

目录

- 目录
- p1 为什么要学习编程
- p2 程序是什么
- p3 python历史及现状
- p4 运行环境
- p5 pycharm介绍
- p6 第一个python程序
- p7 上机练习
- p8 Python程序设计风格
- p9数据对象及其组织
 - 数据类型归纳
 - 数据组织方式
- p10 计算和控制流
 - 运算语句
 - 控制流语句
 - 定义语句
- p11 基本数据类型：数值
 - 整数类型：int
 - 浮点数类型：float
 - 复数类型
 - math模块
 - cmath模块
- p12 基本类型：逻辑值
 - 逻辑运算
 - 类型对应的真值
- p13 基本类型：字符串
 - 常见操作
 - 序列
- p14 变量和引用
 - 命名规范
 - 其他赋值语句
- p16容器类型：列表和元组
 - 创建
 - 列表独有的操作
 - 两者共有的操作
- p17字典类型
 - 创建
 - 更新
 - 访问
- p18集合类型
 - 创建
 - 更新
 - 访问
 - 集合运算

- 集合关系
- p19可变类型和不可变类型
 - 不可变
 - 可变
 - 可变类型的赋值引用问题
- p20建立复杂的数据结构
- p21输入输出IO
 - 输入
 - 输出
 - 格式化输出
- p23自动计算过程
 - 冯诺依曼机
 - 五大构成部分
 - 程序执行过程时
- p24 控制流程
- p25 条件分支
- p26 while 循环
 - 循环嵌套
- p27for循环
 - range函数
- p29 代码组织：函数（def）
 - 函数的定义
 - 函数调用
 - 变量的作用域
 - map函数
 - 匿名函数lambda
- p30 函数的参数
 - 形参与实参的概念
 - 定义函数的参数
 - 方法一，固定所有参数
 - 方法二，可变参数
 - 调用函数的参数
- p32 引用扩展模块
 - 引用方式
- p33 datetime模块
 - 主要类
 - 获取当前时间
 - 日期格式化
 - 时间戳
- p34 calendar模块
- p35 time模块
- p36 算数模块
 - math模块
 - cmath模块
 - decimal模块
 - fractions模块

- random模块
- p37 持久化模块: shelve
- p38 文件读写
 - 打开文件
 - 文件读写操作
 - 文件关闭
 - csv 文件读写
 - Excel文件
 - Pdf文件
- p43面向对象
 - 对象的概念
- p44 类的定义与调用
 - 类的定义
 - 实例化对象
- p45 类定义中的特殊方法
 - 对象构造器
 - 析构器
 - 算数运算
 - 字符串操作
- p46 自定义对象的排序
 - 自定义比较方法
- p47 类的继承
- p49 例外处理
- p50推导式
 - 列表推导式
 - 字典推导式
 - 集合推导式
 - 生成器推导式
- p51生成器函数
- p53 图像处理库-Pillow
 - 图片打开
 - 图片处理
 - thumbnail
 - filter
 - 添加文本
 - 图片保存
- p54 web框架
 - 基本概念
- p55 网络爬虫
 - request
 - Beautiful Soup
- p56数据可视化
 - Numpy 库
 - matplotlib

