Binary file

- When a file is opened in *binary mode by adding a b (lowercase only) to the mode string* argument in the built-in open call, reading its data does not decode it in any way but simply returns its content raw and unchanged, as a bytes object; writing similarly takes a bytes object and transfers it to the file unchanged. Binary-mode files also accept a bytearray object for the content to be written to the file.

Binary file

*filesimple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\file\fi

File Edit Format Run Options Window Help

```
x=b"\x0a\x1b\x2c"
print(x)
file = open('binfile.bin', 'wb')
file.write(x)
file.close()
file = open('binfile.bin', 'rb')
data = file.read()
print(data)
file.close()

#Output

b'\n\x1b,'
b'\n\x1b,'
```

Binary file

*filesimple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\file\f

File Edit Format Run Options Window Help

```
newFileBytes = [123, 3, 255, 0, 100]
file = open('binfile.bin', 'wb')
data=bytearray(newFileBytes)
file.write(data)
file.close()
file = open('binfile.bin', 'rb')
data = file.read()
newFileBytes=list(data)
print(newFileBytes)
file.close()

#Output

    [123, 3, 255, 0, 100]
```

Writing Big integers

filesimple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\file\filesimple.py (3.4.4)

File Edit Format Run Options Window Help

```
i = 627710173538668076383578942317605901376719477318284228408 1
with open('out.bin', 'wb') as file:
    file.write((i).to_bytes(24, byteorder='big', signed=False))

with open('out.bin', 'rb') as file:
    j = int.from_bytes(file.read(24), byteorder='big')

print(j)

#Output

    627710173538668076383578942317605901376719477318284228408 1
```

Storing Packed Binary Data: struct

- The struct module knows how to both compose and parse packed binary data. In a sense, this is another data-conversion tool that interprets strings in files as binary data.

Storing Packed Binary Data: struct

- struct.pack(*fmt*, *v1*, *v2*, ...)

  Return a bytes object containing the values *v1*, *v2*, ... packed according to the format string *fmt*. The arguments must match the values required by the format exactly.

- struct.unpack(*fmt*, *buffer*)

  Unpack from the buffer *buffer* (presumably packed by pack(fmt, ...)) according to the format string *fmt*. The result is a tuple even if it contains exactly one item. The buffer's size in bytes must match the size required by the format, as reflected by calcsize().

- **struct.calcsize(*fmt*)**

  Return the size of the struct (and hence of the bytes object produced by pack(fmt, ...)) corresponding to the format string *fmt*.

Format Strings

- Format strings are the mechanism used to specify the expected layout when packing and unpacking data
- struct.pack(*fmt*, *v1*, *v2*, *...*)

- struct.unpack(*fmt*, *buffer*)

  .
- **struct.calcsize(*fmt*)**
- **fmt is a string with format : ‘[command][type]’**

Byte Order, Size, and Alignment

Character	Byte order	Size	Alignment
@	native	native	native
=	native	standard	none
<	little-endian	standard	none
>	big-endian	standard	none
!	network (= big-endian)	standard	none

- @ is default

Format Characters

- The ‘Standard size’ column refers to the size of the packed value in bytes when using standard size; that is, when the format string starts with one of '<', '>', '!' or '='. When using native size, the size of the packed value is platform-dependent.

Format	C Type	Python type	Standard size	Notes
x	pad byte	no value		
c	char	bytes of length 1	1	
b	signed char	integer	1	(1),(3)
B	unsigned char	integer	1	(3)
?	_Bool	bool	1	(1)
h	short	integer	2	(3)
H	unsigned short	integer	2	(3)
i	int	integer	4	(3)
I	unsigned int	integer	4	(3)
l	long	integer	4	(3)
L	unsigned long	integer	4	(3)
q	long long	integer	8	(2), (3)
Q	unsigned long long	integer	8	(2), (3)
n	ssize_t	integer		(4)
N	size_t	integer		(4)
f	float	float	4	(5)
d	double	float	8	(5)
s	char[]	bytes		
p	char[]	bytes		
P	void *	integer		(6)

Format Characters

For the 's' format character, the count is interpreted as the size of the string, not a repeat count like for the other format characters; for example, '10s' means a single 10-byte string, while '10c' means 10 characters. If a count is not given, it defaults to 1. For packing, the string is truncated or padded with null bytes as appropriate to make it fit. For unpacking, the resulting string always has exactly the specified number of bytes. As a special case, '0s' means a single, empty string (while '0c' means 0 characters).

Binary file

*filesimple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\file\filesimple

File Edit Format Run Options Window Help

```
import struct

x=34523
file = open('binfile.bin', 'wb')
data=struct.pack('i', x)
file.write(data)
file.close()
file = open('binfile.bin', 'rb')
intsize = struct.calcsize('i')
data = file.read(intsize)
num = struct.unpack('i', data)
print(num)

file.close()

#Output

    (34523,)
```

'@i'

filesimple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\file\filesimple.py (3.4.4)

