

PYTHON

PARA MIS COMPAÑEROS



Víctor Hernando de Juan
V.1. - Junio 2021

AGRADECIMIENTOS

A Raquel, por ser lo máspreciado que tengo y por la ilustración de la portada.
A Reme y Beatriz, por siempre estar ahí para apoyarme.
A José Manuel y Daniel, por introducirme en este mundo.
A mis compañeros y profesores, por estar conmigo en mis primeros pasos.
A todos los profesionales del mundo IT, para que puedan afrontar este sector tan cambiante.

INTRODUCCIÓN

Este libro está pensado como una guía para aprender Python 3, para aquellas personas que ya saben programar. No encontrarás explicaciones de programación básica, solo de como se hace en este lenguaje, y de lo que lo diferencia de otros como Java o JavaScript. Aquellos puntos que como programador me han resultado más llamativos, o en aquellas cosas en las que vea gran potencial o importancia.

Está dividido en 4 secciones de diferente dificultad. Según avanza por ellas, me paro a explicar aquello que considere necesario que su dificultad lo requiere: Python básico, Python avanzado, Algoritmia avanzada con Python, y Frameworks, IDEs y bibliotecas.

Este libro no ha sido escrito por un alumno de C.F. de Grado Superior en Desarrollo de Aplicaciones Multiplataforma, del IES Virgen de la Paloma, como proyecto del Modulo de Proyecto, y que todos los conocimientos han sido adquirido mediante investigación, la cual está reflejada en el guión de la práctica.

Está escrito de forma muy ligera al principio, no deteniéndome en explicar miles de funciones, o el proceso de instalación, se da por hecho que si eres programador sabrás buscar esa información, igual que con cualquier otro lenguaje de programación.

Por último desearos suerte, si yo he conseguido hacer este libro, tú podrás aprender este lenguaje.

Seguiré subiendo versiones mejoradas de este libro en:

<https://github.com/VictorHernando/Guias-Lenguajes-Programacion.git>

ÍNDICE

Página -02- ---Python básico
Página -07- ---Python avanzado
Página -11- ---Algoritmia avanzada
Página -14- ---Frameworks, IDEs y bibliotecas



PYTHON BÁSICO

Creado por Guido van Rossum a finales de los ochenta, y su nombre referencia a los Monty Python.

Python es un lenguaje simple, de fácil lectura, indentado (tabulado), interpretado, fuertemente tipado aunque dinámico, sensible a mayúsculas, de alto nivel, de propósito general, multiplataforma, multiparadigma, open source, incrustable en C y C++, con muchas librerías y frameworks. La elección del IDE lo dejo a elección de cada uno, hay decenas en internet.

VARIABLES

Son dinámicas, pueden cambiar el tipo del dato que contienen al contrario que Java, al declararlas no se pone el tipo de dato ($a = 8 \Rightarrow a = \text{"hola"}$). Pero el tipo de dato del valor que contienen solo puede cambiar de forma explícita con castings, al contrario que JavaScript.

Los nombres de las variables pueden llevar de A-Z, a-z, 0-9 y `_`, y no pueden empezar por un número. No pueden tener espacios en blanco, por lo que se usa el snake_case: usar `_` en lugar de espacios.

La asignación o reasignación es con `=` como en la mayoría de lenguajes. No se pone `;` al final de las líneas, el final de una línea es el salto de línea (tecla enter).

TIPOS DE DATOS BÁSICOS

Booleanos: True y False (con la primera letra en mayúscula).

Entero: Número sin decimales, tanto positivo, negativo y 0.

Real: Número con decimales, tanto positivo, negativo y 0.

Complejo: Número con parte imaginaria ($8j$).

Cadenas: Caracteres entre `"`. Como en Java tiene muchas funciones.

Ante la duda del tipo de dato que tiene dentro la variable: **type(var)**

<code>boolean = False</code>	<code>print(type(boolean))</code>	<code><class 'bool'></code>
<code>entero = -23</code>	<code>print(type(entero))</code>	<code><class 'int'></code>
<code>real = 2.75</code>	<code>print(type(real))</code>	<code><class 'float'></code>
<code>complejo = 4 + 3j</code>	<code>print(type(complejo))</code>	<code><class 'complex'></code>
<code>cadena = "Hola"</code>	<code>print(type(cadena))</code>	<code><class 'str'></code>

ENTRADA Y SALIDA POR CONSOLA

Para imprimir los valores de las variables sin tener que convertirlas a string se puede usar la f-String (**f"Pon las variables así {var}"**).

Función **print(string)**, imprime por consola.

Función **input(string)**, imprime por consola y se queda esperando la entrada de datos, hasta que se da a la tecla "enter". Sin datos devuelve "".

