Introduction to Information Science and Technology

(SI100B Python Part)

Fu Song
School of Information Science and Technology
ShanghaiTech University

Review

1. Constants, variables and expressions
2. Control flow:
   - Assignment
   - If-else,
   - For/while loop
   - Function/lambda

Outlines

1. Compound data
2. Input and Output (IO)

Data type classes

Numeric types: int, float, complex, Bool
Sequence types: list, tuple, range, str
Mapping types: dict
Set types: set, frozen set
None type: type(None)

Each class share the basic means of accessing elements.

Range

range(start,end,step)

- range object
- lazy evaluation: computes elements of the list on demand

Example:

range(2,7,2) → 2, 4 , 6

range(2,7,1) = range(2,7) # default step is 1

range(0,7,1) = range(7) # default start is 0

List

Construction

- L=[ ],
- L=[1, 2, 3]
- L=[1, 2, 3, 'abc'] **// items can be any type**
- L=[1, 2, 3, 'abc', 'd', [3,4,5], [7,8,9],range(3)]

List comprehension**:** **// a concise way to create lists**

- S = [x**2 for x in range(10)]
- V = [2**i for i in range(13)]
- M = [x for x in S if x % 2 == 0]

Slicing: L = S[start:end:step]

L = [i for i in range(10)]

S = L[1:10:3] → [1,4,7]

List Operations

- **Test**: in, not in
- **Slice**: l[start:end:steps]
- **Composition**: +, *
- **Adding**: append(x), extend(l), insert(i,x), l[start:end:steps] = t
- **Removing**: pop([i]), remove(x), del s[start:end:steps], clear()
- **Manipulation**: count(x), index(x), reverse(), copy(), sort()
- **Comparison**: **lexicographical ordering**
- **help(list)**
- **help(tuple)**

```
>>> L = [1,2,3,4,5,6]
>>> L.pop(1)
2
>>> L
[1, 3, 4, 5, 6]
>>> L.remove(3)
>>> L
[1, 4, 5, 6]
>>> del L[2]
>>> L
[1, 4, 6]
>>> L.index(4)
1
```

```
>>> L = [1,2,3]
>>> 1 in L
True
>>> 1 not in L
False
>>> False
False
>>> L[1]
2
>>> L[1:2:1]
[2]
>>> L*2
[1, 2, 3, 1, 2, 3]
>>> L + ['a','b','c']
[1, 2, 3, 'a', 'b', 'c']
>>> L.append('a')
>>> L
[1, 2, 3, 'a']
>>> L.extend(['a','b','c'])
>>> L
[1, 2, 3, 'a', 'a', 'b', 'c']
>>> L.insert(1,'m')
>>> L
[1, 'm', 2, 3, 'a', 'a', 'b', 'c']
>>> L[0:len(L):2]=[0,2,4,6]
>>> L
[0, 'm', 2, 3, 4, 'a', 6, 'c']
```

Matrices as Nested Lists

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Form 1

- L = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
- M[I][j] is L[i - 1] [j - 1]

Form 2 (good for sparse matrices)

- L = [[1,1,1],[1,2,2],[1,3,3], [2,1,4],[2,2,5],[2,3,6], [3,1,7],[3,2,8],[3,3,9]]
- M[i][j] is item [i, j, -] in L

Tuple

Construction

- L=()
- L=(1,2,3)
- L=(1,2,3,'abc')
- L=(1,2,3,'abc', 'd', (3,4,5),[7,8,9])

Tuple comprehension: (generator)

- S = (x**2 for x in range(10))
- V = (2**i for i in range(13))
- M = (x for x in S if x % 2 == 0)

Slicing: L = S[start:end:step]

L = (1,2,3,4)

S = L[1:4:2] → (2,4)

```
>>> V = (2**i for i in range(13))
>>> LV = [2**i for i in range(13)]
>>> V
<generator object <genexpr> at 0x02DA46C0>
>>> LV
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
>>> list(V)
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
>>> V[2]
Traceback (most recent call last):
  File "<pyshell#287>", line 1, in <module>
    V[2]
TypeError: 'generator' object is not subscriptable
>>> V = (1,2,3,4)
>>> V[2]
3
```

