

# Introduction to Computation

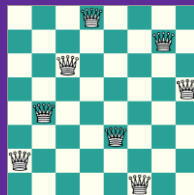
Autumn, 2023

Prof. Fan Cheng

Shanghai Jiao Tong University

chengfan85@gmail.com

<https://github.com/ichengfan/itc>



12

# Outline

- Random number
- File processing
- List comprehension
- Functional Programming

---

# Random



# Random number



“God does not play **dice** with the universe.” -- Einstein

- To play a game of chance where the computer needs to throw some dice, pick a number, or flip a coin
- To shuffle a deck of playing cards randomly
- To allow/make an enemy spaceship appear at a random location and start shooting at the player
- To simulate possible rainfall when we make a computerized model for estimating the environmental impact of building a dam
- For encrypting banking sessions on the Internet
- **Integers**: uniform distributed, random permutation, random sampling
- **Real numbers**: uniform, Gaussian

```
import random # py module
```

# Random integer

- `random.randint(a, b)`: Return a random integer N such that  $a \leq N \leq b$

```
import random

print("You lucky number is {}".format(random.randint(1,6)))

print("You first three dices are {}, {}, {}".format(random.randint(1,6),random.randint(1,6),random.randint(1,6)))
```

```
You lucky number is 2
You first three dices are 6, 2, 4.
```

```
import random
def test_uniform(rounds):
    li = [0]*6
    for i in range(rounds):
        li[random.randint(1, 6)-1] += 1

    for i in range(6):
        li[i] /= rounds

    return li
```

```
rounds = 6
print(test_uniform(rounds))
rounds = 60
print(test_uniform(rounds))
rounds = 600
print(test_uniform(rounds))
rounds = 6000
print(test_uniform(rounds))
rounds = 60000
print(test_uniform(rounds))
rounds = 600000
print(test_uniform(rounds))
```

```
[0.0, 0.16666666666666666, 0.16666666666666666, 0.16666666666666666, 0.3333333333333333, 0.16666666666666666]
[0.2, 0.16666666666666666, 0.15, 0.2, 0.15, 0.13333333333333333]
[0.17, 0.18666666666666668, 0.17, 0.145, 0.185, 0.14333333333333334]
[0.16616666666666666, 0.1695, 0.1695, 0.16816666666666666, 0.15933333333333333, 0.16733333333333333]
[0.16603333333333334, 0.16655, 0.16588333333333333, 0.1679, 0.16505, 0.16858333333333334]
[0.16646833333333333, 0.16722333333333333, 0.16558, 0.16685, 0.1667, 0.16717833333333335]
```

As rounds grows, the probability goes to  $1/6 \sim 0.166$

```
rounds = [6, 60, 600, 6000, 60000, 600000]
for r in rounds:
    print(test_uniform(r))
```

# Random integer(cont'd)

- `random.randrange(start, stop, step):`
  - Return a randomly selected element from `range(start, stop, step)`
    - `random.randrange(stop)`
- Recall: `range(a, b) = [a, b)` (b is not included!)
- `Random.randint(a, b) = random.randrange(a, b+1)`

Implement the following functions (the same with the previous lectures)

1. Roll the dice to get your lucky number
2. Roll three dices to see your luckiness
3. Roll the dices to validate that the dice is uniformly handcrafted

# Random floating number

- <https://docs.python.org/3/library/random.html>
- `random.random()` Return the next random floating point number in the range [0.0, 1.0).
- `random.uniform(a, b)` Return a random floating point number N such that  $a \leq N \leq b$  for  $a \leq b$  and  $b \leq N \leq a$  for  $b < a$ .
- The end-point value b may or may not be included in the range depending on floating-point rounding in the equation  $a + (b-a) * \text{random}()$ .

