**Python**

1. Python开发环境搭建即安装Python解释器。

Python的解释器分类：

CPython（官方）：用c语言编写的Python解释器

PyPy：用Python语言编写的Python解释器

IronPython：用.net编写的Python解释器

Jython：用Java编写的Python解释器

1. 基本概念：
2. 表达式：

表达式一般仅仅用于计算一些结果，不会对程序产生实质性的影响。

如果在交互模式中输入一个表达式，解释器会自动将表达式的结果输出。

1. 语句：

语句的执行一般会对程序产生一定的影响。

语句在交互模式中不一定会输出执行结果。（eg. a = 3）

1. 程序（program）：

程序就是由一条一条的语句和一条一条的表达式构成的。

1. 一条语句可以分多行编写，语句后以‘\’结尾
2. Python是缩进严格的语言
3. 注释：#（#后一般会有一个空格）
4. 字面量和变量

字面量：一个一个的值

变量(variable)：

1. Python中使用变量不需要声明
2. Python是一个动态类型的语言，可以为变量赋任意类型的值
3. Python中数值分为整数、浮点数（小数）、复数

整数：(1) Python整数都是int类型，且整数大小无限制

(2) 如果数字长度过长，可以使用下划线作为分隔符（eg. 123\_456\_789）

(3) 其他进制整数：二进制以ob开头，八进制以0o开头，十六进制以0x开头

浮点数：(1) Python浮点数都是float类型

(2) 对浮点数进行计算时可能会得到一个不精确的结果。（因为二进制计算时

无法表示1/10）

1. 字符串
2. Python中的字符串需要用引号引起来，可以是单引号或者双引号
3. 相同的引号间不能嵌套
4. 长字符串：使用三重引号，三重引号可以换行，并且会保留字符串中的格式('''或者""")
5. 转义字符：\uxxxx表示Unicode编码
6. 格式化字符串：

* Python中字符串只能和字符串进行‘+’拼接
* 在创建字符串时，可以在字符串中指定占位符

%s为字符串占位符，%f为小数占位符，%b为整数占位符

|  |
| --- |
| b = 'Hello %s' % '孙悟空'  b = 'Hello %s 你好 %s' % ('tom','xun')  # 字符串长度限制在3~5之间，不够位数时用空格补  b = 'Hello %3.5s' % 'ab'  b = 'Hello %.2f' %123 (自动四舍五入) |

* 格式化字符串：在字符串前添加一个f来创建格式化字符串，在格式化字符串中可以直接嵌入变量（eg. b = f'hello {a}'）

1. 字符串的复制

|  |
| --- |
| a = '123'  a = a\*2 |

1. bool值：True、False

bool值实际上也属于整型，True相当于1，False相当于0

1. None(空值)：专门用来表示不存在
2. 对象：

* 对象就是内存中专门用来储存指定数据的一块区域。
* 对象结构：

每个对象中都要保存三种数据：

1. id(标识)：- id用来标识对象的唯一性，每一个对象都有唯一的id

- 通过id()函数可以查看对象的id

- id是由解释器生成的，而在CPython中，id就是对象的内存地址

- 对象一旦创建，id永远不会被改变

(2) type(类型)：- 类型用来表示当前对象所属的类型，类型决定了有哪些功能

- 通过type()函数可以查看对象的type

- Python是一门强类型语言，类型一旦被创建，就不会被改变

(3) value(值)

* 对象分类：

可变对象：

不可变对象：目前的对象皆是不可变对象

* 变量和对象：

对象并没有直接储存到变量中，变量中储存的是对象的id

1. 类型转换：

* 类型转换不是改变对象本身的类型，而是创建一个新的对象。
* 四个转换函数：

int()、float()、str()

bool()：任何对象都可以转换为布尔值，任何表示空的对象都会被转换为False

（eg. 0，''，None）

1. 运算符(操作符)：

* 算术运算符
* +、-、\*、%
* 除法运算符(/)：运算时总返回一个浮点类型。
* 整除运算符(//)：只会保留计算后的整数位
* 幂运算符(\*\*)
* 关系运算符
* 不支持字符串和数字之间的比较，但支持字符串与字符串之间的比较(逐位比较字符串的Unicode编码)
* ==、!=比较的是对象的值（value）
* is、is not比较两个对象是否为同一对象（id）

|  |
| --- |
| **a = [1,2,3]**  **b = [1,2,3]**  **print(a == b)** # True  **print(a is b)** # False |

* Python中关系运算符可以连着使用

|  |
| --- |
| **result = 1 < 2 < 3** #True  # 相当于1 < 2 and 2 < 3 |

* 逻辑运算符
* not逻辑非(一元运算符)

对于布尔值，非运算会对其进行取反运算

对于非布尔值，非运算会将其转化为布尔值，然后再取反

* and逻辑与
* or逻辑或
* 非布尔值进行与或运算时，Python会将其当作布尔值进行运算，最终会返回原值。

|  |
| --- |
| **print(1 and 2)** # 2  **print(0 or 1)** # 1  **print(None and 0)** # None |

* 运算符的优先级

从上到下优先级逐渐升高



1. 流程控制语句

* 条件判断语句：如果代码直接写在冒号后边，则if语句只覆盖冒号后的语句

|  |
| --- |
| # if语句  **if 条件表达式 ：**  **代码块**  # if – else语句  **if 条件表达式 ：**  **代码块**  **else ：**  **代码块**  # if-elif-else语句  **if 条件表达式 :**  **代码块**  **elif 条件表达式 ：**  **代码块**  **else :**  **代码块** |