Tuple Operations

- Test: in, not in
- Slice: l[start:end:steps]
- Composition: +, *
- ~~Adding: append(x), extend(l), insert(i,x), l[start:end:steps] = t~~
- ~~Removing: pop([i]), remove(x), del s[start:end:steps], clear()~~
- Manipulation: count(x), index(x), ~~reverse(), copy(), sort()~~
- comparison: ****lexicographical ordering****

```
>>> V = (1,2,3,4)
>>> V[2]
3
>>> V = (1,2,3,4)
>>> 1 in V
True
>>> 1 not in V
False
>>> V*2
(1, 2, 3, 4, 1, 2, 3, 4)
>>> V+('a','b','c')
(1, 2, 3, 4, 'a', 'b', 'c')
>>> V.append('a')
Traceback (most recent call last):
  File "<pyshell#295>", line 1, in <module>
    V.append('a')
AttributeError: 'tuple' object has no attribute 'append'
>>> V.pop(0)
Traceback (most recent call last):
  File "<pyshell#296>", line 1, in <module>
    V.pop(0)
AttributeError: 'tuple' object has no attribute 'pop'
>>> V.count(1)
1
```

Immutable vs mutable

- List is mutable: read, add, remove, reorder
- Tuple in immutable: only read, cannot add or remove or reorder
- Why both?

Tuple is more efficient than list

Range is more efficient than tuple

```
for i in range(5):
    print(i)
```

Lazy evaluation

```
for i in [0,1,2,3,4]:
    print(i)
```

```
for i in (0,1,2,3,4):
    print(i)
```

String, Bytes and Bytearray

- A string is a sequence of Unicode characters (default UTF-8, up to 4 bytes) `>>> x="你好"`
- A bytes/bytearray is a sequence of ascii characters (one byte, 0-256)

- string and bytes are immutable
- bytearray is mutable

```
>>> x = bytes(b'abc')
>>> x[0]
97
>>> y = bytearray(b'abc')
>>> y.append(100)
>>> y
bytearray(b'abcd')
>>> y.extend(b'efg')
>>> y
bytearray(b'abcdefg')
```

Set and Frozenset

- A set/frozenset is an unordered collection with no duplicate elements
- both support mathematical operations like union, intersection, difference, and symmetric difference
- set is mutable
- frozenset is immutable

```
>>> x={1,2,3,4,5}
>>> 1 in x
True
>>> x.add(6)
>>> x
{1, 2, 3, 4, 5, 6}
>>> x.remove(1)
>>> x
{2, 3, 4, 5, 6}
>>>
```

```
>>> x = frozenset({1,2,3,4,5})
>>> 1 in x
True
>>> 1 not in x
False
>>> x.add(6)
Traceback (most recent call last):
  File "<pyshell#330>", line 1, in <module>
    x.add(6)
AttributeError: 'frozenset' object has no attribute 'add'
>>> x.remove(6)
Traceback (most recent call last):
  File "<pyshell#331>", line 1, in <module>
    x.remove(6)
AttributeError: 'frozenset' object has no attribute 'remove'
>>>
```

```
>>> x ={1,2,3}
>>> y ={2,3,4}
>>> x.intersection(y)
{2, 3}
>>> x.union(y)
{1, 2, 3, 4}
>>> x.difference(y)
{1}
>>>
```

Dict

A dict is a collection of key-value pairs: one value for each key

- values are indexed by keys
- key can be any immutable type
  e.g., string, number, tuple, bytes, and frozenset

```
>>> x = {1:'a', 2:'b', 3:'c', 4:'d'}
>>> x[1]
'a'
>>> x[3]
'c'
>>> x.keys()
dict_keys([1, 2, 3, 4])
>>> x.values()
dict_values(['a', 'b', 'c', 'd'])
```

Is {} an empty set or empty dict?

```
>>> x={}
>>> type(x)
<class 'dict'>
>>> y = set({})
>>> type(y)
<class 'set'>
```

Data type classes

Numeric types: int, float, complex, Bool
Sequence types: list, tuple, range, str
Mapping types: dict
Set types: set, frozen set
None type: type(None)

Each class share the basic means of accessing elements.

IO

How to store students information?

- **file, database, list, dict**

How to input students' information?

- **file, console/stdio**

详情 导出 1 - 20 of 215 后页› 末页»

学号	姓名	课程序号	课程代码	课程名称	课程类别	课程安排	学分	修读类别	轮次	选课状态
 31005857	刘瑞恺	SI100B.01	SI100B	信息科学技术导论B	信息课程群	1-16周64课时	4	正常		已选
 21170341	辜泳盛	SI100B.01	SI100B	信息科学技术导论B	信息课程群	1-16周64课时	4	正常		已选
 14880678	刘傲	SI100B.01	SI100B	信息科学技术导论B	信息课程群	1-16周64课时	4	正常		已选
 38608390	李卓儒	SI100B.01	SI100B	信息科学技术导论B	信息课程群	1-16周64课时	4	正常		已选
 20162352	晏伟星	SI100B.01	SI100B	信息科学技术导论B	信息课程群	1-16周64课时	4	正常		已选
 72196352	万翘楚	SI100B.01	SI100B	信息科学技术导论B	信息课程群	1-16周64课时	4	正常		已选
 75939940	向博	SI100B.01	SI100B	信息科学技术导论B	信息课程群	1-16周64课时	4	正常		已选
 99807733	张博炜	SI100B.01	SI100B	信息科学技术导论B	信息课程群	1-16周64课时	4	正常		已选

