Programming For Data Science

Part Fourteen:

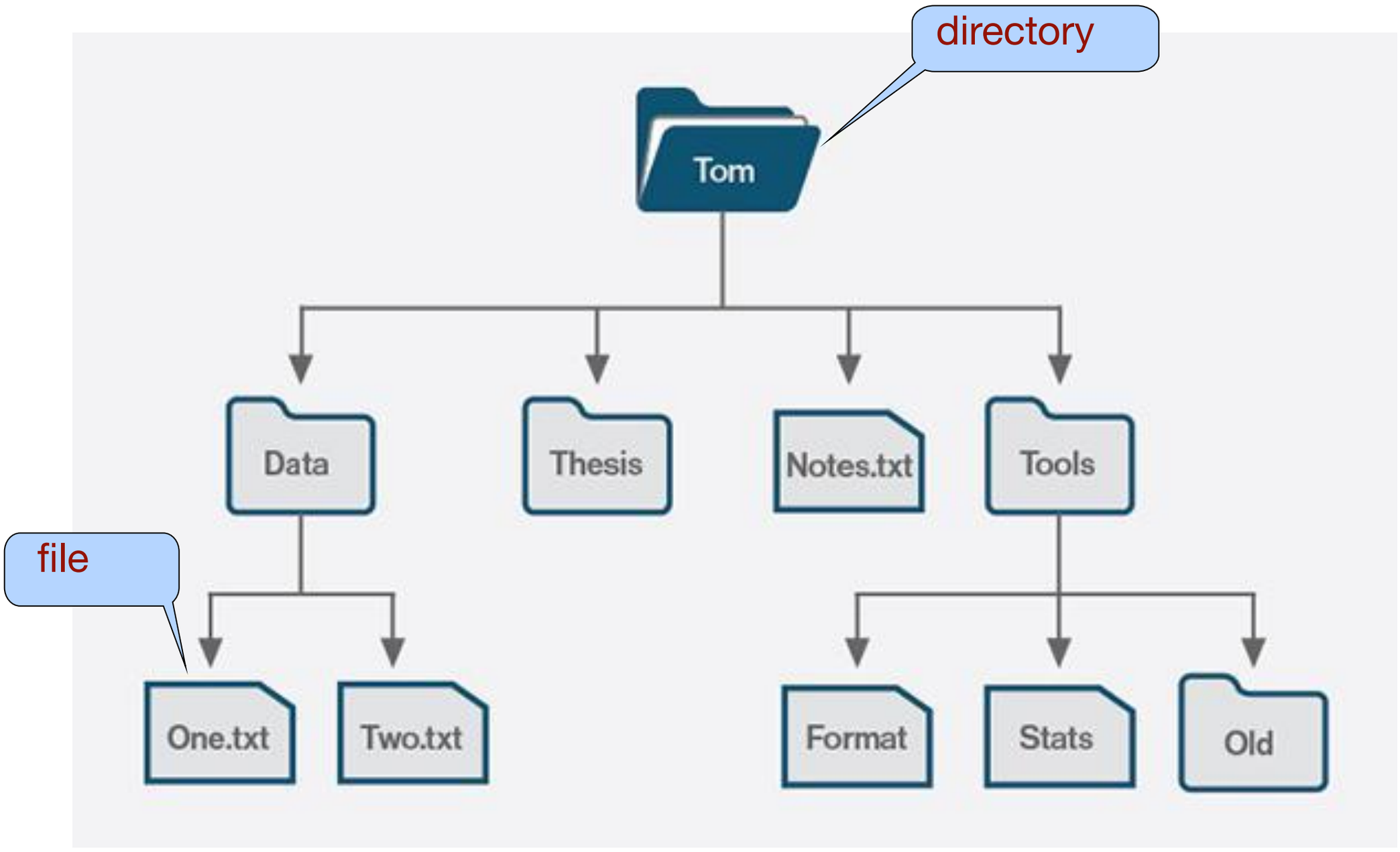
Text Files

Giulio Rossetti
giulio.rossetti@isti.cnr.it

Credits: Giuseppe Prencipe
giuseppe.prencipe@unipi.it

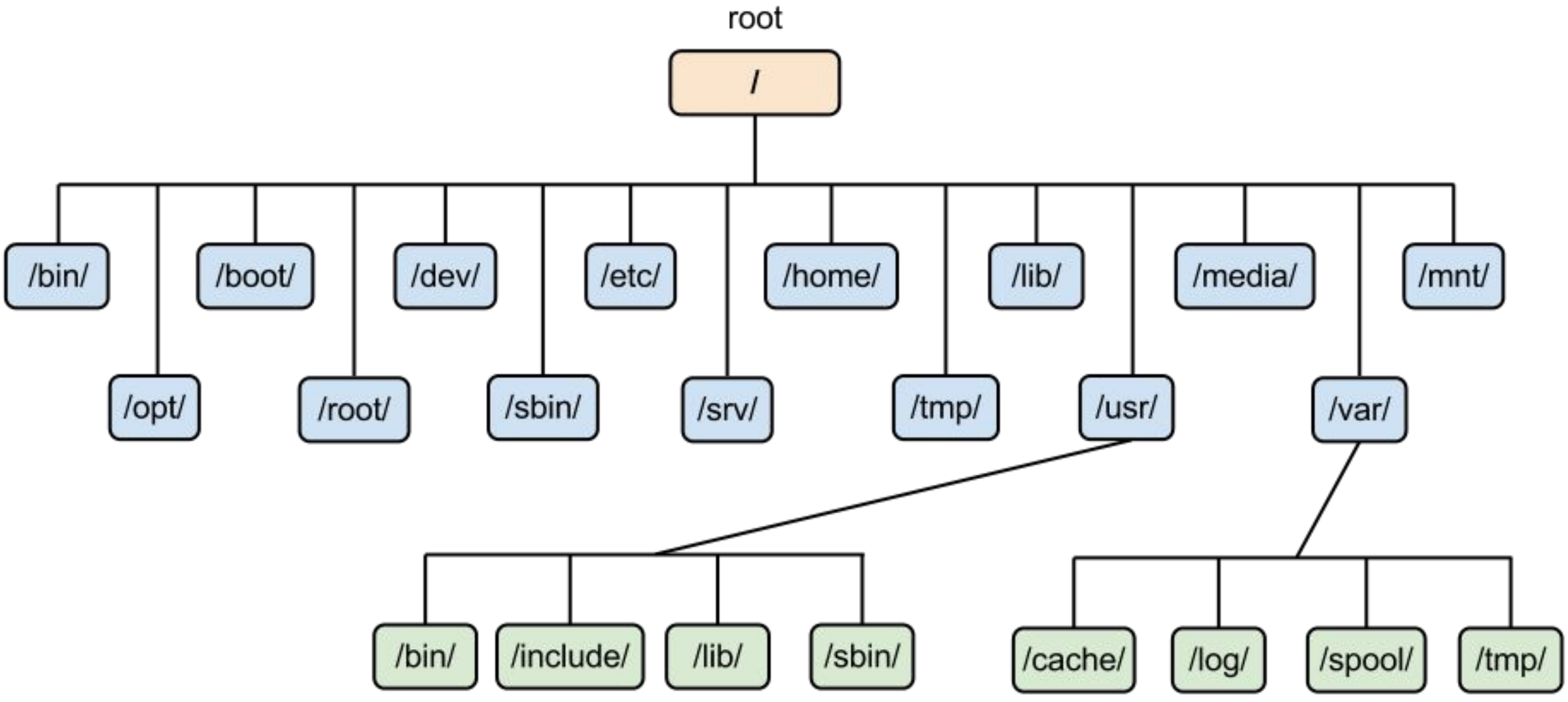
File System

- A computer's file system consists of a tree-like structured organization of directories and files



