Python Exercises*

September-October 2023

1 Days 1-3. Language basis

1.1 Day 1. Python console: data types, operators and built-in functions

Table 1: Python basic data types.

int	1 -3 -1000
float	12.34 -0.234
bool	True False
str	"hi there!"
	'hi there!'
	"""hi there!"""
list	[1, '23', True]

Table 2: Python operators.

Arithmetic	+ - * / ** // %
Assignment	= += -= *= /= //= %=
Comparison	== != > >= < <=
Logical	and or not
Identity	is is not
Membership	in not in
Bitwise	...

Table 3: Python built-in functions.

abs()	bin()	chr()	dir()
divmod()	help()	hex()	input()
len()	max()	min()	oct()
ord()	pow()	print()	range()
round()	sum()	type()	

Exercises Solve the following exercises using a Python console –either in a command terminal or the Python interpreter within PyScripter–:

1. Write a multiline string with some verses of a song you like.
2. Add two strings, such as "Call me when you get home" and "Don't forget to buy milk!". Yes, you need to introduce a blank in between by adding it as well.

*Plenty of these exercises have been extracted from *The Python Workbook* by Ben Stephenson, Springer.

3. Now it's time to repeat sentences: "I'll be back in a minute!" repeated 10 times.

4. Calculate how many chocolate bars will recive each of the 3 friends when sharing 19 chocolate bars. And how many chocolate bars left over?

5. Compute $(x^3 - y^2 - 10)^{-2/5}$ for two given real values x and y.

6. Test if the three sides of a triangle are equal. Now, find the condition to test if a triangle is escalene.

7. Round a real value –such as $\pi = 3.1415927$– to three decimal digits.

8. Find the minimum value of 3 real numbers. And now the minimum of 3 strings

9. Create a list containing 5 numbers. Add all the numbers in the list.

10. Calculate the ASCII code for letters 'a', 'b' 'c' and 'z'

11. Calculate the character with ASCII code 97.

12. Lest convert from lowercase to uppercase: i) find the ASCII code for a letter, ii) find and substract the ASCII code of letter 'a', iii) add the ASCII code of letter 'A', and iv) find the character for the obtained result.

1.2 Day 2. Python program editing: Variables and data types

Variables: storing data!

You must name them; a variable name must start with a letter or an '_'; then, the name can include digits, letters and '_'. In Python we use dynamic typing.

Exercises

Solve the following exercises using a Python console –either in a command terminal or the Python interpreter within PyScripter–:

1. Create a variable called *dpy* –days per year– and assign it to 365.

2. Create a variable called *num_years*, set it to a positive integer number.

3. Create a variable called *age*, set it to your age, including the number of months. For instance, 18 years 3 months corresponds to $18 + 3/12$.

4. Compute the number of days in num_years –by multiplying this variable times dpy. The result is an integer: check it using the buitl-in function *type*.

5. Compute the number of days you have lived as dpy times age. Notice that the result is a real value: test this using the buitl-in function *type*.

6. Determine the ratio between num_years and age. Please, note the result is a real number as long as one of the operands is a real value.

1.3 Days 2-3. Python I/O and fancy strings

Prompting from keyboard using the *input* function to read a **string**. You can sshow a personalized message as well.

Type these expressions in a Python console; check the results: are they what you expected?

- Example 1 :

```
resp = input("The message in a bottle was ... ")
type(resp)
resp
```

- Example 2 :

```
resp = input("Enter your age, please ")
type(resp)
resp
```

- Example 3

```
resp = int( input("Enter your age, please "))
type(resp)
resp
```

Important: input ALWAYS returns a str. Transformation is needed whenever a different data type is required, either by means of a casting (example 3) or through a code that transforms the text.

Printing information to the screen using the *print* function. Separators and ending strings can be modified. Also, the flux can be redirected to a file.

Type these expressions in a Python console:

- Example 1: printing data and variables.

```
var1 = 34
var2 = "no way!"
print("Texto", -10.2, [1,2,3], var1, var2)
```

- Example 2: changing the separators.

```
var1 = 34
var2 = "no way!"
print("Texto", -10.2, [1,2,3], var1, var2, sep=" =|= ")
```

- Example 3: changing the end of the print.

```
print("Sin nueva linea", end=" <-- texto final   ")
print("Este tampoco...", end=" <-- check it!  ")
print("Este tiene nueva linea")
```

Important: the precision of numbers and the length of strings can be defined. We will check fancy strings to do that, but print has methods to set these options.

