Ayudantía FPI / FCyP 13 de Julio: Archivos de Texto + Repaso

Ayudante: John Serrano - john.serrano@usach.cl

1 Listas en Python

Las listas en Python son una forma de agrupar elementos. Podemos realizar distintas operaciones con listas y los elementos de estas listas pueden ser Integers, Float, Booleanos, Strings e incluso sub-listas. **Debemos tener en consideración que una lista SIEMPRE se escribe entre paréntesis cuadrados ([]) y los elementos se separan utilizando comas (,)**

Lista	3	2	1	4
Posicion	0	1	2	3

Los elementos de una lista se cuentan desde la posición **0** hasta la posición **n-1**, donde **n** es la cantidad de elementos que contiene una lista (También conocido como el largo de una lista).

Podemos utilizar la función **len(nombre_lista)** para poder obtener el largo de una lista, donde **nombre_lista** es el nombre de la variable que contiene a la lista.

1.1 Definir una lista vacía

En Python podemos definir una lista inicialmente vacía igualando la variable a utilizar a **[]**

[1]:
```
# En Python podemos definir una lista inicialmente vacia igualando la variable a␣
↪utilizar a []
lista_vacia = []
print("Una lista vacia es: ", lista_vacia)

# Tambien podemos definir una lista que ya viene con elementos
lista = [3,4,2]
print("Una lista con elementos es: ", lista)
```

```
Una lista vacia es:  []
Una lista con elementos es:  [3, 4, 2]
```

1.2 Unir dos listas y repetir los elementos de una lista

Podemos unir dos listas haciendo una "suma" (+) de ellas. Podemos repetir elementos de una lista utilizando la multiplicación.

[2]:
```
lista1 = [3,4,2]                          # Se crea una lista con 3, 4 y 2
lista2 = [9,5,6]                          # Se crea una lista con 9, 5 y 6
print("La lista 1 es: ", lista1)
print("La lista 2 es: ", lista2)
lista3 = lista1 + lista2                  # Se obtiene la union de ambas listas
print("La union de ambas listas es: ", lista3)
lista4 = lista1 * 3                       # Se repiten 3 veces los elementos de␣
 ↪lista 1
print("La lista que contiene los elementos de lista 1 repetidos 3 veces es: ",␣
 ↪lista4)
```

```
La lista 1 es:  [3, 4, 2]
La lista 2 es:  [9, 5, 6]
La union de ambas listas es:  [3, 4, 2, 9, 5, 6]
La lista que contiene los elementos de lista 1 repetidos 3 veces es:  [3, 4, 2,
3, 4, 2, 3, 4, 2]
```

1.3 Acceder a un elemento específico de una lista

Para acceder a un elemento de una lista, podemos utilizar **nombre_lista[x]**, donde **nombre_lista** corresponde a la variable que contiene a la lista y **x** corresponde a la posición del elemento al que queremos acceder.

[3]:
```
lista = [3,4,2]                           # Se crea una lista con 3, 4 y 2
print("La lista es: ", lista)
elemento = lista[1]                       # Se accede al elemento de la posicion 1␣
 ↪de la lista (4)
print("Accediendo al segundo elemento de la posicion 1 de la lista: ", elemento)
```

```
La lista es:  [3, 4, 2]
Accediendo al segundo elemento de la posicion 1 de la lista:  4
```

1.4 Modificar un elemento específico de una lista

[4]:
```
lista = [3,4,2]                           # Se crea una lista con 3, 4 y 2
print("La lista es: ", lista)
elemento = lista[1]                       # Se accede al elemento de la posicion 1␣
 ↪de la lista (4)
print("Accediendo al segundo elemento de la posicion 1 de la lista: ", elemento)
lista[1] = 99
print("La lista modificada es: ", lista)
elemento = lista[1]                       # Se accede al elemento de la posicion 1␣
 ↪de la lista (99)
print("Accediendo al segundo elemento de la posicion 1 de la lista: ", elemento)
```

```
La lista es:  [3, 4, 2]
Accediendo al segundo elemento de la posicion 1 de la lista:  4
La lista modificada es:  [3, 99, 2]
Accediendo al segundo elemento de la posicion 1 de la lista:  99
```

1.5 Agregar elementos a una lista

Podemos utilizar el método **nombre_lista.append(x)** para agregar un elemento a una lista, donde **nombre_lista** es el nombre de la variable que contiene a la lista y **x** es el elemento que queremos agregar a la lista. **Append** agrega el elemento automáticamente al final de la lista (O en otras palabras, en la última posición)

[5]:
```
# Utilizamos .append() para agregar elementos a una lista

lista = []                              # Utilizamos [] para definir una lista␣
 ↪vacia
print("La lista inicial es: ", lista)
lista.append(2)                         # Agregamos el numero 2 a la lista
print("La lista luego del primer append es: ", lista)
lista.append(3)                         # Agregamos el numero 3 a la lista
print("La lista luego del segundo append es: ", lista)
```

```
La lista inicial es:  []
La lista luego del primer append es:  [2]
La lista luego del segundo append es:  [2, 3]
```

También podemos utilizar el método **nombre_lista.insert(x,y)**, donde **nombre_lista** es el nombre de la variable que contiene la lista, **x** es la posición de la lista donde se quiere agregar un elemento e **y** corresponde al elemento a agregar.