Exercise: Compute Pi = 3.1415926≈3.14

```
def inside(x,y):  
    return x*x + y*y <= 1  
  
n = 1000000  
m = 0  
for _ in range(n):  
    x = random.uniform(-1,1)  
    y = random.uniform(-1,1)  
    if inside(x,y):  
        m += 1  
  
print("PI is: {}".format(4*m/n))
```

PI is: 3.1413432

$$0 \sqrt{\frac{1}{n}}$$

$$\pi \approx \sqrt{7 + \sqrt{6 + \sqrt{5}}}$$

# Seed (随机数种子)

- The `random.seed()` method is used to initialize the random number generator
- The random number generator needs a number to start with (a seed value), to be able to generate a random number
- By default the random number generator uses the current system time
- Use the `seed()` method to customize the start number of the random number generator
- If you use the same seed value twice you will get the same random number twice

```
import random

def test_random_seed(n):
    random.seed(10)
    lst = []
    for i in range(n):
        lst.append(random.random())

    return lst

for i in range(10):
    print(test_random_seed(4))
```

```
[0.5714025946899135, 0.4288890546751146, 0.5780913011344704, 0.20609823213950174]
[0.5714025946899135, 0.4288890546751146, 0.5780913011344704, 0.20609823213950174]
[0.5714025946899135, 0.4288890546751146, 0.5780913011344704, 0.20609823213950174]
[0.5714025946899135, 0.4288890546751146, 0.5780913011344704, 0.20609823213950174]
[0.5714025946899135, 0.4288890546751146, 0.5780913011344704, 0.20609823213950174]
[0.5714025946899135, 0.4288890546751146, 0.5780913011344704, 0.20609823213950174]
[0.5714025946899135, 0.4288890546751146, 0.5780913011344704, 0.20609823213950174]
[0.5714025946899135, 0.4288890546751146, 0.5780913011344704, 0.20609823213950174]
[0.5714025946899135, 0.4288890546751146, 0.5780913011344704, 0.20609823213950174]
[0.5714025946899135, 0.4288890546751146, 0.5780913011344704, 0.20609823213950174]
```

仿真实验需要



# Random: choice(), shuffle(), sample()

- `random.choice()`: Return a random element from the non-empty sequence `seq`
- `random.choices(population, weights=None, *, cum_weights=None, k=1)`: Return a `k` sized list of elements chosen from the population with replacement.
- `random.shuffle()`: Shuffle the sequence `x` in place
- `random.sample()`: Return a `k` length list of unique elements chosen from the population sequence or set.
- <https://docs.python.org/3/library/random.html>

```
import random

lst = list(range(7))
random.shuffle(lst)
print(lst)

lst = list(range(2020))
print(random.choice(lst))

spl = random.sample(lst, k = 4)
print(spl)
```

```
[1, 6, 5, 0, 2, 4, 3]
113
[340, 790, 71, 703]
```

```
[5, 3, 4, 2, 1, 0, 6]
3
[1148, 36, 358, 911]
```

# File processing



# Shell

## 绝对路径 VS 相对路径

Shell中，路径名有空格用” xxx ” 处理

- In powershell (windows) or shell (linux, mac), “cd” means **change directory**
- current working directory (CWD): 当前工作目录
- An **absolute path**, which always begins with the root folder (C:\windows\...\a.txt)
- A **relative path**, which is relative to the program's current working directory (\a\b.txt)
  - . : current directory. “./a.py”
  - .. : the parent directory. “cd ..” 返回上级目录
- Run python program via shell: **python filename.py**
  - filename.py should be in the current working directory (当前工作目录)
  - Or python C:\xxxx\...\filename.py (绝对路径)

```
1 import b # File a.py
2 import sys
3
4 for x in sys.argv:
5     print(x)
6
7 b.spam("Test")
8
9 print("Hell world")
10
11 print("Test for loop")
12
13 for x in range(3):
14     print(x)
```

```
PS C:\Users\fcheng\OneDrive\CS124计算导论\2020 秋季\lecture notes> python a.py
a.py
Test spam
Hell world
Test for loop
0
1
2
```

```
PS C:\Users\fcheng\OneDrive\CS124计算导论\2020 秋季> python a.py
C:\Program Files (x86)\Python36-32\python.exe: can't open file 'a.py': [Errno 2] No such file or directory
```

```
PS C:\Users\fcheng\OneDrive\CS124计算导论\2020 秋季> python C:\Users\fcheng\OneDrive\CS124计算导论\2020 秋季\lecture notes\
a.py
Test spam
Hell world
Test for loop
0
1
2
```

