

Python 基础

1.python 的运算符

1.1 算数运算符

+

-

*: 一个小妙招。例如输出 100 个 “! % “。可以写 “!%” * 100。拼接字符串。

/

//: 整除

?: 取余

**: 幂运算

1.2 比较运算符

== : 等于

!= : 不等于

>

<

>=

<=

1.3 逻辑运算符

and, or, not

2.Pycharm 安装、快捷方式、缩进、统一增加#

2.1linux 版本安装:

```
tar -zxvf pycharm-edu-3.5.1.tar.gz
```

#解压后直接可以使用了，建议移动解压文件夹到/opt 目录下，这样所有用户都可以用了。

```
pycharm_path/pycharm.sh
```

#找到文件夹中的 pycharm.sh 文件执行即可使用

2.2 软连接:

在 ubuntu 中，用户程序启动的快捷方式都放在/usr/share/applications 目录下。因此我们可以建立一个软连接放在此目录下即可。

```
#ln -s [pycharm/pycharm.sh 路径] /usr/share/applications/pycharm.desktop
```

2.3 Tab 增加 Shift+Tab 减少缩进

2.4 Ctrl+/ 可以为选中的代码统一加上注释 ‘#’

3.python 数据类型

字符串 (str)、列表、元组、字典

整型 (int)、浮点型 (float)、布尔型 (bool)、复数型 (complex)

3.1type(name)

查看变量类型

3.2 列表

列表定义: [1, 2, 'a', '&']

列表索引从 0 开始

3.2.1 列表排序

列表.sort()升序排序

列表.sort(reverse = T)降序

列表.reverse()反向, 逆序

3.2.2 列表增加

列表.insert(索引, 数据) 在指定位置后增加元素

列表.append(数据) 在末尾增加元素

列表.extend(列表 2) 将列表 2 追加到列表末尾

3.2.3 列表修改

列表[索引] = 数据 修改指定索引的数据

3.2.4 列表删除

del 列表[索引] 删除指定索引的数据

列表.remove(数据) 删除第一次出现的指定数据

列表.pop 删除末尾的数据

列表.clear 清空列表

3.2.5 列表统计

len(列表) 统计列表长度

列表.count(数据) 统计数据在列表中出现的次数

3.3 字符串

3.3.1 字符串判断

```
'hello'.isidentifier() #判断 hello 是不是合法字符串
't'.isspace() #判断\t 是不是空白字符串（空白字符串有空格，回车，水平制表符）
'abc'.isalpha() #判断字符串是不是全部由字母组成
'1234'.isdecimal() #判断字符串是否全部由十进制数字组成
'123'.isnumeric() #判断字符串是否全部由数字组成，罗马数字，汉字数字也是数字，Ⅱ，二
'123abc'.isalnum() #判断字符串是否全部由数字和字母组成，汉字属于字母
```

3.3.2 字符串内容对齐

```
a='python'
#居中对齐,center
print(a.center(8,'*')) #*python*,第一参数指定宽度，第二个参数指定填充符号，默认空格
#左对齐,ljust
print(a.ljust(8,'*')) #python**,第一参数指定宽度，第二个参数指定填充符号，默认空格
#右对齐,rjust
print(a.rjust(8,'*')) #**python,第一参数指定宽度，第二个参数指定填充符号，默认空格
```

3.3.3 字符串分割

```
a='python hello|world' #使用指定的分隔符从左侧开始分割字符串，split
lst=a.split() #默认采用空格作为分隔符,分割结果为列表
lst=a.split(seq='|') #使用 seq=指定分隔符，分割结果为列表
lst=a.split(seq=' ',maxsplit=1) #使用 seq=指定分隔符，使用 maxsplit=指定最大分割次数，达到最大分割次数后，后面的字符串将作为一个整体
#使用指定分隔符从右侧开始分割字符串，rsplit,使用方法和 split 完全一样，只是从右侧开始分割而已
```

3.3.4 字符串合并

```
#join（）将列表或者元组中的字符串合并成一个新字符串,注意必须是列表或者元组
a=['hello','python']
print('|'.join(a)) #使用|作为连接符，把 a 列表中的字符串连接起来，注意 '|' 是一个点
print(' '.join(a)) #使用空格作为连接符，把 a 列表中的字符串连接起来
print('*'.join('python')) #p*y*t*h*o*n
```

3.3.5 字符串大小写转换

```
a='PYTHON'    b='python'    c='PYThon'
b1=b.upper()   #upper,把所有字母全部转换为大写 #转成大写后产生一个新字符串
a1=a.lower()   #把所有字母全部转换为小写转成大写，产生一个新字符串
c.swapcase()   #把大写转小写，把小写转大写
c.capitalize() #capitalize,把第一个字符转换为大写，其余字符全部小写
```

