Computing (ES 112)

Yogesh K. Meena
Shouvick Mondal

October 2024

Computer Science & Engineering
IIT Gandhinagar

Lists

Collection: multiple items together: Recall

https://www.clarehall.cam.ac.uk/bellcollection/

What is Not a “Collection”?: Recall

- Most of our variables have one value in them - when we put a new value in the variable, the old value is overwritten

```
$ python3
Python 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] on
 linux Type "help", "copyright", "credits" or "license" for more
 information.
>>> x=2
>>> x=4
>>> print(x)
4
```

List as a Collection:Recall

- A collection allows us to put many values in a single “variable”
- A collection is nice because we can carry all many values (even of different types) around in one convenient package.

```
>>> x = 2
>>> x = 4
>>> print(x)
4
>>> x = [1, 2.35, True, 'A', 5]
print(x)
>>> [1, 2.35, True, 'A', 5]
```

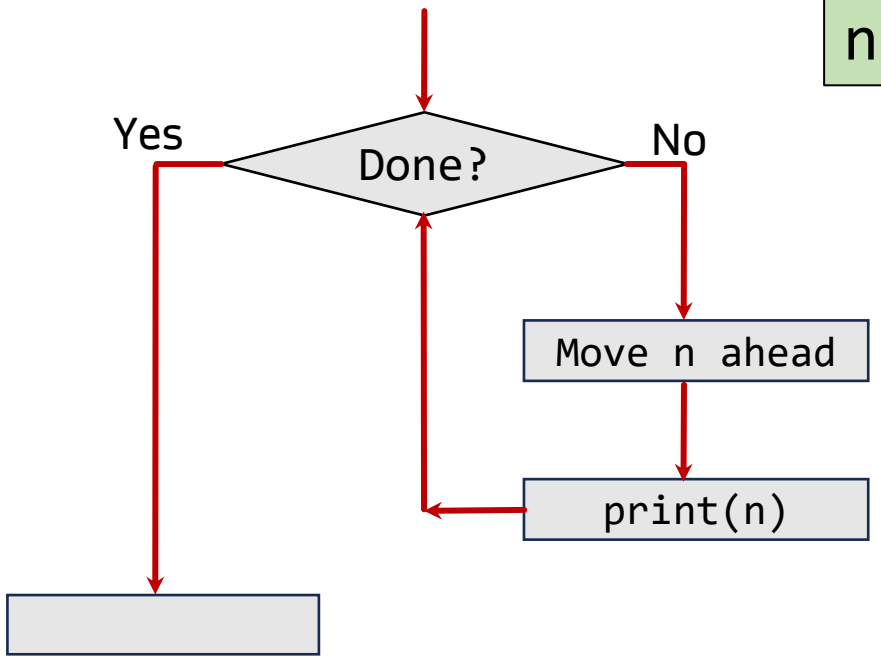
The indexing is exactly like strings. It can even go negative!

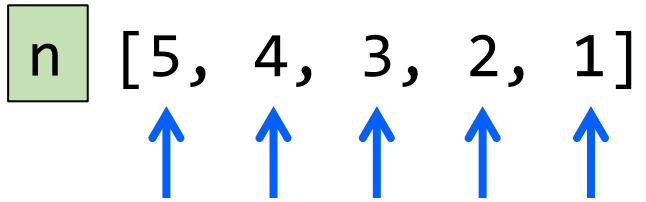
```
>>> print(x[0])
>>> print(x[-1])
5
x=[[],[]]
>>> print(len(x))
2
```

Why 2? Should it be zero?

