

Python

Python

- 1. 计算机基础
 - 1.1 计算机语言
 - 1.2 语言类型
 - 1.3 基本编程概念
 - 1.4 编码问题
- 2. Python 语言基础
 - 2.1 语言特点
 - 2.2 运行程序
 - 2.3 代码格式
 - 2.4 变量
 - 2.5 变量赋值
 - 2.6 补充话题
- 3. 标准数据类型
 - 3.1 数据类型
 - 3.2 操作符
 - 3.3 内建函数
 - 3.3 拷贝问题
 - 3.4 数值类型
- 4. 数据结构
 - 4.1 string 字符串
 - 4.2 string 内建函数
 - 4.3 格式化操作符
 - 4.4 格式化字符串
 - 4.5 list 列表
 - 4.6 tuple 元组
 - 4.7 序列对象操作
 - 4.8 dictionary 字典
 - 4.9 set 集合
- 5. 条件 & 循环
 - 5.1 条件测试
 - 5.2 if 语句
 - 5.3 for 循环
 - 5.4 while 循环
 - 5.5 循环控制
 - 5.6 else 子句
 - 5.7 列表解析
 - 5.8 迭代器
 - 5.9 生成器
 - 5.10 内建函数
- 6. 函数 & 模块
 - 6.1 背景知识
 - 6.2 定义函数
 - 6.3 参数传递
 - 6.4 函数&模块
 - 6.5 补充内容
 - 6.6 函数式编程
 - 6.7 匿名函数

6.8 高阶函数

6.9 偏函数

6.10 返回/回调函数

6.11 递归函数

6.12 装饰器

7. 模块

8. 面向对象编程

8.1 基本概念

8.2 类的概念

8.3 类的操作

8.4 类的继承

8.5 类的编码风格

8.6 类的内建函数

8.7 其他

9. 补充知识

9.1 标识符

9.2 变量类型

9.3 Python对象

9.4 动态类型

9.5 内存管理

9.6 存储数据

10. 文件对象

10.1 文件对象

10.2 标准文件对象

10.3 读取文件

10.4 写入文件

10.5 内建方法

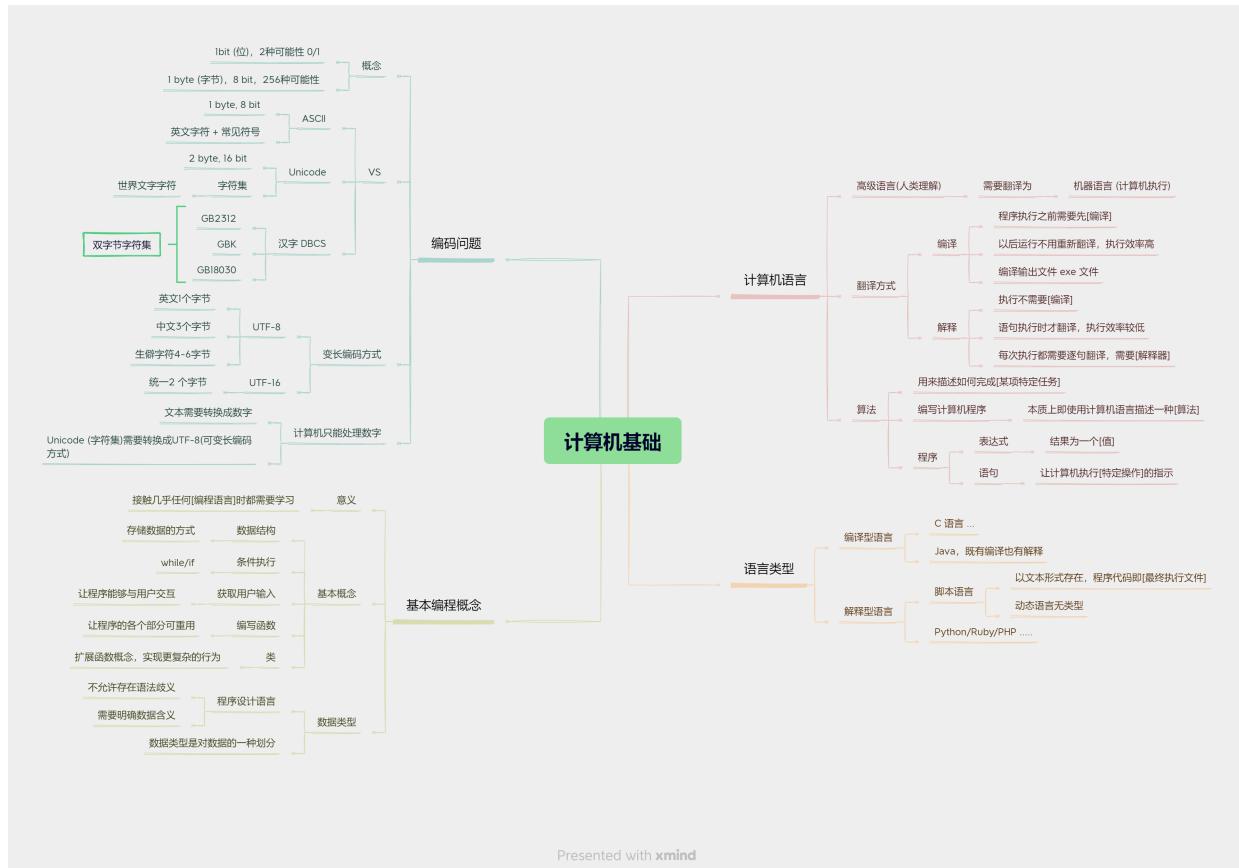
10.6 分隔符

11. 异常处理

12. 测试 & 调试

13. 面试题

1. 计算机基础



1.1 计算机语言

- 高级语言(人类理解) - 需要翻译为 - 机器语言 (计算机执行)
- 翻译方式
 - 编译
 - 程序执行之前需要先[编译]
 - 以后运行不用重新翻译, 执行效率高
 - 编译输出文件 exe 文件
 - 解释
 - 执行不需要[编译]
 - 语句执行时才翻译, 执行效率较低
 - 每次执行都需要逐句翻译, 需要[解释器]
- 算法
 - 用来描述如何完成[某项特定任务]
 - 编写计算机程序, 本质上即使用计算机语言描述一种[算法]
 - 程序由表达式和语句组成, 表达式结果为一个[值], 语句是让计算机执行[特定操作]的指示

1.2 语言类型

- 编译型语言
 - C 语言 ...
 - Java, 既有编译也有解释
- 解释型语言
 - 脚本语言
 - 以文本形式存在, 程序代码即[最终执行文件]
 - 动态语言无类型
 - Python/Ruby/PHP

1.3 基本编程概念

- 意义
 - 接触几乎任何[编程语言]时都需要学习
- 基本概念
 - 数据结构: 存储数据的方式
 - 条件执行: while/if
 - 获取用户输入: 让程序能够与用户交互
 - 编写函数: 让程序的各个部分可重用
 - 类: 扩展函数概念, 实现更复杂的行为
- 数据类型
 - 程序设计语言
 - 不允许存在语法歧义
 - 需要明确数据含义
 - 数据类型是对数据的一种划分

1.4 编码问题

- 概念
 - 1bit (位), 2种可能性 0/1
 - 1 byte (字节), 8 bit, 256种可能性
- VS
 - ASCII
 - 1 byte, 8 bit
 - 英文字符 + 常见符号
 - Unicode
 - 2 byte, 16 bit
 - 字符集: 世界文字字符

- 汉字 DBCS, 双字节字符集

- GB2312
- GBK
- GB18030

- 变长编码方式

- UTF-8
 - 英文1个字节
 - 中文3个字节
 - 生僻字符4-6字节
- UTF-16
 - 统一2个字节

- 计算机只能处理数字

- 文本需要转换成数字
- Unicode (字符集)需要转换成UTF-8(可变长编码方式)

2. Python 语言基础



2.1 语言特点

- 跨平台，可运行于各大操作系统
- 解释型脚本语言
 - 内建高级的数据结构
- 面向对象的语言
 - 便于数据和逻辑相分离
- 动态语言 - 变量本身
 - 类型不固定
 - 可随意转换
 - 不需要声明
- 不用考虑内存问题
- 可处理的数据量无限制
- 默认编码 UTF-8

2.2 运行程序

- 两种模式
 - 脚本式编程：一次性执行源代码脚本
 - 交互式编程：逐行输入再执行显式调用
- 运行脚本
 - 命令行/终端模式：python *.py
 - Linux下可执行脚本
 - 首行添加 #!/usr/local/bin/python; #!/usr/bin/env python (Python 解释器路径)
 - 赋予权限 chmod 755 *.py
 - 运行脚本 *.py
 - IPython
 - %run *.py

2.3 代码格式

- 缩进：语句 [代码块] 用 [缩进深度] 区分，最好为 [4个空格]
- 行长：每行不超过 80 字符
- 空行：将程序的不同部分分开
- \
 - 继续上一行
 - 跨行特例
 - 闭合操作符 [] {} ()
 - 三引号 常用于 [多行注释]

- ; 在同一行连接多个语句，降低可读性，不提倡
- : 分开代码块(组)

2.4 变量

- 内涵：存储了一个值，与[变量]相关联的信息
- 命名规则
 - 只能包含字母、数字和下划线
 - 不能
 - 以数字开头
 - 包含空格
 - 使用 Python 保留字：用于[特殊用途]的单词
 - 建议
 - 简短又具有描述性，驼峰命名法
 - 使用小写字母

2.5 变量赋值

- 值符 =
- 增量赋值 +=
- 多重赋值
 - 同一个引用被赋值给 x, y, z
 - x=y=z=1
- 多元赋值
 - x,y,z = 1,2,'a'，通常元组需要小括号，但可以忽略
 - (x,y,z)= (1,2,'a')，加上小括号以增加可读性
- 变量交换 x,y = y,x

2.6 补充话题

- Python 注释
 - 单行 #
 - 多行 ``````
 - 目的阐述代码要做什么，是如何做的
- Python 编码
 - 类型
 - 3.X 版本默认编码 UTF-8
 - 2.X 版本默认编码 ASCII
 - 指定：# - * - coding: - * -