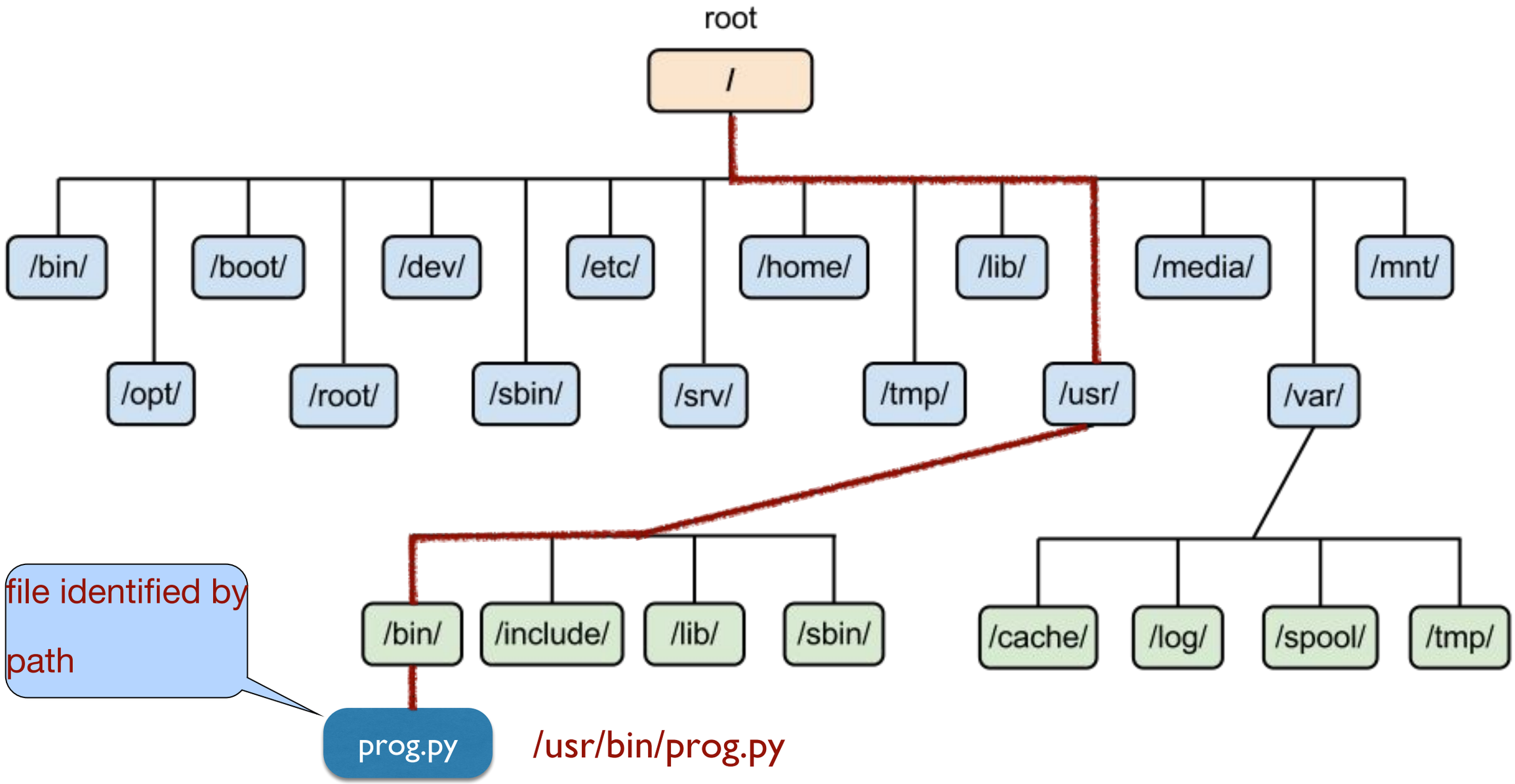
File System

- Each OS has its own restrictions on file names and extensions
  - In Linux, *extension* does not identifies the kind of file (info in the header of the file)



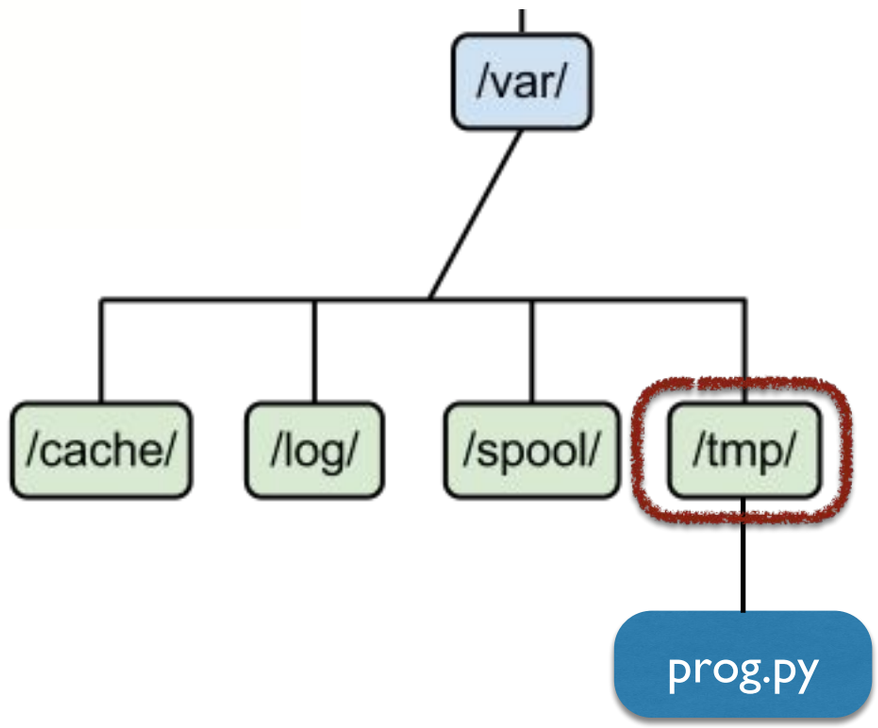
File System

- Each OS has its own restrictions on file names and extensions
  - In Linux, *extension* does not identifies the kind of file (info in the header of the file)



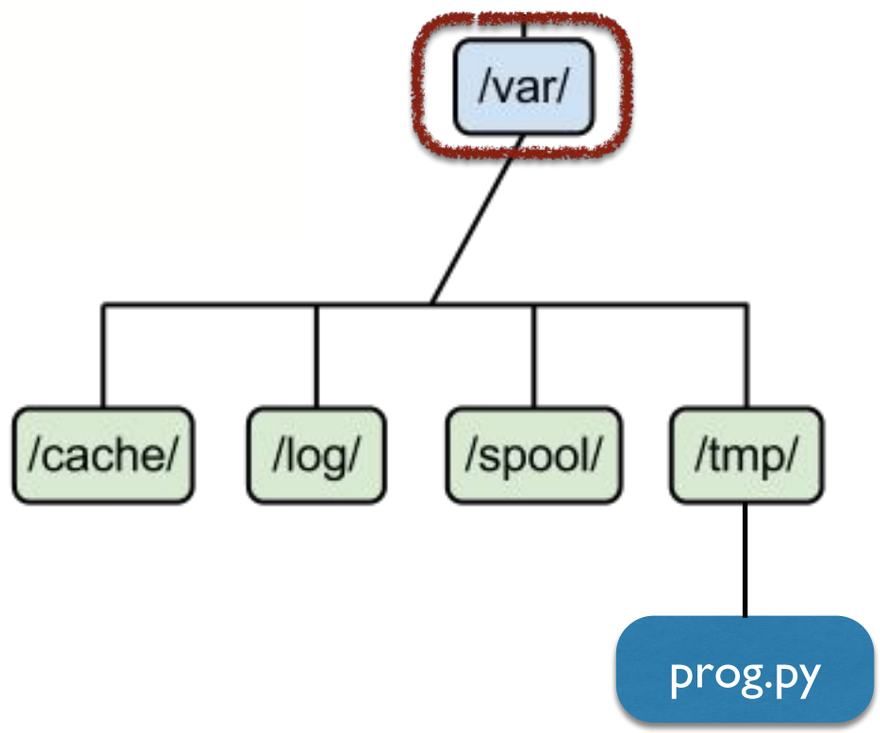
File system

- Current directory
  - Denoted by .
  - ./prog.py
- cd path —> changes the current directory

