目录

P1 课程介绍

热门，大数据人工智能，经济、金融方面也有应用。

P2 我们为什么要学习编程

无处不在的计算机，购物、餐饮、商旅出行、资料查找。。。

P3 程序是什么?

“程序”就是做一件事情或者解决一个问题所采取的一系列固定步骤。
计算机以文字、图像、声音、动画等各种形式向人反馈执行命令的结果。
程序设计语言的发展：从机械编程到高级语言。
计算思维：用机器、程序解决问题。

P4 Python 语言的历史和现状

十大最流行的计算机语言之一；语法简洁，极大地提高了生产力；跨平台，代码可读性高；软件开源，可以被自由传播和分享
应用领域广：网站、图像、系统、大数据、人工智能
Python 语言继承了多种优秀语言的特性，是一种高级动态、完全面向对象的语言；函数、模块、数字、字符串都是对象；并且完全支持继承、重载、派生、多继承，有益于增强源代码的复用性。
面向对象和模块设计的模式，Python 成为数据科学和机器学习的最常用语言，能与高性能的 C 语言程序对接

P5 Python 语言的历史和现状

Windows、Linux、MacOS
IDLE 自带集成开发环境
集成开发环境 – PyCharm
Anaconda，科学计算，conda 包管理器

P6 第一个 Python 程序

两种方式：1.交互式运行 Python 语句；2.保存源文件运行程序
IDLE 里编程的步骤："File->New File" 打开文件编辑窗口，输入代码，"File->Save" 保存文件，"Run->Run Module" 运行，查看结果

P7 集成开发工具 PyCharm

PyCharm。
Check out from Version Control，版本控制的高级功能

P8 上机练习：体验 Python 程序

上机。

P9 Python 程序设计风格

Python 程序的风格：优雅、明确、简单
便于阅读：代码强制缩进
良好的编程规范：变量、函数、类命名；注释和文档；一些编程设计上的良好风格
The Zen of Python。
优美 > 丑陋
明确 > 隐晦 （1）（显式地罗列引用的包，不要给方法搞不清楚的其他影响，约定优于配置必须是公认的某种约定）
简单 > 复杂
复杂 > 繁复 （2）（Simple > Complex > Complicated > Chaotic）
扁平 > 嵌套
稀疏 > 拥挤（3）（代码别写的太挤太长）
可读性很重要（4）
固然代码的实用性比洁癖更重要，
所谓的“特例”也往往没有特殊到必须违背上述规则的程度
除非必要，否则不要无故忽视异常（5）
如果遇到模棱两可的逻辑，请不要自作聪明地瞎猜。
应该提供一种，且最好只提供一种，一目了然的途径
立刻着手好过永远不做。然而，永远不做也好过闷头蛮干；
倘若你的实现很难解释，它一定不是个好主意，倘若你的实现一目了然，它可能是个好主意
命名空间好！

P10 数据对象及其组织

数据(data)是信息的表现形式和载体，对现实世界实体和概念的抽象
大数据(big data)：Volume、Velocity、Variety、Value、Veracity
复杂数据类型：图像、音频、视频
整数 int、浮点数 float、复数 complex、逻辑值 bool、字符串 str
容器类型用来组织这些值：列表 list、元组 tuple、集合 set、字典 dict
数据结构：需要建立各种各样的数据组织，以便提高计算效率

P11 计算和控制流

计算与流程：对现实世界处理和过程的抽象。
赋值语句用来实现处理与暂存：表达式计算、函数调用、赋值
定义语句：def、class，源代码的各种复用
所有可调用的事物称为 callable

P12 基本类型：数值

整数类型：int
不限制大小

运算符	功能	备注
m + n	加法	
m - n	减法	
m * n	乘法	
m // n	整数除法	结果是商的整数部分
m / n	除法	“真”除法，得到小数
m % n	求余数	
divmod(m, n)	求整数除法和余数	会得到两个整数，一个是 m // n，另一个是 m % n
m ** n	求乘方	整数 m 的 n 次方
abs(m)	求绝对值	

连续比较判断 >>> 7 > 3 >= 3 True
直接用二进制、八进制和 十六进制来表示整数，只要加一个前缀用以标识几进制即可

进制	表示	例子
十进制 decimal	无前缀数字	367
二进制 binary	0b 前缀	0b101101111
八进制 octal	0o 前缀	0o557
十六进制 hexadecimal	0x 前缀	0x16f

浮点数类型：float
17 位有效数字
特点：科学记数法、有效位数
特性：进制转换导致精度误差

```
>>> 4.2 + 2.1 == 6.3
False
>>> 4.2 + 2.1
6.300000000000001
```

复数类型
相互比较只能比较是否相等

```
>>> abs((3 + 3j) - (4 + 4j))
1.4142135623730951
```

数学函数：math 模块（整数、浮点数），cmath 模块（复数）
删除空格：

P13 基本类型：逻辑值

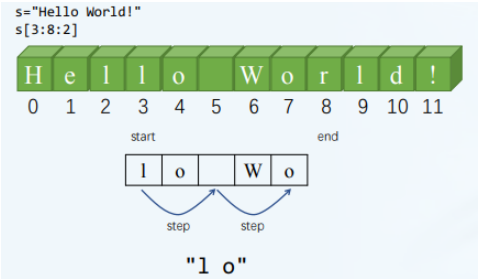
逻辑(bool)类型，True、False
and, or, not，not 最高，and 次之，or 最低
0 是“假”，所有非 0 的数值都是“真”
字符串中空串("")是“假”，所有非空串都是“真”
序列（包括字符串）空序列是“假”，所有非空的序列都是“真”
None 表示“无意义”或“不知道”，也是“假”

