Programming Logic Course
Handbook

Professor Libânia Paes

Table of Contents

Introduction

Welcome to the Programming Logic Course! You will learn tools that will be useful even if you are not programming!

Most people think programming is difficult because the code is hard to learn. Actually, this is the easy part… You are going to see in this handbook that you can do great and useful programs using only two or three commands.

Although I have programmed a little when I was a kid, I started my career a little bit late, around 25 years old. I got lots of books and tried to get by alone. One thing that no one told me when I was starting was that learning the commands is not enough. The problem is the logic part. Or: how to structure a program. It is more or less like building the sculpture on the left: the tools are simple – wood, chisels and knives. But there is more than that: there is planning, training, work and – of course – talent. At least to be a good programmer (not the best one!), you do not reeeally need talent. (I know that myself…) But you do need the other things…. ☺ (By the way, this is the face of a Java programmer….)

There is another thing that nobody told me: practically **nothing** is going to work the first time you run. You are going to receive tons and tons of error messages during your exercises. Do not panic! Do not give up! You are going to learn how to read the error messages and correct them.

And if you really do not have a clue about what is happening, paste the error message in Google… Believe me, lots of programmers have been there before you and there will be lots of answers…

These characteristics – logical thinking, patience, resilience, hard work – are being more and more demanded by companies. Actually, recruiters in Brazil complain that the problem is not the lack of jobs, but the lack of good people to fill the opportunities they have.

Ok, you are not going to get a job, you will open your startup: good entrepreneurs – even non-programmers – can talk to the IT guys on the same level. They discuss database strategies, business rules and user interaction. And do not leave all the responsibility to the programmers… Believe me… I've been at both sides of the table: dealing with a client that cannot understand a flowchart or having to work with a programmer that wants to do whatever he wants with my company code are situations I do not desire to anyone.

So, this handbook is organized in chapters and each chapter has a specific topic. Each of them is divided in sections:

a) Content: this first section explains the "theory" with basic examples. At the end of it, there is an item called "Also check this out" that lists some external links about the topics.

b) Project Explained: it is a much more complex code than the basic examples. The goal here is to show you how real world programs are made and how to think about your program before coding it.

c) Exercises: list of exercises for you to train (remember the hard work?)

I hope we have a great course this semester! See you in class!

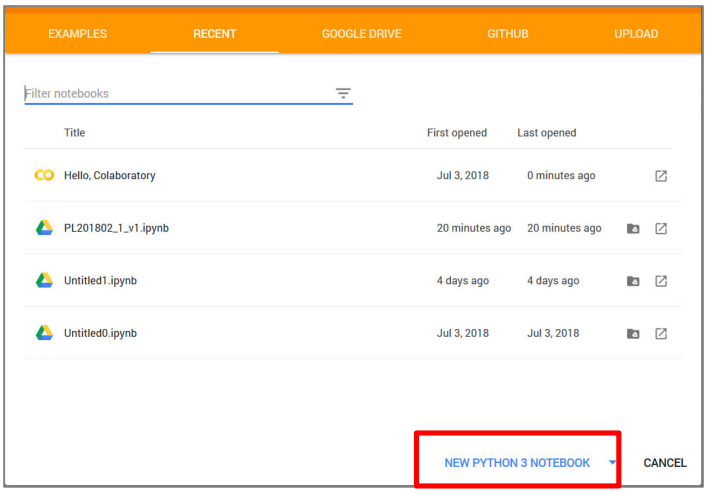
Topic 0 – Choose your tools

1.1. Content

There are lots of programming languages available and each one is most used for some tasks and/or environments. For instance, Javascript HTML5, PHP and .Net are tools for web development. Java and Swift are used for apps. Guys that develop security tools use Java and C. And, for analyzing data, programmers around the world are using **Python**. So that was our choice! Python is a very powerful tool and has a massive database of libraries for artificial intelligence and Machine Learning.

To program in Python, you have to get an IDE – Integrated Development Environment – and a Python interpreter. We can have both using Google's colab. Sign in to your Google account and type http://colab.research.google.com/. This will be our programming environment.

To begin, click **New Python 3 Notebook** button. Attention! We are using **Python 3**!



Any code you write can be run using the "play" button on the left side.

```
import getpass

# Variables
numAttempts = 0
username = input('Enter your username: ')
password = getpass.getpass('Eneter your password: ')

#Conditions and messages
if username != 'libania':
  print("Invalid username")
elif password != 'love':
  print("Invalid password")
else:
  print("You are online!")
```

If you want another tool, search for Jupyter Notebook on web!

No more talking… Let's work!

Topic 1 - My first code

1.1. Content

First commands: input and print

The first thing we learn in a programming language is to write something on the screen. In Pyhon, the command to do this is **print()**. Anything you write inside the parenthesis will be shown (printed) on the screen.
There is a very important thing here: if you want to print a number, just put it inside the parenthesis, like the example below:

```
print(123)
```

But when you want to write texts, you must use quotes or double quotes:

```
print("Python is fun... so far…")

print('I wonder if all programming languages are too…')
```

Both sentences will be printed, one below the other. You may use the quote you prefer. Just do not mix them. The example below will not work:

```
print("Python is fun... so far…')
```

If you print a number between quotes, Python will treat it as a text or, as programmers prefer to say, a string (because it is a string of characters…☺).

After printing things, programmers like to interact with the user of the program. It is very common: when you log in any website or app, you provide your username and password. To ask for user any information, we use the command **input()**.

Check the example below

```
input("What is your name? ")
print("You typed your name.")
```

When you run, the question will be printed and a field will be available so the user can fill some information. But this code is not very useful, right? To store the name and use later (to print, for instance), we need variables.

Variables

Variables are containers to store data. They work more or less the same way of a cell in Excel. In the

example below, *myNumber* stores the number 5. *MyOtherNumber* stores 10. When I multiply them, I get the result 50.

```
# You can name a variable (almost) anything you want
myNumber =  5
myOtherNumber = 10
print(myNumber  * myOtherNumber)
```

The main advantage of using a variable is to use its value without having to provide the information all the time.

You may also have noticed that the first line of the code above has a hashtag (#). This symbol is used to write comments through your code. Anything you write **after** the hashtag will not be considered by Python. It is a very good practice to keep you code commented so that you can remember what you did.

Besides storing the value, you can use and change it throughout the program.

```
# You can name a variable (almost) anything you want
num =  5
print(num)

num = num + 1   # changing the variable value print(num)
print(num)
```

The code will print 5 and then 6.
You can also use a variable to store input values:

```
name = input("What is your name?")
print("your name is " + name)
```

The code will print "your name is " and the name you typed. The plus (+) sign joins (concatenates) two or more pieces of text.

Attention! The value you get from the input command is a STRING (a text). If you need to make any calculation with it, you must CAST (transform) it into a number (integer or float). To do this, use int() or float().

```
num1 = input("Type an integer: ")
num2 = input("Type an decimal number: ")
product = int(num1) * float(num2)
print(product)
```

Attention II! To print ANY integer or float variable with a text, you must cast it as a string (a text)! Take a look at a variation of the former example:

```
num1 = input("Type an integer: ")
num2 = input("Type an decimal number: ")
product = int(num1) *  float(num2)
print("The product of the numbers is: " + str(product))
```

Conditionals: the basics

Besides printing and interacting with the user, programs are always evaluating conditions and making decisions. If you inform a wrong password, the app or site will not allow you in. So it compares what you wrote with the password they have stored. Are they equal? Great, welcome! Are the not? Sorry…

In this chapter we will only see the basics of conditionals just to make things more interesting. But we will get back to the subject in the topic 3.

The basic structure of a conditional in Python is:

```
if something is true:
   Execute A
else (if not):
   Execute B
```

Note that there are empty spaces before “execute A” an “execute B”. This means that “Execute A” is inside the if block and “Execute B” is inside the else block. Python is really annoying about these spaces, called indentation. You may have more than one line below the if command. If all of them are indented, they all will be executed if the condition is true.

Let’s see a real (and lousy!) example.

```
result = int(input("How much is 2+2? "))

if result == 4:    # yes! Two equal signs!
  print("Great job!")
else:
  print("Well... you tried...")
```

First, we get the result from the user. Then we test: if result equals 4, we print “great job”. If not (else), we print a consolation message.

For now, it is important that you know some of the comparison operators used by Python:

Operator	Meaning
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Also check this out

This extra links bring other examples of this chapter's content.
Introduction
Python basics
Syntax
Comments
Variables

1.2. Project Explained

The Project Explained section builds a much more complex example with the content you saw in this chapter. The main goal here is to help you to think like a programmer and learn how to structure your programs.

With only inputs and prints, you are already able to create three small programs that:

a) Asks a user to insert three grades (P1, P2 and PF) and it calculates the final grade (average). Suppose that all grades have the same weight.
b) Asks a user to insert only the two first grades (P1 and P2) and calculates how many points you need on your final exam to pass with grade 6. Suppose that all grades have the same weight.
c) Improve the first program to receive also the grades weights and calculate the average.

a) Asks a user to insert three grades (P1, P2 and PF) and it calculates the final grade (average).

Forget Python for a while. Let's list the steps the program has to accomplish. When I work with very complex apps/websites/programs, I get pencil and paper to think how it is going to work before coding.

```
Step 1: ask user P1
Step 2: ask user P2
Step 3: ask user P3
Step 4: Calculate the average
```

(So far, so good… Are you sure?)
This step 4 is a little bit generic… What is "calculate"? How to do it? Also, if I follow strictly the first 3 steps, I will get grades, but will not store them… So let's improve the steps and move on.

```
Step 1: ask user P1 and store in var P1
Step 2: ask user P2 and store in var P2
Step 3: ask user P3 and store in var P3
Step 4: create a variable myAvg
Step 5: myAvg = (P1 + P2 + P3)/3  * Remember to transform variables in float!
Step 6: show user the average
```

Especially when a project is new or complex to you, it is very important to make a list like this to organize the ideas. Translating it to Python is going to be very easy now.

```
# Step 1: ask user P1 and store in var P1
```

```
P1 = input("What was your first grade? ")

# Step 2: ask user P2 and store in var P2
P2 = input("What was your second grade? ")

# Step 3: ask user P3 and store in var P3
P3 = input("What was your third grade? ")

# Step 4: create a variable myAvg
# Step 5: myAvg = (P1 + P2 + P3)/3 * Remember to transform variables in float!
myAvg = (int(P1) + int(P2) + int(P3)) / 3

#Step 6: show user the average
print("Your average is: " + str(myAvg))
```

Without all the comments, the code would be like this:

```
P1 = input("What was your first grade? ")
P2 = input("What was your second grade? ")
P3 = input("What was your third grade? ")

myAvg = (int(P1) + int(P2) + int(P3)) / 3

print("Your average is: " + str(myAvg))
```

b) **Asks a user to insert only the two first grades (P1 and P2) and calculates how many points you need on your final exam to pass with grade 6. You can use weights to each grade.**

```
Step 1: ask user P1 and store in var P1 - transform in float
Step 2: ask user P2 and store in var P2 - transform in float
Step 3: create the variable myAvg with value 6
```

Now we have some math to do… If:

$$\frac{P1 + P2 + P3}{3} = 6$$

Then:

$$P1 + P2 + P3 = 18$$

And:

$$P3 = 18 - P1 - P2$$

With the formula structured, let's get back to our steps:

```
Step 1: ask user P1 and store in var P1 - transform in float
Step 2: ask user P2 and store in var P2 - transform in float
```

```
Step 3: create the variable myAvg with value 6
Step 4: create var P3 with formula: P3 = 18 - P1 - P2
Step 5: show the grade user has to get to pass with 6
```

And now, the code:

```
# Step 1: ask user P1 and store in var P1 - transform in float
P1 = float(input("What was your first grade? "))

# Step 2: ask user P2 and store in var P2 - transform in float
P2 = float(input("What was your second grade? "))

# Step 3: create the variable myAvg with value 6
myAvg = 6

# Step 4: create var P3 with formula: P3 = 18 - P1 - P2
P3 = 18 - P1 - P2

# Step 5: show the grade user has to get to pass with 6
print("You have to get " + str(P3) + " points to pass with grade 6")
```

c) Improve the first program to receive also the grades weights and calculate the average.

Now we are really going to use everything that we learnt so far. We know that professors use different weights to their grades. So we are going to ask the user for the weights.

But imagine that you have 5 courses with the weight structure 30-30-40 (30% for P1 and P2 and 40% for P3) and only one with 40-40-20… It would be horrible to have to inform the weights every time just to deal with ONE exception.

So we are going to ASK the user if he/she wants to use de default weights of not. If he/she does, W1, W2 and W3 will be 30, 30 and 40 respectively. Otherwise (else), we will provide the inputs so he/she can inform it.

Prepared? Let's begin with the basic inputs:

```
Step 1: ask user P1 and store in var P1
Step 2: ask user P2 and store in var P2
Step 3: ask user P3 and store in var P3
Step 4: ask user if the weights are 30-30-40 - store in defWeight
```

Now, we test it the answer is yes:

```
Step 1: ask user P1, P2 and P3 and store in P1, P2 and P3
Step 2: ask user if the weights are 30-30-40 - store in defWeight
Step 3: if defWeight is equal y
Step 4.1: set W1 = 30, W2 = 30 and W3 = 40
Step 5: if not, ask user W1, W2 and W3
Step 6: calculates myAvg = (W1*P1 + W2*P2 + W3*P3) / (W1 + W2 + W3)
Step 7: print myAvg
```

The important thing here is that you only have to think about the formula once. Any combination among the weights will work to calculate the average. Now, let's code it!

```
# Step 1: ask user P1, P2 and P3 and store in P1, P2 and P3
P1 = float(input("What was your first grade? "))
P2 = float(input("What was your second grade? "))
P3 = float(input("What was your third grade? "))

# Step 4: ask user if the weights are 30-30-40 - store in defWeight
defWeight = input("Are the weights 30%-30%-40%? y/n ")

# Step 5: if defWeight is equal y, set W1 = 30, W2 = 30 and W3 = 40
if defWeight == "y":
  W1 = 30
  W2 = 30
  W3 = 40
# Step 6: if not, ask user W1, W2 and W3
else:
  W1 = int(input("What was your first weight? "))
  W2 = int(input("What was your second weight? "))
  W3 = int(input("What was your third weight? "))

# Step 7: calculates myAvg = (W1*P1 + W2*P2 + W3*P3) / (W1 + W2 + W3)
myAvg = (W1*P1 + W2*P2 + W3*P3) / (W1 + W2 + W3)

# Step 8: print myAvg
print("Your average is: " + str(myAvg))
```

It seems complicated (and long!) but the beginning is always hard… Practice makes it more comfortable! Train your skills with the exercises (2020Topic1_IntroExerc.ipynb).