Python中缩进有两种方式：一种是使用tab键，一种是使用4个空格（官方推荐使用空格），但在代码中使用的风格必须一致。

1. input()函数

该函数用来获取用户输入

用户输入完成之后，其输入内容会以返回值形式返回

input()返回值永远是一个字符串

input()函数中可以设置一个字符串作为参数，这个字符串将会作为提示文字显示

1. 循环语句

|  |
| --- |
| # 循环结束时，如果有对应的else，执行else中的代码块  **while 条件表达式 ：**  **代码块**  **else**  **代码块** |
| **for 变量 in 序列 ：**  **代码块** |

1. break 可以用来立刻退出循环语句，包括else

continue 可以用来退出本次循环

pass 用来在判断和循环中占位，没有实际意义

1. 补充

* print不换行：**print(i,end=’’)**
* 导入time包，计算程序运行时间

|  |
| --- |
| # 模块，通过模块可以对python进行扩展  # 引入一个time模块，来统计程序运行时间  from time import \*  # time()函数可以用来获取当前时间，返回值以秒作为单位  begin = time() |

1. 列表（list）

* 列表是python中的一个对象。
* 对象是内存中专门用来储存数据的一块区域。
* 列表创建：my\_list = []
* 创建只包括一个元素的列表：my\_list = [10]
* 列表中可保存任意对象：my\_list = [10,’hello’,True,None]
* 获取列表中的元素：my\_list[索引]

列表中的索引可以为负数，若为负数，则表示从后往前获取元素，-1表示倒数第一个元素。

* 获取列表中元素的个数：len(my\_list)
* 列表切片
  + 切片：从现有列表中获取一个子列表。
  + 语法：列表[起始位置：结束位置]

列表[起始位置：结束位置：步长]

* + 通过切片获取子列表时，会包括起始位置元素，但不会包括结束位置元素。
  + 起始位置和结束位置都可省略不写。
  + 步长不能为0，但可以为负数。如果为负数，会从列表的后部向前获取元素。
* 通用操作
* + 可以将两个列表连接成一个列表
* \* 可以将列表重复指定的次数
* in 用来检查指定元素是否在列表中

not in 用来检查元素是否不在列表中

|  |
| --- |
| ‘a’ not in my\_list |

* min() 获取列表中的最小值

max() 获取列表中的最大值

* s.index(‘a’) 获取指定元素在列表中第一次出现的索引。若该序列中无该元素，则会抛出异常。

该方法第二个参数表示查找的初始位置

该方法第三个参数表示查找的结束位置

* s.count(‘a’) 统计指定元素在列表中出现的次数
* 修改列表中的元素
* 通过索引修改元素
* 通过切片来修改列表（在给切片进行赋值时，只能使用序列）

当设置了步长时，序列中的元素个数必须和切片中个数一致。

|  |
| --- |
| stus = [‘m’,’n’,’d’,’e’,’f’]  stus[::2] = [‘a’,’b’,’c’] |

* 删除索引为2的元素：del stus[2]

1. 序列（sequence）

* 分类：
* 可变序列：列表（list）
* 不可变序列：字符串（str）、元组（tuple）

不可变序列无法通过索引修改，但可通过list()函数将其他序列转化为list序列

* 可变序列的方法：
* s.append(x) 向序列最后添加元素x
* s.insert(i,x) 向序列指定位置插入元素
* s.extend(t) 需要一个序列作为参数，会将序列中的元素添加到当前序列中
* s.clear() 清空序列
* s.pop(i) 根据索引删除并返回被删元素。若省略参数，则删除最后一个元素。
* s.remove(x) 删除指定值元素，如果相同值元素有多个，只会删除第一个
* s.reverse() 反转列表
* s.sort() 对列表中元素进行排序，默认为升序排序

s.sort(reverse = True) 则为降序排序

* s.copy() 对列表进行浅复制，相当于c[:]

1. range() 是一个函数，可用来生成一个自然数的序列。

* range() 需要三个参数。

第一个参数为起始位置（可省略，默认为0）

第二个参数为结束位置。

第三个参数为步长（可省略，默认为1，也可为负数）

* 通过range() 可创建一个执行指定次数的for循环

|  |
| --- |
| **for i in range(10) :**  **print(i)** |

1. 元组（tuple）：不可变序列

* 一般当我们希望数据不能改变时，使用元组，其他情况使用列表。

eg. **my\_tuple = ()**

* 当元组不是空元组时，括号可以省略。

eg. **my\_tuple = 10,20,30,40**

* 若元组不是空元组时，里边至少有一个“,”

eg. **my\_tuple = 40,**

* 元组的解包（解构）：将元组中的每一个元素都赋值给一个变量

|  |
| --- |
| **my\_tuple = 10,20,30,40**  **a,b,c,d = my\_tupel** |

元组解包的应用：

* 交换两个变量的值：

eg. **a,b = b,a**

* 在对元组进行解包时，变量的个数必须和元组中的元素个数一致；

另外可以在一个变量前加“\*”，则该变量会获取该元组所剩余所有元素。

但不能出现两个或多个“\*”变量。

eg. **a,b,\*c = my\_tuple**

1. 字典（dict）

* 字典属于一种新的数据结构，称为映射（mapping）。
* 字典的作用和列表类似，都是用来储存对象的容器。
* 列表储存数据的性能很好，但是查询数据的性能很差。
* 在字典中每一个元素都有一个唯一的名字，通过这个唯一的名字可以快速查找到指定的元素。在查找元素时，字典的效率很高。
* 在字典中可以保存多个对象，每个对象都有一个唯一的名字。

