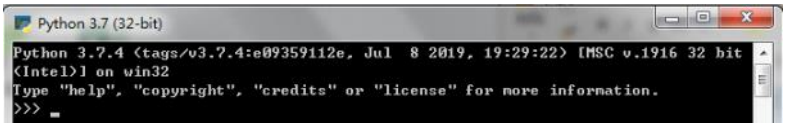
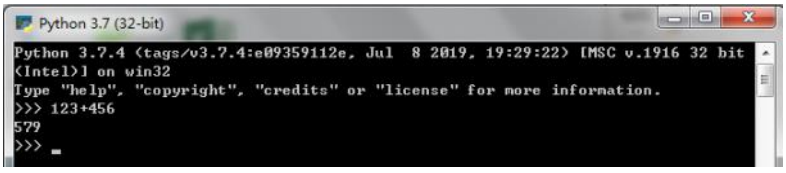
Task 1

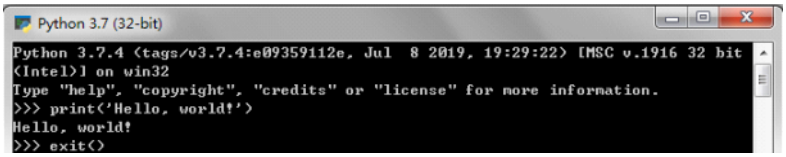
2019年8月5日　　12:26

1. 环境搭建

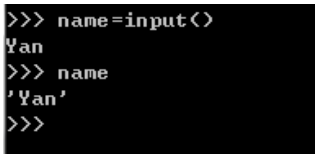


2. Python初体验

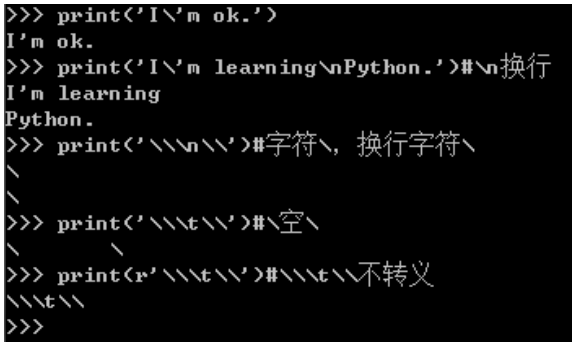




3. Python基础讲解
   1) Python变量特性、命名规则



   - 变量可以是整数或者浮点数，也可以是字符串。
   - Python中等号=是赋值语句，可以吧任意数据类型赋值给变量，同一个变量可以反复赋值，且可以是不同类型的变量。
   - 命名规则：变量在程序中用一个变量名表示，变量名必须是大小写英文、数字和下划线的组合，不能用数字开头。
   - Python程序是大小写敏感的，即区分大小写。
   - 包含’和“的字符串可以用转移字符\标识：
     - \n ——表示换行；
     - \t ——表示制表符；
     - \\ ——表示字符\；
     - r'' ——表示引号内部的字符串默认不转义；



   2) 注释方法
      - # ——表示单行注释；
      - ''' '''（开头结尾各三个单引号）——表示多行注释；

- """ """（开头结尾各三个双引号）——表示多行注释；

3) Python中“：”的作用
   - 当语句以冒号：结尾时，缩进的语句视为代码块；
4) 学会使用dir()、help()
   - dir()函数不带参数时，返回当前范围内的变量、方法和定义的类型列表；带参数时，返回参数的的属性、方法列表
   - help()函数用于查看函数或模块用途的详细说明
5) import使用：用于导入模块
6) Pep8介绍：
   - Pep8是Python的编码规范

