Python 基础知识总结

一、 变量

1. 定义变量：

```
a = 10
a, b = 10, 20
a = b = c = 100
# a = 1, b = 2   # 报错
a, *b, c = 1, 2, 3, 4
```

2. 命名规范：

    a. 只能使用字母数字下划线，且不能以数字开头

    b. 不能是关键字

        import keyword
        keyword.kwlist()

    c. 区分大小

    d. 尽量见名知意，多单词情况下：

        i. 使用下划线：my_student_name

        ii. 使用小驼峰: myStudentName

3. 删除变量

    del name

二、 运算符

1、算术运算符：+，-，*，/，%，**，//

2、赋值运算符：=，+=，-=，*=，/=，%=，**=，//=

3、关系运算符：>，>=，<，<=，=，==，!=

4、逻辑运算符：and，or，not

5、成员运算符：in，not in

6、身份运算符：is，is not

7、位运算符：&，|，~，^，<<，>>

三、 分支结构

1、单分支：if

2、双分支：if - else

3、多分支：if - elif - else

4. 类似三目运算符的语法

```
# if-else

a = 10

b = 11

x = a if a>b else b

print(x)
```

四、 循环结构

1. while 循环:
2. for 循环：
3. while-else 和 for-else
4. 循环相关的关键字
    a. break 关键字：
        i. 在循环中使用；
        ii. break 之后的代码不会执行；
        iii. break 只会跳出当前这一层循环
        IV. break 可以配合 for - else /while - else 使用
    b. continue 关键字：结束当次循环直接进入下一次循环
    c. pass 关键字：空语句，占位语句，作用就是保证代码的一致性，完整性，防止报错
5. range 的使用

五、 数据类型

1. 数据类型:
        不可变类型/值类型/基本类型:
            number(int,float)
            bool: True,False
            str: 字符串
            tuple: 元组
            None: 空值
            bytes: 二进制

可变类型/引用类型:

list: 列表

dict: 字典

set: 集合(了解)

2. 复杂的数据类型详解(list, dict, str)

列表:

定义列表:

ages = [1,2,3,4,5,6]

基本操作:

+: 拼接

*: 重复

len: 列表长度

[:]: 切片

index: 下标

列表的方法:

增:

[重点] append(n): 追加一个元素

extend([]): 追加一个列表中的所有元素

insert(index, n): 在指定的下标位置插入元素

删:

[重点] pop(index): 弹出指定下标的元素, 默认弹出最后一个元素

remove(n): 删除指定的第一个元素

count(n): 统计元素出现的次数

clear(): 清空列表

del ages[1]: 了解

改:

```
ages[1] = 100
```

查:

```
print(ages[1])
```

遍历列表:

```
for n in ages:
    pass
for i in range(len(ages)):
    pass
for i,n in enumerate(ages):
    pass
```

排序:

升序

sort():

sorted(): 升序,不会改变原列表,主要针对元组,字符串的排序

降序:

sort(reverse=True)

sorted(list, reverse=True)

倒序/逆序/反转:

reverse():

reversed(): 倒序,不会改变原列表,主要针对元组,字符串的排序

拷贝:

copy(): 浅拷贝, 一般用于一维列表

import copy

copy.deepcopy() : 深拷贝,主要用于二维列表或多维列表.

字符串:

定义字符串:

s = "hello"

基本操作:

+: 拼接

*: 重复

len: 字符串长度

[:]: 切片

index: 下标

字符串方法:

find( ): 查找子字符串在长字符串中第一次出现的下标位置, 如果不存在则返回-1

rfind(): 和 find 功能类似,但是是从右往左

split(): 根据分隔符取进行分割/拆分

splitlines(): 按行分割,等价于 split('\n')

replace(old, new): 替换

join(): 跟 split 反过来, 拼接列表中的元素

如: "+".join(['a','b'])

upper(): 变成大写
lower(): 变成小写
isupper(): 是否为大写
islower(): 是否为小写
isdigit(): 是否为数字
isalpha(): 是否为字母
isalnum(): 是否为字母或数字

encode(): 字符串=>二进制
decode(): 二进制=>字符串

eval(), ord(), chr(), startswith(), endswith()