这个唯一的名字，我们称其为键（key）,通过key可以快速的查询到value

这个对象，我们称其为值（value）。

所以字典，我们也称做键值对（key-value）结构。

每个字典中都可以有多个键值对，而每一个键值对我们称其为一项（item）

* 字典的值可以是任意对象，字典的键可以是任意的不可变对象
* 字典的键是不能重复的，若有重复的，后边的会替换掉前边的
* 如果使用了字典中不存在的键，会抛出异常

|  |
| --- |
| **d = {}** #创建一个空字典  **d = {‘name’:’孙悟空’,’age’:18}** |

1. 字典常用操作

* 创建字典
* **d = {k1:v1,k2:v2}**
* **d = dict(name=’sun’,age=18)**

通过该方式创建的字典，key的类型都是字符串

* **d = dict([(‘name’,’sun’),(‘age’,18)])**

可将一个包含有双值子序列的序列转化为字典。

* 修改字典
* **d[key] = value**

若key存在则覆盖，不存在则添加。

* **d.setdefault('name’,’默认值’)**

向字典中添加key-value，若key存在于字典中，则返回key的value值，不做任何修改；若key不存在，则向字典中添加key，并设置默认值。

* **d.update(d2)**

将其他字典(d2)中的key-value添加到当前字典(d)中。如果有相同的key，则后边的会覆盖前边的。

* 删除字典
* **del d['sun’]**
* **d.popitem()** 删除字典中最后一个键值对，并返回由被删元素组成的元组。当删除空字典时会抛出异常。
* **d.pop(‘name’,’默认值’)** 根据key删除字典，并返回被删除的value值。

若指定了默认值，删除不存在的key时，不会抛出异常，而是直接返回默认值。

* **d.clear()** 清空字典
* 遍历字典
* **d.keys()** 该方法会返回一个序列，其中包括字典中所有的key
* **d.values()** 该方法会返回一个序列，其中包括字典中所有的value
* **d.items()** 该方法会返回一个序列，序列中包含有双值子序列，代表字典中所有的项。
* 其他操作
* **len()** 用于获取字典中键值对个数
* **in** 判断字典中是否有指定的键

**not in** 判断字典中是否不含指定的键

* **d.copy()** 该方法用于对字典进行浅复制，浅复制只会简单的复制对象内部的值，如果值也是一个对象，它不会被复制。
* **d.get(‘name’,’默认值’)**

不带默认值时，该方法用来根据键获取字典中的值，若该键不存在，则返回None；

当指定一个默认值作为第二个参数时，若键不存在，则返回默认值。

1. 集合（set）

* 集合和列表的不同点：

集合中只能储存不可变对象。

集合中储存的对象是无序的。（不是按照元素的插入顺序保存）

集合中不能出现重复的元素。

* 创建集合：
* s = {1,2,3,4}
* 创建空集合：s = set()

set()可将序列和字典转化为集合。使用set()将字典转化为集合时，只会包含键。

* 添加元素
* s.add(x) 将x添加到集合中
* s.update() 将一个集合/列表/字典中的元素添加到当前集合中。
* 删除元素
* s.pop() 随机删除并返回集合中的一个元素。
* s.remove(x) 删除集合中指定的元素。
* s.clear() 清空集合。
* 其他
* s.copy() 对集合进行浅复制。
* in/not in 判断一个元素是否在/不在集合中
* len() 获得集合中元素的个数
* 集合的运算
* & 交集运算
* | 并集运算
* - 差集运算
* ^ 异或集（对称差），获取只在一个集合中出现的元素
* <= 检查一个集合是否是另一个集合的子集。
* < 检查一个集合是否是另一个集合的真子集。
* >= 检查一个集合是否是另一个集合的超集。
* > 检查一个集合是否是另一个集合中的真超集。

1. 函数（function）

* 函数也是一个对象，可以用来保存一些可执行的代码。
* 语法：

|  |
| --- |
| **def 函数名([形参]) ：**  **代码块** |

* 形参（形式参数）：定义形参就相当于在函数内部声明了变量，但是并不赋值。

定义形参时，可以为形参指定默认值。

eg. **def fn(a,b,c=10) :**

* 实参（实际参数）的传递方式：
* 位置参数：将对应位置的实参赋值给对应位置的形参。
* 关键字参数：可以不按照形参定义的顺序传递，而直接根据参数名传递参数。

|  |
| --- |
| **def fn(a,b,c) :**  **pass**  **fn(b=1,c=2,a=3)** #调用 |

* 位置参数和关键字参数可混合使用，但此时位置参数必须写在关键字参数前面
* 函数在调用时，Python解析器不会检查实参的类型。
* 不定长参数
* \* 参数