[6]:
```
lista = [3,4,2]                                 # Se define una lista con 3, 4 y 2
print("Lista antes de insert", lista)
lista.insert(2, 10)                             # Se agrega un 10 en la penultima␣
 ↪posicion de la lista
print("Lista despues de insert", lista)
```

```
Lista antes de insert [3, 4, 2]
Lista despues de insert [3, 4, 10, 2]
```

Notemos que con estas operaciones y la gran mayoría de las que vienen a continuación, no es necesario igualar a una nueva variable. Una vez se ejecuta la línea de código, Python hace el cambio dentro de la variable original!

1.6 Quitar elementos de una lista

El método **nombre_lista.remove(x)** nos permite eliminar un elemento de una lista, donde **nombre_lista** es el nombre de la variable que contiene a la lista y **x** es el elemento a eliminar de la lista. **Remove** elimina solamente la primera instancia que encuentre de un elemento, por lo que no elimina todas las instancias inmediatamente!

[7]:
```
lista = [3,4,2,3,4,2]                           # Se define una lista con 3, 4,␣
 ↪2, 3, 4 y 2
print("Lista antes de los remove", lista)
lista.remove(3)                                 # Se elimina el primer 3
print("Lista luego del primer remove", lista)
```

```
lista.remove(3)                                    # Se elimina el segundo 3
print("Lista luego del segundo remove", lista)
```

```
Lista antes de los remove [3, 4, 2, 3, 4, 2]
Lista luego del primer remove [4, 2, 3, 4, 2]
Lista luego del segundo remove [4, 2, 4, 2]
```

También podemos utilizar el método **nombre_lista.pop(x)**, donde **nombre_lista** es el nombre de la variable que contiene a la lista y **x** es la posición del elemento que se quiere eliminar. **Si no se especifica un x, elimina al último elemento de la lista.**

[8]:
```
lista = [3,4,5,2,3]                                # Se crea una lista con 3, 4, 5, 2␣
 ↪y 3
print("Lista antes de los pop", lista)
lista.pop()                                        # Se elimina el ultimo elemento de␣
 ↪la lista (3)
print("Lista luego del primer pop (Elimina el último 3)", lista)
lista.pop(3)                                       # Se elimina el elemento de la␣
 ↪posicion 3 (2)
print("Lista luego del segundo pop (Elimina el 2)", lista)
```

```
Lista antes de los pop [3, 4, 5, 2, 3]
Lista luego del primer pop (Elimina el último 3) [3, 4, 5, 2]
Lista luego del segundo pop (Elimina el 2) [3, 4, 5]
```

1.7 Contar cuantas veces se repite un elemento de una lista

Para contar cuantas veces se repite un elemento en una lista, podemos utilizar el método **nombre_lista.count(x)**, donde **nombre_lista** es el nombre de la variable que contiene a la lista y x es el elemento que queremos saber cuántas veces se repite dentro de la lista

[9]:
```
lista = [3,4,2,2,2,2,2,2,1]                        # Se crea una lista con 3, 4, 2,␣
 ↪2, 2, 2, 2, 2 y 1
print("La lista es: ", lista)
contador = lista.count(3)                          # Se cuenta las veces que se␣
 ↪repite el 3
print("Veces que se repite el 3: ", contador)
contador = lista.count(2)                          # Se cuenta las veces que se␣
 ↪repite el 2
print("Veces que se repite el 2: ", contador)
```

```
La lista es:  [3, 4, 2, 2, 2, 2, 2, 2, 1]
Veces que se repite el 3:  1
Veces que se repite el 2:  6
```

1.8 Obtener la posición de un elemento de una lista

Podemos utilizar el método **nombre_lista.index(x)** para obtener la posición de un elemento de una lista, donde **nombre_lista** es el nombre de la variable que contiene a la lista y **x** es el elemento

al cual le queremos averiguar la posición dentro de la lista. Si un elemento se repite, entonces entrega la primera posición donde se encuentra el elemento.

[10]:
```
lista = [3,4,2,4,2]                                    # Se crea una lista con 3, 4, 2,␣
 ↪4 y 2
print("La lista es:", lista)
posicion = lista.index(2)                              # Se obtiene la posicion del␣
 ↪primer 2 (2)
print("Posicion de el 2 en la lista:", posicion)
```

```
La lista es: [3, 4, 2, 4, 2]
Posicion de el 2 en la lista: 2
```

1.9 Ordenar una lista, obtener su máximo, su mínimo y la suma de elementos

Podemos ordenar una lista de elementos utilizando el método **nombre_lista.sort()**, donde **nombre_lista** es el nombre de la variable que contiene a la lista. Además, podemos utilizar las funciones **max(nombre_lista)**, **min(nombre_lista)** y **sum(nombre_lista)** para poder obtener el máximo, mínimo y la suma de elementos correspondientes de una lista.