4. Python数值基本知识
   1) Python中数值类型：
      - int ——整型，表示整数；
      - long ——长整型，可以表示无线大的整数；
      - float ——浮点型，表示实数；
      - bloor ——布尔型，一种特殊的整型，只有True（1）和False（0）E两个取值；
      - e记法 ——科学计数法，用e代替10，如1.23*10^9=1.23e9，0.000012=1.2e-5；
      - 字符串 ——以单引号或双引号括起来的任意文本；
      - 空值 ——用None表示（一个特殊的空值）
   2) 算术运算符
      - + ——加
      - - ——减
      - * ——乘
      - ** ——幂（返回x的y次幂）
      - / ——除 （精确除法）
      - // ——地板除（只取结果的整数部分）
      - % ——取模（返回除法的余数）
   3) 逻辑运算符：and（与）、or（或）、not（非）
   4) 成员运算符：检测给定值是否为序列中的成员
      - in ——若在指定序列中找到一个变量的值，则返回True，否则返回False；
      - not in ——若在指定序列中找不到变量的值，则返回True，否则返回False；
   5) 身份运算符：用于比较两个对象的存储单元
      - is ——判断两个标识是不是引用自一个对象；
      - not is ——判断两个标识符是不是引用自不同对象；
   6) 运算符优先级

运算符	描述
**	指数 (最高优先级)
~ + -	按位翻转, 一元加号和减号 (最后两个的方法名为 +@ 和 -@)
* / % //	乘，除，取模和取整除
+ -	加法减法
>> <<	右移，左移运算符
&	位 'AND'
^ |	位运算符
<= < > >=	比较运算符
<> == !=	等于运算符
= %= /= //= -= += *= **=	赋值运算符
is is not	身份运算符
in not in	成员运算符
not and or	逻辑运算符

Task 2

2019年8月7日　12:09

- 列表（list）
  - 标志；
  - 基本操作：创建，append（），pop（），del（），拷贝；
  - 列表；
  - 列表相关方法；

1. 概念：
   - 列表：由一系列按特定顺序排列的元素组成（有序的集合）。可以创建包含字母表中所有字母、数字或所有家庭成员姓名的列表；也可将任何东西加入列表中，其中的元素之间可以没有任何关系（元素也可以是另一个list）。鉴于列表通常包含多个元素，一般给列表指定一个表示复数的名称（如letters、digits，names等）；
   - 列表可以修改；
   - Python中，用方括号（[]）表示列表，并用逗号来分隔其中的元素。
2. 基本操作：
   a. 创建一个名为“classmates”的列表，存储全班同学的姓名（列表名=['元素1','元素2','元素3',]）；

```
>>> #创建一个全班同学姓名的列表
... classmates=['Michael','Bob','Tracy']
>>> print(classmates)
['Michael', 'Bob', 'Tracy']
```

   b. len()——获得list元素的个数（len(列表名)）；

```
>>> len(classmates)
3
```

   c. 索引（列表名[i]）：
      - 用索引来访问list中每一个位置的元素，记得索引是从0开始：

```
>>> classmates[0]
'Michael'
>>> classmates[1]
'Bob'
```

      - 当索引超出范围时Python就会报错，记得最后一个元素的索引是len(classmates)-1；

```
>>> classmates[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

      - 若要调取最后一个元素，除了计算索引位置外，还可以用 -1 做索引直接获取最后一个元素，以此类推 -2倒数第二个，-3倒数第三个（注意不可越界）；

```
>>> classmates[-1]
'Tracy'
>>> classmates[-3]
'Michael'
```

   d. append()——向list末尾追加元素（列表名.append('插入元素内容')）；

```
>>> classmates.append('Yan')#向列表classmates末尾添加元素‘Yan’
>>> print(classmates)
['Michael', 'Bob', 'Tracy', 'Yan']
```

   e. insert(i)——向list的指定位置i处插入元素（列表名.insert(i,'插入元素内容')）；

```
>>> classmates.insert(2,'Gakki')
>>> print(classmates)
['Michael', 'Bob', 'Gakki', 'Tracy', 'Yan']
```

      - 注意：把元素插入到指定的位置，原来这个位置上的元素会后移，按倒数的方法插入也是如此，如：