1.3. Exercises

```
# Exercise 1
# Print the multiplication between 2020 and 2019
```

```
# Exercise 2
# Ask the user for a number and print its triple
# Important! When you get a data using input, you get a STRING (a text)
# To transform it in a number, use int(variable)
```

```
# Exercise 3
# Get two int values from the user and return their product.
# If the product is greater than 1000, then return their sum.
```

```
# Exercise 4
# Create a program that calculates the area of a square, a rectagle or a
triangle.
# 1. Ask user which polygon he/she wants
```

```
# 2. If it is a square, ask for 1 side (one variable, one input)
#    If it is a rectagle or a triangle, ask for 2 sides (two variables, two
inputs)
# 3. If it is a square of a rectagle, multiply sides
#    If it is a triangle, multiply the sides and divide them by 2
# 4. Print the result
```

```
# Exercise 5
# Write a program that asks the user for three positive integers (a, b and
div).
# If a and b are divisible by (remainder = zero) div and ALSO by d+1, print
"LOTR". Else, print "Narnia".
# If the user inserts negative numbers OR empty values, print "Avatar".
# To calcularte the remainder,use %. Example: 10%5 = 0 (10 is divisible by 5);
11%5 = 1 (11 is not divisible by 5)
```

```
# Exercise 6
# Ask user if he/she prefers dogs or cats
# Get his/her age and calculate in dogs/cats years

# Dogs; For the first two years, a dog year is equal to 10.5 human years.
After that, each dog year equals 4 human years.
# Cats: 15 years for first year, 10 for second then 4 per extra year
```

```
# Exercise 7
# Calculate the age of a person
# Inputs: the year of birth
#         has he/she had a birthday this year?
```

```
# Exercise 8
# Create a Celsius - Fahrenheit conversion tool
# (32°F − 32) × 5/9 = 0°C
```

```
# Exercise 9
# Create a Inch - Centimeter conversion tool
# 2.54cm = 1in
```

```
# Exercise 10
# Suppose that you always go out for a pizza with a group of friends.
# Since one of them drives everybody to their homes, you decided to exempt
him/her from the 10% of the restaurant bill.
# Create a program that calculates the ammount each one has to pay.

# Inputs: total amount billed (without 10%)
#         number of people
```

Topic 2 - Accessing data from web with Pandas

2.1. Content

Basics

Remember Excel? Imagine that you could read (import) data from a spreadsheet (or any other database) and use it in Python. Lots of apps and websites do this for even simple tasks, such as logins. They have to compare your username and password with the ones in their database, right?

Since Python cannot read a spreadsheet "alone", you have to use an "external" group of tools called **Pandas**. Thus, you must write the following line each time you need it.

```
import pandas as pd

# We use the "as pd" to shorten the name pandas
# because we use it  a lot  in the code! :-)
```

The next step is to locate the Excel file you want to import. If you are using Jupyter on your own computer (you had to install it), you can access any file in your hard drive.

But if you use Colab, you have to know the online address where the file is stored (probably somewhere in the cloud). To find out how to access your files in GoogleDrive, check the link in the section “What you REALLY should know”. In the example below, I created a variable called url that will store the file address. If you copy the url and paste it in your browser, you will be able to download the file. (It is a good way to check you you did not misspelled the address).

```
import pandas as pd
url =  "http://profalibania.com.br/python/olympicMedals2016.xlsx"
```

So far, nothing happens. We have to import the file to Python. Excel stores data in a spreadsheet; Pandas does it in a **dataframe**. Excel has columns and rows. Pandas has the same structure: columns and rows.
Let’s import the spreadsheet into a dataframe called df, using the command read_excel. Notice that we have to use the "pd" prefix to show Python that the command is from Pandas.

```
import pandas as pd
url = "http://profalibania.com.br/python/olympicMedals2016.xlsx"
df =  pd.read_excel(url)
```

Now, to see the data, just write df or print(df). When you put just df, Pandas will show you a well formatted table. If you use print(df), it will show you the same data, in a raw format.

```
import pandas as pd
url = "http://profalibania.com.br/python/olympicMedals2016.xlsx"
```

```
df =  pd.read_excel(url)
df
```

Basic Analytics

There are some basic tasks you do when analyzing any kind of data. You really do not have to memorize ALL of them now. It is important that you learn that they exist… ☺ You are going to need them one day! Let's see some examples.

a) **Number of rows (length)**: the function len() serves to get how many rows (lines) there are in the dataframe

```
import pandas as pd
url = "http://profalibania.com.br/python/olympicMedals2016.xlsx"
df = pd.read_excel(url)
print(len(df))  # result: 86
```

b) **Show only a few columns**: this is useful when you are presenting a report of when your dataframe has lots and lots of columns.

```
import pandas as pd
url = "http://profalibania.com.br/python/olympicMedals2016.xlsx"
df = pd.read_excel(url)

df['Country']           # shows only Country column
df[['Country','Gold']]  # shows Country and Gold columns.
                        # Attention: yes, there are two brackets
```

c) **Sort the values**: in this example, how to discover the first country in alphabetical order? You use the command sort_values. The first attribute is the column you want to sort. The second is if the order is ascending (from A to Z, from 1 to 10 etc.) or descending (from Z to A, from 10 to 1 etc.). If it is the latter, use ascending=False.

```
import pandas as pd
url = "http://profalibania.com.br/python/olympicMedals2016.xlsx"
df = pd.read_excel(url)

df.sort_values(['Country'],  ascending=True)
```

d) Filter the values: you may want to deal with only a part of the data. In this example, let's check the countries that did not receive any gold medal. So, we will get only the rows (lines) where the gold column equals 0.

```
import pandas as pd
url = "http://profalibania.com.br/python/olympicMedals2016.xlsx"
df = pd.read excel(url)
df[df['Gold']==0] # inside brackets, put the expression:
                  # column Gold (see item b) equals zero
```

What if you want to get those who ONLY got bronze medals? It means that gold = 0 and ALSO silver =

0. In this case, you have to use the & sign to indicate AND (to indicate OR, use the | sign):

```
import pandas as pd
url = "http://profalibania.com.br/python/olympicMedals2016.xlsx"
df = pd.read_excel(url)
df[(df['Gold']==0) & (df['Silver']==0)]
#  Notice  the  parenthesis  to  separate  each  parameter!
```

e) **Select for specific records, by their index (number)**: sometimes you want to get, for example, the 10th item of the dataframe. Important! The records begin at number ZERO. So, the first item of the dataframe is item 0, the second is 1, the tenth is 11 etc.

The command **iloc** is very useful. It works like this: iloc[rows,columns].

In the example below, we are selecting only the first row (remember that they begin at zero!) and the first column (also zero).

```
import pandas as pd
url = "http://profalibania.com.br/python/olympicMedals2016.xlsx"

df = pd.read excel(url)
df = df.iloc[0,0]
df
```

Note: the output is: 'United States\xa0'. This '\xa0' code represents a space after the name of the country. This is very common when you bring data from the internet... :-(If you are just showing the data, you can just ignore it. The problem will be when you compare values... Wait for it at the chapter Data Manipulation with Pandas... :-)

The iloc command also allows us to select range of data, using colons (:). When you do not specify the range numbers, Pandas gets all of the items.

So iloc[0:2,:] brings: rows 0 to 2 (2 is not included!) and all columns.

```
import pandas as pd
url = "http://profalibania.com.br/python/olympicMedals2016.xlsx"

df = pd.read_excel(url)
df = df.iloc[0:1,0:3] # First row and three columns
df
```

```
    Country             Short       Gold
0  United States        USA           46
```

f) **Select rows by value**: suppose you want to check how many medals your country got in the 2016 Olympic Games. You can check the whole list or just select the row with the value using the command loc:

```
import pandas as pd
url = "http://profalibania.com.br/python/olympicMedals2016.xlsx"

df = pd.read_excel(url)
df = df.loc[df["Short"]=="BRA"]
df
```

In the example above, we used the Short code for the country because our database is "contamined" with the "\xa0" (see previous example). If we want to find Brazil, we should add the "\xa0" in the end of the name:

```
import pandas as pd
url = "http://profalibania.com.br/python/olympicMedals2016.xlsx"

df = pd.read_excel(url)
df = df.loc[df["Country"]=="Brazil\xa0"]
df
```

g) **Consolidate values**: remember pivot table in Excel? We can do the same things with Pandas. Some of the commands are: sum, min, max, mean, std.

```
import pandas as pd
url = "http://profalibania.com.br/python/olympicMedals2016.xlsx"
df = pd.read_excel(url)

# Total of golden medals
print(df['Gold'].sum())     #  result:  307
# Max of golden medals
print(df['Gold'].max())     #  result:  46
# Min of golden medals
print(df['Gold'].min())     #  result:  0
# Average (mean) of golden medals
print(df['Gold'].mean())    #  result:  3.5697674418604652
# Standard Deviation of golden medals
print(df['Gold'].std())     #  result:  6.896786459989327
```

h) **Get a list of unique values**: in the example below, we have a list of movies and their genres. Let's suppose that you want to get a list of all genres. You can use the function **unique()** to do this. Do not forget to choose the column from where you want to get the distinct values.

```
import pandas as pd
url = "http://profalibania.com.br/python/miniMoviesBasic.xlsx"

df = pd.read_excel(url)

print(df['Main Genre'].unique())
```

i) **Return the first n or the last n records**: if you want to see the first 3 lines or the last 5 ones, you may use the functions **head()** or **tail()**.

```
import pandas as pd
url = "http://profalibania.com.br/python/olympicMedals2016.xlsx"
df = pd.read_excel(url)

# First and second places
print(df.head(2))

# Three worst places
print(df.tail(3))
```

j) **Find empty cells**: sometimes you want to check if you have missing values (empty values). In the example below, we check if there are missing numbers in the Gold column

```
import pandas as pd
url = "http://profalibania.com.br/python/olympicMedals2016.xlsx"
df = pd.read_excel(url)

df = df[df["Gold"].isnull()]

print("There are " + str(len(df)) + " missing values")
```

Also check this out

Extra links fromGeeks for Geeks and TutorialsPoint:

Intro to Pandas and Dataframe
Lots of Statistics functions
Loc
iLoc
Access file in GoogleDrive

2.2. Project Explained

With this content, you can access and analyze data from any Excel file.
For this project, let's suppose that you do not know what to watch on TV in a Saturday night. You have a list of movies available at (http://profalibania.com.br/python/moviesBasic.xlsx). You have three columns: the title, the main genre and IMDB rating.

First offer the user a list of genres. He/she can choose one OR leave empty to have the program choose for him/her. Filter the chosen filtered items and choose randomly one of them. Show all the information available.

Let's make two versions of this program:
a) the first suggests the best and the worst movie of the chosen genre
b) the second suggests a random movie

a) the first suggests the best and the worst movie of the chosen genre

Again, let's begin structuring (in English!) the program:

```
Step 1: import pandas and the movie database
Step 2: show the list of possible genres
Step 3: ask for genre input and store in myGenre variable
Step 4: filter database comparing myGenre with the Genre colum
Step 5: check how many items there is: if none, send error message
Step 6: if there are items, sort them by the column IMDBrating
Step 7: show the first and the last item
```

Coding gets easier now: just follow the steps:

```
#Step 1: import movie database
import pandas as pd
url = "http://profalibania.com.br/python/moviesBasic.xlsx"
movies = pd.read_excel(url)

#Step 2: show the list of possible genres
print("Choose one of the genres above: ")
print(movies["Main Genre"].unique())

# Step 3: ask for genre input and store in myGenre variable
myGenre = input("\nWhat do you want to watch today? ")    # The \n jumps a line

# Step 4: filter database comparing myGenre with the Genre colum
movies = movies[movies["Main Genre"] == myGenre]

# Step 5: check how many items there is: if none, send error message
if len(movies) == 0:
  print("Sorry, no movies available...")

# Step 6: if there are items, sort them by the column IMDBrating
else:
  movies = movies.sort_values("IMDB Score", ascending=False)

  # Step 7: show the first and the last item
  print("The best movie: " + movies["Title"].head(1))
  print("The worst movie: " + movies["Title"].tail(1))
```

Check the code without the comments:

```
import pandas as pd
url = "http://profalibania.com.br/python/moviesBasic.xlsx"
movies = pd.read excel(url)

print("Choose one of the genres above: ")
print(movies["Main Genre"].unique())

myGenre = input("\nWhat do you want to watch today? ")

movies = movies[movies["Main Genre"] == myGenre]

if len(movies) == 0:
```

```
  print("Sorry, no movies available...")
else:
  movies = movies.sort values("IMDB Score", ascending=False)

  print("The best movie: " + movies["Title"].head(1))
  print("The worst movie: " + movies["Title"].tail(1))
```

b) the second suggests a random movie

The second program is pretty much like the previous, but, instead of sorting the movies by their score and getting the first and the last record, you will get one randomly. So the first 5 steps are the same. Check the others:

```
Step 1: import pandas and the movie database
Step 2: show the list of possible genres
Step 3: ask for genre input and store in myGenre variable
Step 4: filter database comparing myGenre with the Genre colum
Step 5: check how many items there is: if none, send error message
```

```
import pandas as pd
url = "http://profalibania.com.br/python/moviesBasic.xlsx"
movies = pd.read_excel(url)

print("Choose one of the genres above: ")
print(movies["Main Genre"].unique())

myGenre = input("\nWhat do you want to watch today? ")

movies = movies[movies["Main Genre"] == myGenre]

if len(movies) == 0:
  print("Sorry, no movies available...")
else:
   # SO FAR THE SAME CODE...
```

Hint: to generate a random number between 0 and x, you may you the code below. Remember that, when you filter the movies, you may put them in another dataframe. Then you can get its length and use the random between 0 and length-1.

```
import random
print(random.randint(0,9))
```

```
Step 6: if there are items, choose randomly a number from 0 to the length of the list
        Store this number in a variable called pos
Step 7: show user the record in the position number pos
```

```
else:
  # Step 6: if there are items,
  #         choose a number from 0 to the length of the list
```

```
  #         Store this number in a variable called pos

  import random   # import a library (just like pandas) to get random numbers
  pos = random.randint(0,len(movies)-1)  # (start,end)
  # It is -1 because if you have 10 items, their position goes from 0 to 9

  # Step 7: show user the record in the position number pos
  print("How about watching: " + movies.iloc[pos,0])
```