Fancy strings or f-strings

allows to mix fixed text together with dynamic values. They are equivalent to *str.format()*, but with a radical different approach. f-strings has the following format:

- The most basic case: notice the f letter before the string.

```
n = 3.4
t = 'This is my life'
b = True
print(f'Un numero: {n}; {t} es str, {b} es bool')
print(f'Mezcla variables y constantes: {t} y {3.45}')
```

  Each value to insert into the str must be surrounded in braces ({, }).

- f-strings allows formatting issues as shown in this next example.

```
n = 1234.12345678
print(f'{n}; {n:,}, {n:16.3f}, {n:1.2%}, {n:+1.2f}')
print(f'{n:.0f}, {n:0>10.3f}')
print(f'{10:0>5d} {10:x>5d}, {10:x<5d}')
print(f'|{12:^20d}|')
```

- f-strings also has formatting issues with text, as follows.

```
print(f'|{"text":<20}|, |{"text":->20}|, |{"text":^20}|')
```

- f-strings has the = operator that shows the left-hand part of the formatting expression.

```
print(f'{h= :>10d}')
```

Exercises

Solve the following exercises writing the code in a python file using PyScripter. Always try to print as much information as possible, so the use of formatting issues is implicit.

1. In many jurisdictions a small deposit is added to drink containers to encourage people to recycle them. In one particular jurisdiction, drink containers holding one liter or less have a 0.10 deposit, and drink containers holding more than one liter have a 0.25 deposit.
   Write a program that reads the number of containers of each size from the user. Your program should continue by computing and displaying the refund that will be received for returning those containers –both partial and total values–. Format the output so that it includes a dollar sign and always displays exactly two decimal places.

2. In many countries, the restaurants charges their clients with the price of the meal together with the tax and the tip for that meal. Write a program that reads the price of the meal, computes the tax and the tip, and shows the three values and the grand total –formatting them with 2 decimal places—. Use the 21% as the tax, and 18% as the tip rate (calculated without the tax).

3. Write a program that reads to integers, a and b, computing and displaying: i) the sum of a and b, ii) the difference when b is substracted from a, iii) the quotiend and remainder of a divided by b, iv) the $\log_{10}$ of a, and v) a^b. You need to import the math package for using the logarithm function.

4. In USA, the efficiency of a motor is measured in MPG –miles per gallon– , while in Europe this is measured L/100 Km. Write a program that translates the read MPG efficiency to the European unit. You must find out the equivalences that apply.

5. Because the curvature of the earth surface, distances between degrees of longitude varies with the latitude; hence, the Pithagoras theorem does not apply. The distance between two earth's points (t_1, p_1) and (t_2, p_2) is calculated as stated in Eq. 1, with $\rho = 6371.01$ being the average radius of the Earth in kilometers. Write a program that reads the latitude and longitude of two points on the average earth surface, computing and showing the distance among them.

$$distance = \rho \times \arccos(\sin(t_1) \times \sin(t_2) + \cos(t_1) \times \cos(t_2) \times \cos(g_1 - g_2)) \quad (1)$$

6. Consider the software on a self-checkout machine and its capacity to determine the change to provide to a customer. Write a program that reads the amount of euros to return, rounding it to two decimal places and converting it to cents. Then, the program must determine the amount of each type of coin to provide: 1€, 50¢, 20¢, 10¢, 5¢, 2¢, and 1¢. The change should be given using as few coins as possible.

7. Given that one foot is 12 inches and one inch is 2.54 centimeters, write a program that reads a height in meters, producing the height in foot and inches.

2 Day 4. Conditionals

Conditionals represents the first tool for flow control within a program: how to implement different actions according to the conditions that hold. The syntax is simple, as the example included in the python documentation. There can be 0, 1 or more elif divertions; similarly, there can be 0 or 1 else conditions.