```
>>> print(classmates)
['Michael', 'Bob', 'Gakki', 'Tracy', 'Yan']
>>> classmates.insert(-2,'黄大胖')#在倒数第二个位置插入元素‘黄大胖’
>>> print(classmates)
['Michael', 'Bob', 'Gakki', '黄大胖', 'Tracy', 'Yan']
>>> #原来倒二位置上的‘Tracy’顺次后移
```

   f. pop()——删除list末尾的元素（列表名.pop()）；

```
... classmates.pop()#删除list末尾元素‘Yan’
'Yan'
>>> print(classmates)
['Michael', 'Bob', 'Gakki', '黄大胖', 'Tracy']
```

      - pop(i)——删除指定位置i处的元素（列表名.pop(i)）；

```
>>> classmates.pop(-2)#删除倒二元素‘黄大胖’
'黄大胖'
>>> print(classmates)
['Michael', 'Bob', 'Gakki', 'Tracy']
```

g. del() ——删除列表i处的元素（del 列表名[i]）；

```
>>> del classmates[0]
>>> print(classmates)
['Bob', 'Gakki', 'Tracy']
```

h. remove('元素值') ——删除元素（不知道其在list中所处位置，但是知道该元素的值）（列表名.remove('元素内容')）；

```
>>> classmates.remove('Bob')
>>> print(classmates)
['Gakki', 'Tracy']
```

i. 替换元素 ——要把某个元素替换成别的元素，可以直接赋值给对应的索引位置（列表名[i]='新元素内容'）；

```
>>> print(classmates)
['Gakki', 'Tracy', 'Amy']
>>> classmates[1]='Lisa'
>>> print(classmates)
['Gakki', 'Lisa', 'Amy']
```

j. 拷贝

- 元组（Tuple）
  - 标志；
  - 基本操作（创建及不可变性）；

1. 概念：
   - 元组是一种有序列表，和list非常相似，但是tuple一旦初始化就不能修改，即不可变的列表称为元组；
   - 因为tuple不可变，所有代码更安全；
   - 元组使用圆括号()；
2. 基本操作：
   a. 创建元组：
      - 定义一个元组时，其元素就必须被确定下来；

```
>>> t=(2,7,4)#创建一个元组t
>>> print(t)
(2, 7, 4)
```

      - 定义一个空的tuple，可以写成()；

```
>>> t2=()#创建一个空的元组t2
>>> print(t2)
()
```

      - 定义一个只有一个元素的tuple时，必须加一个逗号，来消除歧义；

```
>>> t3=(77,)#创建一个只含一个元素‘77’的元组t3
>>> print(t3)
(77,)
```

      - 定义一个“可变的”元组：因为元组包含一个可变的列表，所以该元组的内容改变；

```
>>> change=('a','b',['c','d'])#创建元组change，其第三个元素是一个list
>>> change[2][0]='X'#赋值c=X
>>> change[2][1]='Y'#赋值d=Y
>>> print(change)
('a', 'b', ['X', 'Y'])
>>> #元组change元素“改变”
```

   b. 元组的不可变性：
      - 由于“指向不变”的原因，要创建一个内容不变的tuple，就必须保证tuple的每一个元素本身也不能变。

- string字符串
  - 定义及基本操作（+，*，读取方式）；
  - 字符串相关方法；

1. 概念：
   - 字符串就是一系列字符；
   - 在Python中，用引号括起的都是字符串，引号可以是单引号、双引号、三引号；
   - 字符串一经创建就是不可变的；
2. 基本操作：

a. 创建和查看字符串：

```
>>> s1='123'
>>> s2='abc'
>>> print(s1)
123
>>> print(s2)
abc
```

b. 加号 + ——可以直接将两个字符串收尾相连，形成一个新的字符串；

```
>>> print(s1+s2)
123abc
```

c. * ——

• 字符串格式化问题

1. Python中，采用的格式化方式和C语言一致，用%实现；