Check the complete code without the comments:

```
import pandas as pd
url = "http://profalibania.com.br/python/moviesBasic.xlsx"
movies = pd.read_excel(url)

print("Choose one of the genres above: ")
print(movies["Main Genre"].unique())

myGenre = input("\nWhat do you want to watch today? ")

movies = movies[movies["Main Genre"] == myGenre]

if len(movies) == 0:
  print("Sorry, no movies available...")
else:
  import random
  pos = random.randint(0,len(movies)-1)

  print("How about watching: " + movies.iloc[pos,0])
```

2.3. Exercises

```
#########################################################
# Top Youtubers -  Exercises 1 to 5
#########################################################

# The file 20TopYoutubers2019.xlsx has some informations about the biggest
digital influencers:
#   - Pos: position in the rank
#   - Username: he/she uses on Youtube
#   - Uploads: number of videos uploaded by this user
#   - Subscriptions: number of subscriptions to the channel
#   - Views: number of video views


# Exercise 1
# Connect to the database at
http://profalibania.com.br/python/TopYoutubers2018.xlsx
# Get the number of items in the rank


# Exercise 2
# Are the three top users the same as the three top in subscriptions number?
```

```
# If not, who are them?

# Tip: Python "remembers" the dataframe loaded on the previous cell, so you do
# not really have to import the data again....
# But personally I like to do it to have a "fresh start"...


# Exercise 3
# How many youtubers have more than 1 billion of subscribers?
# How many youtubers have less than 1 thousand of uploads?


# Exercise 4
# Considering all youtubers, their videos and views, how many times each video
was watched, on average?
# If these channels were subscribers-only, how many videos each subscriber has
watched?


# Exercise 5
# Build a program where a user gets the youtuber information by its name
(username column).
# If the record is not found (number of records = 0), print an error message.

# Steps:
# a) user inputs the name
# b) find the name
# c) if you find, print the line
# d) if noe (else), send a message
# Attention! Python is case sensitive! "abc" is different from "aBc" or "AbC"



# Exercise 6
# This is a variation of the previous exercise...
# Build a program where a user gets the youtuber information by its rank (pos
column) or name (username column).
# If the record is not found (number of records = 0), print an error message.

# Steps:
# The difference here from the previous code is that you have to decide if you
are looking for a position or an username
# a) user inputs if he/she will look for pos or username
# b) user inputs the name
# b) find the name OR find the position
# c) if you find, print the line
# d) if noe (else), send a message
# Attention! Python is case sensitive! "abc" is different from "aBc" or "AbC"


#########################################################
# Big Movies -  Exercises 7 to 12
#########################################################

# The file bigMovies.xlsx has lots of information about movies:
#  - Title: movie title
#  - Year: year of release
#  - Genres: all the categories to which the movie can relate
```

```
#  - Main genre: the main category
#  - Language: language spoken
#  - Country: studio's country
#  - Content Rating: if kids can watch or not
#  - Duration: length of the movie
#  - Aspect Ratio: screen size
#  - Budget: amount spent to make the movie
#  - Gross Earnings: movie revenue
#  - Director: director
#  - Actor 1: main actor or actress
#  - Actor 2: second main actor or actress
#  - Actor 3: third main actor or actress
#  - Facebook Likes - Director: how many likes the director has
#  - Facebook Likes - Actor 1: how many likes the main actor/actress has
#  - Facebook Likes - Actor 2: how many likes the 2nd main actor/actress has
#  - Facebook Likes - Actor 3: how many likes the 3rd main actor/actress has
#  - Facebook Likes - cast Total: how many likes the whole cast has
#  - Facebook likes - Movie: how many likes the movies has
#  - Facenumber in posters: how many faces appear in the movie poster
#  - User Votes: how many votes were given to build the score
#  - Reviews by Users: how many reviews users wrote
#  - Reviews by Critics: how many reviews critics wrote
#  - IMDB Score: the IMDB score/grade


## HINT! The BigMovies.xlsx file has more than 3.7k movies listed.
## If you want to begin with a smaller file, you can use
http://profalibania.com.br/python/miniMovies.xlsx

# Exercise 7
# Connect to the database at http://profalibania.com.br/python/bigmovies.xlsx
# Get the number of movies that have posters with no one on it (Facenumber in
posters = 0)


# Exercise 8
# Suppose you work for IMDB now. You and your colleagues have to fill some
movie's missing information.
# Your first task is to get the proper rating for each movies.
# To organize how your team will work, you suggest you divide the movies among
you.
# There are 4 people, incluing you.
# a) Get how many movies have no info about rating (it is empty)
# b) Divide by the number of people to see the amount of work
# c) Divide the movies in alphaetical order and list which movies you will
work with.
#    Example: if each one of you gets 12 movies, the first person will get
movie 0 to 11; the second movies 12 to 23 etc.
#    Consider you are the third one.


# Exercise 9
# What is the average score of comedies? Is it higher than dramas'? Use Main
Genre column to get the categories.


# Exercise 10
# In how many movies Tom Hanks appear? Attention: he may be actor 1, 2 or 3!
```

```
# Exercise 11
# Who has more likes on average? The main actor (actor 1) or the director?


# Exercise 12
# How many genres are in the list? Consider only the main genre column
```

Topic 3 - Conditionals

3.1. Content

Conditional basics

```
if [something is true]:
    print("This is true")
else:
    print("This is false")
```

The **if** command will test if a condition is true. If it is, it will run the lines below that are indented (they have empty space at the beginning of the line). If the condition is not true, Python will run the lines below the else.

Important: a) Notice the colon (:) at the end of the **if** line and of the **else** line. They are mandatory! b) The lines "inside" the **if** (that will be run if true) must be indented so that Python understands they "belong" to the **if** or to the **else**.

```
# This is correct!
# It will print "The number  is bigger than 10" and  "The number  is 10"

a = 20
if a > 10:
   print("The number is bigger than 10")
   print("The number is " + str(a))
else:
   print("The number is smaller or equal to 10")
   print("The number is " + str(a))
```

The next code will not work: the collon is missing (line 2) and the else has no commands inside it - because the last line (line 4) is ***not*** indented.

```
# This is NOT correct! a = 20
if a > 10
   print("The number is bigger than 10")
else:
print("The number is smaller or equal to 10")
```

The code below will run, but with a logical error.

```
# It will print "The number is bigger than 10" and "The number is 20"
***twice***.
# Python read that the last line is "outside" the else, i.e., it will run if a
> 10 or not.

a = 20
if a > 10:
   print("The number is bigger than 10")
```

```
   print("The number is " + str(a))
else:
   print("The number is smaller or equal to 10")
print("The number is " + str(a))
```

To equal or not to equal

When Python needs to check if two items are equal, it uses TWO equal signs: ==. To check for inequality, it uses !=. Check the example below:

```
#  Check  if value  is empty
# Check if value is not "your name" value = input("Type your name: ")

if value == "":
   print("Which  part  of  'Type  your  name'  did  you  not  understand?")
else:
   if value != "your name"
     print("Thanks!")
   else
     print("Nice try.")
```

Python also understand that capital letters are different from lowercase:

```
choice = input("Do you like Programming Logic? y/n")
if choice == "N":
   print("why not?!?!?!?")
else:
   print("Uêêêêêêbaaaaaaa!")
```

If the user types a small "n", it will print ’Uêêêêêêbaaaaaaa!’ because "n" is different from "N". One way to solve it is to check for both types:

```
choice = input("Do you  like Programming  Logic?  y/n")
if choice == "N" or choice == "n":
   print("why not?!?!?!?")
else:
   print("Uêêêêêêbaaaaaaa!")
```

What if you have lots of ifs?

Sometimes you need more than one if. Let's check a soccer game between Brazil and Argentine:
- IF Brazil makes more goals than Argentine -> Brazil wins
- IF Argentine makes more goals than Brazil -> Argentine wins
- ELSE (if none of the results above is true) -> draw

To make this second IF, we use the expression **elif**, that means ELSE + IF.

Check the example below:

```
BRA = 3
ARG = 1

if BRA > ARG:
  print("Brazil wins")
elif ARG > BRA:
  print("Argentine wins")
else:
  print("We got a draw!)
```

You can have any **elif**s you need! Hint: to check how many ifs and elifs you need, think about how man y possibilities you have. In the example above, we had 3 options (Brazil wins, Argentina wins or draw). So we had 2 if and elif. If we had 10 possibilities (ice cream flavours, for instance) we would need 9 if + elifs...

Also check this out

Extra links from W3Schools:
Conditionals

3.2. Project Explained

With this content, let's create a small program that checks if the username and the password are correct. In the file http://profalibania.com.br/python/fakeUsers.xlsx, there is a list of usernames and passwords. Take a look at them and memorize one or two examples.

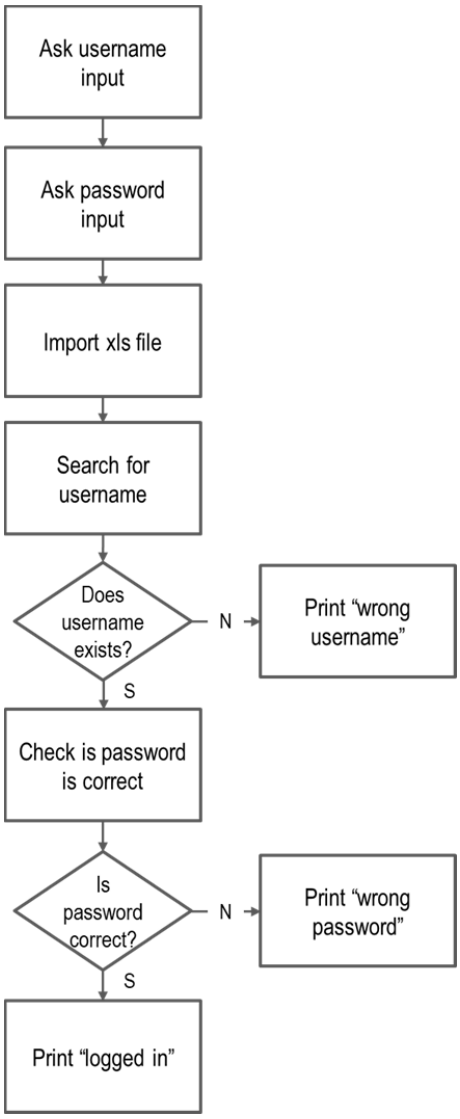
Create a program that asks a user for the name and the password. Check if the username exists. If it does, check if the password is correct.
If the username does not exist, print "wrong username".
If the password is wrong, print "wrong password".

We have worked so far with a list of steps to organize our programs. However, as programs get more complex, the steps get messy, mainly when we have to deal with several conditions and different scenarios.

To get the ideas organized, programmers often design a flowchart to show how the program is going to work. A flowchart is a diagram commonly used by IT professionals. Is our examples, we'll use only two symbols: a rectangle and a diamond.

The rectangle is used to express actions and the diamond, to represent decisions (conditionals). Similar to the if, the diamond only have two possibilities: yes or now (same as true or false).

In our project, it would be like this:



Notice that this is another way for you to write step by step the commands you have to code to make it work.

Now, let's code:

```
##############################
# Chapter 3 - Item a
##############################

# Ask username and password inputs
username = input("Type your username: ")
password = input("Type your password: ")

# Import xls file
import pandas as pd
url = "http://profalibania.com.br/python/fakeUsers.xlsx"
usersList = pd.read_excel(url)

# Search for username
usersList = usersList[usersList["Username"] == username]
```

```
# Does username exists?
if len(usersList) == 0:
  # print "wrong username"
  print("Wrong username")
else:
  # check is password is correct
  usersList = usersList[usersList["Password"] == password]
  if len(usersList) == 0:
    print("Wrong password")
  else:
    print("Logged in")
```

3.3. Exercises

```
# Exercise 1
# Let's roll the dices.
# Create a program that "rolls" two dices: one for player A and the other, for
player B.
# Show the result and who won

# Tip: to generate a random number from 1 to 6, use the code below:
import random
playerA = random.randrange(1,6)
```

```
# Exercise 2
# You have to organize a queue on an event. Here are the premises:
# People with numbers from 1 to 55 will watch "Indiana Jones" movies.
# People with numbers from 78 to 98 will watch "Twilight" movies.
# People with numbers from 102 to 150 will watch "Superman" movies.
# Any other number will watch "Peppa Pig" cartoons

# Generate a random number from 1 to 150 to test your logic
```

```
# Exercise 3
# Variation from the previous exercise!
# You have to organize a queue on am event. Here are the premises:
# People with numbers from 1 to 55 will watch "Indiana Jones" movies.
# People with numbers from 78 to 98 will watch "Twilight" movies.
# People with numbers from 102 to 150 will watch "Superman" movies.
# Any other number will watch "Peppa Pig" cartoons
# Any person with numbers that can be divided by 2 will receive free popcorn!

# Generate a random number from 1 to 150 to test your logic
```

```
# Exercise 4
# Another variation from the previous exercise! :-)
# You have to organize a queue on am event. Here are the premises:
```

```
# People with numbers from 1 to 55 will watch "Indiana Jones" movies.
# People with numbers from 78 to 98 will watch "Twilight" movies.
# People with numbers from 102 to 150 will watch "Superman" movies.
# Any person with numbers that can be divided by 2 will receive free popcorn!
# Any person with numbers that can be divided by 3 will receive free coke!
# Any person with numbers that can be divided by 2 and 5 will receive a movie
poster!
# Any other number will watch "Peppa Pig" cartoons

# Generate a random number from 1 to 150 to test your logic
# TIP: draft your exercise on a paper before coding!
```

```
#########################################################
# Big Movies -  Exercises 5 and 6
#########################################################

# Exercise 5
# Remember our movie list from last chapter?
# Build a program where the user inputs an name
# and get if the person is a director, an actor or both.

# The columns you will probably use are:
#    - Director: director
#    - Actor 1: main actor or actress
#    - Actor 2: second main actor or actress
#    - Actor 3: third main actor or actress

# Connect to the database at http://profalibania.com.br/python/bigmovies.xlsx
```

```
# Exercise 6
# The movies in this list are from year 2000 to 2016
# Create a program that will suggest a movie to the user based on the year,
main genre, duration and score.
# For each choice, you have to show user how many movies are available.
# Example: user types 2006 and the program tells him there are 239 movies
#          then he/she types comedy and the program returns 47 movies
#          ......
# In the end, the program will suggest a movie based on a random choice.
# If there are no movies available (check each option!), print "no movies for
you" and stop the inputs

# Attention: for duration and score, ask user the following choices:
#     a) duration: short - less than 90 min
#                  medium - from 91 to 120 min
#                  long - from 120 to 240 min
#                  marathon - more than 240 min
#     b) score: trash - less than 4.0
#               viewable - from 4.0 to 6.5
#               nice - from 6.6 to 8
#               OMG - more than 8
```

