

Python 基础教学

By @鹤翔万里 (TonyCrane)

 Github

 B 站直播间

前言

这次的教学面向没有学过 python 的人，不论有没有编程基础都可以听懂

这里说的东西有些只是为了更好的理解，可能并不严谨，请不要完全听信（免责声明

为什么讲 python

- `manim` 动画引擎的基础

`manim` 是用 `python` 编写的库，使用需要有 `python` 的基础知识（不然会被我劝退.jpg）

- 易用易学、作为工具

`python` 这个语言很易学，特别是对于有编程基础的人（几乎直接上手）。所以有些教程、课程其实过于冗余

- 大学某些课程、老师会默认你会

什么是 python

1. 解释性的脚本语言

通过解释器来直接运行，不需要编译链接成二进制文件

2. 动态类型语言

类型在运行时确定，不需要通过代码明文规定

3. 面向对象语言

python 中一切皆对象

4. ...

怎么装 python

- 装的是什么?

- 是一个 python 解释器, 以及运行需要的环境

- 怎么装?

- 官方网站 <https://www.python.org/downloads/>
 - conda (一个好用的 python 环境管理工具)
 - anaconda (大、有预装环境) <https://www.anaconda.com/>
 - miniconda (小) <https://docs.conda.io/en/latest/miniconda>
 - 极不建议通过微软应用商店安装 python

- 装什么版本?

- 两个大版本, 2.* 和 3.*, 差别较大, 建议 3.*
 - 一些小版本, 3.6 及之前不推荐, 3.7 3.8 稳定, 3.9 3.10 完善中, 3.11 预览中
 - 细分版本, 选择最新, 3.7.13、3.8.13、3.9.12、3.10.4
 - conda 不必担心版本, 默认 3.9, 可以通过创建虚拟环境来使用不同版本

怎么编写、运行 python

- 怎么用 python?
 - 记住你下载的是一个解释器，建议通过命令行运行 `python code.py`
- 什么是命令行?
 - 通过输入命令来通知电脑执行某指令、或者运行某程序
 - Windows: cmd、Powershell → 单独运行 / Windows Terminal / ...
 - macOS: zsh、... → 终端 / iTerm / ...
 - Linux: bash、zsh、... → 终端 / ...
- 用什么写代码?
 - 记住你编写的只是一个 `.py` 作为扩展名的文本文件 只要文本编辑器都可以写
 - ~~记事本、自带 IDLE、word~~
 - Notepad++、Sublime Text
 - VSCode (Visual Studio Code, 不是 VS) code.visualstudio.com
 - Pycharm (Community Edition 就够用) [jetbrains.com/pycharm](https://www.jetbrains.com/pycharm/)

Hello world!

```
print("Hello world!")
```

变量

- 给一个内容绑定一个标签即变量名（注意请不要认为变量类似一个“盒子”）
- 通过 = 来定义变量，变量名 = 内容
- 动态类型，不需要规定类型（可以通过 变量名：类型 = 内容 来进行类型标注）
- 变量名
 - 只能包含字母、数字、下划线、中文，不能有空格和其它符号
 - 只能以字母、下划线开头，不能以数字开头，而且大小写敏感
 - 不能用关键字（例如 if def 等）作为变量名，不推荐使用内置函数名作为变量名
 - 清晰明确、风格统一
 - 全大写一般表示常量、不建议使用双下划线开头或者开头结尾、不建议使用 _ 作为变量名

数字与运算

- 1 是整数, 1. 是浮点数
- 整数与浮点数转换
 - `int(...)`: 向 0 舍入
 - `round(...)`: 向偶舍入 (四舍六入五凑偶, 可以当成四舍五入)
 - `math.floor(...)`、`math.ceil(...)`: 下取整、上取整 (需要 `import math`)
- 运算
 - + - * 加减乘, 左右都是整数结果也是整数, 有浮点数结果就是浮点数
 - / 除法, 结果是浮点数 (即使可以整除)
 - // 整除, 结果是整数, 向下取整
 - % 取模, $a \% b = a - (a//b)*b$ (和 c 的行为不一致)
 - ** 乘方, 可以是浮点数, 比如 `a ** 0.5` 表示开根号
 - `pow(a, b, mod)`: 也是乘方, `mod` 可以省略, 如果有 `mod` 则对结果取模, 如果 `mod` 为 -1 则计算乘法逆元
 - 更多运算通过 `math`、`numpy`、`scipy` 等包来进行