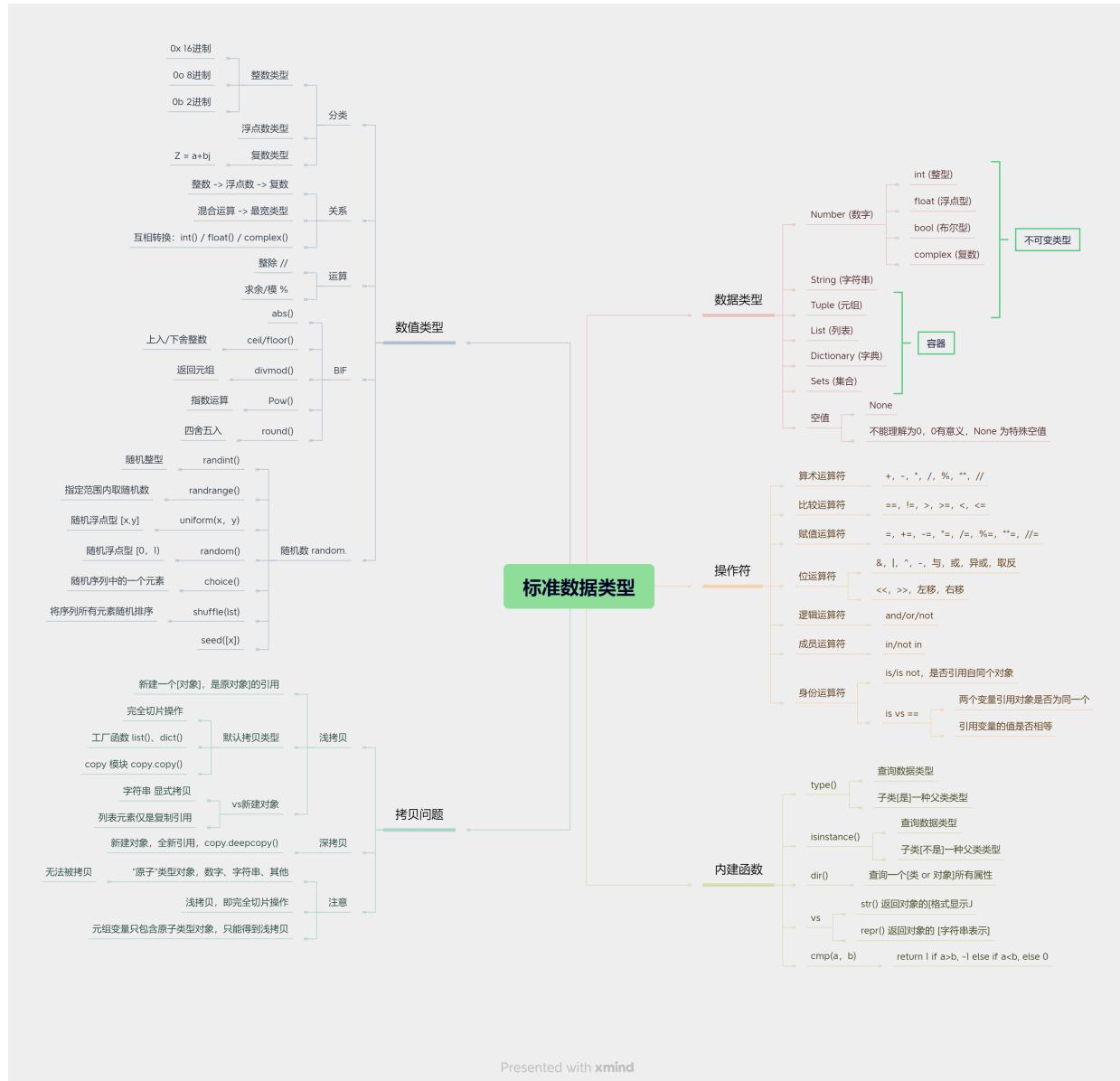
○ 转换

```
u'ABC'.encode('utf-8') #UTF-8编码
```

• IO 编程

- 打印到屏幕 print()
- 获取用户输入
 - 获取字符输入 input()
 - 获取数值输入 int(input())

3. 标准数据类型



3.1 数据类型

- Number (数字)
 - int (整型)
 - float (浮点型)
 - bool (布尔型)
 - complex (复数)
- String (字符串)
- Tuple (元组)
- List (列表)
- Dictionary (字典)
- Sets (集合)
- 空值
 - None
 - 不能理解为0, 0有意义, None 为特殊空值

3.2 操作符

- 算术运算符: +, -, *, /, %, **, //
- 比较运算符: ==, !=, >, >=, <, <=
- 赋值运算符: =, +=, -=, *=, /=, %=, **=, //=
- 位运算符:
 - &, |, ^, -, 与, 或, 异或, 取反
 - <<, >>, 左移, 右移
- 逻辑运算符: and/or/not
- 成员运算符: in/not in
- 身份运算符:
 - is/is not, 是否引用自同个对象
 - is vs ==
 - 两个变量引用对象是否为同一个
 - 引用变量的值是否相等

3.3 内建函数

- type()
 - 查询数据类型
 - 子类[是]一种父类类型
- isinstance()
 - 查询数据类型

- 子类[不是]一种父类类型
- dir(): 查询一个[类 or 对象]所有属性
- vs
 - str() 返回对象的[格式显示]
 - repr() 返回对象的 [字符串表示]
- cmp(a, b): return 1 if a>b, -1 else if a<b, else 0

3.3 拷贝问题

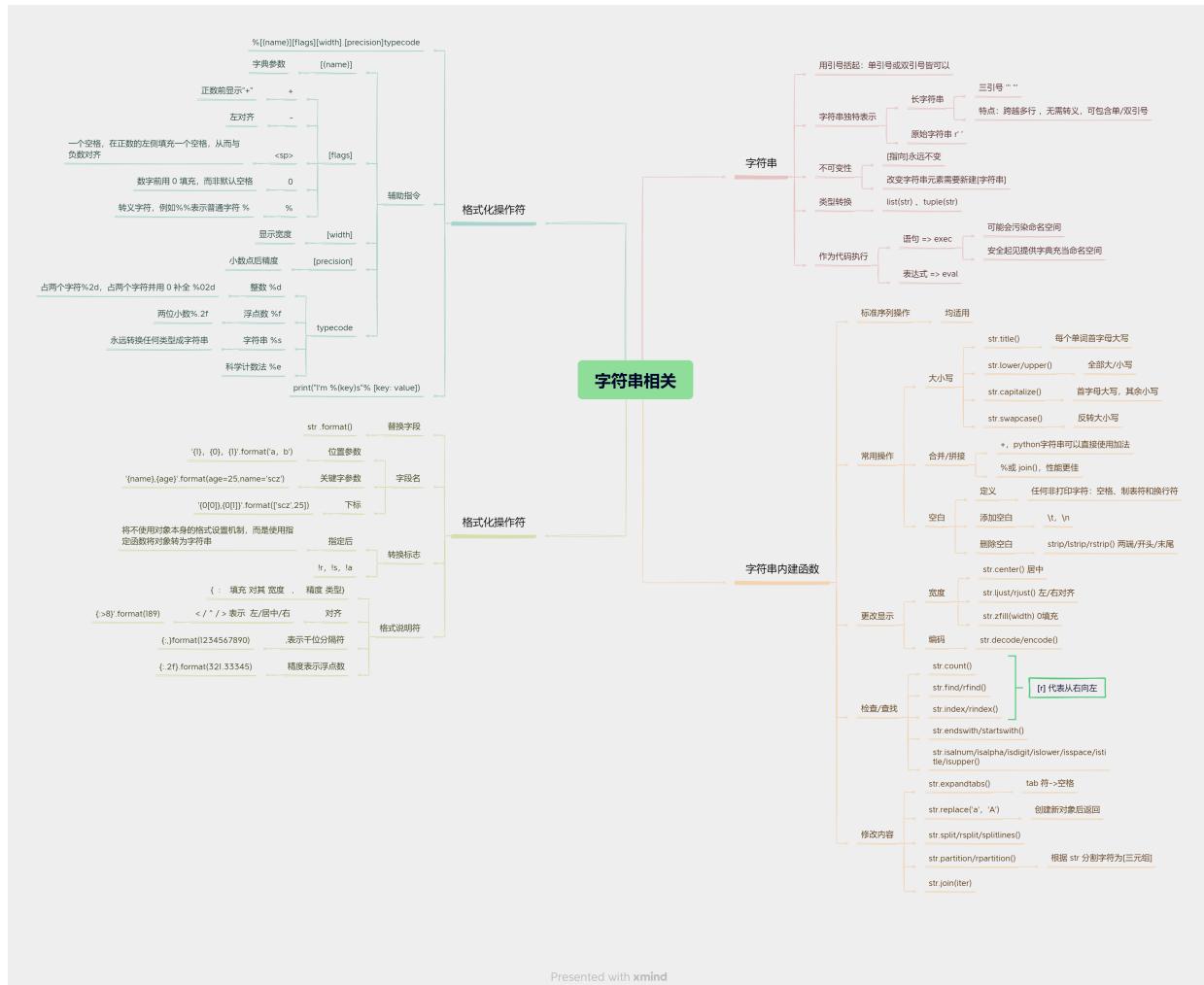
- 浅拷贝
 - 新建一个[对象], 是原对象]的引用
 - 默认拷贝类型
 - 完全切片操作
 - 工厂函数 list()、dict()
 - copy 模块 copy.copy()
 - vs新建对象
 - 字符串 显式拷贝
 - 列表元素仅是复制引用
- 深拷贝
 - 新建对象, 全新引用, copy.deepcopy()
- 注意
 - “原子”类型对象, 数字、字符串、其他, 无法被拷贝
 - 浅拷贝, 即完全切片操作
 - 元组变量只包含原子类型对象, 只能得到浅拷贝

3.4 数值类型

- 分类
 - 整数类型
 - 0x 16进制
 - 0o 8进制
 - 0b 2进制
 - 浮点数类型
 - 复数类型: $Z = a+bj$
- 关系
 - 整数 -> 浮点数 -> 复数
 - 混合运算 -> 最宽类型
 - 互相转换: int() / float() / complex()

