

Programming and Data Structures with Python

Madhavan Mukund

<https://www.cmi.ac.in/~madhavan>

Programming and Data Structures with Python

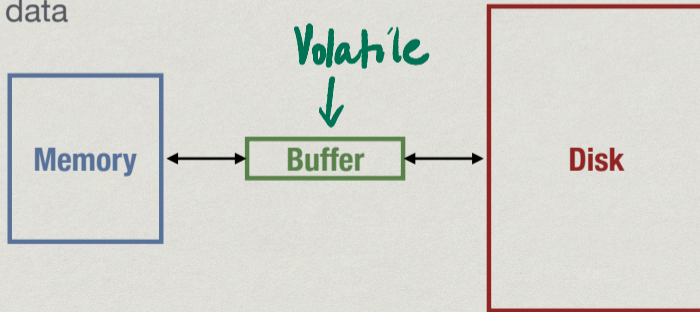
Lecture 25, 23 Nov 2023

Dealing with files

- * Standard input and output is not convenient for large volumes of data
- * Instead, read and write files on the disk
- * Disk read/write is much slower than memory

Disk buffers

- * Disk data is read/written in large blocks
- * “Buffer” is a temporary parking place for disk data



Reading/writing disk data

- * Open a file — create **file handle** to file on disk
 - * Like setting up a buffer for the file
- * Read and write operations are to file handle
- * Close a file
 - * Write out buffer to disk (**flush**)
 - * Disconnect file handle

Opening a file

```
fh = open("gcd.py", "r")
```

- * First argument to `open` is file name
 - * Can give a full path
- * Second argument is mode for opening file
 - * Read, `"r"`: opens a file for reading only
 - * Write, `"w"`: creates an empty file to write to
 - * Append, `"a"`: append to an existing file

Read through file handle

```
contents = fh.read()
```

- * Reads entire file into name as a single string

input()

Read through file handle

```
contents = fh.read()
```

- * Reads entire file into name as a single string

```
contents = fh.readline()
```

- * Reads one line into name—lines end with '`\n`'
 - * String includes the '`\n`', unlike `input()`

Read through file handle

```
contents = fh.read()
```

- * Reads entire file into name as a single string

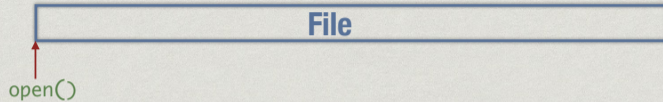
```
contents = fh.readline()
```

- * Reads one line into name—lines end with `'\n'`
 - * String includes the `'\n'`, unlike `input()`

```
contents = fh.readlines()
```

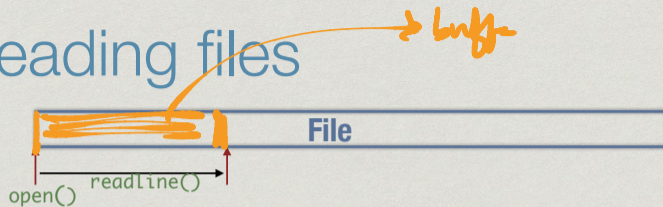
- * Reads entire file as list of strings
 - * Each string is one line, ending with `'\n'`

Reading files



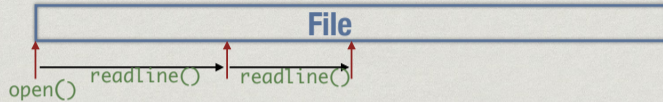
- * Reading is a sequential operation
 - * When file is opened, point to position 0, the start

Reading files



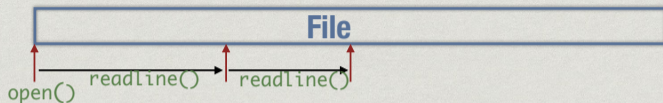
- * Reading is a sequential operation
 - * When file is opened, point to position 0, the start
 - * Each successive `readline()` moves forward

Reading files



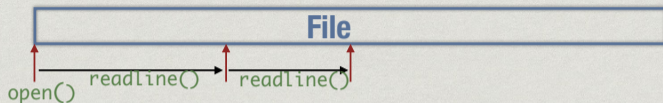
- * Reading is a sequential operation
 - * When file is opened, point to position 0, the start
 - * Each successive `readline()` moves forward