字典:

定义字典:
d = {"name": "李小璐", "age": 29}
基本操作:
len: 字典长度
字典的方法:
增/改:
d['name'] = "PGone"
update(): 拼接另一个字典
如: d.update({"sex": "男"})
删:
[重点]pop(k): 根据 key 删除元素
popitem(): 了解, 随机删除一个元素
clear(): 了解, 清空字典
del d[key]: 了解
查:
print(d['name']): 如果 key 不存在会报错
d.get('name'): 如果 key 不存在不会报错,会返回 None

遍历字典:

```
for k in d:
    pass
for k in d.keys():
    pass
for v in d.values():
```

```
        pass
    for k,v in d.items():
        pass
```

六、 函数

1. 函数的创建：函数名，函数体，返回值

```
def fn(x, y):
    print(x + y)
    return x + y
```

2. 函数的参数：
   a. 位置参数/必须参数，默认参数，关键字参数，不定长参数。
   b. 参数的书写顺序：
      声明时： def fn(x, y, *args, z=1, **kwargs):
      调用时：fn(1,2,3, z=1,k=2)

3. 函数的返回值：return 关键字
   a、会立即终止函数/退出函数；
   b、return 后面的代码不会再执行；
   c、可以返回指定的值(返回结果)

4. 匿名函数：使用 lambda 来创建匿名函数

```
f2 = lambda x: x * x
print(f2(3))      #9

f3 = lambda x, y: x + y
print(f3(4, 5))    #9
```

5. 函数的特殊用法：函数名既是函数的名称，也是指向函数的一个变量，也可以作为函数的参数使用。例如，回调函数：

```
#回调函数
#m1 是普通函数
def m1(x, f):
    # print(x, f)
    s = f(x)
    return s
```

```
#m2 是回调函数
def m2(a):
    return a + 1

#m2 是函数名，被当作参数传递给了 m1
print(m1(3, m2))
```

6. 函数的作用域：
   a. 局部变量：在函数内部定义的变量，不可以在函数外部使用。
   b. 全局变量：在函数外部定义的变量，可以在函数内被获取，修改全局变量需要用 global
   c. 局部作用域：L
   d. 函数作用域：E　nonlocal 可以让内部函数修改指定的外部函数中的变量
   e. 全局作用域：G
   f. 内建作用域：B　（Python 作用域）

```
#1.不同作用域变量的定义
num4= int(2.9)
num3 =  3      #G：全局作用域
def outer():
    num2=2#E:函数作用域
    def inner():
        num1=1   #L: 局部作用域
        print(num4,num3,num2,num1)
    return  inner
f = outer()
f()
#注意：当所有的变量不同名的时候，在闭包中，可以任意访问四种不同作用域对应的变量
```

7. 函数的嵌套和闭包：
   如果在一个外部函数中定义一个内部函数，并且外部函数的返回值是内部函数，就构成了一个闭包，则这个内部函数就被称为闭包【closure】

```
def out():
    p = 10
    def inner():
        nonlocal p
        p += 1
        print('p =', p)
    return inner
```

8. 装饰器：
   在代码运行期间，可以动态增加函数功能的方式，被称为装饰器【Decorator】

```
#通用装饰器
def outer2(f2):
    def inner(*args, **kwargs):
        print('开始唱歌了')
        r = f2(*args, **kwargs)
        print('唱完了', r)
        return r
    return inner

@outer2     #@语法糖
def sing2(singer, song, a):
    print(singer, '小红唱：', song, a)
    return 'nice'

print(sing2('小玲', '老女孩', a=10))
```

9. 函数递归：

一个会调用自身的函数【在一个函数的内部，自己调用自己】

a、找临界值，递归终止的条件

b、找规律，相邻两次循环的关系，用公式表达出来

```
# 求阶乘
# 临界值:f(1) = 1
# 公式:f(n) = n*f(n-1)
def f(n):
    if n == 1:
        return 1
    return n * f(n - 1)
```

七、 包和模块

1. 包:

包含__init__.py 文件的文件夹，一般用来存储模块

2. 模块:

模块就是 python 文件

3. 包和模块的命名规范:

和变量命名规范一致

4. 导入包和模块

a, import

格式:

import package

示例:

import math

import os, random

b, from-import

格式:

from package import module

from package.module import variable/function

from module import variable/function

示例:

from package import module

from package.module import sex, login

八、 系统模块

1. os 模块

os.listdir(): 获取指定目录下的所有文件和文件夹名称

os.mkdir(): 创建一个目录

os.makedirs(): 可以创建多层目录

os.rmdir(): 删除空目录

os.rename(): 重命名文件或目录

os.remove(): 删除文件

os.getcwd(): 获取当前路径

os.curdir(): 获取当前路径,得到的是一个点.