- 运算
 - 整除 //
 - 求余/模 %
- BIF
 - abs()
 - ceil/floor() 上入/下舍整数
 - divmod() 返回元组
 - Pow() 指数运算
 - round() 四舍五入
- 随机数 random.
 - randint() 随机整型
 - randrange() 指定范围内取随机数
 - uniform(x, y) 随机浮点型 [x,y]
 - random() 随机浮点型 [0, 1)
 - choice() 随机序列中的一个元素
 - shuffle(lst) 将序列所有元素随机排序
 - seed([x])

4. 数据结构



Presented with xmind

4.1 string 字符串

- 用引号括起: 单引号或双引号皆可以
- 字符串独特表示
 - 长字符串
 - 三引号 "" ""
 - 特点: 跨越多行, 无需转义, 可包含单/双引号
 - 原始字符串 r''
- 不可变性
 - [指向]永远不变
 - 改变字符串元素需要新建[字符串]
- 类型转换: list(str)、tuple(str)
- 作为代码执行
 - 语句 => exec, 可能会污染命名空间, 安全起见提供字典充当命名空间

```
scope={}
exec('sqrt = 1', scope)
scope['sqrt']
```

- 表达式 => eval

4.2 string 内建函数

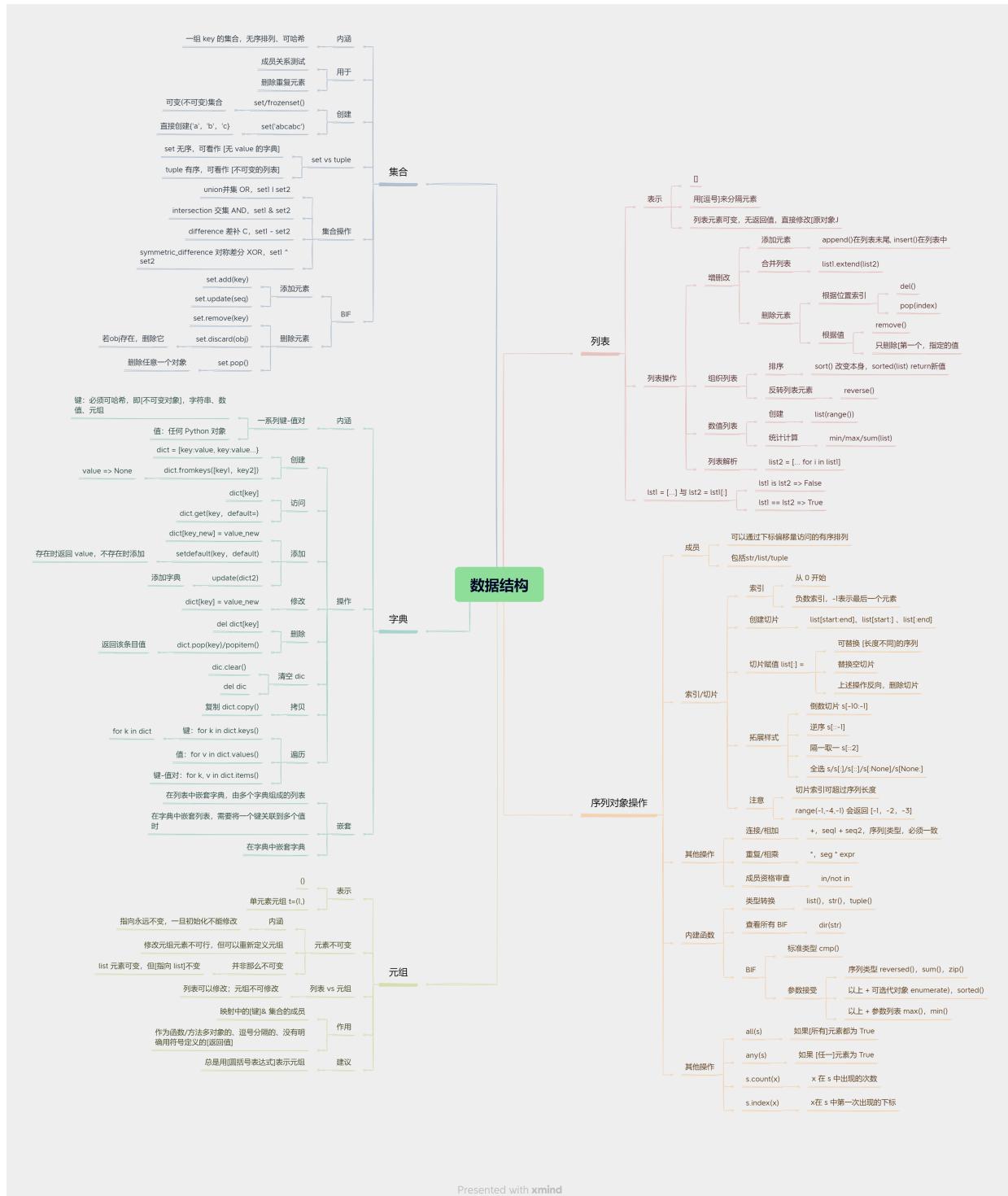
- 标准序列操作：均适用
- 常用操作
 - 大小写
 - str.title() 每个单词首字母大写
 - str.lower/upper() 全部大/小写
 - str.capitalize() 首字母大写，其余小写
 - str.swapcase() 反转大小写
 - 合并/拼接
 - +, python字符串可以直接使用加法
 - %或 join(), 性能更佳
 - 空白
 - 定义：任何非打印字符：空格、制表符和换行符
 - 添加空白 \t, \n
 - 删除空白 strip/lstrip/rstrip() 两端/开头/末尾
- 更改显示
 - 宽度
 - str.center() 居中
 - str.ljust/rjust() 左/右对齐
 - str.zfill(width) 0填充
 - 编码：str.decode/encode()
- 检查/查找
 - str.count()
 - str.find/rfind()
 - str.index/rindex()
 - str.endswith/startswith()
 - str.isalnum/isalpha/isdigit/islower/isspace/istitle/isupper()
- 修改内容
 - str.expandtabs() tab 符->空格
 - str.replace('a', 'A') 创建新对象后返回
 - str.split/rsplit/splitlines()
 - str.partition/rpartition() 根据 str 分割字符串为[三元组]
 - str.join(iter)

4.3 格式化操作符

- 替换字段 `str.format()`
 - 位置参数 `'{1}, {0}, {1}'.format('a', b)'`
 - 关键字参数 `'{name},{age}'.format(age=25,name='scz')`
 - 下标 `'{0[0]}, {0[1]}'.format(['scz'],25)`
- 转换标志
 - 指定后，将不使用对象本身的格式设置机制，而是使用指定函数将对象转为字符串
 - `!r, !s, !a`
- 格式说明符
 - `{: 填充 对其 宽度 , 精度 类型}`
 - 对齐 `< / ^ / >` 表示 `左/居中/右`, `'{:>8}'.format(189)`
 - `,` 表示千位分隔符, `'{:,}format(1234567890)`
 - 精度表示浮点数, `'{:.2f}'.format(321.33345)`

4.4 格式化字符串

- `%[(name)][flags][width].[precision]typecode`
- 辅助指令
 - `[(name)]`: 字典参数
 - `[flags]`
 - `+`: 正数前显示“+”
 - `-`: 左对齐
 - `<sp>`: 一个空格，在正数的左侧填充一个空格，从而与负数对齐
 - `0`: 数字前用 0 填充，而非默认空格
 - `%`: 转义字符，例如`%%`表示普通字符 %
 - `[width]`: 显示宽度
 - `[precision]`: 小数点后精度
 - `typecode`
 - 整数 `%d`, 占两个字符`%2d`, 占两个字符并用 0 补全 `%02d`
 - 浮点数 `%f`, 两位小数`.2f`
 - 字符串 `%s`, 永远转换任何类型成字符串
 - 科学计数法 `%e`
- `print("I'm %(key)s"% [key: value])`



4.5 list 列表

- 表示 []

- 用[逗号]来分隔元素
- 列表元素可变, 无返回值, 直接修改[原对象]

- 列表操作

- 增删改

- 添加元素: append()在列表末尾, insert()在列表中
 - 合并列表: list1.extend(list2)

- 删除元素：
 - 根据位置索引 `del()`, `pop(index)`
 - 根据值 `remove()`, 只删除[第一个, 指定的值]
- 组织列表
 - 排序: `sort()` 改变本身, `sorted(list)` return新值
 - 反转列表元素: `reverse()`
- 数值列表
 - 创建: `list(range())`
 - 统计计算: `min/max/sum(list)`
- 列表解析 `list2 = [... for i in list1]`
- `lst1 = [...]` 与 `lst2 = lst1[:]`
 - **lst1 is lst2 => False**
 - **lst1 == lst2 => True**

4.6 tuple 元组

- 表示 (): 单元素元组 `t=(1,)`
- 元素不可变
 - 内涵: 指向永远不变, 一旦初始化不能修改
 - 修改元组元素不可行, 但可以重新定义元组
 - 并非那么不可变: list 元素可变, 但[指向 list]不变
- 列表 vs 元组: 列表可以修改; 元组不可修改
- 作用
 - 映射中的[键]& 集合的成员
 - 作为函数/方法多对象的、逗号分隔的、没有明确用符号定义的[返回值]
- 建议: 总是用[圆括号表达式]表示元组

4.7 序列对象操作

- 成员: 可以通过下标偏移量访问的有序排列, 包括str/list/tuple
- 索引/切片
 - 索引
 - 从 0 开始
 - 负数索引, -1表示最后一个元素
 - 创建切片: `list[start:end]`、`list[start:]`、`list[:end]`
 - 切片赋值 `list[:] =`
 - 可替换 [长度不同]的序列
 - 替换空切片