Syntax :

```
if_stmt ::=  "if" assignment_expression ":" suite
             ("elif" assignment_expression ":" suite)*
             ["else" ":" suite]
```

Example :

```
x = int(input("Please enter an integer: "))

if x < 0:
    x = 0
    print('Negative changed to zero')
elif x == 0:
    print('Zero')
elif x == 1:
    print('Single')
else:
    print('More')
```

Exercises

Solve the following exercises writing the code in a python file using PyScripter. Always try to print as much information as possible, so the use of formatting issues is implicit.

1. It is said that a dog's year is equivalent to 7 human years. However, it fails ni reaching to adulthood: dogs reach it in 2 years, while humans need 18 or 21 –it depends on the country–. So, some authors claim that a better equivalence could be using 10.5 human years for the firs 2 dog's years, and 4 human years for each additional dog's year. Write a program that given a dog's age, prints the equivalent human's age for each of the two equivalence options.

2. Table 4 shows the decibel levels for several common noises. Write a program that given a decibel level, prints the specific noise that is generating that level of sound. Whenever the decibel level is between two listed noises, the program must display a message telling which noises the level is between. Ensure messages for noise lower or higher than thos included in the table.

3. Table 5 shows the frequencies f_4^N) for each music note N at the 4th octave. For a note N in a different octave o (with o from 0 to 8), you can calculate

Noise	Decibel level (dB)
Jackhammer	130
Gas lawnmower	106
Grandpa's alarm clock	70
Quiet room	40

Table 4: Decibel level for common noises.

Table 5: Relación entre notas musicales y su frecuencia.

Note	Frequency (Hz)	Note	Frequency (Hz)
C4-Do_4	261.63	**G4**-Sol_4	392.00
D4-Re_4	293.66	**A4**-La_4	440.00
E4-Mi_4	329.63	**B4**-Si_4	493.88
F4-Fa_4	349.23		

the corresponding frequency as $f_o^N = 2^{4-o} \times f_4^N$. Write a program that prompts for a note and octave –i.e., "D5"–, printing the corresponding frequency of note D at octave 5.

4. Take a look to chess board in Fig. 1. Write a program that, given the row number and column id, displays whether the cell is WHITE or BLACK.

   Requirement: limit rows and columns id to the given limits. Whenever incorrect values are introduced, print an error message and end the program.

   Hint: odd rows are equal, and so are the even rows.

   Hint: use function *ord* to obtain the ASCII code for the column id. Again, odd columns are equal, and so are the even columns.

5. The chinese zodiac calendar assigns an animal to each year according to the 12-years cicle shown in Table 6. Write a program that prints the

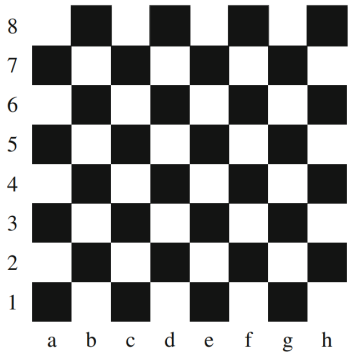

Figure 1: Chess board.

Table 6: Chinese's calendar cycle.

Year	Animal	Year	Animal
2000	Dragon	2006	Dog
2001	Snake	2007	Pig
2002	Horse	2007	Rat
2003	Sheep	2007	Ox
2004	Monkey	2007	Tiger
2005	Rooster	2007	Hare

animal that correspond to a year prompted by the user.

6. Who says that before life was better? Several years ago, a typical cell plan charged even the SMS. Here is an example of that era...

   A particular cell phone plan includes 50 minutes of air time and 50 text messages for $15.00 a month. Each additional minute of air time costs $0.25, while additional text messages cost $0.15 each. All cell phone bills include an additional charge of $0.44 to support 911 call centers, and the entire bill (including the 911 charge) is subject to 5 percent sales tax.

   Write a program that reads the number of minutes and text messages used in a month from the user. Display the base charge, additional minutes charge (if any), additional text message charge (if any), the 911 fee, tax and total bill amount. Only display the additional minute and text message charges if the user incurred costs in these categories. Ensure that all of the charges are displayed using 2 decimal places.