1. 在定义函数时，可以在形参前加上“\*”，这样这个形参会获取到所有的实参，它将会将所有的实参保存到一个元组中。（装包）

**def fn(\*nums) :**

**pass**

1. 带“\*”号的参数只能有一个，同时其也可和其他参数配合使用。
2. 带“\*”的参数不是必须写在最后，但是带“\*”的参数后的所有参数必须以关键字参数形式进行传递。
3. 如果在形参的开头直接写一个“\*”，则要求所有的参数必须以关键字参数的形式进行传递。

eg. **def fn(\*,a,b,c) :**

**pass**

1. \*形参只能接受位置参数，而不能接受关键字参数。

* \*\* 形参

1. \*\*形参可以接受其他关键字参数，它会将这些参数统一保存到一个字典中。

eg. **def fn(\*\*a) :**

**pass**

1. \*\*形参只能有一个，必须写在所有参数最后

* 参数的解包
* 传递参数时，可以在序列类型的参数前加“\*”号，这样会自动将序列中的元素依次作为参数传递。但此时要求序列中的元素个数必须与形参的个数一致。

|  |
| --- |
| **def fn(a,b,c) :**  **pass**  **t = (10,20,30)**  **fn(\*t)** |

* 通过\*\*可对一个字典的值进行解包操作。
* 返回值（return）
* return 后可跟任意对象，甚至可以是一个函数。

|  |
| --- |
| **def fn1() :**  **def fn() :**  **print("hello")**  **return fn** |

* 如果return后不跟任何值，或者不写return，则相当于return None
* 文档字符串（doc str）
* help(函数对象) 该函数可以查询Python中函数的用法。
* 在定义函数时，可在函数内部编写文档字符串，文档字符串就是函数的说明，当我们编写文档字符串时，就可以通过help()函数来查看函数的说明。
* 文档字符串直接在函数的第一行写一个字符串（通常使用三引号）

|  |
| --- |
| **def fn(a:int,b:bool,c:str) ->int :**  **'''**  **这是一个文档字符串的示例**  **'''**  **return 100** |

* 作用域（scope）
* 作用域指的是变量生效的区域。
* 在Python中一共有两种作用域：

1. 全局作用域
   * + 全局作用域在程序执行时创建，在程序执行结束时销毁。
     + 所有函数以外的区域都是全局作用域。
     + 在全局作用域中定义的变量都属于全局变量，可以在程序的任意位置访问。
2. 函数作用域
   * + 函数作用域在函数调用时创建，在函数结束时销毁。
     + 函数每调用一次就产生一个新的函数作用域。
     + 在函数作用域中定义的变量，都是局部变量，只能在函数内部被访问。

* Python中可以进行函数的嵌套定义，在C++中不允许。
* 如果希望在函数内部修改全局变量，则需要使用global关键字来声明变量。

|  |
| --- |
| **a = 30**  **def fn3() :**  **global a**  **a = 10**  **fn3()**  **print(a)** # 10 |

* 命名空间（namespace）
* 命名空间指的是变量储存的位置，每一个变量都需要储存到指定的命名空间中。
* 每一个作用域都有一个对应的命名空间。
* 命名空间实际上就是一个字典，是一个专门用来存储变量的字典。
* locals() 函数用来获得当前作用域的命名空间。如果在全局作用域中调用locals()则获取全局命名空间，如果在函数作用域中调用locals()则获取函数命名空间，结果以一个字典的形式返回。
* 向命名空间的字典中添加key-value相当于在全局中创建一个变量（一般不建议这么做）
* globals() 函数可以在任意位置获取全局命名空间。

1. 递归

* 整体思想：将一个大问题分解为一个个的小问题，直到问题无法分解时，再去解决问题。
* 递归式函数的两个条件：

1. 基线条件

问题可以被分解为的最小问题，当满足基线条件时，递归就不再执行。

1. 递归条件

将问题继续分解的条件。

1. 函数式编程

* Python支持函数式编程。在Python中，函数式一等对象。
* 一等对象一般会具备如下特点：

1. 一等对象是在执行时创建的。
2. 一等对象能赋值给变量或作为数据结构中的元素。
3. 一等对象能作为参数传递。
4. 一等对象能作为返回值返回。

* 高价函数

高阶函数至少要符合以下两个特点中的一个：

1. 接收一个或多个函数作为参数。当我们使用一个函数作为参数时，实际上是将指定的代码传递进入了目标函数。
2. 将函数作为返回值返回

* **filter(function,iterator)** 过滤器可以从序列中过滤出符合function条件的元素，并保存到一个新的序列中。
* **map(function,iterator)** 函数可以对可迭代对象中的所有元素做指定操作，然后将其添加到一个新的对象中返回。
* **sort()** 方法可以对列表进行排序，可接收一个关键字参数key，key需要一个函数作为参数，当设置了函数作为参数，每次都会以列表中的一个元素作为参数来调用函数，并且使用函数的返回值来比较元素的大小。

eg. **l.sort(key=len)** #按字符串长度由小到大排序

**l.sort(int)**

* **sorted(l)** 函数和sort()的用法基本一致，但是sorted()可以对任意的序列进行排序，并且用sorted()排序不会影响原来的对象，而是返回一个新对象。

1. 匿名函数（lambda函数表达式/语法糖）

专门用来创建一些简单的函数，它是函数创建的又一种方式。

语法：**lambda 参数列表 ：返回值**

1. 闭包（高阶函数）

* 将函数作为返回值返回。通过闭包可以创建一些只有当前函数能访问的变量，可以将一些私有的数据藏到闭包中。
* 形成闭包的条件：

1. 函数嵌套
2. 将内部函数作为返回值返回
3. 内部函数必须要用到外部函数的变量

|  |
| --- |
| **def make\_averager() :**  **nums = []**  **def averager(n) :**  **nums.append(n)**  **return sum(nums)/len(nums)**  **return averager** |