Lists and Definite Loops: Recall

```
for n in [5, 4, 3, 2, 1]:
    print(n)
```





Output

5

4

3

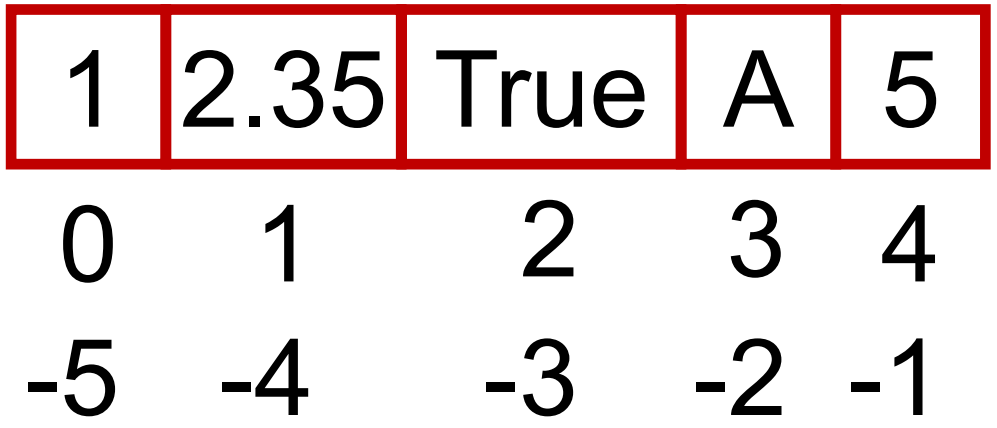
2

1

Definite loops (for loops) have explicit iteration variables that change each time through a loop. These iteration variables move through the collection of items in order.

List indexing and slicing: Recall

- Just like strings, we can get at any single element in a list using an index/slice specified in square brackets




```
>>> x = [1, 2.35, True, 'A', 5]
>>> print(x[-4])
2.35
>>> print(x[1:4])
[2.35, True, 'A']
>>> print(x[::-1])
[5, 'A', True, 2.35, 1]
>>> print(x[len(x)-1::-1])
[5, 'A', True, 2.35, 1]
```

Lists are Mutable: Recall

- Strings are “immutable” - we cannot change the contents of a string - we must make a new string to make any change
- Lists are “mutable” - we can change an element of a list using the index operator

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback
TypeError: 'str' object does not support item assignment
>>> lotto = [2, 14, 26, 41, 63]
>>> print(lotto)
[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print(lotto)
[2, 14, 28, 41, 63]
```

Using the range( ) function

- In Python 2.x, the range function returns a list of numbers that range from zero to one less than the parameter
- In Python 3.x, the range function returns a range type , i.e., a sequence of numbers that range from zero to one less than the parameter

```
>>> print(range(4))
[0, 1, 2, 3]
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(range(len(friends)))
[0, 1, 2]
```

```
>>> print(range(4))
range(0, 4)
>>> list(range(4))
[0, 1, 2, 3]
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(range(len(friends)))
range(0, 3)
```

Concatenation and Other Operations

- We can create a new list by concatenating two existing lists together using '+'
- **sum**() adds the elements in a list provided none of them are characters.
- **max**() finds the largest number in a list provided with the same condition as max

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> c = sum(a) + max(b)
>>> print(c)
12
>>> s=[1.2,3.4]
>>> sum(s)
4.6
```

concatenation

addition

Append and Extend Operations

- We can append to an existing list using append()
- We can extend an existing list using extend()

The . operator MUST be written before append() and extend()

```
>>> a = [1, 2, 3]
>>> a.append(5)
>>> print(a)
[1, 2, 3, 5]
```