P14 基本类型：字符串

文字字符包括各种语言文字字符，双引号或者单引号都可以表示字符串，但必须成对；多行字符串用三个连续单引号表示

转义字符	描述
(在行尾时)	续行符
\\	反斜杠符号
\'	单引号
\"	双引号
\a	响铃
\b	退格(Backspace)
\e	转义
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\oyy	八进制数 yy 代表的字符，例如：\o12 代表换行，数值参见 ASCII 码表
\xyy	十进制数 yy 代表的字符，例如：\x0a 代表换行，数值参见 ASCII 码表
\other	其它的字符以普通格式输出

0，1，2…
…-3，-2，-1
len()，slice：s[start:end:step] （左闭右开）



+：连接，*：重复
判断字符串内容是否相同(==)，判断字符串中是否包含某个字符串(in)
str.strip：去掉字符串前后的所有空格，内部的空格不受影响
str.lstrip：去掉字符串前部（左部）的所有空格
str.rstrip：去掉字符串后部（右部）的所有空格
str.strip：去掉字符串前后的所有空格，内部的空格不受影响
str.lstrip：去掉字符串前部（左部）的所有空格

str.rstrip：去掉字符串后部（右部）的所有空格
split：分割；join：合并；upper/lower/swapcase：大小写相关；just/center/rjust：排版左中右对齐；replace：替换子串

序列（sequence） | 能够按照整数顺序排列的数据
可以通过从 0 开始的连续整数来索引单个对象；
可以执行切片，获取序列的一部分；
可以通过 len 函数来获取序列中包含多少元素；
可以用加法“+”来连接为更长的序列；
可以用乘法“*”来重复多次，成为更长的序列；
可以用“in”来判断某个元素是否在序列中存在

P15 变量和引用

名字（name）和数值的关联，称为引用
关联数值后的名字，就拥有了数据的值（value） 和类型（type）
一个数值可以和多个名字关联
最基本的赋值语句形式 =
合并赋值 a = b = c = 1
按顺序依次赋值 a, b, c = 7, 8, 9
简写赋值语句 price += 1；price *= 1.5；price /= 3 +4；

P16 上机练习：基本数据类型

上机。

P17 容器类型：列表和元组

列表可以删除、添加、替换、重排序列中的元素（**可变类型**）
元组是不能再更新（**不可变**）序列
创建列表：方括号法[]，指明类型法 list()
创建元组：圆括号法()，指明类型法 tuple()
列表：append 操作/insert 操作/extend 操作，pop 操作/remove 操作/clear 操作，reverse / sort，reversed / sorted

方法名称	使用例子	说明
append	alist.append(item)	列表末尾添加元素
insert	alist.insert(i,item)	列表中i位置插入元素
pop	alist.pop()	删除最后一个元素，并返回其值
pop	alist.pop(i)	删除第i个元素，并返回其值
sort	alist.sort()	将表中元素排序
reverse	alist.reverse()	将表中元素反向排列
del	del alist[i]	删除第i个元素
index	alist.index(item)	找到item的首次出现位置
count	alist.count(item)	返回item在列表中出现的次数
remove	alist.remove(item)	将item的首次出现删除

列表和元组的操作
+：列表和元组的操作
*：复制 n 次，生成新列表 / 元组
len()：列表 / 元组中元素的个数
索引：alist[n]或 atuple[n]
切片：alist[start : end : step]或 atuple[start : end : step]
查找：1. in 操作：判断某个元素是否存在于列表/元组中 2. index 操作：指定的数据在列表/元组的哪个位置 3. count 操作：指定的数据在列表/元组中出现过几次
计算：sum 函数：将列表中所有的数据元素累加
min/max 函数：返回列表中最小/最大的数据元

P18 容器类型：字典

标签（key）和数据值（value）
字典容器中保存着一系列的 key-value 对
通过键值 key 来索引元素 value
花括号法和指明类型法 student = {}; student = dict() s = {'a':'b', 'dd':12}
数据项（item）
字典中保存的各个标签-数据值（key-value） l 标签和数据值之间用冒号“：”连接
批量添加 item：student = dict.fromkeys(("name", "age"))
dict：可变类型
value 没有顺序，可以是任意类型，甚至也可以是字典
key 可以是任意不可变类型（数值 / 字符串 / 元组）
合并字典：update 方法，
增长字典：“关联”操作 update 操作：以 key=value 的形式批量添加数据项（d0.update(key = value)）
缩减字典：
del 操作：删除指定标签的数据项 del student["age"]
pop 操作：删除指定标签的数据项并返回数据值
popitem 操作：删除并返回任意一个数据项
clear 操作：清空字典
字典大小 len 函数
索引：dict[key]，获取值、更新值，== dict.get(key)
获取字典的标签、数据值和数据项：
keys 函数：返回字典中的所有标签；
values 函数：返回字典中的所有数据值；
items 函数：将每个数据项表示为二元元组，返回 所有的数据项
查找：
in 操作：判断字典中是否存在某个标签
in 操作和 values 函数的组合：判断字典中是否存在某个数据

P19 容器类型：集合

集合是不重复元素的无序组合（dict 的 key 集合）
创建集合：{}或者 set()　　{a,b,c}
用 set()创建空集 可用 set()从其它序列转换生成集合
自动忽略重复的数据，**不能加入可变类型数据**(加入字符串自动拆成单个字符的集合)
增长集合 add：添加一个数据
　　update：批量添加数据
缩减集合 remove/discard（不存在不会出错）：删除指定数据
　　pop：删除任意数据并返回值
　　clear：清空集合
集合大小 len 函数
in 判断元素是否属于集合
pop 删除数据元素的同时，返回它的值
迭代循环（遍历） for a in aset:
运算：

运算	运算符	新集合方法	更新原集合方法
并	a | b	union	update
交	a & b	intersection	intersection_update
差	a - b	difference	difference_update
对称差	a ^ b	symmetric_difference	symmetric_difference_update

关系判定 <=,=,>=,>：是否是子集/真子集/超集/真超集
交集 isdisjoint()：两集合交集是否为空
使用场合：去除重复的数据项、判断元素是否在一组数据中