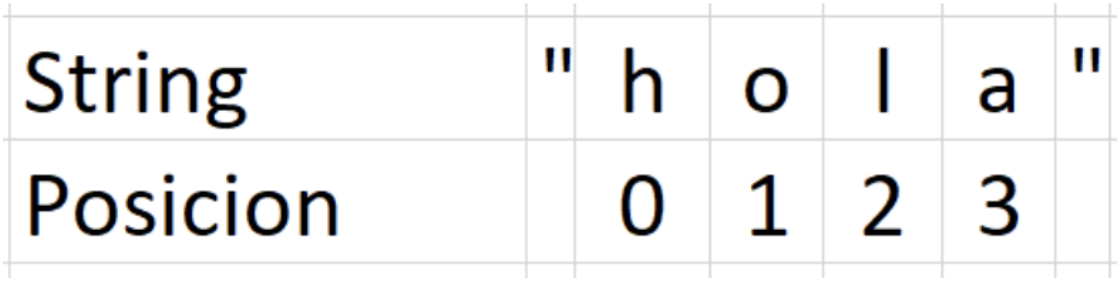
[11]:
```
lista = [9,6,3,1,2]                                    # Se crea una lista con␣
 ↪9, 6, 3, 1 y 2
print("La lista original es: ", lista)
lista.sort()                                           # Se ordena la lista␣
 ↪utilizando sort()
print("La lista ordenada de menor a mayor es: ", lista)
maximo = max(lista)                                    # Se obtiene el máximo␣
 ↪de la lista
print("El máximo número de la lista es: ", maximo)
minimo = min(lista)                                    # Se obtiene el minimo␣
 ↪de la lista
print("El minimo numero de la lista es: ", minimo)
suma = sum(lista)                                      # Se obtiene la suma de␣
 ↪los elementos de la lista
print("La suma de los elementos de la lista es: ", suma)
```

```
La lista original es:  [9, 6, 3, 1, 2]
La lista ordenada de menor a mayor es:  [1, 2, 3, 6, 9]
El máximo número de la lista es:  9
El minimo numero de la lista es:  1
La suma de los elementos de la lista es:  21
```

2 Strings en Python

Los Strings corresponden a caracteres o cadenas de textos. En palabras simples, son letras o palabras. Se escriben utilizando comillas dobles o comillas simples. Comparten algunas características con las listas. También se cuentan desde 0 hasta **n-1** elementos y podemos obtener el largo del

string utilizando **len(nombre_string)**, donde **nombre_string** es el nombre de la variable que contiene al string.



2.1 Unir dos strings

Similar a las listas, podemos utilizar la suma para unir dos strings. Los strings se unen exactamente en la última y primera posición del primer y segundo string respectivamente.

[12]:
```
string1 = "hola "                          # Se crea el string "hola "
print("El primer string es: ", string1)
string2 = "mundo"                          # Se crea el string "mundo"
print("El segundo string es: ", string2)
union = string1 + string2                  # Se crea la union de ambos strings␣
 ↪()"hola mundo")
print("La union de ambos strings es: ", union)
```

```
El primer string es:  hola
El segundo string es:  mundo
La union de ambos strings es:  hola mundo
```

Nótese que en el primer string el último elemento es un espacio para que asi haya una separación entre hola y mundo. De lo contrario, el resultado hubiera sido **holamundo** (Sin el espacio entremedio)

2.2 Repetir un string

Similar a las listas, utilizando la multiplicación, podemos repetir un string x veces.

[13]:
```
string1 = "hola!"                    # Se crea el string "hola!"
string2 = string1 * 5                # Se repite el string "hola!" 5 veces.
print("El string original es: ", string1)
print("El string repetido 5 veces es: ", string2)
```

```
El string original es:  hola!
El string repetido 5 veces es:  hola!hola!hola!hola!hola!
```

2.3 Acceder a un elemento específico de un string

Similar a las listas, podemos acceder a un elemento especifico de un string utilizando la forma **nombre_string[x]**, donde **nombre_string** es el nombre de la variable que contiene al string y **x** es la posición del elemento al que se quiere acceder.

[14]:
```
string1 = "hola mundo"                # Se crea el string "hola mundo"
elemento = string1[5]                 # Se accede al elemento de la posicion 5 del␣
 ↪string ("m")
print("El string original es: ", string1)
print("El elemento en la posicion 5 del string es: ", elemento)
```

```
El string original es:  hola mundo
El elemento en la posicion 5 del string es:  m
```

A diferencias de las listas, no podemos utilizar lo anterior para modificar un elemento especifico de una lista. Una alternativa a eso es transformar el string a lista y luego transformarlo a String, algo que se mostrará más adelante

2.4 Eliminar espacios y saltos de línea de un string

Podemos utilizar el método **nombre_string.strip()** para eliminar los saltos de línea (****) o los espacios al inicio y al final de un string.