. 表示当前目录

.. 表示父目录

2. os.path 模块

os.path.abspath(): 获取到绝对路径

os.path.join(): 拼接路径

os.path.split(): 拆分路径/分割路径

os.path.splittext(): 拆分文件名,比如: *hello.py*

os.path.isdir(): 判断是否为目录

os.path.isfile(): 判断是否为文件

os.path.exists(): 判断是否存在

os.path.getsize(): 获取文件大小

os.path.dirname(): 父目录

os.path.basename(): 文件名

3. time 模块

概念:

UTC: 国际标准时间

时间戳: 从 1970 年 1 月 1 日到指定时间的秒数

time 模块中的方法

time.time()：获取当前时间的时间戳

time.sleep(): 让程序暂停

time.gmtime(secs):时间戳=>时间元组, UTC

time.localtime(secs): 时间戳=>时间元组, 本地时间

time.mktime(t): 时间元组=>时间戳

time.strftime(): 时间元组=>时间字符串

time.strptime(): 时间字符串=>时间元组

time.asctime()

time.ctime()

4. datetime 模块[掌握]

a. 创建日期对象

d = datetime.datetime.now()：当前时间的日期对象

d = datetime.datetime(2020, 1, 2, 10, 10, 10)：创建指定的日期对象

b. 日期对象的方法和属性

d.year, d.month, d.day：年, 月, 日

d.hour, d.minute, d.second: 时,分,秒

d.date(): 年月日

d.time(): 时分秒

d.strftime(): 输出一个格式化的时间字符串

d.timestamp(): 时间戳

datetime.timedelta(days=8, hours=10): 时间差

5. calendar 模块

c = calendar.calendar(2020, w=2, l=1, c=6)

c = calendar.month(2020, 7)

calendar.isleap(2020)

calendar.leapdays(1900, 2020)

calendar.monthcalendar(2020, 7)

calendar.monthrange(2020, 6)

6. hashlib 模块

md5 加密:

```
m = hashlib.md5()
m.update("hello".encode())
m2 = m.hexdigest()
```

九、 第三方模块

安装第三方模块

1. 使用 pycharm 去安装
2. 使用 pip 命令

pip -V ：查看 pip 版本

pip install numpy : 安装包

pip uninstall numpy: 卸载包.

pip list : 查看所有的包

pip freeze: 查看自己安装的包

pip show numpy: 查看包详情

十、 遍历目录

```
import os

# 遍历目录 p
def search_dir(path):
    filename_list = os.listdir(path)
    for filename in filename_list:
        # file_path: 每个子文件或子文件夹的绝对路径
        file_path = os.path.join(path, filename)

        # 如果是文件
        if os.path.isfile(file_path):
            print("文件名:", filename)

        # 如果是目录
        elif os.path.isdir(file_path):
            print("目录名:", filename)
            # 递归遍历当前目录的子目录
            search_dir(file_path)
search_dir(r'C:\Users\ijeff\Desktop\Python2003\day11\code\newdir')
```

十一、 面向对象

1. 面向对象和面向过程

   面向过程: 侧重解决问题的步骤, 按顺序一步一步执行, 亲力亲为.

   面向对象: 侧重问题中的对象, 以指挥官的身份去使用这些对象做事情.