P20 可变类型和不可变类型

不可变(immutable)类型：一旦创建就无法修改数据值的数据类型
整数、浮点数、复数、字符串、逻辑值、 元组
可变(mutable)类型：可以随时改变的数据类型
列表、字典、集合
灵活性强会花费一些计算或者存储的代价 去维持这些强大的功能
可变类型的变量操作：多个变量通过赋值引用同一个可变类型对象时，通过其 中任何一个变量改变了可变类型对象，其它变量也随之改变

```
>>> a[0]='hello'
>>> a
['hello', 2, 3, 4]
>>> b
['hello', 2, 3, 4]
```

P21 建立复杂的数据结构

列表、元组、字典每种类型中，都可以通过方括号[]对单个元 素进行访问
对于列表和元组，方括号里是整型的偏移量
对于字典，方括号里的是键
都返回元素的值
将这些内置的数据结构自由地组合成 更大、更复杂的结构
嵌套列表/元组
list_of_lists = [[1,2,3], ['Hello','Python'], [True,False]]
tuple_of_lists = ([1,2,3], ['Hello','Python'], [True,False])
嵌套字典
元素可以是任意类型，键值可以是任意不可变类型

P22 输入和输出

input(prompt)，返回值是字符串
print([object,...][,sep=' '][,end='\n'] [,file=sys.stdout])
sep：表示变量之间用什么字符串隔开，缺省是空格
end：表示以这个字符串结尾，缺省为换行
file：指定了文本将要发送到的文件、标准流或其它类似的文件的对象；默认是 sys.stdout，默认把对象打印到 stdout 流，并且添加了一些自动的格式化(可以自定义分隔符和行末符)

```
>>> '(%4d):K:%s' % (12, 'Hello')
'(  12):K:Hello'
>>> '(%04d):K:%10s' % (12, 'Hello')
'(0012):K:     Hello'
```

```
>>> print (1, 23, 'Hello', sep=',')
1,23,Hello
```

P23 上机练习：容器类型操作

上机。

P24 自动计算过程

五大部件
赋值语句的执行语义为：计算表达式的值，存储起来，贴上变量标签以便将来引用。与计算机运行过程中的“计算”和“存储”相对应
“控制器确定下一条程序语句”即对应“控制”

P25 控制流程

决定“下一条语句”的机制，在程序设计语言 中称作“控制流程”
顺序结构、条件分支结构、循环结构
简单类型是实体对象，容器类型是结构，将实体对象进行各种组织和编排

类别	对象实体	容器
数据	• 数值类型（整数、浮点数、复数） • 逻辑类型 • 字符串类型	• 列表 • 元组 • 字典 • 集合
计算	• 赋值语句	• 顺序结构 • 条件分支结构 • 循环结构

P26 控制流：条件分支语句（if）

if <逻辑表达式 1>:
    <语句块 1>
……
else:
    <语句块 2>
if 和 else 都是“保留字”
“逻辑表达式”是指所有运算的结果为逻辑类型 （True 或 False）的表达式
“语句块”就是条件满足后执行的一组语句
冒号表示语句层次
语句块缩进
各种类型中某些值会自动被转换为 False，其它值则是 True：None, 0, 0.0, '', [], (), {}, set()
用 elif 语句进行判定
if <逻辑表达式 1>:
    <语句块 1>
elif <逻辑表达式 2>:
    <语句块 2>
elif <逻辑表达式 3>:
    <语句块 3>
… …
else:
    <语句块 n>

P27 控制流：条件循环（while）

根据需要对一系列操作进行设定次数或者设定条件的重复，这样的控制流程，就称作循环结构
作用：能持续对大量数据进行处理，在长时间里对一些未知状况进行连续监测循环结构
循环前提的类型：从某个容器或者潜在的数据集中逐一获取数据项，取不到数据项，循环的

前提就消失；只要逻辑表达式计算结果为真(True)，循环的前提就存在
与条件分支结构的区别：循环结构会多次检查循环前提
while 循环语法
while <逻辑表达式>:
    <语句块>
    break #跳出循环
    continue #略过余下循环语句
    <语句块>
else: #条件不满足退出循环，则执行
    <语句块>

P28 控制流：迭代循环（for)

可迭代对象表示从这个数据对象中可以逐个取出数据项赋值给“循环变量”
可迭代对象有很多类型，如字符串、列表、元组、字 典、集合等，也可以有后面提到的生成器、迭代器等
range(<起点>,<终点>,<步长>)，起点默认 0，步长默认 1，因此可以只写终点该数列，包含起点整数，而不包含终点整数
range 类型的对象 直接当做序列转换为 list 或者 tuple 等容器类型
continue 和 break 均作用于离它们最近的一层循环
break 是跳出当前循环并结束本层循环
continue 是略过余下循环语句并接着下一次循环

P29 上机练习：基本的计算程序

上机。

P30 代码组织：函数(def)

封装：容器是对数据的封装、函数是对语句的封装、类是对方法和属性的封装
定义函数
用 def 语句创建一个函数
用 return 关键字指定函数返回的值
def <函数名> (<参数表>):
    <缩进的代码段>
    return <函数返回值>
调用函数
<函数名>(<参数>)
无返回值：<函数名>(<参数表>)
返回值赋值：v = <函数名>(<参数表>)
局部、全局变量

在一个函数内部使用某个全局变量同名的变量，若未在函数内定义，会用全局变量的值，但是无法进行修改，Python 会在函数内部创建一个同名的局部变量
global 关键字可以在函数中改变全局变量的值
map(func, list1, list2….)
map(function, iterable, ...)
返回一个将 function 应用于 iterable 中每一项并输出其结果的迭代器。 如果传入了额外的 iterable 参数，function 必须接受相同个数的实参并被应用于从所有可迭代对象中并行获取的项。 当有多个可迭代对象时，最短的可迭代对象耗尽则整个迭代就将结束。
lambda 表达式
返回一个匿名函数
lambda <参数表>:<表达式>