Reading files



- * Reading is a sequential operation
 - * When file is opened, point to position 0, the start
 - * Each successive `readline()` moves forward
- * `fh.seek(n)` — moves pointer to position `n`

Reading files



- * Reading is a sequential operation
 - * When file is opened, point to position 0, the start
 - * Each successive `readline()` moves forward
- * `fh.seek(n)` — moves pointer to position `n`
- * `block = fh.read(12)` — read a fixed number of characters

End of file

- * When reading incrementally, important to know when file has ended
- * The following both signal end of file
 - * `fh.read()` returns empty string ""
 - * `fh.readline()` returns empty string ""

Writing to a file

```
fh.write(s)
```

- * Write string `s` to file
 - * Returns number of characters written
 - * Include `'\n'` explicitly to go to a new line

Writing to a file

```
fh.write(s)
```

- * Write string `s` to file
 - * Returns number of characters written
 - * Include `'\n'` explicitly to go to a new line

```
fh.writelines(l)
```

- * Write a list of lines `l` to file
 - * Must includes `'\n'` explicitly for each string

Closing a file

```
fh.close()
```

- * Flushes output buffer and decouples file handle
- * All pending writes copied to disk

Closing a file

`fh.close()`

- * Flushes output buffer and decouples file handle
 - * All pending writes copied to disk

`fh.flush()`

- * Manually forces write to disk

Processing file line by line

```
contents = fh.readlines()
for l in contents:
    . . .
```

- * Even better

```
for l in fh.readlines():
    . . .
```


Copying a file

```
infile = open("input.txt", "r")
outfile = open("output.txt", "w")
for line in infile.readlines():
    outfile.write(line)
infile.close()
outfile.close()
```

will already have '\n'
from readlines()

Copying a file

```
infile = open("input.txt", "r")  
outfile = open("output.txt", "w")  
contents = infile.readlines()  
outfile.writelines(contents)  
infile.close()  
outfile.close()
```

Strip whitespace

— blank
tab
\n
--

- * `s.rstrip()` removes trailing whitespace

for line in contents:

```
    s = line.rstrip()
```

- * `s.lstrip()` removes leading whitespace
- * `s.strip()` removes leading and trailing whitespace

Splitting a string

- * Export spreadsheet as “comma separated value” text file

- * Want to extract columns from a line of text

- * Split the line into chunks between commas

```
columns = s.split(",")
```

- * Can split using any separator string

- * Split into at most `n` chunks

```
columns = s.split(" : ", n)
```

```
for i in len(columns):  
    columns[i].strip()
```

Joining strings

- * Recombine a list of strings using a separator

```
columns = s.split(",")  
joinstring = ","  
csvline = joinstring.join(columns)
```

```
date = "16"  
month = "08"  
year = "2016"  
today = "-".join([date, month, year])
```

String function

List function

columns.join(-)


Formatted printing

- * Recall that we have limited control over how `print()` displays output
 - * Optional argument `end="..."` changes default new line at the end of print
 - * Optional argument `sep="..."` changes default separator between items

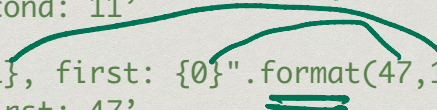
String format() method

- * By example

```
>>> "First: {0}, second: {1}".format(47,11)  
'First: 47, second: 11'
```



```
>>> "Second: {1}, first: {0}".format(47,11)  
'Second: 11, first: 47'
```



- * Replace arguments by position in message string

format() method ...

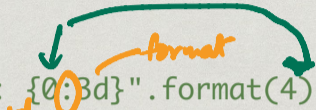
- * Can also replace arguments by name

```
>>> "One: {f}, two: {s}".format(f=47,s=11)
'One: 47, two: 11'
```

```
>>> "One: {f}, two: {s}".format(s=11,f=47)
'One: 47, two: 11'
```


Now, real formatting

```
>>> "Value: {0:3d}".format(4)
```



- * `3d` describes how to display the value `4`
- * `d` is a code specifies that `4` should be treated as an integer value
- * `3` is the width of the area to show `4`

```
'Value:  4'
```