File Edit Format Run Options Window Help

```
import struct

filename=input('enter the file name: ')
file = open(filename, 'wb')
n=int(input('enter the number of data: '))
for i in range(n):
    num=int(input('enter the data: '))
    square=num * num
    data=struct.pack('i', square)
    file.write(data)
file.close()

file = open(filename, 'rb')
intsize = struct.calcsize('i')
for i in range(n):
    data = file.read(intsize)
    num = struct.unpack('i', data)
    print (data, ' corresponding to: ', num)
file.close()

#Output
enter the file name: test.bin
enter the number of data: 3
enter the data: 4
enter the data: 5
enter the data: 6
b'\x10\x00\x00\x00'  corresponding to:  (16,)
b'\x19\x00\x00\x00'  corresponding to:  (25,)
b'$\x00\x00\x00'  corresponding to:  (36,)
```

Storing Packed Binary Data: struct

*filesimple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\file\filesimple.p

File Edit Format Run Options Window Help

```
import struct

packed = struct.pack('>i4sh', 7, b'spam', 8)
print(packed)
file = open('data.bin', 'wb')
file.write(packed)
file.close()
data = open('data.bin', 'rb').read()
print(data)
unp=struct.unpack('>i4sh', data)
print(unp)

#Output

    b'\x00\x00\x00\x07spam\x00\x08'
    b'\x00\x00\x00\x07spam\x00\x08'
    (7, b'spam', 8)
```

Group of Data

However, sometimes a counting loop must be used with input. The data sometimes consist of several groups where each group starts with a header that says how much data is in that group. For example:

> There are two groups of computer science students. The first group (called group "A") uses on-line lessons. The other group (called group "B") uses traditional printed text. All the students are given the same mid-term examination. Which group has the higher test average?

The file of test scores looks like this (the blue comments are not in the file):

```
3          <-- number of students in group "A"
87
98
95
4          <-- number of students in group "B"
78
82
91
84
```

Group "A" has three students in it and group "B" has four students in it. The program is to compute two averages from the data in this file.

Group of Data

```
import struct

filename=input('enter the file name: ')
file = open(filename, 'wb')
n=int(input('enter the number of data fo Group A: '))
data=struct.pack('i', n)
file.write(data)
for i in range(n):
    num=int(input('enter the data: '))
    data=struct.pack('i', num)
    file.write(data)

n=int(input('enter the number of data fo Group B: '))
data=struct.pack('i', n)
file.write(data)
for i in range(n):
    num=int(input('enter the data: '))
    data=struct.pack('i', num)
    file.write(data)
file.close()
```

Group of Data

```
intsize = struct.calcsize('i')
filename=input('enter the file name: ')
file = open(filename, 'rb')
data=file.read(intsize)
sizeA = struct.unpack('i', data)
sumA=0
count=0
while(count<sizeA[0]):
    data=file.read(intsize)
    value = struct.unpack('i', data)
    sumA=sumA+value[0]
    count=count+1

if (sizeA[0]>0):
    print('Group A average: ', sumA/sizeA[0])
else:
    print('Group A has no student')
```

Group of Data

```
data=file.read(intsize)
sizeB = struct.unpack('i', data)
sumB=0
count=0
while(count<sizeB[0]):
    data=file.read(intsize)
    value = struct.unpack('i', data)
    sumB=sumB+value[0]
    count=count+1

if (sizeB[0]>0):
    print('Group B average: ', sumB/sizeB[0])
else:
    print('Group B has no student')
```

Group of Data

```
enter the file name: ga.b
enter the number of data fo Group A: 3
enter the data: 87
enter the data: 98
enter the data: 95
enter the number of data fo Group B: 4
enter the data: 78
enter the data: 82
enter the data: 91
enter the data: 84
enter the file name: ga.b
Group A average:  93.33333333333333
Group B average:  83.75
```

Storing Native Objects: pickle

- The pickle module is a more advanced tool that allows us to store almost any Python object in a file directly, with no to- or from-string conversion requirement on our part.
- It's like a super-general data formatting and parsing utility. To store a dictionary in a file, for instance, we pickle it directly.
- We can write any data
- dump for write and load for read in to files
- dumps return pickle representation of a real data
- loads convert a pickle data to real data

Storing Native Objects: pickle

filesimple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\file\filesimple.py (3.4.4)

File Edit Format Run Options Window Help

```
import pickle

D = {'a': 1, 'b': 2}
F = open('datafile.pkl', 'wb')
pickle.dump(D,F)                    # Pickle any object to file
F.close

F = open('datafile.pkl', 'rb')
E = pickle.load(F)                  # Load any object from file
print(E)

#Output
{'a': 1, 'b': 2}
```

Storing Native Objects: pickle

filesimple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\file\filesimple.py (3.4.4)

File Edit Format Run Options Window Help

```
data=open('datafile.pkl', 'rb').read()
print(data)

#Output

b'\x80\x03}q\x00(X\x01\x00\x00\x00aq\x01K\x01X\x01\x00\x00\x00bq\x02K\x02u.'
```

Writing Big integers

- We can convert the big integers to bytes using the conversion methods:
- int.to_bytes(*length*, *byteorder*, *signed=False*)
- int.from_bytes(*bytes*, *byteorder*, *signed=False*)

- Byteorder=big or little

Writing Big integers

filesimple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\file\filesimple.py (3.4.4)

File Edit Format Run Options Window Help

```
i = 627710173538668076383578942317605901376719477318284228408 1
with open('out.bin', 'wb') as file:
    file.write((i).to_bytes(24, byteorder='big', signed=False))

with open('out.bin', 'rb') as file:
    j = int.from_bytes(file.read(24), byteorder='big')

print(j)

#Output

    627710173538668076383578942317605901376719477318284228408 1
```