P31 代码组织：函数的参数

形式参数(**parameter**) 函数创建和定义过程中，函数名后面括号里的参数
实际参数(**argument**) 函数在调用过程中传入的参数
定义函数时，参数有两种
一种是在参数表中写明参数名 key 的参数，固定了顺序和数量的固定参数
def func(key1, key2, key3…):
def func(key1, key2=value2…):
一种是定义时还不知道会有多少参数传入的可变参数
def func(*args): #不带 key 的多个参数
def func(**kwargs): #key=val 形式的多个参数

```
def func_test2(*args):
    for arg, i in zip(args, range(len(args))):
        print("arg%d=%s" % (i, arg))


print("====func_test2")
func_test2(12, 34, 'abcd', True)
```

```
39  # 可以随意传入0个或多个带名参数
40  def func_test3(**kwargs):
41      for key, val in kwargs.items():
42          print("%s=%s" % (key, val))
43
44
45  print("====func_test3")
46  func_test3(myname="Tom", sep="comma", age=23)
```

调用函数
一种是没有名字的位置参数 func(arg1, arg2, arg3…) 会按照前后顺序对应到函数参数传入
一种是带 key 的关键字参数 func(key1=arg1, key2=arg2…) 由于指定了 key，可以不按照顺序对应
若混用，所有**位置参数必须在前，关键字参数必须在后**

P32 上机练习：创建并调用函数

上机。

P33 引用扩展模块

每个扩展名为.py 的 Python 程序都是一个 独立的模块(Module)
模块能定义函数、类和变量，让你能够有逻辑地组织你的 Python 代码段
包(package)是放在一个文件夹里的模块集合
import <模块> [as <别名>]
在调用模块中的函数的时候，需要加上模块的命名空间
可以给导入的命名空间替换一个新的名字，引用方法：<模块>.<名称>
from <模块> import <函数>
数字和数学模块
numbers：数字抽象基类；math：数学函数；cmath：复数的数学函数；decimal：十进制定点和浮点算术；fractions：有理数；random：生成伪随机数；statistics：数学统计功能；
数据类型
datetime：基本日期和时间类型；calendar：与日历相关的一般功能；collections：容器数据类型；heapq：堆队列算法；bisect：数组二分算法；array：高效的数值数组；weakref：弱引用；types：动态类型创建和内置类型的名称；copy：浅层和深层复制操作；pprint：格式化输出；reprlib：备用 repr()实现；enum ： 支持枚举
。。。
命名空间(namespace)
表示标识符(identifier)的可见范围
一个标识符可以在多个命名空间中定义，在不同命名空间中的含义互不相干
dir()函数：列出名称的属性
help()函数：显示参考手册

P34 时间相关：datetime 模块

可对 date、time、datetime 三种时间模式进行单独管理
datetime.date() 处理日期（年月日）
datetime.time() 处理时间（时分秒、毫秒）
datetime.datetime() 处理日期+时间
datetime.timedelta() 处理时段（时间间隔）
获取今天的日期
datetime.date.today()
datetime.datetime.now()
修改日期格式
使用 strftime 格式化

```
>>> datetime.date.today().strftime('%Y-%m-%d %H:%M:%S')
'2018-09-04 00:00:00'
```

datetime.datetime.isoformat()

```
>>> d = datetime.datetime.now()
>>> d
datetime.datetime(2018, 9, 4, 11, 4, 11, 387056)
>>> d.isoformat()
'2018-09-04T11:04:11.387056'
```

时间戳是指格林威治时间 1970 年 01 月 01 日 00 时 00 分 00 秒起至现在的总秒数
将日期转换成时间戳

```
>>> import time,datetime
>>> today = datetime.date.today()
>>> today.timetuple()
time.struct_time(tm_year=2018, tm_mon=9, tm
_mday=4, tm_hour=0, tm_min=0, tm_sec=0, tm_
wday=1, tm_yday=247, tm_isdst=-1)
>>> time.mktime(today.timetuple())
1535990400.0
>>> datetime.date.fromtimestamp(1535990400.
0)
datetime.date(2018, 9, 4)
```

timetuple 函数 将时间转换成 struct_time 格式
time.mktime 函数 返回用秒数来表示时间的浮点数
将时间戳转换成日期 datetime.date.fromtimestamp()

```
>>> datetime.date.fromtimestamp(1535990400.
0)
datetime.date(2018, 9, 4)
```

timedelta()方法 表示两个时间点的间隔

```
>>> hours = today - datetime.timedelta(hours=1)
>>> print(hours)
2018-09-04 10:08:05.530315
```

P35 时间相关：calendar 模块

calendar 模块
常用函数
calendar.calendar(<年>)返回多行字符串
calendar.month(<年>,<月>)返回多行字符串
calendar.prmonth(<年>,<月>)相当于 print (calendar.month (<年>,<月>))
calendar.prcal(<年>)相当于 print (calendar.prcal (<年>))

calendar.monthcalendar()
返回某一年的某一个月份日历，是一个嵌套列表
最里层的列表含有七个元素，代表一周(从周一到周日)
如果没有本月的日期，则为 0

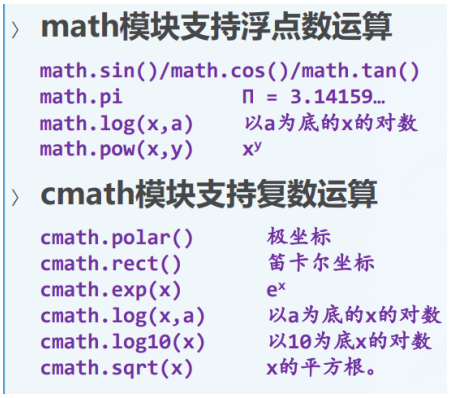
判别闰年
普通闰年：能被 4 整除但不能被 100 整除的年份
世纪闰年：能被 400 整除的年份
calendar.isleap(<年>)

计算某月共有多少天，从周几开始（calendar.monthrange(2021,7)）
从 0 开始，依次为周一、周二…
计算某天是周几（calendar.weekday(2021,7,21)）
返回值为 0~6，依次对应的是周一到周日

