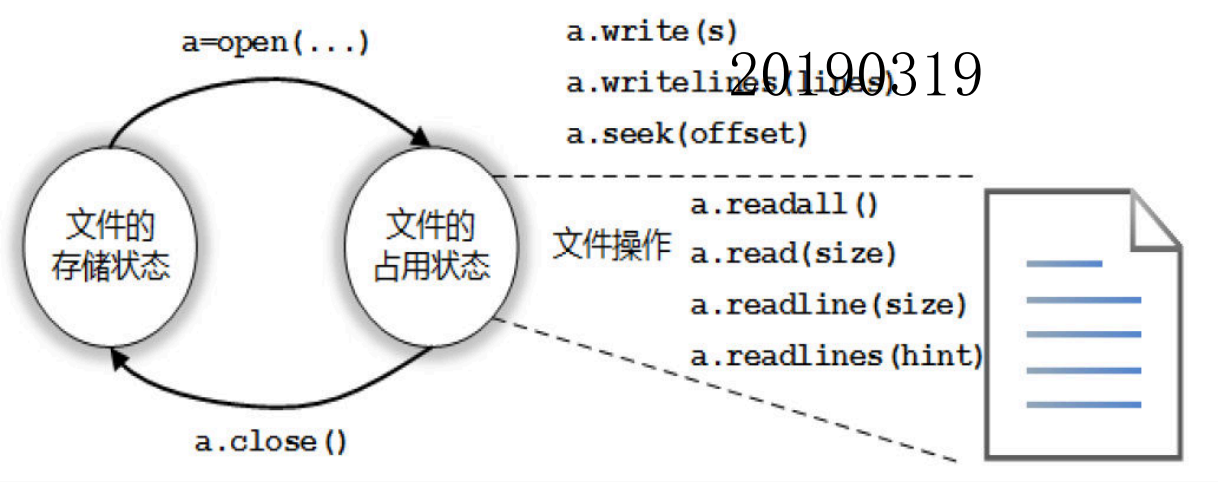
```
In [16]: #以文本文件方式打开文件
         f = open("test.txt","rt",encoding="utf-8")
         print(f.read())
         f.close()

         #以二进制文件方式打开文件
         f = open("test.txt","rb")
         print(f.read())
         f.close()
```

20190319

```
你好
b'\xe4\xbd\xa0\xe5\xa5\xbd'
```

文件打开关闭

- 两种文件采用统一的操作步骤，即“打开-操作-关闭”



文件打开

- open()函数打开一个文件，并实现该文件与一个程序变量的关联，其格式：

  <变量名> = open(<文件名>, <打开模式>)

  参数文件名：实际文件名，可以包含文件所在的完整路径

20190319

打开模式	含义
'r'	只读模式，如果文件不存在，返回异常FileNotFoundError，默认值
'w'	覆盖写模式，文件不存在则创建，存在则完全覆盖源文件
'x'	创建写模式，文件不存在则创建，存在则返回异常FileExistsError
'a'	追加写模式，文件不存在则创建，存在则在原文件最后追加内容
'b'	二进制文件模式
't'	文本文件模式，默认值
'+'	与r/w/x/a一同使用，在原功能基础上增加同时读写功能

文件操作

- Python针对文件操作有许多内建的函数库可以调用

```
# 写文件
with open("test.txt","wt") as out_file:
    out_file.write("I love python")

# 读文件
with open("test.txt","rt") as in_file:
    text = in_file.read()

print(text)
```

```
I love python
```

20190319

常用库应用

- Math库
- Datetime库
- Random库
- jieba库
- PIL库

20190319