p1 为什么要学习编程

p2 程序是什么

做一件事情或者解决一个问题所采取的一系列固定步骤

p3 python历史及现状

1989年诞生

继承多种优秀语言特性，高级动态面向对象的语言，支持继承、重载、派生、多继承等。

在自动化运维，问开发，机器学习，科学建模等领域广泛应用。

p4 运行环境

- 自带的IDE: shell-idle
- pycharm等
- Anaconda 针对科学计算的安装包，附带了常用第三方库，使用conda作为包管理器

p5 pycharm介绍

p6 第一个python程序

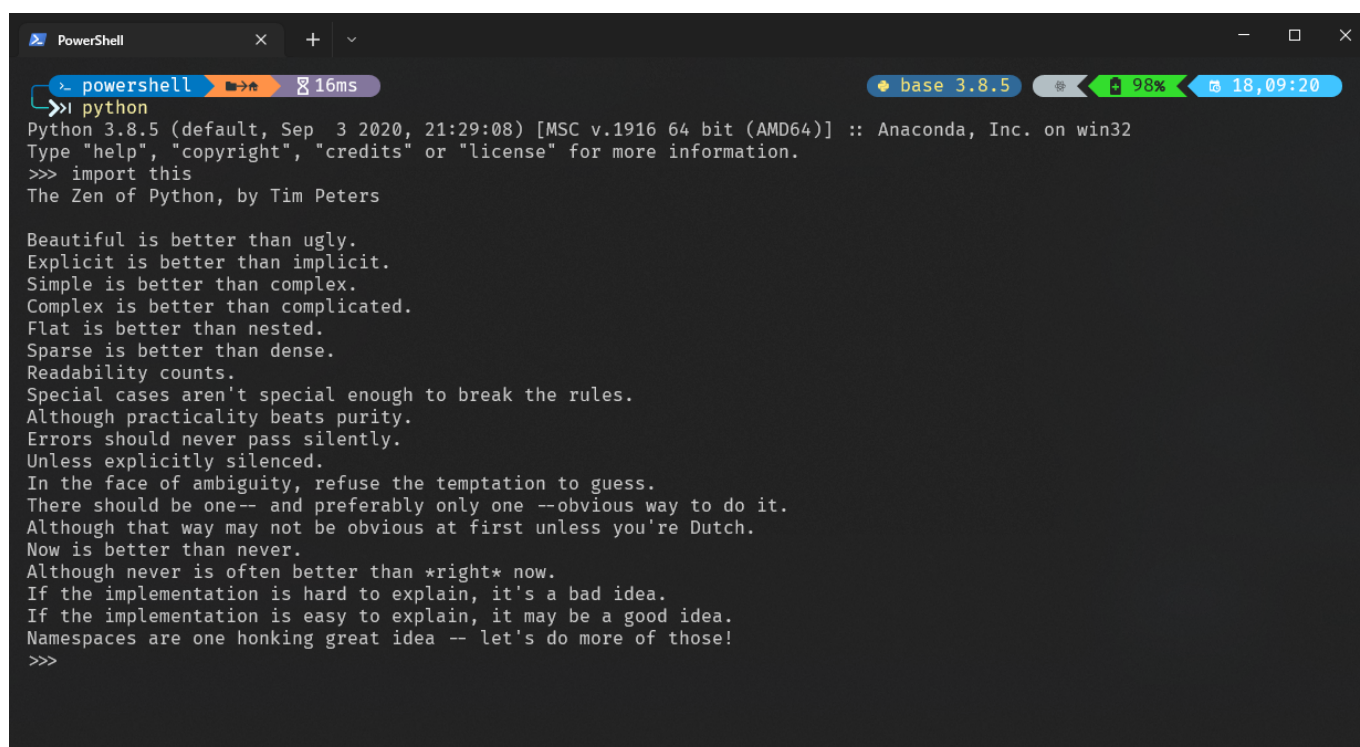
- idle交互式运行
- 编写源文件运行

p7 上机练习

p8 Python程序设计风格

优雅、明确、简单

代码强制缩进



```
PowerShell
> powershell
>>> python
Python 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

p9数据对象及其组织

数据是信息的表现形式和载体 是对现实世界实体和概念的抽象

数据类型归纳

- 简单类型
 - int
 - float
 - complex
 - bool
 - str
- 容器类型
 - list
 - tuple
 - set
 - dict

数据组织方式

1. 无组织
2. 标签式组织数据
3. 队列、栈、树、图等

p10 计算和控制流

各种类型的数据对象，可以通过各种运算组织成复杂的表达式

运算语句

- 表达式计算 $12*3$
- 函数调用 `math.sqrt(2)`
- 赋值 `x=2`

控制流语句

- 顺序结构
- 条件分支
- 循环结构

定义语句

- `def` 函数定义
- `class` 类定义

p11 基本数据类型：数值

整数类型：int

python中int 不限制大小!

除 // (整数除法) , m**n(乘方) 外基本与C语言保持一致

比较 : > < >= <= ==

数制表示:

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

最多17位有效数字

可以用科学计数法表示: 如 1.2234e+10

注意不要试图比较浮点类型的大小

复数类型

python 内置了复数类型, 支持常见的的加减乘除, 以及乘方运算。

a.imag 获得虚部, a.real 获得实部

```
PowerShell
> powershell
>>> python
Python 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a=1+3j
>>> b=2+4j
>>> a+b
(3+7j)
>>> a**b
(0.0461397137607447+0.04945614122637528j)
>>> a.imag
3.0
>>> a.real
1.0
>>> |
```

! 注意: 复数之间仅支持比较相等

math模块

用于整数和浮点数的计算

```
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb',
'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp',
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma',
'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt',