3 Days 5-6. Loops

While loops

Syntax :

```
while_stmt ::=  "while" assignment_expression ":" suite
                ["else" ":" suite]
```

Example :

```
a, b = 0, 1
while a < 10:
    print(a)
    a, b = b, a+b
```

For loops

Syntax :

```
for_stmt ::=  "for" target_list "in" starred_list ":" suite
              ["else" ":" suite]
```

Example :

```
words = ['cat', 'window', 'defenestrate']
for w in words:
    print(w, len(w))

for i in range(5):
    print(i)

a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)):
    print(i, a[i])
```

Break, continue and else clauses

- **break** ends the loop, so it jumps to the next expression after it –of course, out of the loop–.
- **continue** ends the iteration, returning to the loop expression in search of the next repetition, if possible.
- The code within an **else** after a loop is executed once the loop ends – either because the condition of repetition doesn't hold anymore, it has been reached the number of repetitions, ...–. However, with a break, the else clause is not run.

Exercises

Solve the following exercises writing the code in a python file using PyScripter. Always try to print as much information as possible, so the use of formatting issues is implicit.

1. Write a program that displays a temperature conversion table for degrees Celsius and degrees Fahrenheit. The table should include rows for all temperatures between 0 and 100 degrees Celsius that are multiples of 10 degrees Celsius. Include appropriate headings on your columns. The formula for converting between degrees (C)elsius and degrees (F)ahrenheit is $F = C \times 9/5 + 32$.
2. Write a program to compute the perimeter of a geometrical figure. Two possible methods to compare; in both cases, the length of a side is determined as the distance between two consecutive vertixes:
   (a) Ask the number of sides, then request the $< x, y >$ as real number for each of the vertices –storing the first vertex so it could be possible to calculate the last side–.

(b) Without requesting the number of sides, ask for the vertices coordinates until a blank line is introduced.

3. A zoo charges their customer with the fees shown in Table 7. For each person in the queue, the system request if it is an individual or a group ticket. In case of an individual, the system request the age and show the fees. Otherwise, the system starts requesting the age of each person in the group, displaying the total fee for the group. Write a program that allows to print the cost of the ticket –either individual or group–.

Table 7: Fees used in a zoo.

Age	**Fee**
< 3	Free
$\in [3, 12]$	5.00 €
$>= 65$	Free
Otherwise	20.0€

4. A parity bit is a simple mechanism for detecting errors in data transmitted over an unreliable connection such as a telephone line. The basic idea is that an additional bit is transmitted after each group of 8 bits so that a single bit error in the transmission can be detected.

   Parity bits can be computed for either even parity or odd parity. If even parity is selected then the parity bit that is transmitted is chosen so that the total number of one bits transmitted (8 bits of data plus the parity bit) is even. When odd parity is selected the parity bit is chosen so that the total number of one bits transmitted is odd.

   Write a program that computes the parity bit for groups of 8 bits entered by the user using even parity. Your program should read strings containing 8 bits until the user enters a blank line. After each string is entered by the user your program should display a clear message indicating whether the parity bit should be 0 or 1. Display an appropriate error message if the user enters something other than 8 bits, or if not only 0 and 1 are given in the string.

   Hint: read strings from the keyboard. Then use the *len* function and the *count* method to find the number of 0's and of 1's.

5. The value of π can be approximated by the infinite series shown in Eq. 2. Write a program that displays 15 approximations of π. The first approximation should make use of only the first term from the infinite series. Each additional approximation displayed by your program should include one more term in the series, making it a better approximation of π than any of the approximations displayed previously.

$$\pi = 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} + \frac{4}{6 \times 7 \times 8} - \cdots \tag{2}$$

4 Days 7-8. User-Defined Functions

Syntax :

```
def nombre(pos-arg0, pos-arg1, kwd0 = 123, kwd1 = "hi"):
    code

def nombre(pos-arg0, pos-arg1, /, pos_kwd0, *, kwd-only):
    code
```

Example :

```
def cheeseshop(kind, *arguments, **keywords):
    print("-- Do you have any", kind, "?")
    print("-- I'm sorry, we're all out of", kind)
    for arg in arguments:
        print(arg)
    print("-" * 40)
    for kw in keywords:
        print(kw, ":", keywords[kw])

cheeseshop("cat", 123, 456, '123', '456', edad = 34,
           nombre = "magdaleno")
```

Solve the following exercises writing the code in a python file using PyScripter. Always try to print as much information as possible, so the use of formatting issues is implicit.

