

# Introduction to Computation

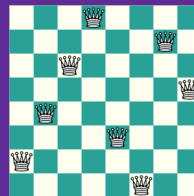
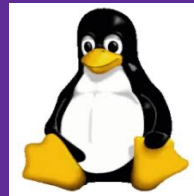
Autumn, 2023

Prof. Fan Cheng

Shanghai Jiao Tong University

chengfan85@gmail.com

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



12

# Outline

- Randomness
- Unicode
- File processing

---

# Randomness



# 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

# Random integer

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

```
1 import random
2
3 print(f"Your lucky number is {random.randint(1, 6)}")
4 print(
5     f"Your first three dices are {random.randint(1, 6)}, {random.randint(1, 6)}, {random.randint(1, 6)}"
6 )
```

```
1 import random
2
3
4 def test_uniform(rounds):
5     lst = [0] * 6
6     for i in range(rounds):
7         lst[random.randint(1, 6) - 1] += 1
8
9     for i in range(6):
10         lst[i] /= 6
11
12     return lst
13
14
15 for rounds in (6, 60, 600, 6000, 60000, 600000):
16     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  $1/6 \sim 0.166$

# 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)`

```
1 for i in range(10):  
2     print(random.randrange(1, 20, 3))
```

```
1 for i in range(10):  
2     print(random.random())
```

13	0.3744124851947369
7	0.9807194952065752
7	0.049552510842784336
19	0.16745393352679827
19	0.37586592846704303
7	0.6629287486194784
10	0.5294961001082706
16	0.23952881542060367
4	0.8805506089116676
10	0.7796646821567711

# Random floating number

- `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**  $\approx$  3.14

```
1 def inside(x, y):
2     return x * x + y * y <= 1
3
4
5 n = 100_000_000
6 m = 0
7 for _ in range(n):
8     x = random.uniform(-1, 1)
9     y = random.uniform(-1, 1)
10    if inside(x, y):
11        m += 1
12
13    print(f"PI is: {4 * m / n}")
```

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

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

# 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

```
1  import random
2
3  lst = list(range(7))
4  random.shuffle(lst)
5  print(lst)
6
7  lst = list(range(2020))
8  print(random.choice(lst))
9
10 spl = random.sample(lst, k=4)
11 print(spl)
```