'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm',
'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'tau', 'trunc']
>>>
```

cmath模块

可用于计算复数!

如平面直角坐标系和极坐标之间的转换

p12 基本类型：逻辑值

其实就是bool类型

逻辑运算

- and
- or
- not

优先级关系：not>and>or

类型对应的真值

- 整数 浮点数 复数 除0位false外其他均为true
- 字符串类型中 空串 ("") 为false 否则为true
- 序列类型，空序列为false
- None 为false

p13 基本类型：字符串

双引号和单引号都可以表示字符串!

三引号可以用于多行字符串的表示

\实现转义字符

字符串编号：从0开始，可随机访问

常见操作

- len 获得长度

- 切片 `s[0:10:1]`
- 拼接 `+`
- 重复 `s*3`
- 比较 `==`
- 判断包含 `in`
- 删空格
 - `str.strip` 去除前后所有空格
 - `str.lstrip` 去除前面的空格
 - `str.rstrip` 去除后面的空格
- 判断字母数字（除了比较asc码外）
 - `isalpha` 是否全部由字母构成
 - `isdigit` 是否全部由数字构成
 - `isalnum` 是否只包含字母和数字
- `split`([分割字符]) 分割，返回列表
- `join` 将列表合并为字符串
- `upper/lower/swapcase` 大小写转化
- `ljust/center/rjust` 排版对其方式
- `s.replace("a","ab")` 子串替换

序列

字符串是一种序列

什么是序列？

能够按照整数顺序排列的数据

序列的内部结构

- 可以从0开始索引
- 可以切片
- 可以用`len`获得长度
- 可用`+` 拼接
- 可用`*` 重复
- 可用`in` 判断子元素存在性

p14 变量和引用

命名规范

- 字母和数字及下划线组合而成，区分大小写
- 无其他特殊字符
- 首字母必须为字母

变量类型随着指向的数据对象类型改变而改变

其他赋值语句

- 合并赋值 `a=b=c=1`
- 顺序赋值 `a,b=1,2`

- 简写赋值语句
 - `a += 1`
 - `a *= 1`

p16容器类型：列表和元组

列表和元组的区别：列表可变，元组不可变，列表比元组多一些操作

创建

列表：`[]`,或`list()` 元组：`()`,或`tuple()`

列表独有的操作

- 增长列表
 - `append`
 - `insert`
 - `extend`
- 缩减列表
 - `pop` 默认删除最后一个，可指定位置
 - `remove` 移走某个元素
 - `clear`
- 重新组织
 - `reverse` 翻转
 - `sort` 从大到小排序
 - `reversed` 返回翻转的新列表，不改变原列表
 - `sorted` 返回重排的列表

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

两者共有的操作

- `+` 连接
- `*` 重复

- len() 获得长度
- 索引
- 切片
- 查找
 - in
 - index 获得指定元素出现位置
 - count
- 统计
 - sum
 - min
 - max

p17字典类型

key-value结构

创建