3.3.6 字符串查找

```
s='hello,hello'
print(s.find('lo'))    #查找 lo 第一次出现的位置，找不到返回-1
print(s.rfind('lo'))   #查找 lo 最后一次出现的位置，找不到返回-1
```

3.3.7 字符串切片

```
a='pythonhello'
b=a[:5]    #从 0 号位开始切，切到 4 号位。pytho
c=a[1:6:2]  #[start:stop:step] step 就是等差值，可以截取 2，4，6 号位的字符，不必连续截取
f=a[::-1]   #step 为负数时，则从末尾开始截取
```

3.3.8 字符串替换

```
a='hello world'
print(a.replace('hello', 'java')) #用 java 去替换 python，第一个数是原字符串，第二个参
数是要替换成的字符串
b='hello python python python'
print(b.replace('python', 'java', 2)) #仅替换前两个 python，第一个数是原字符串，第二
个参数是要替换成的字符串，第三个参数是替换几次
####特别注意，replace 这种方法替换会产生一个新字符串，因此会伴随字符串末尾的
\n 符难以去除
re.sub('原字符串', '新字符串', 变量名) # name = re.sub('.bam\n', '', i) 这种方法会直
接修改原字符串，不会新建字符串。
```

3.3.9 格式化字符串

```
#格式化字符串--方法三：使用 f 声明格式化字符串
print(f我叫{name},今年{age}岁了')    #注意 f 的位置

#固定字符串宽度，例如固定整数的宽度
print('%10d' % 99)    #" % 99 是固定格式，%d 表示整数，%10d 表示代指的整数宽度设
为占 10 个字符，前面多了 8 个空格
print('%0.3f' % 3.1415926)    #保留三位小数，%f 代指小数，%.3f 保留三位小数
```

```
print('%10.3f % 3.1415926) #同时表示宽度和精度，宽度 10，精度保留三位小数
```

3.4 元组

定义元组：`tuple = (1, 2, 'a')`

定义完就不能修改，其他的和列表基本一致。

3.5 字典

键值对。键必须是唯一的（键就类似于索引一样）

使用`{}`定义，元素是无序的，列表是有序的。

字典`.key()` 返回所有键组成的一个列表

字典`.values()` 返回所有值组成的一个列表

字典`.items()` 返回所有键值对（`key, values`）元组组成的一个列表（列表中的每个元素是一个元组，一个键值对）

字典`['key']` #字典取值

字典`['key'] = 18` #字典增加键值对，如果键已经存在，则变成修改键值对

字典`.pop['key']` #删除键值对

4 python 内置公共方法

字符串，列表，元组，字典都可以统一使用的方法就是公共方法。

4.1 内置公共函数

`len(item)` 统计变量中元素个数
`del(item)` 删除指定变量
`max(item)` 返回容器中元素最大值，若果是字典，只针对 key
`min(item)` 返回容器中元素最小值，若果是字典，只针对 key
#字符串比较时：'a' > 'A' > '0' ACS II 码排序

4.2 切片操作

变量[start:stop:step] 支持字符串，列表，元组

4.3 公共运算符

+ : [1,2] + [3,4], 会创建新列表
* : ['HI'] * 5
in : 3 in [1,2,3]
not in
> < >= <= ==.都支持字符串，列表，元组

4.4 完整 for 循环

```
for x in range(1,10):  
    执行 1  
else:  
    执行 2  
###else 只有在遍历完全后才会执行
```

5.常用循环与函数

5.1 input 函数

```
# filepath = input("请输入文件路径：")
# number = int(input("请输入数字：")) 可以修改变量类型，默认都是字符串。
# number = float(input('请输入另一个数字：'))
```

5.2 if 语句

```
if [要判断的条件]:
    条件成立时执行内容
else:
    条件不成立执行内容
```

5.3 if elif 语句

```
if [判断条件 1]:
    执行内容 1
elif [判断条件 2]:
    执行内容 2
...
else:
    执行内容 n
```

5.4 if 嵌套

```
if [判断条件 1]:
    if [判断条件 2]:
        执行内容 1
    else:
        执行内容 2
else:
    执行内容 3
```

5.5 import 导入 python 包

5.5.1 随机数包 random

```
# import random
# 输入 random. 再按 Tab 键，可以把 random 包中所有工具全部列出来。
# random.randint(a, b) 返回[a,b]区间内的整数
```