2. 类和对象

   类: 封装属性和方法, 创建类的目的是用来创建对象, 类是对象的抽象

   对象: 是类的一个具体, 我们要使用类来创建对象

3. 构造函数和析构函数

   构造函数:

```
class A:
    def __init__(self, name):
        self.name = name
```

   i. 创建对象时会自动调用构造函数

   ii. 作用是对属性进行初始化

   析构函数:

```
class B:
    def __del__(self):
        print("析构函数")
```

4. self

   a. self 存在于类中的函数里面的形参

   b. self 不是关键字

   c. self 是指向当前类的对象, 哪个对象调用了该函数,则 self 就是这个对象

5. __slots__

   限制属性的名称

```
__slots__ = ("name", "age")
```

6. 封装

   a. 私有化

      使用双下划线开头: __name

      私有化的特点: 只能在当前类内部使用

      i. 属性私有化

ii. 方法私有化

b. property 装饰器

作用: 让方法可以当成属性来使用

条件: 必须要写返回值

7. 继承

a. 单继承

定义: 只有一个父类

```
# 父类
class Father:
    def __init__(self, name, age):
        self.name = name
        self.age = age

# 子类
class Son(Father):
    def __init__(self, name, age, sex):
        super().__init__(name, age)
        # Father.__init__(self, name, age)
        self.sex = sex
```

b. 多继承

定义: 有 2 个或 2 个以上的父类

```
# 父类
class Father:
    def __init__(self, name):
        self.name = name
# 父类
class Mother:
    def __init__(self, age):
        self.age = age
# 子类
class Son(Father, Mother):
    def __init__(self, name, age, sex):
        # Father.__init__(self, name)
        # Mother.__init__(self, age)
        super(Son, self).__init__(name)
        super(Father, self).__init__(age)
        self.sex = sex
```

c. 私有属性和私有方法不可以继承

d. 重写

定义：在继承的基础上，子类有和父类相同的方法，则子类重写了父类的方法

8. __str__和__repr__

作用：让打印的对象值为我们__str__函数中的返回值(字符串)

条件：必须返回字符串类型(一般返回属性值)

9. 多态:

在继承的基础上,而且需要用重写. 要在父类中定义一个方法,让所有子类重写该方法,那么我们可以使用父类对象去指向不同的子类来调用同一个方法名,则有不同的状态或结果

10. isinstance 和 type

isinstance：判断某个对象是否是某个类(父类)创建的，返回 bool 值

type：打印某个对象的类型

11. 对象属性和类属性的区别

对象属性:

调用：对象.对象属性

赋值：如果属性存在则修改，如果属性不存在则会动态添加新属性

内存：独立的，每个对象的对象属性是独立的内存

类属性:

调用:

1. 对象.类属性
2. 类.类属性(推荐)

赋值:

要使用类.类属性的方式赋值

内存:

共享的内存

12. 类方法和静态方法

类方法:

@classmethod

静态方法:

@staticmethod

```
class Person:
    age = 10

    # 构造方法/构造函数
    def __init__(self, name):
        self.name = name

    # 成员方法(常用)
    def sleep(self):
        print("睡觉:", self)

    # 私有方法
    def __run(self):
        print("跑步")

    # 类方法:
    #     1. 可以使用类和对象调用,但是建议使用类来调用,可以节省内存
    #     2. 可以使用类属性和其他类方法, 但是不能使用对象属性和成员方法和私有方法
    #     3. 一般用在功能比较单独,和类中其他属性和方法无关的情况下
    # cls: class
    @classmethod
    def eat(cls):
        # print(cls.name)    # 在类方法中不能使用对象属性和成员方法,因为没有 self
        print("吃饭:", cls==Person)    # True

    # 静态方法:
    #     1. 可以使用类和对象调用,但是建议使用类来调用,可以节省内存
    #     2. 不可以使用对象属性,成员方法和私有方法,一般也不用类属性和类方法
    #     3. 就是一个非常普通的函数,只是写类里面
    @staticmethod
    def sing():
        print("唱歌")
```

13. 特殊属性

__name__ : 得到类名
__class__: 打印对象所属类
__dict__: 得到所有属性组成的字典
__module__: 所在模块
__bases__: 所有父类
__mro__: 继承链

14. 运算符重载

可以让对象使用运算符

```
class Girl:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # + 加法
    def __add__(self, other):
        return self.age + other.age

# 对象
p1 = Girl('杨超越', 22)
p2 = Girl('关晓彤', 20)

print(p1.age + p2.age)
print(p1 + p2)
```