# Shell: parameter

VS Code: Ctrl+F5

与Code Runner的默认配置不一样

- Run python program via shell with parameters:

`python filename.py parameter1 parameter2 parameter3 ...`

```
PS C:\Users\fcheng\OneDrive\CS124计算导论\2020 秋季\lecture notes> python a.py 1 2 3 4 5
a.py
1
2
3
4
5
Test spam
Hell world
Test for loop
0
1
2
PS C:\Users\fcheng\OneDrive\CS124计算导论\2020 秋季\lecture notes> python a.py "Hello" "world" "荷利"
a.py
Hello
world
荷利
Test spam
Hell world
Test for loop
0
1
2
```

# Ubuntu

VS Code: Ctrl+F5  
与Code Runner的默认配置不一样

## 计算机方向的本科生一定要尽早接触Linux

- <https://ubuntu.com/>
- 练习用它来替代windows做一切事情: 写报告, 写代码等等
  - C++
  - Python
  - Shell
  - Office等等
- 开源: 没什么比源码更重要了
- 可以在windows下面装: 虚拟机+ubuntu
- Mac和Linux用起来差距不大

打开一个新世界  
自己寒假折腾



# Module: sys

<https://docs.python.org/3/library/sys.html>  
[https://www.python-course.eu/sys\\_module.php](https://www.python-course.eu/sys_module.php)

- System-specific parameters and functions: This module provides access to some **variables** used or maintained by **the interpreter** and to functions that interact strongly with the interpreter. It is always available
- Manipulate different parts of the Python runtime environment
- **sys.version**: This attribute displays a string containing the version number of the current Python interpreter
- **sys.argv**: `sys.argv` returns a list of command line arguments passed to a Python script.
- **sys.path**: This is an environment variable that is a search path for all Python modules.
  - **sys.path[0]**: current directory of the file
- **sys.exit**: This causes the script to exit back to either the Python console or the command prompt. This is generally used to safely exit from the program in case of generation of an exception.
- **sys.maxsize**: Returns the largest integer a variable can take.
- **sys.stdin, sys.stdout and sys.stderr**: Standard data streams

```
import sys
print(sys.version)
print(sys.argv)
print(sys.path)
print(sys.maxsize, 2**63)
print(sys.stdin)
print(sys.stdout)
print(sys.stderr)
sys.exit()
print("Byebye")
```

```
3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC v.1927 64 bit (AMD64)]
['C:\\Users\\fcheng\\OneDrive\\CS124计算导论\\2020 秋季\\lecture notes\\course_code.py']
['C:\\Users\\fcheng\\OneDrive\\CS124计算导论\\2020 秋季\\lecture notes',
'C:\\Users\\fcheng\\AppData\\Local\\Programs\\Python\\Python38\\python38.zip',
'C:\\Users\\fcheng\\AppData\\Local\\Programs\\Python\\Python38\\DLLs',
'C:\\Users\\fcheng\\AppData\\Local\\Programs\\Python\\Python38\\lib',
'C:\\Users\\fcheng\\AppData\\Local\\Programs\\Python\\Python38',
'C:\\Users\\fcheng\\AppData\\Roaming\\Python\\Python38\\site-packages',
'C:\\Users\\fcheng\\AppData\\Roaming\\Python\\Python38\\site-packages\\win32',
'C:\\Users\\fcheng\\AppData\\Roaming\\Python\\Python38\\site-packages\\win32\\lib',
'C:\\Users\\fcheng\\AppData\\Roaming\\Python\\Python38\\site-packages\\Pythonwin',
'C:\\Users\\fcheng\\AppData\\Local\\Programs\\Python\\Python38\\lib\\site-packages']
9223372036854775807 9223372036854775808
<_io.TextIOWrapper name='<stdin>' mode='r' encoding='utf-8'>
<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'>
[Finished in 0.1s]
```