1. In a particular jurisdiction, taxi fares consist of a base fare of 4.00 €, plus 0.25 €for every 140 meters traveled. Write a function that takes the distance traveled (in kilometers) as its only parameter and returns the total fare as its only result; in addition, let the fares be also parameterized. Write a main program that demonstrates the function.

2. Let's write a function to generate random passwords. The function must receive the length of the password. Moreover, whether captions, digits, and symbols are required, together with the valid symbols to used are also given –allowed symbols are '!?-_'–; these parameters must have a default value (i.e. True, True, True). Then, the function will generate a password accomplishing with the constraints given as parameter. Write a program that calls this function several times with different parameter subsets, checking its behaviour.

    - Can you extend your function, so the allowed symbols are also given as a parameter? **Hint**: you may need to use indexes on that parameter... ask your teacher.

3. Write a function to check whether a password is valid or not. The function shall receive the password to test, but also if lower and uppercase letters,

symbols –and which symbols–, and digits are required. Write a program calling this function.

- Use this function within the function that generates passwords, rewriting what might be needed. Test which option is faster and which one is more reliable.

4. Write a function to determine the greates common divisor of two positive integers. Then, using this funciton, write a second funtion that reduces fractions to lowest terms. This second function receives the numerator and denominator, returning two integers with the reduced numerator and denominator. For instance, calling this function with 30 and 45 produces the tuple (2, 3) as 15 is the mcd.

5. We are writing a tool for assisting cooks following recipes. The idea that when a recipe es scaled for more serves, the tool shall receive the scaled measurements and return it using the largest possible units. We are using volume measurements from the imperial system, so a cup is equivalent to 16 tablespoons, and one tablespoon equal 3 teaspoons. Therefore, 62 teaspoons leads to 3 cups, 4 tablespoons and 2 teaspoons. Similarly, 50 tablespoons is simplified to 3 cups and 2 tablespoons.

6. Write a function to determine whether a date is magic date or not. A magic date is the one that the number of day times the number of month equals the two-digit year. So, February 11, 2022 is a magic date. Write a program that prints all the magic dates for the current century. Consider that leap years have 29 days in february. Also consider the corresponding number of days for each month! that there are months with 30 and 31 days.

5 Days 9-10. Lists and strings

Lists

Lists are used to store several data entities in an ordered sequence, so you can modify its content. They are defined as comma separated values delimited by brackets, so

```
[1, 2, 3, 4]
```

is a list containing four integer numbers.

Each element in a list has a position –a positive integer from 0 and on–, which can be used in indexing. So, the next code defines a list, indexing on it to modify its content and to print one of the elements.

Table 8 shows the most common operators and functions used with lists, while Table 9 shows some useful methods for lists.

```
a = [1, 2, 3, "This is a string"]
a[2] = "Trucutu"
print(a[3])
print(a)
```

Table 8: Some operators and defined functions for lists.

Function	is for...
len(a)	obtaining the number of elements in a
a[p]	brackets are used for indexing, p is an integer such that $-len(a) < p < len(a)$
val in a	True if a includes val
a + b	concatenates two lists
a * 3	repeates the list 3 times
max(a)	returns the maximum in a
min(a)	returns the minimum in a
sum(a)	sums the values in a if all numerical

Table 9: Some useful list's methods.

Method	is for...
a.append(v)	adds v at the end of a
a.count(v)	counts the number of occurrences of v in a
a.extend(s)	adds s at the end of a
a.index(o)	returns the position of o in a
a.insert(p, o)	inserts o at position p in a
a.pop()	deletes the last element in a
a.remove(o)	deletes the first occurrence of o in a
a.reverse()	swaps the elements in a
a.sort()	sorts the elements in a

Strings (str)

Strings store characters –either letters, digits or symbols–. They are inmutable, so it can not be chenged once defined. You define a string using quotes (’), double-quotes (") or triplets of them –for a multi-line string– as delimiters. Examples:

```
a = "1 es un texto !!!!"
b = '1 esto es otro texto :-)'
c = """ Este texto
ocupa varias
lineas"""
```

Función	Parar
len(a)	number of elements in a
a[p]	[] operator for indexing, p ∈ {0, ..., len(a)-1}
ch in a	True if ch is contained in a
a + b	concatenates two str
a * 3	repeates a 3 times
max(a)	returns the symbol with the higher ASCII code
min(a)	returns the symbol with the lower ASCII code