Topic 4 - Data manipulation

4.1. Content

Numbers

You will work mainly with two types of numbers in Python: integers and floats. Integer is a whole number and can be positive or negative. Float may have one or more decimals.

```
# These are numbers
myInt = 23
myFloat = 23.65
```

Operations

Probably you will need to perform operations to your numbers. In Python they are pretty much the same as Excel. Check the examples below.

```
a = 10
b = 3
# Addition
add = a + b     # result: 13
# Subtraction
sub = a - b     # result: 7
# Multiplication
mul = a * b     # result: 30
# Division
div = a / b     # result: 3.3333333333333335
# Exponentiation
exp = a ** b    # result: 1000
# Remainder (Modulo in Computing Science)
rem  =  a % b   # result: 1
```

Boolean

Besides numbers, programming deals a lot with boolean data. It can be True or False. Look the example below:

```
# True or false?
print(100<90)
```

The result is False. Attention! The boolean value of False is different from the string "False"! Boolean logic is very important to decision making in programming and is often used in conditional statements. Suppose you work in a hospital and have to organize patients in the Emergency. These are the rules: - Children unaccompanied: go to social service - Children accompanied and adults: wait for triage - Any inconscient person goes directly to physician - Ambulance patients go to trauma room
How do you create this program? First, before programming, try to organize the rules. We have kids and adults. Let’s check the adults:

1. IF inconscient -> physician
2. IF ambulance -> trauma room
3. ELSE -> wait for triage Now, the kids:
1. IF inconscient -> physician
2. IF ambulance -> trauma room
3. ELSE -> IF unaccompanied -> social service ELSE -> wait for triage

Notice that what varies between the two groups is when the kid is unaccompanied.
Thus we can use the other blocks to everyone:
1. IF inconscient -> physician
2. IF ambulance -> trauma room
3. ELSE -> IF child AND unnacompanied -> social service
ELSE -> waits for triage

Strings

Strings are texts between quotes ("double" or 'single'). You can choose whatever you prefer. Attention: even numbers between quotes are considered strings!
Probably you'll need to manipulate strings when programming. For example, let's suppose that you have a program where the user must create an username with at least 5 characters. To check ir he/she is making it right, you have to check for the length of the string.

```
# Function len() username = "abobrinha"
size = len(username)  # it is 9
if size < 6:
   print("Please create an username with minimum of 5 characters")
else:
   print("Success!!")
```

On the topic about Conditionals, we saw an example with "n" or "N". In Python you can "force" a string to lowercase or uppercase.

```
choice = input("Do you like Programming Logic? y/n")

if  choice.lower == "n":    # Forcing choice to be lowercase
   print("why not?!?!?!?")
else:
   print("Uêêêêêêbaaaaaaa!")
```

Some other useful things to do with strings:

a) **Erase the space before or after a string**: we are going to begin to deal with databases in a few topics and you'll notice that the data never comes "clean". Example: the string "Star Wars" is different from " Star Wars ". Extra spaces - mainly in the end of the strings - are common in data from web. To get rid of them, use the method strip().

```
movie = " Star Wars "
cleanMovie = movie.strip()
```

```
if cleanMovie == "Star Wars":
   print("Love it!")
else:
   print("Which movie?")
```

b) **Replace a string with other string**: suppose you want to get rid of ALL spaces, including the ones between words. You can replace them with an empty string.

```
movie = " Star Wars "
spaceMovie = movie.replace(" ", "") # first what you look for, second the
replacement
if cleanMovie == "StarWars":
   print("A non space movie!")
else:
   print("What?!?!?!?")
```

c) **Join (concatenate) strings (+)**: we have done this before. Remember that, when you mix string and numbers, you must cast the number to str.

```
movie1 = "Star"
movie2 = "Wars"
grade = 10

fullMovie = movie1 + " " + movie2 + " got grade " + str(grade)
```

d) **Extract a part of a string (substring)**: let's suppose that you want to get the second character of a string. Just inform the position between square brackets [].

```
myStr = "Star Trek"
print(myStr[1])  # result: t
```

Notice that the second position is position 1. Python - as many other computer languages - begins to count positions in a string, a list of a dataframe from number zero. In the text below, check the position of each character:

```
string   -> Star Trek
position -> 012345678
```

If you need more than one character - the first three, for example - use inside the brackets the initial position and the final position - 1, separated by a colon:

```
myStr = "Star Trek"
print(myStr[0:3])  # result: Sta     -> from 0 to 2 (3 minus 1)
```

Sometimes you need to get the characters from the right side of the string. You may use negative numbers to do it:

```
myStr = "Star Trek"
```

```
print(myStr[-4])  # result: T   -> the fourth character from the right
```

If you need more than one character, use the colon:

```
myStr = "Star Trek"
print(myStr[-4:-2])  # result: Tr   -> position -4 to -2 minus 1 (-3)
```

And to get the last four characters (from -4 to the end), leave the final position empty:

```
myStr = "Star Trek"
print(myStr[-4:])  # result: Trek   -> position -4 to the end of the string
```

e) **Check if a string is inside another string and return the position**: a very useful function for programmers is find(). It locates a string inside another string and returns the position of it. Let's suppose you have an email address and need to extract the username (text before the @). If all usernames had the same length, you could use the brackets we saw in the previous examples.

Check the example below with two different emails:

```
emailA = "ilovepython@google.com"  # @ is at 11th position
emailB = "python@pyton.com.br"     # @ is at 7th position
```

To discover where @ is, we use the function find:

```
emailA = "ilovepython@google.com"  # @ is at position 11 (count from zero!)
emailB = "python@pyton.com.br"     # @ is at 6 (count from zero!)

posA = emailA.find("@")
posB = emailB.find("@")

print("email A: " + str(posA))    # result: 11
print("email B: " + str(posB))    # result: 6

print("username A: " + emailA[0:posA])
print("username B: " + emailB[0:posB])
```

Also check this out

Extra links from W3Schools:
Numbers
Boolean
Strings
Operators

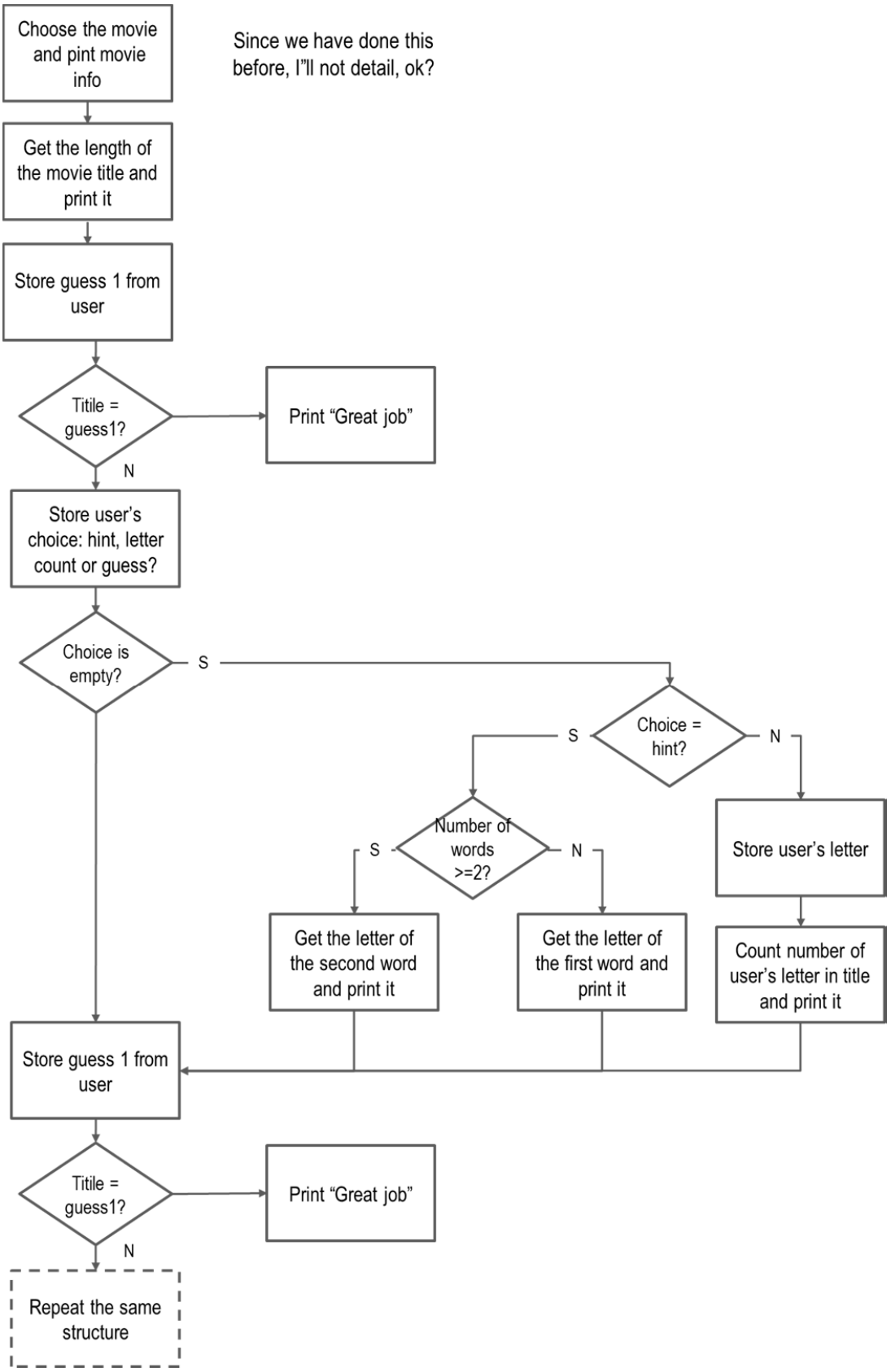
4.2. Project Explained

In this project, we are going to make a nice trivia game using all the commands we have seen so far.

The game will choose randomly a movie from the database and inform its year, director and main actor. It will also inform the number of words of the title. The user has 3 chances to guess the title. Each time he/she does not guess, the game will offer two possibilities: another hint or tell user the number of a specific letter in the title. This letter will be inputted by the user.

The two hints available for the second and third guess are: the first letter of the second word (if the title has only one, get from the first one); and the length of the first word.

So let's draw the flowchart (if you prefer, write as a list). I prefer the flowchart because we have to make lots of decisions. Although we are going to give 3 chances to our user, the diagram below will only show the two first. That is because the third will be exactly the same as the second.
Actually, Python and other programming languages have great ways to deal with repeatable tasks and we will check them in a few chapters. But, for now, let's work with what we have.



To the code! It is a long one this time! ☺ Follow the flowchart to have a better understanding of the code.

```
import pandas as pd
import random

# Choose the movie and print movie info
url = "http://profalibania.com.br/python/bigMovies.xlsx"
movies = pd.read_excel(url)

numberMovies = random.randint(0,len(movies))
movies = movies.iloc[numberMovies]

print("Movie year: " + str(movies["Year"]))
print("Director: " + str(movies["Director"]))
print("Main Actor: " + str(movies["Actor 1"]))

# Get the length of the movie title and print it
# Attention: "clean" the extra spaces in the end of the titles with strip()
title = movies["Title"]
title = title.strip()

spaces = title.count(" ")
print("The movie has " + str(spaces + 1) + " words")

# Store guess from user
guess = input("What is the name of the movie? (first guess) ")

# Title = guess? Yes!
if guess.lower() == title.lower():
  print("Fabulous job!")
else:
  choice = input("Ops... Sorry! \n\nType 1 for a hint, 2 to ask for other lett
er or empty to guess the movie directly")

  if choice != "":
    if choice == "1":  # all inputs are strings, remember? Get hint

      if spaces == 0:  # only one word
        # gets the letter in the first position
        firstLetter = title[0]
      else:
        # in which position is the first space?
        firstSpace = movie.find(" ")
        # So, the first letter of the second word will be at firstSpace + 1
        print("The first letter of the second word is " + movie[firstSpace+1])

    else: # Get chance to choose a letter
      userLetter = input("Give me a letter and I'll say how many are in the ti
tle: ")
      print("There are " + str(title.count(userLetter)) + " occurrences of the
 letter " + userLetter)


  # ATTENTION here to the indentation!
  # The guess has to run if the choice is filled or not,
  # so it has to be OUTSIDE the choice's if
  # Check in the flowchart how this actitvity is at the bottom of the diagram!
  guess = input("What is the name of the movie? (second guess) ")
```

```
  # Now we begin everything again...
  # ATTENTION TO THE INDENTATION!
  if guess.lower() == title.lower():
    print("Great job!")
  else:
    choice = input("Ops... Sorry! \n\nType 1 for a hint, 2 to ask for other le
tter or empty to guess the movie directly")

    if choice != "":
      if choice == "1":  # all inputs are strings, remember? Get hint

        if spaces == 0:  # only one word
          # gets the letter in the first position
          firstLetter = title[0]
        else:
          # the second hint is how many letters are in the first word
          # Remember that firstLetter brought us the position of the space
          # The lenght of the first word has the same value! :-)
          firstSpace = movie.find(" ")
          print("The first word has " + str(firstSpace) + " charaters")

      else: # Get chance to choose a letter
        userLetter = input("Give me a letter and I'll say how many are in the
title: ")
        print("There are " + str(title.count(userLetter)) + " occurrences of t
he letter " + userLetter)

    # Store guess from user
    guess = input("What is the name of the movie? (third guess) ")

    # Final check
    if guess.lower() == title.lower():
      print("Nice job!")
    else:
      print("Not this time.... ")
```

Of course there are some details that we haven't considered in the diagram and that can mess with our game. For instance, if the movie's title is "Jurassic Park" and the user asks how many "p" are in it, Python will find ZERO occurrences. Remember that "P" is different from "p".

Note that we can do lots of things using only basic commands! The main programmers' challenges are the logic to structure the programs and the attention to what can be wrong….

