




MAY 10, 2023

# LABORATORY 1: PYTHON

## OPERATING SYSTEMS

FRANCISCO SECCHI – ALEX HERNANDEZ  
TECNOCAMPUS MARESME-MATARÓ



## Index

Introduction .....	2
Requirements.txt.....	2
Libraries.....	2
Code Explanation .....	3
Parsing .conf file .....	3
FPDF class and methods.....	4
Main methods .....	6
1 – 1.2.....	7
1.3.....	7
2.....	8
2.1.....	8
2.2.....	8
2.3.....	9
2.4.....	9
2.5.....	10
2.6.....	10
2.7.....	11
2.8.....	11
2.9.....	12
2.10 – 2.12.....	12
2.13.....	13

## Introduction

Repository: [https://github.com/OS-TecnoCampus/PYTHON-LAB1\\_FranciscoSecchi-AlexHernandez](https://github.com/OS-TecnoCampus/PYTHON-LAB1_FranciscoSecchi-AlexHernandez)

In this report we will be explaining the first Python laboratory.

We will explain how to extract information from a configuration file to create a PDF file with it afterwards. The PDF file will have static and dynamic information from the configuration file specified.

First, we need to explain the requirements.txt file.

### Requirements.txt

This file contains which packages will be needed to execute our python program.

```
1 defusedxml==0.7.1
2 fonttools==4.38.0
3 fpdf==1.7.2
4 fpdf2==2.6.1
5 pdfmerger==0.5.0
6 Pillow==9.4.0
7 pypdf2==3.0.1
8 typing-extensions==4.4.0
```

It's a text file that consists of the name of each package and its version required in your python environment to execute our program correctly.

To make use of this, all you've got to do is open a terminal and navigate to the directory where this requirements.txt file has been cloned. Next, activate your Python or Python3 environment by prompting *"source env/bin/activate"*. Once you've got the environment activated you can install the packages: *"pip(or pip3 if using python3) install -r requirements.txt"*, and all the libraries required will now be installed.

The creation and constant update of this text file follows a similar procedure, prompting *"pip freeze > requirements.txt"* instead, and, of course, without the environment activated:

```
devasc@fsecchi: ~/labs/devnet-src/python/lab1-python-secchi_hernandez
File Edit View Search Terminal Help
devasc@fsecchi:~$ cd labs
devasc@fsecchi:~/labs$ cd devnet-src
devasc@fsecchi:~/labs/devnet-src$ cd python
devasc@fsecchi:~/labs/devnet-src/python$ ls
devices.txt lab1-python-secchi_hernandez
devasc@fsecchi:~/labs/devnet-src/python$ cd lab1-python-secchi_hernandez
devasc@fsecchi:~/labs/devnet-src/python/lab1-python-secchi_hernandez$ pip3 freeze > requirements.txt
```

### Libraries

For this laboratory, the main libraries we've used are FPDF and PyPDF2.

FPDF is really useful to create PDF files, write text, images, lines and color.

PyPDF2 is used to read PDF files and, in our case, to merge PDF files like the front-page and index to the main file.

## Code Explanation

### Parsing .conf file

First we open the .conf file requested by the client and then we call the method *parse()*:

```
filename = input("Enter file name: ")
with open(filename, "r") as f:
    lines = f.readlines()
    dict = parse(lines)
```

From here, "dict" will be our dictionary with all the information of the configuration file requested. Let's see how we parse this file:

```
def parse(lines):
    dict = {}
    current_config = None
    current_edit = None
    temp_lines = []
    for line in lines:
        line = line.strip()
        # First 4 lines
        if line.startswith("#"):
            current_config = line.split("=")[0][1:]
            value = re.split("[=:]|:", line)[1:]
            dict[current_config] = value
        elif len(temp_lines) > 0:
            if line.startswith("next"):
                value = parse(temp_lines)
                key = temp_lines[0][7:].strip()
                dict[current_config][current_edit][key] = value[key]
                current_edit = None
                temp_lines = []
            else:
                temp_lines.append(line)
        elif line.startswith("config"):
            if current_edit is None:
                current_config = " ".join(line.split(" ")[1:])
                dict[current_config] = {}
            else:
                temp_lines.append(line)
        elif line.startswith("edit"):
            current_edit = line.split(" ")[1].strip("\n")
            dict[current_config][current_edit] = {}
        elif line.startswith("set"):
            key = line.split(" ")[1].strip("\n")
            value = line.split(" ")[2:]
            if current_edit is None:
                dict[current_config][key] = value
            else:
                dict[current_config][current_edit][key] = value
        elif line.startswith("next"):
            current_edit = None
    return dict
```