Writing Big integers

- **We can use the pickle**

```
import pickle

i = 627710173538668076383578942317605901376719477318284228408 1
j = -627710173538668076383578942317605901376719477318284228408 1

with open('out.bin', 'wb') as file:
    pickle.dump(i,file)
    pickle.dump(j,file)

with open('out.bin', 'rb') as file:
    i = pickle.load(file)
    j = pickle.load(file)

print(i)
print(j)
```

Group of Data

However, sometimes a counting loop must be used with input. The data sometimes consist of several groups where each group starts with a header that says how much data is in that group. For example:

> There are two groups of computer science students. The first group (called group "A") uses on-line lessons. The other group (called group "B") uses traditional printed text. All the students are given the same mid-term examination. Which group has the higher test average?

The file of test scores looks like this (the blue comments are not in the file):

```
3          <-- number of students in group "A"
87
98
95
4          <-- number of students in group "B"
78
82
91
84
```

Group "A" has three students in it and group "B" has four students in it. The program is to compute two averages from the data in this file.

Group of Data

```
import pickle

filename=input('enter the file name: ')
file = open(filename, 'wb')
n=int(input('enter the number of data fo Group A: '))
pickle.dump(n,file)
for i in range(n):
    num=int(input('enter the data: '))
    pickle.dump(num,file)

n=int(input('enter the number of data fo Group B: '))
pickle.dump(n,file)
for i in range(n):
    num=int(input('enter the data: '))
    pickle.dump(num,file)
file.close()
```

Group of Data

```
filename=input('enter the file name: ')
file = open(filename, 'rb')
sizeA = pickle.load(file)
sumA=0
count=0
while(count<sizeA):
    value = pickle.load(file)
    sumA=sumA+value
    count=count+1

if (sizeA>0):
    print('Group A average: ', sumA/sizeA)
else:
    print('Group A has no student')
```

Group of Data

```
sizeB = pickle.load(file)
sumB=0
count=0
while(count<sizeB):
    value = pickle.load(file)
    sumB=sumB+value
    count=count+1

if (sizeB>0):
    print('Group B average: ', sumB/sizeB)
else:
    print('Group B has no student')
```

Group of Data

```
enter the file name: ga.b
enter the number of data fo Group A: 3
enter the data: 87
enter the data: 98
enter the data: 95
enter the number of data fo Group B: 4
enter the data: 78
enter the data: 82
enter the data: 91
enter the data: 84
enter the file name: ga.b
Group A average:  93.33333333333333
Group B average:  83.75
```

Copy binary file

cpyb.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/file/cpyb.py (3.4.4)

File Edit Format Run Options Window Help

```
import sys

def copyfile(source, destination):
    with open(source, 'rb') as fin, open(destination,'wb') as fout:
        while True:
            buf=fin.read(1024)
            if not buf:
                break
            n=fout.write(buf)
    return

if len(sys.argv) < 3:
    print("Wrong parameter")
    print("./cpy.py filename1 filename2")
    sys.exit(1)


copyfile(sys.argv[1], sys.argv[2])
```

Writing Text reading Binary

filesimple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\file\filesimple.py (3.4.4)

File Edit Format Run Options Window Help

```
S = 'sp\xc4m'
file = open('unidata.txt', 'w', encoding='utf-8')
print(file.write(S))
S=' New \n text in file \n'
file.write(S)
file.close()

file = open('unidata.txt', 'rb')
line=file.read()
print(line)
file.close()


#Output

4
b'sp\xc3\x84m New \r\n text in file \r\n'
```