15. 单例模式

```
# 方法一: 使用 new
class Person(object):

    # 构造函数: 用来对属性初始化
    def __init__(self, name):
        print("__init__")
        self.name = name

    # 类属性
    instance = None

    # new 方法:用来创建对象
    @classmethod
    def __new__(cls, *args, **kwargs):
        print("__new__")
        if not cls.instance:
            print("---创建新对象---")
            cls.instance = super().__new__(cls)  # 新建对象
        return cls.instance

# 方法二: 使用装饰器
def outer(cls):
    instance = None
    print("instace 初始化为 None")
```

```
    def inner(*args, **kwargs):
        nonlocal instance
        if not instance:
            print("---创建了新对象---")
            instance = cls(*args, **kwargs)
            print('--返回对象--')
        return instance
    return inner

@outer
class Person:
    pass

# 方法三: 使用类方法
class Singleton:
    instance = None

    @classmethod
    def get_instance(cls):
        if not cls.instance:
            cls.instance = cls()
        return cls.instance

s1 = Singleton.get_instance()
s2 = Singleton.get_instance()
print(s1 is s2)    # True
```

十二、 异常处理

1. 异常: 代码运行时可能报错
2. 捕获异常

```
# 第一种写法
try:
    a =  9 / 0
except:
    print("报错了")

# 第二种写法
try:
    a = a + 1
except NameError as e:
    print("报错了:", e)
```

```
# 第三种写法
try:
    a = a + 1
except Exception as e:
    print("报错了:", e)

# 第四种写法
try:
    a = a + 1
except Exception as e:
    print("报错了:", e)
else:
    print("没有出错")

# 第五种写法
try:
    a = a + 1
except Exception as e:
    print("报错了:", e)
finally:
    print("不管有没有错误，这里的代码都会执行")
```

3. 抛出异常

```
raise NameError("hello")
```

4. 自定义异常

```
class MyException(Exception):
    pass
```

5. 断言

```
def fn(n):
    assert n!=0, "错误,除数不能为 0"
    a = 1/n
fn(0)
```

十三、 文件操作

1. 文件打开模式

r 只读，如果文件不存在则报错

rb 只读二进制，如果文件不存在则报错

w　清空写，如果文件不存在则创建该文件

wb　清空写二进制，如果文件不存在则创建该文件

a　追加写，如果文件不存在则创建该文件

ab　追加写二进制，如果文件不存在则创建该文件

2. 文件操作流程
   a. 打开文件
      fp = open('file', mode, encoding)
   b. 文件操作
      fp.read()
         fp.read()　所有内容
         fp.read(1024)　指定长度的内容
         fp.readline()　一行行读取
         fp.readlines()　读取所有行,返回列表
      fp.write()
   c. 关闭文件
      fp.close()
3. 编码,解码

   encode()：编码（字符串=>二进制)

   decode()：解码（二进制=>字符串)
4. csv 文件读写

十四、　正则表达式

1. 正则表达式的函数

   re.match()：匹配是否以指定正则开头

   re.search()：匹配是否包含正则的内容

   re.findall()：查找所有正则匹配的内容,返回一个列表

   re.compile()：编译正则,创建正则对象

   re.finditer()：查找所有正则匹配的内容，返回一个迭代器

   re.split()：分割,可以写正则

   re.sub()：替换,可以写正则

   re.subn()：替换,可以写正则,返回包含替换的次数的元组
2. 正则的符号
   a. 匹配单个字符

. 表示匹配任意单个字符,但是不包含换行
[ ] 表示匹配单个字符的一个范围
\d 数字
\w 数字字母下划线
\s 空格

b. 匹配字符数量
? 表示前面字符的数量可以出现 0 或 1 次
+ 表示前面字符的数量可以出现 1 或多次
* 表示前面字符的数量可以出现 0 或多次
{ } 表示可以出现指定次数或次数范围