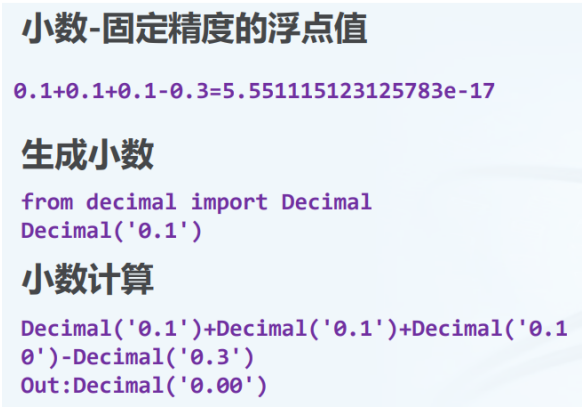
P36 时间相关：time 模块

获取时间戳 》 time.time()方法
获取当前时间 time.asctime();time.ctime()
将元组数据转化为日期 t = (2018, 8, 13, 11, 42, 31, 0, 0, 0) ;time.asctime(t)
年、月、日、时、 分、秒、周几、一年中的第几天、是否为 夏令时间
struct_time 类
Out:time.struct_time(tm_year=2018, tm_mon=8, tm_mday=13, tm_hour=12, tm_min=24, tm_sec=11, tm_wday=0, tm_yday=225, tm_isdst=0)
索引获取时间信息 t = time.localtime() year = t[0]
time.sleep()

P37 基本算术模块



decimal 模块（解决精度导致的误差问题）



fractions 模块

分数-实现了一个有理数对象
生成分数
    from fractions import Fraction
    Fraction(1,4)/Fraction('0.25')
浮点数转换为分数
    Fraction.from_float(1.75)
尽管可以把浮点数转换为分数，在某些情况下，这么做会有不可避免的精度损失，因为这个

数字在其最初的浮点形式上是不精确的

random 模块

random.seed(a=None)

random()，生成范围在[0,1)之间的随机实数

uniform()，生成指定范围的内的随机 浮点数

randint(m,n)，生成指定范围[m,n]内的整数

randrange(a,b,n)，可以在[a,b)范围内，按 n 递增的集合中随机选择一个数

getrandbits(k)，生成 k 位二进制的随机整数

choice()，从指定序列中随机选择一个元素

sample()，能指定每次随机元素的个数

shuffle()，可以将**可变序列**中所有元素随机排序

P38 持久化模块

标准库模块

pickle 任意 Python 对象格式化和解格式化

dbm 实现一个可通过键访问的文件系统，以存储字节串

shelve 按照键把 pickle 处理后的对象存储到一个文件中

shelve 模块 提供基本的存储操作，通过构造一个简单的数据库， 像操作字典一样按照键存储和获取本地的 Python 对象，使其可以跨程序运行而保持持久化

键 必须是字符串，且是唯一的

值 任何类型的 Python 对象

与字典类型的区别：打开 shelve，修改后关闭

数据处理：不支持类似 SQL 的查询工具，通过键获取到保存在文件的对象

d = shelve.open(filename) open 函数在调用时返回一个 shelf 对象，通过该对象可以存储内容

操作：

d[key] = data

value = d[key]

del d[key]

d.close()

P39 文本文件读写

普通流式文件

open()函数

f = open(filename[,mode[,buffering]])

f: open()返回的文件对象

filename：文件的字符串名

mode：可选参数，打开模式和文件类型

buffering：可选参数，文件的缓冲区，默认为-1

mode 第一个字母表明对其的操作： ‘r’表示读模式；‘w’表示写模式；‘x’表示在文件不存在的情况下新创建并写文件；‘a’表示在文件末尾追加写内容；‘+’表示读写模式
mode 第二个字母是文件类型
‘t’表示文本类型；‘b’表示二进制文件
文件的写操作
f.write(str)
f.writelines(strlist)：写入字符串列表
文件的读操作
f.read()
f.readline()：返回一行
f.readlines()：返回所有行、列表
文件关闭的作用是终止对外部文件的连接，同时将缓存区的数据刷新到硬盘上
调用 close()方法 f.close()
使用上下文管理器(context manager) 确保在退出后自动关闭文件
with open(‘textfile’,’rt’) as myfile:
    myfile.read()

CSV

纯文本文件，以“ ， ”为分隔符，值没有类型，所有值都是字符串
文件读取 - reader re = csv.reader()
接受一个**可迭代**对象（比如 csv 文件），能**返回一个生成器**，可以从其中解析出内容
文件读取 - DictReader re = csv.DictReader()
与 reader 类似，但返回的**每一个单元格都放在一个元组的值**内
文件写操作 w = csv.writer()
w.witerow(rows)
当文件不存在时，自动生成，支持单行写入和多行写入
字典数据写入 w = csv.DictWriter()
w.writeheader() 表头
w.writerow(rows)

Excel

一个 Workbook 对象代表一个 Excel 文档，使用该方法创建一个 Worksheet 对象后才能打开一个表
from openpyxl import Workbook
wb = Workbook() ws = wb.active
读取 Excel 文件
from openpyxl import load_workbook
wb = load_workbook(filename)
ws = wb.file.active
获取单元格信息
获取 Cell 对象
    c = wb[‘sheet’][‘A1’]
    c = wb[‘sheet’].cell(row=1,column=1)
c.coordinate：返回单元格坐标
c.value：返回单元格的值
c.row：返回单元格所在的行坐标

c.column：返回单元格所在列坐标

PDF

包含了 PdfFileReader、PdfFileMerger、PageObject 和 PdfFileWriter 四个主要类
能进行读写、分割、合并、文件转换等多种操作
只能提取文本并返回为字符串，而无法提取图像、图表
读取 PDF 文件

```
readFile = open('test.pdf','rb')
pdfFileReader = PdfFileReader(readFile)
```

pdfFileReader 类
.getNumPages():计算 PDF 文件总页数
.getPage(index):检索指定编号的页面
PDF 文件的写操作 writeFile = 'output.pdf' pdfFileWriter = PdfFileWriter()
pdfFileWriter 类
.addPage(pageObj):根据每页返回的 PageObject，写入到文件
.addBlankPage():在文件的最后一页后面写入一个空白页,保存到新文件
合并多个文档

```
pdf_merger = PdfFileMerger()
pdf_merger.append('python2018.pdf')
pdf_merger.merge(20, 'insert.pdf')
pdf_merger.write('merge.pdf')
```