```
a={} a=dict()
```

元素可以是任意类型，也可以是字典（有点类似json）

更新

- update({})可以合并、增长字典
- del 删除指定标签的数据项
- pop
- popitem 删除并返回任意一个数据项
- clear

访问

可以直接索引，也可以用get方法拿到key对应的value

- keys 以列表的方式返回所有key
- values 返回所有数据值
- items 以二元组列表的方式返回所有数据项

p18集合类型

特点：**无序且不重复**

创建

- {}
- set()

注意set中不可加入可变类型的数据

更新

- add
- update
- remove/discard
- pop
- clear

访问

- in
- pop 删除一个并返回值，所以可以复制一个set然后一直pop到空，以此来遍历
- 迭代 for i in set

集合运算

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

集合关系

- \leq 子集
- $=$ 真子集
- $>$ 超集
- \supseteq 真超集

isdisjoint() 方法可以判断两个集合是交集是否为空

p19可变类型和不可变类型

不可变

- 整数
- 浮点数
- 复数
- 字符串
- 逻辑值
- 元组

可变

- 列表
- 字典
- 集合

可变类型的赋值引用问题

```
a=[1,2,3]
b=a
b.pop()# 也会造成a的改变, 因为他们引用了同一个地址的数据
```

p20建立复杂的数据结构

可以将多种数据结构使用嵌套等方式自由地组合成更复杂的结构

p21输入输出IO

输入

input("提示信息")输入, 注意需要使用强制类型转换来将输入转换为期待的数据类型!

输出

```
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

格式化输出

例如: '%d %s' % (v1, v2)