复数类型

- python 中内置了复数类型，`1+2j` 形式就表示一个复数，其中 `j` 即虚数单位 `i`
- 或者使用 `complex(实部, 虚部)` 形式定义复数
- 可以进行复数的加减乘除运算
- 属性与方法
 - `c.real`: 实部
 - `c.imag`: 虚部
 - `c.conjugate()`: 返回共轭复数

字符串

- 单引号 '...', 双引号 "...", 三引号 ''' ... ''' """ ... """ (可以内部换行)
- \n 换行, \t 制表符, \r 回车, ' 单引号, " 双引号, \\ 斜杠 (只打一个 \ 会出问题) ,
- 前缀
 - r-string: r"...", 引号中不进行转义, 即一个 \ 就代表斜杠字符本身
 - f-string: f"...", 格式化字符串
 - b-string: b"...", 将字符串转为 bytes, 只能包含 ASCII 字符
- 常用方法
 - 拼接: 直接将字符串“相加”
 - "...".upper()、"...".lower(): 转为全大写、全小写
 - "...".title(): 单词首字母大写
 - "...".strip(): 删除字符串首尾空白 (包含空格和制表符)
 - "...".lstrip()、"...".rstrip(): 删除左、右端空白
 - "...".split(c): 根据字符 c 来拆分字符串得到列表, 默认拆分空白

f-string

- 格式化字符串方式: "... % ... , "....format(...), f"..."
- 字符串内大括号括起来的计算后转为字符串填入 f" ...{...} ..."
- 如果字符串要用大括号原始字符要写两个 f" ...{{...}}
- 格式化 (在填入内容后面加冒号 f" ... {表达式:格式} ...")
 - 宽度填充: :[填充字符][对齐方式][宽度], < 左对齐, > 右对齐, ^ 居中
 - 字符截断: :[...].n, 只显示字符串的前 n 个字符
 - 数值符号: :+ 正数加正号、负数加负号, :- 原样, : (空格) 正数加空格、负数加负号
 - 数值精度: :[宽度][分隔符(,_)].[精度]f, 没有精度默认为 6
 - 进制显示: x 小写十六进制, X 大写十六进制, o 八进制, b 二进制, 加 # 显示前缀

字节类型

- 类似字符串，但存储的是字节的值，更像列表，显示为 `b"..."` 只是更加易读而已
- `b"..."` 则表示字节类型，其中只能包含 ASCII 字符和 `\x..` 表示的十六进制数
- 与字符串转换
 - `"...".encode(encoding)` 根据 `encoding` 编码字符串，默认 UTF-8
 - `bytes_obj.decode(encoding)` 根据 `encoding` 解码字节序列，解码失败会报错
 - `bytes("...", encoding)` 也是根据 `encoding` 编码字符串
 - 不要使用 `str(b"...")` 来将字节序列转为字符串

布尔类型

- `True` 和 `False`, 记住首字母大写
- 用 `bool(...)` 来转换, 如果是数字则非零都是 `True`, 如果是字符串则非空都是 `True`
- 运算
 - 可以使用 `&` | 来表示与和或 (但并不会短路)
 - 一般使用 `and` `or` `not` 进行与/或/非运算 (会短路)

注释

- 单行注释一个 `#`
- 严格上并没有多行注释, 但可以用 `"""` 字符串来代替 (因为这样不影响运行)

列表

- 类似其它语言的数组，但是功能更多，而且内部元素不要求同一类型
- 方括号 `[]` 表示列表，元素用逗号分隔
- 索引（即下标）从 `0` 开始计数，`lst[n]` 即表示访问第 `n+1` 个元素
- 索引可以是负数，负数即表示倒数，例 `lst[-2]` 表示倒数第二个元素
- 切片（获取列表中的一部分值）
 - `lst[a:b]`: 从 `lst[a]` 到 `lst[b-1]` 的列表
 - `lst[:b]`: 从开头到 `lst[b-1]` 的列表
 - `lst[a:]`: 从 `lst[a]` 到结尾的列表
 - `lst[:]`: 表示整个列表（拷贝一份）
 - `lst[a:b:c]`: 从 `lst[a]` 到 `lst[b-1]` 每 `c` 个（即步长）取一个形成的列表
 - `c` 可以是负数，此时需要 `a > b` 才能获取到值
 - 有步长时若省略 `a`、`b` 记得不要省略冒号，例 `lst[::-1]` 表示列表倒序