[36]:
```
string1 = "    USACH  en  Paro\n"
print("El string original es: ", string1)
string2 = string1.strip()
print("El string modificado con strip es:", string2)
```

```
El string original es:      USACH  en  Paro

El string modificado con strip es: USACH  en  Paro
```

Nótemos que **si es necesario guardar el resultado de strip en una nueva variable!**

2.5 1. ¿Como transformo un string a una lista?

Para esto podemos utilizar el método **nombre_string.split(x)**, donde **nombre_string** es el nombre de la variable que contiene el string y **x** es el elemento donde se quiere que se separen los elementos para transformarlos a elementos de una lista. Si no se especifica un x, entonces se consideran los espacios como el elemento separador.

[16]:
```
string1 = "hola mundo"                                # Se crea el string "hola␣
 ↪mundo"
lista = string1.split()                               # Se transforma el string␣
 ↪a lista, separando los elementos por " "
print("El string 1 original es: ", string1)
print("El string 1 transformado a lista es: ", lista)

string2 = "13/07/2024"                                # Se crea el string "19/10/
 ↪2022"
lista2 = string2.split("/")                           # Se transforma el string␣
 ↪a lista, separando los elementos por "/"
print("El string 1 original es: ", string2)
print("El string 1 transformado a lista es: ", lista2)
```

```
El string 1 original es:  hola mundo
El string 1 transformado a lista es:  ['hola', 'mundo']
El string 1 original es:  13/07/2024
El string 1 transformado a lista es:  ['13', '07', '2024']
```

2.6 2. ¿Como transformo una lista a un string?

Para esto podemos utilizar el método **elemento.join(nombre_lista)**, donde **elemento** es un elemento que se unirá con cada elemento de la lista y **nombre_lista** es el nombre de la variable que contiene a la lista

[19]:
```
lista1 = ["hola " , "como ", "estas"]              # Se crea una lista con "hola␣
 ↪", "como ", "estas"
string1 = "".join(lista1)                          # Se transforma la lista a␣
 ↪string
print("La lista original es: ", lista1)
print("La lista transformada a string es: ", string1)
```

```
La lista original es:  ['hola ', 'como ', 'estas']
La lista transformada a string es:  hola como estas
```

2.7 3. ¿Para que funciona el eval()?

eval() funciona para que Python pueda identificar mediante como está escrita una entrada a que tipo de dato corresponde el elemento ingresado. Esto es esencial si queremos ingresar listas por entradas y ahorrarnos posibles problemas al transformar de strings a listas.

[37]:
```
entrada1 = eval(input("Ingrese un elemento: "))            # Se pido que se␣
 ↪ingrese un elemento por teclado
print("El tipo de dato de la entrada es: ", type(entrada1))
```

```
Ingrese un elemento:  [2, 3, 4, 5, 10, 9]
```

```
El tipo de dato de la entrada es:  <class 'list'>
```

Como Python ve que el elemento ingresado está escrito con paréntesis cuadrados al inicio y al final del elemento y además de que tenemos elementos separados por coma (,), entonces inmediatamente sabe que se trata de una lista y eval() retorna la lista escrita tal cual como la ingresamos.

2.8 4. ¿Que son las "banderas" (flags) y como funcionan?

Las banderas (o flags, como se le puede encontrar en algunos códigos) son solamente variables de tipo Booleano. Usualmente parten con un valor y a medida que se va ejecutando el código cambian a otro valor (Se sube o se baja la "bandera", dependiendo si es True o False. Por eso se les llama como bandera).

[21]:
```
```

```
lista = eval(input("Ingrese una lista por entrada: ")) # Se pide que se ingrese␣
 ↪un elemento por teclado
flag_dos = False                                       # Declaramos una flag␣
 ↪(Variable booleana) inicializada en False
i = 0                                                  # Se declara un contador␣
 ↪para recorrer
while i < len(lista):                                  # Mientras i sea menor al␣
 ↪largo de la lista
    elemento_actual = lista[i]                         # Se obtiene el elemento␣
 ↪en la posicion i de la lista
    if elemento_actual == 2:                           # Si el elemento es igual␣
 ↪a 2
        flag_dos = True                                # La bandera se vuelve␣
 ↪verdadera
    i += 1                                             # Se aumenta i para que␣
 ↪continue el ciclo While

if(flag_dos):                                          # Si se encontro 2
    print("Se pudo encontrar el numero dos")
else:                                                  # Si no se encontro 2
    print("No se encontro el numero dos")
```

```
Ingrese una lista por entrada:  [3, 5, 6, 8, 2, 1, 4, 3, 8]

Se pudo encontrar el numero dos
```

Como podemos observar en el código anterior, la bandera comienza con el valor Falso y luego cuando se encuentra el número dos, cambia a verdadero. Es ahí donde "se sube la bandera". Flag, o Bandera, no es nada más que un nombre a como usualmente se le conocen a estas variables Booleanas. Pero debemos no asustarnos y considerar que solo son Variables que tiene el valor True o False inicialmente y que luego en algún punto del programa, cambian de valor. Nada más, y nada menos.