- 上述操作反向，删除切片
- 拓展样式
 - 倒数切片 `s[-10:-1]`
 - 逆序 `s[::-1]`
 - 隔一取一 `s[::2]`
 - 全选 `s/s[:]//s[:]/s[:None]/s[None:]`
- 注意
 - 切片索引可超过序列长度
 - `range(-1,-4,-1)` 会返回 `[-1, -2, -3]`
- 其他操作
 - 连接/相加: `+`, `seq1 + seq2`, 序列[类型, 必须一致]
 - 重复/相乘: `*`, `seg * expr`
 - 成员资格审查: `in/not in`
- 内建函数
 - 类型转换: `list()`, `str()`, `tuple()`
 - 查看所有 BIF: `dir(str)`
 - BIF
 - 标准类型 `cmp()`
 - 参数接受
 - 序列类型 `reversed()`, `sum()`, `zip()`
 - 以上 + 可迭代对象 `enumerate()`, `sorted()`
 - 以上 + 参数列表 `max()`, `min()`
 - 其他操作
 - `all(s)`, 如果[所有]元素都为 `True`
 - `any(s)`, 如果 [任一]元素为 `True`
 - `s.count(x)`, `x` 在 `s` 中出现的次数
 - `s.index(x)`, `x` 在 `s` 中第一次出现的下标

4.8 dictionary 字典

- 内涵: 一系列键-值对
 - 键: 必须可哈希, 即[不可变对象], 字符串、数值、元组
 - 值: 任何 Python 对象
- 操作
 - 创建
 - `dict = [key:value, key:value...]`
 - `dict.fromkeys([key1, key2]), value => None`

- 访问
 - `dict[key]`
 - `dict.get(key, default=)`
- 添加
 - `dict[key_new] = value_new`
 - `setdefault(key, default)`, 存在时返回 `value`, 不存在时添加
 - `update(dict2)`, 添加字典
- 修改: `dict[key] = value_new`
- 删除
 - `del dict[key]`
 - `dict.pop(key)/popitem()`, 返回该条目值
- 清空 dic
 - `dic.clear()`
 - `del dic`
- 拷贝: 浅复制 `dict.copy()`
- 遍历
 - 键: `for k in dict.keys()`
 - 值: `for v in dict.values()`
 - 键-值对: `for k, v in dict.items()`
- 嵌套
 - 在列表中嵌套字典, 由多个字典组成的列表
 - 在字典中嵌套列表, 需要将一个键关联到多个值时
 - 在字典中嵌套字典

4.9 set 集合

- 内涵: 一组 key 的集合, 无序排列、可哈希
- 用于
 - 成员关系测试
 - 删除重复元素
- 创建
 - `set/frozenset()` 可变(不可变)集合
 - `set('abcabc')` 直接创建{'a', 'b', 'c'}
- set vs tuple
 - set 无序, 可看作 [无 value 的字典]
 - tuple 有序, 可看作 [不可变的列表]
- 集合操作

- union 并集 OR, set1 | set2
- intersection 交集 AND, set1 & set2
- difference 差补 C, set1 - set2
- symmetric_difference 对称差分 XOR, set1 ^ set2

- BIF

- 添加元素
 - set.add(key)
 - set.update(seq)
- 删除元素
 - set.remove(key)
 - set.discard(obj) 若obj存在，删除它
 - set.pop() 删除任意一个对象

5. 条件 & 循环



Presented with xmind

5.1 条件测试

- 核心: 条件表达式 True/False
- 检查
 - 是否相等/不相等: 若需要忽略大小写, 先转为小写
 - 比较数字: ==, !=, >, >=, <, <=
 - 多个条件: and/or
 - 特定值是否包含在列表中: in

- 布尔表达式
 - 条件测试的[别名]
 - 常用于记录条件
- True: 非零数值、非空字符串、非空 list...

5.2 if 语句

- 简单 if 语句
 - if conditional_test: do something
 - 单一语句代码块，可放在同一行
- if-else 语句
- if-elif-else 结构：可使用厂任意数量 elif 代码块
- 三元操作符
 - X if C else Y
 - smaller = x if x < y else y

5.3 for 循环

- 迭代循环
 - 遍历[序列]成员
 - 依次访问可迭代对象
- 用于
 - 序列类型
 - 字符串、列表、元组
 - 迭代方法：序列项，序列索引，项&索引 enumerate()
 - 迭代器类型
 - 自动调用：迭代器对象 next()方法，调用后返回 [下一个条目]
 - 直到捕获：StopIteration 异常
 - 结束循环

5.4 while 循环

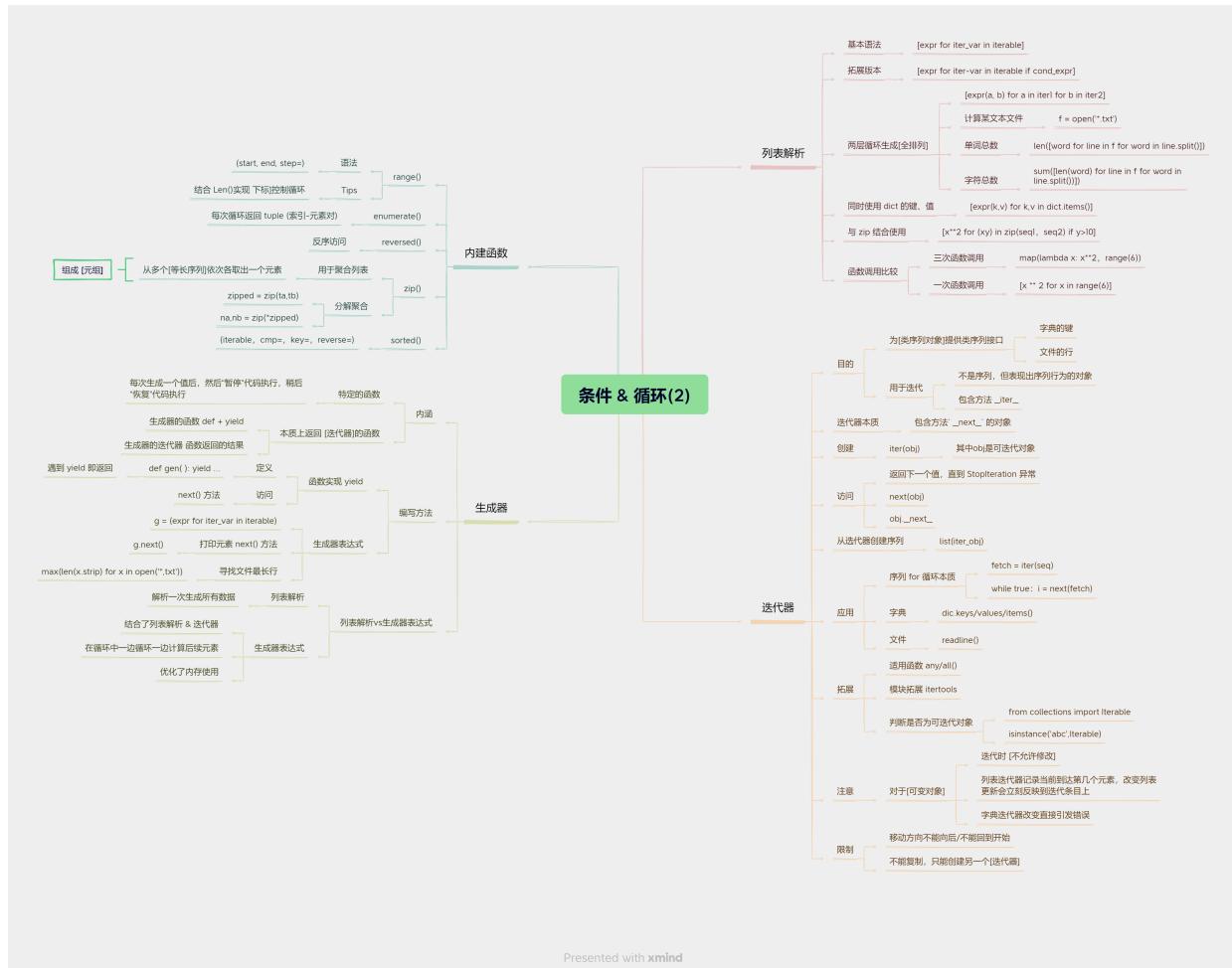
- 不断运行：直到[指定的条件]不满足
- 使用[标志]
 - 用于要求很多条件都满足才继续运行的程序中
 - 内涵
 - 定义一个[变量]作为[标志]
 - 用于判断整个程序是否处于[活动状态]
 - True 继续运行； False 停止运行

5.5 循环控制

- break: 停止执行, 退出循环
- continue
 - 跳过本次执行, 返回到[循环开头]
 - 根据条件测试, 是否继续执行循环
 - 验证[条件表达式]是否成立, 验证[是否还有[可迭代元素]]
- pass
 - 可用在
 - if-elif-else
 - while/for/def/class
 - try-except-finally
 - 用于
 - 保持结构完整性
 - 开发和调试时先确定结构; 或暂时先不处理 异常处理中

5.6 else 子句

- 放在循环末尾
- 循环后处理
 - while...else...
 - for...else...
- 只在循环[正常完成后]执行
- break会跳过 [else 子句]



5.7 列表解析

- 基本语法: `[expr for iter_var in iterable]`
- 拓展版本: `[expr for iter-var in iterable if cond_expr]`
- 两层循环生成[全排列]
 - `[expr(a, b) for a in iter1 for b in iter2]`
 - 计算某文本文文件: `f = open('*.txt')`
 - 单词总数: `len([word for line in f for word in line.split()])`
 - 字符总数: `sum([len(word) for line in f for word in line.split()])`
- 同时使用 dict 的键、值: `[expr(k,v) for k,v in dict.items()]`
- 与 zip 结合使用: `[x**2 for (xy) in zip(seq1, seq2) if y>10]`
- VS
 - 三次函数调用: `map(lambda x: x**2, range(6))`
 - 一次函数调用: `[x ** 2 for x in range(6)]`

5.8 迭代器

- 目的
 - 为[类序列对象]提供类序列接口：字典的键、文件的行
 - 用于迭代
 - 不是序列，但表现出序列行为的对象
 - 包含方法`__iter__`
- 迭代器本质：包含方法`__next__`的对象
- 创建：`iter(obj)`，其中obj是可迭代对象
- 访问：返回下一个值，直到`StopIteration`异常
 - `next(obj)`
 - `obj.__next__`
- 从迭代器创建序列：`list(iter_obj)`
- 应用
 - 序列 for 循环本质
 - `fetch = iter(seq)`
 - `while true: i = next(fetch)`
 - 字典：`dic.keys/values/items()`
 - 文件：`readline()`
- 拓展
 - 适用函数`any/all()`
 - 模块拓展`itertools`
 - 判断是否为可迭代对象`from collections import Iterable, isinstance('abc', Iterable)`
- 注意
 - 对于[可变对象]
 - 迭代时[不允许修改]
 - 列表迭代器记录当前到达第几个元素，改变列表更新会立刻反映到迭代条目上
 - 字典迭代器改变直接引发错误
 - 限制
 - 移动方向不能向后/不能回到开始
 - 不能复制，只能创建另一个[迭代器]