Now, real formatting

↙ f = float

```
>>> "Value: {0:6.2f}".format(47.523)
```

- * 6.2f describes how to display the value 47.523
- * f is a code specifies that 47.523 should be treated as a floating point value
- * 6 — width of the area to show 47.523
- * 2 — number of digits to show after decimal point

```
"Value: 47.52"  
      123456
```

Real formatting

- * Codes for other types of values
 - * String, octal number, hexadecimal ...
- * Other positioning information
 - * Left justify
 - * Add leading zeroes
- * Derived from `printf()` of C, see Python documentation for details

Passing values to functions

- Argument value is substituted for name

```
def power(x,n):  
    ans = 1  
    for i in range (0,n):  
        ans = ans*x  
    return(ans)
```

- Like an implicit assignment statement

```
power(3,5)
```



```
x = 3  
n = 5  
ans = 1  
for i in range ...
```

Passing arguments by name

```
def power(x,n):  
    ans = 1  
    for i in range (0,n):  
        ans = ans*x  
    return(ans)
```

- Call `power(n=5,x=4)`

Default arguments

- Recall `int(s)` converts string to integer
 - `int("76")` is 76
 - `int("A5")` generates an error
- Actually `int(s,b)` takes two arguments, string `s` and base `b`
 - `b` has default value 10
 - `int("A5",16)` is 165 ($10 \times 16 + 5$)

Default arguments

- Recall `int(s)` converts string to integer
 - `int("76")` is 76
 - `int("A5")` generates an error
- Actually `int(s,b)` takes two arguments, string `s` and base `b`
 - `b` has default value 10
 - `int("A5",16)` is 165 ($10 \times 16 + 5$)

```
def int(,b=10):  
    ...
```



- Default value is provided in function definition
- If parameter is omitted, default value is used
 - Default value must be available at definition time
 - `def Quicksort(A,l=0,r=len(A):` does not work

Default arguments

```
def f(a,b,c=14,d=22):  
    ...
```

- `f(13,12)` is interpreted as `f(13,12,14,22)`
- `f(13,12,16)` is interpreted as `f(13,12,16,22)`
- Default values are identified by position, must come at the end
 - Order is important

Function definitions

- `def` associates a function body with a name
- Flexible, like other value assignments to name
- Definition can be conditional

```
if condition:
    def f(a,b,c):
        ...
else:
    def f(a,b,c):
        ....
```

Function definitions

- `def` associates a function body with a name
- Flexible, like other value assignments to name
- Definition can be conditional

```
if condition:
    def f(a,b,c):
        ...
else:
    def f(a,b,c):
        ....
```

- Can assign a function to a new name

```
def f(a,b,c):
    ...

g = f
```

- Now `g` is another name for `f`

Passing functions as parameters

- Apply `f` to `x` `n` times

```
def apply(f,x,n):  
    res = x  
    for i in range(n):  
        res = f(res)  
    return(res)
```

*function defn
passed as arg*

Passing functions as parameters

- Apply `f` to `x` `n` times

```
def apply(f,x,n):  
    res = x  
    for i in range(n):  
        res = f(res)  
    return(res)
```

```
def square(x):  
    return(x*x)
```

```
apply(square,5,2)
```

```
square(square(5))
```

↓
"Higher order
functions"

Passing functions as parameters

- Apply `f` to `x` `n` times

```
def apply(f,x,n):  
    res = x  
    for i in range(n):  
        res = f(res)  
    return(res)
```

```
def square(x):  
    return(x*x)
```

```
apply(square,5,2)
```

```
square(square(5))
```

625

- Useful for customizing functions such as `sort`

- Define `cmp(x,y)` that returns `-1` if `x < y`, `0` if `x == y` and `1` if `x > y`

- `cmp("aab","ab")` is `-1` in dictionary order

- `cmp("aab","ab")` is `1` if we compare by length

- `def mysort(l,cmp=defaultcmp):`

if $x[i] < x[j]$
=

$v = \text{cmp}(x[i], x[j])$
if $v == -$

Global variables

"Lexical scope"

```
def f(-):  
    ≡  
    ≡  
def g(-):  
    ≡ x=4  
    def h(-):  
        ≡  
        ≡  
x=2  
y=f(x)  
:
```

global x
but same x as outside

encounter a name x that is not defined