3 Importación de Funciones

Ya hemos utilizado Funciones en Python. A estas funciones se les conocen como **"Funciones Nativas"**, pues no es necesario definirlas ni importalas, ya vienen creadas y listas para utilizar por defecto en Python.

Algunos ejemplos son: * print() * input() * str() * int() * float() * list() * bool() * eval() * max() * min() * sum() * len()

Debemos tener en consideración que no nos interesa como estan definidas estas funciones. Solo nos interesa que recibe como entrada, que devuelve como salida y que es lo que hace.

Las **funciones importadas** provienen de paquetes, modulos o librerias que son llamadas dentro de un código. Por buenas prácticas es buena idea seguir la siguiente estructura de código ahora que estamos hablando de funciones:

[22]:
```
# BLOQUE DE DEFINICION
# IMPORTACION DE FUNCIONES
# DEFINICION DE FUNCIONES
# DEFINICION DE CONSTANTES

# BLOQUE PRINCIPAL
# ENTRADA
# PROCESAMIENTO
# SALIDA
```

Hay dos formas de realizar estos llamados. **La primera no es muy recomendada** y consiste en llamar a todo el paquete / modulo / libreria. Como ejemplo, vamos a utilizar la libreria "math" (Funciones matemáticas).

Esta forma sigue el siguiente formato: **import nombre_libreria**, donde **nombre_libreria** es el nombre de la libreria que queremos importar. Las funciones se llaman usando el formato **nombre_libreria.funcion**

[23]:
```
# BLOQUE DE DEFINICION
# IMPORTACION DE FUNCIONES
import math # Importamos toda la libreria math

# BLOQUE PRINCIPAL
# PROCESAMIENTO
resultado = math.cos(math.pi)    # Se llama a cos de la forma math.cos
# SALIDA
print(resultado)
```

```
-1.0
```

Nótese que estas importaciones no solo traen funciones, si no que a veces traen constantes también.

La segunda forma es mas recomendada y consiste en llamar solo lo que necesitamos de la libreria. Sigue el siguiente formato: **from nombre_libreria import nombre_funcion**, donde **nombre_libreria** es el nombre de la libreria que queremos importar y **nombre_funcion** es el nombre de la funcion a importar. Con esta forma solo llamamos a las funciones con su nombre.

[24]:
```
# BLOQUE DE DEFINICION
# IMPORTACION DE FUNCIONES
from math import pi          # Importamos pi de la libreria math
from math import sqrt        # Importamos sqrt de la libreria math
from math import sin

# BLOQUE PRINCIPAL
# PROCESAMIENTO
resultado = sqrt(pi)         # Se llama a la funcion sin mencionar a la libreria
# SALIDA
print(resultado)
```

```
1.7724538509055159
```

Existen librerias / paquetes que no vienen con python. Estos deben instalarse mediante **pip**. Si tenemos pip instalado, entonces debemos abrir una consola (CMD) y escribir: **pip install nombre_paquete**, donde **nombre_paquete** es el nombre del paquete a instalar.

4 Definición de Funciones

Ahora vamos a crear nuestras propias funciones. Para ello debemos utilizar algunas palabras reservadas: * **def:** Sirve para definir una función. * **return:** Sirve para indicar que devuelve la función

[25]:
```
# BlOQUE DE DEFINICION
# IMPORTACION DE FUNCIONS
from math import pi
# DEFINICION DE FUNCIONES

def obtener_maximo(lista):
    '''
    Entrada: Una lista de numeros enteros
    Salida: Un numero entero correspondiente al maximo de una lista
    Descripcion: Funcion que recorre una lista de numeros enteros y obtiene
    su maximo sin utilizar la funcion nativa max().
    '''
    i = 0
    maximo = lista[0]
    while i < len(lista):
        if lista[i] > maximo:
            maximo = lista[i]
        i += 1
    return maximo


def operar_cada_elemento(lista):
    '''
    Entrada: Una lista de numeros enteros
    Salida: Una lista de numeros flotantes
    Descripcion: Funcion que recorre una lista de numeros enteros y
    multiplica cada elemento con el valor de PI. Devuelve la lista modificada.
    '''
    i = 0
    while i < len(lista_numeros):
        lista.append(lista_numeros[i] * PI)
        i += 1
    return lista

# DEFINICION DE CONSTANTES
PI = pi       # Se define PI como una constante

# BLOQUE PRINCIPAL
```

```python
# ENTRADA
lista_numeros = eval(input("Ingrese una lista de numeros enteros: "))

# PROCESAMIENTO
i = 12
maximo = obtener_maximo(lista_numeros)  # Se llama a la funcion obtener_maximo
lista_numeros2 = operar_cada_elemento([])  # Se llama a la funcion␣
↪operar_cada_elemento
maximo2 = obtener_maximo(lista_numeros2)   # Se llama a la funcion obtener_maximo

# SALIDA
print("La lista original es: ", lista_numeros)
print("El maximo de esta lista es: ", maximo)
print("\n")
print("La lista al llamar a la funcion operar_cada_elemento es: ",␣
↪lista_numeros2)
print("El maximo de esta lista es: ", maximo2)
```

```
Ingrese una lista de numeros enteros:  [3, 1, 9, 27, 8, 13, 10]
```