# Module: sys.argv

<https://docs.python.org/3/library/sys.html>  
[https://www.python-course.eu/sys\\_module.php](https://www.python-course.eu/sys_module.php)

- **sys.argv**: sys.argv returns a list of command line arguments passed to a Python script. sys.argv[0] contains the script file name xxx.py; sys.argv[1] contains the first parameter , while sys.argv[2] contains the second parameter

```
>>>python course_code.py a b c d
```

```
>>>['course_code.py', 'a', 'b', 'c', 'd']
```

```
1 import b # File a.py
2 import sys
3
4 for x in sys.argv:
5     print(x)
6
7 b.spam("Test")
8
9 print("Hell world")
10
11 print("Test for loop")
12
13 for x in range(3):
14     print(x)
```



# Module: sys.path

- **sys.path**: This is an environment variable that is a **search path for all Python modules**.
  - A list of strings that specifies the search path for modules. Initialized from the environment variable PYTHONPATH, plus an installation-dependent default.
- **sys.path[0]**: **current directory of the file**
  - As initialized upon program startup, the first item of this list, path[0], is the directory containing the script that was used to invoke the Python interpreter. If the script directory is not available (e.g. if the interpreter is invoked interactively or if the script is read from standard input), path[0] is the empty string, which directs Python to search modules in the current directory first. Notice that the script directory is inserted before the entries inserted as a result of PYTHONPATH.
  - 绝对路径: sys.path[0] + 文件名

```
import sys
print(sys.version)
print(sys.version_info)
print(sys.argv)
print(sys.path)
print(sys.path[0])
```

```
3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
sys.version_info(major=3, minor=6, micro=2, releaselevel='final', serial=0)
['c:\\Users\\fcheng\\OneDrive\\CS124计算导论\\2020 秋季\\lecture notes\\course_code.py']
['c:\\Users\\fcheng\\OneDrive\\CS124计算导论\\2020 秋季\\lecture notes', 'C:\\Program Files (x86)\\Python36-32\\python36.zip', 'C:\\Program Files (x86)\\Python36-32\\DLLs', 'C:\\Program Files (x86)\\Python36-32\\lib', 'C:\\Program Files (x86)\\Python36-32', 'C:\\Users\\fcheng\\AppData\\Roaming\\Python\\Python36\\site-packages', 'C:\\Program Files (x86)\\Python36-32\\lib\\site-packages']
c:\\Users\\fcheng\\OneDrive\\CS124计算导论\\2020 秋季\\lecture notes
```

# Module: standard streams

- Every serious user of a UNIX or Linux operating system knows standard streams, i.e. input, standard output and standard error. They are known as pipes. They are commonly abbreviated as stdin, stdout, stderr.
- The standard input (stdin) is normally connected to the keyboard, while the standard error and standard output go to the terminal (or window) in which you are working.
- These data streams can be accessed from Python via the objects of the sys module with the same names, i.e. `sys.stdin`, `sys.stdout` and `sys.stderr`.

# Module: os 操作系统

[https://python101.pythonlibrary.org/chapter16\\_os.html](https://python101.pythonlibrary.org/chapter16_os.html)

- OS: Operating system. This module provides a portable way of using operating system dependent functionality.
  - <https://docs.python.org/3/library/os.html>
- os.name
- os.environ, os.getenv() and os.putenv(): 环境变量
- os.chdir() and os.getcwd(): 改变工作目录、查询当前目录
- os.mkdir() and os.makedirs(): 创建目录
- os.remove() and os.rmdir(): 删除文件、目录(当心重要文件!!!!!!)
- os.rename(src, dst): 重命名
- os.startfile(): 修改程序的打开方式
- os.walk(): 遍历一个路径下面所有目录和文件
- os.path
  - basename, dirname, exists, isdir and isfile, join, split

# Module: os

```
import os
import sys

print(os.name)

print(os.getcwd())
f = os.path.join('usr', 'bin', 'spam')
print(f)
l = os.path.split(f)
print(l)

print(os.path.getsize(sys.path[0]+'\\course_code.py'))
```

```
nt
C:\Users\popeC
usr\bin\spam
('usr\\bin', 'spam')
37796
```