p23自动计算过程

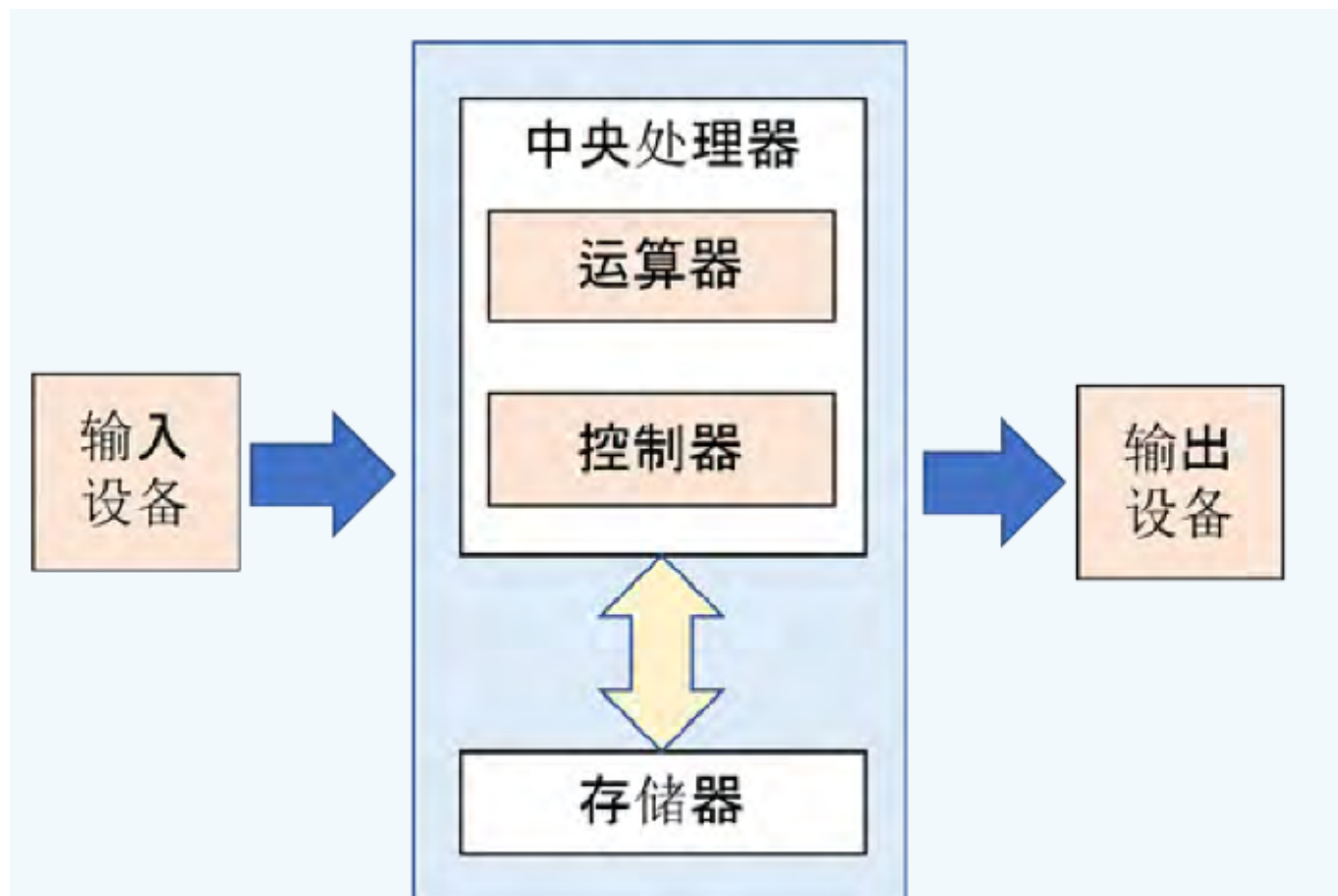
冯诺依曼机

五大构成部分

- 运算器
- 控制器
- 存储器
- 输入设备
- 输出设备

程序执行过程时

1. 控制器从存储器中取出程序语句和额外数据;
2. 数据齐全的语句交给运算器进行算术或者逻辑运算;
3. 运算结果再存回存储器;
4. 控制器确定下一条程序语句, 回到步骤1继续



p24 控制流程

三种流程：

1. 顺序
2. 条件分支
3. 循环

p25 条件分支

```
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>
```

p26 while 循环

```
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>
```

循环嵌套

p27for循环

```
for <循环变量> in <可迭代对象>:
    <语句块1>
    break #跳出循环
    continue #略过余下循环语句
else: #迭代完毕，则执行
    <语句块2>
```

range函数

用于构造数列

- range(<终点>)返回一个从0开始到终点的数列
- range(<起点>,<终点>)从0以外的任何整数开始构建数列
- range(<起点>,<终点>,<步长>)修改数列的步长，通过将步长设置为负数能够实现反向数列

注意：**range构建的数列，包含起点整数，而不包含终点整数**

range 可通过list, tuple 方法转换为容器类型

p29 代码组织：函数（def）

程序中实现明确功能的代码段可以封装成一个函数，以便复用（reuse）

函数的定义

```
def <函数名> (<参数表>):
    <缩进的代码段>
    return <函数返回值>
```

函数调用

```
无返回值: <函数名>(<参数表>)
返回值赋值: v = <函数名>(<参数表>)
```

变量的作用域

- 局部变量是在函数内部定义的变量，仅在该函数内部有效
- 在函数外定义的函数为全局变量，整个代码段有效

一个函数内可以访问全局变量的值，但是**无法修改**，原因是python在内部创建了一个同名的局部变量，对其的修改无法反馈到全局变量上 但是**用global关键字可以在函数中改变全局变量的值。**

map函数

如果需要对列表中每个元素都执行一次某个自定义函数，可以使用map函数实现，如对列表中每个元素都+1可以用如下代码实现:

```
list=[1,2,3,4]
def add1(a):
    return a+1
list=list(map(add1,list))
```