5.9 生成器

- 内涵
 - 特定的函数
 - 每次生成一个值后，然后“暂停”代码执行，稍后“恢复”代码执行

- 本质上返回 [迭代器] 的函数
 - 生成器的函数 def + yield
 - 生成器的迭代器 函数返回的结果
- 编写方法
 - 函数实现 yield
 - 定义: `def gen(): yield ...` 遇到 yield 即返回
 - 访问: next() 方法
 - 生成器表达式
 - `g = (expr for iter_var in iterable)`
 - 打印元素 next() 方法: `g.next()`
 - 寻找文件最长行 `max(len(x.strip) for x in open('*', 'txt'))`
- 列表解析 vs 生成器表达式
 - 列表解析一次生成所有数据
 - 生成器表达式
 - 结合了列表解析 & 迭代器
 - 在循环中一边循环一边计算后续元素
 - 优化了内存使用

5.10 内建函数

- range()
 - 语法 (start, end, step=)
 - Tips: 结合 Len() 实现下标控制循环
- enumerate() 每次循环返回 tuple (索引-元素对)
- reversed() 反序访问
- zip()
 - 用于聚合列表, 从多个[等长序列]依次各取出一个元素
 - 分解聚合
 - `zipped = zip(ta, tb)`
 - `na, nb = zip(*zipped)`

```
zip([1,2,3], [4,5,6])
>>> [(1, 4), (2, 5), (3, 6)]
zip(*zipped)
>>> ((1, 2, 3), (4, 5, 6))
```

- sorted() (`iterable, cmp=, key=, reverse=`)

6. 函数 & 模块

6.1 背景知识

- 函数
 - 对[程序逻辑]进行结构化 or 过程化
 - 可作为[参数], 在函数体内被调用
- 作用域/命名空间
 - 用于[变量存储]
 - 全局变量
 - 函数内可以引用, 不能修改
 - 重新关联 [全局变量]需要声明 global
 - 局部变量
 - 在函数内定义, 只允许函数内访问
 - 可能隐藏/覆盖全局变量
 - 若被遮盖仍需访问, 需要globals()['var']
 - 变量搜索顺序
 - 局部作用域 -> 全局域
 - 函数中的列表
 - 在函数中修改列表, 永久性修改原列表
 - 所以传递[列表的副本], 禁止函数修改原列表
- 代码重构
 - 将代码划分为一系列完成具体工作的函数
 - 每个函数执行单一而清晰的任务

6.2 定义函数

- 形式

```
def funName(para):
    "docstring"
    ...
    return ...
```

- para: 可选, 用来向函数传递信息
- docstring
 - 文档字符串
 - 访问通过 help(funName) 或者 help(funName)

- return
 - 随时返回 函数结果
 - 无[返回值], 不用 return或者仅有 return
 - 返回 [值/对象]: 若返回多个对象, 以[元组]返回
 - 返回值可以是任何类型
 - 执行到 return 时, 停止执行函数内余下语句
- 形参 vs 实参
 - 形参: 在 def 语句中, 位于函数名后面的变量
 - 实参: 调用函数时传递给函数的信息

6.3 参数传递

- 完整语法
 - func(positional_args, keyword_args, *tuple_nonkw_args, **dict_kw_args)
- 值传递
 - 参数: 基本数据类型
 - 内涵: 变量传递给函数后, 函数在内存中复制一个新变量, 参数指向新的引用对象
- 指针传递
 - 参数: 引用数据类型
 - 内涵
 - 变量传递给函数的是指针, 指针指向序列在内存中的位置
 - 在函数中对 List 的操作在原有内存中进行, 从而影响原有变量
- 固定数量参数
 - 位置参数: 要求[实参]的顺序, 必须与[形参]相同
 - 关键字参数
 - 传递给函数的[名称-值]对
 - 无需考虑[实参]顺序
 - 务必准确指定函数定义中的[形参名]
 - 默认值
 - 给[形参]指定默认值后, 调用中可省略相应[实参]
 - [形参]列表顺序, 先列出无默认值的[形参]
- 任意数量参数
 - 任意数量位置参数
 - 元组 *para_tuple
 - a=[1,2,3], fun_name(*a)
 - 任意数量关键字参数
 - 字典 **para_dic

- dir = {a:1...}, func_name(**dir)

6.4 函数&模块

- 将函数存储于[模块]
- 模块
 - 每一个脚本文件均为[模块], 以磁盘文件形式存在
 - 若[模块]过大, 考虑拆解代码, 另建模块
- 导入:
 - 整个模块: import module_name
 - 特定函数: from module_name import function_name
 - 指定别名:
 - from module_name import function_name as fn
 - import module_name as mn
 - 模块中所有函数(不推荐): from module_name import *
- 重新载入模块: importlib.reload()

6.5 补充内容

- 函数的引用vs调用
 - 函数的引用
 - 访问函数
 - 别名变量: 函数名本质上即[变量], 多个别名为对同一[函数对象]的引用
 - 函数的调用
- 例子
 - 新建函数对象 def foo():..., 赋值给 foo
 - 引用: bar = foo, 引用同一函数对象
 - 调用: foo(), 调用函数
- 判断是否可调用: 是否是函数或方法, callable(object)
- 调用函数: 提供要传递给函数的参数, apply(func[, args[, kwargs]])
- 函数编写指南
 - 使用[描述性名称]
 - [名称格式]只包含小写字母 &下划线
 - 包含注释: 紧跟在函数定义后面, 阐述其功能
 - 等号两边不要有空格
 - 给形参指定[默认值]时, 函数调用中的关键字实参
 - 两个空行分隔相邻函数

6.6 函数式编程

- 释义
 - 一种[编程范式]，如何编写程序的方法论：面向对象/面向过程
 - 属于[结构化编程]：子程序 / 程式码区块 / ...
- 思想
 - 把运算过程尽量写成一系列嵌套的函数调用
 - 允许把函数作为参数传入另一个函数，返回一个函数
- 对于Python
 - Python并非函数式编程语言，但支持函数式编程语言构建
 - 匿名函数、BIF、偏函数
 - 本质上通过[封装对象]实现函数式编程

6.7 匿名函数

- 关键字：lambda，不需要以标准的方式来声明
- 表达式：lambda [arg1[, ...argN]]: expression
- 使用
 - 通过变量调用
 - `func = lambda x,y: x + y`
 - lambda 生成一个函数对象，参数为 x, y，返回值为 x+y，函数对象赋给 func
 - 作为返回值返回
 - `def build(x, y): return lambda: x + y`

6.8 高阶函数

- 一个函数接收另一个函数作为参数
- filter()
 - (func, seq)：将函数对象依次作用于每一个元素
 - 返回 True/False，保留/丢弃该元素
 - `filter(lambda n: n%2, allNums)`
- map()
 - (func, seq1[,...])：将函数对象依次作用于每一个元素
 - 返回函数计算过的新元素
 - `map(lambda x: x**2, range(3))`
- reduce()
 - `from functools import reduce`
 - (func, seq[, init])
 - 参数

- func: 二元函数, 接收两个参数: 上一次运算结果, 序列的下一个元素
- seq[, init]: 初始化器, 第一轮计算第一个序列元素
- reduce(lambda x, y: x+y, range(4))

6.9 偏函数

- 概述
 - 偏函数应用: Partial function application
 - 结合了函数式编程、默认参数、可变参数
 - 作用
 - 固定函数的某些参数, 即[设置默认值]
 - 返回一个新的函数以供调用
 - 使用场景
 - 函数的参数太多需要简化
 - 创建一个新函数, 固定住原函数的部分参数
- 应用
 - 模块: functools
 - 创建: from functools import partial
 - partial(func, *args, **keywords)
 - 例子
 - 固定[位置参数]: add1 = partial(add, 1), add1(x) 相当于 add(1, x)

6.10 返回/回调函数

- 函数作为[结果值]返回
 - 每次调用都会返回一个新的函数, 即使传入相同的参数, 调用结果互不影响
 - 可用于使用一个函数创建另一个函数
- 闭包
 - 内部函数: 引用[外部函数] 的参数和局部变量
 - 外部函数: 调用时返回已保存参数和变量的[内部函数]
 - 返回 [闭包] 时
 - 返回函数不要引用任何[循环变量] 以及 后续会[发生变化的变量]
 - 如果一定要引用: 再创建一个函数, 用该函数的参数绑定循环变量当前的值
 - 给 [外部作用域] 变量赋值: 关键字 nonlocal

6.11 递归函数

- 定义：函数在内部调用自身
- 包含
 - 基线条件：满足时直接返回 [一个值]
 - 递归条件：包含 [一个或多个调用]，旨在解决 [问题的一部分]
 - 问题终将分解为基线条件可解决的 [最小问题]
- 防止[栈溢出]
 - 原因：
 - 每进入一个函数调用栈就会加一层栈帧
 - 每当函数返回栈就会减一层栈帧
 - 递归调用的次数过多导致栈溢出
 - 解决：尾递归优化，函数返回时调用自身本身，return 不能包含表达式
 - 缺陷
 - 若没有针对尾递归继续做优化
 - 任何递归函数都存在[栈溢出]的问题

6.12 装饰器

- Decorator
 - 在代码运行期间给函数动态增加功能
 - 本质上：一个[返回函数]的[高阶函数]，仅添加了功能，不改变函数原貌
 - 形式：@decorator(dec_opt_args) def func2Bdecorated(...)
- 多个装饰器“堆叠”

```
@deco2(deco_args)
@deco1
def func()
```

- 相当于 `func = deco2(deco_args)(deco1(func))`
- 例子
 - 定义装饰器

```
def log(func):
    def wrapper(*args, **kw):
        print('call %s():' % func.__name__)
        return func(*args, **kw)
    return wrapper
```

