[ ]:
```python
#Open a File

# 'r' - only-read-mode
# 'w'- only-write-mode
# 'a' - 文件存在时，添加到末尾；文件不存在，创建新文件
# 'x' - 创建新文件，文件存在时失败

# 常用指令:
# 1. with-as order
with open('example.txt', 'r', encoding = 'utf-8') as file1: #优点-不需要关闭文件,命令结束自动关闭
    # doing something

# 2.read 命令
line = file1.readline(size) # 每次调用会读取下一行，适合逐行读取，并返回字符串。带有\n
                            # size （假设为4）不是指行数，而是最多读取的字节数，意思是读取这一行的4个字节
                            # 不到4个字节的，返回整行，知道遇到换行符或文件结束

lines = file1.readlines() #读取所有行，并返回列表，将每一行作为一个列表元素，包含换行符\n=
for line in lines:
    print(line.strip()) #print 每一行，处理换行符

content = file1.read() #读取整个文件内容,包括换行符\n，并返回字符串
                       #适合小文件
# 3. 指针位置

file1.seek()    # 跳转指针位置
file1.tell()    #非常有用，查看指针当前位置

file1.seek(0)  # 跳回开头，常用命令
```

[ ]:
```python
# CSV files

# 开头出现'\ufeff',属于编码错误, encoding =  'utf-8-sig'

import csv

fieldnames = [] #hardcoding key names
# 灵活获取fieldnames
fieldnames = set()
for d in lst:
    fieldnames.update(d.keys())  # 直接把每个字典的 key 加进去

# 或者
for d in lst:
    for key in d.keys():
        fieldnames.add(key)

with open('example.csv','r', newline = '', encoding = 'utf-8') as csvfile:
    reader = csv.reader(csvfile)  #按行读，每行是个列表
    csv.DictReader() #   每行自动变成字典

    writer = csv.writer(csvfile, fieldnames = fieldnames)  #把列表写成 CSV 格式
    csv.DictWriter()  #   把字典写成 CSV 格式


    #写入表头
    writer.writeheader()

    #writer 即上面命名的变量名字
    writer.writerows() #  写入多行数据(plural) write a list of lists
    writer.writerow() # 单行数据(singular)

    # csv也能用with open-as
    # 也能用readlines()

    contents = csvfile.readlines()

    # 打印的时候会多出行
    # 逐行打印可以
    for row in contents:
        print(row.strip()) #去除空行
```

[ ]:
```python
# JSON files(dicts or lists)
# ------------------------------------------------
```

```
# JSON files(dicts or lists)
# ------------------------------------------------
# JSON 中                    Python 中
# 对象（object）             字典（dict）
# 数组（array）              列表（list）
# 字符串（string）           字符串（str）
# 数字（number）             整数、浮点数（int, float）
# 布尔（true/false）         True/False
# null                       None
# ------------------------------------------------

import json

#Read
with open('data.json','r',encoding = 'utf-8') as jsonfile:
    jdata = json.load(jsonfile) # 推荐: 从文件读取json文件

    jstr = jsonfile.read()        # 读取文件内容,返回字符串
    jdata = json.loads(jstr)      # 从字符串中解析json,并非文件,输入类型(type)为str
    #两条命令组合
    jdata = json.loads(jsonfile.read())

#Write 'w' or  'a' mode
with open('data.json','w',encoding = 'utf-8') as file:
    json.dump(data,file, ensure_ascii = False, indent = 4) # 直接写入,最常用
    #ensure_ascii = False 防止转义为 \uXXXX, 保证中文正确显示
    # indent = 4 格式化输出, 使用4空格缩进, 易读模式

    jstr = json.dumps(data, ensure_ascii=False, indent=4) #转为json 字符串
    file.write(jstr)

    dict.get(key, default = None)# 查找key,存在时返回value值,不存在时默认返回None
                                 #不存在的默认返回值可以设置,即default
                                 #key不存在时不会抛出异常(KeyError)