```
[4, 5, 0, 1, 2, 3, 6]
671
[155, 511, 1950, 1951]
```

# Unicode



汝	新	漁	雞	通	追	𠂔
途	祭	福	祐	鳳	星	遠
𠂔	𠂔	𠂔	𠂔	𠂔	𠂔	𠂔
𠂔	𠂔	𠂔	𠂔	𠂔	𠂔	𠂔

# Information Coding Standard



- Computers can only “understand” binary information {0, 1}.
  - Everything inside is represented as a binary sequence.
  - A binary sequence is of no meaning unless we translate it to the human text
- In program, information could be stored at

## Memory and Files.

- Memory information will be erased after the process is finished, while files are permanent.
- Files are stored as 0, 1 sequences.
- In principle, we have infinitely many explanations a sequence like 0011011.
- We need to share a standard
- ASCII is the standard to represent only English information with only 128 characters.
  - We need a standard to process Chinese, Arab, French and so on
- Unicode, formally The Unicode Standard, is a text encoding standard maintained by the Unicode Consortium designed to support the use of text written in all of the world's major writing systems

# Unicode and UTF8

- Version 15.1 of the standard defines **149,813 characters** and 161 scripts used in various ordinary, literary, academic, and technical contexts.
  - Moreover, the widespread adoption of Unicode was in large part responsible for the initial popularization of emoji (表情符号) outside of Japan. Unicode is ultimately capable of encoding more than 1.1 million characters.
  - **The first 128 characters** in the Unicode table correspond precisely to the **ASCII characters** that you'd reasonably expect them to.
- **Unicode itself is *not* an encoding.** Rather, Unicode is **implemented** by different character encodings.
  - Unicode doesn't tell you how to get actual bits from text—just code points. It doesn't tell you enough about how to convert text to binary data and vice versa.
  - Unicode is an **abstract encoding standard**, **not an encoding**
- **UTF-8** as well as its lesser-used cousins, UTF-16 and UTF-32, are **encoding formats** for representing Unicode characters as binary data of one or more bytes per character.
  - Unicode 规定了所有字符的序号，但这个序号不代表编码。譬如学生的序号和学号不是一样的。编码设计是一个复杂的问题，就要考虑时间效率，也要考虑空间占用率



Emoji Frequency



sloth

otter



waffle

ice cube



ringed planet flamingo

# Bytes and Bytearray

The core built-in types for manipulating binary data are bytes and bytearray

- `txt = b'hello world'`
- Bytes objects are **immutable** sequences of **single bytes**. The syntax for bytes literals is largely the same as that for string literals, except that a **b prefix is added**
- `class bytes([source[, encoding[, errors]])`
  - Only ASCII characters are permitted in bytes literals (regardless of the declared source code encoding). Any binary values over 127 must be entered into bytes literals using the appropriate escape sequence. (`\b \o \x`)
  - As with string literals, bytes literals may also use a `r` prefix to disable processing of escape sequences
- Since 2 **hexadecimal digits** correspond precisely to a single byte, hexadecimal numbers are a commonly used format for describing binary data
- `bytes.fromhex(string)`
  - This bytes class method returns a bytes object, decoding the given string object. The string must contain two hexadecimal digits per byte, with ASCII whitespace being ignored
- `hex([sep[, bytes_per_sep]])`
  - Return a string object containing two hexadecimal digits for each byte in the instance
- bytearray objects are a mutable counterpart to bytes objects
  - `class bytearray([source[, encoding[, errors]])`
- `str` and `bytes` have almost **identical functionality**

# Bytes Examples

str—bytes: encode()—decode()

- Python 3's str type is meant to represent **human-readable text** and can contain any **Unicode character**
- The bytes type, conversely, represents binary data, or sequences of raw bytes, that **do not intrinsically have an encoding attached** to it.
- In .encode() and .decode(), the encoding parameter is "utf-8" by default, though it's generally safer and more unambiguous to specify it

```
btxt=b'hello world', type=<class 'bytes'>, len=11
After splitting [b'hello', b'world']
Inverse: b'drow ollah'
Decode = hello world, encoding = b'hello world'

btxt=b'~!@#%&*()_+ {}:"|<?>', type=<class 'bytes'>, len=22
After splitting [b'~!@#%&*()_+ {}:"|<?>', b'{}:"|<?>']
Inverse: b'>?<|":{}({*%$#@!~'
Decode = ~!@#%&*()_+ {}:"|<?>, encoding = b'~!@#%&*()_+ {}:"|<?>'

btxt=b'\xe4\xba\xad\xe9\x80\x9a\xe5\xa4\xa7\xe5\xad\xa6top3', type=<class 'bytes'>, len=16
After splitting [b'\xe4\xba\xad\xe9\x80\x9a\xe5\xa4\xa7\xe5\xad\xa6top3']
Inverse: b'3pot\xa6\xe5\xa7\xa4\xe5\x9a\x80\xe9\xa4\xba\xe4'
Decode = 交通大学top3, encoding = b'\xe4\xba\xad\xe9\x80\x9a\xe5\xa4\xa7\xe5\xad\xa6top3'