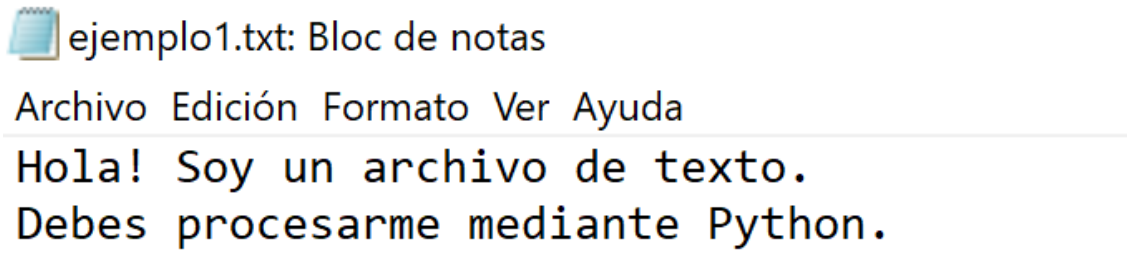
```
La lista original es:  [3, 1, 9, 27, 8, 13, 10]
El maximo de esta lista es:  27


La lista al llamar a la funcion operar_cada_elemento es:  [9.42477796076938,
3.141592653589793, 28.274333882308138, 84.82300164692441, 25.132741228718345,
40.840704496667314, 31.41592653589793]
El maximo de esta lista es:  84.82300164692441
```

- **Variable local:** Variable que solo existe dentro de la función, por lo que una vez que se acaba el procedimiento de la función, se eliminan.
- **Variable global:** Variable que existe para todo el código y las funciones son capaces de acceder a estas variables.

También es posible importar funciones definidas en un archivo Python para utilizar en otro archivo Python.

5 ¿Que son los Archivos de Texto?

Son archivos cuya extensión es **.txt** y contienen una cantidad especifica de texto. Si bien hemos utilizado input() para las entradas, a veces queremos pasar información por otro medio distinto a la consola, es ahí donde entran a jugar un papel clave los Archivos de Texto. Lo mismo pasa para la salida, si queremos mostrar resultados por otro medio que no sea la consola, podemos dar un resultado en un Archivo de Texto. Un Archivo de Texto es de tipo de dato **FILE**.



Es importante siempre guardar los archivos .py y .txt con los que trabajemos en la misma carpeta, para así evitar problemas de directorios.

6 ¿Como puedo abrir un archivo .txt en Python?

Para abrir un archivo en Python, se debe utilizar la función nativa **open()**, la cual recibe dos entradas: * El nombre del archivo en formato String, incluye su extensión (Por ejemplo, "archivo1.txt") * Un modo de procesamiento del archivo, también en formato String.

Para este curso, se consideran tres modos de procesamiento. Los cuales son: * **r**: Modo de lectura del archivo. Solo se lee el archivo, pero no se modifica de ninguna forma. * **w**: Modo de escritura del archivo. Si ya existe un archivo con el nombre indicado, se borra todo el contenido y se escribe el nuevo contenido, o bien, si no existe el archivo con el nombre indicado, se crea un nuevo archivo con el contenido indicado y el nombre. * **a**: Modo de concatenación. Se mantiene el contenido original del archivo y **directamente al final** del archivo se agrega el nuevo contenido. Independiente de que metodo utilicemos para procesar el archivo, siempre se deben cerrar los archivos de texto. Para ello, se utiliza el metodo **nombre_archivo.close()**

Actualmente, en el curso se permite el uso de **with open(nombre_archivo, modo) as nombre_variable**. Cabe señaar que al utilizar esta forma, no es necesario cerrar el archivo.

7 El modo lectura

Para leer el contenido de un archivo, se puede utilizar el método **nombre_archivo.readline()**, donde **nombre_archivo** es el nombre del Archivo de Texto. Este método lo que hace es leer una linea del Archivo de Texto y cada vez que se lee una linea, el cursor del archivo se cambia a la siguiente linea.