Phone Book

- Now here is a new version of the phone numbers program that we made earlier.
- This version uses a binary file for loading and saving the phone numbers

```
def print_menu():
    print()
    print('1. Print Phone Numbers')
    print('2. Add a Phone Number')
    print('3. Remove a Phone Number')
    print('4. Lookup a Phone Number')
    print('5. Load numbers')
    print('6. Save numbers')
    print('7. Quit')
    print()

phone_list = {}
menu_choice = 0
while True:
    print_menu()
    menu_choice =int(input("Type in a number (1-7): "))
    if menu_choice == 1:
        print_numbers(phone_list)
    elif menu_choice == 2:
        print("Add Name and Number")
        name = input("Name: ")
        phone =int(input("Number: "))
        add_number(phone_list, name, phone)
    elif menu_choice == 3:
        print("Remove Name and Number")
        name =input("Name: ")
        remove_number(phone_list, name)
    elif menu_choice == 4:
        print("Lookup Number")
        name = input("Name: ")
        print(lookup_number(phone_list, name))
```

```
phone_list = {}
menu_choice = 0
while True:
    print_menu()
    menu_choice =int(input("Type in a number (1-7): "))
    if menu_choice == 1:
        print_numbers(phone_list)
    elif menu_choice == 2:
        print("Add Name and Number")
        name = input("Name: ")
        phone =int(input("Number: "))
        add_number(phone_list, name, phone)
    elif menu_choice == 3:
        print("Remove Name and Number")
        name =input("Name: ")
        remove_number(phone_list, name)
    elif menu_choice == 4:
        print("Lookup Number")
        name = input("Name: ")
        print(lookup_number(phone_list, name))
    elif menu_choice == 5:
        filename = input("Filename to load: ")
        load_numbers(phone_list, filename)
    elif menu_choice == 6:
        filename = input("Filename to save: ")
        save_numbers(phone_list, filename)
    elif menu_choice == 7:
        break
    else:
        continue

print("Goodbye")
```

```python
import struct

def print_numbers(numbers):
    print("Telephone Numbers:")
    for k, v in numbers.items():
        print("Name:", k, "\tNumber:", v)

def add_number(numbers, name, number):
    numbers[name] = number

def lookup_number(numbers, name):
    if name in numbers:
        return "The number is " + numbers[name]
    else:
        return name + " was not found"

def index(numbers, key):
    for i in numbers.key():
        if k==key : return i
    else:
        return -1

def remove_number(numbers, name):
    if name in numbers:
        del numbers[name]
    else:
        print(name," was not found")
```

```
def find_words(input):
    for i in range(0, len(input)):
        if input[i] == 0:
            return input[0:i]
    return ""

def load_numbers(numbers, filename):
    in_file = open(filename, 'rb')
    while True:
        data=in_file.read(14)
        if not data:
            break
        unp=struct.unpack('>10si', data)
        name=find_words(unp[0]).decode('utf-8')
        phone=unp[1]
        numbers[name] = phone
    in_file.close()

def save_numbers(numbers, filename):
    out_file = open(filename, "wb")
    for name, phone in numbers.items():
        nameb=bytes(name.encode('utf-8'))
        packed = struct.pack('>10si', nameb, phone)
        out_file.write(packed)
    out_file.close()
```

```
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Load numbers
6. Save numbers
7. Quit

Type in a number (1-7): 5
Filename to load: data.bin

1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Load numbers
6. Save numbers
7. Quit

Type in a number (1-7): 1
Telephone Numbers:
Name:   Number: 0
Name: hossien   Number: 881216
Name: kamran    Number: 363131
Name: ali       Number: 774493
Name: kazem     Number: 521745
Name: sohael    Number: 323281
Name: kayvan    Number: 354418
```

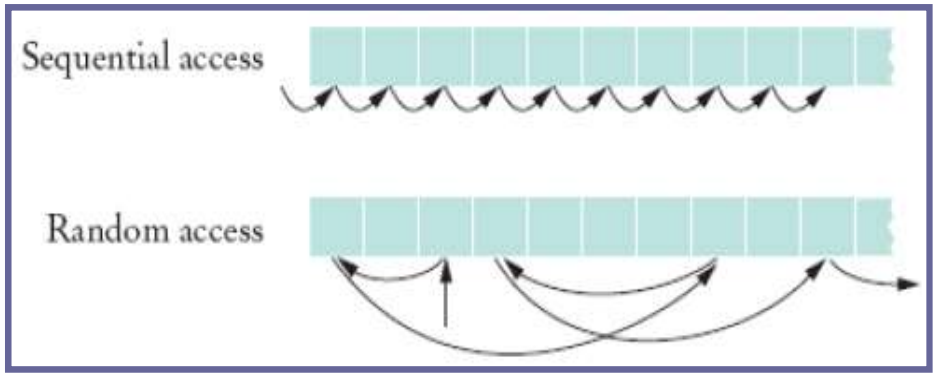
```
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Load numbers
6. Save numbers
7. Quit

Type in a number (1-7): 2
Add Name and Number
Name: bagher
Number: 386613

1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Load numbers
6. Save numbers
7. Quit

Type in a number (1-7): 1
Telephone Numbers:
Name:    Number: 0
Name: hossien   Number: 881216
Name: kamran    Number: 363131
Name: bagher    Number: 386613
Name: ali       Number: 774493
Name: kazem     Number: 521745
Name: sohael    Number: 323281
Name: kayvan    Number: 354418
```

Phone Book

```
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Load numbers
6. Save numbers
7. Quit

Type in a number (1-7): 6
Filename to save: phone.bin

1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Load numbers
6. Save numbers
7. Quit

Type in a number (1-7): 7
Goodbye
```

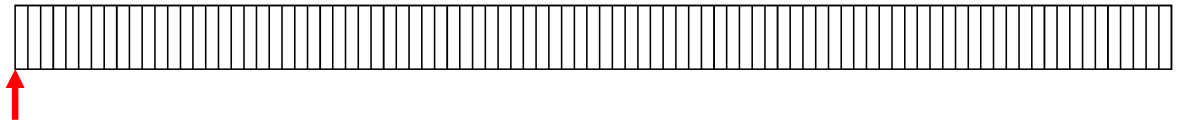
Random Access Files

- **Random access files are files in which records can be accessed in any order**
  - Also called direct access files
  - More efficient than sequential access files



Random Access File

- **Views a file as a sequential list of bytes**

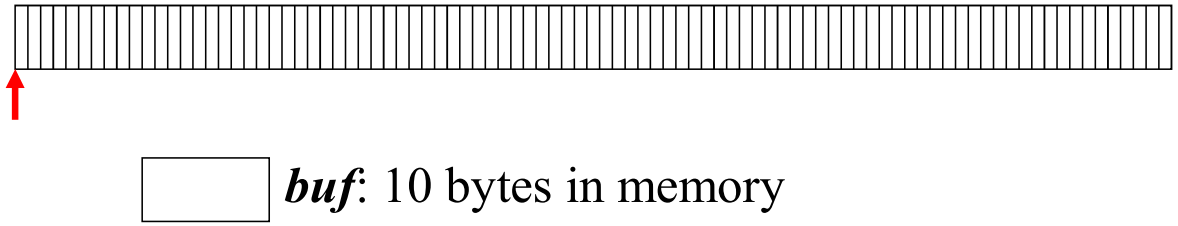


- **Has operators *seek, tell, read, write***

// In the file "./mydata", copy bytes 10-19 to 0-9.