匿名函数lambda

函数只需要用一次，名字也就不重要了！

如上面的要求用lambda实现：

lambda <参数表>:<表达式>

```
list=[1,2,3,4]
list=list(map(lambda a:a+1,list))
```

p30 函数的参数

形参与实参的概念

形参是函数创建和定义过程中，函数名后面括号里的参数

实参是函数在调用过程中传入的参数

定义函数的参数

方法一，固定所有参数

```
def func(key1, key2, key3...):
def func(key1, key2=value2...):
```

方法二，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
```

调用函数的参数

1. 按照顺序传递
2. 使用key关键字可以不按照顺序传递
3. 混用时必须顺序在前，key在后

p32 引用扩展模块

1. 每个python文件都是一个独立的Module
2. package是放在一个文件夹里的模块集合

引用方式

import 模块 as 别名

from 模块 import 函数

dir(名称) 可以获得名称的属性

help(名称) 可以显示参考手册

p33 datetime模块

主要类

- 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）

获取当前时间

- datetime.date.today()
- datetime.datetime.now()

日期格式化

datetime.datetime.strftime()

datetime.datetime.isoformat()(ISO格式)

时间戳

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

timetuple 函数可将时间转为时间戳

datetime.date.fromtimestamp() 可将时间戳转换为时间

timedelta() 用于标识两个时间的时间间隔

p34 calendar模块

跟日历相关的若干函数和类，可以生成文本形式的日历

感觉没有广泛的应用空间，其中的日历计算可能会方便计算

p35 time模块

- time.time() 换取时间戳
- 获取时间
 - time.asctime()
 - time.ctime()
- time.sleep() 程序暂停

p36 算数模块

math模块

math.sin()/math.cos()/math.tan() math.pi $\pi = 3.14159...$ math.log(x,a) 以a为底的x的对数 math.pow(x,y) x^y

cmath模块

主要是复数运算

cmath.polar() 极坐标 cmath.rect() 笛卡尔坐标 cmath.exp(x) e^x cmath.log(x,a) 以a为底的x的对数
cmath.log10(x) 以10为底x的对数 cmath.sqrt(x) x的平方根。

decimal模块

提供了十进制浮点运算支持

为什么要用这个？？

1. Decimal所表示的数是完全精确的。
2. Decimal类包含有效位的概念，因此 $1.30 + 1.20$ 的结果是2.50，保留尾随零以表示有效位。
3. 用户可更改精度

fractions模块

分数模块

生成分数 Fraction(1,4) Fraction('0.25') 注意是个字符串 Fraction.from_float(1.75)

random模块

随机数模块，但其实是一种伪随机，因其结果是可预见的

- random.seed(a=None) 随机数种子 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

p37 持久化模块: shelve

类创建的对象并不是真正的数据库记录存储在内存而不是文件中关闭python, 实例将消失

shelve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

个人理解: 应该就是一种结构化的文件存储, 可以使用类似字典的形式访问

p38 文件读写

打开文件

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

- 返回文件对象
- `filename`: 文件的字符串名
- `mode`: 可选参数, 打开模式和文件类型
 1. 第一个字母是操作类型
 - `'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建并写文件
 - `'a'`表示在文件末尾追加写内容
 - `'+'`表示读写模式
 2. 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
- `buffering`: 可选参数, 文件的缓冲区, 默认为-1

文件读写操作

- `f.write(str)`
- `f.writelines(strlist)`: 写入字符串列表
- `f.read()`
- `f.readline()`: 返回一行
- `f.readlines()`: 返回所有行、列表

文件关闭

方式一:

```
f.close()
```

方式二: 确保在退出后自动关闭文件

```
with open('textfile','rt') as myfile :  
    myfile.read()
```

csv 文件读写

- 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.writerow(rows)
```