The argument lines will be a list of all the lines in the configuration file. Here we define a dictionary and we track the "config" and "edit" sections in the file.

“Temp\_lines” will be used to call this function recursively whenever an “edit” section has a “config” section defined inside, for example:

```
config ips sensor
  edit "default"
    set comment "Prevent critical attacks."
    config entries
      edit 1
        set severity medium high critical
      next
    end
  next
```

Excluding this cases, all configuration files will follow a structure as Config → Edit → Set, where “config” can handle multiple “edit”, edit multiple “set” and each set will have multiple values.

All we need to do is loop through our list of lines to process the whole file and save each section with a Key-Value nested dictionary, since every config will be a key where its value will be another dictionary:

```
{Config 1: {
    Edit1: { Set1: [value1, value2], Set2: [value1] } ,
    Edit2: { Set1: [value1, value2], Set2: [value1] } } ,
 Config 2 ...}
```

We track whenever a line starts with “config”, “edit”, or “set” and save the information in the corresponding place.

### FPDF class and methods

To use FPDF and define new methods, we created a class called PDF that inherits from FPDF. Then we defined the methods we will need for the correct creation of our PDF file, such as the footer and header of the page. Some of the other methods are:

Add\_section():

```
def add_section(self, txt, priority):
    self.set_text_color(255, 180, 0)
    self.set_font('Helvetica', 'B', 14)
    self.cell(0, 7,txt,0)
    self.ln()
    self.reset_format()
    self.sections.append((txt, priority, self.page_no(), self.get_y()))
```

This is used to print a new section with the passed priority, and save it in a defined class variable “sections” to keep track of them.

Create\_Index():

```
def create_index(self):
    self.add_page()
    self.set_font('Helvetica', 'B', 12)
    self.cell(0, 10, "Index", 0)
    self.ln()
    tab_width = [10, 30, 40]

    for section in self.sections:
        tabs = tab_width[section[1]-1]
        self.set_x(tabs)
        self.cell(50, 10, section[0])
        self.cell(0, 10, str(section[2]), align='R')
        self.ln()
```

This method prints an index, using the sections list mentioned before.

Add\_headed\_table():

```
def add_headed_table(self, widths, data):
    self.ln(10)
    self.set_fill_color(255, 200, 60)
    self.set_draw_color(255, 200, 0)
    self.set_font('Arial', 'B')
    height = []
    for i in range(len(data)):
        height.append(0)
    for x, row in enumerate(data):
        height[x] = self.font_size_pt
        for i, item in enumerate(row):
            length = self.multi_cell(widths[i], self.font_size_pt, str(item), 1, 'L', 1, split_only=True)
            if len(length) > 1:
                height[x] = self.font_size_pt * len(length)

    for i, item in enumerate(data[0]):
        self.multi_cell(widths[i], height[0], str(item), 1, 'L', 1, max_line_height=self.font_size_pt, new_x=XPPos.RIGHT, new_y=YPos.TOP)
    self.ln(height[0])
    self.set_draw_color(255, 200, 0)
    self.set_font('Arial', '')
    for x, row in enumerate(data[1:]):
        for i, item in enumerate(row):
            self.multi_cell(widths[i], height[x+1], str(item), 1, 'L', 0, max_line_height=self.font_size_pt, new_x=XPPos.RIGHT, new_y=YPos.TOP)
        self.ln(height[x+1])
    self.reset_format()
    self.ln(15)
```

This method will be very much used to systematically create headed tables, it takes as arguments an array of widths of each column and an array of the data.