IO

- Read from console/stdio

```
>>> x = input("What's your name?\n")
What's your name?
Fu Song
>>> x
'Fu Song'
>>>
```

IO

Read from console/stdio

```
>>> info = {}
>>> while True:
        StudentID = input("Input student's number!\n")
        Name = input("Input student's name\n")
        Birthday = input("Input student's birthday\n")
        info[StudentID] = [Name, Birthday]


Input student's number!
1
Input student's name
A
Input student's birthday
19950101
Input student's number!
2
Input student's name
B
Input student's birthday
19950102
Input student's number!
3
Input student's name
C
Input student's birthday
19950103
Input student's number!
Traceback (most recent call last):
  File "<pyshell#423>", line 2, in <module>
    StudentID = input("Input student's number!\n")
  File "D:\Python3\lib\idlelib\PyShell.py", line 1386, in readline
    line = self._line_buffer or self.shell.readline()
KeyboardInterrupt
>>> print(info)
{'3': ['C', '19950103'], '2': ['B', '19950102'], '1': ['A', '19950101']}
```

When terminate?

Ctrl+c

IO

Read from console/stdio

```
>>> info = {}
>>> while True:
        Stop = input("Input Stop to terminate, other to continue!\n")
        if Stop =="Stop": break
        StudentID = input("Input student's number!\n")
        Name = input("Input Student's name!\n")
        Birthday = input("Input student's birthday!\n")
        info[StudentID] = [Name, Birthday]


Input Stop to terminate, other to continue!
n
Input student's number!
ID1
Input Student's name!
N1
Input student's birthday!
B1
Input Stop to terminate, other to continue!
n
Input student's number!
ID2
Input Student's name!
N2
Input student's birthday!
B2
Input Stop to terminate, other to continue!
Stop
>>> print(info)
{'ID2': ['N2', 'B2'], 'ID1': ['N1', 'B1']}
```

IO

Read from console/stdio

Type conversion

- Str-> int/float/complex

```
>>> info = {}
>>> while True:
        Stop = input("Input 1 to terminate, other to continue!\n")
        if Stop ==1: break

Input 1 to terminate, other to continue!
1
Input 1 to terminate, other to continue!
1
Input 1 to terminate, other to continue!
1
Input 1 to terminate, other to continue!
1
Input 1 to terminate, other to continue!
1
Input 1 to terminate, other to continue!
1
Input 1 to terminate, other to continue!
1
```

Not terminated

```
>>> while True:
        Stop = input("Input 1 to terminate, other to continue!\n")
        if int(Stop) ==1: break

Input 1 to terminate, other to continue!
2
Input 1 to terminate, other to continue!
1
>>>
```

```
>>> x = input("input a complex\n")
input a complex
1+2j
>>> x
'1+2j'
>>> complex(x)
(1+2j)
>>>
```

IO

Read from console/stdio

Pretty output?

```
>>> info = {}
>>> while True:
        Stop = input("Input Stop to terminate, other to continue!\n")
        if Stop =="Stop": break
        StudentID = input("Input student's number!\n")
        Name = input("Input Student's name!\n")
        Birthday = input("Input student's birthday!\n")
        info[StudentID] = [Name, Birthday]


Input Stop to terminate, other to continue!
n
Input student's number!
ID1
Input Student's name!
N1
Input student's birthday!
B1
Input Stop to terminate, other to continue!
n
Input student's number!
ID2
Input Student's name!
N2
Input student's birthday!
B2
Input Stop to terminate, other to continue!
Stop
>>> print(info)
{'ID2': ['N2', 'B2'], 'ID1': ['N1', 'B1']}
```

IO

Pretty output: three alternative ways

1. do all the string handling yourself; using string methods provided in string class, e.g., slicing, concatenation, etc., you can create any layout you can imagine
2. str.format() method
3. Template class from string module