字典数据写入

```
w = csv.DictWriter()  
w.writeheader()  
w.writerow(rows)
```

Excel文件

可使用openpyxl库来操作Excel文件

其实个人觉得用pandas可能会更方便一些

Pdf文件

PyPDF2库实现

可实现：

- 读写(仅文本)
- 分割
- 合并
- 文件转换

读取文件

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

- `getNumPages()`:计算PDF文件总页数
- `getPage(index)`:检索指定编号的页面

写文件

```
writeFile = 'output.pdf'  
pdfFileWriter = PdfFileWriter()
```

- `.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

文件合并

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

p43面向对象

对象的概念

对象=属性+方法

Python语言动态的特征,使得对象可以随时增加或者删除属性或者方法

p44 类的定义与调用

什么是类?

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

把类比作模具,对象则是用模具制造出来的零件

面向对象编程的三个最重要特征

1. 封装
2. 继承
3. 多态

Python中约定,类名用大写字母开头(大驼峰),函数用小写字母开头(小驼峰),以便区分。

类的定义

```
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(指对象实例)!

self: 在类内部，实例化过程中传入的所有数据都赋给这个变量

实例化对象

```
obj = <类名>(<参数表>)
```

使用 . 访问对象方法

p45 类定义中的特殊方法

也被称为魔术方法，以两个下划线(__)开始 和结束

对象构造器

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

算数运算

- **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

当左操作数不支持相应的操作时调用反运算

- **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
- **gt**(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

- **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

p46 自定义对象的排序

sort 方法默认升序，添加参数reverse=true后变为降序排序

sorted() 类似sort 但是原列表内容不变，返回新对象

只有当列表中的所有元素都是同一种类型时，sort()和sorted()才会正常工作

自定义比较方法

```
def __lt__(self, y)
```

其实就类似java重写equals方法

- 返回True视为比y“小”，排在前
- 返回False视为比y“大”，排在后

这样之后就可以比较该类中两个对象的大小，并且实现排序

p47 类的继承

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

利用继承可以从已有类中衍生出新的类，添加或修改部分功能新类具有旧类中的各种属性和方法，而不需要进行任何复制

实例

```

class Car:
    def __init__(self, name):
        self.name = name
        self.remain_mile = 0

    def fill_fuel(self, miles): # 加燃料里程
        self.remain_mile = miles

    def run(self, miles): # 跑miles英里
        print(self.name, end=': ')
        if self.remain_mile >= miles:
            self.remain_mile -= miles
            print("run %d miles!" % (miles,))
        else:
            print("fuel out!")

class GasCar(Car): # 继承Car
    def fill_fuel(self, gas): # 加汽油gas升
        self.remain_mile = gas * 6.0 # 每升跑6英里

class ElecCar(Car):
    def fill_fuel(self, power): # 充电power度
        self.remain_mile = power * 3.0 # 每度电3英里

```

```

class <子类名>(<父类名>):
    def <重定义方法>(self,...):

```

子类可覆盖（重写）父类的方法

<对象>.<方法>(<参数>)

等价于：

<类>.<方法>(<对象>, <参数>)

p49 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，在异常出现时及时捕获并从内部消化掉

如果希望掌控意外，就需要在可能出错误的地方设置陷阱捕捉错误

```
try:
    <检测语句>
except <错误类型> [as e]:
    <处理异常>
else:
    <语句块>
finally:
    <语句块>
```

p50推导式

```
>>> [x*x for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
>>> {'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*x for x in range(10)}
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
>>>
>>> {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}
```

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>}

生成器推导式

返回一个生成器对象，但是不马上产生全部元素，仅当用到该元素时才生成，以极大的节省内存

(<元素表达式> for <变量> in <可迭代对象> if <逻辑条件>)


```
>>> agen = (x*x for x in range(10))
>>> agen
<generator object <genexpr> at 0x1078f5620>
>>> for n in agen:
    print (n)
```