```
>>> print('Hello,%s'%'world')
Hello,world
>>> print('Hi,%s,you have $%d.' % ('Michael', 10000))
Hi,Michael,you have $10000.
```

   - % ——用来格式化字符串；
   - %s ——在字符串内部，表示用字符串替换；
   - %d ——表示用整数替换，有几个%？占位符，后面就跟几个变量或值，顺序要对应好；如果只有一个%？，括号可以省略。

2. 常见的占位符：
   - %d ——整数；
   - %f ——浮点数；
   - %s ——字符串；
   - %x ——十六进制整数；
   - 若不太确定该用什么，%s永远起作用，它会吧任何数据类型转换为字符串；

Task 3

2019年8月9日 1:03

- dict字典
  - 定义
  - 创建
  - 字典的方法

1. 定义：

- Python内置了字典：dict的支持，dict全称dictionary，在其他语言中称为map，使用“键-值（key-value）”存储，具有极快的查找速度。
- dict的实现方法（同查字典一样）：
  - 现在字典的索引表里（比如部首表）查这个字对应的页码，然后直接翻到该页找到这个字。无论找哪个字，这种查找速度都非常快，不会随着字典大学的增加而变慢。
  - list查找原理：假设要在字典里查找一个字，把字典从第一页往后玩，知道找到想要的字为止。list越大，查找越慢。
  - 初始化指定dict：

```
>>> #用dict查找学生成绩
... d={'Michael':90,'Bob':80,'Tracy':70}
>>> d['Michael']
90
```

- dict和list比较：
  - dict：
    i. 查找和插入的速度极快，不会随着可以的增加而增加；
    ii. 需要占用大量的内存，内存浪费多；
  - list：
    i. 查找和插入的时间随着元素的增加而增加；
    ii. 占用空间小，浪费内存很少；
- dict是用空间来换取时间的一种方法；
- dict可以用在需要高速查找的很多地方；
- 牢记dict的key必须是不可变对象
  - 因dict根据key来计算value的存储位置，若每次计算相同的key得出的结果不同，那dict内部就完全混乱了；
  - Python中，字符串、整数等都是不可变的，可以作为key；而list可变，不能作为key；

2. 创建：

- dict采用“key-value”存储方式，在放进去的时候必须根据key算出value的存放为止，这样，取的时候才能根据key直接拿到value。
- 把数据放入dict的方法，除了初始化时指定外，还可以通过可以放入；

```
>>> #通过key把数据放入dict
... d['Adam']=60
>>> d['Adam']
60
```

- 由于一个key只能对应一个value，所以多次对一个key放入value时，后面的值会把前面的值冲掉；若key不存在，dict就会报错；

```
>>> d['Jack']=20
>>> d['Jack']
20
>>> d['Jack']=89#覆盖前面的值
>>> d['Jack']
89
>>> d['Gakki']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Gakki'
```

- 要避免key不存在的错误，有两种办法：
  a. 通过in判断key是否存在；

```
>>> #通过in判断key是否存在:
... 'Gakki' in d
False
```

  b. 通过dict提供的get方法，如果key不存在，可以返回None，或者自己指定的value；

```
>>> #通过get方法判断:
... d.get('Gakki')
>>> d.get('Gakki',-1)#key不存在，返回自己指定的value：-1
-1
```

  - 注意：返回None时，Python的交互命令行不显示结果。
- pop(key) ——删除一个key，对应的value也会从dict中删除；

```
>>> d.pop('Bob')
80
>>> d
{'Michael': 90, 'Tracy': 70, 'Adam': 60, 'Jack': 89}
```

- dict内部存放的顺序和key放入的顺序没有关系；

集合（set）

- 特性
- 创建
- 方法

1. 特性：

- 集合（set）是一个无序的不重复的元素序列；
- set和dict类似，也是一组key的几何，但不能存储value；
- 由于key不能重复，所以在set中，没有重复的key；

2. 创建set：

- 要创建一个set，需要提供一个list作为输入集合：

```
>>> s=set([1,2,3])
>>> s
{1, 2, 3}
```

  - 注意：传入的参数[1,2,3]是一个list，而显示的{1,2,3}只是说明此set内部有1,2,3这三个元素，显示的顺序也不表示set是有序的。
- 重复元素在set中自动被过滤：