btxt=b'(\xe0\xb9\x91\xc2\xb4\xe2\x88\x80\xe0\xb9\x91) \xe2\x82\x8d\xe1\x90\xa2..\xe1\x90\xa2\xe2\x82\x8e (\xe2\x80\xa2\xcc\x86 \xe1\xb5\x95\xe2\x80\xa2\xcc\x86) \xe2\x97\x9e\xe2\x99\xa1', type=<class 'bytes'>, len=53
After splitting [b'(\xe0\xb9\x91\xc2\xb4\xe2\x88\x80\xe0\xb9\x91)', b'\xe2\x82\x8d\xe1\x90\xa2..\xe1\x90\xa2\xe2\x82\x8e', b'(\xe2\x80\xa2\xcc\x86', b'\xe1\xb5\x95\xe2\x80\xa2\xcc\x86', b')\xe2\x97\x9e\xe2\x99\xa1']
Inverse: b'\xa1\x99\xe2\x9e\x97\xe2) \x86\xcc\xa2\x80\xe2\x95\xb5\xe1 \x86\xcc\xa2\x80\xe2( \x8e\x82\xe2\xa2\x90\xe1..\xa2\x90\xe1\x8d\xe2\xe2) \x91\xb9\xe0'\x80\x88\xe2\xb4\xc2\x91\xb9\xe0('
Decode = (ㄟ`ㄟ) (^.^.) (•~•~) ♪♡.encoding = b'(\xe0\xb9\x91\xc2\xb4\xe2\x88\x80\xe0\xb9\x91) \xe2\x82\x8d\xe1\x90\xa2..\xe1\x90\xa2\xe2\x82\x8e (\xe2\x80\xa2\xcc\x86 \xe1\xb5\x95\xe2\x80\xa2\xcc\x86) \xe2\x97\x9e\xe2\x99\xa1'
```

```
1 def test_bytes(btxt):
2     print(f"{btxt=}, type={type(btxt)}, len={len(btxt)}")
3     print(f"After splitting {btxt.split()}")
4     print(f"Inverse: {btxt[::-1]}")
5
6     txt = btxt.decode()
7     print(f"Decode = {txt}, encoding = {txt.encode()}")
8
9
10 btxt = b"hello world"
11 test_bytes(btxt)
12 print()
13
14 btxt = b'~!@#%&*()_+ {}:"|<?>'
15 test_bytes(btxt)
16 print()
17
18 btxt = "交通大学top3".encode() # Error b"交通大学 top3"
19 test_bytes(btxt)
20 print()
21
22 btxt = "(ㄟ`ㄟ) (^.^.) (•~•~) ♪♡".encode()
23 test_bytes(btxt)
24 print()
```

bytes给计算机看, str给人看

# UTF8

- In **Latin-1**, character codes above 127 are assigned to accented and otherwise special characters.
- **UTF-8** encoding: Character codes less than 128 are represented as a single byte; codes between 128 and 0x7ff (2047) are turned into **2 bytes**, where each byte has a value between 128 and 255; and codes above 0x7ff are turned into **3- or 4-byte** sequences having values between 128 and 255.
- **UTF-16** and **UTF-32** format text with a fixed-size 2 or 4 bytes and 4 bytes per each character scheme, respectively, even for characters that could otherwise fit in a single byte.

```
1 import sys
2 print(sys.getdefaultencoding())
3
4 print(chr(196))
5
6 s = 'ni'
7 print(s.encode('ascii'), s.encode('latin1'), s.encode('utf8'))
8 print(s.encode('utf16'), len(s.encode('utf16')))
9 print(s.encode('utf32'), len(s.encode('utf32')))
```