列表操作

- 修改元素：直接通过索引/切片，然后等号赋值
- 有栈的功能
 - `lst.append(...)` 在列表末尾加入元素
 - `lst.pop()` 弹出列表末尾元素并返回
- 任意位置插入弹出
 - `lst.insert(i, x)` 在索引 `i` 的位置插入 `x`, 后面依次后移
 - `lst.pop(i)` 弹出索引 `i` 位置的元素, 后面依次前移
- 列表拼接
 - 直接相加，不改变原列表，得到新的列表
 - `lst.extend([...])`, 把一个列表接到当前列表后面
- 根据值删除元素
 - `lst.remove(value)` 删除第一个值为 `value` 的元素

列表操作

- 排序列表
 - `lst.sort()` 永久排序（即排序后赋值给当前列表）
 - `sorted(lst)` 临时排序，返回排序好的新列表
 - 默认从小到大，如果传入 `reverse=True` 则从大到小
- 反转列表
 - `lst.reverse()` 永久反转（意义同上）
 - `lst[::-1]` 返回反转的列表（利用前面说到的切片）
- 统计操作
 - `len(lst)` 得到列表的长度
 - `sum(lst)` 得到列表的元素和（本质上是将 `start` 参数和每个元素依次相加）
 - 可以传入 `start` 参数用来指定加和的起始值
 - `max(lst)` 得到列表中的最大值
 - `min(lst)` 得到列表中的最小值

元组

- 可以看成元素不可变的列表，内部也可以包含不同类型的元素
- 括号表示元组，内部元素间用逗号分隔
- 可以使用和列表一样的方法来读取元素，但并不能修改
- 当只有一个元素的时候要写成 `(a,)` 而不是 `(a)`（后者是单个值）
- 可以使用 `tuple(...)` 来将可迭代对象（列表、字符串等）转为元组
- 元组并不能保证元素完全不可变
 - 避免在元组中存放可变元素

集合

- 大括号括起来，内部元素间用逗号分隔，会自动去重
- 可用 `set(...)` 来将可迭代对象转为元组，自动去重
- 集合中不能包含列表等不可 `hash` 化的元素
- 修改
 - `s.add(...)` 来加入一个元素
 - `s.remove(...)` 删除一个元素，如果没有会抛出异常
 - `s.discard(...)` 来删除一个元素，如果没有则忽略
- 运算
 - `s1 & s2`、`s1 | s2`、`s1 - s2` 交集、并集、差集
 - `s1 ^ s2` 对称差集

字典

- 存储键值对，也是大括号括起来，不过逗号分隔的是键值对 `{key: value, ...}`
- `{}` 是空字典而不是空集合
- 通过 `d[key]` 来访问字典中 `key` 对应的值，可以读取、修改
- 添加键值对可以直接通过 `d[key] = value` 来进行
- 删除键值对可以直接 `del d[key]`
- 通过 `d[key]` 访问值时如果不存在 `key` 这个键会抛出异常
 - 通过 `d.get(key)` 来访问值时如果不存在则会返回 `None`
 - 使用 `d.get(key, default)` 如果没有 `key` 时会返回 `default` 值
- `d.update(d2)` 来用 `d2` 中的键值对更新 `d`

布尔表达式

- `==` 判断相等（相等则返回 `True`），`!=` 判断不等
- 使用 `and` `or` `not` 来进行布尔运算，必要时加括号保证优先级
- 数值比较大小 `<` `<=` `>` `>=`
- 判断元素是否在列表中
 - `value in lst`: 如果在则值为 `True`
 - `value not in lst`: 如果不在则为 `False` (判断是否不在)
- 判断键是否在字典中
 - `key in d`、`key not in d` 与列表同理

条件语句

- if-elif-else 结构 (不是 else if)
- elif、else 均可以省略
- 条件不需要加括号 (加了也没问题)
- condition 会被转换成 bool 类型然后判断
- 注意缩进
- 类三目运算符写法 a if condition else b
 - 类似其它语言中的 condition? a : b

```
if condition1:  
    ...  
elif condition2:  
    ...  
elif condition3:  
    ...  
else:  
    ...
```