1. 装饰器

* 开闭原则（OCP）：程序的设计要求开放对程序的扩展，要关闭对程序的修改。
* 通过装饰器，可以在不修改原来函数的情况下对函数进行扩展。在开发中，我们都是通过装饰器来扩展函数功能的。

|  |
| --- |
| **def begin\_end(old) :**  '''  用来对其他函数进行扩展，使其他函数可以在执行前打印开始执行，执行后打印执行结束  参数：  old 要扩展的函数对象  '''  # 创建一个新函数  **def new\_function(\*args,\*\*kwargs) :** # 装包  **print('开始执行~~~~')**  # 调用被扩展的函数  **result = old(\*args,\*\*kwargs)**  # 拆包  **print('执行结束~~~~')**  **return result**  # 返回新函数  **return new\_function** |

* 在定义函数时，可以通过 @装饰器 来使用指定的装饰器来装饰当前函数。可以同时为一个函数指定多个装饰器，这样函数将会按照从内到外开始装饰。

1. 面向对象（oop）

* 对象（Object）：内存中专门用来储存数据的一块区域。
* Python是一门面向对象的编程语言。面向对象的编程思想，是将所有的功能统一保存到对应的对象中，要使用某个功能，直接找到对应的对象即可，易于维护和复用。

1. 类（class）

* 类也是一个对象，类就是一个用来创建对象的对象。类是type类型的对象。
  + 对象是类的实例。若多个对象是通过一个类创建的，我们称这些对象为一类对象。
  + 创建一个int类的实例，等价于a = 10
  + 自定义类时，使用大驼峰命名法。
  + 可以向对象中添加变量，对象中的变量称为属性。

语法：对象.属性名 = 属性值

* + isinstance(mc,MyClass) 函数用来检查一个对象是否是一个类的实例。
* 使用类创建对象流程

1. 创建一个变量mc
2. 在内存中创建一个新对象。
3. \_\_init\_\_(self) 执行
4. 将对象的id赋值给变量。

* 类的定义
  + 所有的事物都由两部分构成：

1. 数据（属性）
2. 行为（方法）
   * 在类的代码块中，我们可以定义属性和方法。其属性将会成为所有实例的公共属性，通过 对象.属性名 可以访问到该属性；其方法将会成为所有实例的公共方法，通过 对象.方法名() 可以访问到该方法。
   * 方法调用和函数调用的区别：

如果是函数调用，则调用时传入几个参数，就会有几个实参；但如果是方法调用，默认会传递一个参数，所以方法定义时，至少要定义一个形参。

* 属性和方法查找流程：

当我们调用一个对象的属性时，解析器会先在当前对象中寻找是否含有该属性，若没有，则到当前对象的类对象中查找。

* 类对象和实例对象都可以保存属性（方法）

若这个属性（方法）是所有的实例共享的，则应该将其保存到类对象中；

若这个属性（方法）是某个实例所独有的，则应该将其保存到实例对象中。

一般情况下，属性保存到实例对象中；方法保存到类对象中。

* 在方法中不能直接访问类中的属性。
* 方法中第一个参数一般为self。
* 在类中可以定义一些特殊方法（魔术方法）
* 特殊方法均以\_\_开头，以\_\_结尾。
* 特殊方法不需要我们自己调用，特殊方法将会在特殊时刻自己调用。

1. 封装

* 封装是面向对象的三大特征之一。封装指的是隐藏对象中一些不希望被外部所访问的属性或者方法。
* 隐藏一个对象中的属性：
* 将对象的属性名修改为一个外部不知道的名字。（\_name）

增加gettter和setter方法，可以很好的控制该属性是否可读。

使用setter方法设置属性，可以增加数据的验证，确保数据的值是合法的。

* 可以为对象的属性使用双下划线开头，此时该属性为对象的隐藏属性。

隐藏属性只能在类的内部访问，无法通过对象访问。

其实隐藏属性只不过是Python自动为属性改了一个名字，实际上修改为了

\_类名\_\_属性名

一般我们会将一些私有属性（不希望被外部访问的属性）以单下划线开头。

我们可以通过object.\_\_dict\_\_ 查看对象的所有属性。

* property装饰器
* property装饰器用来将一个get方法转化为对象的属性。添加为property装饰器后，我们就可以像调用属性一样使用get方法。
* 使用property装饰器的方法，方法名必须和属性名是一样的。

|  |
| --- |
| class People :  def \_\_init\_\_(self,name,age):  self.\_name = name  self.\_\_age = age  @property  def name(self):  return self.\_name    @name.setter  def name(self,name):  self.\_name = name |

1. 继承

* 继承是面向对象的三大特征之一。通过继承我们可以使子类获取到父类中的属性和方法。所以我们经常需要通过继承来对一个类进行扩展。
* 在定义类时，可以在类名后的的括号中指定当前类的父类（超类、基类）

eg. **class Dog(Animal)**

* 在创建类时，如果省略了父类，则默认父类为object；object是所有类的父类，所有类都继承自object。
* **issubclass(Dog,Animal)** 检查一个类是否是另一个类的子类。
* 如果在子类中有和父类同名的方法，则通过子类实例去调用方法时，会调用子类的方法而不是父类的方法，这个特点我们叫做方法的重写（override）。
* super() 可以用来获取当前类的父类，并通过super()返回的对象调用父类方法时，不需要传递self。通过类名调用方法时需要传递self。

1. 多重继承

* 在Python中支持多重继承。
* 类名.\_\_bases\_\_ 这个属性可以用来获取当前类的所有父类。
* 如果多个父类中有同名的方法，则会先在第一个父类中寻找，前边父类的方法会覆盖后边父类的方法。

1. 多态

* 多态是面向对象的三大特征之一。多态从字面上理解是多种形态。一个对象可以以不同的形态去呈现。
* 之所以一个对象能通过len()来获取长度，是因为对象中有一个特殊方法：\_\_len\_\_()