More details refer to

“The Standard Library” 6.1 String—Common string operations

IO

Format String

```
>>> for key,value in info.items():
        s1 = "Student number: {0}, Name: {1}".format(key,value)
        print(s1)
```

argument name

```
Student number: 1, Name: 张三
Student number: 2, Name: 李四
Student number: 3, Name: 王某某
```

```
>>> for key,value in info.items():
        s2 = "Student number: {}, Name: {}".format(key,value)
        print(s2)
```

default argument name

```
Student number: 1, Name: 张三
Student number: 2, Name: 李四
Student number: 3, Name: 王某某
```

```
>>> for key,value in info.items():
        s3 = "Name: {name}, Student number {ID}".format(ID=key,name=value)
        print(s3)
```

keyword name

Aligned?

```
Name: 张三, Student number 1
Name: 李四, Student number 2
Name: 王某某, Student number 3
```

IO

```
>>> for key,value in info.items():
        s3 = "Name: {name}, Student number {ID}".format(ID=key,name=value)
        print(s3)
```

Format String

```
Name: 张三, Student number 1
Name: 李四, Student number 2
Name: 王某某, Student number 3
```

```
>>> for key,value in info.items():
        s3 = 'Name: {name:}, Student number {ID}'.format(ID=key,name=value.ljust(4,'　'))
        print(s3)
```

Full-width Space
Shift+space

```
Name: 张三　　, Student number 1
Name: 李四　　, Student number 2
Name: 王某某　, Student number 3
```

```
>>> for key,value in info.items():
        s3 = 'Name: {name:}, Student number {ID}'.format(ID=key,name=value.rjust(4,'　'))
        print(s3)
```

```
Name:　　张三, Student number 1
Name:　　李四, Student number 2
Name:　王某某, Student number 3
```

IO

```
>>> for key,value in info.items():
        s3 = "Name: {name}, Student number {ID}".format(ID=key,name=value)
        print(s3)
```

Format String

```
Name: 张三, Student number 1
Name: 李四, Student number 2
Name: 王某某, Student number 3
```

```
>>> for key, value in info.items():
        s3 = "Name: {0:  <4}, Student number {1}".format(key,value)
        print(s3)
```

- <: left-aligned
- >: right-aligned
- ^:centered

```
Name: 王某某  , Student number 3
Name: 张三    , Student number 1
Name: 李四    , Student number 2
```

```
>>> for key, value in info.items():
        s3 = "Name: {0:  <3}, Student number {1}".format(key,value)
        print(s3)
```

```
Name: 王某某, Student number 3
Name: 张三  , Student number 1
Name: 李四  , Student number 2
```

IO

$$\text{Area } \Delta ABC = \sqrt{s(s-a)(s-b)(s-c)} \qquad \text{where } s = \frac{a+b+c}{2}$$

Can you write a program which

1. ask a user to input three side lengths of the triangle
2. check whether the side lengths are good or not, if not, ask the user to input again
3. Otherwise, outputs the area

IO

$$\text{Area } \Delta ABC = \sqrt{s(s-a)(s-b)(s-c)} \qquad \text{where } s = \frac{a+b+c}{2}$$

```
import math

while True:
    a = input("Input the 1st side length: ")
    if a.isdigit() == False: continue
    else: a = int(a)

    b = input("Input the 2nd side length: ")
    if b.isdigit() == False: continue
    else: b = int(b)

    c = input("Input the 3nd side length: ")
    if c.isdigit() == False: continue
    else: c = int(c)

    if a+b<=c or a+c<=b or b+c<=a: continue
    if a<=0 or b<=0 or c<=0: continue
    s = (a+b+c)/2
    area = math.sqrt(s*(s-a)*(s-b)*(s-c))
    print("Side lengths are {0}, {1}, {2}. The ares is {3}.\n".format(a,b,c,area))
    break
```

Congratulations!

Now you are able to write a program that read data from a user and output data to the user

Have you ever needed to jot something down to remember it later?

- A list of ingredients to buy for a recipe?
- A guest list?
- A phone number?

Sometimes even programs need to jot something down so they can remember it later

- Remember what page I was reading in my e-book
- Remember what treasures I had collected when I took a break from the game

- But, programs will lose its data once terminate

One of the ways a program can make a note of something is to write it to a file

How do you write to a file with code?