Random Access File

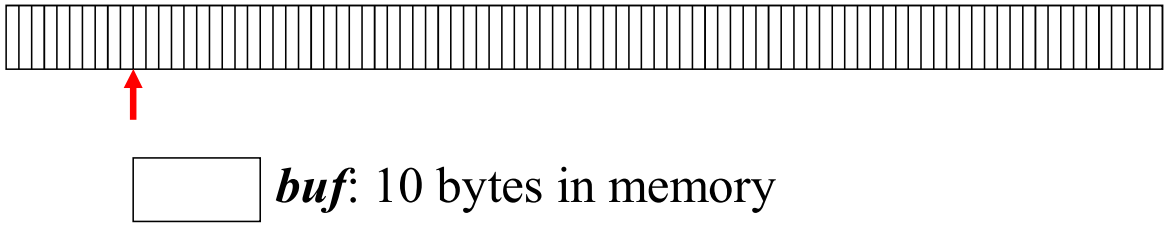
- **Views a file as a sequential list of bytes**




```
// In the file "./mydata", copy bytes 10-19 to 0-9.
file.seek(10);  buf=file.read(10);
file.seek(0);  file.write(buf);
```

Random Access File

- **Views a file as a sequential list of bytes**



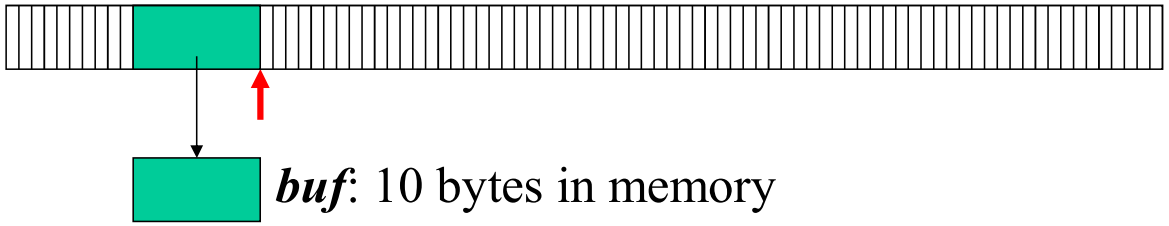
// In the file "./mydata", copy bytes 10-19 to 0-9.

file.seek(10); buf=file.read(10);

file.seek(0); file.write(buf);

Random Access File

- **Views a file as a sequential list of bytes**



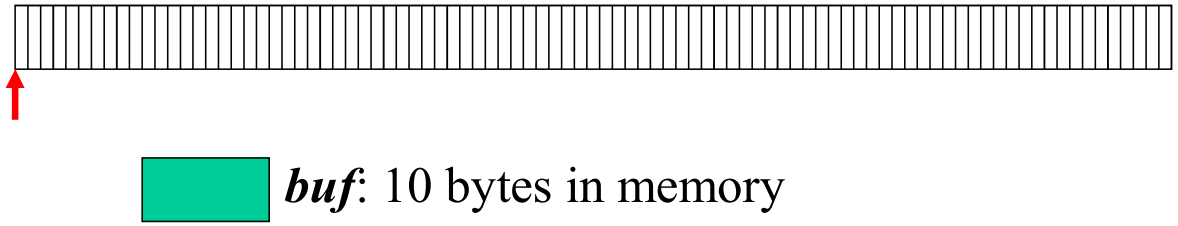
// In the file "./mydata", copy bytes 10-19 to 0-9.

file.seek(10); buf=file.read(10);

file.seek(0); file.write(buf);

Random Access File

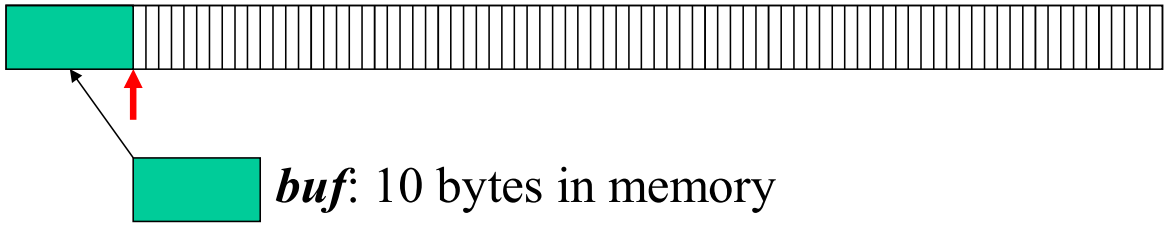
- **Views a file as a sequential list of bytes**




```
// In the file "./mydata", copy bytes 10-19 to 0-9.
file.seek(10);  buf=file.read(10);
file.seek(0);  file.write(buf);
```