```
>>> s=set([1,1,1,3,5,5,6,2,2,1])
>>> s
{1, 2, 3, 5, 6}
```

- add(key)——添加元素到set中；可以重复添加，但不会有效果；

```
>>> s.add(4)
>>> s
{1, 2, 3, 4, 5, 6}
>>> s.add(4)
>>> s
{1, 2, 3, 4, 5, 6}
```

- remove(key)——删除元素；

```
>>> s.remove(3)
>>> s
{1, 2, 4, 5, 6}
```

- set可以看成数学意义上的无序和无重复元素的集合，因此，两个set可以做数学意义上的交集、并集等操作；

```
>>> s1=set([1,2,3])
>>> s2=set([1,1,3])
>>> s1&s2
{1, 3}
>>> s1|s2
{1, 2, 3}
```

3. set和dict的区别：

- set和dict的唯一区别在于没有存储对应的value，但是set的原理和dict一样，所以同样不可以放入可变对象，因无法判断两个可变对象是否相等，也就无法保证set内部“不会有重复元素”。

判断语句（要求掌握多条件判断）

- 条件判断用if语句实现：if 条件表达式：
  代码块

```
>>> age=20
>>> if age>=18:
...     print('your age is',age)
...     print('adult')
```

- if-elif-else语句：

```
if 〈条件判断 1〉:
    〈执行 1〉
elif 〈条件判断 2〉:
    〈执行 2〉
elif 〈条件判断 3〉:
    〈执行 3〉
else:
    〈执行 4〉
```

```
>>> if age>=18:
...     print('adult')
... elif age>=6:
...     print('teenager')
... else:
...     print('kid')
```

- 三目表达式
- Python中的格式为：
  - 为真时的结果 if 判定条件 else 为假时的结果
- 先输出结果，再判定条件；

```
>>> print(1 if 5>3 else 0)
1
>>> #先输出结果，再判定条件；
... #输出1，如果5大于3，否则输出0
```

- 一般用于判断赋值中；

```
>>> x,y=50,25
>>> small=x if x<y else y
>>> print(small)
25
```

- 还可以嵌套使用，还可以多层嵌套；

```
>>> a,b,c=10,33,90
>>> min_num=a if a<b and a<c else (b if b<a and b<c else c)
>>> print(min_num)
10
```

- 循环语句

1. while循环

```
while 条件表达式 :
    代码块
else :
    代码块
```

   - 注意：print一定要写在while里面！

2. for循环

Task 4

2019年8月11日 20:18

• 函数关键字

- 函数关键字：def

• 函数的定义

1. 概念：
   - 函数也是一个对象
     - 对象是内存中专门用来存储数据的一块区域；
     - 函数可以用来保存一些可执行的代码，并且可以在需要是对这些语句进行多次的调用；

```
>>> max(1,2)
2
>>> min(837,4,3,3,1)
1
>>> abs(230)
230
>>> abs(-0.9)
0.9
```

2. 函数的定义规则：
   - 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；
   - 函数内容以冒号开始，并缩进；
   - 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；

3. 创建函数：
   - def 函数名([形参1，形参2，.....]):
     代码块
   - 函数名必须要符号标识符的规范，即可以包含字母、数字、下划线，但是不能以数字开头；
   - 函数中保存的代码不会立即执行，需要调用函数代码才会执行；
   - 调用函数：函数对象()；