```
0
1
4
9
16
25
36
```

p51生成器函数

当推导式比较复杂时，一行表达式无法完成，此时可以定义生成器函数

生成器函数很简单 只是将return 换成了yield

注意：遇到yield时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

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

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

p53 图像处理库-Pillow

可实现缩放、裁剪、旋转、滤镜、文字、调色板等等

图片打开

`image.open(<路径+图像名+文件格式>)`

图片处理

thumbnail

`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

filter

图片模糊

添加文本

```
from PIL import Image, ImageDraw, ImageFont
#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('simsum.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)
#保存图片
img.save('cat1.jpg', 'jpeg')
```

图片保存

- `im.show()`
- `im.save(文件名)`

p54 web框架

基本概念

- 路由
- 模板
- 认证和授权
- session

Flask是一种非常容易上手的Python web开发框架，功能强大，支持很多专业Web开发需要的扩展功能

(现在Django用的比较多)

p55 网络爬虫

通过向网站发起请求获取资源，提取其中有用的信息

request

用于发起请求

- 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Response对象

- .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

可以通过传递header参数来设置请求头

设置代理: 若网站有IP访问次数限制，可以使用proxies参数来替换代理

Beautiful Soup

用于解析HTML

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

```
# 初始化
from bs4 import BeautifulSoup

# 方法一，直接打开文件
soup = BeautifulSoup(open("index.html"))

# 方法二，指定数据
resp = "<html>data</html>"
soup = BeautifulSoup(resp, 'lxml')

# soup 为 BeautifulSoup 类型对象
print(type(soup))
```

```
soup = BeautifulSoup(resp, 'lxml')

# 返回一个标签名为"a"的Tag
soup.find("a")

# 返回所有tag 列表
soup.find_all("a")

## find_all方法可被简写
soup("a")

#找出所有以b开头的标签
for tag in soup.find_all(re.compile("^b")):
    print(tag.name)

#找出列表中的所有标签
soup.find_all(["a", "p"])

# 查找标签名为p, class属性为"title"
soup.find_all("p", "title")

# 查找属性id为"link2"
soup.find_all(id="link2")

# 查找存在属性id的
soup.find_all(id=True)

#
soup.find_all(href=re.compile("elsie"), id='link1')

#
soup.find_all(attrs={"data-foo": "value"})

#查找标签文字包含"sisters"
soup.find(string=re.compile("sisters"))

# 获取指定数量的结果
```

```
soup.find_all("a", limit=2)

# 自定义匹配方法
def has_class_but_no_id(tag):
    return tag.has_attr('class') and not tag.has_attr('id')
soup.find_all(has_class_but_no_id)

# 仅对属性使用自定义匹配方法
def not_lacie(href):
    return href and not re.compile("lacie").search(href)
soup.find_all(href=not_lacie)

# 调用tag的 find_all() 方法时,Beautiful Soup会检索当前tag的所有子孙节点,如果只想搜索tag的直接子节点,可以使用参数 recursive=False

soup.find_all("title", recursive=False)
```

- find_parents() 所有父辈节点
- find_parent() 第一个父辈节点
- find_next_siblings() 之后的所有兄弟节点
- find_next_sibling() 之后的第一个兄弟节点
- find_previous_siblings() 之前的所有兄弟节点
- find_previous_sibling() 之前的第一个兄弟节点
- find_all_next() 之后的所有元素
- find_next() 之后的第一个元素
- find_all_previous() 之前的所有元素
- find_previous() 之前的第一个元素

p56数据可视化

Numpy 库

numpy是Python用于处理大型矩阵的一个速度极快的数学库,很多底层的函数都是用C写的,可以得到在普通Python中无法达到的运行速度

- 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

matplotlib是Python的一个绘图库。它包含了大量的工具,可以使用这些工具创建各种图形(基于numpy)

其实类似matlab绘图，可以用seaborn库美化图表

除此之外，Pyecharts绘制的图片更加精美