<code>entrada = input("Introduce valor: ")</code>	Introduce valor: 55
<code>print(f"El valor introducido es {entrada}")</code>	El valor introducido es 55



TIPOS DE DATOS AVANZADOS

Tuplas: Lista inmutable, entre "()", con los elementos separados por ",". Peculiaridades: Aunque solo tenga un elemento, se declara con una coma igualmente (tupla = (8,)), los valores se pueden consultarse según su índice y pueden volcarse en variables, no se pueden añadir, eliminar o modificar los valores de dentro de la tupla.

```
notupla = (8)
tupla1 = (8,)
tupla2 = (True, 9.7, "hola")
print(type(notupla))
print(type(tupla1))
print(type(tupla2))
print(tupla2[2])
```

```
C:\Users\Victor\Desktop>py p.py
<class 'int'>
<class 'tuple'>
<class 'tuple'>
hola
C:\Users\Victor\Desktop>
```

Listas: Grupo de elementos entre "[]" separados por ",". Como un ArrayList. Es el tipo de conjunto de elementos más común. Se accede a los valores a través del índice, y pueden añadir, eliminar o modificar los elementos. Más adelante explico las listas en profundidad.

```
lista1 = []
lista2 = [True, lista1, 8, "Hola"]
print(type(lista1))
print(lista2)
print(lista2[2])
```

```
C:\Users\Victor\Desktop>py p.py
<class 'list'>
[True, [], 8, 'Hola']
8
```

Sets: También llamados conjuntos, lista de elementos entre "{}" separados por ",". Los elementos no se repiten y no están ordenados, no se puede acceder a ellos directamente. Como un HashSet de Java. Se usa "in" para saber si el set contiene o no un elemento. Para añadir **set.add(elemento)**, si está repetido no lo añade. Eliminar, **set.remove(elemento)**, comprueba antes que lo contiene o saltará una excepción.

```
conj = {8, "hola"}
print(type(conj))
conj.add(6j)
print("hola" in conj)
print(conj)
conj.remove("hola")
print(conj)
```

```
C:\Users\Victor\Desktop>py p.py
<class 'set'>
True
{8, 6j, 'hola'}
{8, 6j}
C:\Users\Victor\Desktop>_
```

Diccionarios: Lista de clave-valor, entre "{}" separados por ":". Como un Map de Java, o estructura de un JSON. La clave (dato básico o tupla), ":" y valor (cualquier tipo). Se añaden o modifican valores de forma directa, y se eliminan con **dic.pop(clave)**, comprobar que existe con "in" primero.

```
dicc = {8:True, "hola":[6,"a"]}
print(type(dicc))
dicc[(3,)] = 7j
dicc[8] = False
print("hola" in dicc)
dicc.pop("hola")
print(dicc)
```

```
C:\WINDOWS\system32\cmd.exe
C:\Users\Victor\Desktop>py p.py
<class 'dict'>
True
{8: False, (3,): 7j}
C:\Users\Victor\Desktop>_
```



OPERADORES LOGICOS

Los tres básicos se escriben literales **"and"**, **"or"** y **"not"**.

De igualdad **"=="**, de diferencia **"!="**, mayor **">"**, mayor o igual **">="**, menor **"<"**, menor o igual **"<="**, es decir, igual que en Java y JavaScript.

Ejemplo: **"not 8 != 9"** daría False, **"8 != 9"** es True y **"not"** lo hace False.

OPERADORES MATEMÁTICOS

Suma **"+"**, resta **"-"**, multiplicación **"*"**, división **"/"**, cociente **"//"**, módulo o resto **"%"**, exponente o elevar **"**"**. Como Java tiene el **"+="** y demás.

Ejemplo: **"a=2"**, **"b=7"**, **"b **= a"** equivale a **"b = b**a"** es decir 49.

Nota: **"+"** también se usa para concatenar cadenas de texto.

CASTINGS

Como pongo en el apartado variables, Python es de tipado dinámico, los valores no cambian por si solos de tipo, pero las variables pueden contener cualquier tipo de valor, lo que nos permite volcar en la misma variable su valor cambiado de tipo. Son: **bool()**, **int()**, **float()**, **complex()**, **str()**, **list()**, **set()**, **map()**, **tuple()**, poniendo dentro del paréntesis el elemento a convertir. Ejemplos: **"nuevoInt = int(viejoFloat)"** o **"var = str(var)"**.

Existen más castings para pasar a hexadecimal **hex()**, octal **oct()**, binario **bin()**, o pasar entre entero y ASCII, **chr()** y **ord()**.

Nota: Las operaciones matemáticas entre dos tipos, realizan el casting de forma automática al tipo de dato más complejo.