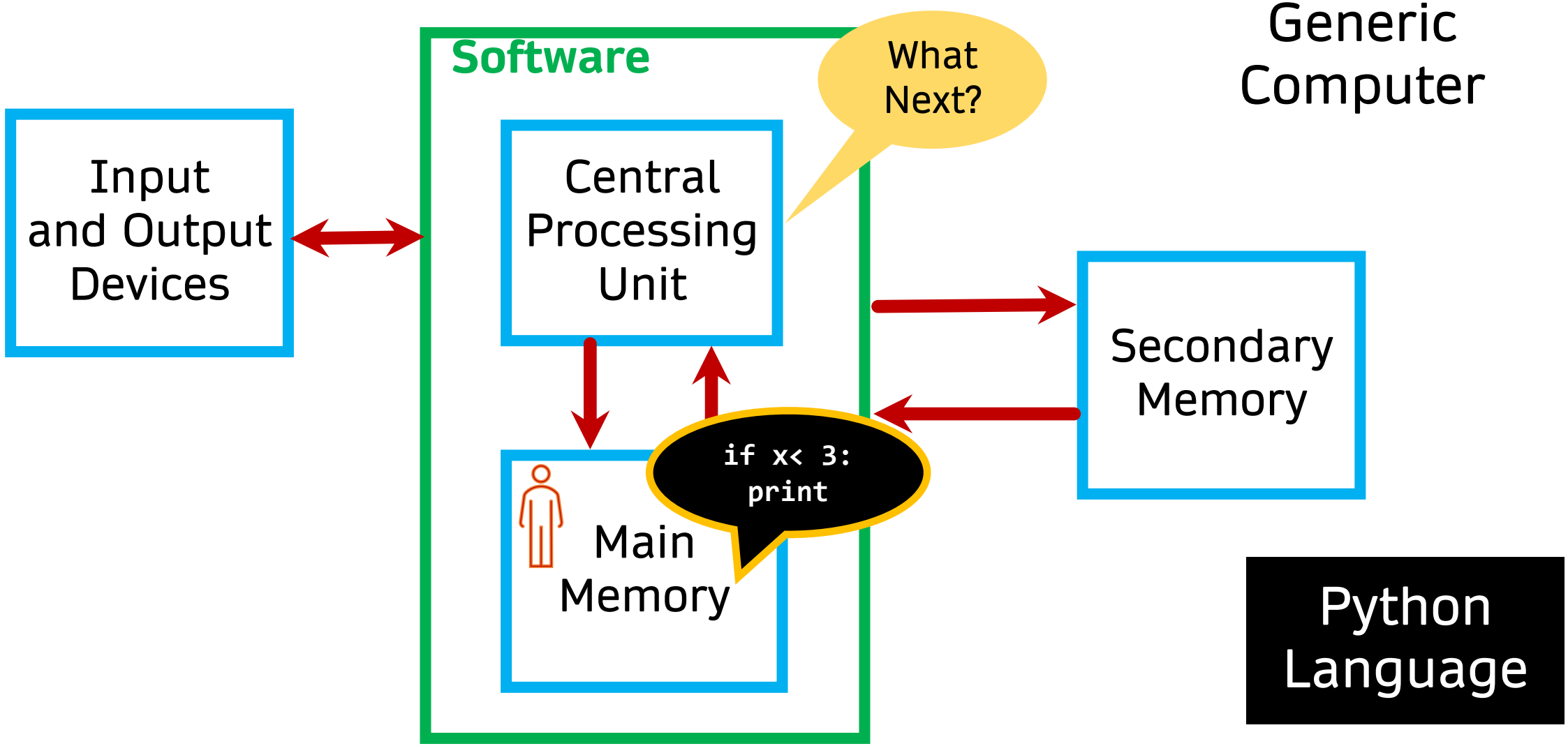
```
>>> a = [1, 2, 3]
>>> a.append([5])
>>> print(a)
[1, 2, 3, [5]]
```

```
>>> a = [1, 2, 3]
>>> a.extend(5)
Traceback (most recent call last): File
  "", line 1, in TypeError: 'int' object
  is not iterable
```