单个页面操作 – PageObject 类
.extractText()：按照顺序提取文本
.getContents()：访问页面内容
.rotateClockwise(angle)：顺时针旋转
.scale(sx,sy)：改变页面大

P44 面向对象：什么是对象

Python 中的所有事物都是以对象形式存在
对象(object) 既表示客观世界问题空间中的某个具体事物，又表示软件系统解空间中的基本元素
对象 = 属性 + 方法 对象以 id 作为标识，既包含数据(属性)，也包含代码(方法)，是某一类具体事物的特殊实例
对象的名称
赋值语句给予对象以名称，对象可以有多个名称（变量引用），但**只有一个 id**
例：a = complex(1, 2)
对象实现了属性和方法的封装，是一种数据抽象机制提高了软件的重用性、灵活性、扩展性
引用形式
<对象名>.<属性名>
动态特性，对象可以随时增加或者删除属性或者方法
面向对象编程(OOP)是一种程序设计范型，同时也是一种程序开发方法
程序中包含各种独立而又能互相调用的对象，每个对象都能接受、处理数据并将数据传递给其他对象，数据依附于对象

P45 面向对象：类的定义与调用

类(class)是对象的模版，封装了对应现实实体的性质和行为
实例对象(Instance Objects)是类的具体化
封装性、继承性、多态性
类名用大写字母开头，函数用小写字母开头
class 语句
class <类名>:
    <一系列方法的调用>
类的初始化
class <类名>:
    def __init__(self, <参数表>):
    def <方法名>(self, <参数表>):
__init__()是一个特殊的函数名，用于根据类的定义创建实例对象，第一个参数必须为 self
<类名>(<参数>)
调用类会创建一个对象，(注意括号！)
obj = <类名>(<参数表>)
返回一个对象实例，类方法中的 self 指这个对象实例
点(.)操作符来调用对象里的方法

P46 面向对象：类定义中的特殊方法

特殊方法(special method) 也被称作**魔术方法**(magic method)
在类定义中实现一些特殊方法，可以方便地 使用 python 中一些内置操作 所有特殊方法的名称以**两个下划线(__)开始和结束**
构造器 __init__(self,[…) 对象的构造器，实例化对象时调用
析构器 __del__(self,[…) 销毁对象时调用
算术操作符
__add__(self,other)：使用+操作符
__sub__(self,other)：使用-操作符
__mul__(self,other)：使用*操作符
__div__(self,other)：使用/操作符
反运算 **当左操作数不支持相应的操作时被调用** other + self
__radd__(self,other)，__rsub__(self,other)
__rmul__(self,other)，__rdiv__(self,other)
大小比较
__eq__(self,other)：使用==操作符
__ne__(self,other)：使用!=操作符
__lt__(self,other)：使用操作符
__le__(self,other)：使用<=操作符
__ge__(self,other)：使用>=操作符
字符串操作
不仅数字类型可以使用像+(__add__())和-（__sub__())的数学运算符，例如字符串类型可以使

用+进行拼接，使用*进行复制
__str__(self)：自动转换为字符串
__repr__(self)：返回一个用来表示对象的字符串
__len__(self)：返回元素个数

P47 自定义对象的排序

可变对象能排序
列表方法 sort()
对原列表进行排序，改变原列表内容；如果列表中的元素都是数字，默认按升序排序 通过添加参数 reverse = True 可改为降序排列；如果元素都是字符串，则会按照字母表顺序排列
通用函数 sorted()
排好序的列表副本，原列表内容不变
列表中的所有元素都是同一种类型时才工作
特殊方法__lt__
每种数据类型可以定义特殊方法 def __lt__(self, y) 返回 True 视为比 y“小”，排在前；返回 False 视为比 y“大”，排在后
只要类定义中定义了特殊方法__lt__，任 何自定义类都可以使用 x<y 这样的比较
直接调用列表 sort 方法可以根据__lt__定义排序
直接检验 Student 对象的大小：S[i]<s[j]
另外可以定义其它比较符：__gt__等

P48 面向对象：类的继承

继承(inheritance) 如果一个类别 A 继承自另一个类别 B，就把继承者 A 称为子类，被继承的类 B 称为父类、基类或超类
代码复用
如果两个类具有“一般-特殊”的逻辑关系，那么特殊类就可以作为一般类的 “子类”来定义，从“父类”继承属性 和方法
class <子类名>(<父类名>):
    def <重定义方法>(self,…):
覆盖(Override)
子类对象可以调用父类方法，除非这个方法在子类中重新定义了。如果子类同名方法覆盖了父类的方法，仍然还可以调用父类的方法
子类还可以添加父类中没有的方法和属性

```
class GasCar(Car):
    def __init__(self, name, capacity):  # 名称和排量
        super().__init__(name)  # 父类初始化方法，只有名称
        self.capacity = capacity  # 增加了排量属性
```

Super().func()
在类定义中，所有方法的首个参数一般都是 self
self 的作用 在类内部，实例化过程中传入的所有数据都赋给这个变量
self 实际上代表对象实例

<对象>.<方法>(<参数>)
等价于：
<类>.<方法>(<对象>, <参数>)
这里的对象就是 self

```
gcar.run(200.0)
GasCar.run(gcar, 200.0)
```