COMENTARIOS

De una línea con **"#"**, multilinea poniendo **"''' "** o **"`"'"`"** (tres comillas simples o dobles) al principio y al final.

BLOQUES DE CÓDIGO

Python no usa **"{ }"** para rodear los bloques de código de condicionales, bucles, etc., se usa **":"** más la **indentación** o tabulación, o dos espacios en blanco, y para salir del bloque se vuelve a la indentación anterior.

CONDICIONALES: IF, ELIF, ELSE

Los condicionales son igual que en Java o JavaScript salvo en dos cosas, la condición **no** va entre **"()"** y que es **"elif"** como bash, no **"else if"**.

Nota: Python carece de Switch, en internet hay alternativas, ej. Varios elifs.

```
if not True:
    print("a")
elif True and False:
    print("b")
else:
    print("c")
print("d")
```

C:\Users\Victor\Desktop>py p.py
c
d
C:\Users\Victor\Desktop>



BUCLE WHILE

Como cualquier "while" pero igual que "if" con la condición sin paréntesis.

```
condicion = True
while condicion:
    print(condicion)
    condicion = False
print(condicion)
```

C:\Users\Victor\Desktop>py p.py
True
False
C:\Users\Victor\Desktop>_

BUCLE FOR IN

Se usa para avanzar por objetos iterables como los tipos de datos avanzados u objetos especiales como **range()**; range(x) crea una "lista" iterable de tipo "range" con valores de 0 a x-1. Es decir, hará x ciclos.

```
a = range(2)
for i in a:
    print(i)
print(type(a))
print(i)
```

C:\Users\Victor\Desktop>py p.py
0
1
<class 'range'>
1

Nota: El "for in" crea la variable "i", una vez terminado "i" sigue existiendo.

AMBITO DE LAS VARIABLES

Las variables creadas fuera de funciones y objetos tienen ámbito global, aún creadas dentro de condicionales o bucles, si no, tienen ámbito local. Las locales prevalecen en caso que dos variables se llamen igual. Para modificar una global en entorno local, hay que asegurar la variable global por medio de la palabra reservada **"global"**.

Otro método es mediante las funciones **globals()** y **locals()**, que contienen un Map con todas las variables de su ámbito, asignando nuevo valor:

Ejemplo: en un entorno local "globals()["nombreVariable"] = valorNuevo".

FUNCIONES

Las funciones se definen mediante **"def"** seguido del nombre y argumentos separados por ",", entre **"()"**. Puedes usar **"return"** para retornar un elemento. Si se vuelca en una variable sin return, esta variable será "None".

Nota: **"pass"** se usa para decir que no sucede nada.

```
def funci(a, b):
    if a==b:
        return 0
    else:
        pass
x = funci(3,3)
print(x)
y = funci(3,"3")
print(y)
```

C:\WINDOWS\system32\cmd.exe
C:\Users\Victor\Desktop>py p.py
0
None
C:\Users\Victor\Desktop>_



OBJETOS

Primero definimos la clase, mediante la palabra "**class**" y el nombre. La palabra "**self**" se usa para referenciar a las variables de su propia instancia (sería el equivalente a **no** poner "static" en Java) en lugar de a variables de uso común para todas las instancias.

El constructor de una clase se hace mediante "**def __init__()**" y los argumentos separados por "," dentro del paréntesis; luego se asignan los valores de la instancia o la clase con "self.atributo" o "nombClase.atributo". El primer argumento de las funciones y constructores debe ser "**self**" si va a usar los argumentos en la instancia.

ENCAPSULACION

Para hacer los atributos privados se añaden dos "_" antes del nombre.

HERENCIA

Se hace metiendo el nombre de las clases padre en el paréntesis de después del nombre (opcional si no hay herencia), y pasando los parámetros al constructor del padre con "casePadre.init__(params)".

POLIMORFISMO

En Python, para que un sobre escribir un método heredado, solo hay que volver a definirlo.

```
class Humano(object):
    def __init__(self, a,b):
        self.__A = a
        self.B = b
    def getA(self):
        return self.__A
class Gamer(object):
    def __init__(self, c):
        self.C = c
    def saluda(self, h):
        print("Hola "+h)
class Friki(Humano, Gamer):
    def __init__(self, a, b, c):
        Humano.__init__(self, a, b)
        Gamer.__init__(self, c)
    def saluda(self, h):
        print("Hi "+h)
luis = Friki(1,2,3)
print(f"a:{luis.getA()}, b:{luis.B} y c:{luis.C}")
luis.saluda("Juan")
```

C:\WINDOWS\system32\cmd.exe

```
C:\Users\Victor\Desktop>py p.py
a:1, b:2 y c:3
Hi Juan
```

```
C:\Users\Victor\Desktop>
```