c. 边界字符
^ 开头匹配
$ 结尾匹配
^$ 完全匹配
\A \Z
\b \B

d. 分组和捕获
( ) : 当成整体, 分组
group( ): 获取分组内容
groups() : 获取所有分组内容

e. 非捕获性分组
(?:) 不会捕获

f. 或
| 或者

g. 分组别名
(?P<name> pattern) : 给分组取别名

3. flags
re.I : 忽略大小写
re.S: 让点语法(.)可以匹配到换行
re.M: 换行模式

十五、 网络通信

1. TCP
    a. TCP 的特点:
        i. 可靠的, 发送数据前会建立可靠的长连接('三次握手')
        ii. 双向通讯: 客户端可以给服务端发数据,服务端可以向客户端发数据
        iii. 即时性: 随时可以发数据(因为有长连接)
        iv. 断开连接时会进行'四次挥手'
        v. 相对 UDP 慢
    b. TCP 的使用
        i. 服务器端

```
# TCP 通讯 服务端
import socket

# 1. 创建一个 socket 对象(服务端)
#       AF_INET: IPV4
#       AF_INET6: IPV6
#     SOCK_STREAM: 表示 TCP 协议
#     SOCK_DGRAM: 表示 UDP 协议
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 2. 绑定 IP 和 PORT
server_socket.bind( ('10.36.139.197', 6668) )

# 3. 设置监听数(客户端连接数)
server_socket.listen(5)

# 4. 等待客户端来连接我
print("服务器已启动, 等待客户端连接...")
client, addr = server_socket.accept()  # 会让程序暂停
#  (<socket.socket fd=420, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('10.36.139.197', 6668), raddr=('10.36.139.197', 60773)>, ('10.36.139.197', 60773))
# print(client)
# print(addr)
print("有客户端连接成功了")

# 5. 连接成功后,就可以开始收发数据了
while True:
# 接收数据, 阻塞程序(让程序暂停)
data = client.recv(1024)
```

```
print("客户端说:", data.decode())

#  发送数据给客户端
client.send("今晚吃鸡吗".encode())


# 6. 关闭连接
# server_socket.close()
```

ii. 客户端

```
# TCP 通讯  客户端
import socket

# 1. 创建 socket 对象(客户端)
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 2. 主动连接服务器端
client_socket.connect( ('10.36.139.197', 6668) )

# 3. 发送和接收服务器数据
while True:
#  发送给服务器
data = input("输入发送给服务器的数据:")
client_socket.send(data.encode())

#  接收服务器数据
data2 = client_socket.recv(1024)
print("服务器:", data2.decode())
```

2. UDP
    a. UDP 的特点
        i. 不可靠的, 不会建立长连接
        ii. 发数据报, 广播形式
        iii. 相对 TCP 快
    b. UDP 的使用
        i. 服务端

```
# UDP 通讯  服务端
import socket
```

```
# 1. 创建 UDP 的 socket 对象
# SOCK_DGRAM: 表示 UDP
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# 2. 绑定 ip 和 port
server_socket.bind( ('10.36.139.197', 5677) )

# 3. 收发数据
while True:
#  接收数据, 会阻塞程序
data, addr = server_socket.recvfrom(1024)
print(f"客户端{addr}说:", data.decode())

#  发送
server_socket.sendto('今晚吃鸡'.encode(), addr)
```

ii. 客户端

```
# UDP 通讯 客户端
import socket

# 1. 创建 UDP 的 socket 对象
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# 2. 收发数据
while True:
#  发送数据
data = input("客户端说:")
client_socket.sendto(data.encode(), ('10.36.139.197', 5677))

#  接收数据
data2, addr = client_socket.recvfrom(1024)
print("服务器说:", data2.decode())
```

十六、 发送邮件

1. 邮箱和邮件的区别
2. 代码实现

```
import smtplib
from email.mime.text import MIMEText

#  邮箱
```

```
# 邮件

# 163 的邮箱服务器: smtp.163.com
# smtp 端口:25
# 邮箱账号: niejeff@163.com
# 授权码: 123456abcde

smtp_server = "smtp.163.com"
smtp_port = 25

from_email = 'niejeff@163.com'
email_code = '123456abcde'    # 这里不是邮箱密码,而是授权码

to_email = ['niejeff@163.com', '904552498@qq.com']

# 创建邮件
msg = MIMEText("今晚来我家吃鸡")
msg['Subject'] = '大吉大利 22'    # 标题
msg['From'] = from_email
msg['To'] = 'niejeff@163.com'

# 创建邮箱对象,来发送邮件
smtp = smtplib.SMTP(smtp_server, smtp_port)    # 连接 163 邮箱服务器
smtp.login(from_email, email_code)    # 登录 163 邮箱

smtp.sendmail(from_email, to_email, msg.as_string())

# 关闭
smtp.close()
```