P49 上机练习：类和对象

上机。

P50 例外处理

程序的逻辑错误、用户输入不合法等都会引发异常，但它们不会导致程序崩溃，可以利用 python 提供的异常处理机制
语法错误：SyntaxError
除以 0 错误：ZeroDivisionError
列表下标越界：IndexError
类型错误：TypeError
访问变量不存在：NameError
字典关键字不存在：KeyError
未知的变量属性：AttributeError
try:
<检测语句>
except <错误类型> [as e]:
<处理异常>
finally:
<语句块>
try: # 为缩进的代码设置陷阱
except: # 处理错误的代码
针对不同异常可以设置多个 except
finally: # 无论出错否，都执行的代码
如果 try 语句块运行时没有出现错误，会跳过 except 语句块执行 finally 语句块的内容
try:
<检测语句>
except <错误类型> [as e]:
<处理异常>
else:
<语句块>
else: # 没有出错执行的代码

P51 推导式

推导式是从一个或者多个迭代器快速 简洁地创建数据结构的一种方法
将循环和条件判断结合，从而避免语 法冗长的代码
可以用来**生成列表、字典和集合**
列表推导式
[<表达式> for <变量> in <可迭代对象> if <逻辑条件>]
字典推导式
{<键值表达式>:<元素表达式> for <变量> in <可迭代对象> if <逻辑条件>}
集合推导式
{<元素表达式> for <变量> in <可迭代对象> if<逻辑条件>}

```
>>> {'K%d'%(x,):x**3 for x in range(10)}
{'K2': 8, 'K8': 512, 'K5': 125, 'K6': 216, 'K3': 27, 'K9': 729, 'K0': 0,
'K7': 343, 'K1': 1, 'K4': 64}
```

```
>>> {x+y for x in range(10) for y in range(x)}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17}
```

生成器推导式
与推导式一样语法：(<元素表达式> for <变量> in <可迭代对象> if<逻辑条件>)
返回一个生成器对象，也是可迭代对象，但生成器并不立即产生全部元素，仅在要用到元素的时候才生成，可以极大节省内存

P52 生成器函数

生成器(generator)是用来创建 Python 序列的一个对象
使用它可以迭代庞大的序列，且不需要在内存中创建和存储整个序列
通常生成器是为迭代器产生数据的迭代器的一种实现
如果要创建一个比较大的序列，生成器推导式将会比较复杂，一行表达式无法容纳， 这时可以定义生成器函数
生成器函数与普通函数生成器函数的定义与普通函数相同，只是将 return 换成了 yield
yield 与 return
yield 语句 立即返回一个值，下一次迭代生成器函数时，**从 yield 语句后的语句继续执行**，直到再次 yield 返回，或终止
return 语句 终止函数的执行，下次调用会重新执行函数
协同程序 可以运行的独立函数调用，函数可以暂停或挂起， 并在需要的时候从离开的地方继续或重新开始

```
def even_number(max):
    n = 0
    while n < max:
        yield n
        n += 2

for i in even_number(10):
    print (i)
```

P53 上机练习：生成器

P54 图像处理库

Pillow 库
打开图像
image.open() Pillow 库能自动根据文件内容确定格式
若图片在程序目录下，则不需要附带路径，直接将图 像名+文件格式作为参数
处理图像 image 模块中提供了大量处理图像的方法
存取或显示图像 im.show() im.save()
thumbnail 函数 thumbnail(size, Image.ANTIALIAS)
参数 size 为一个元组，指定生成缩略图的大小，直接对内存中的原图进行了修改，但是修改完后的图片需要保存，处理后的图片不会被拉伸

```
# 应用模糊滤镜:
im2 = im.filter(ImageFilter.BLUR)
```

```
#打开程序目录下的图片cat
img = Image.open('cat.jpg')
#设置待添加文字大小为200，字体为宋体
font = ImageFont.truetype('simsun.ttc',100)
#在img上创建可绘图对象draw
draw = ImageDraw.Draw(img)
#添加红色文字“可爱的小猫”
draw.text((100,10), '可爱的小猫',(255,0,0),font=font)
#保存照片
img.save('cat1.jpg','jpeg')
```

小程序：PIL 生成验证码
随机字母、随机颜色

```
image = Image.new('RGB', (width, height), (255, 255, 255))
# 创建Font对象:
font = ImageFont.truetype('Arial.ttf', 36)
# 创建Draw对象:
draw = ImageDraw.Draw(image)
# 填充每个像素:
for x in range(width):
    for y in range(height):
        draw.point((x, y), fill=rndColor())
# 输出文字:
for t in range(4):
    draw.text((60 * t + 10, 10), rndChar(), font=font, fill=rndColor2())
# 模糊:
image = image.filter(ImageFilter.BLUR)
image.save('code.jpg', 'jpeg')
```

P55 Web 服务框架

Web 服务器会处理与浏览器客户端交互的 HTTP 协议具体细节，但对具体内容的处理还需要自己编写代码 一个 Web 框架至少要具备处理浏览器客户端请求和服务端响应的能力
框架的特性
路由 解析 URL 并找到对应的服务端文件或者 Python 服务器代码
模板 把服务端数据合并成 HTML 页面。
认证和授权 处理用户名、密码和权限
Session 处理用户在多次请求之间需要存储的数据

Flask
App = Flask(__name__)
@app.route() 路由
Def fun(): 动作
    Return page 返回页面模板
关于表单的扩展库 使用 Flask-WTF 时，能把每个表单都抽象成一个类

```
from wtforms import StringField
from wtforms.validators import DataRequired

class MyForm(Form):
    user = StringField('Username', validators=[DataRequired()])
```

```
@app.route('/login', methods=('GET', 'POST'))
def login():
    form = MyForm()
    if form.validate_on_submit():
        # if form.user.data == 'admin':
        if form.data['user'] == 'admin':
            return 'Admin login successfully!'
        else:
            return 'Wrong user!'
    return render_template('login.html', form=form)
```