```

```
# NumPy 核心-多维数组
 # NaN-safe意思是：  (Not a Number)
 #   遇到 NaN 时，不会报错
 #   不会把 NaN 当做正常值参与运算
 #   会自动忽略它们

import numpy as np

np.zeros((2, 3)) #全是0的数组，2行3列 同样可以指定int/float 默认type 为 float
np.ones((3, 3)) #全是1的数组，3行3列

a2d = np.array() # 创建数组 同样可以指定int/float/str(unicode = U3)
rows, columns = a2d.shape #获取行列
size = a2d.size #获取所有的数的数量


np.arange(start, stop, step, dtype = int)    # 生成数组 (array range)
                                             #dtype = int / float
                                             #使用方法近似range() 命令同样类似[:]
                                             # 多维数组切片(slicing).array[:2, :3] 为前2行,前3列
                                             # array[::-1, ::-1] 每行每列都相反
np.arange().reshape(rows, columns)    # 多维数组
np.reshape(array, new_shape)
np.concatenate([])                           # 使用[] 或tuple()传入多个数组 在非拼接轴上长度需要一致
np.vstack()  #rows
np.hstack()  #columns

np.split(array, 3) #切片
np.vsplit()    #rows
np.hsplit()    #columns

np.copy()
array.copy()
```

```
# 基础计算都不会修改源列表,而是返回一个新列表
# 注意, 复合计算, 如 arr *= 2 是直接修改原数组
# 括号中带out =a 也是直接修改原数组
# .sort()也是修改原数组,返回None!!! 跟np.sort()不一样

# 基础计算
np.add              # +
np.subtract         # -
np.negative         #- 负值
np.multiply         # *
np.divide           # /
np.power            # 指数运算
np.floor_divide     #整除
np.mod              #取余数

# 其余函数功能 [Nan-safe 基础命令+nan 如：np.sum() →np.nansum()]

axis = 0            # ↓ axis=0  ##按照列操作 columns
axis = 1            # → axis=1 ##按照行操作 rows

np.sum()            #求和，axis = 1 & 0 控制如何求和
np.mean()           #取平均值 axis = None（默认）对整个数组求平均
np.median()         #取中位数
np.prod()           #求积
np.std()            #标准差

np.max()            #最大值
np.argmax()         #最大值的位置 index
np.min()
np.argmin()

np.all()            #所有元素为True则返回True 简单视为 and 判断
                    # True 非空/非零 是为 True；False/0/空视为False
                    #可以逐 行/列 判断 axis = 1 / 0
np.any()            #至少一个为True 返回True 简单视为 or 判断

np.sort()           #排序 返回值：原数组不变
#注意
arr.sort()          #直接修改原数组,跟np.sort()不一样

np.random           #random功能跟基础库的random命令一致
```

```
# Refactoring 重构 优化代码

Pattern #1 - Use readable variable and function names

Pattern #2 - Use boolean values directly

Pattern #3 - Collapse nested 'if' statements into one

Pattern #4 - Look for and remove redundant statements

Pattern #5 - Use list comprehensions

Pattern #6 - Use any() instead of a loop
# any() 是 Python 中的一个内建函数，用来检查可迭代对象中是否有一个元素为真（True），只要有一个为真，它就返回 True，否则返回 False。

Pattern #7 - Replace a manual loop counter with enumerate()
# 遍历可迭代对象（如列表）时同时获得元素的索引和值，非常适合需要下标的 for 循环。
fruits = ['apple', 'banana', 'cherry']

for index, fruit in enumerate(fruits):
    print(index, fruit)

Pattern #8 - Move widely duplicated code to functions

Pattern #9 - Use default values on functions where appropriate

Pattern #10 - Reduce logical expressions to their simplest form
```

```
# TDD
# test case
def test_function():
    try:

    except:

    # return True or False
```