Try to create small programs like this one to practice the logic…. Because the commands are quite simple… ☺

4.3. Exercises

```
# Exercise 1
# Below are the first sentences of all Harry Potter books
```

```
# Which of them could not fit in a tweet (140 characters long)?

h1 = "Mr. and Mrs. Dursley of number four, Privet Drive, were proud to say
that they were perfectly normal, thank you very much."
h2 = "Not for the first time, an argument had broken out over breakfast at
number four, Privet Drive."
h3 = "Harry Potter was a highly unusual boy in many ways."
h4 = "The villagers of Little Hangleton still called it 'the Riddle House,'
even though it had been many years since the Riddle family had lived there."
h5 = "The hottest day of the summer so far was drawing to a close and a drowsy
silence lay over the large, square houses of Privet Drive."
h6 = "It was nearing midnight and the Prime Minister was sitting along in his
office, reading a long memo that was slipping through his brain without
leaving the slightest trace of meaning behind."
h7 = "The two men appeared out of nowhere, a few yards apart in the narrow,
moonlit lane."
```

```
# Exercise 2
# In a register service, users have to input their document number using the
# mask XXX.XXX.XXX-XX, where X are numbers.
# Build a program that receives an input and validates its format.
# Hint: replace the dots and the dash and see if the rest is a number!
```

```
# Exercise 3
# Gandalf was not able to open the Doors of Durin in the Lord of the Rings
book and movie.
# Let's design a similar trivia:
# User will read: type password and enter!
# If he/she does not type the word password, print "not this time"
# Else, print "Welcome to Nerd World"
```

```
# Exercise 4
# This is a different hangman game. Instead of getting the whole word, user
# has to guess how many specific letters are in the word.
# Choose one word and store in a variable.
# Choose randomly one of the letters
# Ask user how many of this letter are there in the word. Inform the total
length of the word
```

```
# Exercise 5
# This is a variation of the previous exercise
# Choose one word and store in a variable.
# Choose randomly one of the letters.
# Inform the total length of the word.
# Ask user how many of this letter are there in the word.
# If user does not get it right, give him/her the first and the last letter
# If user is wrong again, choose randomly another letter to show
```

```
# Exercise 6
# Suppose that a document number has an specific alghorithm:
# The format is XXX.XXX-XX, where X are numbers
# The last two digits are a verification code:
```

```
# You have to take the three first numbers and subtract from the second block
of three numbers.
```

```
# If the result is positive or zero, divide the result by 8 (round the number
to zero decimal places) and add 12
# If the result is negative, the result is divided by -6 (round the number to
zero decimal places).
# If this second result is bigger than 99, ou have to discard the digit on the
left.

# Example:
# 123.321-33 is a valid document?
# 123 - 321 = -198    --> negative number
# -198 / -6 = 33
# Yes!

# Example 2:
# 321.123-44
# 321 - 123 = 198
# 198 / 8 = 24.75  -> Round to 25
# 25 + 12 = 37
# No! It should be 321.123-37
```

```
# Exercise 7
# For security reasons, you cannot show the whole credicard number at the user
screen.
# Hide the numbers leaving only the three first and the last two digits.
# Note: all credit card numbers have 16 digits

# This looks like a replace exercise, but it is not because we cannot control
# the numbers in the middle of the credit card.

# We have to create a string with:
# a) 3 first numbers
# b) 11 # symbols  --> 11 because we will show 5 (3+2) and all cards have 16
numbers
# c) 2 last numbers
```

```
# Exercise 8
# Write a program that calculates the length of a sentence without the spaces!
# Tip: replace the spaces with nothing...
```

```
# Exercise 9
# Write a program that checks if user inputs only lower cases.
```

```
# Exercise 10
# Given two words, make the biggest one the same size of the other by deleting
# the "extra" characters in the beginning of the string

# Example: a = "Pokeball"
#          b = "Morty"
#  Result: eball and Morty  (both with 5 characters long)
```

Topic 5 - Data manipulation with Pandas

5.1. Content

We have already accessed data using Pandas and also have dealt with numbers and strings. Let's combine them.
In this topic, we'll import an Excel file that contains ID numbers, emails and usernames. Our task is to check three things:
a) If ID numbers are valid: they must be exactly 5 digits long.
b) If emails are valid: they must have a @.
c) If usernames are valid: minimum of 6 characters long, no spaces and lowercase. If there is any uppercase, you must convert it to lowercase.

Differently from our previous examples, you are not going to deal with the whole data, but analyze individually each record.

Length of a string

We have seen already how to calculate the length of a string. Now we are going to do the same thing with Pandas Data.
When we read a column in Pandas, using df["Name of the Column"], it returns the equivalent of a Excell cell. This cell contains different types of contents: strings, integer, floats, dates etc. So when you want to deal with the specific content, you have to cast (convert) the "cell".
In this example, we separate the Username column (df["Username"]), convert the data into a string (str) and calculate the length (len).

```
import pandas as pd
url = "http://profalibania.com.br/python/emailsDoctors.xlsx"
df = pd.read_excel(url)
df["Username"].str.len()
```

The results show a list of two numbers: the ones on the right are the index of each record (similar as the line numbers in Excel) and the others are the length of the strings.

```
0  8
1  7
2  6
3  8
....
28 6
29 8
Name: Username, dtype: int64
```

This is great, but does not show which users have each value. To do so, we can create a new column to store this new data.

Creating a new column

To create a new column in a dataframe is very simple: just name it and give it a value! Be careful not to indicate a column name that is already in use! It will replace the content of the current column!!!

```
import pandas as pd
url = "http://profalibania.com.br/python/emailsDoctors.xlsx"
df = pd.read_excel(url)
df["NewColumn"] = 0   # Creates a new column with all values = 0
```

Now, let's insert a new column with the length of the usernames:

```
import pandas as pd
url = "http://profalibania.com.br/python/emailsDoctors.xlsx"
df = pd.read_excel(url)
df["userLength"] = df["Username"].str.len()
df
```

Deleting a column

We create, we destroy. You can delete any column in a dataframe using the comand del.

```
import pandas as pd
url = "http://profalibania.com.br/python/emailsDoctors.xlsx"
df = pd.read_excel(url)
del df["Gender"]
df
```

String conversion

Now let's convert all the usernames to lowercase. Remember the lower() and the upper() from string manipulation? It is the same thing:

```
import pandas as pd
url = "http://profalibania.com.br/python/emailsDoctors.xlsx"
df = pd.read excel(url)
df["Username"] =  df["Username"].str.lower()  # could be upper() too
df
```

Another useful things beyond upper and lower with "programming LOgIC":
- str.title: converts the first character of each word to uppercase and the rest to lowercase (Programming Logic).
- str.capitalize: converts first character to uppercase and remaining to lowercase (Programming logic).
- str.swapcase: converts uppercase to lowercase and lowercase to uppercase (PROGRAMMING loGic).

String containing something

We have already seen how to filter content from a dataframe. But the filter only works when the whole content is found. What if we need to find only a piece of a string? We can use str.contains(). Let's separate the fgv.br emails from the others, creating a new column called "insider".

```
import pandas as pd
url = "http://profalibania.com.br/python/emailsDoctors.xlsx"
df = pd.read_excel(url)
df["Insider"] = df["Email"].str.contains("@fgv.br")
df
```

Add something to a number column

Let's suppose that our ID data is going to receive another number, on the left side. For example, all of them are going to begin now with the number 1. So, ID number 9981793 will be 19981793. It is quite simple: just sum 10000000 to the ID the same way you would on a variable or to print.

```
import pandas as pd
url = "http://profalibania.com.br/python/emailsDoctors.xlsx"
df = pd.read_excel(url)
df["ID"] = 10000000 + df["ID"]
df
```

You may also subtract, multiply and divide!

Insert letters and numbers to a string data

But let's suppose now that our ID data is going to receive a prefix. For example, all of them are going to begin now with the letter C. So, ID number 9981793 (notice that we are getting the data again from the original file!) will be C9981793. It is quite simple: just "add" the string the same way you would do to print something.
The catch here is that ID is a number (integer) column and Python does not deal with this automatically. So you must first convert to a string using astype(str)and then "adding" the string.

```
import pandas as pd
url = "http://profalibania.com.br/python/emailsDoctors.xlsx"

df = pd.read excel(url)
df["ID"] = "C" +  df["ID"].astype(str)
df
```

Replacing a piece of string with other

Imagine that one day gmail changes its name to geemail (I know, it's quite ridiculous.).
You have to correct all the email adressess in the dataframe. You can do this using str.replace(what_you_have,what_you_want).

```
import pandas as pd
url = "http://profalibania.com.br/python/emailsDoctors.xlsx"
df = pd.read_excel(url)
df["Email"] = df["Email"].str.replace("gmail","geemail")
df
```

Split a column in two or more

Sometimes you just need to break things apart. First and last name, hour, minute and second…
To accomplish it, there is a special method in Pandas called **split()**. (Actually, you'll see in the next chapters that you can also split strings!)

Check the example below. Suppose we have the same list of emails from our previous example. The column Name brings the full name of our doctors. Imagine that you have to send a personal email for each one of them, using – of course – only their first name on the top of the text. So, for Dr. Derek McDreamy Shepherd, we will compose the email with "Dr. Derek".

```
import pandas as pd
url = "http://profalibania.com.br/python/emailsDoctors.xlsx"
df = pd.read_excel(url)

# Split in the "group" called newColumns
newColumns = df["Name"].str.split(" ", expand=True)

# Pick the first item of the group and insert in a new column
df["First Name"] = newColumns[0]

# If I wanted to create another column with the third name:
df["Third Name"] = newColumns[2]
df
```

Split will "break" the "Derek McDreamy Shepherd" in parts using whatever I inform as the separator. In this case, a space is separating the names. So split() will break "Derek McDreamy Shepherd" in three colums: one with "Derek", the second with "McDreamy" and the third with "Sheperd".

To make sure Pandas will break in different columns, you must use the argument *expand=True*.

Also check this out

Extra links from Geeks for Geeks:
Convert Dataframe data with Pandas
Working with Text in a dataframe
About split()

5.2. Project Explained

In this file, you have a list of music stored in your pendrives and hard drives: "http://profalibania.com.br/python/musics.xlsx". It has seven columns: Artist, Album, Title (music), Year, Duration (in minutes), Genre and BPS (beats per second).

You are going to travel with some friends and want to organize three different playlists:
a) For when we're asleep: dance music, from 1965 to 2000 and pop, with bpm higher than 180
b) The best of the best: all music with rating higher than 4.5, except Bon Jovi: the car's owner hates them.
c) Easy listening: soundtracks from this century

If you take a look at the data, you will see that there are some problems:
a) BPMs are actually BPS – beats per second: you have to convert before filtering them
b) Dance music appears as "dance", "dance music" and "disco". Also, some are capitalized (only the first letter is uppercase), others are all upper…
c) If Bon Jovi is with other singer or group, it is ok to put in the list.

Another challenge: your friend's car has an old sound model that only accepts pen drives (no bluetooth available!) and you have to save the songs. You have three pendrives. Each one has 32Gb and will store one play list. To calculate the space available, suppose that one minute of music occupy 1.12Mb of space.

Let's work!

The first thing to do is to check if you have the data necessary to filter the songs. For example, to get those with BPM higher than 120, you must first adjust the BPS column. In the end, we will need to calculate the size of each file. Another problem: the duration is in a terrible format: 3min40sec. By the way, who creates this kind of data?!?

```
Step 1: Import data from excel
Step 2: Adjust BPS: multiply value by 60 and round - store at new column (BPM)
Step 3: Adjust Duration: "clean" values and store minutes at new column
```

```
# Step1: Import data from excel
import pandas as pd
url = "http://profalibania.com.br/python/musics.xlsx"
songs = pd.read_excel(url)

# Step2: Adjust BPS: multiply value by 60 and round -
 store at new column (BPM)
#        To do this, let's multiply all values by 60 and round them
songs["BPM"] = songs["BPS"] * 60
songs["BPM"] = songs["BPM"].round()

# Step3: Adjust Duration: "clean" values and store minutes at new column
#   This is going to give us some work
#   We have to separate min and sec using split (3.1)
#   The minutes will be ok, but the seconds, will still have the word "sec"
#   Clean the "sec" from the column sec (3.2)
```

```
#   Calculate the right duration in minutes using minutes * 60 + seconds (3.3)

# Step 3.1:  split the Duration columns in two
newColumns = songs["Duration"].str.split("min", expand=True)
songs["min"] = newColumns[0]
songs["sec"] = newColumns[1]

# Print the dataframe to check how your code is doing...
# songs

# 3.2.  Clean the "sec": replace the sec with empty
songs["sec"] = songs["sec"].str.replace("sec","")

# 3.3. Calculate the right duration
# Use the astype so Pandas knows for sure it is a integer!
songs["DurationMin"] = songs["min"].astype(int) * 60 + songs["sec"].astype(int
)
```

Now that we have clean data, let's create the playlists:

```
Step 4: Playlist A: Dance, from 1965 to 2000 + Pop, bpm > 180
Step 5: Playlist B: Grade > 4.5, except Bon Jovi solo
Step 6: Playlist C: soundtrack from 2000
```

```
################################
# step 4: create playlist A - the worst one!
#         We will put musics at a new dataframe: playlistA
#         There are two "inner" lists here: dance and pop
#         To simplify, let's create two and then join them
# 4.1. First get all disco/dance
playlistA_part1 = songs[(songs["Genre"].str.lower() == "dance") | (songs["Genr
e"].str.lower() == "disco") | (songs["Genre"].str.lower() == "dance music")]

# 4.2. The filter only the ones from 1965 to 1980
playlistA_part1 = playlistA_part1[(playlistA_part1["Year"] > 1964) & (playlist
A part1["Year"] < 2000)]

# 4.3. Make the second one: any pop wit BPM higher than 150
#      The problem here is that the pops are everywhere
#      So let's note which musics have pop in it
# First, make a copy of the whole database
playlistA_part2 = songs
playlistA_part2["Genre"] = songs["Genre"].str.lower()
playlistA_part2["isPop"] = songs["Genre"].str.contains("pop")

playlistA part2 = playlistA part2[playlistA part2["isPop"] == True]
playlistA_part2 = playlistA_part2[playlistA_part2["BPM"] > 180]

# 4.4. Join the dataframes!
playlistA = pd.concat([playlistA_part1, playlistA_part2], sort=True)

################################
# step 5: create playlist B - now this one is easy!
# 5.1. First get all Ratings > 4.5
```

```
playlistB = songs[songs["Rating"].astype(float) > 4.5]

# 5.2. Keep everything else than Bon Jovi
playlistB = playlistB[playlistB["Artist"] != "Bon Jovi"]

################################
# step 6: create playlist c - also easy!
# 6.1. First get all "soundtrack"
playlistC = songs[songs["Genre"].str.lower() == "soundtrack"]

# 6.2. Keep everything else than Bon Jovi
playlistC = playlistC[(playlistC["Year"] > 2000)]
```