<https://docs.python.org/3.3/library/os.path.html?highlight=path#module-os.path>

# Binary File

- 所有文件都是二进制存储的：二进制文件
- 把单个字节(Byte)的0/1串和人类可读的文字一一对应上来→ 文本文件
- 0/1串可以有任意多种文字的解读，ASCII只是其中一种

- Binary file are those typical files that store data in the form of sequence of bytes grouped into eight bits or sometimes sixteen bits. These bits represent custom data and such files can store multiple types of data (images, audio, text, etc) under a single file
- Binary file can have custom file formats and the developer, who designs these custom file formats, converts the information, to be stored, in bits and arranges these bits in binary file so that they are well understood by the supporting application and when needed, can easily be read by the supporting application
- One most common example of binary file is image file is .PNG or .JPG. If one tries open these files using a text editor then, he/she may get unrecognizable characters, but when opened using the supporting image viewer, the file will be shown as a single image. This is because the file is in binary format and contains data in the form of sequence of bytes. When the text editor tries to read these bytes and tries to convert bits into characters, they get undesired special characters and display it to the user

# Text File

- Text files **are special subset of binary files** that are used to store human readable characters as a rich text document or plain text document. Text files also store data in sequential bytes but bits in text file represents characters.
- Text files are less prone to get corrupted as any undesired change may just show up once the file is opened and then can easily be removed.
- Text files are of two types:
- Plain text files: These files store End of Line (EOL) marker at the end of each line to represent line break and an End of File (EOF) at the end of the file to represent end of file.
- Rich text files: These files also follow the same schema as the plain text files but may also store text related information like text colour, text style, font style etc.
- Because of simple and standard format to store data, text files are one of the most used file formats for storing textual data and are supported in many applications.
- 和文本的编码格式相关：UTF， GB2312等等

# bytes and unicode

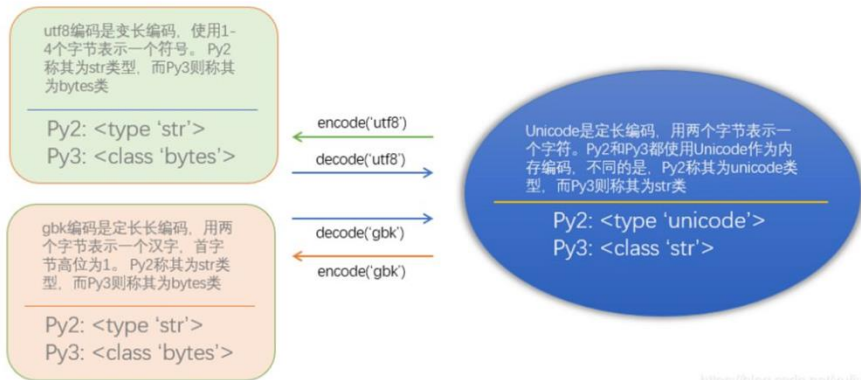
<https://blog.csdn.net/csdnnews/article/details/102852829>



<https://blog.csdn.net/csdnnews>

- 无论是PY2还是PY3，都使用Unicode作为内存编码，简称内码。
- 保存在Python解释器内存中的文本，输出到屏幕、编辑器，或者保存成文件的时候，都要将内码转换成UTF8或者GBK等编码格式；
- 同样，Python解释器从输入设备接收文本，或者从文件读取文本的时候，都要将UTF8或者GBK等编码转换成Unicode编码格式。因此，无论是PY2还是PY3，想要在Unicode、UTF8、GBK等编码格式之间转换的话，右上角的图是通用的：
- 我们之所以会产生困惑，是因为PY2和PY3给这些编码格式指定了令人困惑的名字。PY2的字符串有两种类型：Unicode类型和str类型。PY2的Unicode类型就是Unicode编码，PY2的str类型泛指除Unicode编码之外的所有编码，包括ASCII编码、UTF8编码、GBK编码、cp936编码等。PY3的字符串也有两种类型：bytes类型和str类型。PY3的str类型就是Unicode编码，PY3的bytes类型泛指除unicode编码之外的所有编码，包括ASCII编码、UTF8编码、GBK编码、CP936编码等。同样是str类型，在PY2和PY3中完全颠倒了！下图稍微补充了一点内容，更有助于理解编码问题。