```
>>> def my_abs(x):
...     if x>=0:
...             return x
...     else:
...             return -x
... #定义一个求绝对值的函数my_abs
```

• 函数参数与作用阈

1. 函数参数
   - 在定义函数时，可以在函数名后的圆括号中定义数量不等的形参，多个形参之间用逗号隔开；
   - 形参（形式参数）：定义形参就相当于在函数内部声明了变量，但并不赋值；
   - 实参（实际参数）：如果函数定义时，指定了形参，那么在调用函数时也必须传递实参，实参将会赋值给对应的形参，简单来说，有几个形参就得传几个实参；

2. 作用域（scope）：指变量生效的区域，Python中一共有两种作用域：
   a. 全局作用域
      - 全局作用域在程序执行时创建，在程序执行结束时销毁；
      - 所有函数以外的区域都是全局作用域；
      - 在全局作用域中定义的变量都属于全局变量，全局变量可以在程序的任意位置被访问；
   b. 函数作用域

- 函数作用域在函数调用时创建，在调用结束时销毁；
- 函数每调用一次就会产生一个新的函数作用域；
- 在函数作用域顶一顶变量都是局部变量，它只能在函数内部被访问；

c. 变量查找

- 当使用变量时，会优先在当前作用域中寻找该变量，如果有则使用，若没有则继续去上一级作用域中寻找；

函数返回值

- 返回值就是函数执行以后返回的结果；
- 可以通过return来指定函数的返回值；
- 可以直接使用函数的返回值，也可以通过一个变量来接收函数的返回值；

File

- 打开文件方式（读写两种方式）
- 文件对象的操作方法
- 学习对excel及CSV文件进行操作

- 想要对一个文件进行读写操作，需要先知道该文件的绝对路径，打开文件需要使用Python内置函数open；
- open函数有三个常用参数：文件路径（file），打开文件的模式（mode），编码格式（encoding);

```
open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, clo
sefd=True, opener=None)
```

- 打开文件的模式：
  - 'r'——只读；
  - 'w'——只写；新写入的内容会把原文件中的内容替换掉；
  - 'a'——追加写；在光标处添加想要写入的内容，假如光标在文件原先内容中，那新添加的内容会把光标后面的原有等量文件内容替换掉，若光标在原文件内容末尾，那只会追加新内容，不会替换原有内容；
- 对文件进行读取操作：read函数；
- 写入操作：write函数
- 关闭操作：close函数

os模块

- os.sep——取代操作系统特定的路径分隔符；
- os.name——指示当前正在使用的工作平台；
- os.getcwd——得到当前的工作目录，即当前Python脚本工作的目录路径；
- os.getenv()和os.putenv——分别用来读取和设置环境变量；
- os.listdir()——返回指定目录下的所有文件和目录名；
- os.remove(file)——删除一个文件；
- os.stat(file)——获得文件属性；
- os.chmod(file)——修改文件权限和时间戳；
- os.mkdir(name)——创建目录；
- os.rmdir(name)——删除目录；
- os.removedirs(r "c:\python")——删除多个目录；
- os.system()——运行shell命令；
- os.exit()——终止当前进程；
- os.linesep——给出当前平台的行终止符；
- os.path.split()——返回一个路径的目录名和文件名；
- ps.path.isfile()和os.path.isdir()——分别检验给出的路径是一个目录还是文件；

Task 5

2019年8月13日 20:02

• 类和对象

1. 类（class）

- 目前学习的对象都是Python内置的对象，但内置对象并不能满足所有的需求，所有在开发中景墙需要自定义一些对象；
- 类，简单理解就相当于一个图纸，在程序中需要根据类来创建对象；
- 类就是对象的图纸，我们也称对象是类的实例（instance）；
- 如果多个对象是通过一个类创建的，我们称这些对象是一类对象；
- 类也是一个对象，类就是一个用来创建对象的对象；
- 类是type类型的对象，定义类实际上就是定义了一个type类型的对象

2. 对象

- 对象是内存中专门用来存储数据的一块区域；
- 对象中科院存放各种数据
- 对象由三部分组成：对象的标识（id）、对象的类型（type）、对象的值（value）；

3. 使用类创建对象的流程：

a. 创建一个变量；
b. 在内存中创建一个新对象；
c. 将对象的id赋值给变量；

4. 类的定义

- 类和对象都是对现实生活的事物或程序中的内容的抽象；
- 实际上所有的事物都由两部分组成：数据（属性）、行为（方法）；
- 在类的代码块中可以定义变量和函数；
  - 变量会成为该类实例的公共属性，所有的该类实例都可以通过“对象.属性名”的形式访问；
  - 函数会成为该类实例的公共方法，所有该类实例都可以通过“对象.方法名()”的形式调用方法；
  - 方法调用时，第一个参数由解析器自动传递，所有定义方法时，至少要定义一个形参；
  - 实例为什么能访问到类中的属性和方法？
    - 类中定义的属性和方法都是公共的，任何该类实例都可以访问；
    - 属性和方法查找的流程：
      1) 当调用一个对象的属性时，解析器会先在当前对象中寻找是否含有该属性，
      2) 若有，则直接返回当前对象的属性值；
      3) 若没有，则去当前对象的类对象中去寻找，若有则返回类对象的属性值，若无则报错。
  - 类对象和实例对象中都可以保存属性（方法）；
    - 若这个属性（方法）是所有的实例共享的，则应该将其保存到类对象中；
    - 若这个属性（方法）是某个实例独有，则应该保存到实例对象中；
    - 一般情况下，属性保存到实例对象中，而方法需要保存到类对象中；

5. 创建对象的流程：

- P1=Person()的运行流程
  a. 创建一个变量；
  b. 在内存中创建一个新对象；
  c. _init_(self)方法执行；

d. 将对象的id赋值给变量；

6. 类的基本结构