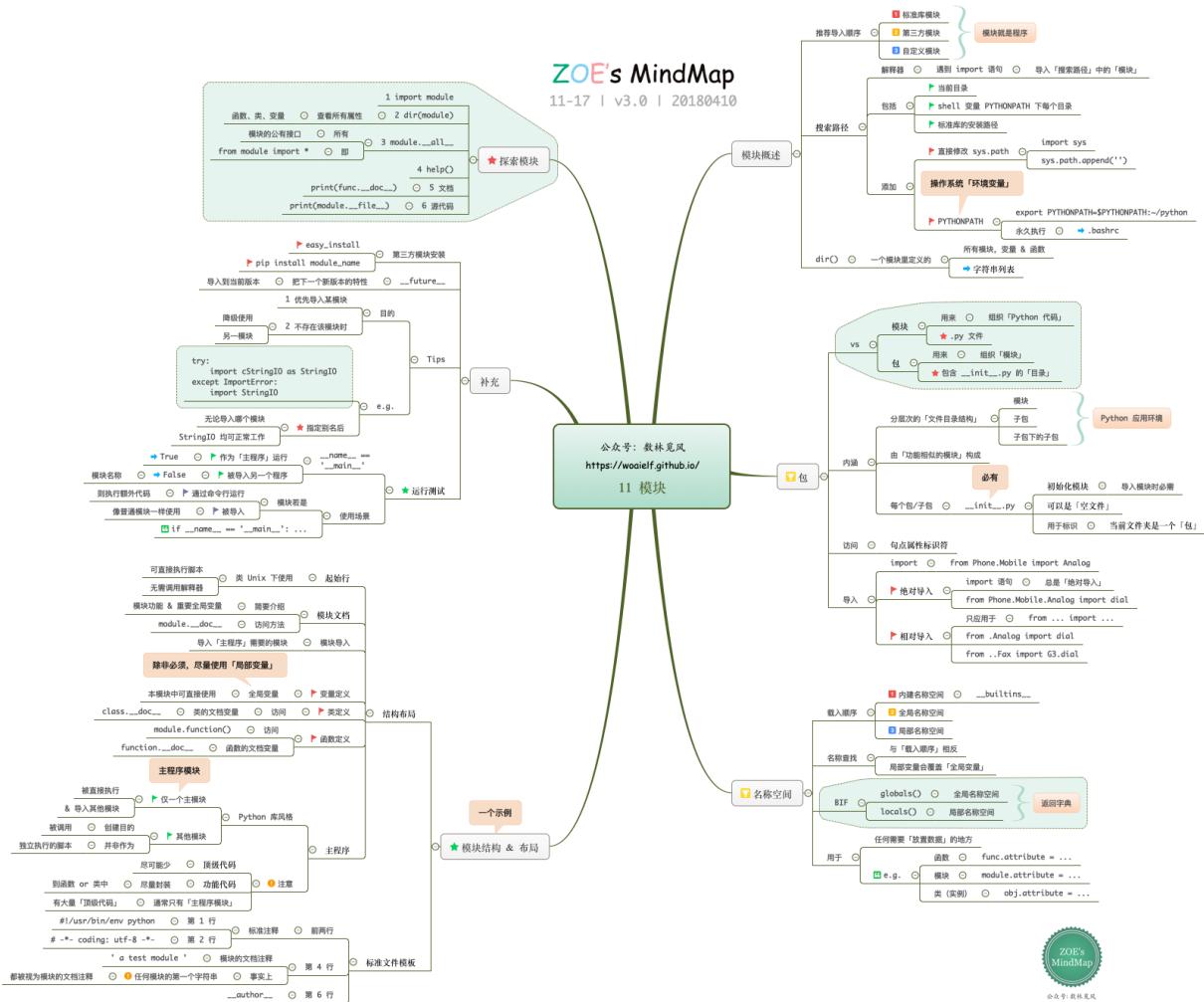
- log 是装饰器函数
- *args, **kw 表示可以接受任意参数的调用

○ 使用装饰器

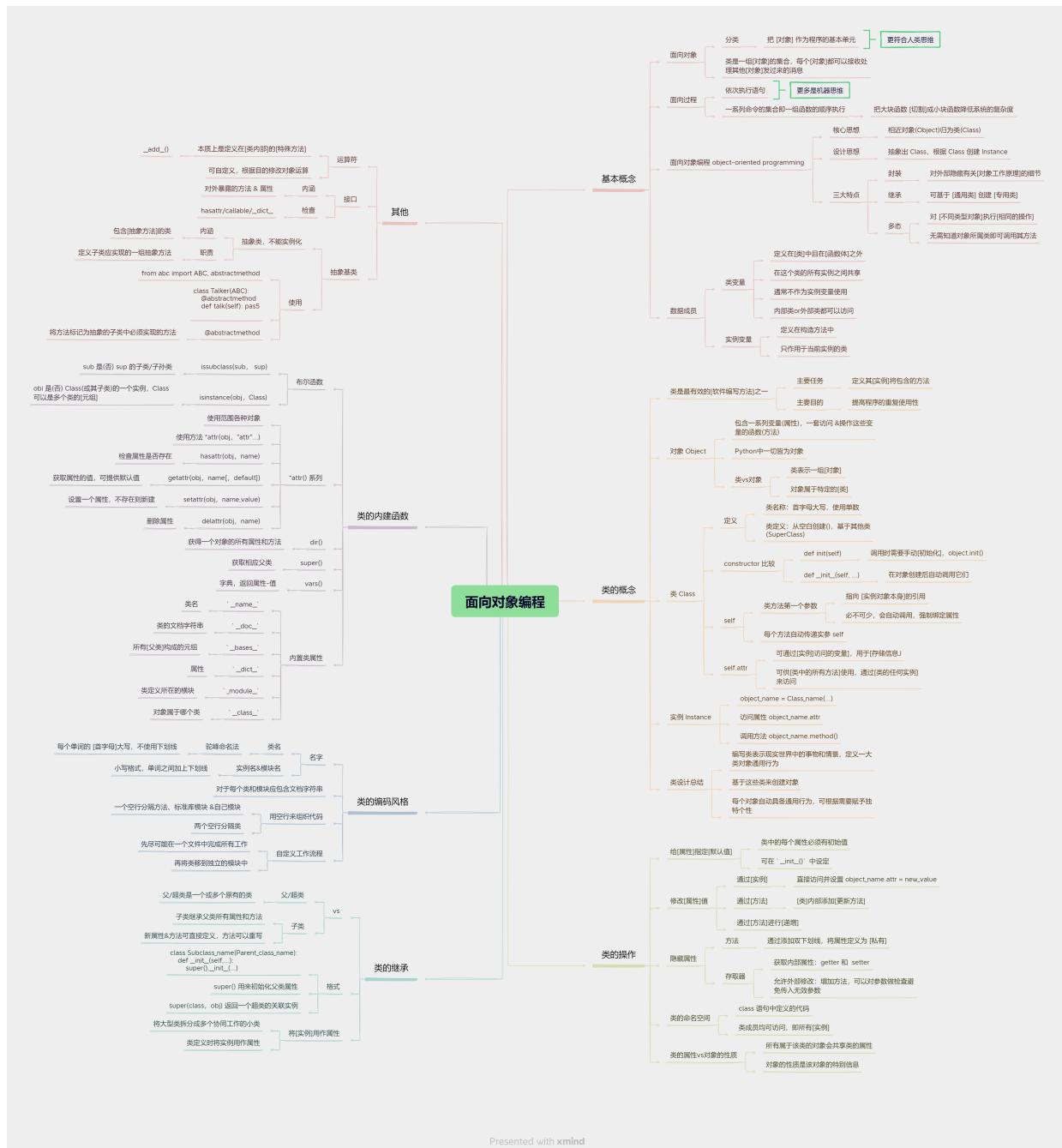
```
@log
def now():
    print('2013-12-25')
```

- @log 放到 now() 函数的定义处
- 相当于执行了语句 now = log(now)

7. 模块



8. 面向对象编程



8.1 基本概念

- 面向对象
 - 分类把[对象]作为程序的基本单元, 更符合人类思维
 - 类是一组[对象]的集合, 每个[对象]都可以接收处理其他[对象]发过来的消息
- 面向过程
 - 依次执行语句, 更多是机器思维
 - 一系列命令的集合即一组函数的顺序执行
 - 把大块函数[切割]成小块函数降低系统的复杂度
- 面向对象编程 object-oriented programming
 - 核心思想: 相近对象(Object)归为类(Class)

- 设计思想：抽象出 Class，根据 Class 创建 Instance
- 三大特点
 - 封装：对外部隐藏有关[对象工作原理]的细节
 - 继承：可基于 [通用类] 创建 [专用类]
 - 多态：对 [不同类型对象] 执行 [相同的操作]，无需知道对象所属类即可调用其方法
- 数据成员
 - 类变量
 - 定义在[类]中且在[函数体]之外
 - 在这个类的所有实例之间共享
 - 通常不作为实例变量使用
 - 内部类or外部类都可以访问
 - 实例变量
 - 定义在构造方法中
 - 只作用于当前实例的类

8.2 类的概念

- 类是最有效的[软件编写方法]之一
 - 主要任务：定义其[实例]将包含的方法
 - 主要目的：提高程序的重复使用性
- 对象 Object
 - 包含一系列变量(属性)，一套访问 & 操作这些变量的函数(方法)
 - Python 中一切皆为对象
 - 类 vs 对象
 - 类表示一组[对象]
 - 对象属于特定的[类]
- 类 Class
 - 定义
 - 类名称：首字母大写，使用单数
 - 类定义：从空白创建()，基于其他类(SuperClass)
 - constructor 比较
 - `def __init__(self)`：调用时需要手动[初始化]，`object.init()`
 - `def __init__(self, ...)`：在对象创建后自动调用它们
 - self
 - 类方法第一个参数
 1. 指向 [实例对象本身] 的引用
 2. 必不可少，会自动调用，强制绑定属性