# bytes and Unicode (cont'd)



<https://blog.csdn.net/u01ve>

```
110 print(type(b'hello world'))
111 print(type('hello world'))
112
113 zen = 'life is short, use python.--哲学'
114 b_zen = zen.encode()
115
116 print(b_zen, type(b_zen))
117
118 barr_zen = bytearray(zen.encode())
119 barr_zen[5]='-'.encode()[0]
120 print(barr_zen)
121
122 print(zen.encode('unicode-escape'))
123 print(zen.encode('utf8'))
124 print(zen.encode('gbk'))
```

```
<class 'bytes'>
<class 'str'>
b'life is short, use python.--\xe5\x93\xb2\xe5\xad\xa6' <class 'bytes'>
bytearray(b'life -s short, use python.--\xe5\x93\xb2\xe5\xad\xa6')
b'life is short, use python.--\u54f2\u5b66'
b'life is short, use python.--\xe5\x93\xb2\xe5\xad\xa6'
b'life is short, use python.--\xd5\xdc\xdl\xa7'
```



# bytes and unicode

- 1. python bytes 也称字节序列，并非字符。取值范围  $0 \leq \text{bytes} \leq 255$ ，输出的时候最前面会有字符b修饰；string 是python中字符串类型；
- 2. bytes主要是给在计算机看的，string主要是给人看的；
- 3. string经过编码encode，转化成二进制对象，给计算机识别；bytes经过解码decode，转化成string，让我们看，但是注意反编码的编码规则是有范围，\xc8就不是utf8识别的范围；

# bytes and unicode

- `bytes(string, encoding)`
- `str(bytes, encoding)`
- `bytes.decode()`
- `str.encode(bytes, encoding)`
- `byte`, `bytearray`: `bytearray` objects are a mutable counterpart to `bytes` objects

# Store data in file

- While a program is running, its data is stored in random access memory (RAM). RAM is fast and inexpensive, but when the program ends, or the computer shuts down, **data in RAM will be lost**
- To make data available the next time the computer is turned on and the program is started, it has to be written to a **non-volatile** storage medium, such a hard drive, usb drive, or CD-RW
- Data on non-volatile storage media is stored in named locations on the media called files
- Working with files is a lot like working with a notebook.
  - To use a notebook, it has to be opened.
  - When done, it has to be closed
- In python, a file object will be created to operate the file
  - open(filename, mode):** returns a file object, default mode is “read”
  - close():** free up any system resources taken up by the open file.

- **Reminder:** Computers have cache, memory, hard disk.
- **What are the major differences in speed and price?**

# open()

- Before you can read or write a file, you have to open it using Python's built-in `open()` function. This function creates a file object, which would be utilized to call other support methods associated with it.
- Syntax: `file object = open(file_name [, access_mode][, buffering])`
- Here are parameter details –
  - **file\_name** – The file\_name argument is a string value that contains the name of the file that you want to access.
  - **access\_mode** – The access\_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is an optional parameter and the default file access mode is read (r).
  - **buffering** – If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).
- Here is a list of the different modes of opening a file –
  - `r: read; w: write; b: binary; +: read and write; a: appending`
  - Modes can be mixed

Mode	Description
<b>r</b>	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
<b>rb</b>	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
<b>r+</b>	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
<b>rb+</b>	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
<b>w</b>	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
<b>wb</b>	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
<b>w+</b>	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
<b>wb+</b>	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
<b>a</b>	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
<b>ab</b>	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
<b>a+</b>	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
<b>ab+</b>	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