[26]:
```
archivo = open("ejemplo1.txt", "r") # Se lee el archivo de texto "ejemplo1.txt"
linea = archivo.readline()          # Se lee la primera linea. El cursor se␣
 ↪mueve a la siguiente linea.
print("Linea 1:", linea)            # Se imprime la primera linea del archivo.

linea = archivo.readline()          # Se lee la segunda linea. El cursor se␣
 ↪mueve al final del archivo (No hay mas lineas)
print("Linea 2:", linea)            # Se imprime la segunda linea del archivo.

archivo.close()                     # Se cierra el archivo de texto.
```

```
Linea 1: Hola! Soy un archivo de texto.

Linea 2: Debes procesarme mediante Python.
```

[27]:
```
with open("ejemplo1.txt", "r") as archivo:
    linea = archivo.readline()
    print("Linea 1:", linea)
    linea = archivo.readline()
    print("Linea 2:", linea)
```

```
Linea 1: Hola! Soy un archivo de texto.

Linea 2: Debes procesarme mediante Python.
```

Utilizando un ciclo While, podemos guardar todo el contenido del archivo de una variable.

[28]:
```
archivo = open("ejemplo1.txt", "r") # Se lee el archivo de texto "ejemplo1.txt"
texto_archivo = ""
linea = archivo.readline()
while linea != "":
    texto_archivo += linea
    linea = archivo.readline()
archivo.close()
print(texto_archivo)
```

```
Hola! Soy un archivo de texto.
Debes procesarme mediante Python.
```

Pero lo anterior se puede hacer de una manera mucho mas facil y simple. Podemos utilizar el metodo **nombre_archivo.readlines()** para así leer inmediatamente todo el contenido del archivo. Pero debemos tener en consideración que **readline()** entrega un **String**, en cambio, **readlines()** entrega una **lista de Strings**. Sin embargo, ambos metodos incluyen en el contenido el salto de linea, representado por el string ‘’

[29]:
```
archivo = open("ejemplo2.txt", "r") # Se lee el archivo de texto "ejemplo1.txt"
lineas = archivo.readlines()
archivo.close()
print(lineas)
```

```
['El mes es Julio\n', 'El mes de las vacaciones\n', 'Pero nos fuimos a paro']
```

[39]:
```
archivo = open("ejemplo2.txt", "r")
lineas = archivo.read()
archivo.close()
print(lineas)
```

```
<class 'str'>
```

8 Modo de escritua

Para escribir en un Archivo de Texto, se debe utilizar el metodo **nombre_archivo.write(lineas)**, donde **lineas** es el contenido del archivo a escribir, en formato String. **Solo 1 String**.

[40]:
```
archivo = open("nuevo.txt", "w")
archivo.write("He creado un Archivo\nY es de texto.")
archivo.close()
```

Recordemos que si el archivo no existe, entonces se creará. Si ya existe, se borrará todo el contenido y luego se agregará el contenido indicado.

9 Modo de concatenación

Similar al modo de escritura, nuevamente podemos utilizar el metodo **nombre_archivo.write(lineas)**. Aplican las mismas reglas, pero debemos tener en consideración que el nuevo contenido se agregará **inmediatamente después del final del archivo original**.

[42]:
```
archivo = open("nuevo.txt", "a")
archivo.write("\nHe creado un nuevo Archivo")
archivo.close()
```

10 Ciclo For in

El ciclo For in es una nueva forma de iterar bastante util cuando se trabaja con archivos. Sigue la siguiente estructura: * For **variable** in **elemento_iterable** * **variable** puede ser cualquier nombre. Usualmente y por buenas practicas, el nombre debe hacer alusión al contenido del elemento iterable. * **elemento_iterable** es cualquier elemento que se puede recorrer, como por ejemplo, listas.

[33]:
```
lista = [3,4,1,2]

for numero in lista:
    numero = numero + 100
    print(numero)
```

```
103
104
101
102
```

For in también es una herramienta muy potente cuando estamos trabajando con **listas bidimensionales**. Supongamos que tenemos una matriz 3x3 y queremos imprimir todos sus elementos.