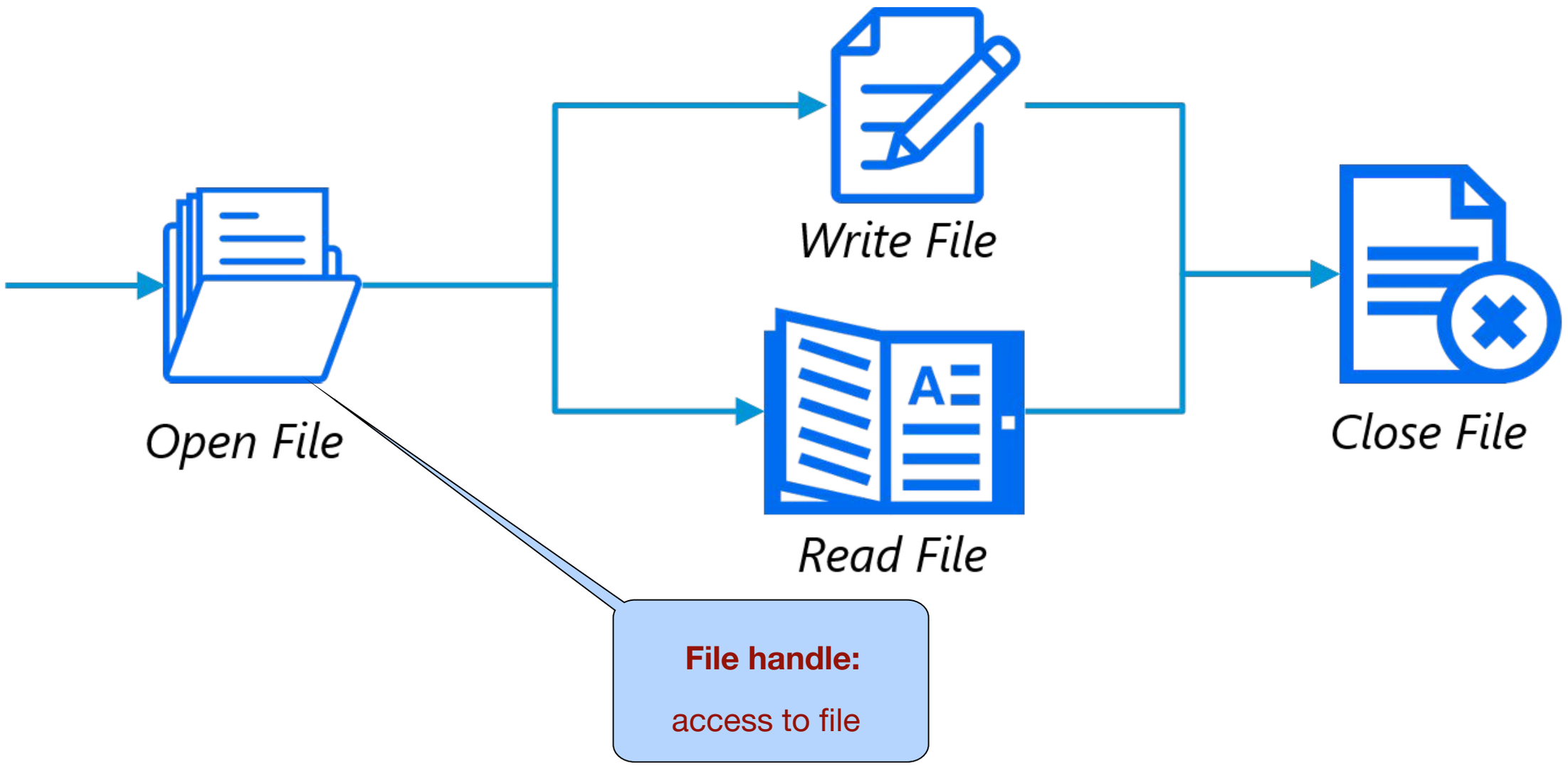


File system

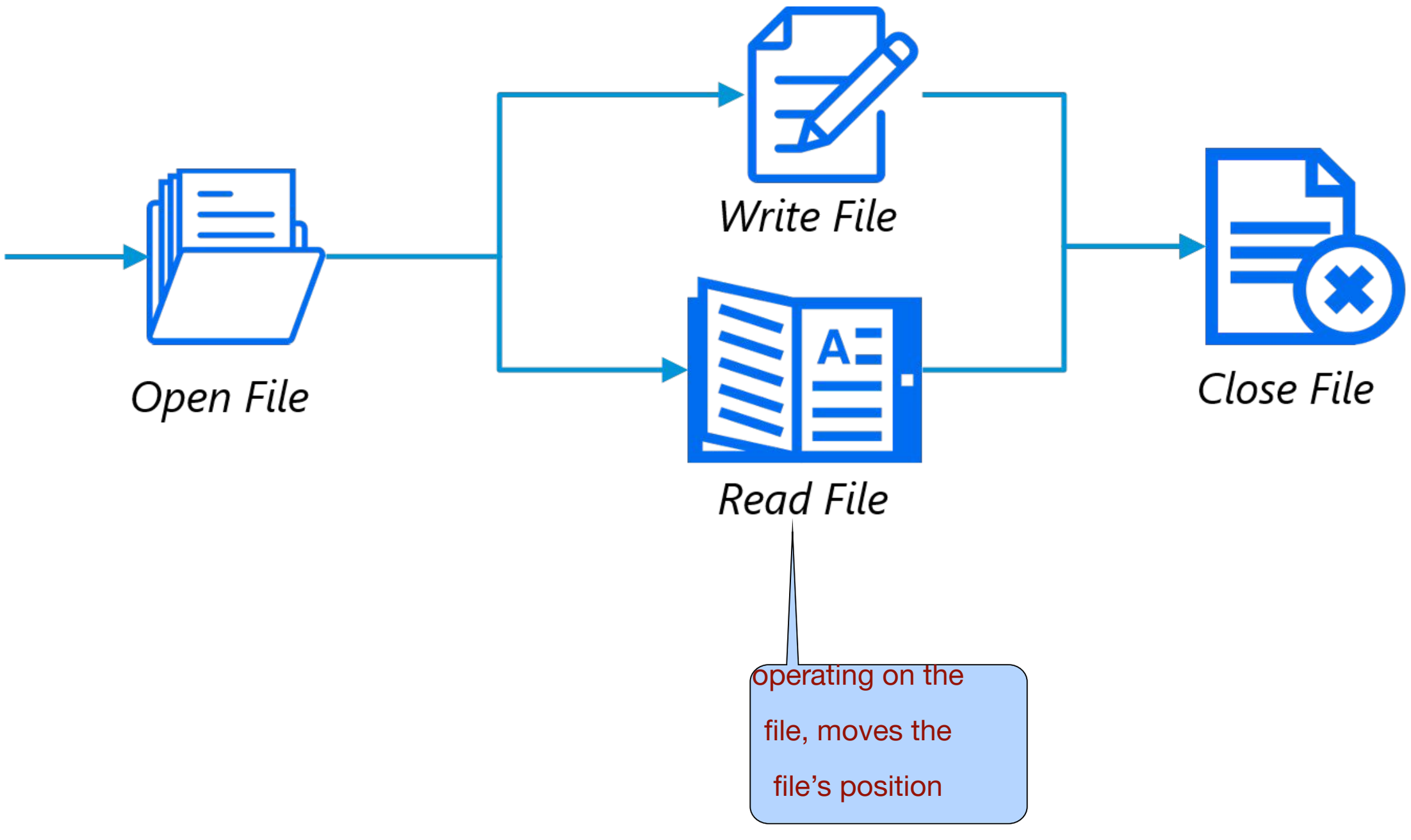
- Current directory
  - Denoted by .
  - ./prog.py
- cd path —> changes the current directory
- Parent directory
  - Denoted by ..
  - cd ..