## Main methods

We've created some methods to work easier on the sections to be printed in the PDF:

Get() is used to access the dictionary data and return the information as a string, if there's any KeyError raised, we return an empty string:

```
def get(config, edit=None, set=None, value=0):
    try:
        if(edit is None):
            if(set is None):
                return dict[config][value]
            return dict[config][set][value]
        return dict[config][edit][set][value]
    except KeyError: return ""
```

Using Get() we can define the next method, Get\_data() used to get an array of the data requested in the arguments used to create tables, giving an array with the first row acting as the header of the table:

```
def get_data(final, config, keys, values=None, edits=None, include=False, separator=""):
    try:
        temp = dict[config] if edits is None else edits
        except KeyError: return final
        for edit in temp:
            row = []
            if(include):
                row.append(edit)
            for key in keys:
                try:
                    if(values is None):
                        v = ""
                        for value in range(len(dict[config][edit][key])):
                            v += get(config, edit, key, value).strip("\n")+separator
                        row.append(v)
                    except KeyError:
                        row.append("")
            final.append(row)
        return final
```

Front\_page() is used to create a front page for the PDF, it creates a new PDF file with only the front page to be merged later on to the main PDF. This way this page will not have a footer and header as it is the front page:

```
def front_page():
    pdf = PDF()
    pdf.add_page('P', 'A4')
    pdf.image('utils/logo.png', 110, 40)
    pdf.set_left_margin(30)
    pdf.set_font("Helvetica", size=28)
    pdf.ln(80)
    pdf.cell(0, 5, txt="Migració de la infraestructura de", align="L")
    pdf.ln()
    pdf.cell(0, 20, txt="seguretat perimetral per a", align="L")
    pdf.ln()
    pdf.cell(0, 20, txt="TecnoCampus", align="L")
    pdf.ln()
    pdf.ln()
    pdf.set_font("Helvetica", size=14)
    pdf.cell(0, 20, txt="Febrer, 2023", align="L")
    pdf.ln(65)
    pdf.set_font("Helvetica", "B", size=8)
    pdf.write(3, "La informació continguda en aquest document pot ser de caràcter privilegiat y/o confidencial. Qualsevol"+
        " disseminació, distribució o copia d'aquest document per qualsevol altre persona diferent als receptors"+
        " originals queda estrictament prohibida. Si ha rebut aquest document per error, sis plau notifiqueu"+
        " immediatament al emissor i esborri qualsevol copia d'aquest document.")
    pdf.set_line_width(5)
    pdf.set_draw_color(255,165,0)
    pdf.line(20, 10, 20, 280)
    pdf.output('portada')
    return 'portada'
```

## 1 – 1.2

```
frontpage = front_page()
content = PDF()
content.add_page()
content.reset_format()

# 1.
content.add_section('1. Introducció', 1)
# 1.1
content.add_section('1.1. Descripció', 2)
content.write(7,"El present document descriu la configuració realitzada en el dispositiu Fortigate-"
              +dict["config-version"][1][3:]+ " de Fortinet "+
              "a la empresa TecnoCampus resultat de la substitució de un Firewall perimetral Cisco de "+
              "l'organització.")
# 1.2
content.ln(15)
content.add_section('1.2. Objectius', 2)
content.write(7, "El objectiu d'aquest document és la de formalitzar el traspàs d'informació al equip tècnic"+
              " responsable del manteniment de les infraestructures instal·lades. Aquesta informació fa"+
              " referència al disseny, instal·lació i configuració dels dispositius i sistemes afectats per implementació.")
content.ln(15)
content.cell(0, 7, "La present documentació inclou:")
content.ln(15)
content.list_strings(["Descripció general de les infraestructures instal·lades.",
                    "Polítiques de filtratge de tràfic.",
                    "Perfils de seguretat.", "Connexions Túnel."], 10)
```

We start initializing a new PDF() as “content” and creating a front page with its method.

For each section we make use of the method *add\_section()* first and then the internal FPDF method *write()* to write our content in a justified format. We access to the config-version by calling our dictionary and specifying the “config” key, with the first value which is “FG080D”. Since all we want is the last 3 characters we use slice at the end: [3:].

