**Daily Report**

26th February, 2018

1. **多任务**

多进程模式；

多线程模式；

多进程+多线程模式

线程是最小的执行单元，而进程由至少一个线程组成

多进程和多线程的程序涉及到同步、数据共享的问题，编写起来更复杂

1. **Fork（）（适用于UNIX/LINUX系统）**

fork()调用一次，返回两次，因为操作系统自动把当前进程（称为父进程）复制了一份（称为子进程），然后，分别在父进程和子进程内返回

子进程永远返回0，而父进程返回子进程的ID。这样做的理由是，一个父进程可以fork出很多子进程，所以，父进程要记下每个子进程的ID，而子进程只需要调用getppid()就可以拿到父进程的ID

1. **正则表达式**

用\d可以匹配一个数字，\w可以匹配一个字母或数字，\s可以匹配一个空格或tab

.可以匹配任意字符，用\*表示任意个字符（包括0个），用+表示至少一个字符，用?表示0个或1个字符，用{n}表示n个字符，用{n,m}表示n-m个字符

对于如同‘-’等特殊字符，需要使用‘\’进行转义

可以使用[]表示范围，如：[0-9a-zA-Z\\_]可以匹配一个数字、字母或者下划线

A|B可以匹配A或B

^表示行的开头，如^\d表示必须以数字开头

$表示行的结束，\d$表示必须以数字结束

可以使用re.match()对两个str值进行匹配，成功时返回match，失败时返回None

1. **分组**

如果正则表达式中定义了组，就可以在Match对象上用group()方法提取出子串来。

注意到group(0)永远是原始字符串，group(1)、group(2)……表示第1、2、……个子串

1. **预编译**

如果一个正则表达式要重复使用几千次，出于效率的考虑，我们可以预编译该正则表达式，接下来重复使用时就不需要编译这个步骤了，直接匹配：

>>> import re

# 编译:

>>> re\_telephone = re.compile(r'^(\d{3})-(\d{3,8})$')

# 使用：

>>> re\_telephone.match('010-12345').groups()

('010', '12345')

>>> re\_telephone.match('010-8086').groups()

('010', '8086')

1. **Datetime**

Datetime是pytohn处理日期和时间的标准库。使用方法如：

>>> from datetime import datetime

>>> now = datetime.now() # 获取当前datetime

>>> print(now)

2015-05-18 16:28:07.198690

也可以直接使用参数构造一个datetime：

>>> dt = datetime(2015, 4, 19, 12, 20) # 用指定日期时间创建datetime

>>> print(dt)

2015-04-19 12:20:00

1. **Datetime转换为timestamp**

只需要调用timestamp（）方法即可，获得的结果为浮点数，若有小数点，小数位表示为毫秒数

1. **Timestamp转换为datetime**

使用datetime提供的fromtimestamp（）作用

使用datetime提供的utcfromtimestamp（）功能，能够转换到UTC标准时区的时间

1. **Strptime（）**

当用户输入的时间为字符串时，需要进行处理，转换方法为使用springtime（），需要一个日期和时间的格式化字符串

>>> from datetime import datetime

>>> cday = datetime.strptime('2015-6-1 18:19:59', '%Y-%m-%d %H:%M:%S')

>>> print(cday)

2015-06-01 18:19:59

1. **Datetime转换为str**

转换方法是通过strftime()实现的，同样需要一个日期和时间的格式化字符串

>>> from datetime import datetime

>>> now = datetime.now()

>>> print(now.strftime('%a, %b %d %H:%M'))

Mon, May 05 16:28

1. **datetime加减**

对日期和时间进行加减实际上就是把datetime往后或往前计算，得到新的datetime。加减可以直接用+和-运算符，不过需要导入timedelta这个类：

>>> from datetime import datetime, timedelta

>>> now = datetime.now()

>>> now

datetime.datetime(2015, 5, 18, 16, 57, 3, 540997)

>>> now + timedelta(hours=10)

datetime.datetime(2015, 5, 19, 2, 57, 3, 540997)

>>> now - timedelta(days=1)

datetime.datetime(2015, 5, 17, 16, 57, 3, 540997)

>>> now + timedelta(days=2, hours=12)

datetime.datetime(2015, 5, 21, 4, 57, 3, 540997)

1. **Namedtuple**

属于collections模块，用于创建自定义的tuple对象，规定了tuple元素的个数，并可以用属性而不是索引来引用tuple的某个元素，如：

>>> from collections import namedtuple

>>> Point = namedtuple('Point', ['x', 'y'])

>>> p = Point(1, 2)

1. **Deque**

deque是为了高效实现插入和删除操作的双向列表，适合用于队列和栈：

>>> from collections import deque

>>> q = deque(['a', 'b', 'c'])

>>> q.append('x')

>>> q.appendleft('y')

>>> q

deque(['y', 'a', 'b', 'c', 'x'])

deque除了实现list的append()和pop()外，还支持appendleft()和popleft()

1. **Defaultdict**

如果希望key不存在时，返回一个默认值，就可以用defaultdict：

>>> from collections import defaultdict

>>> dd = defaultdict(lambda: 'N/A')

>>> dd['key1'] = 'abc'

>>> dd['key1'] # key1存在

'abc'

>>> dd['key2'] # key2不存在，返回默认值

'N/A'

注意默认值是调用函数返回的，而函数在创建defaultdict对象时传入

1. **OrderedDict**

对于dict做迭代时，无法确定key的顺序，若要保持key的顺序，可以使用OrderedDict，它的key会按照插入的顺序排列，而不是key本身排序

1. **Counter**

使用counter能够简单的计数，实际为dict的一个子类

1. **struct.pack**

能够将任意数据类型变为bytes

1. **Hashlib**

提供了常见的摘要算法，如MD5，SHA1等

当使用MD5时，如：

import hashlib

md5 = hashlib.md5()

md5.update('how to use md5 in python hashlib?'.encode('utf-8'))

print(md5.hexdigest())

且当数据量较大时，可以分块多次调用update（），最后计算结果相同；对于其他的摘要算法用法相似，如SHA1，只需将md5换为sha1即可