缩进

- 缩进是 python 中很重要的东西，python 靠缩进来得到代码结构，而不是大括号
- 缩进可以使用空格或制表符
- 如果一些代码处于同一层缩进下，则属于同一个代码块
- 同一个代码块的缩进要统一
 - 不仅仅是看着像，要区分好空格与制表符
 - 4 个空格与一个显示宽度为 4 的制表符并不是同一缩进
- 一般使用 4 空格缩进，或者 1 制表符缩进
- 编辑器中按 Tab 打出的也不一定是制表符，要分清
- 缩进不正确会报 `IndentationError`，此时注意检查缩进

for 循环

- python 中的 for 循环并不像 c 中是指定一个变量的变化方式，而是从列表/元组/迭代器等可迭代对象中遍历值
- for 循环会产生一个用于循环的变量，这个变量在循环结束后并不会删除，而是保留最后一次的值
- 可以使用 range 来生成一串数字用来循环
 - range(a, b) 生成从 a 到 b-1 的连续整数
 - range(a, b, c) 以 c 为步长生成
 - range 得到的并不是列表，如果要用其生成列表要使用 list(range(...))

```
for value in lst:
```

```
...
```

```
for value in range(...):
```

```
...
```

for 循环遍历字典

- 有三种方法来遍历字典
- 在 `d.keys()` 中循环遍历所有键
- 在 `d.values()` 中循环遍历所有值
- 在 `d.items()` 中遍历键值对（需要解包）

```
for key in d.keys():
    ...
for value in d.values():
    ...
for item in d.items():
    ... # item 为一个元组
for key, value in d.items():
    ... # 将 item 解包
```

元素解包

- 赋值时等号左侧可以是用逗号分隔的多个值，这时会将右侧解包分别赋值给左侧的各个变量
- 右侧也可以是多个值（只要出现逗号就会视为一个元组）
 - 可以通过 `a, b = b, a` 实现元素交换
- 星号表达式
 - 可以用来在可迭代对象内部解包
 - 也可用来标记一个变量包含多个值
- `for` 循环可以解包

```
t = (1, 2, 3)
a, b, c = t # a = 1, b = 2, c = 3
t = (1, 2, (3, 4))
a, b, (c, d) = t # c = 3, d = 4

l = [1, 2, *[3, 4]] # [3, 4] 被解包
# l = [1, 2, 3, 4]
a, *b = [1, 2, 3, 4]
# a = 1, b = [2, 3, 4]

lst = [[1, 2], [3, 4]]
for a, b in lst:
    ... # 第一次循环 a, b 为 1, 2
    # 第二次循环 a, b 为 3, 4
```

for 循环技巧

- enumerate 计数
 - 可以指定初始值
- zip 同时循环多个可迭代对象
 - 循环次数为最短的对象的长度

```
for i, value in enumerate(lst):  
    ... # i 依次为 0, 1, 2, ....
```

```
for i, value in enumerate(lst, 1):  
    ... # i 依次为 1, 2, 3, ....
```

```
for a, b in zip(lst1, lst2):  
    ... # a 在 lst1 中循环  
    # b 在 lst2 中循环
```

列表推导

- 一种很方便的生成列表的方式
- 即在列表中包含循环，逐次记录循环前表达式的值
- 可以有多重循环，即生成笛卡尔积
- 可以包含条件，即在条件成立时才记录值
- 列表推导中的循环变量有局部作用域
 - 即在列表推导外不能访问循环变量

```
lst = []
for i in range(1, 10):
    lst.append(i**2)
# 等价于
lst = [i**2 for i in range(1, 10)]  
  
lst1 = [x*y for x in l1 for y in l2]  
  
lst2 = [... for ... in ... if ...]
```

生成元组/字典

- 可以使用和列表推导类似的方法生成元组和字典
- 生成元组的时候要用 `tuple()`
 - 只写 `()` 的话则只是生成器表达式
- 生成字典时循环前用 `:` 将键值隔开

```
tuple(i**2 for i in range(1, 10))

(i**2 for i in range(1, 10))
# ^ generator object

{a: b for a in ... for b in ... }
```

控制循环

- 和其它语言一样，在循环代码块中可以控制循环的进行
- `break` 立刻结束循环
- `continue` 立刻进行下一轮循环

while 循环

- `while` 循环即进行条件检查，如果为 `True` 则继续运行直到条件不满足停止

```
while condition:
```

```
    ...
```