5.6 循环 while

5.6.1 while 基本循环

```
i = 0
while 100 > i:
    执行内容
    i = i + 1
```

5.6.2 while break 循环

```
i = 0
while [判断条件 1]:
    i = i + 1
    执行内容 1
    if [判断条件 2]:
        执行内容 2
    else:
        break
print('over')
```

5.6.3 while continue 循环

continue 会直接此次循环跳出来并重新回到 while 循环条件判断。

```
i = 0
while [条件 1]:
    执行内容 1
    if [条件 2]:
        i = i + 1
        continue #特别注意 continue 后循环计数是否正常，别死循环了
print('over')
```

5.6.4 while 嵌套

```
while 条件 1:
    while 条件 2:
        执行内容 2
    执行内容 1
print('over')
```

5.7 print 函数 `print('*', end = '')`指定内容末尾符号

`#end = ''`可以指定输出内容末尾是否加什么东西。默认加换行符。

6. 转义字符

`\\` : 反斜杠

`\'` : 单引号

`\"` : 双引号

`\n` : 换行符

`\t` : 制表符

`\r` : 回车

7.函数

7.1 定义函数

7.1.1 具体格式

```
def num():  
    执行  
    return (x,y)    #可以使用元组来接收多个函数返回值  
gl_x, gl_y = num()    #可以使用多个变量接受函数的多个返回值，要求两者数量一致
```

7.1.2 Pycharm 调试工具

F8 单步执行

F7 遇到函数时，F7 可以进入函数内部单步执行

7.1.3 函数的标准注释文档

在函数定义的下方，使用连续三对引号进行注释。并且可以在函数使用的地方 **Ctrl+Q** 快速查看函数注释信息

7.2 函数传参

```
def sum_num(num1, num2):  
    result = num1 + num2  
    print(f'{num1} + {num2} = {result}')  
sum_num(10, 20)
```

7.2.1 形参和实参

形参：定义函数时传递的参数，为了让函数可以接受外部数据而设置的。例如 **num1**, **num2**

实参：实际函数调用时给的参数，例如 10, 20

7.2.2 函数返回值 return

```
def sum_num(num1, num2):  
    return num1+num2  
result = sum_num(10, 20)    #调用函数，使用一个变量接收结果即可。
```

这样可以在函数外部知道函数返回结果，否则只有函数内部知道结果，我们无法利用结果。

7.3 函数的缺省参数（默认参数）

- 定义函数时，可以给 某个参数 指定一个 **默认值**，具有默认值的参数就叫做 **缺省参数**
- 调用函数时，如果没有传入 **缺省参数** 的值，则在函数内部使用定义函数时指定的 **参数默认值**
- 函数的缺省参数，将常见的值设置为参数的缺省值，从而 **简化函数的调用**
- 例如：对列表排序的方法

```
def print_info(name, gender=True):  
    """  
    :param name: 班上同学的姓名  
    :param gender: True 男生 False 女生  
    """  
  
    gender_text = "男生"  
  
    if not gender:  
        gender_text = "女生"  
  
    print("%s 是 %s" % (name, gender_text))  
  
# 假设班上的同学，男生居多！  
print_info("小明")  
print_info("老王")  
print_info("小美", False)
```

想给函数设置默认值，就在定义函数参数时直接写上默认值。如上 `gender = True`

缺省参数必须放在函数参数末尾

函数有 多个缺省参数时，注意使用调用参数名指定参数值，避免混乱。

7.4 多值参数

`*args`: 接受元组型多值参数

`**kwargs`: 接受字典型多值参数

```
def sum_numbers(*args):  
    num = 0  
  
    print(args)  
    # 循环遍历  
    for n in args:  
        num += n  
  
    return num  
  
result = sum_numbers(1, 2, 3)  
print(result)
```

7.5 拆包函数

```
def demo(*args, **kwargs):  
    print(args)  
    print(kwargs)  
  
    # 元组变量/字典变量  
    gl_nums = (1, 2, 3)  
    gl_dict = {"name": "小明", "age": 18}  
  
    # demo(gl_nums, gl_dict)  
    |  
    # 拆包语法, 简化元组变量/字典变量的传递  
    demo(*gl_nums, **gl_dict)
```

demo(*gl_nums, **gl_dict) 我们在调用函数输入参数时, 前面加一个“*”表示这个变量指派给元组, 两个“**”表示这个变量指派给字典变量。和多值参数是对应的。