```
class 类名([父类]) :

    公共的属性...

    # 对象的初始化方法
    def __init__(self,...):
        ...

    # 其他的方法
    def method_1(self,...):
        ...

    def method_2(self,...):
        ...
```

```
>>> #创建一个类“Student”
... class Student:
...     stuCount=0
...     def _init_(self,name,salary):#构造函数
...             self.name=name
...             self.age=age
...             Student.stuCount += 1
...     def printInfo(self):#类方法
...             print("Student Name:{},Age:{},format(self.name,self.age))
```

- ## 正则表达式
- 正则表达式，简称regex，是文本模式的描述方法。它的设计思想是用一种描述性的语言来给字符串定义一个规则，凡是符合规则的字符串，我们就认为它“匹配”了，否则该字符串就是不合法的。

- ## re模块
- 从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功的话就返回none
  - re.match(pattern, string, flags=0)
- pattern:匹配的正则表达式 string:要匹配的字符串
- flags:标志位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等
- 扫描整个字符串并返回第一个成功的匹配。
  - re.search(pattern, string, flags=0)
- 检索和替换
  - re.sub(pattern, repl, string, count=0, flags=0)
- 在字符串中找到正则表达式所匹配的所有子串，并返回一个列表
  - findall(string[, pos[, endpos]])
- 按照能够匹配的子串将字符串分割后返回列表
  - re.split(pattern, string[, maxsplit=0, flags=0])

- ## datetime模块学习
- datetime模块体公用一些处理日期、时间和实践间隔的函数。这个模块使用面向对象的交互取代了time模块中整型/元组类型的时间函数。在这个模块中的所有类型都是新类型，能够从Python中继承和扩展。这个模块包含如下的类型：
  - datetime代表了日期和一天的时间；
  - date代表日期，在1到9999之间；
  - time代表时间和独立日期；
  - timedelta代表两个时间或者日期的间隔；

- tzinfo实现时区支持；
- datetime类型代表了某一个时区的日期和时间，除非有特殊说明，datetime对象使用的是“naïve time”，也就是说程序中datetime的时间跟所在的时区有关联。datetime模块提供了一些时区的支持。

```
>>> #获取当前日期和时间
... from datetime import datetime
>>> now=datetime.now()
>>> print(now)
```

```
>>> print(type(now))
<class 'datetime.datetime'>
```

- http请求

- http超文本传输协议
- http是一种请求/响应式的协议。
- 一个客户机与服务器建立连接后，发送一个请求给服务器，请求的格式是：统一资源标识符（URI）、协议版本号，后面是类似MIME的信息，包括请求修饰符、客户机信息和可能的内容。服务器接到请求后，给予相应的响应信息，其格式是：一个状态行包括信息的协议版本号、一个成功或错误的代码，后面也是类似MIME的信息，包括服务器信息、实体信息和可能的内容。