MÓDULOS

Los módulos son como las librerías de Java, se importan al principio del programa con: "**import** modulo1, modulo2", si solo queremos alguna de las funcionalidades "**from** modulo **import** funcionalidad", y si queremos ponerle un alias "**import** modulo **as** mod".

```
import math as m
print(f"pi {m.pi} y e {m.e}")
```

C:\Users\Victor\Desktop>py p.py
pi 3.141592653589793 y e 2.718281828459045



PYTHON AVANZADO

LISTAS, INDICES Y LIST COMPREHENSIONS

Como puse en listas, he preferido dejar un apartado más completo solo para ellas donde poner los métodos más comunes, índices y una utilidad.

lista.append(elem), añade el elemento al final de la lista.

lista.insert(índice, elem), añade el elemento en el índice seleccionado.

lista.pop(), remueve y devuelve el último elemento.

lista.pop(índice), remueve y devuelve el elemento que ocupe ese índice.

lista.count(elem), número de veces que aparece ese elemento en la lista.

lista.sort(), ordena los elementos de la lista.

len(lista) = número de elementos (también en tuplas, sets y diccionarios).

max(lista) = el elemento mayor de la lista.

min(lista) = el elemento menor de la lista.

sum(lista) = suma los elementos de la lista.

lista1 + lista2 = una lista con los elementos de las listas (lista1 += lista2).

En Python los índices se pueden mirar de principio a fin de forma positiva, desde el 0 a (longitud-1) como Java, o de final a principio de forma negativa, desde el -1 a -longitud. En una lista de 4 elementos, la primera posición es tanto [0] como [-4] y la última [3] y [-1].

También tiene una forma de que a través de los índices, se devuelva varios valores "[inicio:fin(no incluido):paso]", los ":" del paso, si no vas a usarlo puedes omitirlos; inicio y fin pueden ser positivos, negativos, 0 o sin valor, el paso puede ser positivo si posición inicio < p. fin, o negativo si no.

```
a = [1,2,3,4,5,6,7,8,9,10]
print(a[:5])#no incluye el elemento fin
print(a[-3:0])# fin<inicio, paso negativo
print(a[::-2])#no inicio, paso -, coge in=-1
print(a[:9:3])
```

C:\Users\Victor\Desktop>py p.py
[1, 2, 3, 4, 5]
[]
[10, 8, 6, 4, 2]
[1, 4, 7]

Las list comprehensions es un formato por el cual crear una lista a partir de otra, realizando modificaciones y filtros sobre los elementos de otra lista.

Se crea entre "[" "]" aplicando o no una **expresión** sobre un **for in** de la lista y pudiendo aplicar un **condicional** al final. La expresión puede ser cualquier cosa que modifique un elemento, y la condición cualquiera (es un if).

"listaNueva = [x for x in listaVieja]" esta forma simple, solo copia tal cual.

"nueva = [n**2 for n in vieja if esPrimo(n)]" esto se traduciría de la así:

Lista nueva es igual al cuadrado de los elem. de la vieja que sean primos.

```
def esPar(x):
    return not bool(x%2)
vieja = range(20)
print([x//3 for x in vieja if esPar(x)])
```

C:\Users\Victor\Desktop>py p.py
[0, 0, 1, 2, 2, 3, 4, 4, 5, 6]

C:\Users\Victor\Desktop>

El cociente de 3, de los números pares un la lista de números de 0 a 19.



FUNCIONES LAMBDA

Es una manera acortada de definir funciones sencillas, similar a las funciones anónimas de JavaScript o las funciones lambda de Java.

"nombre = lambda par1, par2, ... : expresión" y se llama "nombre(params)"

```
esPar = lambda x : not bool(x%2) C:\Users\Victor\Desktop>py p.py
print (esPar(5)) False
```

TRY, EXCEPT, FINALLY Y RAISE

Igual que en Java, a excepción de que se dice "**except**" en lugar de catch, y que no se puede capturar la excepción, aunque si filtrar por tipo.

Se pueden lanzar excepciones con "**raise**", poniendo el tipo y el mensaje que se verá si no es capturada: "raise tipoDeExcepción(mensaje)".

```
try:
    raise TypeError("Falló")
    #raise Excepcion("Petó")
except TypeError:
    print("Excepcion tipo")
except:
    print("Otro tipo")
finally:
    print("Al final")

try:
    #raise TypeError("Falló")
    raise Excepcion("Petó")
except TypeError:
    print("Excepcion tipo")
except:
    print("Otro tipo")
finally:
    print("Al final")

try:
    raise Exception("Petó")
except TypeError:
    print("Excepcion tipo")
finally:
    print("Al final")
```

C:\Users\Victor\Desktop>py p.py
Excepcion tipo
Al final
C:\Users\Victor\Desktop>
Otro tipo
Al final
C:\Users\Victor\Desktop>
C:\Users\Victor\Desktop>py p.py
Al final
Traceback (most recent call last):
File "p.py", line 2, in <module>
 raise Exception("Petó")
Exception: Petó

DEL

Del es una palabra reservada que se usa para borrar de memoria objetos.