7.6 递归函数

自己调用自己, 并且一定有出口

```
def sum_numbers(num):  
    # 1. 出口  
    if num == 1:  
        return 1  
  
    # 2. 数字的累加 num + (1...num - 1)  
    # 假设 sum_numbers 能够正确的处理 1...num - 1  
    temp = sum_numbers(num - 1)  
  
    return num + temp  
  
result = sum_numbers(100)  
print(result)
```

8.类与方法

8.1 类的基本要素

在程序开发中，要设计一个类，通常需要满足一下三个要素：

1. **类名** 这类事物的名字，满足大驼峰命名法
2. **属性** 这类事物具有什么样的特征
3. **方法** 这类事物具有什么样的行为

大驼峰命名法

CapWords

1. 每一个单词的首字母大写
2. 单词与单词之间没有下划线

8.2 定义类与对象

- 在 Python 中要定义一个只包含方法的类，语法格式如下：

```
class 类名:
    def 方法1(self, 参数列表):
        pass
    def 方法2(self, 参数列表):
        pass
```

- 方法的定义格式和之前学习过的函数几乎一样
- 区别在于第一个参数必须是 `self`，大家暂时先记住，稍后介绍 `self`

注意：类名的命名规则要符合大驼峰命名法

2.2 创建对象

- 当一个类定义完成之后，要使用这个类来创建对象，语法格式如下：

```
对象变量 = 类名()
```

8.3 初始化方法定义类属性

class 类名：

```
def __init__(self, new_name):    #这是初始化方法标准格式。
    self.name = new_name    #在初始化方法内使用 self.来定义属性
def eat(self):
    pass
```

#在这个函数下定义的属性都会作为整个类标准属性，用这个类创建的对象都会具有如下属性

#初始化方法也可以加入形参，调用类创建对象时加入实参，这样创建的对象更加灵活

4.3 在初始化方法内部定义属性

- 在 `__init__` 方法内部使用 `self.属性名 = 属性的初始值` 就可以定义属性
- 定义属性之后，再使用 `Cat` 类创建的对象都会拥有该属性

```
class Cat:

    def __init__(self):

        print("这是一个初始化方法")

        # 定义用 Cat 类创建的猫对象都有一个 name 的属性
        self.name = "Tom"

    def eat(self):
        print("%s 爱吃鱼" % self.name)

# 使用类名()创建对象的时候，会自动调用初始化方法 __init__
tom = Cat()

tom.eat()
```

8.4 类属性的继承

1) 继承的语法

```
class 类名(父类名):

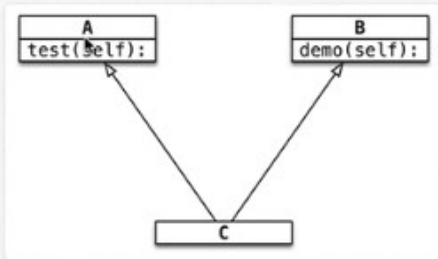
    pass
```

- 子类 继承自 父类，可以直接享受 父类中已经封装好的方法，不需要再次开发
- 子类 中应该根据 职责，封装 子类特有的 属性和方法

8.5 类属性的多继承

概念

- 子类 可以拥有 多个父类，并且具有 所有父类 的 属性 和 方法
- 例如：孩子 会继承自己 父亲 和 母亲 的特性



语法

```
class 子类名(父类名1, 父类名2...)  
    pass
```

9.异常抛出机制

9.1 所有异常捕获的完整语法

```
try:  
    # 尝试执行的代码  
    pass  
except 错误类型1:  
    # 针对错误类型1, 对应的代码处理  
    pass  
except 错误类型2:  
    # 针对错误类型2, 对应的代码处理  
    pass  
except (错误类型3, 错误类型4):  
    # 针对错误类型3 和 4, 对应的代码处理  
    pass  
except Exception as result:  
    # 打印错误信息  
    print(result)  
else:  
    # 没有异常才会执行的代码  
    pass  
finally:  
    # 无论是否有异常, 都会执行的代码  
    print("无论是否有异常, 都会执行的代码")
```

except Exception as error:

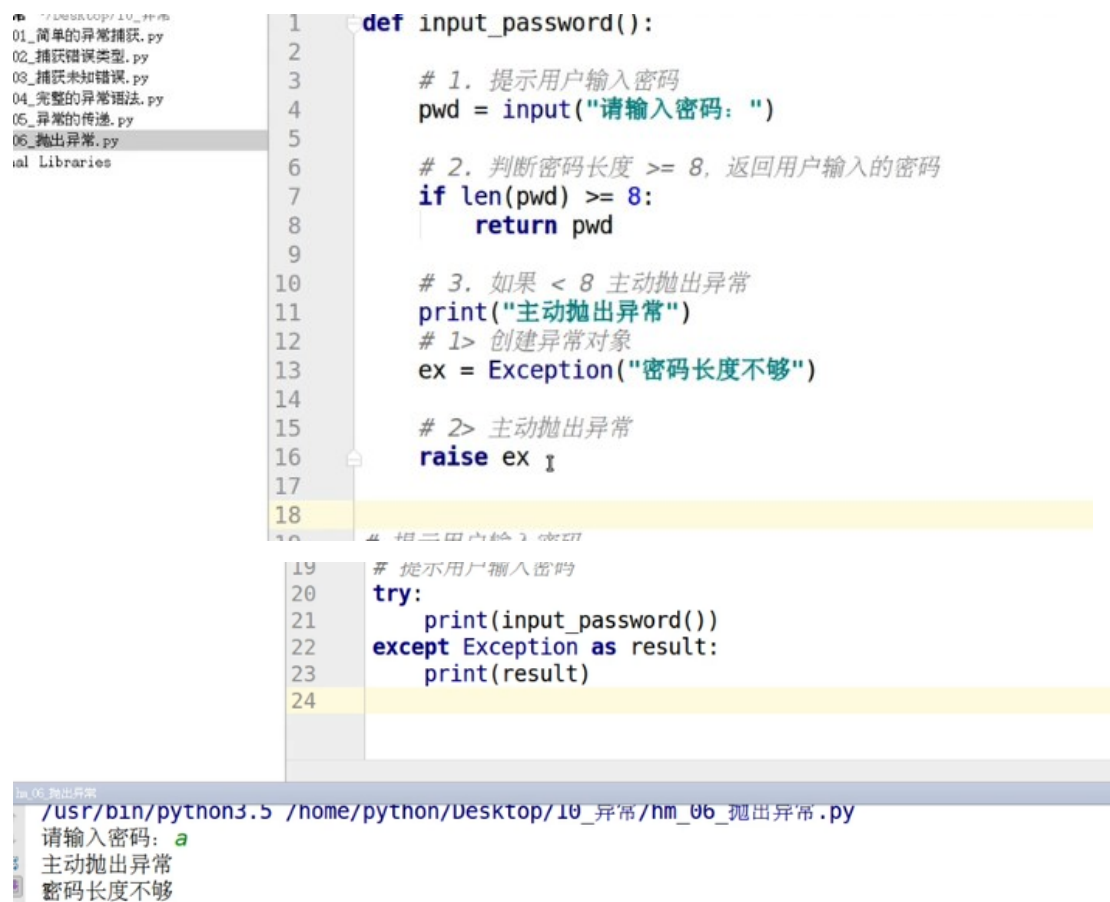
print(f“出现未知异常，为{error}”)

#这是捕获位置异常的方法。error 只是一个变量，可以改变。Exception 是位置异常捕获的专业代码，不可改变。

- else 只有在没有异常时才会执行的代码
- finally 无论是否有异常，都会执行的代码

注意异常捕获机制不要滥用，只在主程序用上就行了

9.2 主动抛出异常



```
1 def input_password():
2
3     # 1. 提示用户输入密码
4     pwd = input("请输入密码: ")
5
6     # 2. 判断密码长度 >= 8, 返回用户输入的密码
7     if len(pwd) >= 8:
8         return pwd
9
10    # 3. 如果 < 8 主动抛出异常
11    print("主动抛出异常")
12    # 1> 创建异常对象
13    ex = Exception("密码长度不够")
14
15    # 2> 主动抛出异常
16    raise ex
17
18
19 # 提示用户输入密码
20 try:
21     print(input_password())
22 except Exception as result:
23     print(result)
24
```

hm_06_抛出异常

/usr/bin/python3.5 /home/python/Desktop/10_异常/hm_06_抛出异常.py

请输入密码: a

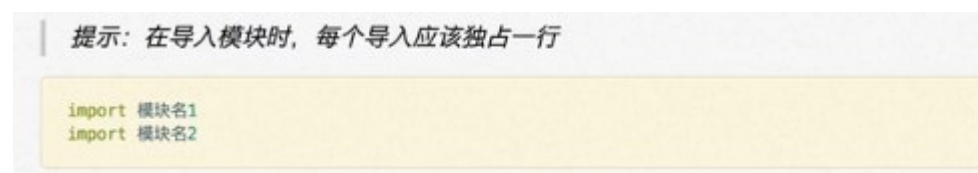
主动抛出异常

密码长度不够

Exception()函数创建异常对象，对象就是“密码长度不够”，然后利用 **raise** 函数抛出异常。最后使用 **except Exception as error:** 捕获异常并提示。