Random Access File

- **Views a file as a sequential list of bytes**



// In the file "./mydata", copy bytes 10-19 to 0-9.

file.seek(10); buf=file.read(10);

file.seek(0); file.write(buf);

Functions

- **To move the file pointer to a specific byte**

  `f.seek(offset,[where])`

  offset -- This is the position of the read/write pointer within the file.

  where -- This is optional and defaults to 0 which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

- **To get current position of the file pointer.**

  `f.tell()`

  The *tell()* method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

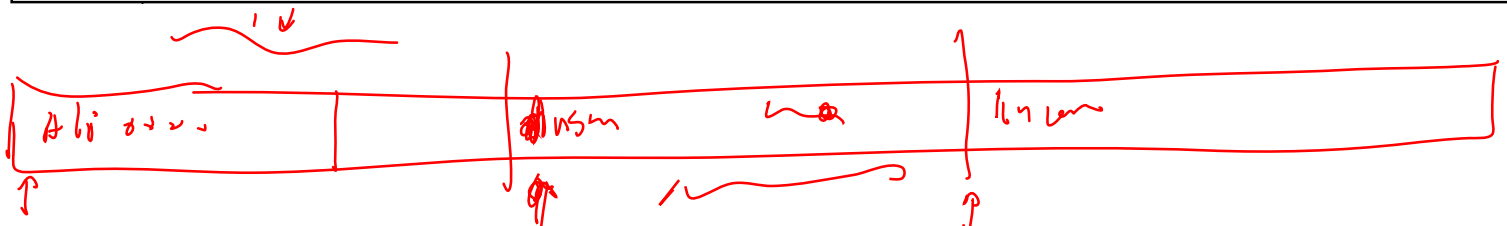
Phone Book

- Now here is a new version of the phone numbers program that we made earlier.
- This version uses a random access binary file for loading and saving the phone numbers

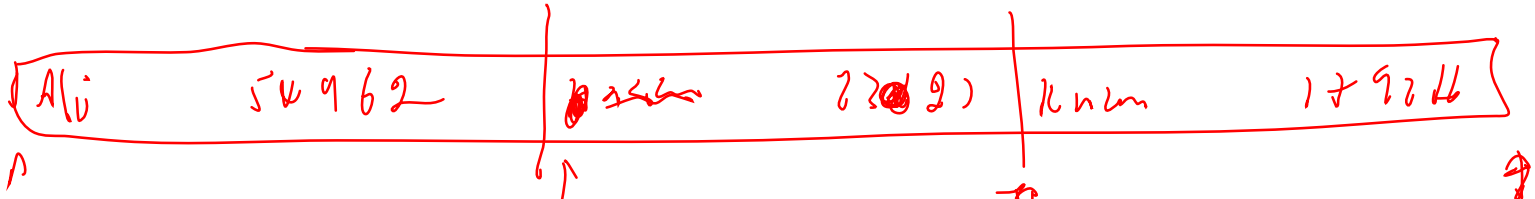
```
def print_menu():
    print()
    print('1. Print Phone Numbers')
    print('2. Add a Phone Number')
    print('3. Remove a Phone Number')
    print('4. Lookup a Phone Number')
    print('5. Change a Phone Number')
    print('6. Get by index')
    print('7. Quit')
    print()
    return

phone_list = {}
menu_choice = 0
file = open('data.bin', 'r+b')
while True:
    print_menu()
    menu_choice =int(input("Type in a number (1-7): "))
    if menu_choice == 1:
        print_numbers(file)
    elif menu_choice == 2:
        print("Add Name and Number")
        name = input("Name: ")
        phone =int(input("Number: "))
        save_number(file, name, phone)
    elif menu_choice == 3:
        print("Remove Name and Number")
        name =input("Name: ")
        remove_number(file, name)
    elif menu_choice == 4:
        print("Lookup Number")
        name = input("Name: ")
        print(lookup_number(file, name))
```

```
phone_list = {}
menu_choice = 0
file = open('data.bin', 'r+b')
while True:
    print_menu()
    menu_choice =int(input("Type in a number (1-7): "))
    if menu_choice == 1:
        print_numbers(file)
    elif menu_choice == 2:
        print("Add Name and Number")
        name = input("Name: ")
        phone =int(input("Number: "))
        save_number(file, name, phone)
    elif menu_choice == 3:
        print("Remove Name and Number")
        name =input("Name: ")
        remove_number(file, name)
    elif menu_choice == 4:
        print("Lookup Number")
        name = input("Name: ")
        print(lookup_number(file, name))
    elif menu_choice == 5:
        print("Change a phone number")
        name = input("Enter the name for changing: ")
        phone = int(input("Enter the new number: "))
        change_number(file, name, phone)
    elif menu_choice == 6:
        print("Get a phone number by index")
        index = int(input("Enter the index: "))
        get_number(file, index)
    elif menu_choice == 7:
        break
    else:
        continue

file.close()
print("Goodbye")
```

```python
def find_words(input):
    for i in range(0, len(input)):
        if input[i] == 0:
            return input[0:i]
    return ""

def print_numbers(file):
    print("Telephone Numbers:")
    file.seek(0, 0)
    while True:
        data=file.read(14)
        if not data:
            break
        unp=struct.unpack('>10si', data)
        name=find_words(unp[0]).decode('utf-8')
        if name!='':
            phone=unp[1]
            print("Name:", name, "\tNumber:", phone)
    return

def save_number(file, name, phone):
    file.seek(0, 0)
    while True:
        data=file.read(14)
        if not data:
            nameb=bytes(name.encode('utf-8'))
            packed = struct.pack('>10si', nameb, phone)
            file.write(packed)
            break
        unp=struct.unpack('>10si', data)
        namef=find_words(unp[0]).decode('utf-8')
        if namef=='':
            file.seek(-14, 1)
            nameb=bytes(name.encode('utf-8'))
            packed = struct.pack('>10si', nameb, phone)
            file.write(packed)
            break
    return
```



```python
def lookup_number(file, name):
    file.seek(0, 0)
    while True:
        data=file.read(14)
        if not data:
            break
        unp=struct.unpack('>10si', data)
        namef=find_words(unp[0]).decode('utf-8')
        if namef==name:
            return "The number is " + str(unp[1])
    return name + " was not found"

def remove_number(file, name):
    file.seek(0, 0)
    while True:
        data=file.read(14)
        if not data:
            break
        unp=struct.unpack('>10si', data)
        namef=find_words(unp[0]).decode('utf-8')
        if namef==name:
            nameb=b''; phone=0
            file.seek(-14, 1)
            packed = struct.pack('>10si', nameb, phone)
            file.write(packed)
            return
    print(name," was not found")
```



```python
def change_number(file, name, phone):
    file.seek(0, 0)
    while True:
        data=file.read(14)
        if not data:
            break
        unp=struct.unpack('>10si', data)
        namef=find_words(unp[0]).decode('utf-8')
        if namef==name:
            phoneb=phone
            file.seek(-14, 1)
            nameb=bytes(name.encode('utf-8'))
            packed = struct.pack('>10si', nameb, phoneb)
            file.write(packed)
            return
    print(name," was not found")

def get_number(file, index):
    file.seek(14*(index-1))
    data=file.read(14)
    if not data:
        print("index out of rang")
        return
    unp=struct.unpack('>10si', data)
    name=find_words(unp[0]).decode('utf-8')
    if name==':
       print("the index", index, "of file is empty")
    else:
        phone=unp[1]
        print(name, "with number:", phone, "is the", index, "record of file")
    return
```