函数定义

- 使用 `def` 关键字来定义函数
- 先函数名，然后括号列出参数，下面接代码块
- 使用 `return` 返回
 - 没有 `return` 运行到结尾，返回 `None`
 - 只有 `return`，返回 `None`
 - `return` 后接内容，返回内容
 - `return` 的值类型不要求一致
 - `return` 可以返回多个值（利用元组）

```
def func_name(arg1, arg2):  
    ...  
  
def func_name(arg1, arg2):  
    ...  
    return ...  
  
def func_name(arg1, arg2):  
    ...  
    return ..., ...
```

函数参数

- 括号中要列出参数名，供函数体内使用
- 可以在参数后接等号赋默认值
 - 使用默认值的参数在调用时可以不用传
- 利用 * 来接收任意多参数
 - 接收进来是一个元组
 - * 参数后面不能再有其它非关键字参数
- 利用 ** 来接收任意多关键字参数
 - 接收进来是一个字典

```
def func(arg1, arg2):
    ...
    ...

def func(arg1, arg2="..."): # 默认值
    ...
    ...

def func(arg1, *arg2): # 任意多参数
    ...
    ...

def func(arg1, **arg2): # 任意多关键字参数
    ...
    ...

def func(arg1, *arg2, **arg3):
    ... # *arg2 后可以加 **arg3
```

函数调用

- 通过 函数名(参数) 来调用函数，得到返回值
- 直接传参的话要将参数与定义对应上
- 通过关键字传参 (参数名) 可以打乱顺序
- 带有默认值的参数如果不传则使用默认值
- 如果读任意多关键字参数，则多余的读到字典中

```
def func(a, b):  
    ...  
  
func(1, 2) # a = 1, b = 2  
func(b=1, a=2) # a = 2, b = 1  
  
def func2(a, **b):  
    ...  
  
func2(a=1, b=2, c=3)  
# a = 1, b = {"b": 2, "c": 3}
```

引用变量

- python 中的变量都是引用的（这也就是为什么前面说不要将变量理解为盒子）
- 用 = 实际上是定义了一个别名
 - lst1 = lst2, 则 lst1 和 lst2 会同时变化（要用 [:] 创建副本）
 - 数值类型有优化，所以不会这样
 - == 检查值是否相等，is 检查值是否相同
 - 观察 pythontutor.com
- 函数参数传递只有“共享传参”一种形式（即传引用）
 - 可变变量（例如列表）在函数内部可以被改变
 - 避免向函数传递可变变量（列表可传入 [:] 创建的副本）

匿名函数

- 可以通过 `lambda` 表达式来定义匿名函数
- `lambda` 输入：输出表达式
- 可以有多个输入
- 可以将一个函数赋值给一个变量
- 避免用 `lambda` 赋值的形式定义函数
 - 例如 `__name__` 属性不会是函数名，而是 "`<lambda>`"

```
lambda a: a**2 + 2*a + 1  
(lambda a: a**2 + 2*a + 1)(2) # 9  
  
lambda a, b: a*2 + b  
  
f = lambda a: a**2 + 2*a + 1  
# 近似等价于  
def f(a):  
    return a**2 + 2*a + 1
```

用户输入

- 读取用户输入使用内置的 `input` 函数
- 函数参数为要显示的提示符，例如 `input("> ")`
- 函数的返回值为一个字符串
- 每次读入一行（即读到换行为止）

高阶函数用法

- 接收函数作为参数的函数被称为高阶函数
- 比较常用的有 `map`、`filter`

```
list(map(lambda x: x*2, [1, 2]))  
# [2, 4]  
list(filter(lambda x: x>1, [1, 2, 3]))  
# [2, 3]
```

类

- 类可以看成包含一些属性和方法的框架
- 根据类来创建对象 → 实例化
- 用 `class` 关键字来定义类
- 类中的函数 → 方法
 - 特殊方法 `__init__`, 在类实例化的时候会被自动调用
 - 其它一般的方法第一个参数都要为"self", 调用的时候会自动传入
- 直接写在类中的是属性, 也可以通过为 `self.<name>` 赋值的形式创建属性
- 用类似函数调用的形式实例化类, 参数为`__init__`方法的参数
- 直接通过 `.<method> .<attribute>` 的形式调用方法/获取属性

```
class ClassName():
    a = 1
    def __init__(self, arg1, arg2):
        self.arg1 = arg1
        self.arg2 = arg2
    def method(self):
        print(self.arg1, self.arg2, self.a)

obj = ClassName(2, 3)
obj.method() # 2 3 1
print(obj.a, obj.arg1) # 1 2
```