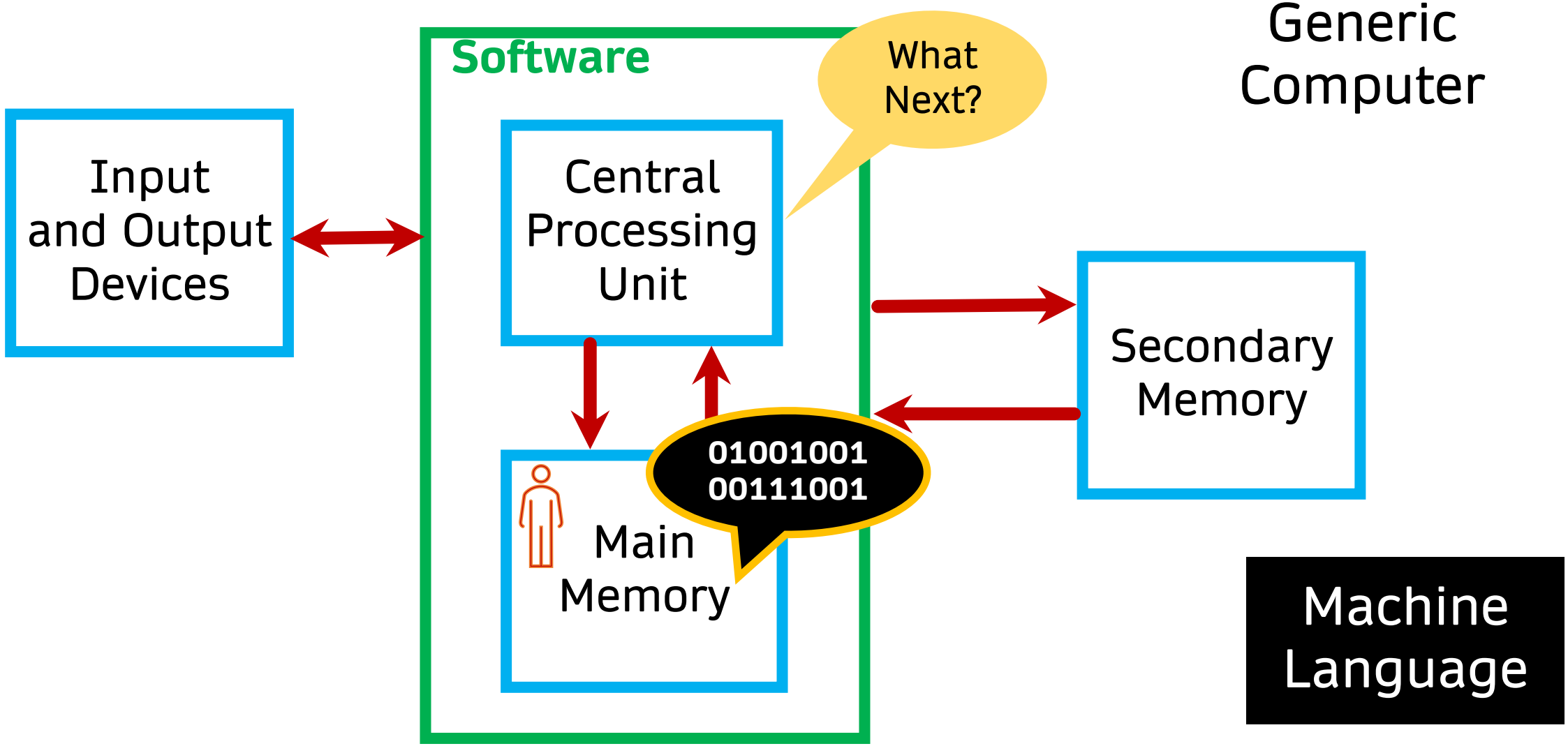
```
>>> a = [1, 2, 3]
>>> a.extend([5,1,2])
>>> print(a)
[1, 2, 3, 5, 1, 2]
```