Now let's calculate if we have space for all songs in each pendrive. We have to multiply the sum of minutes of each playlist by 1.2.

```
Step 7: Calculate space needed for each playlist
Step 7.1: sum the minutes in playlistA
Step 7.2: multiply by 1.2Mb
Step 7.3: Compare with 32.000Mb (~32Gb)

Step 8: repeat step 7 to playlistB
Step 9: repeat step 7 to playlistC
```

```
################################
# Step 7: Calculate space needed for each playlist
# Step 7.1: sum the minutes in playlistA
minA = playlistA["DurationMin"].sum()
# Step 7.2: multiply by 1.2Mb
minA = minA * 1.2
# Step 7.3: Compare with 32.000Mb (~32Gb)
if minA > 32000:
  print("Get yourself a bigger pendrive for playlist A: at least " + str(minA
- 32000) + "Mb biger!")
else:
  print("Playlist A ok!")

# Step 8: repeat step 7 to playlistB
minB = playlistB["DurationMin"].sum()
minB = minB * 1.2
if minB > 32000:
  print("Get yourself a bigger pendrive for playlist B: at least " + str(minB
- 32000) + "Mb biger!")
else:
  print("Playlist B ok!")

# Step 9: repeat step 7 to playlistC
minC = playlistC["DurationMin"].sum()
minC = minC * 1.2
if minC > 32000:
  print("Get yourself a bigger pendrive for playlist C: at least " + str(minC
- 32000) + "Mb biger!")
else:
  print("Playlist C ok!")
```

5.3. Exercises

```
#########################################################
# Top Youtubers -  Exercises 1 and 2
#########################################################

# The file 20TopYoutubers2019.xlsx has some informations about the biggest
digital influencers:
#   - Pos: position in the rank
#   - Username: he/she uses on Youtube
#   - Uploads: number of videos uploaded by this user
#   - Subscriptions: number of subscriptions to the channel
#   - Views: number of video views
```

```
# Exercise 1
# Connect to the database at
http://profalibania.com.br/python/TopYoutubers2018.xlsx
# Create a new column called ViewsPerVideo and calculate how many views per
uploads each channel has
```

```
# Exercise 2
# Insert in a new column the length of each username
```

```
#########################################################
# Doctors emails -  Exercises 3 to 6
#########################################################

# The file EmailDoctors.xlsx has some information about healthcare characters:
#    - ID
#    - Name
#    - Gender
#    - Email
#    - Username
```

```
# Exercise 3
# Convert the usernames to lowercase.
```

```
# Exercise 4
# Login validation: ask user for username and password.
# Check if the person exists: the password is the 5 last digits of the ID
# The program cannot be case sensitive: if username is PYTHon and user types
# python, it has to accept
# Hint: separate the 5 last digits in a new column to make the search easier
```

```
# Exercise 5
# The IT department is complaining about the length of the users' email alias
# (text before the @). They say it should be between 10 and 14 characters
long.
```

```
# Your job is to discover how many emails have problems and get in touch with
# the owners.
```

```
#######################################################
# Big Movies -  Exercises 6 to 13
#######################################################

# The file bigMovies.xlsx has lots of information about movies:
#    - Title: movie title
#    - Year: year of release
#    - Genres: all the categories to which the movie can relate
#    - Main genre: the main category
#    - Language: language spoken
#    - Country: studio's country
#    - Content Rating: if kids can watch or not
#    - Duration: length of the movie
#    - Aspect Ratio: screen size
#    - Budget: amount spent to make the movie
#    - Gross Earnings: movie revenue
#    - Director: director
#    - Actor 1: main actor or actress
#    - Actor 2: second main actor or actress
#    - Actor 3: third main actor or actress
#    - Facebook Likes - Director: how many like the director has
#    - Facebook Likes - Actor 1: how many like the main actor/actress has
#    - Facebook Likes - Actor 2: how many like the second main actor/actress
has
#    - Facebook Likes - Actor 3: how many like the third main actor/actress
has
#    - Facebook Likes - cast Total: how many like the whole cast has
#    - Facebook likes - Movie: how many like the movies has
#    - Facenumber in posters: how many faces appear in the movie poster
#    - User Votes: how many votes were given to build the score
#    - Reviews by Users: how many reviews users wrote
#    - Reviews by Crtiics: how many reviews critics wrote
#    - IMDB Score: the IMDB score/grade


## HINT! The BigMovies.xlsx file has more than 3.7k movies listed.
## If you want to begin with a smaller file, you can use
http://profalibania.com.br/python/miniMovies.xlsx
```

```
# Exercise 6
# Connect to the database at http://profalibania.com.br/python/bigmovies.xlsx
# Calculate all movies profits or losses: Gross Earnings - Budget
# Which Genre had more losses?
```

```
# Exercise 7
# Which movies have the smallest name?
# Which movies have the biggest name?
```

```
# Exercise 8
# The column Genre have multiple genres for each movie:
# How many movies have only one genre?
```

```
# Which movies have the biggest number of genres?
```

```
# Exercise 9
# In the Genres column, all the categories ares listed
# Create a program where user can type two genres.
# Suggest a movie to the user based on:
# a) Genre A (genre 1, must be the main genre)
# b) Genre B (genre 2, must be in the Genres column)
# If the user tries to be "funny" and types the same genre, send him/her a
error message
```

```
# Exercise 10
# Improvement of the previous exercise
# Use the same instructions from the previous exercise
# Now you have to check if the genre exist in the database or if user made a
mistake.
# If he/she mispelled the genre, offer a list of genres.
# Make your program case insensitive: it does not matter ir user types in
lower or uppercase
```

```
# Exercise 11
# Trivia game v. 1
# Create a trivia game where user has to guess the main actor (Actor 1).
# Give him/her the following information/hints:
# a) title of the movie (duh...)
# b) Length of the actor's name
# c) First letter of the actor's name
```

```
# Exercise 12
# Trivia game v. 2
# Let's make this game more interesting...
# If user gets it right the first time, receive 100 points
# If not, he/she gets another hint: actor's first name
# If user gets, receive 50 points; else, receives nothing
```

```
# Exercise 13
# Trivia game v. 3
# Sometimes a movie has more than one main actor...
# Use the same instructions of the previous exercise,
# but now the actor's name may come from columns Actor 1, Actor 2 or Actor 3.
```

Topic 6 - Loops

6.1. Content

There are so many repeatable tasks in programming... For example, let's suppose you have to find which numbers between 0 and 100 are divisible by 4. Imagine the scenario below:

```
if 0 % 4 == 0:
  print("0 is divisible by 4.")
if 1 % 4 == 0:
  print("1 is divisible by 4.")
if 2 % 4 == 0:
  print("2 is divisible by 4.")
if 3 % 4 == 0:
  print("3 is divisible by 4.")
if 4 % 4 == 0:
  print("4 is divisible by 4.")

# ..... until 100.....
```

I do not know you, but I am bored already... Python and other programming languages deal with repeatable tasks is to use **loops**. Imagine that each loop is one repetition. The main command to create these loops is **for**.

For command

The command **for** executes any task *n* times you indicate. Check the (lousy!) example below:

```
for x in range(1,5):
  print("Now the number is: " + str(x))
```

The result for this code is:

```
Now the number is: 1
Now the number is: 2
Now the number is: 3
Now the number is: 4
```

The for command created a variable called x and it "counted" from 1 to 4 (range: 1 to 5-1). Each time it "counted", it executed the print command.

Let's go back to the first example. To execute 100 times the same command, you can combine the for and the if command.

```
for x in range(0,100):
  if x % 4 == 0:    # using the x and divide by 4
```

```
    print(str(x) + " is divisible by 4.")
```

The result is:

```
0 is divisible by 4.
4 is divisible by 4.
8 is divisible by 4.
....
92 is divisible by 4.
96 is divisible by 4.
```

There are other ways to ask Python to "count" and repeat tasks using for. Check the examples below:

```
# You do not need to indicate the start number
for x in range(6):      # count from zero to 5 (10-1)
  print("I like number: " + str(x))

# You may skip numbers: two by two for example
for x in range(1, 6, 2):      # count from 1 to 5 (10-1), two by two
  print("Two by two: " + str(x))
```

The result is:

```
I like number: 0
I like number: 1
I like number: 2
I like number: 3
I like number: 4
I like number: 5
Two by two: 1
Two by two: 3
Two by two: 5
```

Loops and Pandas

We have seen already that you can loop through letters in a word. You can also loop through records (lines) in a dataframe.

This is especially useful when you need to do something with the data and Pandas cannot help you. Let's take a look at our email list.

Suppose you want to extract the alias (portion before the @) from the address.
We have used str.count() to do this in the Topic about Pandas. The problem is to extract the characters from position 0 to the position of the @. To do this, we have to "read" each line separately, using the for command.

```
import pandas as pd
url = "http://profalibania.com.br/python/emailsDoctors.xlsx"
```

```
df = pd.read_excel(url)
df = df["Email"]

for rowEmail in df:
  posAt = rowEmail.find("@")  # We got this using Pandas
  alias = rowEmail[0:posAt]   # Extract from position 0 to the @ position
  print(alias)
```

Nested loops

Imagine a repeatable task inside another one... Imagine that you have to count, in the email list, how many letters "a" are in the first three positions.
So there are two loops: the first is to run through the list of emails. The second is, for EACH email, to look at the characters 0, 1 and 2 (range 0 to 3-1) and check for "a" letters.

```
import pandas as pd
url = "http://profalibania.com.br/python/emailsDoctors.xlsx"
df = pd.read_excel(url)
df = df["Email"]

numOfAs = 0  # We are going to count how many numOfAs: so we begin with 0

for rowEmail in df:             # run through the emails
  for x in range(0,3):          # count numbers 0 to 2
    letter = rowEmail[x]
    # in each inner loop, get letter x (rowEmail[0] gets the first character

    # if the letter is equal to a, I sum 1 to numOfAs
    if letter.lower() == "a":
      numOfAs += 1

print(numOfAs)
```

While command

Python has other way to deal with repeatable jobs. The command **while** is used to execute tasks WHILE something is true. Take a look at the Lord of The Rings example below:

```
myPass = ""  # you have to

while myPass != "password":
  myPass = input("Type password and enter")

print("You are a LOTR fan...!")
```

Also check this out

Extra links from Geeks for Geeks:
GeeksForGeeks Loops
W3Schools Loops

6.2. Project Explained

Now that we know lots of tools, we can mix them to get the best of Python.
For instance, when we use the str in a dataframe to deal with strings, there are some things that Pandas cannot deal directly. Let's check the example below.

The Brazilian CPF is a document that identifies each person nationally. You will not find two CPFs alike. What most people do not know is that the third last digit indicates from which state is the person. Number 8 is for SP. Number 3 can be Ceará, Maranhão or Piauí.

There is no way to extract this digit from a string using only the dataframe. You have to ask Python to do it line by line. And, for repetitions, we use: loops! So we are mixing in this project Loops and Pandas.

Another thing: we are going do deal here with TWO dataframes: the first one stores a list of the states and numbers; the other has lots of random CPFs.

So, let's get organized:

```
Step 1: Import both files and store in dataframes
Step 2: Create a new column called State in the CPF dataframe
Step 3: Begin the loop for each CPF - For each CPF:
Step 4: Extract the last third digit
Step 5: Look for it in the states dataframe
Step 6: Insert in the State column the result -  end of loop
```

```
############################
# Chapter 6 - Loops
############################

# Step 1: Import both files and store in dataframes
import pandas as pd

codes = pd.read_excel("http://profalibania.com.br/python/CPFstates.xlsx")
cpfs  = pd.read_excel("http://profalibania.com.br/python/CPF.xlsx")

# Step 2: Create a new column called State in the CPF dataframe
cpfs["State"] = "hello"

cpfs["CPF"] = cpfs["CPF"].astype(str)  # Force the string to deal with the num
ber as a string

# Step 3: Begin the loop for each CPF - For each CPF:
```

```
for tempPos in range(len(cpfs)):
  # Step 4: Extract the last third digit
  tempCPF = cpfs["CPF"][tempPos]
  tempDigit = tempCPF[-3]

  # Step 5: Look for it in the states dataframe
  state = codes[codes["Code"] == int(tempDigit)]

  # Step 6: Insert in the State column the result - end of loop
  cpfs["State"][tempPos] = state["State"].to_string(index=False)  # The index=
False will remove the annoying index number

cpfs
```

6.3. Exercises

```
# Exercise 1
# If you pay really attention, the letter "i" looks like a candle.... :-0
# Ask user how old is he/she and print the number of candles for his/her next
birthday: each candle has to be in a different line
```

```
# Exercise 2
# Write a Python Program to create a table of power of two, from zero to 20.
# The output should be like:
# 2 ** 0 = 1
# 2 ** 1 = 2
# 2 ** 2 = 4
# 2 ** 3 = 8
# ...
# 2 ** 20 = xxxxxxx
```

```
# Exercise 3
# Write a Python program to sum numbers from 1 to 20 and print the final
result.
```

```
# Exercise 4
# Write a Python program to show the numbers from 1 to 20 in the same line
# and show their sum.

# Hint: to show the numbers in the same line, we have to join the numbers
# in the same string
```

```
# Exercise 5
# Write a Python program to show the mean of numbers between 20 and 50.

# Hint: the mean is the sum of the items divided by the number of items ☺
```

```
# Exercise 6
# In the 1970s, there was a children's TV game called "pin".
```

```
# The participant had to count from 1 to 50.
# For multiples of 3, he/she had to say "pin" instead of the number.
# Write a program that wins this game.
```

```
# Exercise 7
# Let's go crazy with the pin game....
# Now, the participant has to count from 1 to 50.
# For multiples of 3, he/she has to say "pin" instead of the number.
# For multiples of 5, he/she has to say "pon" instead of the number.
# For multiples of 3 and 5 at the same time, he/she has to say "pinpon".
# Write a program that wins this game.
```

```
# Exercise 8
# If you roll two dices, what are the possible combinations?
# How many are there?
# Result:
# 1x1
# 1x2
# 1x3
# ...
# 6x6
# There are x possible combinations

# Hint: Nested loops: the outer rolls the first die; the inner rolls the
second
```

```
# Exercise 9
# Print the following pattern
#     *
#    * *
#   * * *
#  * * * *
# * * * * *

# Hint: nested loops
# The first counts the rows; the second prints the spaces; the third prints
the stars
```

```
# Exercise 10
# There is a very well-known number series called Fibonacci
# (If you have watched "The Da Vinci's Code", you have heard of it....)
# In Fibonacci, the number is the sum of its two predecessors.
# So, the beginning of Fibonacci is: 0 1 1 2 3 5 8 13 21.....
# 0
# 1
# 1 = 0+1
# 2 = 1+1
# 3 = 2+1
# 5 = 3+2
# .....

# Question: what is the 100th item of the series?
```

```
# Hint: structure your loop on paper first!
```

```
# Exercise 11
# A palindrome is a word that is the same if you write backward and forward.
# Write a program that ask user for a word.
# Check if the word is a palindrome.
# If it is not, tell user what the word would be backwards

# Hint: use word[number] and word[-number] to compare the letters
```

```
# Exercise 12
# We have used the count() function to get how many times a string is inside
the other
# Suppose that you do not have the count function in Python and have to count
how many
# "a" are there in a giver word by the user.
```

```
# Exercise 13
# A csv file is a type of file often used in programming to transfer
information.
# Its structure is preety much like Excel (rows and columns), but,
# instead of visible rows, colums data is separated by a comma.
# (That's why csv means comma separated values)

# Imagine that the first line of the data contains the following string:
# morty,rick,patrick,spongebob,stewe,brian,homer,bart

# Write a program that separate the data in different lines
# (Actually there is already a command in Python to do this, called split,
# but let's pretend we are back in the 1960s...)
```

```
# Exercise 14
# Let's go to Vegas! But first let's know the roulette better:
# The numbers go from 0 to 36.
# If number ranges from 1 to 10 and 19 to 28, odd numbers are red and even are
 black.
# If ranges from 11 to 18 and 29 to 36, odd numbers are black and even are red
.
# There is a green pocket numbered 0 (zero).
# Print the numbers and their colors
```

Topic 7 - Lists

7.1. Content

We have already seen data files with Pandas. They bring us a group of data to work with. But sometimes we need a temporary list of values.