Open file: f = Open(Filename, mode)

- ‘r’: read, the file must exist
- ‘w’: write, truncating the exist file, otherwise create a new file
- ‘a’: append, exists but don’t truncating, otherwise create a new file
- ‘x’: open for exclusive creation, failing if the file already exists
- ‘b/t’: binary/text mode, bytes/str objects, ‘t’ for default
- ‘+’: for updating

  E.g., mode r+b, w+t,…
- The file will be created in the same folder as your program
- By default that directory is:

```
import os
os.getcwd()
```

Close file f.close()

How do you write to a file with code?

```
>>> f = open("myfile.txt",'r')
Traceback (most recent call last):
  File "<pyshell#148>", line 1, in <module>
    f = open("myfile.txt",'r')
FileNotFoundError: [Errno 2] No such file or directory: 'myfile.txt'
>>> f = open("myfile.txt",'w')
>>> f.close()
>>> f = open("myfile.txt",'r')
>>>
>>> f.close()
```

File: write

Open file: f = open(Filename, mode) # w, w+, wb, w+b, a, a+,a+b

Write file:

f.writelines(list)

- Write a list into the file, one item for one line

f.write(str)

- Write str into the file

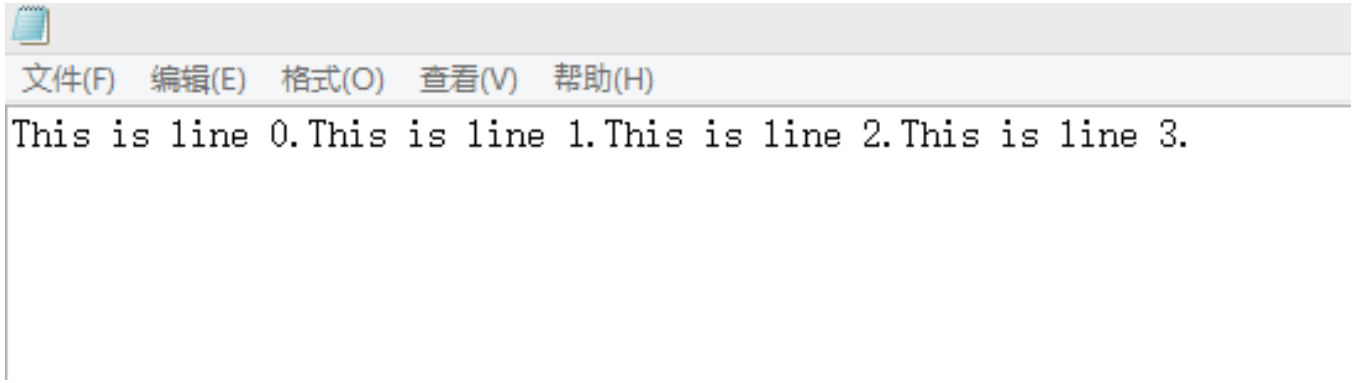
File: write

Write file:
f.writelines(list)

- Write a list into the file, one item for one line

```
>>> f = open("D://test.txt",'w')
>>> l = ["This is line {0}.".format(i) for i in range(4)]
>>> l
['This is line 0.', 'This is line 1.', 'This is line 2.', 'This is line 3.']
>>> f.writelines(l)
>>> f.close()
>>>
```

Only one line?



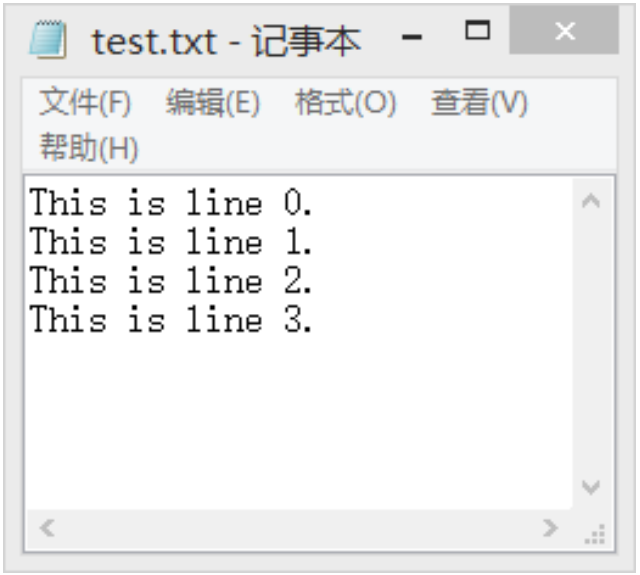
How did we specify to display text over multiple lines?