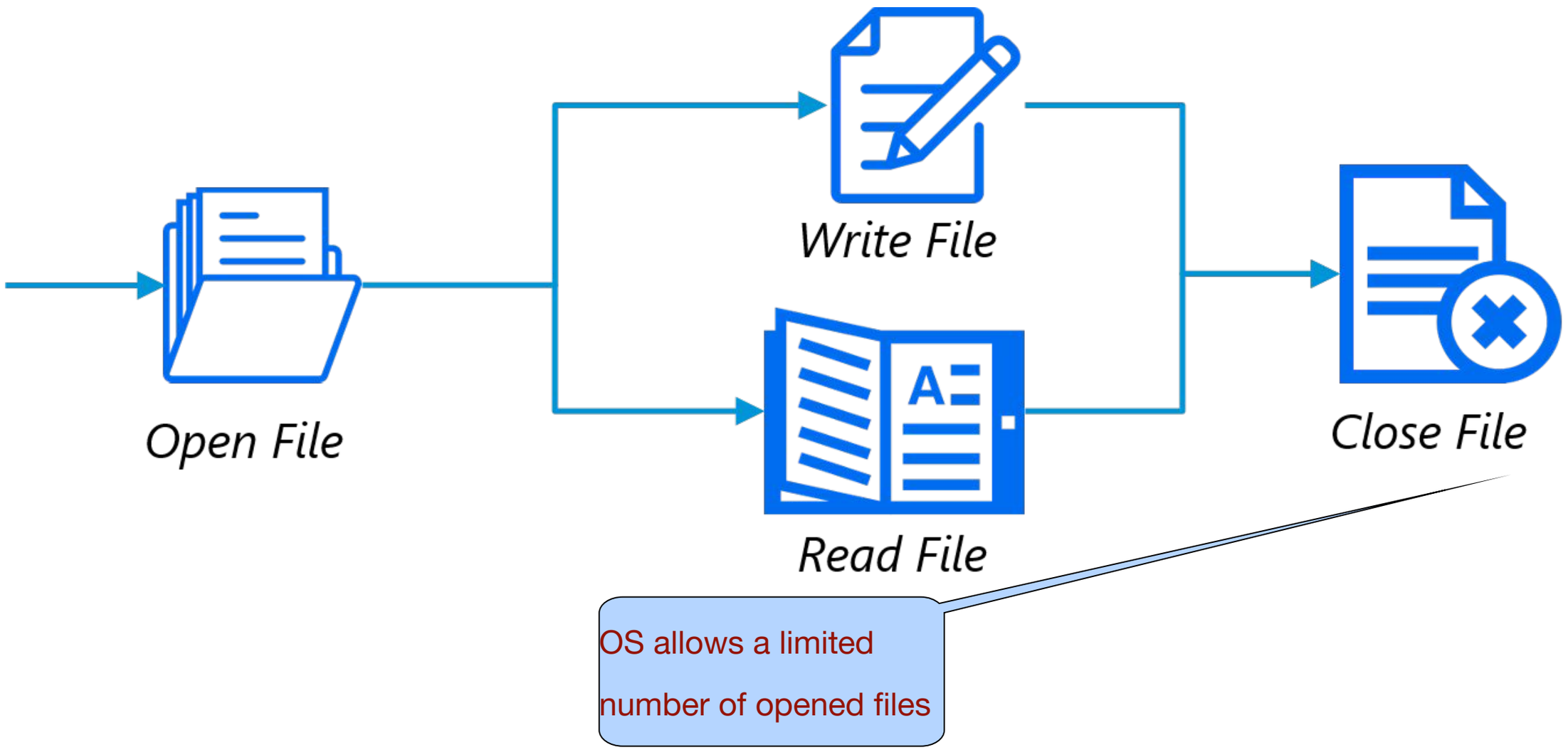
Working with files



Working with files

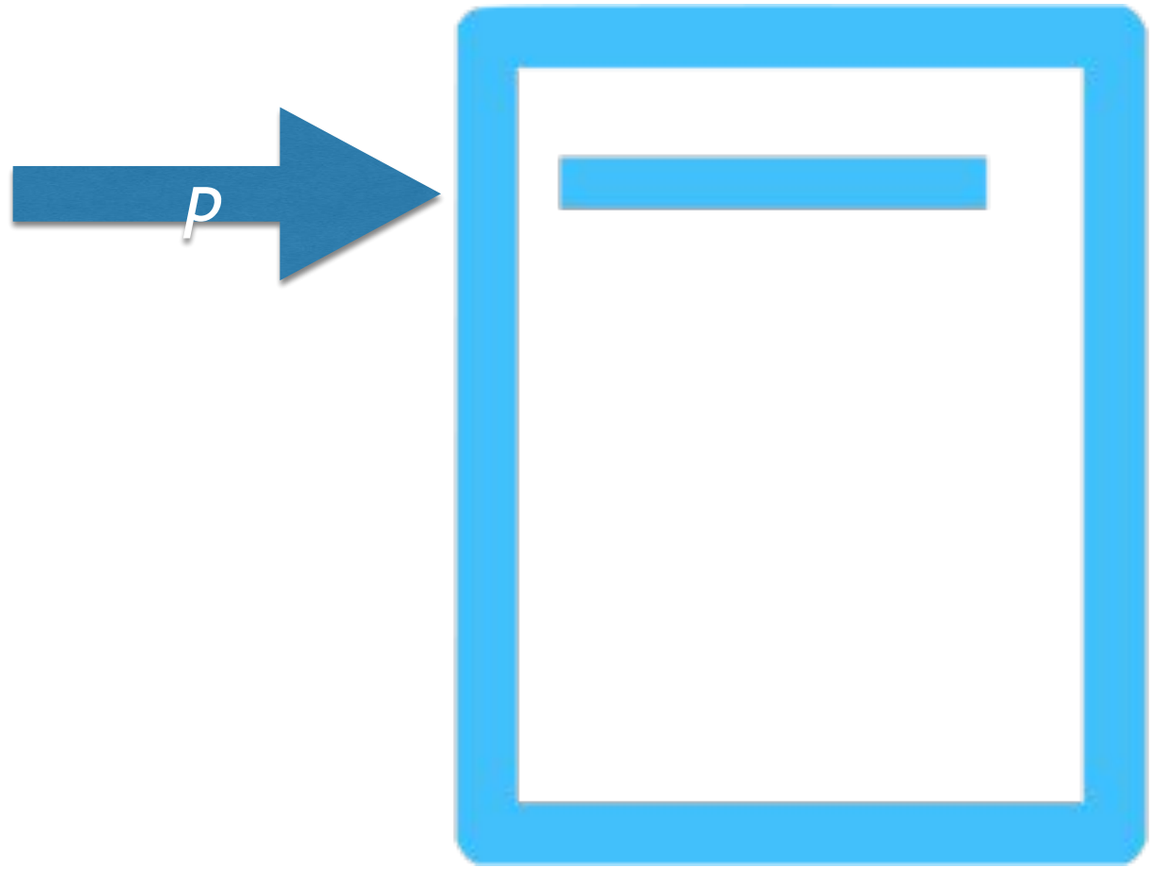


Working with files



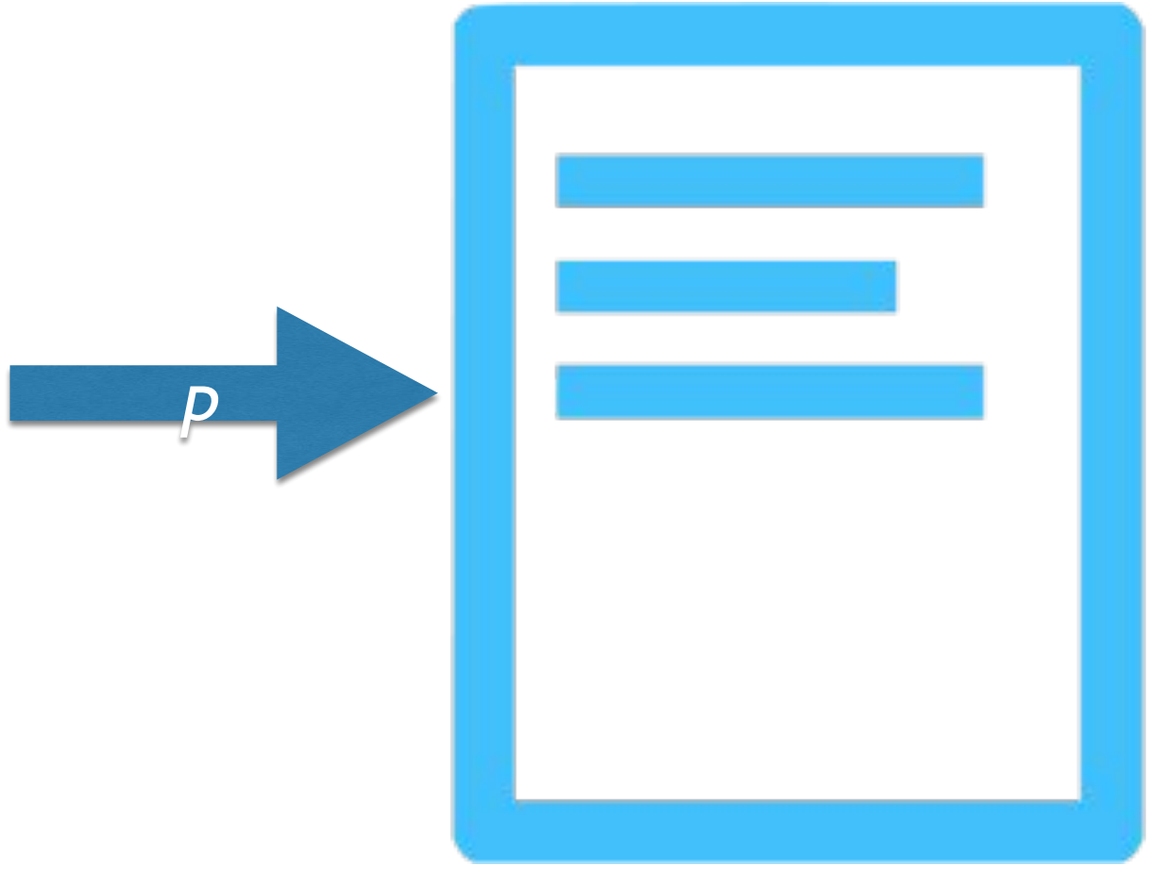
Position

- With the handler, you have access to the current position *p* in the file