- 每个方法自动传递实参 self
- self.attr
 - 可通过[实例]访问的变量，用于[存储信息]
 - 可供[类中的所有方法]使用，通过[类的任何实例]来访问
- 实例 Instance
 - object_name = Class_name(...)
 - 访问属性 object_name.attr
 - 调用方法 object_name.method()
- 类设计总结
 - 编写类表示现实世界中的事物和情景，定义一大类对象通用行为
 - 基于这些类来创建对象
 - 每个对象自动具备通用行为，可根据需要赋予独特个性

8.3 类的操作

- 给[属性]指定[默认值]
 - 类中的每个属性必须有初始值
 - 可在 `__init__()` 中设定
- 修改[属性]值
 - 通过[实例]：直接访问并设置 `object_name.attr = new_value`
 - 通过[方法]：[类]内部添加[更新方法]
 - 通过[方法]进行[递增]
- 隐藏属性
 - 方法：将属性定义为 [私有]，通过添加双下划线
 - 存取器
 - 获取内部属性：getter 和 setter
 - 允许外部修改：增加方法，可以对参数做检查避免传入无效参数
- 类的命名空间
 - class 语句中定义的代码
 - 类成员均可访问，即所有[实例]
- 类的属性vs对象的性质
 - 所有属于该类的对象会共享类的属性
 - 对象的性质是该对象的特别信息

8.4 类的继承

- 父/超类 vs 子类
 - 父/超类是一个或多个原有的类
 - 子类继承父类所有属性和方法，新属性&方法可直接定义，方法可以重写
- 格式

```
class Subclass_name(Parent_class_name):  
    def __init__(self, ...):  
        super().__init__(...)
```

- super() 用来初始化父类属性
- super(class, obj) 返回一个超类的关联实例
- 将[实例]用作属性
 - 将大型类拆分成多个协同工作的小类
 - 类定义时将实例用作属性

8.5 类的编码风格

- 名字
 - 类名：驼峰命名法，每个单词的 [首字母]大写，不使用下划线
 - 实例名&模块名：小写格式，单词之间加上下划线
- 对于每个类和模块应包含文档字符串
- 用空行来组织代码
 - 一个空行分隔方法、标准库模块 &自己模块
 - 两个空行分隔类
- 自定义工作流程
 - 先尽可能在一个文件中完成所有工作
 - 再将类移到独立的模块中

8.6 类的内建函数

- 布尔函数
 - issubclass(sub, sup): sub 是(否) sup 的子类/子孙类
 - isinstance(obj, Class): obj 是(否) Class(或其子类)的一个实例，Class 可以是多个类的[元组]
- *attr() 系列
 - 使用范围各种对象
 - 使用方法 *attr(obj, "attr"...)
 - hasattr(obj, name): 检查属性是否存在
 - getattr(obj, name[, default]): 获取属性的值，可提供默认值
 - setattr(obj, name,value): 设置一个属性，不存在则新建

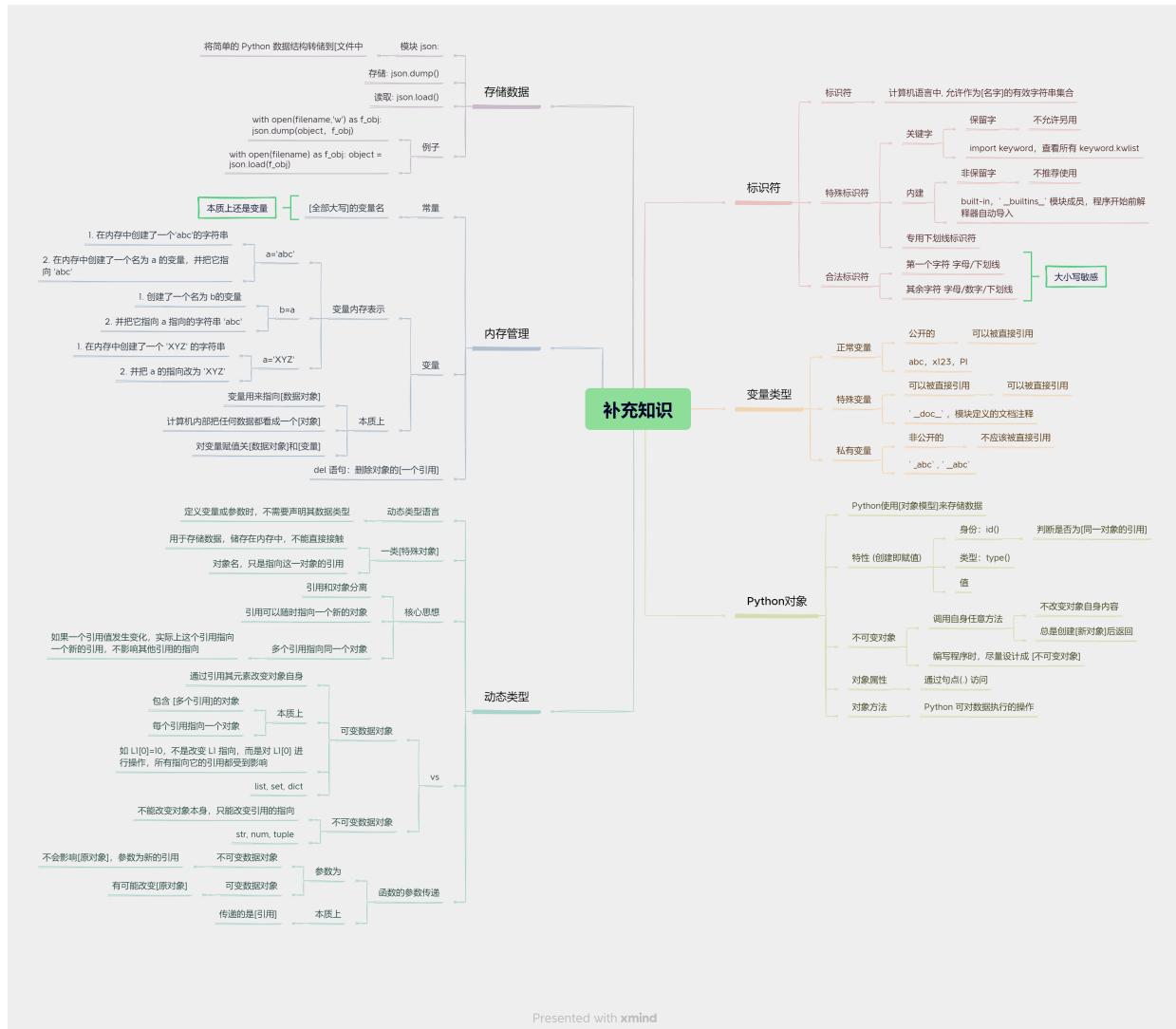
- `delattr(obj, name)`: 删除属性
- `dir()`: 获得一个对象的所有属性和方法
- `super()`: 获取相应父类
- `vars()`: 字典, 返回属性-值
- 内置类属性
 - `__name__`: 类名
 - `__doc__`: 类的文档字符串
 - `__bases__`: 所有[父类]构成的元组
 - `__dict__`: 属性
 - `__module__`: 类定义所在的模块
 - `__class__`: 对象属于哪个类

8.7 其他

- 运算符
 - 本质上是定义在[类内部]的[特殊方法] `__add__()`
 - 可自定义, 根据目的修改对象运算
- 接口
 - 内涵: 对外暴露的方法 & 属性
 - 检查: `hasattr/callable/__dict__`
- 抽象基类
 - 抽象类, 不能实例化
 - 内涵: 包含[抽象方法]的类
 - 职责: 定义子类应实现的一组抽象方法
 - 使用
 - `from abc import ABC, abstractmethod`
 - ```
class Talker(ABC):
 @abstractmethod
 def talk(self): pass
```
    - `@abstractmethod`: 将方法标记为抽象的子类中必须实现的方法

## 9. 补充知识

---



Presented with xmind

## 9.1 标识符

- 标识符
  - 计算机语言中, 允许作为[名字]的有效字符串集合
- 特殊标识符
  - 关键字
    - 保留字, 不允许另用
    - import keyword, 查看所有 keyword.kwlist
  - 内建
    - 非保留字, 不推荐使用
    - built-in, \_\_builtins\_\_ 模块成员, 程序开始前解释器自动导入
  - 专用下划线标识符
- 合法标识符
  - 第一个字符 字母/下划线
  - 其余字符 字母/数字/下划线

## 9.2 变量类型

- 正常变量
  - 公开的，可以被直接引用
    - abc, x123, PI
- 特殊变量
  - 可以被直接引用，可以被直接引用
    - \_\_doc\_\_，模块定义的文档注释
- 私有变量
  - 非公开的，不应该被直接引用
    - \_\_abc, \_\_abc

## 9.3 Python对象

- Python使用[对象模型]来存储数据
- 特性(创建即赋值)
  - 身份: id(), 判断是否为[同一对象的引用]
  - 类型: type()
  - 值
- 不可变对象
  - 调用自身任意方法
    - 不改变对象自身内容
    - 总是创建[新对象]后返回
  - 编写程序时，尽量设计成[不可变对象]
- 对象属性：通过句点(.)访问
- 对象方法：Python可对数据执行的操作

## 9.4 动态类型

- 动态类型语言：定义变量或参数时，不需要声明其数据类型
- 一类[特殊对象]
  - 用于存储数据，储存在内存中，不能直接接触
  - 对象名，只是指向这一对象的引用
- 核心思想
  - 引用和对象分离
  - 引用可以随时指向一个新的对象
  - 多个引用指向同一个对象，如果一个引用值发生变化，实际上这个引用指向一个新的引用，不影响其他引用的指向
- VS

- 可变数据对象
  - 通过引用其元素改变对象自身
  - 本质上是包含 [多个引用] 的对象，每个引用指向一个对象
  - 如 L1[0]=10，不是改变 L1 指向，而是对 L1[0] 进行操作，所有指向它的引用都受到影响
  - list, set, dict
- 不可变数据对象
  - 不能改变对象本身，只能改变引用的指向
  - str, num, tuple
- 函数的参数传递
  - 参数为
    - 不可变数据对象：不会影响 [原对象]，参数为新的引用
    - 可变数据对象：有可能改变 [原对象]
  - 本质上传递的是 [引用]