\n

File: write

Write file:
f.writelines(list)

- Write a list into the file, one item for one line




```
>>> f = open("D://test.txt", 'w')
>>> l = ["This is line {0}.\n".format(i) for i in range(4)]
>>> l
['This is line 0.\n', 'This is line 1.\n', 'This is line 2.\n', 'This is line 3.\n']
>>> f.writelines(l)
>>> f.close()
```

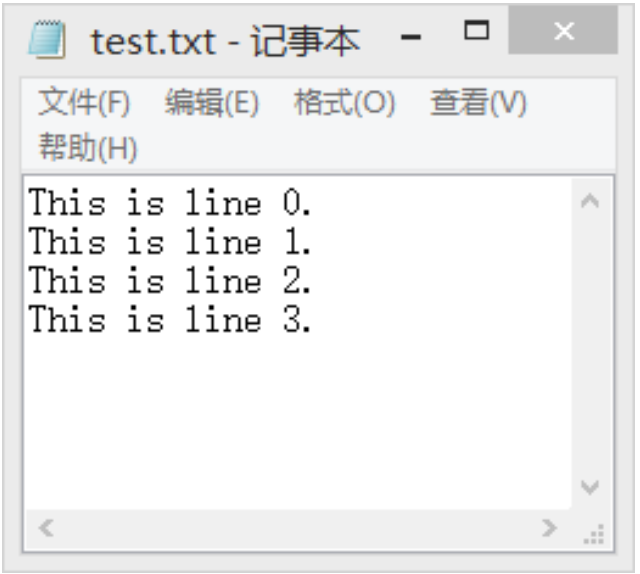
File: write

f.write(str)

- Write str into the file

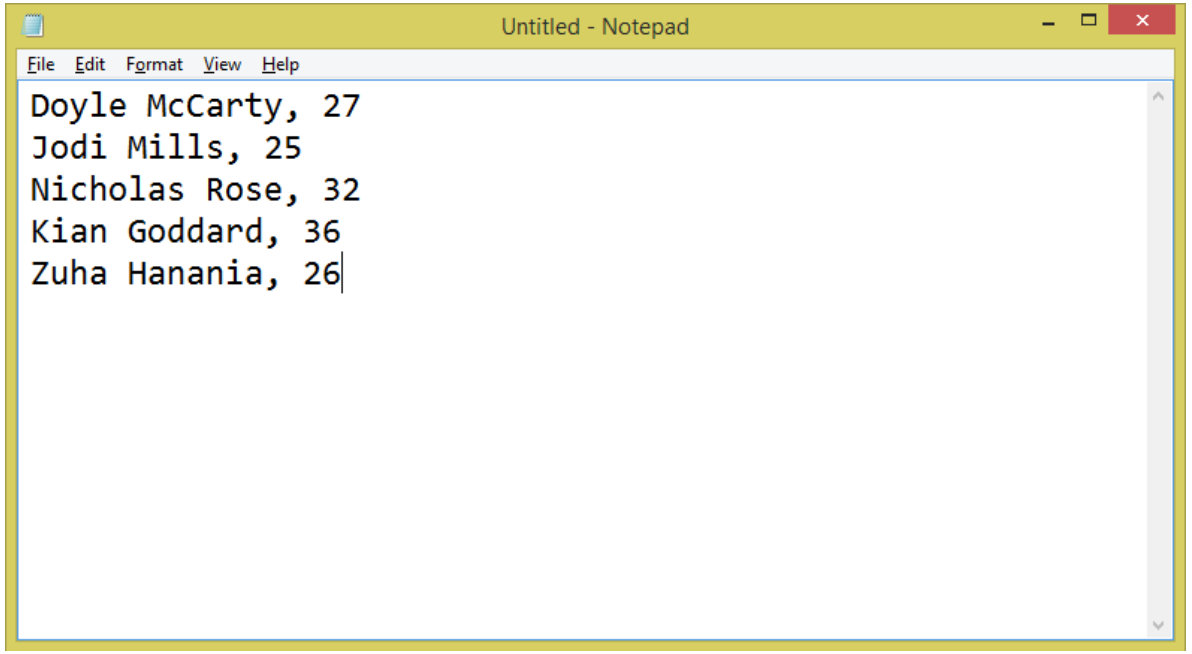
```
>>> f = open("D://test.txt", 'w+')
>>> for i in range(4):
        f.write("This is line {0}.\n".format(i))


16
16
16
16
>>> f.close()
```

← Number of characters written to a line



A common format for storing information in a file is Comma Separated (CSV)

- A CSV file contains data separated by a character (usually a comma).
- Each row represents one record of data
- It is sometimes called a Character Separated Values file because the separating character could be a different character such as a semi colon ‘;’



Writing something down to remember it is only helpful if you can read it when you need it later!

- Reading a shopping list at AUCHAN so you know what to buy
- Checking the number of guests on a guest list so you can see if you have enough food
- Looking up a phone number so you can call someone

In programs we often have to read information that was saved in files

- When you start your e-book reader, it looks up what page you were on when you last shut down
- When you start up your game, it looks up what treasures you had already collected so you can pick up where you left off
- There all also thousands of interesting OpenData files out there you can read to find out cool information you can use in your programs. For example: you can find out what keywords in Python Tutorial (You would be amazed at the data files you can find on the internet!)