1. 类的属性和方法

* 类属性可以通过类或类的实例访问到，但是类属性只能通过类对象来修改，无法通过实例对象修改。
* 实例属性只能通过实例对象来访问和修改，类对象无法访问和修改。
* 实例方法：在类中定义，以self为第一个参数的方法。

实例方法可以通过实例对象和类对象调用。

当通过实例对象调用时，会自动将调用对象作为self传入。

当通过类对象调用时，不会自动传入self。

**Person.say\_hello(p)** 相当于 **p.say\_hello()**

* 类方法：在类内部使用 @classmethod 来修饰的方法

类方法的第一个参数是cls,也会被自动传递，cls是当前的类对象。

类方法可以通过类和实例调用。

|  |
| --- |
| **@classmethod**  **def class\_method(cls) :**  **print('我是类方法',cls)** |

* 静态方法：在类内使用 @staticmethod 来修饰的方法

静态方法不需要指定默认参数，静态方法可以通过类对象和实例对象去调用。

静态方法基本上是一个和当前类无关的方法，常是一些工具方法。

|  |
| --- |
| **@staticmethod**  **def tool\_method() :**  **print('我是静态方法')** |

1. 垃圾回收

* 在程序中没有被引用的对象就是垃圾。这种垃圾对象过多会影响到程序运行的性能。
* 在Python中有自动的垃圾回收机制，它会将这些没有被引用的对象自动删除。
* **\_\_del\_\_(self)** 是一个特殊方法，它会在对象被回收前自动调用。

1. 特殊方法

* **object.\_\_init\_\_(self)** 对象初始化方法，产生对象后立刻调用
* **object.\_\_del\_\_(self)** 对象被回收前调用
* **object.\_\_str\_\_(self)** 当我们打印一个对象时，实际上打印的是对象中的该方法的返回值（用于print函数）
* **object.\_\_repr\_\_(self)** 该方法会在对当前对象使用repr()函数时调用，作用是指定对象在交互模式中直接输出的结果。
* **object.\_\_len\_\_(self)** 该方法指定len()函数的输出规则
* **object.\_\_bool\_\_(self)** 该方法指定了转化为bool值的规则
* **object.\_\_lt\_\_(self,other)** 小于

**object.\_\_le\_\_( self,other)** 小于等于

**object.\_\_eq\_\_( self,other)** 等于

**object.\_\_ne\_\_( self,other)** 不等于

**object.\_\_gt\_\_( self,other)** 大于

**object.\_\_ge\_\_( self,other)** 大于等于

上述方法的返回值将会作为比较的结果。

1. 模块（module）

* 模块化指将一个完整的程序分解为一个一个小的模块。
* 采用模块化，将程序分别编写到多个文件中。

优点：方便开发；方便维护；模块可以复用

* 在Python中一个py文件就是一个模块，要想创建模块，实际上就是创建一个python文件。故模块名要符号标识符的规范。
* 引入外部模块：
* **import 模块名**

可以引入同一个模块多次，但是模块实例只会创建一个。

* **import 模块名 as 模块别名**
  + - import可以在程序的任意位置调用，但一般情况下，import语句会统一写在程序的开头。
    - 在每一个模块内部都有一个\_\_name\_\_属性，通过该属性可以获取到模块的名字。
    - \_\_name\_\_属性值为\_\_main\_\_的模块是主模块，一个程序中只有一个主模块。（主模块就是我们直接通过python执行的模块）
* **from 模块名 import 变量,变量,…**

也可以只引入模块中的一部分。

**from 模块名 import \***  引入模块中所有内容，但一般不推荐使用

* **from 模块名 import 变量 as 变量别名**
* 模块中添加了\_的变量，通过import \* 引入时，不会引入\_开头的变量。
* 测试代码只有在当前文件是主模块时才需要执行，故我们在书写测试代码时，就必须检查当前模块是否为主模块。

**if \_\_name\_\_ == ‘\_\_main\_\_’ :**

1. 包（Package）

* 当我们模块中代码过多时，或者一个模块需要被分解为多个模块时，这时就需要使用到包。
* 包也是一个模块。普通的模块就是一个py文件，而包是一个文件夹。
* 包中必须要一个\_\_init\_\_.py文件，该文件中可以包含包中的主要内容。
* \_\_pycache\_\_ 是模块的缓存文件。

py代码在执行前，需要被解析器先转换为机器码，然而再执行；所以我们在使用模块（包）时，也需要将模块的代码先转换为机器码，然后再交由计算机执行，而为了提高程序运行的性能，Python会在编译一次后，将代码保存到一个缓存文件中。

1. Python标准库 [https://docs.python.org/3/py-modindex.html]

* **sys**模块，它里面提供了一些变量和函数，使我们可以获取到Python解析器的信息，或者通过函数来操作Python解析器。
* **sys.argv** 获取执行代码时命令行中所包含的参数，该属性是一个列表，列表中保存了当前命令的所有参数。
* **sys.modules** 获取当前程序中引入的所有模块，该属性是一个字典，字典的key是模块名字，字典的value是模块对象。
* **sys.path** 是一个列表，列表中保存的是模块的搜索路径。
* **sys.platform** 返回当前python运行后的平台。
* **sys.exit()** 函数用于退出程序。
* **pprint**模块给我们提供了一个方法pprint()，该方法可以用来对打印的数据做简单的格式化。
* **os**模块让我们可以对操作系统进行访问
* **os.environ** 通过这个属性可以获取到系统的环境变量。