Reading Files

Hardware Architecture: Recall

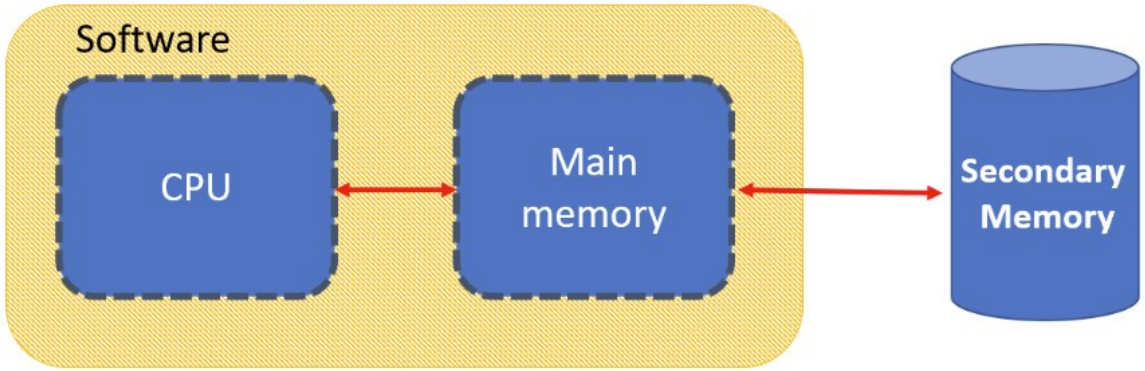


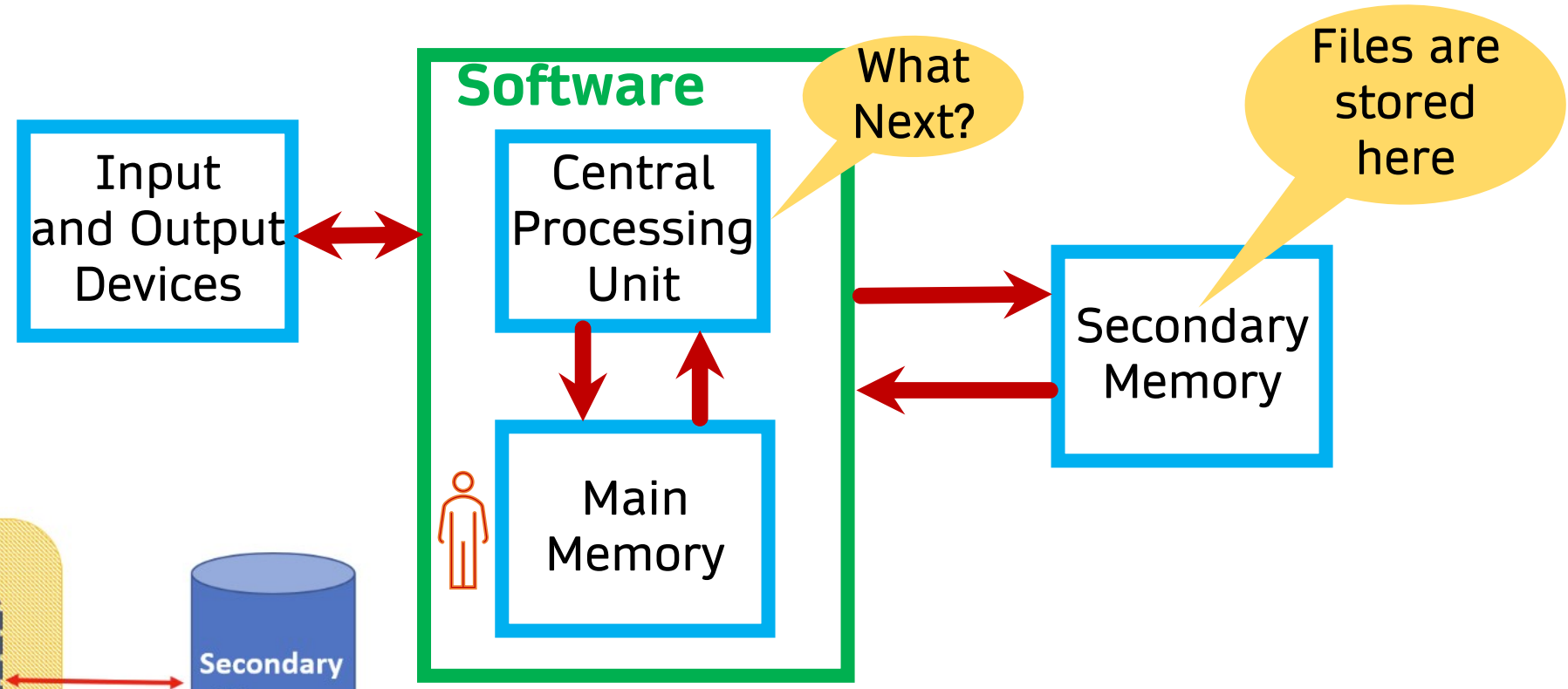
Hardware Architecture: Recall



Persistence and File Loading into the Main Memory

- Files are loaded from secondary memory to the main memory and then processed by the CPU. Once the processing is done, the data is written back to the secondary memory.