File: read

Open file: f = open(Filename, mode) # r, r+, rb, r+b

Read file:

s = f.read(size=-1)

- size: the number of bytes to read until EOF is reached
- If size is negative or omitted, read until EOF is reached

s = f.readline(size=-1)

- Read size bytes up to one line until EOF is reached

L = f.readlines(size=-1)

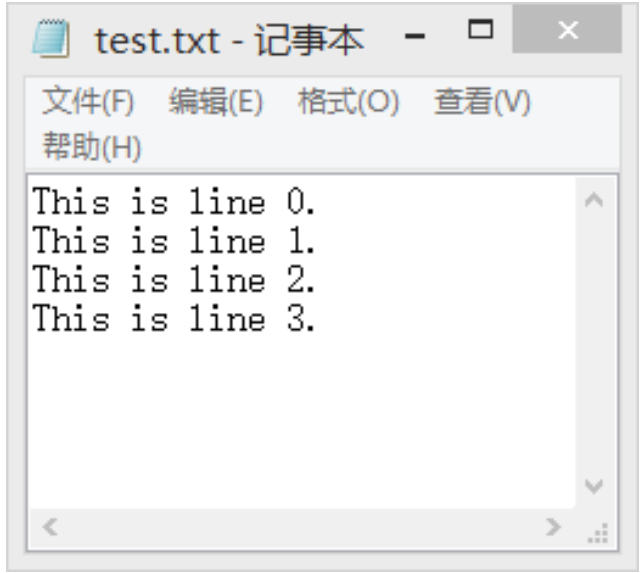
- a list of string, one item for one line
- no more lines will be read if the total size (in bytes/characters) of all lines so far exceeds size

File: read

s = f.read(size)

- size: the number of bytes to read until EOF is reached
- If size is negative or omitted, read until EOF is reached

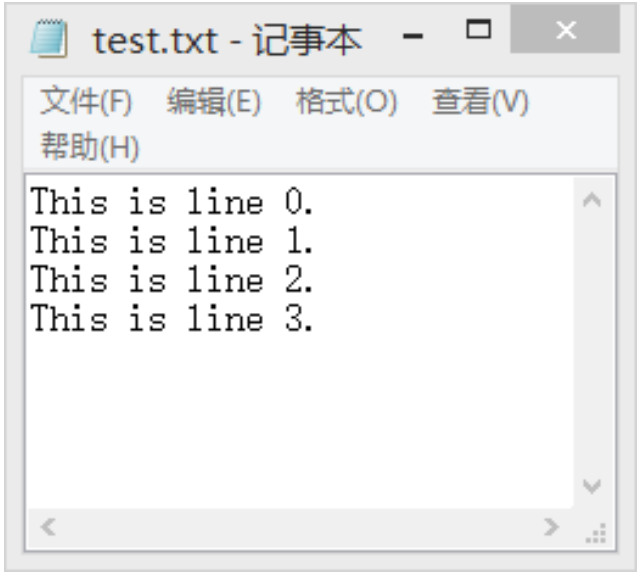
```
>>> f = open("D://test.txt", 'r')
>>> f.read(16)
'This is line 0.\n'
>>> f.read(14)
'This is line 1'
>>> f.read()
'.\nThis is line 2.\nThis is line 3.\n'
>>>
```



File: read

s = f.readline(size)

- Read size bytes up to one line until EOF is reached



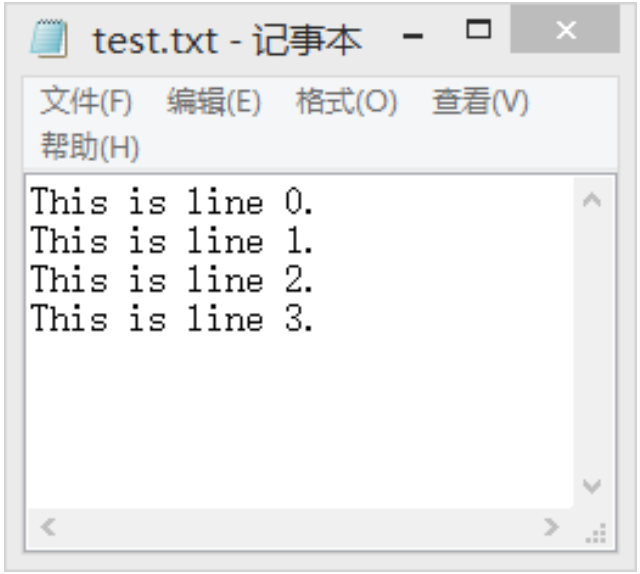

```
>>> f = open("D://test.txt", 'r')
>>> f.readline()
'This is line 0.\n'
>>> f.readline()
'This is line 1.\n'
>>> f.readline()
'This is line 2.\n'
>>> f.readline()
'This is line 3.\n'
>>> f.readline()
''
>>>
```

```
>>> f = open("D://test.txt", 'r')
>>> f.readline(20)
'This is line 0.\n'
>>> f.readline(15)
'This is line 1.'
>>> f.readline(15)
'\n'
```