The same way we use variables to store strings and numbers, we can store lists of items. Take a look at the example below:

```
myList = [1,2,3,4,5]
```

The variable myList stores the 5 numbers, separately. We can also have a list of strings:

```
myMovies = ["Star Wars","LOTR","Jurassic Park"]
```

Working with lists

Much of what we can do to strings works with lists:

a) **Get the number of items** (length): we use len() to check the length of a list.

```
myMovies = ["Star Wars","LOTR","Jurassic Park"]
print("There are " + str(len(myMovies)) + " movies in my list")
# Result: There are 3 movies in my list
```

b) **Access a specific item in the list**: as characters in a string, items in a list begin in the position 0.

```
myMovies = ["Star Wars","LOTR","Jurassic Park"]

second = myMovies[1] # If the initial position is zero, this is the second
item
print("The second movie in my list is " + second)

last = myMovies[-1]
print("The last movie in my list is " + last)

# Result: The second movie in my list is LOTR
#         The last movie in my list is Jurassic Park
```

c) **Access a subset of items of a list**: if you want to access the two first movies, use the brackets.

```
myMovies = ["Star Wars","LOTR","Jurassic Park"]

best = myMovies[1:3]

print(best)
```

```
# Result: ["LOTR","Jurassic Park"]
```

d) **Insert and delete an item at the end of the list**: let's insert a new movie and remove Avatar from the list of good movies. ☺

```
myMovies = ["Star Wars","LOTR","Avatar","Jurassic Park"]

myMovies.append("Terminator")
myMovies.remove("Avatar")

myMovies

# result: ['Star Wars', 'LOTR', 'Jurassic Park', 'Terminator']
```

e) Insert an item in any position: to insert a new movie in an specific position (index), you can use insert (position,value):

```
myMovies = ["Star Wars","LOTR","Avatar","Jurassic Park"]

myMovies.insert(2,"Terminator")

myMovies

# result: ['Star Wars', 'LOTR', 'Terminator', 'Jurassic Park', 'Terminator']
```

f) **Find an item in the list and replace it with another value**: we can use the previous example again.

```
myMovies = ["Star Wars","LOTR","Avatar","Jurassic Park"]

pos = myMovies.index("Avatar") # finds the position of Avatar
myMovies[pos] = "Terminator"   # Replaces the item in this position with a
good movie :-)

myMovies

# result: ['Star Wars', 'LOTR', 'Terminator', 'Jurassic Park']
```

g) **Loop through a list**: suppose you have a list of integers and want to calculate the sum of them.

```
myInt = [132,234,268,444,908]
sum = 0

for eachNum in myInt:
  sum += eachNum

print(sum)
# result: 1986
```

h) **Check if a specific value is in a list**: if the movie is in the list, just send a message. If not, insert the

movie and show it.

```
myMovies = ["Star Wars","LOTR","Avatar","Jurassic Park"]
newFilm = "Jurassic Park"

if newFilm is in myMovies:
  print("This movie is already in the list")
else:
  myMovies.append(newFilm)
  myMovies

# result: This movie is already in the list
```

i) **Sort a list**: alphabetical or numerically, you can order the values of a list:

```
myMovies = ["Star Wars","LOTR","Avatar","Jurassic Park"]

myMovies.sort()

myMovies

# result: ['Avatar', 'Jurassic Park', 'LOTR', 'Star Wars']
```

A more complex example

In the example below, we have two lists. Boys contains names of male cartoon characters. The girls list has their partners/wives/girlfriends.

Notice that the couple occupies the same position. So the guy in boys[0] is married with girls[0].

```
boys = ["Homer","Peter","Fry","Jerry","Mickey","Donald"]
girls = ["Marge","Lois","Leela","Beth","Minie","Daisy"]
```

So let's make a small trivia game. Ask user to name a male cartoon character. If it is not in the list, we will send an error message. If it is, we will ask the partner's name and check the answer.

```
boys = ["Homer","Peter","Fry","Jerry","Mickey","Donald"]
girls = ["Marge","Lois","Leela","Beth","Minie","Daisy"]

maleName = input("Name a male cartoon character: ")

if maleName not in boys:
  print("Try another one....")
else:
  femaleName = input("Who is his partner/wife? ")

  # First, get the position of the guy
  pos = boys.index(maleName)

  if femaleName.lower() == girls[pos].lower():
    print("Good job!")
  else:
```

```
    print("Wrong... You nothing about love, Jon Snow!")
```

Lists and strings

There is a very useful method that breaks any string in pieces. With the **split()** command, you define which separator you are using and Python separates the pieces for you. Check the example below:

```
myString = "The words are separated by a space."
wordList = myString.split()

wordList
# Result: ['The', 'words', 'are', 'separated', 'by', 'a', 'space.']
```

Split is useful to break, for example, an email adress in alias + server name.
Imagine that you have a list os emails and you have to extract their alias and put in another list:

```
emails = ["mickey@mouse.com","homer@simpsons.com",
"bart@simpsons.com","stewie@griffin.com"]

alias = []     # empty list

for email in emails:      # look every email
  tempEmail = email.split("@")    # this is a temporary list to store the
pieces

  # When spliting mickey@mouse.com, having @ as the separator,
  # tempEmail will be ["mickey","mouse.com"]. So we will take only the first
  # element (mickey), or tempEmail[0]

  alias.append(tempEmail[0])

alias
```

Also check this out

Extra link from W3Schools:
Python Lists

7.2. Project explained

What happens in Vegas stays in Vegas! So let's create our own roulette game. If you know nothing about roulette games, take a look at: https://www.youtube.com/watch?v=ru2bYDG2FBw

Just to keep our code simple, a small customization: if our user wants to choose numbers (inside bet), he/she cannot bet in corners or splits: only in solo numbers (straight).

Our user begins with R$1.000 and he/she can play as many times as the money allows: if he/she loses

everything, the game is over. User can also quit writing “withdraw” in the bet space.

Very useful information:

a) About bets and gains: User can bet in the options below. Between parentheses, it is how much the value bet is returned to user if he/she gains.

- Numbers alone (35x)
- Dozens: first: 1-12; second: 13-24; third: 25-36 (2x – not official!)
- Odd/even (1x)
- Red/Black (1x)
- High: 1-8; low 19-36 (1x)

Example: odd/even pays one time the amount bet; but if you choose a specific number and get it, the bank pays 35 times what you bet!

b) The lists below contain the colors of each number. Since Python and the roulette begin with zero, there is no need to have a numbers list. The order of the colors is correct. Then the color of number five in colors[5] = “red”.

colors = [‘green’,
'red','black','red','black','red','black','red','black','red','black','black','red','black','red','black','red','black','red','red','black','red','black','red','black','red','black','red','black','black','red','black','red','black','red','black',
'red','black','red','black']

c) How user bets: we are going to give the options and he/she will input all options he/she wants separated by commas. Misspelling will not be considered! Example: 1,5,red,even,dozen1 – user is betting in numbers 1 and 5, red, even and the first dozen (1-12).
User also will input the bet value, also separated by commas. If the number of bets is different from the number of values, the bet is canceled.
Let’s organize the steps first. Since user can bet until the money is over, we have a repeatable action: a loop. We do not know how many bets will happen, so the best option here is **while**.

Roulette is quite a simple game: you take a number, compare to the ones the player chose. Then compare to odd/even, red/black, dozens or high/low. If player gets right, he/she gets the money; if not, looses.

The main challenge here is to deal with the odds (no kidding…. ☺). For instance: what if user writes a wrong bet? Or try to be more than what he/she has? And what if the length of bets is different from the “chips”?

So take a look at the steps:

```
Step 1: declare initial amount user has (myMoney) and variable myGains
Step 2: Create the loop: conditions are “still have money” (myMoney >0)
Step 3: Reset variables that will star over each time we bet
Step 4: Show how much money he/she gets and the betting options
Step 5: Get the bet
Step 6: Check if bet is “withdraw”. If it is, leave the game
Step 7: If not, ask for amounts
```

```
Step 8: If number of bets is different from number of values, cancel this round
Step 9: Check if user has the money: if not, continue (jump the rest of the loop)
Step 10: Roll the roulette! Get a random number between 0 and 36
Step 11: First check the numbers: does user has it? If yes, get the bet and multiply by
its factor
Step 12: Then check the color: user has it? If yes, get the bet and multiply by its
factor
Step 13: Then check odd/even: user has it? If yes, get the bet and multiply by its
factor
Step 14: Then check for high/low: user has it? If yes, get the bet and multiply by its
factor
Step 15: Finally check for dozens: user has it? If yes, get the bet and multiply by its
factor
```

```
############################
# Chapter 7 - Lists
############################
import random

colors = ['green',
'red','black','red','black','red','black','red','black','red','black','black',
'red','black','red','black','red','black','red','red','black','red','black','r
ed','black','red','black','red','black','black','red','black','red','black','r
ed','black','red','black','red','black']

# Step 1: declare initial amount user has (myMoney) and variable myGains
myMoney = 1000
myGains = 0      # will store the gains in each round

# Step 2: Create the loop: conditions are "still have money" (myMoney >0)
while myMoney > 0:
  # Step 3: Reset variables that will star over each time we bet
  gains = 0
  total = 0
  myValues = []
  myBet = []

  # Step 4: Show how much money he/she gets and the betting options
  print("Your balance is: " + str(myMoney))
  print("Your betting options are:\n- numbers\n- odd or even\n- red or
black\n- high or low\n- dozen1 (1-12), dozen2 (13-24) or dozen3 (25-36)")

  # Step 5: Get the bet
  inputBet = input("\nPlace you bets, separated by commas. Do not use spaces!
Write withdraw to leave: ")
  myBet = inputBet.split(",")

  # Step 6: Check if bet is "withdraw". If it is, leave the game
  if inputBet == "withdraw":
    break       # get out of the loop
  #Step 7: If not, ask for amounts
  else:
    inputValues = input("Place the ammount for each bet, separated by commas:
")
    myValues = inputValues.split(",")

    # Step 8: If number of bets is different from number of values, cancel
this round
    if len(myValues) != len(myBets):
```

```
      print("The number of bets must be equal to the number of values.")
      continue   # stops the loop and run again

    # Step 9: Check if user has the money: if not, continue (jump the rest of
the loop)
    # First, let's sum all the values in the second list
    for tempValue in myValues:
      total += int(tempValue)

    if total > myMoney:
      print("\n\nYou do not have enough money... Bet again.... \n")
      continue
    else:
      myMoney = myMoney - total

    # Step 10: Roll the roulette! Get a random number between 0 and 36
    ball = random.randrange(0,37)

    # Step 11: First check the numbers: does user has it? If yes, get the bet
and multiply by its factor
    if ball in myBet:
      index = myBet.index(ball)
      gains += int(myValues[index]) * 35

    # Step 12: Then check the color: user has it? If yes, get the bet and
multiply by its factor
    ballColor = colors[ball]
    if ballColor in myBet:
      index = myBet.index(ballColor)
      gains += int(myValues[index])

    # Step 13: Then check odd/even: user has it? If yes, get the bet and
multiply by its factor
    if ball % 2 == 0:
      ballEven = "even"
    else:
      ballEven = "odd"

    if ballEven in myBet:
      index = myBet.index(ballEven)
      gains += int(myValues[index])

    # Step 14: Then check for high/low: user has it? If yes, get the bet and
multiply by its factor
    if ball > 18:
      ballHigh = "high"
    else:
      ballHigh = "low"

    if ballHigh in myBet:
      index = myBet.index(ballHigh)
      gains += int(myValues[index])

    # Step 15: Finally check for dozens: user has it? If yes, get the bet and
multiply by its factor
    if ball <= 12:
      ballDozen = "dozen1"
    if ball >= 25:
      ballDozen = "dozen3"
```

```
    else:
      ballDozen = "dozen2"

    if ballDozen in myBet:
      index = myBet.index(ballDozen)
      gains += int(myValues[index]) * 2

    print("\n----------------------------------------")
    print("The number is: " + str(ball))
    print("Your balance before gains is: " + str(myMoney))
    print("Your gains were: " + str(gains))
    myMoney = myMoney + gains
    print("Your balance is: " + str(myMoney))
    print("----------------------------------------\n")
```