十七、　发送短信

1. 互亿无线

```
import requests

# 用户名 查看用户名请登录用户中心->验证码、通知短信->帐户及签名设置->APIID
account    = "C80604386"
# 密码 查看密码请登录用户中心->验证码、通知短信->帐户及签名设置->APIKEY
password = "16874f2ed0a2d0bb225747370f9aedc4"
mobile = "18566218480"
```

```
text = "您的验证码是：121254。请不要把验证码泄露给其他人。"

url = 'http://106.ihuyi.com/webservice/sms.php?method=Submit'
data = {'account': account, 'password' : password, 'content': text, 'mobile':mobile,'format':'json'}

res = requests.post(url, data=data)
print(res, res.text)
```

2. 云之讯

```
# 云之讯
import requests
import random

def send_sms(phone, vcode):
'''
发送短信
:param phone: 手机号
:param vcode: 验证码
'''

# 云之讯的短信接口
url = "https://open.ucpaas.com/ol/sms/sendsms"

data = {
"sid": "7d946861e6b2d74b2db38a442a19fd38",
"token": "8712938e22c34a179b3cea43ff783b68",
"appid": "36c083fcfdcf47b5b50d6a94fbb0e50c",

"templateid": "422930",   # 短信模板 id
"param": vcode,   # 验证码
"mobile": phone,   # 接收验证码的手机号
}

# requests
res = requests.post(url, json=data)
return res.text

def generate_vcode():
'''
生成随机 6 位验证码
'''
vcode = ""
for _ in range(6):
```

```
vcode += str(random.randint(0, 9))
return vcode

if __name__ == '__main__':
vcode = generate_vcode()
print("vcode:", vcode)

res = send_sms('18566218480', vcode)
print(res)
```

十八、 线程

1. GIL
    a. 全局解释器锁
    b. Python 默认的解释器 cpython 自带的功能
    c. 作用: 控制只允许同一时间有一个线程在使用
    d. 好处: 保证程序的完整性,避免线程的部分问题.
2. 创建线程
    a. _thread
    b. Thread
    c. 自定义线程
3. 多线程
    a. 有多个线程
4. 线程冲突和线程锁
    a. 线程冲突: 多个线程使用同一个资源时造成的冲突问题
    b. 互斥锁: 解决线程冲突, Lock
    c. 死锁: 多个线程使用多个资源时造成的相互占用对方资源问题
    d. 递归锁: 解决死锁
5. 信号量
    a. 控制最大的线程并发数
6. 相关概念
    a. 同步: 在同一个线程中,按照顺序依次执行
    b. 异步: 在不同的线程中,执行不同的任务,没有顺序
    c. 串行: 类似同步
    d. 并行: 类似异步
    e. 并行: CPU 数量 >= 任务数量
    f. 并发: CPU 数量 < 任务数量

十九、 进程

1. 创建进程
    a. Process
    b. 自定义进程
2. 多进程
    a. 多个进程
3. 进程锁和信号量
    a. multiprocess.Lock( )
    b. 信号量 Semaphore: 控制进程的最大并发数

二十、 协程

1. 创建协程
    a. greenlet+switch
    b. gevent+sleep
    c. gevent+monkey

二十一、 高阶函数

1. sorted( )
2. reversed( )
3. map( )
4. reduce( )
5. filter( )

二十二、 Python3 和 Python2 的区别

参考<Python2.x 与 Python3.x 的区别.pdf>

二十三、 JSON 模块和 Pickle

1. json 模块
    a. json 解析/json 反序列化
        i. json.loads( )
    b. json 序列化
        i. json.dumps( )

2. pickle 模块
    a. python 对象存入文件
        i. pickle.dump( )
    b. 从文件中取出
        i. pickle.load( )