**w mode is extremely dangerous! It may destroy your data**

# File object

- Here is a list of all the attributes related to a file object –
  - **file.closed:** Returns true if file is closed, false otherwise
  - **file.mode:** Returns access mode with which file was opened
  - **file.name:** Returns name of the file
- Binary file
- Text file

```
f = open("file.txt", "wb")
print("Name of the file: {}".format(f.name))
print("Close or not: {}".format(f.closed))
print("Opening mode: {}".format(f.mode))
```

```
('Name of the file: ', 'file.txt')
('Closed or not : ', False)
('Opening mode : ', 'wb')
```

- 文件是一串0/1序列。文件操作就是将这串数列读入内存或者写到文件
- 文件**常见错误**
  - Python源文件和读写的文件不在同一个目录 (相对路径)
  - 当前VS Code的工作目录和读写文件不同，特别注意 (绝对路径或者改变cwd)

# Path (路径): pathlib

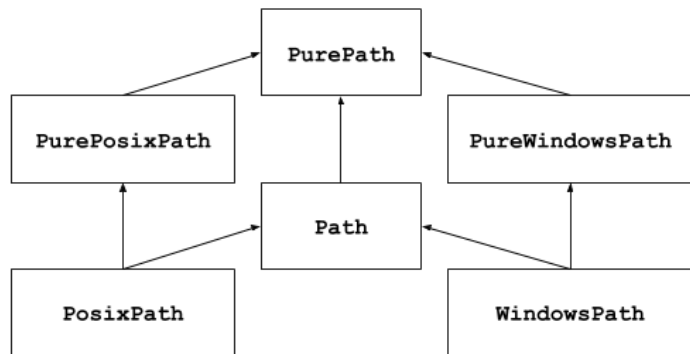
- Windows: C:\windows\xxxx\
- Mac and Linux: C:/usr/xxxxx
- 历史遗留问题: early 1980's computer history
- By hand: No!!!!!!
- os.path: You can use `os.path.join()` to build a path string using the right kind of slash for the current operating system
- **pathlib: python 3.4**
  - You should use forward slashes / with **pathlib functions**
  - The Path() object will convert forward slashes into the correct kind of slash for the current operating system. Nice!
  - If you want to add on to the path, you can use **the / operator** directly in your code. Say goodbye to typing out `os.path.join(a, b)` over and over

# Path (路径): pathlib

- pathlib — Object-oriented filesystem paths
- <https://docs.python.org/3/library/pathlib.html>
- This module offers classes representing filesystem paths with semantics appropriate for different operating systems. Path classes are divided between pure paths, which provide purely computational operations without I/O, and concrete paths, which inherit from pure paths but also provide I/O operations.
- **Path, PosixPath, WindowsPath**

推荐使用

所有相关方法





# Path (路径): pathlib

```
from pathlib import Path

data_folder = Path("source_data/text_files/")
file_to_open = data_folder / "raw_data.txt"
print(file_to_open)

filename = Path("source_data/text_files/raw_data.txt")
print(filename.name)
print(filename.suffix)
print(filename.stem)

data_folder = Path("source_data/text_files/")
file_to_open = data_folder / "raw_data.txt"

if not filename.exists():
    print("Oops, file doesn't exist!")
else:
    print("Yay, the file exists!")
```

直接用 “/”，不需要考虑OS平台

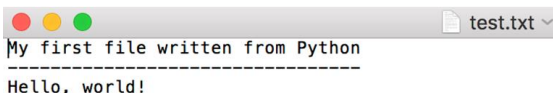
```
source_data\text_files\raw_data.txt
raw_data.txt
.txt
raw_data
Oops, file doesn't exist!
```

# Write into Files: write(), writelines()

When you open() a file, do remember to close() it

- A file object can call write() to write data into the file
- **f.write(string)** writes the contents of string to the file, returning the number of characters written. The write() method does not add a newline character ('\n') to the end of the string
- **f.writelines (lines)** Writes a list of lines to the file. Must be str. **No “\n” added**


```
myfile = open("test.txt", "w")
myfile.write("My first file written from Python\n")
myfile.write("-----\n")
myfile.write("Hello, world!\n")
myfile.close()
```



test.txt

My first file written from Python  
-----  
Hello, world!

```
res = ["123", "We", "US Election" ]
wf = open("wtest.txt", "w+")
wf.writelines(res)
wf.close()
```