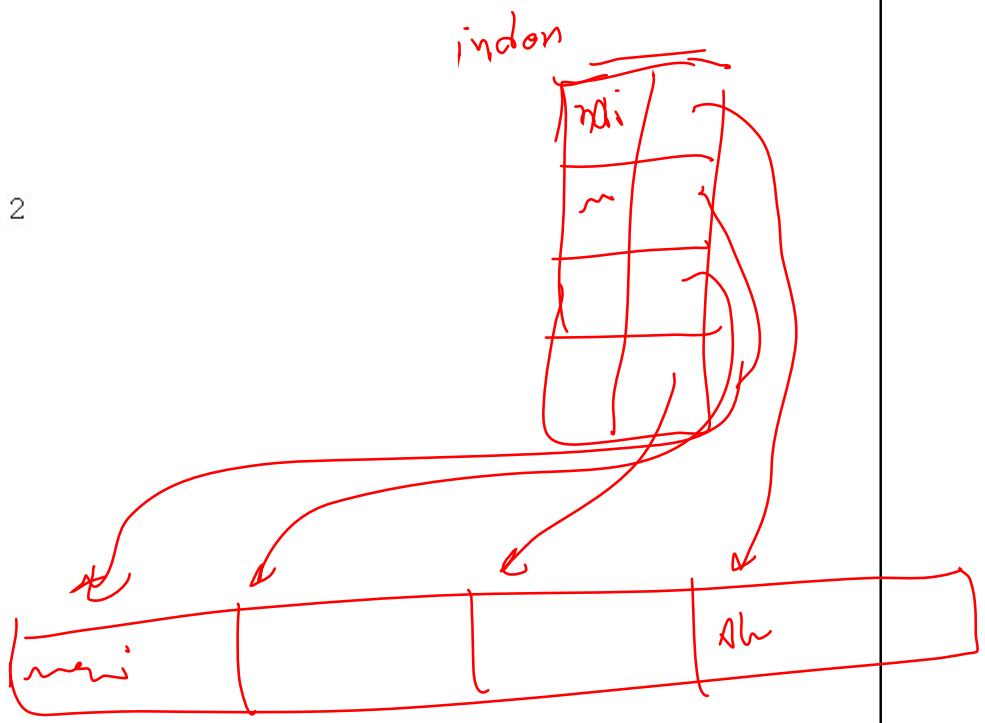
s file

```
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Change a Phone Number
6. Get by index
7. Quit
```

```
Type in a number (1-7): 2
Add Name and Number
Name: ali
Number: 774493
```

```
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Change a Phone Number
6. Get by index
7. Quit
```

```
Type in a number (1-7): 2
Add Name and Number
Name: hasan
Number: 443356
```