Método	Para
a.**count(s)**	returns the number of occurrences of s in a
a.**find(s,)**	returns the position of s within a, -1 if not found
a.**replace(s, n)**	returns a copy of a with the occurrences of s replaced with n, s and n two str
a.**isalnum()**	returns True when a only contains letters and digits
a.**isalpha()**	returns True when the str only contains letters
a.**isdigit()**	returns True if the str a only include digits
a.**split(s)**	splits str a by the occurences of str s, returning a list of str
a.**splitlines()**	create a list with the strs obtained from a by splitting by the "\n" character
a.**join(seq)**	creates a copy of the str in the sequence seq, joining them with the str a
a.**lower()**	returns a copy with lowercase letters
a.**upper()**	returns a copy with uppercase letters

Indexing on list and str

Remember that, in the context of str and lists, by indexing we refer to accessing either an element in a certain position or a sequence of elements between two positions. We can index using:

- A non-negative integer p is a position if $0 \leq p < len(myStr)$. Enables to index an element counting positions from the first element.

```
a = "abcde"
b = [1, 11, 111, 1111, 11111]
print(a[2], b[2]) # prints c 111
```

- A negative integer p is a position if $-len(myStr) \leq p < 0$. Allows indexing elements from the last element.

```
a = "abcde"
b = [1, 11, 111, 1111, 11111]
print(a[-1], b[-2]) # prints e 1111
```

- It is possible to index several elements using **slices**; in this case, the colon caracter (:) delimites the initial and last position –last position is not included in the slice–:

```
a = "abcde"
b = [1, 11, 111, 1111, 11111]
print(a[1:3], b[-4:-1]) # prints bc [11. 111, 1111]
```

Exercises

Solve the following exercises writing the code in a python file using PyScripter. Always try to print as much information as possible, so the use of formatting issues is implicit.

1. We want to get rid of the outliers from a numerical list. To do so, we will follow the rule that any number n not in $[\mu - 3 \times \sigma, \mu + 3 \times \sigma]$ is an outlier, and must be removed from the list.

   Write the following functions:

   - Function **average** that computes the average of all the values in a list.
   - Function **std** that computes the standard deviation of the values in a list.
   - Function **remove_all** that receives a value and a list, returning the list with all the occurrences of the value removed from the list.
   - Function **outliers_remover** similar to the previous one, but updating the list by removing all its outliers.
   - Function **outliers_filter** that receives a list, returning a copy of it –without modifying the received list– instead.

   Write a program for testing all of these functions.

2. We want to start reading str from the keyboard until the empty str is received. For each str, we will split it using the blanks; the non-empty elements –those with a length higer than 0– are assumed words.

   We want a code for counting the occurrences of each word, as well as storing the order in the sequence where the word appears.

   As an example, given the following inputs:

```
Escribe algo: Cancion  el patio de mi casa
Escribe algo: El Patio      de MI CaSa es particular
Escribe algo:
```

   then, the outcome of the code could be:

```
Palabra: cancion, cuenta: 1, posiciones: [0]
Palabra: el, cuenta: 2, posiciones: [1, 6]
Palabra: patio, cuenta: 2, posiciones: [2, 7]
Palabra: de, cuenta: 2, posiciones: [3, 8]
Palabra: mi, cuenta: 2, posiciones: [4, 9]
Palabra: casa, cuenta: 2, posiciones: [5, 10]
Palabra: es, cuenta: 1, posiciones: [11]
Palabra: particular, cuenta: 1, posiciones: [12]
```

3. Complete this exercise with the following modifications:

   (a) Function **only_words** that receives a str and returns a copy containing letters only. But when English possessive apostrophes (such as for elephant in *elephant's* house) or negation's contraction forms (such as don't), they are accepted. So for *elephant's!* the function returns *elephant's*, for *elephants'* it returns *elephants'*, and for *don't* it returns *don't*. But for *elephant:* we get *elephant*.

   (b) Modify the code from the previous exercise so we only admit well-formed words.

4. Write a program that reads integers until a blank line is entered. The program must print all the negative numbers, followed by the zeros, and ending with the positive numbers –all of them in the same order in the sequence–. So, when the inputs are 4, -1, 3, 0, 1, -2, -1, and 0, then the outcome would be -1, -2, -1, 0, 0, 4, 3, 1.