```
utf-8
Ä
b'ni' b'ni' b'ni'
b'\xff\xfe\x00i\x00' 6
b'\xff\xfe\x00\x00n\x00\x00\x00i\x00\x00\x00' 12
```

\xff\xfe thing is the byte order mark (BOM), used to indicate the endianness (大小端) of the data.

Big endian and little endian: Big-endian is an order in which the "big end" (most significant value in the sequence) is stored first, at the lowest storage address. Little-endian is an order in which the "little end" (least significant value in the sequence) is stored first.

# Coding Unicode Strings

- To code arbitrary Unicode characters in your strings, some of which you might not even be able to type on your keyboard, Python string literals support both `"\xNN"` **hex byte value escapes** and `"\uNNNN"` and `"\UNNNNNNNNN"` **Unicode escapes** in string literals.
- In Unicode escapes, the first form gives four hex digits to encode a 2-byte (16-bit) character code point, and the second gives eight hex digits for a 4-byte (32-bit) code point.
- Byte strings support only hex escapes for encoded text and other forms of byte-based data.
- `"\N{name}"` **Character named name** in the

Unicode database `"\N{LATIN SMALL LETTER A}"`

- Useful Functions

- `ascii()`
- `bin()`
- `bytes()`
- `chr()`
- `hex()`
- `int()`
- `oct()`
- `ord()`
- `str()`

<https://docs.python.org/3/library/unicodedata.html>

<https://home.unicode.org/>

<https://symbl.cc/en/>



# unicodedata — Unicode Database

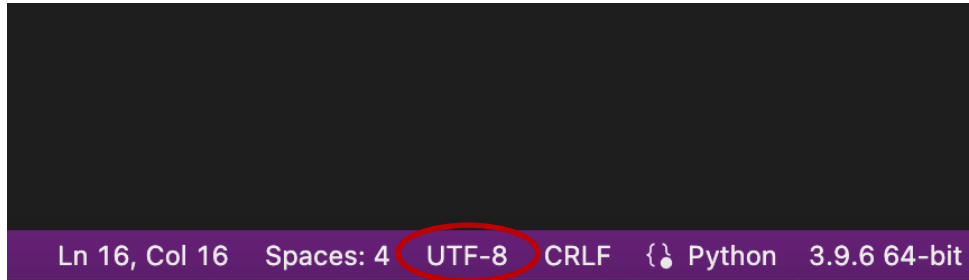
- This module provides access to the Unicode Character Database (UCD) which defines character properties for all Unicode characters.
- `unicodedata.lookup(name)`
- `unicodedata.name()`

```
1 import unicodedata
2
3 print(unicodedata.name("€"))
4 print(unicodedata.lookup("EURO SIGN"))
5
6 print(unicodedata.name(u"\U00003B1"))
7 print(unicodedata.lookup('Greek Small Letter Alpha'))
8
9 print(unicodedata.name(u"\U0001F44B"))
10 print(unicodedata.lookup('WAVING HAND SIGN'))
11
12 print(unicodedata.name(u"\U0001F440"))
13 print(unicodedata.lookup("EYES"))
14
15 print(unicodedata.name(u"\U0001F9F8"))
16 print(unicodedata.lookup("TEDDY BEAR"))
17
18 print(unicodedata.name(u"\U0001F3A4"))
19 print(unicodedata.lookup("MICROPHONE"))
20
21 print(unicodedata.name(u"\U00004dC7"))
22 print(unicodedata.lookup("HEXAGRAM FOR HOLDING TOGETHER"))
```

```
EURO SIGN
€
GREEK SMALL LETTER ALPHA
α
WAVING HAND SIGN
👋
EYES
👁️
TEDDY BEAR
🧸
MICROPHONE
🎤
HEXAGRAM FOR HOLDING TOGETHER
🔱
```

# Source File Character Set Encoding Declarations

- Finally, Unicode escape codes are fine for the occasional Unicode character in string literals, but they can become tedious if you need to embed non-ASCII text in your strings frequently.
  - To interpret the content of strings you code and hence embed within the text of your script files, **Python uses the UTF-8 encoding by default**, but it allows you to change this to support arbitrary character sets by including a comment that names your desired encoding.
- The comment must be of this form and must appear as either the first or second line in your script in either Python 2.X or 3.X:
  - `# -*- coding: latin-1 -*-`
  - When a comment of this form is present, Python will recognize strings represented natively in the given encoding.
  - This means you can edit your script file in a text editor that accepts and displays accented and other non-ASCII characters correctly, and Python will decode them correctly in your string literals



VS Code right corner

# File processing



# Shell

- An **absolute path** (绝对路径), which always begins with the root folder (C:\windows\...\a.txt)
- A **relative path** (相对路径), which is relative to the program's current working directory (.\a\b.txt)
- 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)