**pprint.pprint(os.environ[‘path’])**

* **os.system()** 函数可以用来执行操作系统的命令。

1. 异常

* 程序在运行过程中不可避免会出现一些错误，这些错误在程序中，我们称为异常。
* 程序运行过程中，一旦出现了异常，程序将会立即终止，异常以后的代码不会执行。
* 处理异常：

|  |
| --- |
| **try :**  **代码块（可能出错的语句）**  **except [异常类型 as xx] :**  **代码块（出错以后的处理方式）**  **else :**  **代码块（没出错时要执行的语句）**  **finally :**  **代码块（无论是否出现异常均会执行的代码块）** |

* 如果except后不跟任何内容，则此时会捕获到所有的异常；如果在except后跟一个异常类型，则此时只会捕获该类型的异常。
* try语句必须有，else语句可有可无，except和finally语句至少要有一个
* Exception是所有异常类的父类，故except后跟Exception会捕获到所有的异常。
* 可以在异常类型后加as xxx，则此时xxx就是异常对象。
* 异常的传播
* 当在函数中出现异常时，如果在函数中对异常进行了处理，则异常不会再传播；如果在函数中对异常没有进行处理，则异常会继续向函数的调用处传播，直到传播到全局作用域（主模块）
* 当程序运行过程中出现异常后，所有的异常信息会被保存到一个专门的异常对象中，而异常传播时，实际上就是异常对象抛给了调用处。
* 抛出异常

可以使用raise语句抛出异常，raise语句后需跟一个异常类或者一个异常的实例。

|  |
| --- |
| **def add(a,b):**  **if a < 0 or b < 0 :**  **raise Exception('不支持小于零的数进行运算')**  **return a+b** |

* 自定义异常

只需要创建一个类继承Exception即可。

1. 文件（File、I/O）

* 打开文件
* 使用open()函数来打开一个文件

参数：file 要打开文件的名字（路径）

返回值：返回一个对象，这个对象就代表了当前打开的文件。

* 调用open()打开一个文件，可以将文件分为两种类型：

1. 纯文本文件（使用utf-8等编写的文本文件）
2. 二进制文件（图片、mp3、ppt等）

* open()打开文件时，默认是以文本文件的形式打开，但是open()的默认打开编码方式是None，所以处理文本文件时，必须要指定文件的编码。
* 字符串前加r表示原始字符串，该方法会忽略内部的转义字符。
* 关闭文件
* 使用file\_obj.close()方法关闭文件。
* with…as语句

with open(file\_name) as file\_obj :

此时这个文件只能在with中使用，一旦with结束则会自动调用close()

|  |
| --- |
| **def readFile(file\_name):**  **try :**  **with open(file\_name,encoding='utf-8') as file\_obj:**  **chunk = 10**  **while True :**  **content = file\_obj.read(10)**  **if not content :**  **break**  **print(content,end='')**  **except FileNotFoundError :**  **print(f'无此文件{file\_name}，打开失败！')** |

* 读取文件中的内容
* **file\_obj.read()**

1. file\_obj.read()会以一个字符串的形式返回文件中的所有内容。
2. 如果要读取的文件比较大的话，该方法会一次性将文件的内容加载到内存中，容易导致内存泄露，所以对于较大的文件，不要直接调用read()。
3. read()中可以接收一个size作为参数，该参数用来指定要读取的字符的数量，默认值为-1，会读取到文件中的所有字符。
4. 可以为size指定一个值，这样read()会读取指定数量的字符，而且每一次读取都是从上一次读取的位置开始读取；如果字符的数量小于size，则会读取到剩余的所有；如果已经读取到文件的最后，则会返回空串。
5. 读取文本文件时，size以字符为单位；读取二进制文件时，size以字节为单位。

* **file\_obj.readline()**

该方法用来读取一行

* **file\_obj.readlines()**

该方法用于一行一行读取，但它会一次性将读取到的内容封装到一个列表中返回。

* **for t in file\_obj:**

该方法用来读取一行。

* 文件的写入
* **write()**

1. write()方法用来向文件中写入内容。如果操作的是文本文件的话，需要传递所要写入的内容字符串为参数。
2. 使用open()打开文件时，必须要指定打开文件所要做的操作（读、写、追加等）,如果不指定操作类型，则默认是读取文件。
3. 该方法可以分多次向文件中写入内容，写入完成后，该方法会返回写入的字符的个数。
4. 读写操作模式尽量分开，避免出现不必要的问题。

* 读取模式

1. r ：表示只是可读的，文件不存在则会抛出异常FileNotFoundError。
2. w：表示可写的，使用w写入文件时，若文件不存在会创建文件，如果文件存在则会截断文件（删除原文件中所有内容后再进行写入）
3. a：表示追加内容，文件不存在会创建文件。
4. x：用来新建文件，如果文件不存在则创建，文件存在则报错。
5. +：为操作符增加功能

r+：可读可写，文件不存在会报错，是对原文件相应位置的覆盖。

w+：既可读又可写，文件不存在时会创建。

a+：追加写入也可读，文件不存在时会创建。

1. t：读取文本文件模式（默认模式）
2. b：读取二进制文件（eg. **rb**）

* **tell()** 方法用来查看当前读取的位置
* **seek()** 方法用来修改当前读取的位置

1. 参数：第一个参数是要切换到的位置，也可以为负数；

第二个参数是计算位置的方式（可选值）：

1. 0 从头算起，此为默认值。
2. 1 从当前位置计算
3. 2 从最后位置计算
4. seek()也可以对文本文件进行操作,但utf-8编码下的中文，一个字符占三个字节。