5. **DIFFICULT!!!** **This is the most difficult exercise of this collection, but it is a nice and complete one; take it as a challenge!**

   The memory game! So we have a board of 4×5 cells; in each cell we store a symbol, we are using letters from 'A' to 'E' as valid symbols, and a blank (' ') when the cell is empty.

   Anyway, you can try to solve the exercises individually, without bothering to finish the game; in this way, you have plenty of exercises.

   Here is one possible main code for soving the problem, use it as a guide for solving the problem –at least, you already have the function calls–:

```
#################################################################
###### S T A R T   O F   T H E   C O D E ################
import random
import time

GENERAL_WAIT_TIME = 5  # wait time to clean the console
TIME_BEFORE_REQUESTING_POS = 2
```

```
                    # wait time while showing the state
EMPTY_CELL = ' '              # value for the empy cell
SYMBOLS ='ABCDE'             # symbols used in the board

# Y O U R   C O D E   G O E S   H E R E ! ! !
# Y O U R   C O D E   G O E S   H E R E ! ! !

M, N = 4, 5                  # board dimensions
num_players = 2              # number of players
players = [0] * num_players # score of each player
board = empty_board(M, N)    # the board to show
goal = init_board(M, N, "ABCED")
                             # the symbols to guess

init_message(num_players, M, N)  # initial message
time.sleep(GENERAL_WAIT_TIME)    # just waits

show_state(board, M, N, players)
                          # shows the state of the game

# While we have not finished the game
#   For each player
#      ask for 2 EMPTY and VALID positions on the board
#      analyse the result: is there a match or not?
#      show the result of the go
#      update the score of the corresponding player
#      show the state of the game
while not test_game_over(players, M, N):
    for player in range(num_players):
        go = request_go(M, N, player, board)
        result = go_result(board, goal, go)
        show_go(board, goal, M, N, go, player, result)
        update_players(players, player, result)
        show_state(board, M, N, players)

# Prints who the winner is and the final board
print_winner(players, board)
########## E N D   O F   T H E   C O D E ################
#########################################################
```

A board is a list of lists of symbols. For instance, the following listing represents a valid board:

```
[['D', 'D', 'B', 'B', 'D'],
 ['C', 'E', 'A', 'B', 'A'],
 ['E', 'E', 'A', 'C', 'C'],
 ['E', 'B', 'A', 'C', 'D']]
```

And here are the functions you must develop (almost a function for each code line in the main code shown above,; take a look to the call to each of these functions). Feel free to implement more functions if you feel it is better for developing an easier solution.

- Function **empty_board**: receives the board's dimensions and returns a board with all the cells set to EMPTY_CELL.
- Function **init_board**: receives the board's dimensions and a str with the admitted symbols. It generates and returns the board to guess, with a symbol in each cells. Consider that the number of repetitions of each symbol must be even when filling in the board; in this way, there will not be unmatched symbols in a game. **Important**: the number of rows times the number of columns must be even.
- Function **print_board**: receives a board and prints it to the console. See the next example of a possible outcome, notice that rows are numbered from 0 to 3 (that is, $4-1$) and that columns from 0 to 4 (that is, $5-1$). The position of a cell is given with its row and column numbers.

```
  | 0 | 1 | 2 | 3 | 4
--------------------
0 |   |   |   |   |
--------------------
1 |   |   |   |   |
--------------------
2 |   |   |   |   |
--------------------
3 |   |   |   |   |
```

- Function **clear_screen**: prints 100 blank lines ("\n"), so it seems the screen is cleaned.
- Function **request_go**: propmts the user for the position of two cells to compare.Hence, it receives the ID of the player –integer between 0 and $(num\_players - 1)$–, the board dimensions and the board to compare.

  This function request the 2 different positions, checking that they are valid and that both cells are empty (containing EMPTY_CELL). The function shall request as many times as needed to ensure 2 valid positions are obtained.

  This function returns a list including the 2 positions, each position is a list with 2 integers, the first one for the row and the second for the column.
- Function **init_message**: shows the instructions of the game, the board dimensions and the number of players –all of them are parameters to the function–.