In 1.2 we use *list\_strings()* to list a series of strings statically.

## 1.3

```
content.add_page()
content.add_section("1.3. Descripció general de les infraestructures", 2)
content.write(7, "Actualment la infraestructura te la següent distribució:")
content.image('utils/Infraestructura.png', 20, 70, 170, 140)
content.ln(170)
content.write(7, "En aquest esquema es pot veure com el firewall disposa actualmen
              +"\n\nLa infraestructura disposa de dos xarxes locals, la xarxa de
              +"\ntreball.")
```

All we need to do here is use the method *image()* to print an image defining it's path, position and scale.



## 2

```
# 2.
content.add_page()
content.add_section("2. Configuració del Dispositiu", 1)
content.write(7, "A continuació es detalla la configuració del dispositiu Fortigate-"
               +dict["config-version"][1][3:]+".")
```

We access the data the same way as section 1.1

### 2.1

```
# 2.1
content.ln(15)
content.add_section("2.1 Dispositiu", 2)
widths = [40, 100]
data = [{"Marca-Model", "FortiGate "+dict["config-version"][1][3:]},
        {"OS/Firmware", "v"+dict["config-version"][2]+"."+dict["config-version"][4]+" (" +dict["config-version"][5]+")"},
        {"S/N", ""}]
content.add_table(widths, data)
```

Here we define an array of static and dynamic data, getting it by calling the dictionary, and we print a table defining its columns widths and passing the data array.

### 2.2

```
# 2.2
content.add_section("2.2. Credencials d'accés", 2)
content.set_font('Helvetica', 'B')
content.cell(h= 7, txt="Accés: ", )
content.reset_format()
content.cell(0, 7, "https://10.132.4.254:"+ get("system global","admin-sport",0))
content.ln(7)
content.set_font('Helvetica', 'B')
content.cell(h= 7, txt="Usuari: ")
content.reset_format()
content.cell(0, 7, "admin")
content.ln(7)
content.set_font('Helvetica', 'B')
content.cell(h= 7, txt="Password: ")
content.reset_format()
content.cell(0, 7, "dfAS34")
content.ln(7)
content.set_font('Helvetica', 'B')
content.cell(h= 7, txt="Restriccions d'accés: ")
content.reset_format()
content.cell(0, 7, "xarxes "
               +get("system admin","admin","trusthost1")+ ", "
               +get("system admin","admin","trusthost2")+ ", "
               +get("system admin","admin","trusthost3"))
content.ln(15)
```

We add a new section, create new cells of information and calling our method `get()` whenever the data is dynamic.

## 2.3

```
# 2.3
content.add_section("2.3. General", 2)
content.write(7,"El dispositiu està configurat en mode NAT, és a dir, es separen varies xarxes a nivell tres d'enrutament.")
content.ln(15)
content.cell(h= 7, txt="DNS: ")
content.ln(7)
strings = ["Servidor Primari: "+get("system dns",set="primary"),
           "Servidor Secundari: "+get("system dns",set="secondary"),
           "Nom del domini Local: "+get("system dns",set="domain").strip("\'")]
content.list_strings(strings, 7)
content.ln(7)
```

Once again we need to list a sequence of strings, so we save them in an array and we call the method `list_strings`.

## 2.4

```
# 2.4
content.add_section("2.4. Interfícies", 2)
content.write(7, "El dispositiu instal·lat disposa d'una taula de polítiques de connexió per tal

widths = [35,30,60,0]
data = [["Interfície", "Alias", "Address/FQDN","DHCPRelay"]]

temp = dict["system interface"]
for i,edit in enumerate(temp):
    if(i>3):
        break
    row = []
    row.append(edit)
    row.append(get("system interface", edit, "alias").strip("\'"))
    row.append(get("system interface", edit, "ip"))
    row.append(get("system interface", edit, "dhcp-relay-ip").strip("\'"))
    data.append(row)
content.add_headed_table(widths, data)
```

In this section we create a new table, iterating through the list of “edit” in the config “system interface” and appending new rows of data, each column corresponding to the “alias”, “ip” and “dhcp-relay-ip” sets. We can’t use our pre-defined method since we only want the first 4 edits instead every one of them.