s file

```
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Change a Phone Number
6. Get by index
7. Quit

Type in a number (1-7): 1
Telephone Numbers:
Name: ali        Number: 774493
Name: hasan      Number: 443356
Name: kazem      Number: 521746
Name: kamran     Number: 363131
Name: kayvan     Number: 354418

Type in a number (1-7): 3
Remove Name and Number
Name: hasan

Type in a number (1-7): 1
Telephone Numbers:
Name: ali        Number: 774493
Name: kazem      Number: 521746
Name: kamran     Number: 363131
Name: kayvan     Number: 354418
```

```
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Change a Phone Number
6. Get by index
7. Quit

Type in a number (1-7): 2
Add Name and Number
Name: hossien
Number: 881216

1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Change a Phone Number
6. Get by index
7. Quit

Type in a number (1-7): 1
Telephone Numbers:
Name: ali        Number: 774493
Name: hossien    Number: 881216
Name: kazem      Number: 521746
Name: kamran     Number: 363131
Name: kayvan     Number: 354418
```

file

Random access file

```
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Change a Phone Number
6. Get by index
7. Quit

Type in a number (1-7): 2
Add Name and Number
Name: sohael
Number: 323281


Type in a number (1-7): 1
Telephone Numbers:
Name: ali        Number: 774493
Name: hossien    Number: 881216
Name: kazem      Number: 521746
Name: kamran     Number: 363131
Name: kayvan     Number: 354418
Name: sohael     Number: 323281
```

```
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Change a Phone Number
6. Get by index
7. Quit

Type in a number (1-7): 1
Telephone Numbers:
Name: ali        Number: 774493
Name: hossien    Number: 881216
Name: kazem      Number: 521746
Name: kamran     Number: 363131
Name: kayvan     Number: 354418
Name: sohael     Number: 323281

Type in a number (1-7): 5
Change a phone number
Enter the name for changing: kazem
Enter the new number: 521745

Type in a number (1-7): 1
Telephone Numbers:
Name: ali        Number: 774493
Name: hossien    Number: 881216
Name: kazem      Number: 521745
Name: kamran     Number: 363131
Name: kayvan     Number: 354418
Name: sohael     Number: 323281
```

```
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Change a Phone Number
6. Get by index
7. Quit

Type in a number (1-7): 4
Lookup Number
Name: kamran
The number is 363131


Type in a number (1-7): 6
Get a phone number by index
Enter the index: 3
kazem with number: 521745 is the 3 record of file
```

Directory management

If there are large number of files in Python, we can place related files in different directories to make things more manageable. A directory or folder is a collection of files and sub directories. Python has the **os** module, which provides us with many useful methods to work with directories (and files as well).

Get Current Directory

We can get the present working directory using the **getcwd()** method. This method returns the current working directory in the form of a string. We can also use the **getcwdb()** method to get it as bytes object.

```
>>> import os

>>> os.getcwd()
'C:\\Program Files\\PyScripter'

>>> os.getcwdb()
b'C:\\Program Files\\PyScripter'
```

The extra backslash implies escape sequence. The **print()** function will render this properly.

```
>>> print(os.getcwd())
C:\Program Files\PyScripter
```

Changing Directory

We can change the current working directory using the **chdir()** method. The new path that we want to change to must be supplied as a string to this method. We can use both forward slash (/) or the backward slash (\) to separate path elements. It is safer to use escape sequence when using the backward slash.

```
>>> os.chdir('C:\\Python33')

>>> print(os.getcwd())
C:\Python33
```

List Directories and Files

All files and sub directories inside a directory can be known using the **listdir()** method. This method takes in a path and returns a list of sub directories and files in that path. If no path is specified, it returns from the current working directory.

```
>>> print(os.getcwd())
C:\Python33

>>> os.listdir()
['DLLs',
'Doc',
'include',
'Lib',
'libs',
'LICENSE.txt',
'NEWS.txt',
'python.exe',
'pythonw.exe',
'README.txt',
'Scripts',
'tcl',
'Tools']

>>> os.listdir('G:\\')
['$RECYCLE.BIN',
'Movies',
'Music',
'Photos',
'Series',
'System Volume Information']
```

Making a New Directory

We can make a new directory using the **mkdir()** method. This method takes in the path of the new directory. If the full path is not specified, the new directory is created in the current working directory.

```
>>> os.mkdir('test')

>>> os.listdir()
['test']
```

Renaming a Directory or a File

The **rename()** method can rename a directory or a file. The first argument is the old name and the new name must be supplies as the second argument.

```
>>> os.listdir()
['test']

>>> os.rename('test','new_one')

>>> os.listdir()
['new_one']
```

Removing Directory or File

A file can be removed (deleted) using the **remove()** method. Similarly, the **rmdir()** method removes an empty directory.

```
>>> os.listdir()
['new_one', 'old.txt']

>>> os.remove('old.txt')
>>> os.listdir()
['new_one']

>>> os.rmdir('new_one')
>>> os.listdir()
[]
```

However, note that **rmdir()** method can only remove empty directories. In order to remove a non-empty directory we can use the **rmtree()** method inside the **shutil** module.

```
>>> os.listdir()
['test']

>>> os.rmdir('test')
Traceback (most recent call last):
...
OSError: [WinError 145] The directory is not empty: 'test'

>>> import shutil

>>> shutil.rmtree('test')
>>> os.listdir()
[]
```

End