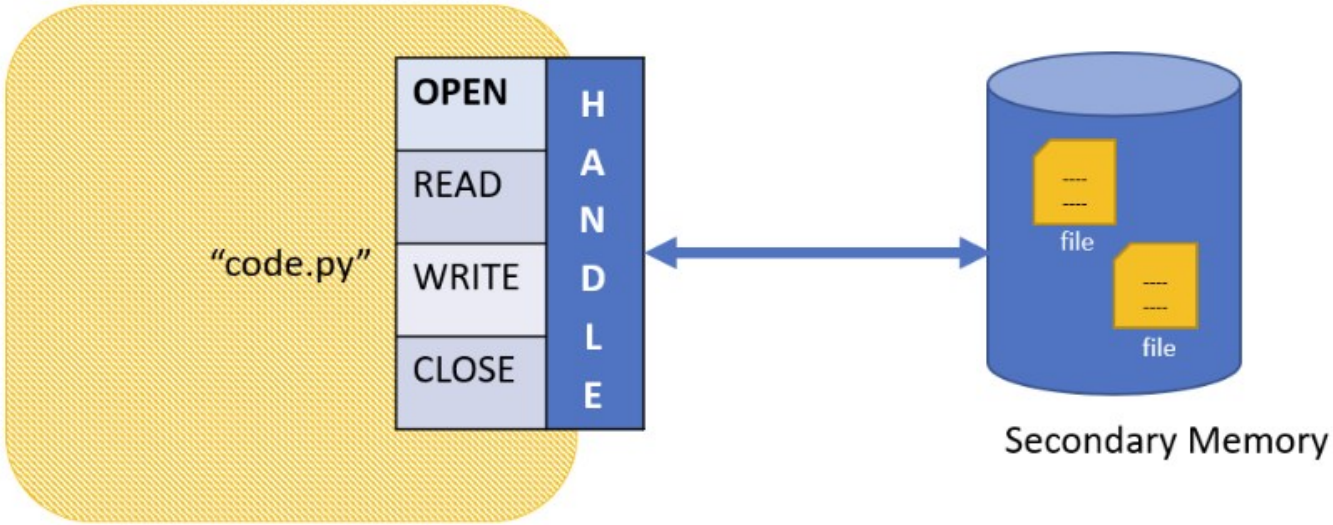


Opening a File

- Before we can read the contents of the file, we must tell Python which file we are going to work with and what we will be doing with the file
- This is done with the open() function
- open() returns a “file handle” - a variable used to perform operations on the file
- Similar to “File -> Open” in a Word Processor

Opening a File and How File Handlers Work

- handle = open(filename, mode)
- returns a handle use to manipulate the file
- filename is a string
- mode is optional and should be 'r' if we are planning to read the file and 'w' if we are going to write to the file




```
fhand = open('filename.txt', 'r')
```

File Handling: Open a file

- The OS will return the file handle in the variable fhand if open is successful

```
>>> fhand = open('clown.txt')
>>> print(fhand)
<_io.TextIOWrapper name='clown.txt' mode='r' encoding='UTF-8'>
```

fhand description

- `name` is the file name
- `mode` is the permission which is `r` (stands for read) in this case.
- `encoding` is the encoding mechanism for the Unicode character set.

File Handling: Open a file

- When files are missing `OSError` exception is raised.

```
>>> fhand = open(‘filename.txt’)
>>> print(fhand)
Traceback (most recent call last):
  File "/home/runner/readingFiles/main.py", line 1, in <module>
    fhand = open('iitgn.txt')
FileNotFoundError: [Errno 2] No such file or directory: 'iitgn.txt'
```

File Handling: Hands-On

```
#Open a file
fhand = open('clown.txt')
#Print the file whole at time one shot
fhand = open('clown.txt')
print(fhand.read())
#Print the file, one line at time (method 1)
# Get the file handler
fhand = open('words.txt')
# Loop through each line via file handler
for line in fhand:
      print(line)
```

File Handling: Hands-On

```
#Print the file, one line at time (method 2)
fhand = open('pattern.txt')
for line in fhand.readlines():
      print(line)
#fhand.readlines() returns a list of repr string representation
of the object (each line), example of repr() below
>>> h='\nhello\t'
>>> print(h)


hello
>>> print (repr(h))
'\nhello\t'
```

File Handling: Hands-On

```
#Print the file, multiple line stitched together
# Get the file handler
fhand = open('pattern.txt')
#Loop through each line via file handler
for line in fhand:
     #substitute \n at the end each line with empty character
     print(line,end="")
#Print the file one line at time and split the words per line by whitespace ('\n', '\t', ' ')
fhand = open('clown.txt’)
for line in fhand:
     print(line.split()) # split by whitespace
```

File Handling: Hands-On

```
#Print the file, one line at time with rstrip()
# Get the file handler
fhand = open('pattern.txt')
#Loop through each line and remove extra line character with
rstrip()
for line in fhand:
      line = line.rstrip()
      print('('+ repr(line) +')') #actual representation
```