10.模块与包

10.1 import 导入模块



提示：在导入模块时，每个导入应该独占一行

```
import 模块名1
import 模块名2
```

我们自己创建的.py 结尾的文件都可以作为模块导入。

包 包括若干模块

模块 包括若干类

类 包括若干方法

10.2 __name__ 保证测试代码在导入后不被执行

__name__ 属性

- `__name__` 属性可以做到，测试模块的代码 只在测试情况下被运行，而在 被导入时 不会被执行！
- `__name__` 是 Python 的一个内置属性，记录着一个 字符串
- 如果 是被其他文件导入的，`__name__` 就是 模块名
- 如果 是当前执行的程序 `__name__` 是 `__main__`

在很多 Python 文件中都会看到以下格式的代码：

```
# 导入模块
# 定义全局变量
# 定义类
# 定义函数

# 在代码的最下方
def main():
    # ...
    pass

# 根据 __name__ 判断是否执行下方代码
if __name__ == "__main__":
    main()
```

def main():

pass #这部分内容是用于测试代码的代码，如果自己的脚本就没有测试代码，完全不必理会这点。

10.3 包的新建

案例演练

1. 新建一个 `hm_message` 的包
2. 在目录下，新建两个文件 `send_message` 和 `receive_message`
3. 在 `send_message` 文件中定义一个 `send` 函数
4. 在 `receive_message` 文件中定义一个 `receive` 函数
5. 在外部直接导入 `hm_message` 的包

__init__.py

- 要在外界使用 包 中的模块，需要在 `__init__.py` 中指定 对外界提供的模块列表

```
# 从 当前目录 导入 模块列表
from . import send_message
from . import receive_message
```

需要一个 `__init__.py` 文件。把你所有像提供的模块全部以

`from . import [模块名]` 格式写入。

这样别人才可以导入你的包。

10.4 制作一个 python 的安装压缩包

3.1 制作发布压缩包步骤

1) 创建 setup.py

- setup.py 的文件

```
from distutils.core import setup

setup(name="hm_message", # 包名
      version="1.0", # 版本
      description="itheima's 发送和接收消息模块", # 描述信息
      long_description="完整的发送和接收消息模块", # 完整描述信息
      author="itheima", # 作者
      author_email="itheima@itheima.com", # 作者邮箱
      url="www.itheima.com", # 主页
      py_modules=["hm_message.send_message",
                  "hm_message.receive_message"])
```

有关字典参数的详细信息，可以参阅官方网站：

<https://docs.python.org/2/distutils/apiref.html>

2) 构建模块

```
$ python3 setup.py build
```

3) 生成发布压缩包

```
$ python3 setup.py sdist
```

注意 setup.py 文件放置的位置，目录树结构如下：

包名

setup.py

包中所有模块目录

__init__.py

模块 1.py

模块 2.py

10.5 安装和卸载 python 包

3.2 安装模块

```
$ tar -zxvf hm_message-1.0.tar.gz  
$ sudo python3 setup.py install
```

卸载模块

直接从安装目录下，把安装模块的目录删除就可以

```
$ cd /usr/local/lib/python3.5/dist-packages/  
$ sudo rm -r hm_message*
```

11 目录或文件操作

11.1 文件打开方式

访问方式	说明
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。如果文件不存在，抛出异常
w	以只写方式打开文件。如果文件存在会被覆盖。如果文件不存在，创建新文件
a	以追加方式打开文件。如果该文件已存在，文件指针将会放在文件的结尾。如果文件不存在，创建新文件进行写入
r+	以读写方式打开文件。文件的指针将会放在文件的开头。如果文件不存在，抛出异常
w+	以读写方式打开文件。如果文件存在会被覆盖。如果文件不存在，创建新文件
a+	以读写方式打开文件。如果该文件已存在，文件指针将会放在文件的结尾。如果文件不存在，创建新文件进行写入

11.2 文件对象的读取写入方法

文件对象的常用方法

方法名	说明
read([size])	从文件中读取size个字节或字符的内容返回。若省略[size]，则读取到文件末尾，即一次读取文件所有内容
readline()	从文本文件中读取一行内容
readlines()	把文本文件中每一行都作为独立的字符串对象，并将这些对象放入列表返回
write(str)	将字符串str内容写入文件
writelines(s_list)	将字符串列表s_list写入文本文件，不添加换行符
seek(offset[, whence])	把文件指针移动到新的位置，offset表示相对于whence的位置： offset: 为正往结束方向移动，为负往开始方向移动 whence不同的值代表不同含义： 0: 从文件头开始计算（默认值） 1: 从当前位置开始计算 2: 从文件尾开始计算
tell()	返回文件指针的当前位置
flush()	把缓冲区的内容写入文件，但不关闭文件
close()	把缓冲区的内容写入文件，同时关闭文件，释放文件对象相关资源