P56 网络爬虫

爬虫是按照一定规则，自动地提取并保存网页中信息的程序
requests 库
Python 实现的一个简单易用的 HTTP 库，支持 HTTP 持久连接和连接池、SSL 证书验证、cookies 处理、流式上传等
向服务器发起请求并获取响应，完成访问网页的步骤
简洁、容易理解，是最友好的网络爬虫库
http 请求类型
requests.request()：构造一个请求
requests.get()：获取 HTML 网页
requests.head()：获取 HTML 网页头信息
requests.post()：提交 POST 请求
requests.put()：提交 PUT 请求
requests.patch()：提交局部修改请求
requests.delete()：提交删除请求
requests.options()：获取 http 请求
返回的是一个 response 对象
response 对象
包含服务器返回的所有信息，例如状态码、编码形式、文本内容等；也包含请求的 request 信息
.status_code：HTTP 请求的返回状态
.text：HTTP 响应内容的字符串形式
.content：HTTP 响应内容的二进制形式
.encoding：(从 HTTP header 中)分析响应内容的编码方式
.apparent_encoding：(从内容中)分析响应内容的编码方式
定制请求头 requests 的请求接口有一个名为 headers 的参数，向它传递一个字典来完成请求头定制

设置代理 一些网站设置了同一 IP 访问次数的限制，可以在发送请求时指定 proxies 参数来替换代理，解决这一问题

Beautiful Soup

页面解析器 使用 requests 库下载了网页并转换成字符串后，需要一个解析器来处理 HTML 和 XML，解析页面格式，提取有用的信息

解析器类型

解析器	使用方法	优势
python标准库	BeautifulSoup(markup, “html.parser”)	- Python的内置标准库 - 文档容错能力强
lxml HTML解析器	BeautifulSoup(markup, “lxml”)	- 速度快 - 文档容错能力强
lxml XML解析器	BeautifulSoup(markup, [“lxml-xml”]) BeautifulSoup(markup, “xml”)	- 速度快 - 唯一支持XML的解析器
Html5lib	BeautifulSoup(markup, “html5lib”)	- 最好的容错性 - 以浏览器的方式解析文档 - 生成HTML5格式的文档

解析完了之后可以用 beautifulsoup 来搜索

搜索方法

find_all(name, attrs, recursive, string, **kwargs) 返回文档中符合条件的所有 tag，是一个列表

find(name, attrs, recursive, string, **kwargs) 相当于 find_all()中 limit = 1，返回一个结果 name：对标签名称的检索字符串；attrs：对标签属性值的检索字符串；recursive：是否对子节点全部检索，默认为 True string: <>... 中检索字符串；**kwargs：关键词参数列表；

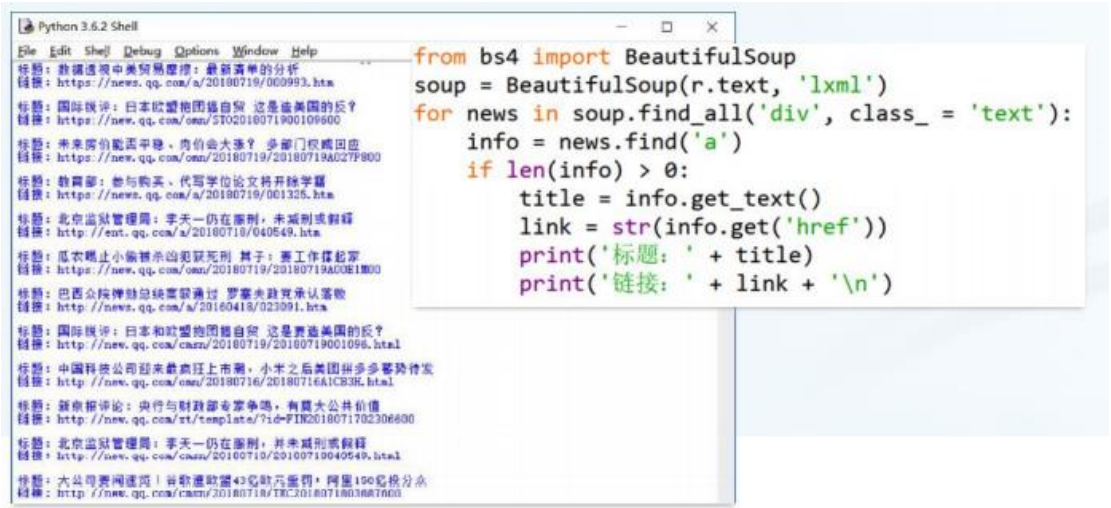
流程：

分析结构-》发送请求获得页面-》解析页面

HTML 代码-网页解析器

Json 数据-json 模块，转换成 Json 对象

二进制数据-以 wb 形式写入文件，再做进一步处理 此处使用 bs4



P57 绘制数据图表

NumPy

矩阵计算 创建矩阵 a = np.matrix([])
矩阵求逆 a.I
矩阵转置 a.T
矩阵乘法 a*b 或 np.dot(a,b)
对象属性
np.shape 数组形状，矩阵则为 n 行 m 列
np.size 对象元素的个数
np.dtype 指定当前 numpy 对象的整体数据
matplotlib
numpy 库的 linspace( )函数生成数组
numpy.linspace(<start>,<stop>,<num>)
matplotlib 库的 plot( )函数用来画图，**其中图形的 x 坐标默认为数组索引**

颜色

颜色	表示方法	颜色	表示方法
blue	'b'	yellow	'y'
cyan	'c'	white	'w'
green	'g'	red	'r'
black	'k'	magenta	'm'

线型与点型

线型	表示方法
实线	-
短线	--
短点相间线	-.
虚点线	:

点型	表示方法
圆形	o
叉	x、+
三角形	^、v、<、>
五角星	*

```
plt.plot(x, np.sin(x), 'r-o',
         x, np.cos(x), 'g--')
```

坐标轴标签 plt.xlabel()、plt.ylabel()
图形标题 plt.title()
散点图 scatter(x,y)
直方图：函数 hist(x, n) x 是横坐标，n 是条状图的数量

P58 上机练习：高级扩展模块应用

上机