"a = 5" => "del a" => "print(a)" => "NameError: name 'a' is not defined".

MÓDULOS AVANZADO

Todos los archivos Python pueden importar sus funciones, si son "publicas", incluso los tuyos. Si creas un archivo.py con una función "def hola()", y en otro archivo pones "import archivo", podrás usar "archivo.hola()".

PILA Y COLAS

Los métodos append() y pop() de las listas (append() añade al final, y pop() saca el último elemento), hacen fácil crear una clase con una fila a la que se le aplican estos métodos, o bien insert() y pop() con índice 0 (insert(0) añade al principio y pop(0) saca el primer elemento), cumpliendo el "Last in, first out". Las colas, se harían con append() y pop() con índice 0, o bien insert() con índice 0 y pop(), cumpliendo "First in, first out".

SLEEP

La librería "time" tiene el método **sleep()**: "time.sleep(segundos)".



FICHEROS

La lectura y escritura es muy fácil con la función "**open(ruta, modo)**", los modos: r= lectura (por defecto), a = escribe/no machaca, w= si machaca, x= crea el fichero, b= abre en modo binario, t=abre en modo texto (por d.), y r+, w+, x+ y a+, igual que sus letras pero lectura y escritura.

Open devuelve un objeto iterable sobre el que se puede usar read() para sacar todo en una sola cadena, readline() para que devuelva línea a línea, o readlines() para que devuelva una lista de las cadenas. Usar **close()** para cerrar el archivo al final. El método write() escribe en el archivo.

```
arc = open("pruebo.txt", "x+") #creo el archivo
arc.write("Hola compañer@s") #escribo en él
arc.close() #lo cierro
arc2 = open("prueba.txt", "r") #creo abriendolo
txt = arc2.read() #lo leo y cierro
arc2.close()
print(txt)
```

C:\Users\Victor\Desktop>py p.py
Hola compañer@s

C:\Users\Victor\Desktop>py p.py
Traceback (most recent call last):
File "p.py", line 1, in <module>
arc = open("pruebo.txt", "x+") #creo el archivo
FileExistsError: [Errno 17] File exists: 'pruebo.txt'

La segunda ejecución da una excepción porque "x+" solo lo crea si no existe, creo que en las últimas versiones, si usas los demás modos sobre rutas que no existen, crean el archivo.

HILOS

Importamos la librería "threading" y creamos una variable que apunta a dicho hilo: "variable = threading.Thread(target=NombreFuncAEjecutar)". Para ejecutar el hilo, al igual que en Java, es con **start()**: "variable.start()". Para que el programa espere a que termine el hilo **join()**: "variable.join()", se puede utilizar junto a "threading.active_count()" (devuelve el número de hilos activos) para que no se crees más hilos mientras este esté activo, lo que nos permite idear formas de sincronización.

Podemos obtener el nombre "threading.current_thread().getName()" o el identificador "threading.current_thread().ident" del hilo que los ejecute.

El Global Interpreter Lock (GIL) es el mecanismo que controla que no haya una ejecución simultánea de los hilos, los cuales van intercambiando el control del GIL, lo que hace que los resultados se entremezclen.

En la creación del hilo se le puede dar un nombre y pasar parámetros:

"threading.Thread(name=nombre, target=func, args=tupla, kwargs=map)".

```
1 import threading as t
2 import time as ti
3 lista = []
4 def soy(a):
5     print(f"Soy el hilo {a} id: {t.current_thread().ident}")
6     ti.sleep(1)
7     lista.pop(0)
8
9 for i in range(7):
10     if t.active_count() < 3:
11         lista.append(t.Thread(target=soy, args=(i,)))
12         lista[len(lista)-1].start()
13     else:
14         lista[0].join()
```

C:\WINDOWS\system32\cmd.exe

C:\Users\Victor\Desktop>py p.py
Soy el hilo 0 id: 179568
Soy el hilo 1 id: 179572
Soy el hilo 3 id: 179580
Soy el hilo 4 id: 179584
Soy el hilo 6 id: 179596

C:\Users\Victor\Desktop>

Nota: debido a los sleep() y join() solo puede haber dos hilos activos a la vez y el main, por ello, hay ciclos del for que no crean hilos. (Gracias Lola).