- Function **go_result**: determining whether a go is a match –the 2 positions refers to cells with the same content– or a fail –otherwise– . It receives the playing board, the board to guess, and the two positions from the go. If the contents of the two cells to guess are the same the function returns True; otherwise, it returns False.
- Function **show_go**: shows the outcome from a go, printing the two valid positions, the result of the go, and the board just after the go.

  The playing board, the board to guess, the board dimensions, the go –as a list of positions–, the player's ID, the result from the go represent the parameters to this function.

  These are the steps of the algorithm of this function:

  (a) Clean the screen.
  (b) Show the positions of the go.
  (c) Display a message indicating if the go is a success –a match– or a fail.
  (d) Copy the contents of the board to guess' cells into the corresponding cells of the playing board.
  (e) Prints the playing board to screen.
  (f) If there were no match, set the content of the two cells from the playing board to the EMPTY_CELL.
  (g) Wait GENERAL_WAIT_TIME seconds. This is performed using the ***time*** package. The program stops 5 seconds if it calls **time.sleep(GENERAL_WAIT_TIME)**.
- Function **update_players**: receives the list of matches for each player, the current player's ID (an integer higher or equal to 0), and the result from the go. The function increments the counter for the current player in 1 when the go gas successful.
- Function **show_state**: receives the playing board, the board's dimensions, and the list of matches for each player. This function cleans the scree, shows the playing board in the screen, and the number of matches for each player. After waiting TIME_BEFORE_REQUESTING seconds –using time.sleep–, the function ends.
- Function **print_winner**: that receives the list with the matches for each player and the playing board. After cleaning the screen, it shows the player that wins –the one with the higher number of matches–. In case of a draw, the program explains that there is match and shows the number of matches for each player.

6 Day 11. File I/O

Some issues we need to remember:

- Files have a type: text files –the default type– and binary. With the binary type, we transfer bytes instead of strings. So we need to proceed accoding to the file format definition to read and write data with the suitable width.
- We open a link to a file using the built-in function *open*. We close this link with the *close* method.
  - We can open for reading (mode set to 'r'), for writing (mode set to 'w') or both (mode set to 'rw'). When writing, we can edit an existing file using the 'a'ppend mode (mode set to 'a') instead; in this case, the writing cursos is placed at the end of the file to avoid overwriting.
- For reading, we use the *read([integer])*, *readline()* or *readlines()* methods; while for writing we use the *write()* method.
- We can move the reading or the writing cursor through the file using the *seek()* method.
- Using the *with open(file,mode) as varname* structure avoids forgetting to close the links.

Interesting links are:

- Core tools for working with streams.
- Open built-in function.

Exercises

Solve the following exercises writing the code in a python file using PyScripter. Always try to print as much information as possible, so the use of formatting issues is implicit.

1. The friends' data base! Write a program that request the user for the name (text), age (real number), and phone number (text) of his/her friends. Before that, the program must ask for the name of a text file where all this information will be stored. The program must be storing the data into the text file, delimiting the fields using commas (','). Type an empty str as the input for a name to end the program. Here is an excerpt of a possible file's content; notice the headers for each field:

```
name,age,phone
Globito Rojo, 12, 111111111
Quijote de la Mancha,510,012345678
Gambito Guarabito,55,987564654
Celestina Ranja, 23, 985232323
```

2. Using the text file generated in the previous exercise –or just creating a file with the contents shown above–, write a program that, given the name of the file, reads it and produces a new file. This new file includes the line number followed by a colon symbol (':') and a blank. Example using the possible friends' data base:

```
0: name,age,phone
1: Globito Rojo, 12, 111111111
2: Quijote de la Mancha,510,012345678
3: Gambito Guarabito,55,987564654
4: Celestina Ranja, 23, 985232323
```

3. Again, we are using the last exercise example. Write a program that reads a text file with numbered lines –and format as in the previous example– and produces a listing of the friends to the screen.

   - Extend this program so the listing is also stored in a file. Chech the outcome is equal to the data text file obtained from the first exercise.
   - Consider the file can have unlimited number of lines, so the ': ' pattern might not be at position 1 or each string.

4. Modify the last program so the fields are separated by a single blank in all the cases, neither comma nor extra blank are allowed If you haven't solved the previous exercise, use the example of outcome for the first exercise as the inputing data file. You can use the split, join, and strip methods of a str (see the str documentation).