1 123WeUS Election

- With mode “w” (means write)
  - if there is no file named “test.txt” on the disk, it will be created.
  - **If there already is one, it will be replaced by the file we are writing!!!!!!**
- **Always open() and close() in pair**

# Read from files: read(), readline(), readlines()

- There are three major methods to read from a file:
  - **readline()** method returns everything in “string” up to and including the newline character. ‘\n’ will be read in
  - input(): terminate when ‘\n’ is input and ‘\n’ is ignored
  - **readlines()**: Reads and returns a list of lines from the file.
  - **read([n])** methods read n bytes from the file. It will read the complete contents of the file into a string by default
- **It is simple to transform a string into a list and see its contents**

```
rf = open("test.txt", "r")
while True:
    line = rf.readline()
    print(line, end="")
    if len(line)==0:
        break
print()
rf.close()
```

```
My first file written from Python
-----
Hello, world!
```

```
f= open("test.txt", "r")
while True:
    line = f.readline()
    l = list(line)
    print(l)
    if len(l) == 0:
        break
f.close()
```

```
[['M', 'y', ' ', 'f', 'i', 'r', 's', 't', ' ', 'f', 'i', 'l', 'e', ' ', 'w', 'r', 'i', 't', 't', 'e', 'n', ' ', 'f', 'r', 'o', 'm', ' ', 'P', 'y', 't', 'h', 'o', 'n', '\n']]
[['H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', ' ', '\n']]
[]
```

# readlines() and EOF

- `readlines()`: Reads and returns a list of lines from the file.
- EOF: end of file. 文件结束标志

```
rf = open("test.txt")
res = rf.readlines()
rf.close()
print(res)
```

```
1 My first file written from Python
2 -----
3 Hello, world!
```

```
['My first file written from Python\n', '-----\n', 'Hello, world!']
```

```
1 My first file written from Python
2 -----
3 Hello, world!
4
```

```
['My first file written from Python\n', '-----\n', 'Hello, world!\n']
```

```
1 My first file written from Python
2 -----
3 Hello, world!
4
5
6
```

```
['My first file written from Python\n', '-----\n', 'Hello, world!\n', '\n', '\n']
```

# Binary files: read()

- `read([n])` methods read `n` bytes from the file. It will **read the complete contents** of the file into a string by default
- When we `read()` from the file we're going to get bytes back rather than a string.

```
f = open("somefile.txt")
content = f.read()
f.close()
words = content.split()
print("There are {0} words in the file.".format(len(words)))
```

```
f = open("somefile.zip", "rb")    # r: read; b: binary file
g = open("thecopy.zip", "wb")    # w: write; b: binary file
while True:
    buf = f.read(1024) # read 1024 bytes
    if len(buf) == 0:  # if buf is empty, then it is the end of the file.
        break
    g.write(buf)
g.close()
f.close()
```

# With statement

Forget `close()` is a common mistake in programming. The solution is to sidestep it

- Python's **with** statement provides a very convenient way of dealing with the situation where you have to do a setup and teardown to make something happen.
- It is good practice to use the **with** keyword when dealing with file objects. The advantage is that the file is properly closed after its suite finishes, even if an exception is raised at some point
- After a file object is closed, either by a **with** statement or by calling `f.close()`, attempts to use the file object will automatically fail

```
with open("test_with.txt", "w") as f:
    f.write("My first file written from Python\n")
    f.write("-----\n")
    f.write("Hello, world!\n")
```

```
1 with open('mylog.txt') as infile, open('a.out', 'w') as outfile:
2     pass
```

- You won't need to `close()` yourself. In fact, you cannot call `close()` now
- **with** will be used repeatedly in other modules later
- The syntax of the with statement now allows **multiple context managers** in a single statement:

- 一件事如果可能会做错，那么就一定会做错！
- 想不犯错？？ 不做 `open + close` → `with`