11.3 with 上下文管理器

with open('filename', '打开方式') as 中间变量:

变量 = 中间变量.readline()

11.4 os.path 模块操作目录的函数

以下函数都是 os.path.abspath(path) 这样应用

os.path模块操作目录相关函数

函数	说明
abspath(path)	用于获取文件或目录的绝对路径
exists(path)	用于判断文件或目录是否存在，如果存在返回True，否则返回False
join(path, name)	将目录与目录或者文件名拼接起来
splitext()	分离文件名和扩展名
basename(path)	从一个目录中提取文件名
dirname(path)	从一个路径中提取文件路径，不包括文件名
isdir(path)	用于判断是否为路径

其他小技巧

1. a,b = b,a

交换赋值，利用的是元组的性质，右边相当于 (b,a)，括号省略了，然后就是利用元组性质，同时给多个变量赋值

2.局部变量和全局变量

- 局部变量 是在 函数内部 定义的变量，只能在函数内部使用
- 全局变量 是在 函数外部定义 的变量（没有定义在某一个函数内），所有函数 内部 都可以使用这个变量

提示：在其他的开发语言中，大多 不推荐使用全局变量 -- 可变范围太大，导致程序不好维护！

- 为了保证所有的函数都能够正确使用到全局变量，应该 将全局变量定义在其他函数的上方

globe [变量] 声明为全局变量

3.1 局部变量

- 局部变量 是在 函数内部 定义的变量，只能在函数内部使用
- 函数执行结束后，函数内部的局部变量，会被系统回收
- 不同的函数，可以定义相同的名字的局部变量，但是 彼此之间 不会产生影响

3.利用元组接受函数返回的多个值

```
def measure():  
    """测量温度和湿度"""  
  
    print("测量开始...")  
    temp = 39  
    wetness = 50  
    print("测量结束...")  
  
    # 元组 - 可以包含多个数据，因此可以使用元组让函数一次返回多个值  
    # 如果函数返回的类型是元组，小括号可以省略  
    # return (temp, wetness)  
    return temp, wetness  
  
result = measure()  
print(result)
```

gl_temp, gl_wetness = measure() 可以一次型使用多个变量接受元组的多个值

```
result = measure()  
print(result)  
  
# 需要单独的处理温度或者湿度 - 不方便  
print(result[0])  
print(result[1])  
  
# 如果函数返回的类型是元组，同时希望单独的处理元组中的元素  
# 可以使用多个变量，一次接收函数的返回结果  
gl_temp, gl_wetness = measure()  
  
print(gl_temp)  
print(gl_wetness)
```


4.打印文本高亮显示

```
1 | '\033[{i};1m这里写需要输出打印的内容\033[0m'# i是指打印颜色的编号
```

```
1 # 格式:
2 #      设置颜色开始 : \033[显示方式;前景色;背景色m
3 # 说明:
4 # 前景色          背景色          颜色
5 # -----
6 # 30              40              黑色
7 # 31              41              红色
8 # 32              42              绿色
9 # 33              43              黄色
10 # 34             44              蓝色
11 # 35             45              紫红色
12 # 36             46              青蓝色
13 # 37             47              白色
14 # 显示方式      意义
15 # -----
16 # 0             终端默认设置
17 # 1             高亮显示
18 # 4             使用下划线
19 # 5             闪烁
20 # 7             反白显示
21 # 8             不可见
```

1.函数详解

函数形式: `dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)`

参数:

axis: 轴。0或'index', 表示按行删除; 1或'columns', 表示按列删除。

how: 筛选方式。'any', 表示该行/列只要有一个以上的空值, 就删除该行/列; 'all', 表示该行/列全部都为空值, 就删除该行/列。

thresh: 非空元素最低数量。int型, 默认为None。如果该行/列中, 非空元素数量小于这个值, 就删除该行/列。

subset: 子集。列表, 元素为行或者列的索引。如果axis=0或者'index', subset中元素为列的索引; 如果axis=1或者'column', subset中元素为行的索引。由subset限制的子区域, 是判断是否删除该行/列的条件判断区域。