```
b.py
1 def spam(text): # File b.py
2     print(text, 'spam')
```

```
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
C:\Users\fcheng\OneDrive\CS124计算导论\2020 秋季\lecture notes> 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
```

**绝对路径 VS 相对路径**  
Shell中，路径名有空格  
用 “ ” 包起来处理

# Shell: parameter

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

- Run python program via shell with parameters:

```
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" "苟利"
a.py
Hello
world
苟利
Test spam
Hell world
Test for loop
0
1
2
```

如果出现shell错误，检查  
python文件是否在当前目录

# Ubuntu

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



打开一个美丽新世界  
自己寒假折腾

# Module sys

- 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

# Module sys: Example

- **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

```
1 import sys
2
3 print(sys.argv)
```

```
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")
```

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

```
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.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

```
1 import sys
2
3 print(sys.version)
4 print(sys.version_info)
5 print(sys.argv)
6 print(sys.path)
7 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\\py
thon36.zip', 'C:\\Program Files (x86)\\Python36-32\\DLLs', 'C:\\Program Files (x86)\\Python36-32\\lib', 'C:\\Progr
am 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**.
- `input()`, `print()`, `sys.stderr()`

# Module os 操作系统

- 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
- 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

```
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
```

# File in Python

- 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

- 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

- 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

# File Mode: Text and Binary

- Text mode implies **str objects**, and binary mode implies **bytes objects**:
- Text-mode files interpret file contents according to **a Unicode encoding**—either the default for your platform, or one whose name you pass in. By passing in an encoding name to open, you can force conversions for various types of Unicode files.
  - Textmode files also perform universal line-end translations: by default, all line-end forms map to the single '\n' character in your script, regardless of the platform on which you run it. As described earlier, text files also handle reading and writing the byte order mark (BOM) stored at the start-of-file in some Unicode encoding schemes.
- Binary-mode files instead return file content to you raw, as a sequence of integers representing byte values, **with no encoding or decoding and no line-end translations**.

# open()

- Syntax: `file object = open(file_name [, access_mode][, buffering])`
  - This function creates a file object, which would be utilized to call other support methods associated with it.
- Here are parameter details –
  - `file_name` – The `file_name` argument is a string
  - `access_mode` – The `access_mode` determines the mode in which the file has to be opened, i.e., read, write, append, etc. 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.
- Here is a list of the different modes of opening a file –
  - `r`: read; `w`: write; `x`: exclusive creation
  - `b`: binary; `+`: read and write; `a`: appending
  - Modes can be mixed
- If the file is not existed, Python will create it. If not, Python will truncating the file first. 'w' mode is very dangerous as all the original data will be erased
- 'x' open for exclusive creation, failing if the file already exists
  - Changed in version 3.3: The opener parameter was added. The 'x' mode was added.
  - Changed in version 3.3: IOError used to be raised, it is now an alias of OSError. FileExistsError is now raised if the file opened in exclusive creation mode ('x') already exists.

# 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

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

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

- 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)

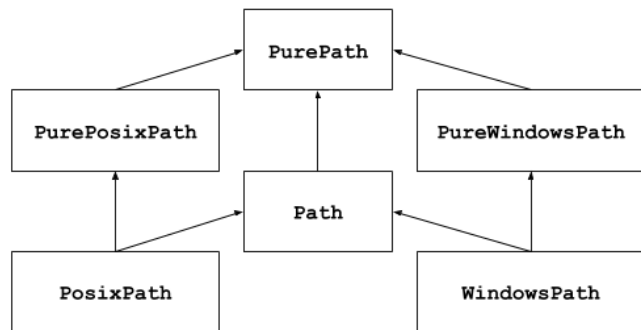
Mode	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	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.
wb	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.
w+	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.
wb+	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.
a	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.
ab	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.
a+	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.
ab+	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.

# 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

- pathlib — Object-oriented filesystem paths **推荐使用**
- <https://docs.python.org/3/library/pathlib.html> **所有相关方法**
- **Path, PosixPath, WindowsPath**





# 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()

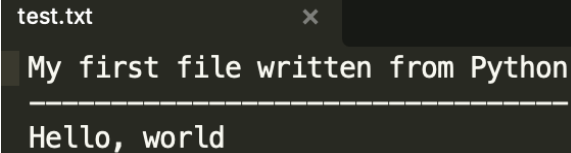
When you open() a file, do remember to close() it

- 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**

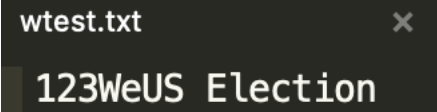
```
1 mf = open("test.txt", "w")
2 mf.write("My first file written from Python\n")
3 mf.write("-----\n")
4 mf.write("Hello, world\n")
5 mf.close()
```

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

- 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