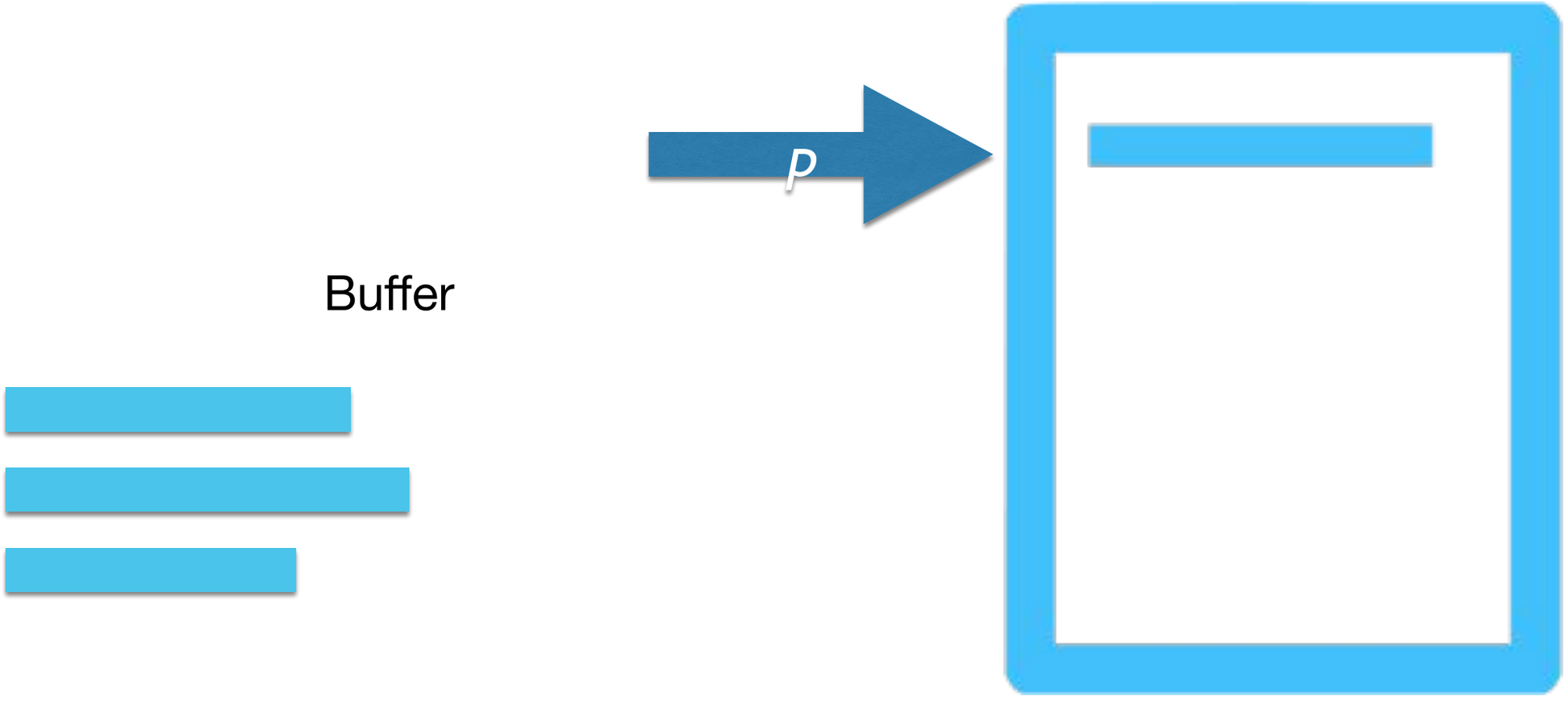
Position

- With the handler, you have access to the current position *p* in the file
- It is moved according to the operations performed on the file (or positioned manually)