File Handling: Hands-On

```
#How user can choose a file and count the number of lines in the file
fname = input('Enter the file name: ')
fhand = open(fname)
count = 0
for line in fhand:
      count = count + 1
print('There are', count, 'lines in', fname)
```

File Handling: Hands-On

```
# How user can choose a file and count the number of words in the file
fname = input('Enter the file name: ')
fhand = open(fname)
count = 0
for line in fhand:
      words = line.split()
      count = count + len(words)
print('There are', count, 'words in', fname)
```

File Handling: Hands-On

```
# How user can choose a file and count the number of character in the file
fname = input('Enter the file name: ')
fhand = open(fname)
count1 = 0
count2 = 0
for line in fhand:
      count1 = count1 + len(line) #including whitespace
      count2 = count2 + len(line.replace(' ','')) #remove space
print('There are', count1, 'characters in', fname)
print('There are', count2, 'characters in', fname)
```

File Handling: File Opening Modes

- Read Only ('r'): This mode opens the text files for reading only.
- Read and Write ('r+'): This method opens the file for both reading and writing. First read the file then write.
- Write Only ('w'): This mode opens the file for writing only.
- Write and Read ('w+'): This mode opens the file for both reading and writing. First overwrite the contents then read.

File Handling: Hands-On

```
# Write to a file
fname = input('Enter file name: ')
fhand = open(fname, 'w')
fhand.write('Now is the time for all good men to come to the aid
of their country.')
fhand.close()
# append to a file
fname = input('Enter file name: ')
fhand = open(fname, 'a')
fhand.write('Now is the time for all good men to come to the aid
of their country.')
fhand.close()
```

File Handling: Hands-On

```
# create new a file and write to it
# Open file with mode 'x'
fout = open('new-file.txt', 'x')
fout.write("Now the new file has some content!")
fout.close()
# If the file already exists then exception handling
```

File Handling: Hands-On

```
# example of exception handling
fname = input('Enter the file name: ')
try:
      fhand = open(fname)
except(OSError):
      print('File nout found and can not be opened:', fname)
      exit()
count=0
for line in fhand:
      count = count + 1
print('There are', count, 'lines in', fname)
```

File Handling: Hands-On

```
# delete entire data but not the file
f = open("iitgn.txt", "r+")
#it can be w, w+, r+ but not r (read only)
f.truncate()

# delete a whole file
import os
os.remove("new-file.txt")
```

File Handling: Hands-On

```
# reading without rstrip()
f=open('pattern.txt',mode='r')
print('reading without rstrip()')
while True:
      k=f.readline()
      if len(k) == 0: #reached EOF (end-of-file)
            break
      else:
            print(k)
f.close()
```

File Handling: Hands-On

```
# reading with rstrip(), strips whitespace from the right side of a string
f=open('pattern.txt',mode='r')
print('reading with rstrip()')
while True:
      k=f.readline().rstrip()
      if len(k) == 0: #reached EOF (end-of-file)
            break
      else:
            print(k)
f.close()
```

File Handling: Hands-On

```
# usage of tell() + readline(). The tell() method returns the
current file position in a file stream.
f=open('pattern.txt',mode='r')
print('usage of tell()')
while True:
      k=f.readline()
      if len(k) == 0: #reached EOF
            break
      else:
            print(f.tell()) #where is my cursor in the file?
f.close()
```

File Handling: Hands-On

```
# usage of tell() + readlines(). The tell() method returns the
current file position in a file stream.
f=open('pattern.txt',mode='r')
print('usage of tell()')
while True:
      k=f.readlines()
      if len(k) == 0: #reached EOF
            break
      else:
            print(f.tell()) #where is my cursor in the file?
f.close()
```

File Handling: Hands-On

```
# usgae of seek() + write(). Using this we can change the current file position with the seek() method.
f=open('pattern.txt',mode='r+')
while True:
        k=f.readline()
        if len(k) == 0: #reached EOF
                break
        else:
                z=f.tell()
                f.seek(z-2)
                f.write('%\n')
                f.seek(z+1)
f.seek(0)
print(f.read())
f.close()
```

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

Contributors 2024 - Yogesh K. Meena and Shouvick Mondal, IIT Gandhinagar