```
test.txt
My first file written from Python
-----
Hello, world
```



```
wtest.txt
123WeUS Election
```

# 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
  - When we read() from the file we’re going to get bytes back rather than a string.
- 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']
[]
```

# 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()
```

# 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!
```

分别测试三种不同文件结尾

```
['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']
```

# With statement

Forget `close()` is a common mistake in programming. The solution is to automatically call 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`

# File: iterable

- We may use `for` to iterate a file just like processing a list/tuple/str/range
- This approach is more **Pythonic** and can be quicker and more memory efficient. Therefore, it is suggested you use this instead.

```
1 with open("test.txt", "r") as reader:
2     # Read and print the entire file line by line
3     for line in reader:
4         print(line, end="")
```

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

# Comments on write(), readline(), readlines()

- Warning Calling `f.write()` without using the `with` keyword or calling `f.close()` might result in the arguments of `f.write()` not being completely written to the disk, even if the program exits successfully.
- `f.readline()` reads a single line from the file; a newline character (`\n`) is left at the end of the string, and is only omitted on the last line of the file if the file doesn't end in a newline. This makes the return value unambiguous; if `f.readline()` returns an empty string, the end of the file has been reached, while a blank line is represented by `\n`, a string containing only a single newline.
- `writelines()` does not add line separators. You can alter the list of strings by using `map()` to add a new `\n` (line break) at the end of each string
  - `items = ['abc', '123', '!@#']`
  - `items = map(lambda x: x + '\n', items)`
  - `w.writelines(items)`
  - `with open(dst_filename, 'w') as f: #py 3.6`
  - `f.writelines(f'{s}\n' for s in lines)`



# tell(), seek() and flush()

- `f.tell()` returns an integer giving the file object's current position in the file represented as number of bytes from the beginning of the file when in binary mode and an opaque number when in text mode.
- To change the file object's position, use `f.seek(offset, whence)`. The position is computed from adding offset to a reference point; the reference point is selected by the whence argument. A whence value of 0 measures from the beginning of the file, 1 uses the current file position, and 2 uses the end of the file as the reference point. whence can be omitted and defaults to 0, using the beginning of the file as the reference point.
- `f.flush()`

```
1 f = open('workfile', 'wb+')
2 f.write(b'0123456789abcdef')
3 f.seek(5)      # Go to the 6th byte in the file
4 print(f.read(1))
5 f.seek(-3, 2)  # Go to the 3rd byte before the end
6 print(f.read(1))
7 f.close()
```

b'5'  
b'd'

# Mixed read and write in the same file

- The effect of mixing reads with writes on a file open for update is **entirely undefined** unless a file-positioning operation occurs between them (for example, a seek()).
- **Read after write: flush()**
- **Write after read: seek()**
- **test\_read\_write.py**
- For files open for appending (those which include a "+" sign), on which both input and output operations are allowed, the stream should be flushed (fflush) or repositioned (fseek, fsetpos, rewind) between either a writing operation followed by a reading operation or a reading operation which did not reach the end-of-file followed by a writing operation.
- SO to summarize, if you need to read a file after writing, you need to fflush the buffer and a write operation after read should be preceded by a fseek, as fd.seek(0, os.SEEK\_CUR)

# Reading

- Ch 37 Unicode and Byte Strings
- <https://realpython.com/read-write-files-python/>