## 9.5 内存管理

- 常量：[全部大写]的变量名
- 变量
  - 变量内存表示
    - a='abc'
      1. 在内存中创建了一个 'abc' 的字符串
      2. 在内存中创建了一个名为 a 的变量，并把它指向 'abc'
    - b=a
      1. 创建了一个名为 b 的变量
      2. 并把它指向 a 指向的字符串 'abc'
    - a='XYZ'
      1. 在内存中创建了一个 'XYZ' 的字符串
      2. 并把 a 的指向改为 'XYZ'
  - 本质上
    - 变量用来指向 [数据对象]
    - 计算机内部把任何数据都看成一个 [对象]
    - 对变量赋值关 [数据对象] 和 [变量]
  - del 语句：删除对象的 [一个引用]

## 9.6 存储数据

- 模块 json: 将简单的 Python 数据结构转储到文件中
- 存储: json.dump()
- 读取: json.load()
- 例子
  - with open(filename,'w') as f\_obj: json.dump(object, f\_obj)
  - with open(filename) as f\_obj: object = json.load(f\_obj)

## 10. 文件对象



## 10.1 文件对象

- 访问
  - 文件只是[连续的字节序列]
  - 普通磁盘文件或类文件
- 内建函数
  - open()
  - file()
- 内建属性
  - file.closed(): 返回 True/False

- file.encoding()
- file.mode(): 访问模式
- file.name()

## 10.2 标准文件对象

- 标准输入 sys.stdin: 来自键盘的输入, input()
- 标准输出 sys.stdout: 输出到显示器, print()
- 标准错误 sys.stderr: 到显示器, 非缓冲输出

## 10.3 读取文件

- 整个文件
  - ```
with open() as file_object:  
    contents = file_object.read()
```
 - open() 打开文件
 - 模式: 'r/w/a' 读取/写入/附加
 - with用法: 不再需要访问文件后手动关闭文件
- 逐行读取
 - ```
with open() as file_object:
 for line in file_object:
```

- 包含[文件各行内容]的列表
  - ```
with open() as file_object:  
    lines = file_object.readlines()
```

10.4 写入文件

- 只能写入 [字符串]
- 空文件
 - ```
with open(filename, 'w') as file_object:
 file_object.write('...')
```
  - 附加到文件使用附加模式 'a'
  - writelines()
- 写入多行: 注意添加 "\n" 指定换行

## 10.5 内建方法

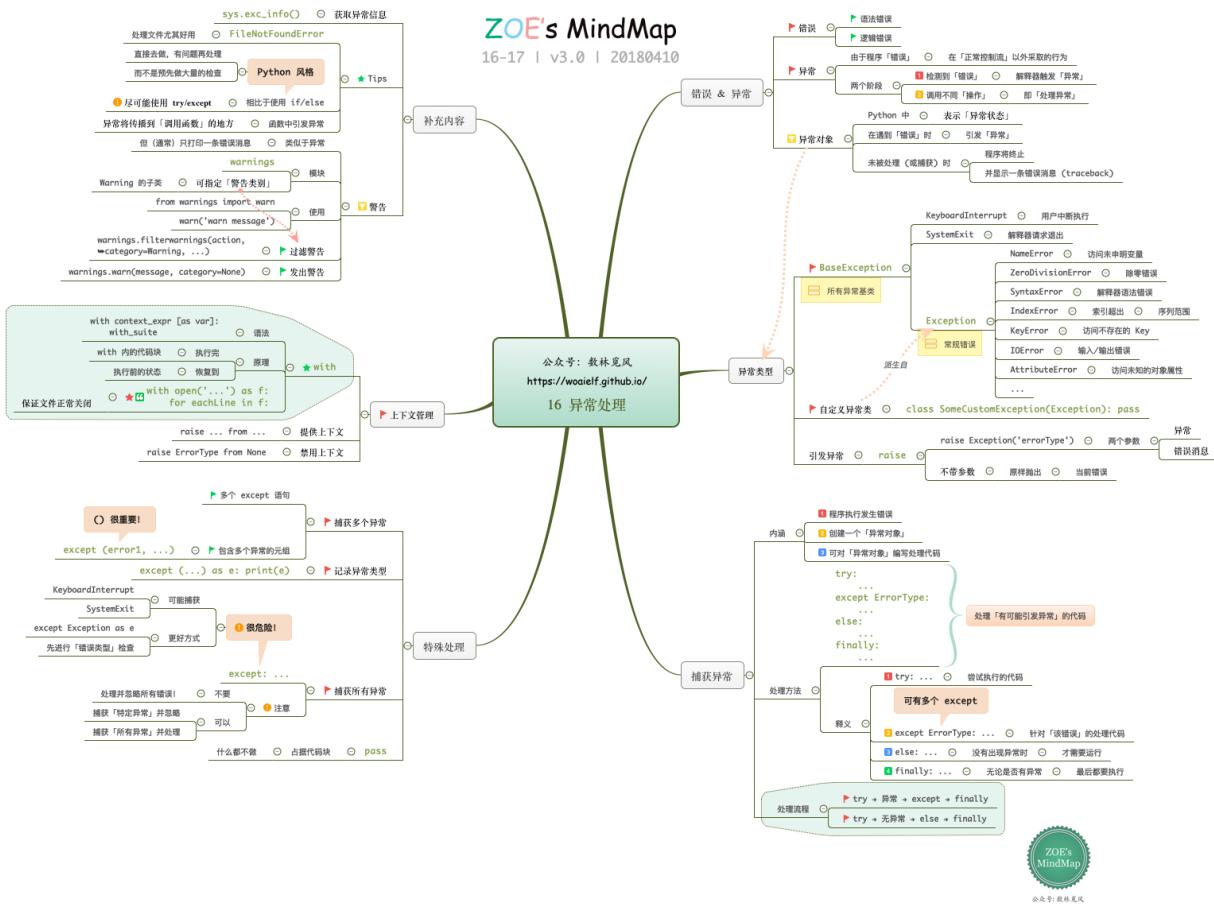
- 文件定位
  - file.seek(): 移动文件指针
  - file.tell(): 文件指针当前位置
- 杂项操作
  - file.close(): 关闭文件，没有显式关闭文件可能会丢失[缓冲区数据]
  - file.flush(): 把内部缓冲区数据立刻写入文件
  - file.truncate()
    - 裁取到当前文件指针位置or给定size
    - 若刚打开即调用函数，文件被删除，从0开始截取

## 10.6 分隔符

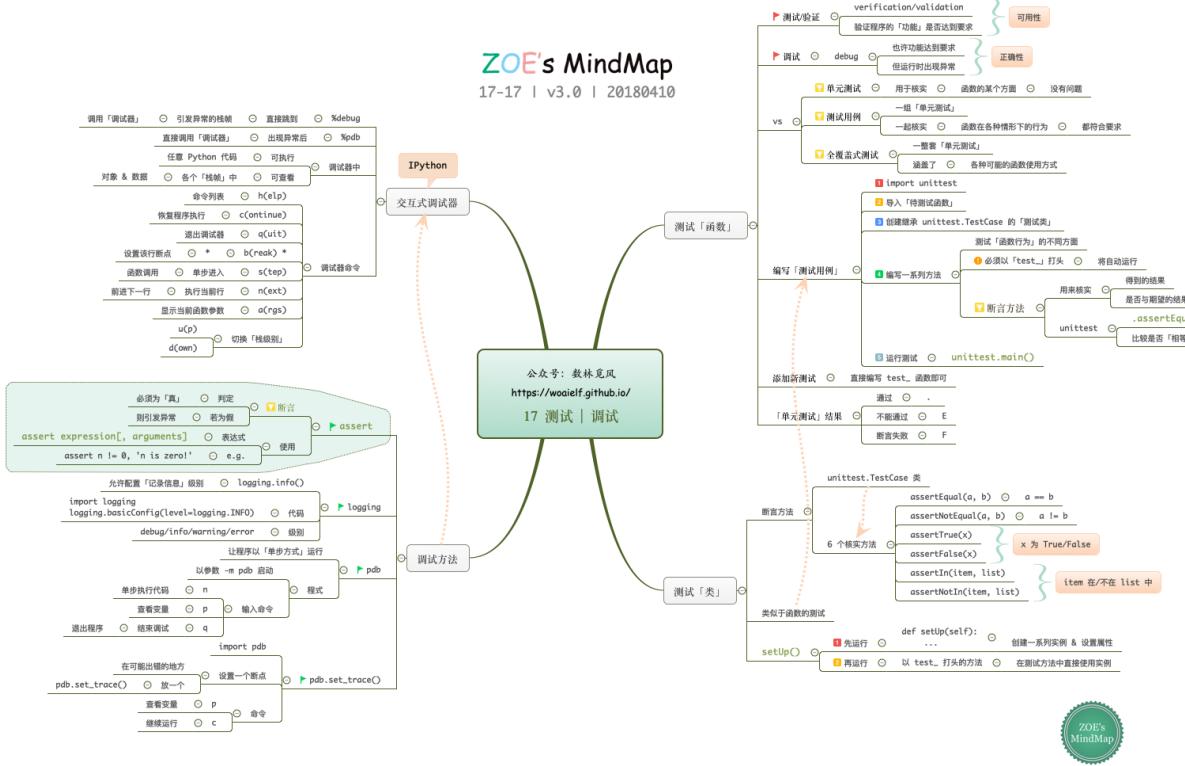
- 行分隔符
  - POSIX (Unix/Mac OS X): \n
  - DOS/Windows: \r\n
  - 旧版 Macos: \r
- 路径分隔符
  - POSIX (Unix/Mac OS X): /
  - DOS/Windows: \
  - 旧版 Macos: :
- import os
  - os.linesep: 行分隔符
  - os.sep: 路径名分隔符
  - os.pathsep: 路径分隔符
  - os.curdir: 当前工作目录字符串表示
  - os.pardir: 父目录字符串表示

## 11. 异常处理

---



## 12. 测试 & 调试



## 13. 面试题

---

[https://blog.csdn.net/qq\\_37085158/category\\_11741646.html](https://blog.csdn.net/qq_37085158/category_11741646.html)

[https://blog.csdn.net/qq\\_37085158/article/details/126821933](https://blog.csdn.net/qq_37085158/article/details/126821933)

<https://github.com/ZhiyuSun/python-pearl/tree/main/interview>