类的继承

- 在 `class` 定义的括号中加上另一个类名则表示继承自那个类定义一个子类
- 子类会继承父类的所有属性和方法
- 子类编写和父类名字一样的方法会重载
- 在重载的方法中调用父类的原方法使用 `super()`
- 也可以为子类定义独有的方法

```
class ClassA():
    def __init__(self, a):
        self.a = a
    def print(self):
        print(self.a)

class ClassB(ClassA):
    def __init__(self, a):
        super().__init__(a)
        self.a *= 2

obj = ClassB(1)
obj.print() # 2
```

私有？

- python 中类并没有严格私有的属性
- 双下划线开头的属性会被隐藏，不能直接读取
- 但这种属性可以通过 `_类名__属性` 但方式读取到
- 使用双下划线开头的属性可以轻微保护属性，但并不代表其是私有的

```
class A():
    a = 1
    _a = 2
    __a = 3

obj = A()
print(obj.a) # 1
print(obj._a) # 2
print(obj.__a) # AttributeError
print(obj._A__a) # 3
```

一切皆对象？

- python 中即使最简单的整数也是一个类的实例
- 通过 `dir(...)` 查看一个对象的所有属性/方法
- 有很多双下划线开头、双下划线结尾的方法，成为魔术方法（dunder method）

魔术方法

- 很多函数、表达式其实是通过调用类的魔术方法来实现的
- `len(obj)` 调用 `obj.__len__()`
- `obj[...]` 调用 `obj.__getitem__(...)`
- `a in obj` 调用 `obj.__contains__(a)`
- `bool(obj)` 调用 `obj.__bool__()`
- 函数的调用本质上是调用 `func.__call__()`
- `a + b` 调用 `a.__add__(b)`
-

一个例子：

- `lst[a:b:c]` 切片操作
- 其中切片也是一个对象，它是一个 `slice` 类的实例
 - 所以它等价于 `lst[slice(a, b, c)]`
- 而通过 `[]` 的操作又是通过 `__getitem__` 魔术方法实现的
 - 所以它又等价于
`lst.__getitem__(slice(a, b, c))`
- `__getitem__` 方法中处理了 `slice`, 读取 `abc` 的值, 再处理返回一个新列表

就这？

- python 中类还有更多更多更好玩的用法
- 静态方法、类方法.....
- 多重继承、mro 顺序.....
- 接口协议、鸭子类型、抽象基类.....
- 猴子补丁.....
- 元类.....
- 垃圾回收.....
-

文件操作

- `open` 函数, 传入文件名、打开模式
- 打开模式（可以叠加）：`r` 读（默认）、`w` 写、`x` 创建并写、`a` 写在末尾、`b` 字节模式、`t` 文本模式（默认）
- 读取
 - 文本模式建议加上 `encoding`, 不然容易报错
 - `f.read()` 读取全部内容（字节模式得到字节序列）
 - `f.readline()` 读取一行
 - `f.readlines()` 读取所有行, 返回一个列表
- 写入
 - 文本模式同样建议加上 `encoding`
 - `f.write(...)` 直接写入
 - `f.writelines(...)` 传入列表, 元素间换行写入
- 通过这种形式操作文件记得用完后要 `f.close()`

```
f = open("filename", "r", encoding="utf-8")
s = f.read()                      # a str
# line = f.readline()    # a str
# lines = f.readlines() # a list
...
f.close()

f = open("filename", "w", encoding="utf-8")
f.write("...")
f.writelines(["...", "..."])
...
f.close()
```

with 块

- `with ... as ...`: 开启一个上下文管理器
- 常用在文件 `open` 上
 - `with` 块开始自动打开
 - `with` 块结束自动结束
- `with` 块结束后变量仍会留存

```
with open("file", "r", encoding="utf-8") as f:  
    s = f.read()  
    ...  
print(f.closed) # True
```

异常与处理

- 产生错误 → 抛出异常 → 程序结束
- `raise` 关键字抛出异常
- `try-except` 块捕获异常
 - 可以有多个 `except`、不可以没有
 - `except` 后接异常类（没有则捕获所有）
 - `as` 字句存下异常
- `finally` 语句
 - 不管是否有异常都会运行

```
raise ...
raise RuntimeError("...")

try:
    input("">>>> ")
except KeyboardInterrupt:
    print("Good bye")

try:
    print(1 / 0)
except ZeroDivisionError as e:
    print("can't devide by zero")
    raise e
finally:
    print("finished")
```

if 外的 else 语句

- else 块不仅仅跟着 if 才能使用
- for-else
 - for 循环结束才会运行
 - for 循环被 break 了不会运行
- while-else
 - condition 不成立退出才会运行
 - 循环被 break 终止了不会运行
- try-else
 - try 块中没有异常出现才会运行
 - else 块中异常不会被前面的 except 捕获
- 程序流跳到块外了不会运行 (return 等)

```
for value in lst:  
    ...  
else:  
    ...  
  
while condition:  
    ...  
else:  
    ...  
  
try:  
    ...  
except ...:  
    ...  
else:  
    ...
```