7.3. Exercises

```
# Exercise 1
# Generate a list with 10 random numbers between 1 and 100 and sum all of them
# Attention! This exercise can be done without using lists...
#            But use them to study, ok?
```

```
# Exercise 2
# Generate a list with 10 random numbers between 1 and 100 and get the largest
of them
```

```
# Exercise 3
# In the list below, there are scariest movies characters in the world.
# Check in how many of them the first letter appears again in the rest of the
name.
# Be careful with lower and upper!

monsters = ['Michael Myers','Jason
Voorhees','Leatherface','Chucky','Pennywise','Freddy Krueger','Jack
Torrance','Ghostface','Hannibal Lecter','Regan
MacNeil','Pinhead','Xenomorph','Samara Morgan','Bruce','The Thing','Norman
Bates','Damien Thorn','Gage Creed','Yautja','Annie Wilkes','Count
Dracula','Joker','Frankensteins monster',]
```

```
# Exercise 4
# Given the list below, remove items that are duplicated. How many duplicates
where deleted?

movieChar = ['Lara Croft ','Mark Lewis ','Abraham Lincoln ','Elf Legolas
Greenleaf ','T. E. Lawrence','Lola Lola ','Lucas "Luke" (Jackson)
','Lady','Lara (Antipova) ','Lassie','Leon ','Colonel Hans Landa ','Buzz
Lightyear','Laurel & Hardy','Loretta Lynn ','Ilsa Lund','Vicki Lester/Esther
Blodgett','Don Lockwood ','Wilma Dean Loomis','Lulu','Lady Lou ','Lolita
(Dolores Haze) ','Lisa','Lorelei Lee ','Tracy Lord','Myra Langtry
','Leatherface','Dr. Hannibal Lecter','Loki','Mabel Longhetti ','The Lone
Ranger ','Lee','Jake LaMotta','Don Logan','Harry Lime','Leeloo','King Leonidas
','Lex Luthor ','Princess Leia (Organa) ','Vesper Lynd ','Ace Lannigan ','Mark
```

```
Lewis ','Lulu','Loki','Lex Luthor ','Leeloo','Lassie','King Leonidas ','Ilsa
Lund','Harry Lime','Elf Legolas Greenleaf ','Colonel Hans Landa ',]
```

```
# Exercise 5
# For Harry Potter fans...
# Below you will find the list of subjects Harry, Hermione and Ron applied to.
# a) In how many of them the three friends will be together?
# b) Is Ron going to be alone in any class?
# c) Which of them will study anything related to Muggles? (Hint: look for the
word Muggle)

Harry = ['Alchemy','Apparition','Care of Magical Creatures','Frog Choir']
Hermione = ['Alchemy','Arithmancy','Care of Magical Creatures','Muggle
Studies','Ancient Studies','Magical Theory','Muggle Art']
Ron = ['Alchemy','Apparition','Arithmancy','Care of Magical Creatures']
```

```
# Exercise 6
# Have you ever played "rock, paper, scissors"? For those who do not remember,
rock breaks scissors, scissors cut paper and paper wraps rock.
# Write a game where you play "rock, paper, scissors" with the computer. The
computer choice is random: use a list to store the options
# This exercise helps you to train logic!
```

```
# Exercise 7
# Someone asked an intern to get some movie data from the internet. He created
a list with all the the data mixed
# You know there are two types of data: strings and numbers. The strings are
movie titles and the numbers, their release year.
# The list is organized like this: ['some title',2010,'another title',2019]
# There is only one problem: some movies have no years, so you may find two
titles in a row.
# Your job is to separate the values in two lists: one for the titles and the
other for the years.
# Print both lists lenghts in the end

internJob = ['The Hoax ',2006,'M*A*S*H','Chasing Liberty ',2004,'Strangers
with Candy','A Beginners Guide to Snuff ',2016,'Shooting the
Warwicks ',2015,'The Living Wake ',2007,'DysFunktional Family ',2003,'The
Honeymooners','Hot Tub Time Machine ',2010]
```

```
# Exercise 8
# Generate a list with 10 random numbers between 1 and 100 and get the first
and the second highest values
```

```
# Exercise 9
# You have to join the two lists below in a third one.
# The problem is that the new list can not contain duplicates.
# Print the third list and its length

genreA =
['Drama','Documentary','Family','Adventure','Comedy','Action','Biography','Thr
```

```
iller','Crime','Sci-
Fi','Musical','Animation','Mystery','Sport','War','Music','Western','News','Fa
ntasy','Short']
genreB =
['Sport','War','Music','Western','News','Fantasy','Short','Comedy','Sci-
Fi','Family','Romance','Horror','History']
```

```
# Exercise 10
# Using our movie database in
"http://profalibania.com.br/python/bigmovies.xlsx",
# create a program in which the user can ask for suggestions in various
genres.
# Note that the column Genres has multiple genres, different from Main genre.
# These are separated by |.

# The user has to input the genres separated by commas: comedy,drama,sci-fy

# List 5 movies of each genre separately, using Main genre as the source.
# Also list all the movies that have ALL the genres selected

# HINT 1: ask your user to write the genres in alphabetical order to make your
life easier!
# HINT 2: check for spaces around the genres:
#         in "comedy, action" the split will generate "comedy" and " action"
(extra initial space)
```

```
# Exercise 11
# Improve the previous exercise: we have a small problem in our previous
exercise:
# if the user writes the genres not in alphabetical order, the movies with
# all genres will not be found.

# How would you organize the following list in alphabetical order?
# Suppose you DO NOT have the sort function!!!!!
# Hint: letters ares more or less like numbers: a < b; ab > aa...

# Take this list to think to create your code

listGenre = ["Horror","Comedy","Bugudum","Drama","Action"]
```

Topic 8 - Functions

8.1. Content

Probably you have used Excel once in your life... Have you ever wondered how the average() function was created? Probably not (sooooo nerdy!), but let's think about it.

A **function** is a command that executes a bunch of commands.

The calculate the average, we have to sum all the items and divide by the number of items. In Excel, we just select the cells and use the function average that does all the work.

To create a function in Python, we use the word **def**. Check the example below:

```
def Lionel():
  print("Hello!")
```

This is a horribly simple function that prints "Hello!". There are some very important things here:
a) The word **def** defines the function:
b) After the name, there are parentheses: they will be used to pass information (parameters), exactly like do in Excel.
c) The command below def is indented, showing that it belongs to function block. Everything you write (right below the def, obviously!) with this indentation will be considered part of the function.

To call the function (or run it), just write its name:

```
def Lionel():
  print("Hello!")

Lionel()
```

Imagine that we want to customize the function, printing the name of the person. The function has to "receive" the name (parameter). This parameter can be used anywhere *inside* the function.

```
def Lionel(theName):
  print("Hello, " + theName + ", is it me you were looking for?")

Lionel("Ritchie")
```

To pass the parameter, put it inside the parenthesis when you call the function.

The greatest advantage of a function is that you can reuse it several times in your code. Suppose you have a program that runs some code three times and - for some reason - you cannot use loop.

The code would have to be written three times... Using functions, you write once and then just call it whenever you need.

Imagine that we are creating a function in Python that calculates the average of a group of values, just like Excel does. But, instead of selecting cells, the user has to inform the number separated by commas.

So the function will receive a string with numbers and commas. So, it needs one parameter:

```
def myAvg(strValues):



```

Next, we have to transform the string into a list of values, cast each one into int (because they are string!), sum the values and divide by the number of them (len).

```
def myAvg(strValues):
  listValues = strValues.split(",")

  for i in range (len(listValues)):
    listValues[i] = int(listValues[i])

  theAvg = sum(listValues)/len(listValues)

  print(theAvg)

myAvg('1,2,3,4,5')       # The result is 3.0
myAvg('10,20,30,40,50')  # The result is 30.0
myAvg('5,8,15,19,75')    # The result is 24.0
```

Returning values

For now, we wrote functions that printed their results. But sometimes we just need the result back, without the printing. Functions have the power to just **return** values. Let's take a look at a variation of our average function.

```
def myAvg(strValues):
  listValues = strValues.split(",")

  for i in range (len(listValues)):
    listValues[i] = int(listValues[i])

  theAvg = sum(listValues)/len(listValues)

  return theAvg


print("The average + 1234 is:")
print(myAvg("1,2,3,4,5") + 1234)   # result 1237
```

Instead of printing the result, I'm returning it to the code. Then I can reuse the value in my code, like adding 1234 to it.

Multiple parameters

You may also pass and return multiple parameters. The function below asks for two numbers and returns the sum and the average:

```
def sumAvg(num1,num2):
  theSum = num1 + num2
  theAvg = theSum/2

  return theSum, theAvg


mySum, myAvg = sumAvg(10,20)    # mySum will get the first value returned
                                # and myAvg, the second

print("The sum is: " + str(mySum))
print("The average is: " + str(myAvg))
```

Also check this out

Extra links from Geeks for Geeks:
Python Functions

8.2. Project Explained

Remember our trip to Vegas and the roulette game? That code can be optimized to be cleaner and easier to use. Check the extract below, from step 11. Each time I analyze a bet, I have to pass through all the ifs. The code gets long and messy.

```
    # Step 11: First check the numbers: does user has it? If yes, get the bet
and multiply by its factor
    if ball in myBet:
      index = myBet.index(ball)
      gains += int(myValues[index]) * 35

    # Step 12: Then check the color: user has it? If yes, get the bet and
multiply by its factor
    ballColor = colors[ball]
    if ballColor in myBet:
      index = myBet.index(ballColor)
      gains += int(myValues[index])

    # Step 13: Then check odd/even: user has it? If yes, get the bet and
multiply by its factor
    if ball % 2 == 0:
      ballEven = "even"
    else:
      ballEven = "odd"

    if ballEven in myBet:
```

```
      index = myBet.index(ballEven)
      gains += int(myValues[index])

    # Step 14: Then check for high/low: user has it? If yes, get the bet and
multiply by its factor
    if ball > 18:
      ballHigh = "high"
    else:
      ballHigh = "low"

    if ballHigh in myBet:
      index = myBet.index(ballHigh)
      gains += int(myValues[index])

    # Step 15: Finally check for dozens: user has it? If yes, get the bet and
multiply by its factor
    if ball <= 12:
      ballDozen = "dozen1"
    if ball >= 25:
      ballDozen = "dozen3"
    else:
      ballDozen = "dozen2"

    if ballDozen in myBet:
      index = myBet.index(ballDozen)
      gains += int(myValues[index]) * 2
```

So, instead of keeping the ifs in the code, how about “moving” it to a function? Imagine a function that receives three parameters: ball, bet and value. And just return the gains!
Another variation: we will compare the characteristics of the number with our bets and not the bets individually. I can even create a list of types and their factors of multiplication to make things easier!

a) Part 1: create the function and test with a “hard” list (not created by the user)

So let’s get the steps:

```
Step 1: list all the possible bets, factors and colors (boring…!)
Step 2: receive the parameters and create the function
Step 3: create list of characteristics and gains = 0
Step 4: analyze the number: odd or even? Put it in a list.
Step 5: analyze the number and get the color
Step 6: analyze the number: high/low? Put it in a list.
Step 7: analyze the number: dozen? Put it in a list.
Step 8: compare this list with the bets list
Step 9: if it is there, get factor and sum the gains
Step 10: return gains
```

```
ball = 34
bet = ["34","odd","red","dozen1"]
values = [100,120,130,140]

# Step 1: list all the possible bets, factors and colors (boring…!)
allBets =
["0","1","2","3","4","5","6","7","8","9","10","11","12","13","14","15","16","1
```

```
7","18","19","20","21","22","23","24","25","26","27","28","29","30","31","32",
"33","34","35","36","red","black","odd","even","high","low","dozen1","dozen2",
"dozen3"]

factors =
[35,35,35,35,35,35,35,35,35,35,35,35,35,35,35,35,35,35,35,35,35,35,35,35,35,35
,35,35,35,35,35,35,35,35,35,35,35,1,1,1,1,1,1,2,2,2]

colors = ['green',
'red','black','red','black','red','black','red','black','red','black','black',
'red','black','red','black','red','black','red','red','black','red','black','r
ed','black','red','black','red','black','black','red','black','red','black','r
ed','black', 'red']

# Step 2: receive the parameters and create the function
def getGains(ball, bet, values):

  # Step 3: create list of characteristics and gains = 0
  ballProfile = [str(ball)]  # To compare with the bet!
  gains = 0

  # Step 4: analyze the number: odd or even? Put it in a list.
  if ball % 2 == 0:
    ballProfile.append("even")
  else:
    ballProfile.append("odd")

  # Step 5: analyze the number and get the color
  ballProfile.append(colors[ball])

  # Step 6: analyze the number: high/low? Put it in a list.
  if ball > 18:
    ballProfile.append("high")
  else:
    ballProfile.append("low")

  # Step 7: analyze the number: dozen? Put it in a list.
  if ball <= 12:
    ballProfile.append("dozen1")
  if ball >= 25:
    ballProfile.append("dozen3")
  else:
    ballProfile.append("dozen2")
  print(ballProfile)
  # Step 8: compare this list with the bets list
  for temp in ballProfile:
    for tempBet in range(len(bet)):
      # Step 9: if it is there, get factor and sum the gains
      if temp == bet[tempBet]:
        gains += int(values[tempBet]) * factors[allBets.index(temp)]

  # Step 10: return gains
  return gains

getGains(ball, bet, values)
```

b) You may be thinking: ok, but the code is almost the same size of the other! Yes, it is, but you remove from the main code:

```
    ##########
    # FROM THE MAIN CODE!!!
    ##########

    ...
    ...
    ...

    # Step 10: Roll the roulette! Get a random number between 0 and 36
    ball = random.randrange(0,37)

    # Step 11: Call the getGains function!
    gains = getGains(ball, myBets, myValues)

    ...
    ...
```

8.3. Exercises

```
# Exercise 1
# Python does not have a function to calculates the sum of the items of a
given list
# Build one that returns the result (no printing!)
```

```
# Exercise 2
# Python also does not have a function to calculates the mean of the items of
a given list
# You may use the function above
# Build one that prints the result
```

```
# Exercise 3
# Remember the funcion unique() used in dataframe to get the distinct values
in a column?
# Create an unique function that works with lists: Example:
# if originalList = [1,2,3,3,3,4,5,6,7,7,7]
# The function returns newList = [1,2,3,4,5,6,7]
```

```
# Exercise 4
# Write a function that returns (not prints!) a boolean variable (True/False)
if a number is prime or not.
```

```
# Exercise 5
# Write a function that receives two parameters:
# a) a date in format dd/mm/yyyy
# b) the region
# If region is 1, returns yyyymmdd; else, returns mm-dd-yyyy.
```

```
# Exercise 6
# Suppose you have a program where the user inputs a date several times.
```

```
# In Brazil, we use the format dd/mm/yyyy
# Write the function checkDate() to see if the date is valid
# Hint: just check lenght, month and days (don't bother with leap years!)

# I'll create a list to store the max days of the months
# We can access inside the function! :-)
maxMonth = [31,29,31,30,31,30,31,31,30,31,30,31]
```

```
# Exercise 7
# Previous exercise complement:
# Suppose now that you program needs to calculate the difference of the date
given
# and 25/10/1985, Marty McFly's first travel in time

# Hint: to calculate this difference, transform the string in a date
# and subtract one from the other
```

```
# Exercise 8
# You have a list of DVDs and needs to know how many discs are there:
# http://profalibania.com.br/python/dvdCollection.xlsx
# You know that:
#   a) simple = 1 disc
#   b) Special Edition = 2 discs
#   c) 3D = 2 discs
#   d) series = 6 discs
#   e) series - sitcom = 4 discs

# You have a file that contains a list of titles and their types.
# Create a function that read each type and return the quantity of discs
# Print the total number of discs.
```