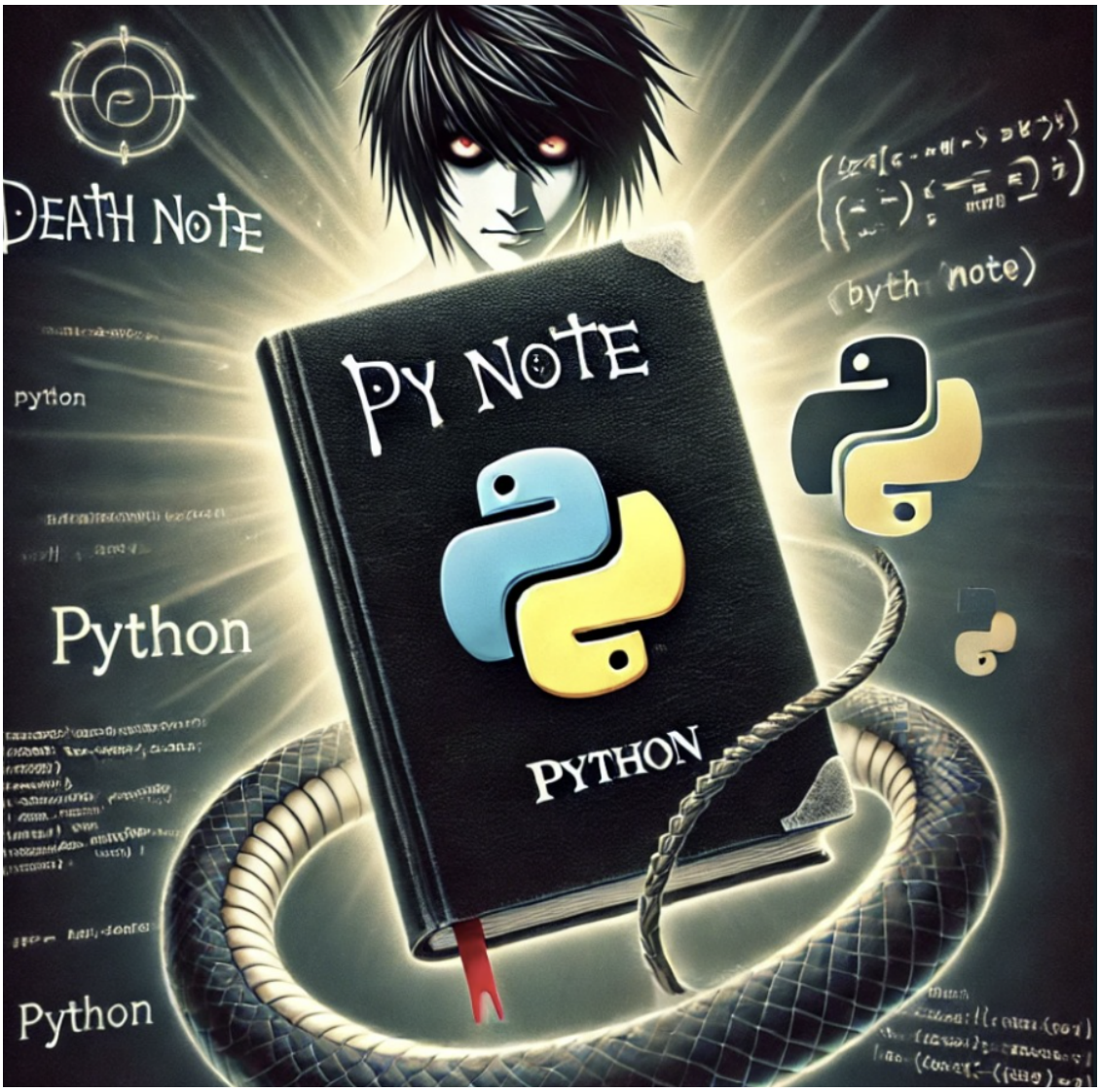
[43]:
```
matriz = [[9,5,2], [3,1,2], [3,4,5]]
for fila in matriz:
    print(fila)
```

```
[9, 5, 2]
[3, 1, 2]
```

```
[3, 4, 5]
```

11 EJERCICIOS

Pyuk, el Dios de la reprobacion es dueño de la PyNote, un archivo de texto capaz de reprobar alumnos con solo escribir su nombre en el. Las primeras cuatro líneas del archivo son encabezados y luego cada línea tiene el nombre, la nota final y la causa de reprobacion, en algunos casos, solo estara escrito el nombre.



PyNote.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
PyNote
================================================
Aquel que su nombre sea escrito, reprobara.
================================================
Hernesto Yagami#19:Se quedara dormido para la POR.
Juan Manzana#10:Copiara en el LAB.
Fabiola Misa#39:Entregara el archivo .py de otro alumno.
Aquiles Baeza
Lorenzo Lawliet#24:Usara diccionarios en la PEP2.
Nate Rivera#20:Tendra problemas estomacales.
Naruto Izimiki#0:Se lo merece.
Mario Verde#39:No escribira su nombre en la PEP2.
Juan Nieves#34:Lo acuchillaran camino a la universidad.
Wilfredo Watari#23:Pisara el escudo del patio de los perros y GGWP.
...
```

1. Dado que Pyuk reprueba alumnos por monton, suele olvidar los alumnos que ha reprobado. Desarrolle la función **reprobado(nombre,archivo)** que retorna la causa de reprobacion. Si no hay causa de reprobacion debe retornar el string **"Perdio las ganas de aprobar"**. Por ultimo, si el nombre no aparece en la lista, retornar **False**.

2. Desarrolle la función **agregar(lista, archivo)**, que reciba una lista con el nombre, nota (int) y la causa de reprobación o solo el nombre de un nuevo alumno. La función debe agregar una línea con la información de la lista al final de la PyNote.

3. Las reglas de la PyNote dicen que si un nombre es tachado antes de reprobar, este aprobará todos sus ramos. Escriba la función **tachar(nombre, archivo)** que modifique el nombre de la persona en el archivo, tachando todas sus letras con un guion (-). **Considere que el nombre siempre estara en el archivo.**

```
[ ]:
```