模块与导入

- 模块可以是一个单独的 .py 文件，也可以是一个文件夹
 - 文件夹相当于导入其下 `__init__.py` 文件
- 模块中正常编写函数、类、语句
- 通过 `import` 语句导入模块
 - `import code`
 - `import code as cd`
 - `from code import ...`
 - `from code import *`
- 导入时相当于运行了一遍导入的代码

```
# code.py
print("hello")
def f():
    print("call func in code.py")
...

import code # hello
code.f()
import code as cd # hello
cd.f()
from code import f # hello
f()
from code import * # hello
f()
```

“main 函数”

- 防止导入时运行代码
- 只允许直接运行脚本时运行
- 通过判断 `__name__`
 - 如果是直接运行，则其等于字符串 `__main__`
 - 如果是被导入的，则其等于模块名

```
# code.py
...
if __name__ == "__main__":
    print("hello")
else:
    print(__name__)
```

```
import code # code
```

```
$ python code.py # hello
```

内部模块

- python 自带了很多实用的模块（标准库）
 - os、sys: 系统操作
 - math: 数学运算
 - re: 正则表达式
 - datetime: 日期与时间
 - subprocess: 子进程管理
 - argparse: 命令行参数解析
 - logging: 日志记录
 - hashlib: 哈希计算
 - random: 随机数
 - csv、json: 数据格式解析
 - collections: 更多类型
 - ...
-
- 看文档: docs.python.org/zh-cn/3/library

外部模块安装

- pypi.org 上有极多别人写好了可以用的模块
 - numpy 矩阵等科学计算、scipy 科学计算、matplotlib 作图.....
- 使用 pip 安装 (pip / python -m pip)
 - pip install pkg_name
 - pip install pkg_name=... 指定版本
 - pip install -r requirements.txt 安装 txt 文件中的所有包
 - pip install ... -i <https://pypi.tuna.tsinghua.edu.cn/simple> 换源
 - pip list、pip show 命令查看安装的所有包/某个包的信息
 - pip uninstall pkg_name 卸载包
- pip 安装本地模块
 - 目录下需要包含 setup.py / pyproject.toml
 - pip install . 安装本地模块（复制到 site-packages 中）
 - pip install -e . 可修改形式安装本地模块（在当前位置，可以直接修改代码）

文档字符串

- 模块开头的三引号字符串
- 类、函数定义下面的三引号字符串
- `help(...)` 的时候可以显示
- `obj.__doc__` 表示这串字符串
- 编辑器用来提示
- 一些文档生成工具（sphinx 等）从中获取文档

```
"""
docstring for module
"""

def func(...):
    """docstring for function"""
    ...

class A():
    """docstring for class"""
    def __init__(self, ...):
        """docstring for method"""
    ...

```

代码规范

- PEP: Python Enhancement Proposals: peps.python.org
- PEP 8 规范, 给出了推荐使用的 python 代码风格规范
 - peps.python.org/pep-0008
 - pep8.org
- 更细致的代码风格
 - [black github.com/psf/black](https://github.com/psf/black)
 - [flake8 flake8.pycqa.org](https://flake8.pycqa.org)
 - ...

更多？

- 加群: [995146332](#)
- 基础
 - 《Python 编程：从入门到实践》 ISBN 978-7-115-54608-1
 - Python 3 菜鸟教程 runoob.com/python3/python3-tutorial.html
 - Python 官方文档 docs.python.org/3/tutorial
 - 中国大学 mooc - 浙江大学 Python 程序设计
- 进阶
 - 《流畅的 Python》 ISBN 978-7-115-45415-7
 - Python 官方文档 docs.python.org/3
 - 学一些实用的第三方库，看文档
 - PEP peps.python.org (注意分清有没有实施)
 - GitHub 找项目读

结

T H A N K S F O R W A T C H I N G