## 2.5

```
# 2.5.
content.add_page()
content.add_section("2.5. Taula d'enrutament", 2)
content.write(7,"S'ha definit "+ str(len(dict["router static"]))+" default gw per permesa
            get("router static",0,"gateway")+" (prioritat menor) i en cas de caiguda
            get("router static",1,"gateway"))

widths = [40,30,30,0]
data = [{"Xarxa Destí", "GW", "Interfície", "Prioritat"}]
keys = ["gateway","device","priority"]
data = get_data(data, "router static", keys)
for i in range(1, len(data)):
    data[i].insert(0, "0.0.0.0/0.0.0.0")
content.add_headed_table(widths, data)

content.write(7, "S'ha definit una sèrie de Health-checks de ping a través de les inter

widths = [35,30,30,20,20,0]
data = [{"Servidor Destí", "GW", "Interfície", "Interval", "failtime", "recovery"}]
keys = ["server","gateway-ip","srcintf","interval","failtime","recoverytime"]
data = get_data(data, "system link-monitor", keys)
content.add_headed_table(widths, data)
```

This time we use our `get_data()` method to create a new headed table. Inputting the correct arguments, the config and an array of the edit's we want to get data from, we get the array of data we will pass with them widths to print the table.

## 2.6

```
# 2.6.
content.add_section("2.6. Objectes Adreces del Firewall", 2)
content.write(7, "El dispositiu actualment te vinculats determinats objectes (noms descript

widths = [30,20,60,20,0]
data = [{"Name", "Category", "Address/FQDN", "Interface", "Type*"}]
edits = ["inside_srv","inside_wrk", "cloud1", "cloud2", "srv-demeter", "srv-devrepo","srv-r
keys = ["-", "subnet", "-", "type"]
data = get_data(data, "firewall address", keys, edits=edits, include=True, separator="/")
for i in data[1:]:
    i[1] = "Address"
    i[3] = "Any"
    i[4] = "Range" if i[4] == "iprange" else "Subnet"
content.add_headed_table(widths, data)
```

Here after getting our data for the table, we need to modify it so we have the correct information in each column, since we want all them rows to be as "Address" in the Category column, "Any" in the Interface and "Subnet" or "Range" in the Type column.

## 2.7

```
# 2.7.
content.add_section("2.7. Objectes Serveis", 2)
content.write(7, "El dispositiu configurat disposa de serveis predeterminats pe

widths = [45,50,30,30,0]
data = [["Nom del Servei", "Categoria", "Ports TCP", "Ports UDP", "Protocol"]]
keys = ["category","tcp-portrange","udp-portrange","protocol"]
data = get_data(data, "firewall service custom", keys, include=True)
content.add_headed_table(widths, data[:-2])
content.write(7, "Els serveis addicionals són:")
del data[1:-2]
for i in data:
    del i[4]
for i in data[1:]:
    i[1] = "Uncategorized"
content.add_headed_table(widths, data)
```

Section 2.7 has two separate tables, the second one corresponding to the last 2 rows of the first one. So, we use slicing to create the first table to pass all the data except the last 2 rows and then we delete all rows except those 2, so we create another table with them.

## 2.8

```
# 2.8.
content.add_section("2.8. NATs d'entrada /Virtual IPs)", 2)
content.write(7, "S'ha definit els següents NATs d'entrada (VIPs en nomenclatura Fortinet)")

widths = [30,45,40,30,0]
data = [["Name", "External IP Address/Range", "External Service Port", "Mapped IP Address/Range", "Map to Port"]]
keys = ["extintf","extip", "extport", "mappedip", "mappedport", "protocol"]
data = get_data(data, "firewall vip", keys, include=True)
for i in data[1:]:
    i[1] = i[1] + "/" + i[2]
    i[3] = i[3] + "/" + i[6]
    i[5] = i[5] + "/" + i[6]
    del i[2]
    del i[-1]
content.add_headed_table(widths, data)
```

Since some of the columns in this table have the data of a specific “set” in the config “firewall vip” slide another specific “set”, the add all the “set” required in the table in the data array and then we override the columns with the correct data and then we delete the columns we no longer need.