inplace: 是否原地替换。布尔值, 默认为False。如果为True, 则在原DataFrame上进行操作, 返回值为None。

语法	说明	表达式实例	完整匹配的字符串
字符			
一般字符	匹配自身	abc	abc
.	匹配任意除换行符“\n”外的字符。 在DOTALL模式中也能匹配换行符。	a.c	abc
\	转义字符，使后一个字符改变原来的意思。 如果字符串中有字符“需要匹配，可以使用“或者字符集“[]”。	a\c a\\c	a.c a\c
[...]	字符集（字符类）。对应的位置可以是字符集中任意字符。 字符集中的字符可以逐个列出，也可以给出范围，如[abc]或[a-c]。第一个字符如果是^则表示取反，如[^abc]表示不是abc的其他字符。 所有的特殊字符在字符集中都失去其原有的特殊含义。在字符集中如果要使用[]、-或^，可以在前面加上反斜杠，或把[]、-放在第一个字符，把^放在非第一个字符。	a[bcd]o	abe ace ade
预定义字符集（可以写在字符集[...]中）			
\d	数字：[0-9]	a\d.c	a1c
\D	非数字：[^0-9]	a\D.c	abc
\s	空白字符：[<空格>\r\n\t\f\v]	a\s.c	a c
\S	非空白字符：[^<空格>]	a\S.c	abc
\w	单词字符：[A-Za-z0-9_]	a\w.c	abc
\W	非单词字符：[^A-Za-z0-9_]	a\W.c	a c
数量词（跟在字符或[...]之后）			
*	匹配前一个字符0次或无限次。	abc*	ab abccc
+	匹配前一个字符1次或无限次。	abc+	abc abccc
?	匹配前一个字符0次或1次。	abc?	ab abc
{m}	匹配前一个字符m次。	ab{2}c	abbc
{m,n}	匹配前一个字符m至n次。 m和n可以省略：若省略m，则匹配0至n次；若省略n，则匹配m至无限次。	ab{1,2}c	abc abbc
? +? ?? {m,n}?	使 + ? {m,n}变成非贪婪模式。	示例将在下文中介绍。	
边界匹配（不消耗待匹配字符串中的字符）			
^	匹配字符串开头。 在多行模式中匹配每一行的开头。	^abc	abc
\$	匹配字符串末尾。 在多行模式中匹配每一行的末尾。	abc\$	abc
\A	仅匹配字符串开头。	\Aabc	abc
\Z	仅匹配字符串末尾。	abc\Z	abc
\b	匹配\w和\W之间。	a\b bc	a bc
\B	[^\b]	a\B bc	abc
取模、分组			
	代表左右表达式任意匹配一个。 它总是先尝试匹配左边的表达式，一旦成功匹配则跳过匹配右边的表达式。 如果 没有被包括在()中，则它的范围是整个正则表达式。 被括起来的表达式将作为分组，从表达式左边开始每遇到一个分组的左括号“（”，编号+1。 另外，分组表达式作为一个整体，可以替换数量词。表达式中的 仅在块级中有效。	abc def	abc def
(...)	被括起来的表达式将作为分组，从表达式左边开始每遇到一个分组的左括号“（”，编号+1。 另外，分组表达式作为一个整体，可以替换数量词。表达式中的 仅在块级中有效。	(abc){2} a(123 456)c	abcaabc a456c
(?P<name>...)	分组，除了原有的编号外再指定一个额外的别名。	(?P<id>abc){2}	abcaabc
\<number>	引用编号为<number>的分组匹配到的字符串。	(\d)abc\1	1abc1 5abc5
(?P= name)	引用别名为<name>的分组匹配到的字符串。	(?P<id>\d)abc(?P=id)	1abc1 5abc5
特殊构造（不作为分组）			
(?:...)	(...)的不分组版本，用于使用 或后跟数量词。	(?:abc){2}	abcaabc
(?ilmsux)	ilmsux的每个字符代表一个匹配模式，只能用在正则表达式的开头，可选多个。匹配模式将在下文中介绍。	(?)abc	AbC
(?#...)	#后的内容将作为注释被忽略。	abc(?:#comment)123	abc123
(?=...)	之后的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	a(?=\d)	后面是数字的a
(?!...)	之后的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	a(?!\d)	后面不是数字的a
(?<=...)	之前的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	(?<=\d)a	前面是数字的a
(?<!=...)	之前的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	(?<!=\d)a	前面不是数字的a
(?(id/name)yes-patternno-pattern)	如果编号为id/别名为name的组匹配到字符，则需要匹配yes-pattern，否则需要匹配no-pattern。 no-pattern可以省略。	(\d)abc(?(1)\d abc)	1abc2 abc abc abc ...