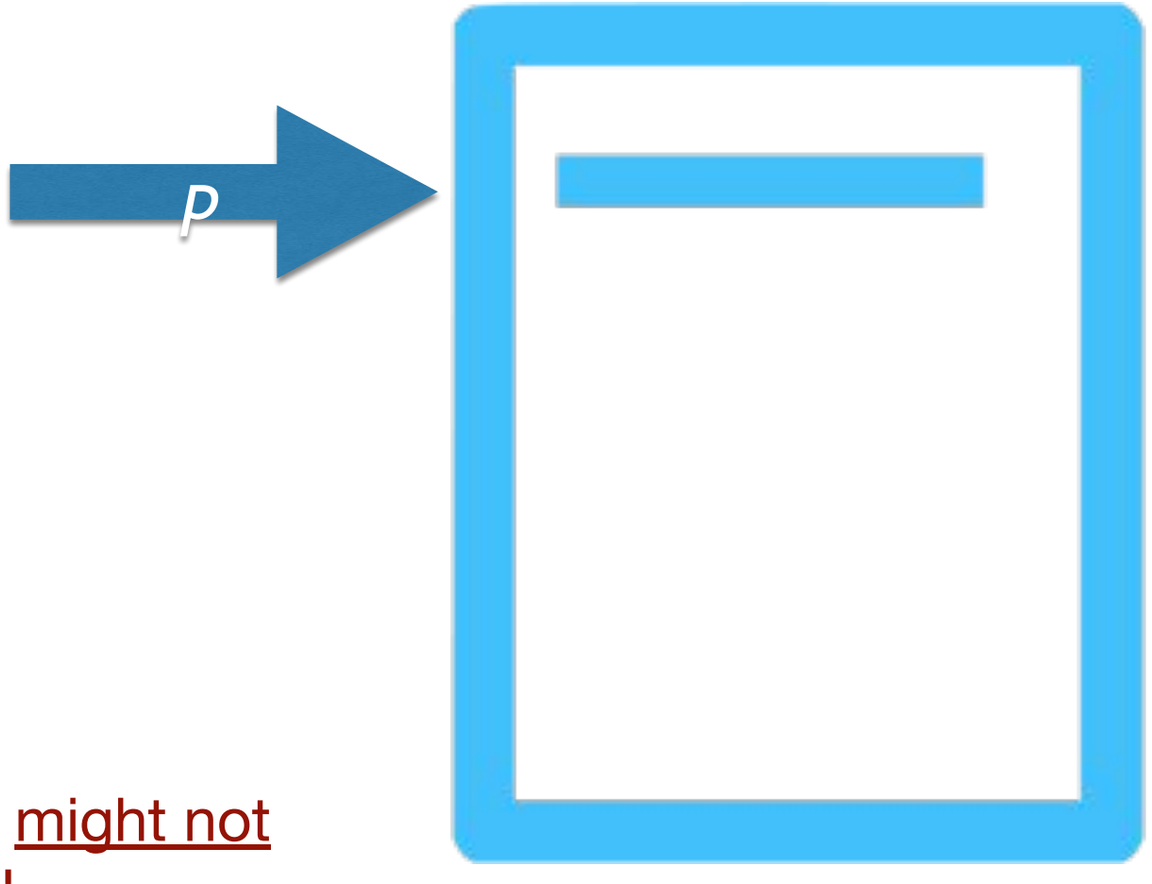
Buffering

- Data is not written immediately to file, but first placed in a zone of the memory (*buffer*)



Buffering

- Data is not written immediately to file, but first placed in a zone of the memory (*buffer*)
- OS flushes the buffer when it sees a need for that
  - Flush can be manual



Because of this mechanism, data might not be on file if your program crashes!

Text files

- open(name, [mode])
  - Opens the file name (either *current dir*, or *complete path*)
  - mode is optional (read, write, etc.)
  - Returns the handle

```
h = open(“pippo.txt”)

open (“pippo.txt”) as h
```

Text files

- read()
  - Given a handle, reads the content of the file
  - In its simplest form, returns the complete content of the file as a string
    - Moves the pointer to the end

```
h = open("pippo.txt")
print(h.read())
print(h.read())
```

What do i get here?

Text files

- read()
  - Given a handle, reads the content of the file
  - In its simplest form, returns the complete content of the file as a string
    - Moves the pointer to the end

```
h = open("pippo.txt")
print(h.read())
print(h.read())
```

Second string is empty

(pointer moved to end)

Text files

- close()
  - Closes the file (using its handle), and releases the handle

```
h = open(“pippo.txt”)
print(h.read())
h.close()
```

Text files

- Alternative syntax

```
with open("pippo.txt") as h:
    buf = h.read()


print(buf)
```

h closed automatically after block

Text files

- readline()
  - This method reads line by line
    - From current pointer's position up to (and including) next newline

```
h = open("pippo.txt")
while True:
    buf = h.readline()
    if buf == "":
        break
    print(buf)
h.close()
```

You will see an empty line after each line….why?

Text files

- readline()
  - This method reads line by line
    - From current pointer’s position up to (and including) next newline

```
h = open(“pippo.txt”)
while True:
    buf = h.readline()
    if buf == “”:
        break
    print(buf)
h.close()
```

One empty from the readline()

One empty from the print()

Text files

- readline()
  - This method reads line by line
    - From current pointer's position up to (and including) next newline

```
h = open("pippo.txt")
while True:
    buf = h.readline()
    if buf == "":
        break
    print(buf)
h.close()
```

How can you avoid this?

Text files

- readline()
  - This method reads line by line
    - From current pointer’s position up to (and including) next newline

```
h = open(“pippo.txt”)
while True:
    buf = h.readline()
    if buf == “”:
        break
    print(buf, end = “”)
h.close()
```

Like this!

Text files

- readlines()
  - Reads all lines, returning them in a list of strings
    - From current pointer’s position up to (and including) next newline

```
h = open(“pippo.txt”)
buf = h.readlines()


h.close()
```

Text files

- open(name, “w”)
  - Opens the file in writing mode
    - If file exists, its content is automatically deleted

```
h = open(“pippo.txt”, “w”)
```

Text files

- write(string)
  - Given an handle, writes string to the file
    - No newline is added at the end of the string

```
h = open(“pippo.txt”, “w”)
h.write(“Hello World!”)
```

Text files

- writelines(string_list)
  - Writes the list of strings to the file
    - No newline is added at the end of each of the strings

```
h = open(“pippo.txt”, “w”)
h.writelines([“Hello”, “ ” , “World”, “!”])
```

Text files

- open(name, “a”)
  - Opens the file in append mode
    - If file exists, its content is not deleted, and new content is appended at the end

```
h = open(“pippo.txt”, “a”)
```

Text files

- Useful functions to interact with the file system in the os module
  - getcwd()
  - chdir(new_dir)
  - listdir(dir)
  - system(command)

Current working dir

Text files

- Useful functions to interact with the file system in the os module
  - getcwd()
  - chdir(new_dir)
  - listdir(dir)
  - system(command)

Changes dir

Text files

- Useful functions to interact with the file system in the os module
  - getcwd()
  - chdir(new_dir)
  - listdir(dir)
  - system(command)

Returns a list of all file and dirs in dir

Text files

- Useful functions to interact with the file system in the os module
  - getcwd()
  - chdir(new_dir)
  - listdir(dir)
  - system(command)

Text files

- Useful functions in the os.path module
  - exists(path)
  - isfile(path)
  - isdir(path)
  - dirname(path)
  - getsize(path)

Text files

- Useful functions in the os.path module
  - exists(path)
  - isfile(path)
  - isdir(path)
  - dirname(path)
  - getsize(path)

Text files

- We already talked about characters' encoding
  - ASCII and UTF-8
- getfilesystemencoding() returns the preferred encoding system used by your system

```
from sys import getfilesystemencoding


print(getfilesystemencoding())
```

Text files

- We already talked about characters' encoding
  - ASCII and UTF-8
- getfilesystemencoding() returns the preferred encoding system used by your system
- You can specify the encoding of a file you open in the open() call

```
h = open("pippo.txt", encoding="ascii")
```

Text files

- We already talked about characters' encoding
  - ASCII and UTF-8
- getfilesystemencoding() returns the preferred encoding system used by your system
- You can specify the encoding of a file you open in the open() call
  - Opening a file with the wrong encoding might rise an exception

```
h = open("pippo.txt", encoding="ascii")
```

(Suggested) Exercises

- Exercise 15.1 in the reference book
- **All** exercises in Chapter 16 of the reference book

Programming For Data Science

Part Fifteen:

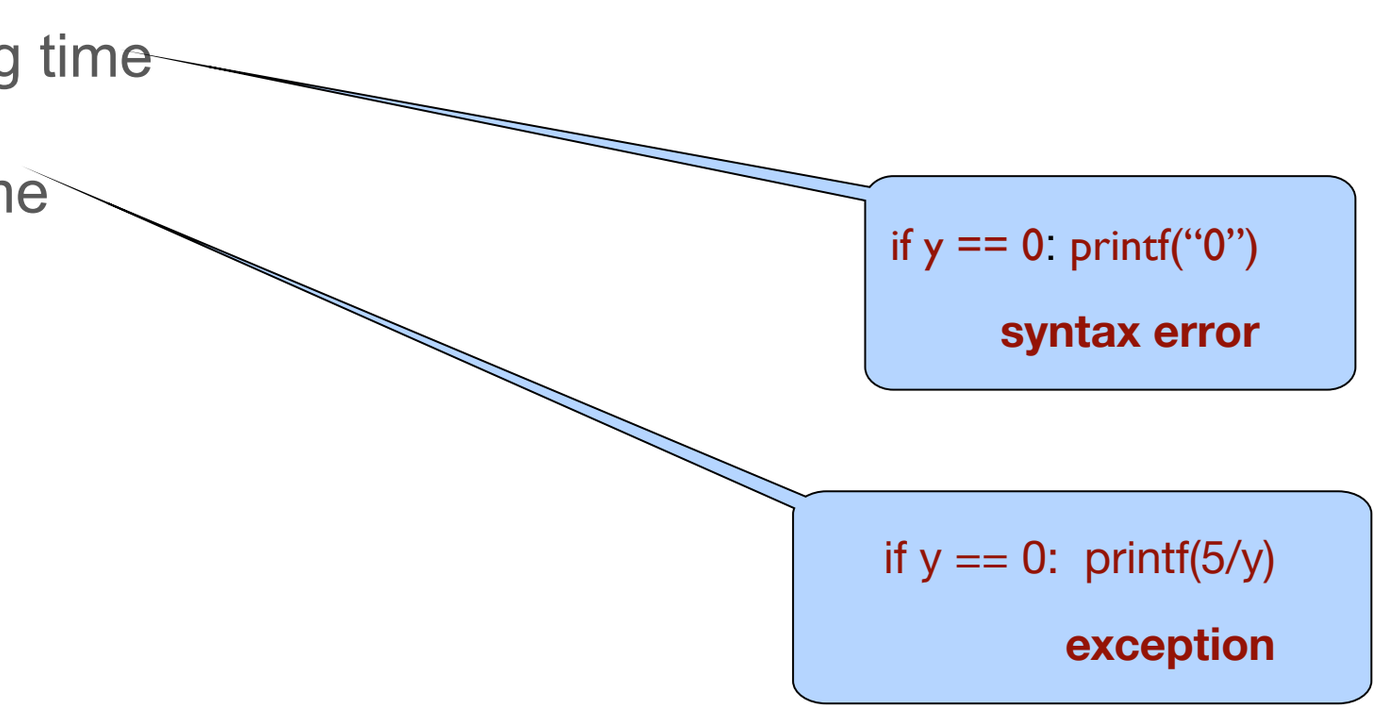
Exceptions

Giulio Rossetti
giulio.rossetti@isti.cnr.it

Credits: Giuseppe Prencipe
giuseppe.prencipe@unipi.it

Errors

- Sometimes sh….errors happens!
  - At writing time
  - At runtime



```
In [1]: 1/0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1-05c9758a9c21> in <module>()
----> 1 1/0
ZeroDivisionError: integer division or modulo by zero

In [2]: d = {}
In [3]: print d[10]
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-3-2a1aa1081c87> in <module>()
----> 1 print d[10]
KeyError: 10

In [4]: l = [1,2,3]
In [5]: print l[42]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-5-cac02ccb7a63> in <module>()
----> 1 print l[42]
IndexError: list index out of range
```

Exceptions

- Exceptions can be captured

Exceptions...

Gotta catch 'em all!

Exceptions

- Exceptions can be captured

```
try:
    # block
except:
    # exception handling
```

handling executed if exception raised in block

Exceptions

- Exceptions can be captured

```
num = getInteger(“Give me a number: ”)
try:
    print (3/num)
except:
    print(“Division by zero!”)
print(“end”)
```

Exceptions

- You can address specific exceptions

```
try:
    print (3/int(input("Give me a number: ")))
except ZeroDivisionError:
    print("Division by zero!")
except ValueError:
    print("Not an integer!")
except:
    print("Something else went wrong!")
```

Exceptions

- else clause

```
try:
    num = 3/int(input(“Give me a number: ”))
except ZeroDivisionError:
    print(“Division by zero!”)
except ValueError:
    print(“Not an integer!”)
else:
    print(num)
```

Executes only if no exception at all occurs

Exceptions

- finally clause
  - Executed regardless of how the `try` clause is exited

```
try:
    h = open("pippo.txt")
    print (h.read())
finally:
    h.close()
```

Makes sure the file is always closed

Exceptions extra info

- except…. as name clause

```
try:
    num = int(input(“Give me a number: ”))
except ValueError as ex:
    print(ex.args)
```

ValueError gets a tuple with only one value (a string)

Other errors gets more values

Other examples

```
d = {}

for i in range(5):
    x = input()
    try:
        d[x] += 1
    except:
        d[x] = 1

print (d)
```

Empty dictionary….

Other examples

Empty dictionary….

```
d = {}

for i in range(5):
    x = input()
    try:
        d[x] += 1
    except (Exception, TypeError, KeyError) as e:
        print (type(e), “:”, e)
        d[x] = 1

print (d)
```

Common exceptions

- ZeroDivisionError
  - y/0
- IndexError
  - List or tuple accessed beyond bounds
- FileNotFoundError
  - Accessing a file that does not exists
- ValueError
  - Error during a type cast operation
- ….

File handling exceptions

- IOError
  - the **first element** of args contains a number that is quite informative to understand exactly what went wrong
  - Use the errno module to have an easier interpretation of these numbers
    - errno.ENOENT
      - No such file or dir
    - errno.EACCESS
      - Permission denied
    - errno.ENOSPC
      - No more space left on the device
    - ….

Raising exceptions

- You are allowed to raise exceptions yourself!

Raising exceptions

- You are allowed to raise exceptions yourself!
  - When you write a module, and an error occurs, it is not nice to just print a message and exit
  - Just let the caller to handle it!

Raising exceptions

- You are allowed to raise exceptions yourself!

```
def getIntegerMax100(string):
    s = int(input(string))
    if (s > 100):
        raise ValueError("Too big!", s)
    return s
```

You can pass here a tuple, that can be accessed as seen before (args)

Raising exceptions

- raise can also be used with try….except

```
try:
    num = 3/int(input(“Give me a number: ”))
except ZeroDivisionError:
    raise
except ValueError:
    print(“Not an integer!”)
else:
    print(num)
```

If you are in the main, there are no other ‘levels’ in the program, so it simply crashes

(Proposed) Exercises

- Exercise 17.1 in the reference book

Programming For Data Science

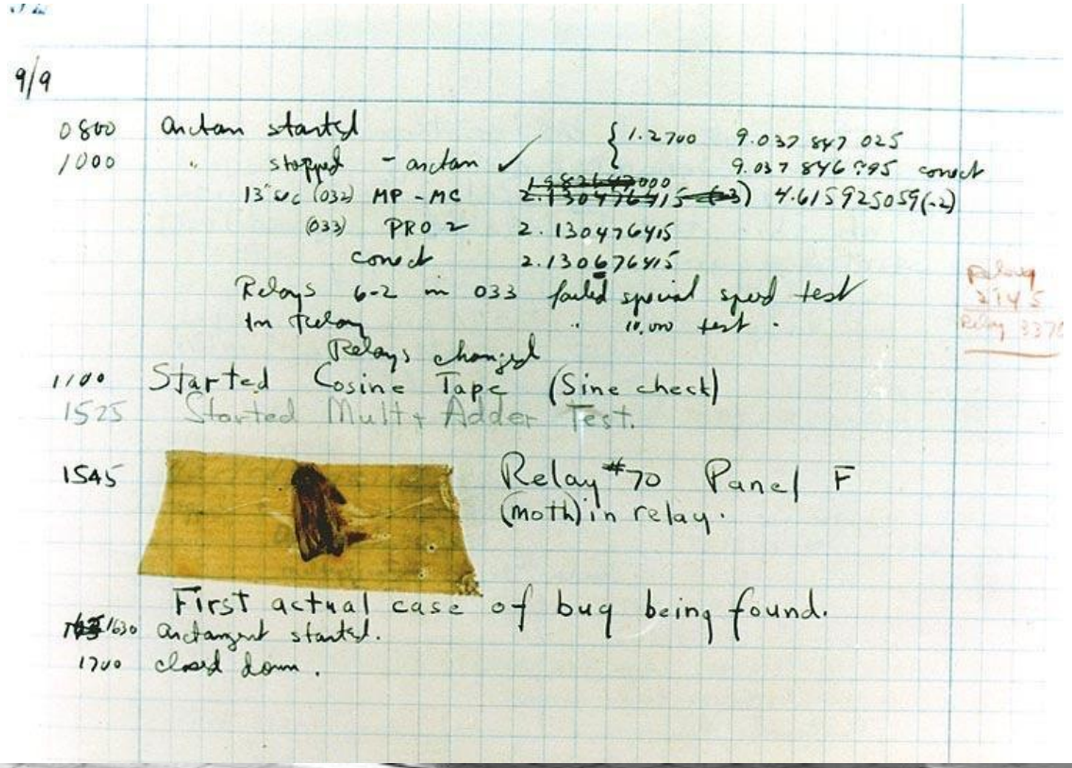
Part Sixteen:

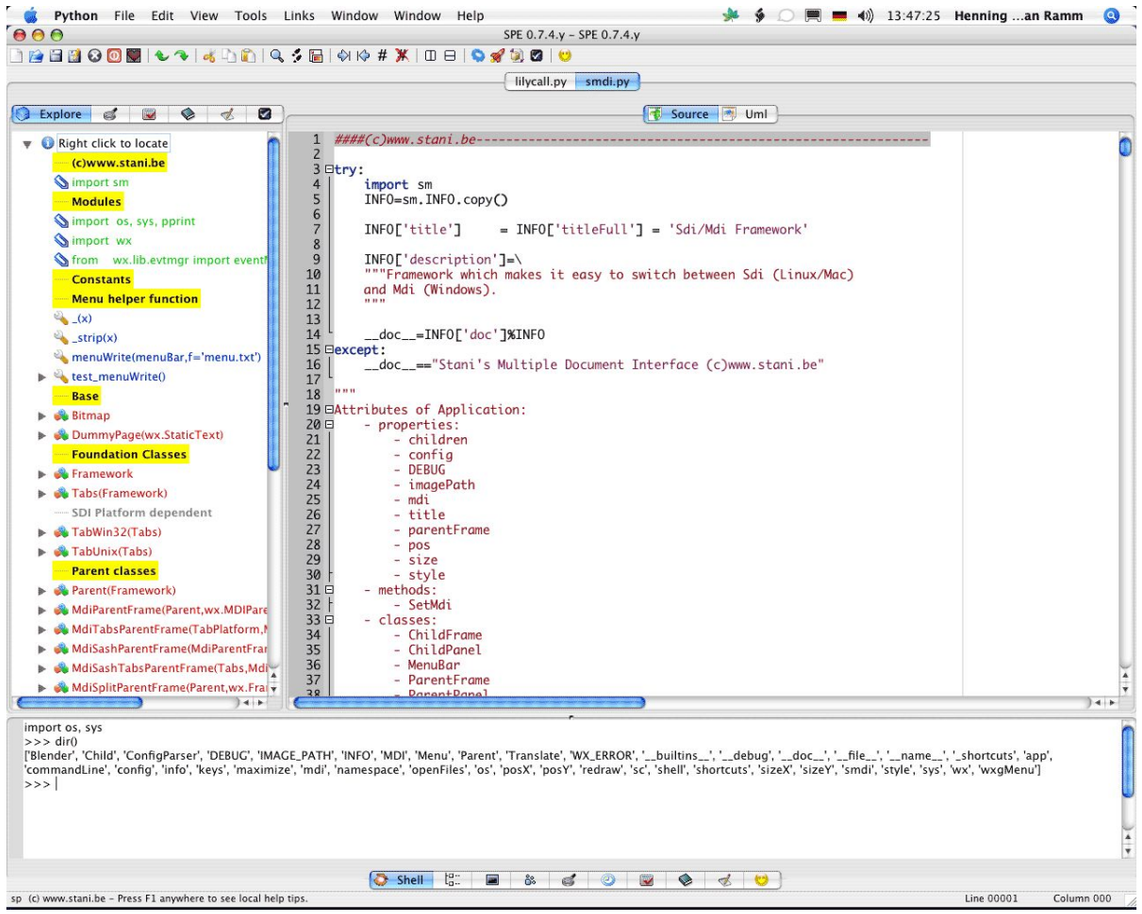
Debugging

Giulio Rossetti
giulio.rossetti@isti.cnr.it

Credits: Giuseppe Prencipe
giuseppe.prencipe@unipi.it

Debugging


9/9
0800 Arctan started
1000 " stopped - arctan ✓
1.2700 9.037 847 025
9.037 846 795 correct
13 MP - MC (032) 4.615925059(-2)
(033) PRO 2 2.13047615
correct 2.13067615
Relays 6-2 in 033 failed special speed test
In relay " 10.00 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1700 closed down.


Python File Edit View Tools Links Window Window Help
13:47:25 Henning ...an Ramm
SPE 0.7.4.y - SPE 0.7.4.y
lilycall.py smdi.py
Explore
Source Uml
Right click to locate
(c)www.stani.be
import sm
Modules
import os, sys, pprint
import wx
from wx.lib.evtmgr import eventm
Constants
Menu helper function
_(x)
_strip(x)
menuWrite(menuBar,f='menu.txt')
test_menuWrite()
Base
Bitmap
DummyPage(wx.StaticText)
Foundation Classes
Framework
Tabs(Framework)
SDI Platform dependent
TabWin32(Tabs)
TabUnix(Tabs)
Parent classes
Parent(Framework)
MdiParentFrame(Parent,wx.MDIPare
MdiTabsParentFrame(TabPlatform,M
MdiSashParentFrame(MdiParentFra
MdiSashTabsParentFrame(Tabs,Mdi
MdiSplitParentFrame(Parent,wx.Fra
####(c)www.stani.be----------
try:
import sm
INFO=sm.INFO.copy()
INFO['title'] = INFO['titleFull'] = 'Sdi/Mdi Framework'
INFO['description']=\
"""Framework which makes it easy to switch between Sdi (Linux/Mac)
and Mdi (Windows).
"""
__doc__=INFO['doc']%INFO
except:
__doc__="Stani's Multiple Document Interface (c)www.stani.be"
"""
Attributes of Application:
- properties:
- children
- config
- DEBUG
- imagePath
- mdi
- title
- parentFrame
- pos
- size
- style
- methods:
- SetMdi
- classes:
- ChildFrame
- ChildPanel
- MenuBar
- ParentFrame
- ParentPanel
import os, sys
>>> dir()
['Blender', 'Child', 'ConfigParser', 'DEBUG', 'IMAGE_PATH', 'INFO', 'MDI', 'Menu', 'Parent', 'Translate', 'WX_ERROR', '__builtins__', '__debug', '__doc__', '__file__', '__name__', '_shortcuts', 'app',
'commandLine', 'config', 'info', 'keys', 'maximize', 'mdi', 'namespace', 'openFiles', 'os', 'posX', 'posY', 'redraw', 'sc', 'shell', 'shortcuts', 'sizeX', 'sizeY', 'smdi', 'style', 'sys', 'wx', 'wxgMenu']
>>>
Shell
sp (c) www.stani.be - Press F1 anywhere to see local help tips.
Line 00001
Column 000

Debugging

- Rubber duck debugging
  - In software engineering, rubber duck debugging or rubber ducking is a method of debugging code. The name is a reference to a story in the book The Pragmatic Programmer in which a programmer would carry around a rubber duck and debug their code by ***forcing themselves to explain it, line-by-line, to the duck***

Debugging

- Print based debugging

```
# Whether n is prime


trovato = 0

i = 2

while i < n:

    if n % i == 0:

        trovato = 1

        print (“n:”, n, “i:”, i)

    i += 1


if trovato == 1:

    print (n, “is not prime”)

else:

    print (n, “is prime”)
```

Debugging

- Debugger
  - https://pythonhosted.org/spyder/debugging.html