PROCESOS

Importamos las librerías "multiprocessing" y "os", creamos una variable: "proceso = multiprocessing.Process(target=nombreFuncionAEjecutar())". Las funciones start() y join() igual que hilos. Obtener id: "os.getpid()" y nombre: "multiprocessing.current_process().name". El paso de parámetros y parámetros variables igual que en los hilos con args y kwargs. Diferencias entre procesos e hilos, un proceso es un programa en ejecución y los hilos son instrucciones dentro uno. Los procesos son independientes, consumen más recursos, no comparten memoria y no requieren ser sincronizados.

ACCESO A DATOS

Lo explicaré con SQLite, pero hay múltiples librerías según el SGBD. Importamos la librería "sqlite3" y creamos la conexión en una variable: "conex = sqlite3.connect('ruta y nombre de la base de datos')". Iniciamos un cursor con: "cursor = conex.cursor()" y ejecutamos las sentencias con: "cursor.execute('sentencia sql')". Recuerda realizar "conex.commit()" y "conex.close()" al realizar modificaciones, como con cualquier base de datos. Tras un select, el cursor recogerá un iterable.

SOCKETS

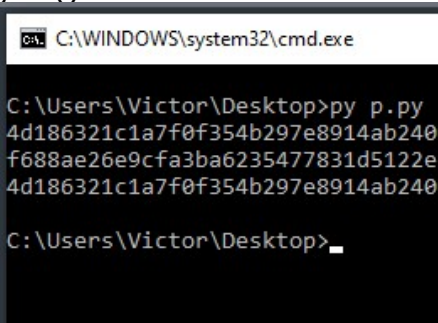
Importamos la librería "socket" la cual tiene los métodos:

socket() =devuelve una instancia; **gethostname()** =devuelve el nombre de la máquina; **bind((nombre, puerto))** =configura la instancia con el nombre de la máquina y el puerto; **listen()** =abre el puerto y lo pone en escucha; **connect((ip,puerto))** =método del cliente que permite conectarse a un socket servidor abierto; **accept()** =método que permite crear la conexión solicitada, y devuelve un objeto "conexión", y la dirección IP del cliente, close =cierra la instancia; **send(b'datos')** =envía a través del obj. "conexión"; **recv(bytesALeer)** =recibe a través del obj. conex. Los datos enviados por "send" primero se pasan a bytes, por eso la b. Los datos de "recv" están en bytes, castearlos con decode("utf-8").

CRIPTOGRAFÍA - HASH

La librería "hashlib" provee diferentes algoritmos criptográficos de hash. Creamos una variable: "cript = hashlib.tipoDeAlgoritmo()". Para ver los disponibles "hashlib.algorithms_guaranteed", "hashlib.algorithms_available" y luego "cript.update(texto.encode('utf-8'))", para hacer visible el hash "cript.digest()" en binario, o "cript.hexdigest()" en exadecimal.

```
import hashlib as has
cr = has.md5()
ct = has.md5()
cy = has.md5()
cr.update("hola".encode('utf-8'))
print(cr.hexdigest())
ct.update("Hola".encode('utf-8'))
print(ct.hexdigest())
cy.update("hola".encode('utf-8'))
print(cy.hexdigest())
```



```
C:\WINDOWS\system32\cmd.exe
C:\Users\Victor\Desktop>py p.py
4d186321c1a7f0f354b297e8914ab240
f688ae26e9cfa3ba6235477831d5122e
4d186321c1a7f0f354b297e8914ab240
C:\Users\Victor\Desktop>_
```



ALGORITMIA AVANZADA

Los patrones de diseño es dar un nombre a los aspectos clave del diseño de una estructura, creando un componente reutilizable, es decir, guiones de diseño para generar una estructura, o patrón, replicable en otros lenguajes.

PATRÓN SINGLETON

Se fundamenta en el hecho de crear una clase de la cual solo se puede crear una instancia, y que cada vez que se llama a esa clase, te devuelve siempre el mismo objeto (o lo crea y se le asocia a si misma si no existe), de tal manera que el objeto es compartido por todos los elementos que se estén ejecutando en ese momento.