File: read

L = f.readlines(size=-1)

- a list of string, one item for one line
- no more lines will be read if the total size (in bytes/characters) of all lines so far exceeds size

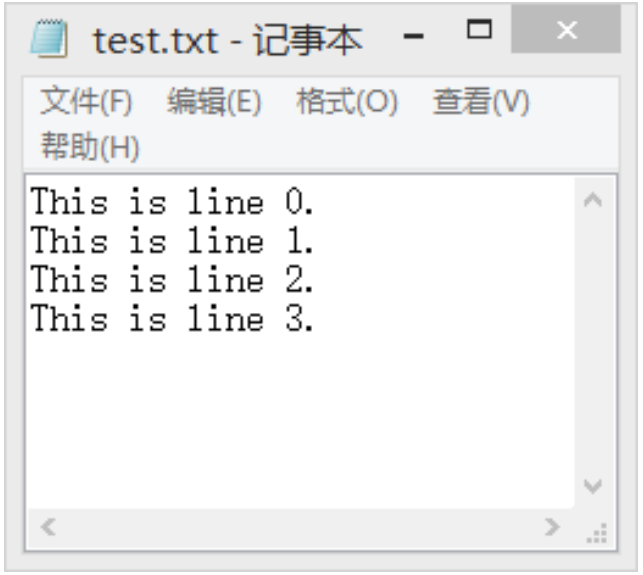



```
>>> f = open("D://test.txt", 'r')
>>> f.readlines()
['This is line 0.\n', 'This is line 1.\n',
'This is line 2.\n', 'This is line 3.\n']
>>> f.close()
>>> f = open("D://test.txt", 'r')
>>> f.readlines(5)
['This is line 0.\n']
>>> f.close()
>>> f = open("D://test.txt", 'r')
>>> f.readlines(20)
['This is line 0.\n', 'This is line 1.\n']
>>> f.close()
```

File: read large files

for oneline in f:
    process_oneline

```
>>> f = open("D://test.txt", 'r')
>>> for oneline in f:
        print(oneline)


This is line 0.

This is line 1.

This is line 2.

This is line 3.
```



File: beyond read and write

- seek(`offset`, `whence`) # change stream position
  - `0`: start of the stream (the default); offset should be zero or positive
  - `1`: current stream position; offset may be negative
  - `2`: end of the stream; offset is usually negative
  - offset should be `0` in text format when `whence` is non-zero
- tell() # get current stream position
- truncate(pos): remove all the text after the position pos (position from start of the stream)

File: beyond read and write

seek(`offset`, `whence`) # change stream position

- `0`: start of the stream (the default); offset should be zero or positive
- `1`: current stream position; offset may be negative
- `2`: end of the stream; offset is usually negative
- offset should be `0` in text format when `whence` is non-zero

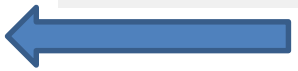
```
>>> f.close()
>>> f = open("D://test.txt", 'r')
>>> f.seek(3,0)
3
>>> f.readline()
's is line 0.\n'
>>> f.readline()
'This is line 1.\n'
>>> f.seek(0,0)
0
>>> f.readline()
'This is line 0.\n'
>>> f.seek(0,2)
68
>>> f.readline()
''
>>> f.seek(0,0)
0
>>> f.readline()
'This is line 0.\n'
```

File: beyond read and write

- tell() # get current stream position
- truncate(pos): remove all the text after the position pos (position from start of the stream)

```
>>> f = open("D://test.txt", 'r+')
>>> f.tell()
0
>>> f.readline()
'This is line 0.\n'
>>> f.tell()
17
>>> f.truncate(5)
5
>>> f.tell()
17
>>> f.readline()
'This is line 1.\n'
```

```
>>> f = open("D://test.txt", 'r+')
>>> f.read()
'This is line 0.\nThis is line 1.\nThis is line 2.\nThis is line 3.\n'
>>> f.seek(17,0)
17
>>> f.truncate(5)
5
>>> f.flush()
>>> f.readline()
''
>>>
```



why

File has a cache, cache should be flushed