* 其他操作

import os

* **os.listdir()** 返回指定目录的目录结构，是一个列表。可以传一个路径作为参数，默认为‘.’即当前目录。
* **os.getcwd()** 获取当前所在的目录。
* **os.chdir()** 切换当前所在的目录
* **os.mkdir(‘aaa’)** 在当前目录下创建一个名字为aaa的目录。
* **os.rmdir(‘abc’)** 删除目录
* **os.remove(‘aaa.txt’)** 删除文件
* **os.rename(‘aa.txt’,’bb.txt’)** 可以对一个文件进行重命名，也可以用来移动一个文件

1. numpy

**import numpy as np**

1. 列表转化为矩阵：**arr = np.array([[1,2,3],[4,5,6],[7,8,9]],dtype=np.int64)**

第二个参数可以指定矩阵中元素的类型：int32、int64、float32、float64

1. **arr.ndim** 返回矩阵的纬度 #2

**arr.shape** 返回矩阵的行列数 #(3,3)

**arr.size** 返回矩阵中元素总数 #9

1. **np.zeros((3,4))** 返回一个全0矩阵

**np.ones((3,4))** 返回一个全1矩阵

**np.empty((3,4))** 返回一个随机矩阵

1. **np.arange(起始值，终值，步长).reshape((3,4))** 返回一个从初始值到终值的矩阵

**np.linspace(起始值，终值，段数)** 返回一个从初始值到终值的矩阵

1. **np.dot(a,b)** 矩阵相乘

**c = a\*b** 矩阵诸个相乘

1. **np.sum(arr)** 求矩阵的和，还可以传一个关键字参数axis，为0表示按列，为1表示按行
2. **np.mean(arr)** 求矩阵的平均值

**np.max(arr)** 求矩阵最大值

**np.min(arr)** 求矩阵最小值

**np.median(arr)** 求矩阵中位数

**np.sqrt(arr)** 为矩阵的每一个元素开方

1. **np.argmin(arr)** 求矩阵最小值索引

**np.cumsum(arr)** 求矩阵前缀和

**np.diff(arr)** 矩阵逐项累差

**np.sort(arr)** 矩阵升序排序

**-np.sort(-arr)** 矩阵降序排序

1. **np.transpose(arr)** 矩阵转置，或者（arr.T）
2. **np.clip(arr,5,9)** 小于5皆为5，大于9皆为9
3. **np.random.random((3,2))** 生成3行2列从0到1的随机数

**np.random.normal(size=(3,2))** 生成3行2列符合标准正态分布的随机数

**np.random.randint(0,10,size=(3,2))** 生成3行2列从0到10的随机整数

1. 矩阵索引从0开始，a[1][1]相当于a[1,1]

矩阵也支持切片：a[:,2]表示提取第2列的数据。

**for i in arr:** 按行迭代

**for i in arr.T:** 按列迭代

**for i in arr.flat:** 按元素迭代

1. **np.vstack((arr1,arr2))** 垂直合并

**np.hstack((arr1,arr2))** 水平合并

**np.concatenate((arr1,arr2),axis=0)** 矩阵合并

1. **arr[np.newaxis,:]** 给行增加维度

**np.atleast\_2d(arr)** 判断是否为2维及以上纬度，若是则不发生变化；若不是则按行增加纬度转化为2维

1. **np.split(arr,[1,2],axis=1)** 按水平方向，将arr分为3份，在第1列和第2列后切分。
2. pandas

**import pandas as pd**

1. **pd.Series([4,7,-5,3])** 创建Series(类似于dict)，索引为默认值

**pd.Series([4,7,-5,3],index=['a','b','c','d'])**

1. **s.values** series的值

**s.index** series的索引

1. matplotlib

**import matplotlib.pyplot as plt**

**import numpy as np**

1. **plt.plot(x,y)** 绘图

**plot(x,y,color=’red’,linewidth=1.0,linestyle=’--’)**

1. **plt.show()** 显示图像
2. **plt.figure(figsize=(2,2))** 创建figure，可以传一个figsize关键字参数设置大小。
3. **plt.xlim((-1,2))** x限制范围

**plt.xlabel(‘I am x’)** x轴描述

**plt.xticks(new\_ticks)** x轴自定义坐标值

1. **ax = plt.gca()** 获取坐标轴对象

**ax.spines['right'].set\_color('none')** 设置边框颜色或者消除边框

|  |
| --- |
| # 设置坐标轴到中心  # 把x轴刻度设置为bottom  **ax.xaxis.set\_ticks\_position('bottom')**  **ax.yaxis.set\_ticks\_position('left')**  # 使x轴过y轴的原点  **ax.spines['bottom'].set\_position(('data',0))**  **ax.spines['left'].set\_position(('data',0))** |

1. xlrd

**import xlrd**

1. **table = xlrd.open\_workbook(path).sheets()[0]** 获取path下的第一个sheet表
2. **table.nrow** 读入的行数
3. **table.ncol** 读入的列数
4. openpyxl

**from openpyxl import Workbook, load\_workbook**

1. **wb = load\_workbook(path)** 读取excel表
2. **sheetnames = wb.get\_sheet\_names()** 得到所有的表名
3. **wb.get\_sheet\_by\_name(sheetnames[0])** 得到表sheet1
4. **ws.max\_row** 获取行数

**ws.max\_column** 获取列数