Esto se consigue haciendo que la clase tenga un atributo que sea una instancia de si misma, capando el constructor (haciéndolo privado), y creando una función pública estática que, si la clase no tiene ninguna instancia asociada, crea una, la asocia y la devuelve, y si sí tiene, devuelve dicha instancia. Lo pongo en Java y Python, aunque un poco diferente.

```
public class Singl{
    private static Single singi;
    private Singl() { }
    public static Single damelo() {
        if (null == singi) {
            singi = new Single();
        }
        return singi;
    }
}
```

Se ve que el registro de memoria, en el "dame()" es la misma instancia.

```
class Single:
    class __impl:
        def dame(self):
            return id(self)
    __instance = __impl()
    def __getattr__(self, attr):
        return getattr(self.__instance, attr)
    def __setattr__(self, attr, value):
        return setattr(self.__instance, attr, value)

s1 = Single()
print(id(s1), s1.dame())
s2 = Single()
print(id(s2), s2.dame())
```

C:\WINDOWS\system32\cmd.exe
C:\Users\Victor\Desktop>py p.p
28729720 28729696
28729864 28729696

ALGORITMOS GENÉTICOS (AGs)

Se engloban en una subrama de la Inteligencia Artificial, la programación evolutiva, donde una serie de "individuos" evolucionan a través del tiempo. Se fundamenta en tres bases: la aleatoriedad (mutación), la recombinación (mezcla de los padres) y la optimización (o selección).

Se genera una población aleatoria, se seleccionan la mitad mejor (según el criterio que se le quiera dar), estos recombinan entre ellos y o se mutan (bien en ellos mismos o creando nuevos individuos), después se vuelve a realizar el proceso de selección sobre esta nueva población.

El criterio de selección puede ser que se acerque a un punto ideal, sin superarlo, por ejemplo:

El famoso caso de la mochila, donde tenemos que seleccionar que objetos meter en ella, de tal manera que ocupemos el máximo volumen y número de objetos sin superar el peso, en este caso el criterio sería que la suma de los objetos de una lista no supere el volumen y peso, pero sea el mayor posible.

Proceso:

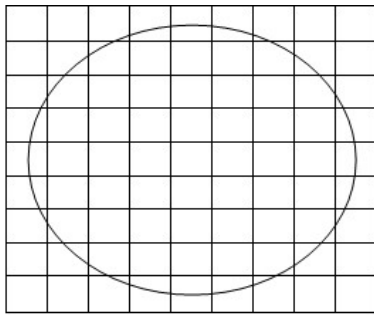


Se generaría un número alto y determinado de listas con elementos aleatorios que no superen el peso, ni el volumen.
Se ordenarían por número de unidades, y se elegiría la mitad mayor.
Sobre esta mitad se realizarían modificaciones aleatorias sobre los elementos de tal manera que solo sean efectivos si ha habido mejora.
O se realizaría una recombinación entre varios elementos para generar nuevas posibles listas. O los dos métodos combinación y modificación.
Repetiríamos estos pasos hasta que queramos, en nuestro caso hasta que solo quede una solución, la mejor.

REDES NEURONALES

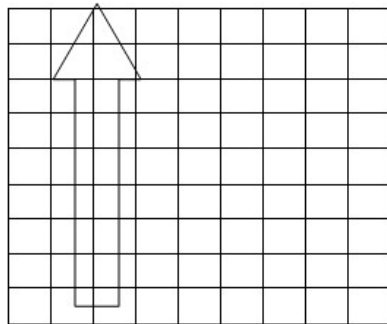
Es uno de los campos de la Inteligencia Artificial, específicamente la subrama de Aprendizaje Supervisado. Está basado en el procesamiento de varias capas para procesar datos, la primera, la capa de entrada, reciben datos del exterior, la segunda, la capa oculta, son neuronas sin conexión con el entorno, que proporciona grados de libertad, y la capa de salida, neuronas que proporcionan la respuesta. Cada neurona realiza sencillos procesos de información, y pueden recibir y enviar información a otras según los criterios establecidos, que se calibran con casos de entrada, con lo que el sistema aprende, para luego poder dar o predecir un resultado. El caso de red neuronal más sencillo es el Perceptron Simple. Está basado en el funcionamiento de la retina, si tomamos las neuronas de la retina como una matriz, cuya salida está conectada complejamente a otras neuronas, en las que existen grupos especializados en reconocer patrones como líneas, verticales, horizontales, ..., y estas a su vez a otro grupo que determina si la señal supera un umbral, para generar su salida. Imagina una cuadrícula de un sudoku 9x9, a la que si pasa una línea por mas del 50%, se activa, luego por cada agrupación de 3x3 contiguas vemos si mas del 40% se ha activado, de tal manera que nos queda un 3X3 de activación o no, que se lo pasamos a otro grupo que sabe que si están activados los extremos, seguramente sea un círculo, si están solo contiguas de una fila, seguramente una línea horizontal, las contiguas de una columna, una vertical, etc. Ahora bien, si dependiendo del nivel de superficie afectada, avisan a unas neuronas u otras, y lo realizamos por todas las posibles agrupaciones, el afinamiento es mayor, la capa de entrada son los primeros cuadros de 9x9, y la salida puede ser un círculo, la capa de salida, con lo que las capas internas con los n grados de libertad serían los múltiples criterios a aplicar.
Ejemplo visual: Se pone 1 si la línea atraviesa suficiente superficie, luego sumo los 1 por cuadrante y le asigno un valor de fuerza, en el primer caso queda una matriz cuyas esquinas son fuertes, sus lados un poco menos y el centro está vacío, es decir, un círculo; en el segundo, todos los cuadrantes de una columna son fuertes y los demás vacíos, se deduce, una línea recta vertical.