## 2.9

```
content.add_section("2.9. Polítiques de Firewall", 2)
content.write(7, "A continuació es mostren les polítiques de filtratge definides en el
widths = [7,15,15,20,20,10,12,13,13,14,15,15,10,10]
data = [{"ID", "From", "To", "Source", "Destination", "Service", "Action", "AV", "Web
keys = ["srcintf", "dstintf", "srcaddr", "dstaddr", "groups", "service", "action", "av-p
data = get_data(data, "firewall policy", keys, include= True)
for i in data[1:7]:
    i[1] = i[1] + "(" + i[3] + ")"
for i in data[1:]:
    i[2] = i[2] + " (" + i[4] + ")"
    i[3] = i[3] + " (" + i[5] + ")"
    del i[5]
data.append(["", "Any", "Any", "All", "All", "ALL", "Deny", "", "", "", "", "", "", ""])
content.set_font_size(7)
content.set_margins(left=10, top=20, right=10)
content.add_headed_table(widths, data)
```

We follow the same method as section 2.8, though this time we need to append a new last row to the table.

2.10 – 2.12

```
# 2.10.
content.add_section("2.10. Servei Antivirus", 2)
content.write(7, "El servei antivirus perimetral proveeix d'una base de dades automatitzada per a
    +"\nActualment el dispositiu te com el perfil d'antivirus activat "+
    list((dict["antivirus profile"]).keys())[-1]
    +"que detecta i neteja malware i possibles connexions a xarxes de Botnets.")

# 2.11
content.ln(15)
content.add_section("2.11. Servei de Filtrage Web", 2)
content.write(7, "El servei de filtratge de web, proveeix d'un servei de filtratge de contingut w
    +"\nActualment en el dispositiu s'ha definit el perfil "+
    list(dict["webfilter profile"])[-1]
    +"que actualment únicament genera logs de tot el tràfic de navegació web.")

# 2.12
content.ln(15)
content.add_section("2.12. Servei Application control", 2)
content.write(7, "El servei de Application Control realitza un filtratge a nivell d'aplicació per
    +"\nEn el dispositiu s'ha activat el perfil "+
    list(dict["application list"])[-1]
    +" i s'ha configurat per a generar logs de totes les aplicacions utilitzades i bl
```

Sections 2.10, 2.11 and 2.12 are just new sections to be added in the PDF, with static text filled with some dynamic text, which we access calling our dictionary and taking the last edit in the specified config for each section.

## 2.13

```
# 2.13
content.ln(15)
edit = dict["ips sensor"]
content.add_section("2.13. Servei Intrusion Protection", 2)
temp = dict["ips sensor"][list(dict["ips sensor"])[-1]]["entries"]["1"]
content.write(7, "El Servei de Intrusion Protection permet detectar possibles atacs de xarxa c
    +"\nEn el dispositiu s'ha activat el perfil UTM-IPS"
    +" en les polítiques de navegació web i s'han activat el comportament per defe
    +temp["location"][0] + ", de criticitat \""+ temp["severity"][1] + "\" i \""
    +temp["severity"][0]+ "\" que afectin a serveis de sistemes operatius"
    +temp["os"][0]+", "+temp["os"][1]+ " i "+temp["os"][2]+".")

merged = merge(front_page(), index(content.sections), 'merg')
content.output('content')
pdf_final = merge(merged, 'content', 'TCM_Report')
```

The last section access to a config inside an “edit” section of our dictionary data. To do so, we first define a “temp” variable as the “1” edit of the “entries” config, inside the last edit of the “ips sensor” config.

“temp” here will be a list of Key-Values, so all we have to do is call the corresponding “set” and its value to print the dynamic text.

Finally, we merge the front page with a new created index, we create our content PDF file using the method “output()” and we merge once again the PDF containing the front page and index with the content PDF.