0	0	0	1	1	1	0	0	0
0	1	1	0	0	0	1	1	0
0	1	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	1	0
0	1	1	0	0	0	1	1	0
0	0	0	1	1	1	0	0	0

4-2	3-1	4-2
3-1	0-0	3-1
4-2	3-1	4-2



0	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0

6-3	0-0	0-0
6-3	0-0	0-0
6-3	0-0	0-0



FRAMEWORKS, IDES Y BIBLIOTECAS

FRAMEWORKS

Django: Es un web framework escrito en Python, pensado para un desarrollo ágil, de mucha relevancia en la actualidad pues permite realizar toda la parte de servidor Web en Python, tiene muy buena conexión con la gran mayoría de Sistemas Gestores de Bases de Datos, es limpio, estable, escalable, muy seguro y con muchas posibilidades. Al encontrarse en continuo crecimiento, sirve para todo tipo de proyectos Web, con una gran comunidad detrás, lo que es un gran apoyo. “www.djangoproject.com”.

Pyramid: También es un web framework basado en los principios de sencillez, minimalismo, documentación, velocidad, fiabilidad y franqueza. Entre sus ventajas frente a otros es que es moderno, muy estable y bien documentado, además que permite empezar a crear con muy pocos conocimientos, pero no se queda corto si eres un experto.

Flask: Es un mini-framework formado por varios módulos que ayuda en la construcción de páginas web dinámicas.

IDES

PyCharm: IDE muy completo, compatible con Django, muy parecido a Android Studio, ya que es de la misma compañía, JetBrains. Con licencia de pago para empresas, pero gratuita para estudiantes.

PyDev: Plugin para Eclipse totalmente gratuito, compatible con Django muy completo. Tiene alguna inestabilidad al combinarlo con algún plugin.

Spyder: IDE muy completo, más pensado para la rama de ciencias de datos de Python, que para la rama Web. De código abierto y gratuito. Gracias a su buena integración con la consola ipython, se puede interactuar y modificar las variables en tiempo de ejecución. “spyder-ide.org”.

Sublime text: Es un editor de texto, no un IDE, que se va adaptando al programa según lo vas escribiendo, y tiene un montón de posibilidades personalizables; es el que he usado para los ejemplos.

IDLE: Es un IDE gratuito y de código abierto que se pre instala al instalar Python, aunque en un primer momento parece más un editor de texto que un IDE, tiene un montón de posibilidades, la mejor, su potente depurador.

BIBLIOTECAS

Son librerías masivas en muchos casos, que contienen cientos de funciones, generalmente utilizadas en ciencias de datos e inteligencia artificial.

NumPy: Como amante de las arrays multidimensionales, esta es mi librería favorita. Crea un gran soporte para trabajar y generar vectores y arrays de múltiples dimensiones, y provee de una grandísima colección de funciones y algoritmos de alto nivel para trabajar con ellas. Es fundamental para la rama de inteligencia artificial, para el control de las múltiples posibilidades de las redes neuronales. Precedido por Numeric, fue creado en 2005 por Travis Oliphant. Es de código abierto y tiene múltiples colaboradores.



Pygame: Librería especializada en la creación de videojuegos 2D. Es un conjunto de módulos que permiten crear prototipos y desarrollar de forma rápida y fácil. Orientado al manejo de sprites o mapas cúbicos de bits.

Pyglet: Un motor de animación y creación de juegos en 3D. Este es el motor con el que se desarrolló Minecraft. Proporciona una interfaz programable orientada a objetos, también permite desarrollar otros tipos de aplicaciones multimedia. Uno de sus fuertes es que es multiplataforma.

SciPi: Es una biblioteca de algoritmos y herramientas matemáticas para usos científicos que está consiguiendo que Python eclipse a Ruby.

Matplotlib: Una biblioteca de trazado numérico muy útil para los científicos y analizadores de datos.

Librerías para hacking: Python se está convirtiendo en uno de los lenguajes más usados en este campo, pero al ser tan cambiante, las librerías a veces se quedan rápidamente obsoletas, si estás interesado en ese campo, te recomiendo más visitar el perfil de "jotta_app" en instagram.

