CURSO UDEMY: Master en Python 3.10

Aprende de 0 a Experto con Práctica.

1. SECCIÓN 1: BIENVENIDA AL CURSO

Python es un lenguaje interpretado, es decir, necesita un tercer programa para su funcionamiento, dicho programa es, p.ej., ***Visual Studio Code***.

Recordemos que un lenguaje compilado necesitará de una función ***main*** para compilarse.

Cabe señalar que, en Python, no es necesario utilizar el ***punto y coma*** (;) al finalizar una sentencia.

2. SECCIÓN 2: PREPARAR EL ENTORNO DE TRABAJO

2.1 Instalar Python 3.10:

2.1 Instalar Visual Studio Code:

2.3 Extensiones para Visual Studio Code:

Se instalararon las siguientes extensiones para VSC:

- *Python;*
- *Terminal;*
- *Bracket Pair Color*

2.4 Ejecución a Archivos:

3. SECCIÓN 3: PRIMEROS PASOS

3.1 Hola Mundo:

Recordemos que, dado que Python es un lenguaje interpretado, no necesita tener una función ***main***, por lo tanto, la cantidad de líneas a emplear en comparación con un lenguaje compilado, **es menor**.

/***/

Ejemplo:

print (“Hola Mundo”)

NOTA: Nótese que, a diferencia, p.ej. de C, Python **NO** necesita una función ***main*** para funcionar **NI** tampoco un ***punto y coma***. La notación es tal cual se observa en el ejemplo.

3.2 Variables y Tipos de Datos (Teoría):

Conceptos Importantes:

Datos:

Un dato es una de las cosas más básicas que utiliza un programa como una letra o un número. Por ejemplo: 5 o 49.99 que son datos numéricos.

Tipos de datos:

• Enteros

• Flotante

• Cadena de texto

Variable:

Una variable no es exclusiva de un lenguaje de programación. Una de las características más potentes de un lenguaje de programación es la capacidad de manipular variables. Una ***variable*** **es un nombre** que se refiere a un ***dato.***

Reglas de las variables:

• No pueden iniciar con un número.

• No pueden llevar caracteres especiales.

• No pueden llevar el nombre de palabras reservadas del lenguaje.

Ejemplos de variables y tipos de datos:

• Se desea almacenar, en una variable, la edad de un usuario que tiene 20 años

edad = 20

• Se quiere almacenar, en una variable, un mensaje de bienvenida

mensaje = “Bienvenido”

3.3 Variables y Tipos de Datos (Práctica):

```
nombre = “Coto Walter”
print(type(nombre))
print(nombre)

numero = 89
print(type(numero))
print(numero)

flotante = 3.14
print(type(flotante))
print(flotante)
```

NOTA: En Python **NO** se utiliza la palabra reservada ***var*** para definir una variable.

• **print**(**type**(nombre)) – Imprime el tipo de dato de la variable “nombre”.

3.4 Reglas y Consejos para las Variables:

Se realizar la prueba de las reglas mencionadas anteriormente para la declaración de una variable:

• No pueden iniciar con un número.

• No pueden llevar caracteres especiales.

• No pueden llevar el nombre de palabras reservadas del lenguaje.

Además, tampoco se permite utilizar espacio.

```
/*********************************************************************************************/
import.keyword
print(keyword.kwlist)
```

NOTA: Permite imprimir en pantalla la lista de palabras reservadas del lenguaje.

3.5 Variables y Constantes:

Python no tiene para declarar ***constantes*** como tal. Para este caso, existen artilugios para declarar “constantes”; se declara una variable en mayúscula para distinguir que dicho valor no debe ser modificado. Esta es una práctica usual entre programadores en Python.

```
/*********************************************************************************************/
cantidadPersonas = 100
print(cantidadPersonas)

cantidadPersonas = 150
print(cantidadPersonas)

PI = 3.14
```

print(PI)

NOTA: Nótese que el valor de la ***variable*** *cantidadPersonas* cambia su valor. Aunque el valor de la ***variable*** *PI* también puede hacerlo, si le otorgamos otro valor, la práctica de denominarla con mayúsculas, indica que ***NO*** debemos cambiar su valor dado que es una constante.

3.5 Comentarios y Asignación Múltiple de Variables

Comentario: es un mensaje en el código que sirve para guiarnos. Los comentarios para una sóla línea se designan a través del ***numeral*** #, para múltiples líneas, es necesario colocar ***tres comillas simples*** al inicio y fin del comentario ***‘’’ ‘’’***.

```
/*************************************************************************/
Comentarios:
nombre = “Walter Coto”
print(nombre)
#Este es un comentario

‘’‘Este es un
comentario de múltiples
líneas’’’

Múltiples variables:
nombre, apellido = ‘Alvaro’, ‘Chirou’
print(nombre)
print(apellido)
```

4. SECCIÓN 4: OPERADORES ARITMÉTICOS

4.1 Números:

En Python sólo existen dos tipos de números: enteros y flotantes.

/**/

Enteros:

num1 = 40

print(num1)

print(**floa**t(num1))

NOTA: **print**(**floa**t(num1)) – Convierte un número entero a ***flotante***.

Flotante:

num2 = 99.99

print(num2)

print(**int**(num2))

NOTA: **print**(**int**(num2))) – Convierte un número flotante a ***entero***.

4.2 Operadores Aritméticos y jerarquía de Operaciones (Teoría):

Operadores Aritméticos:

Son símbolos especiales que representan cálculos, como la suma o la multiplicación. Los valores a los cuales se aplican esos operadores reciben el nombre de ***operandos***. Ej.: Suma (+), resta (-), división (/), multiplicación (*), módulo (%), exponete (**), división entera (//).

El módulo es un operador que sirve para tomar en cuenta la parte del resto o residuo de una operación. El módulo, evalúa el ***resto***.

La división entera sólo tiene en cuenta la parte entera de la división efectuada.

Jerarquía de Operaciones:

Cuando se presentan muchas operaciones aritméticas, la jerarquía determina el orden con el que deben realizarse esas operaciones.

El orden de una operación viene determinado de la siguiente manera:

1- Paréntesis:

2- Exponenciación:

3- Multiplicación y división:

4- Suma y Resta

4.3 Operadores Aritméticos:

• Suma (+):

print(2+2)

• Resta (-):

print(150-75)

• Multiplicación (*)

print(10*10)

• División (/)

print(75/9)

• Exponenciación (**)

print(2**3)

• División Entera (//)

print(75//9)

• Módulo (%)

print(100%75)

4.4 Operadores Aritméticos con Variables:

Ejemplo 1:

num1 = 10

num2 = 5

print(num1+num2)

print(num1-num2)

print(num1*num2)

print(num1/num2)

print(num1//num2)

print(num1**num2)

print(num1%num2)

Ejemplo 2:

Es más común expresar las operaciones de esta manera.

num1 = 10

num2 = 5

suma=(num1+num2)

print(suma)

resta=(num1-num2)

print(resta)

4.5 Jerarquía de Operaciones:

num1 = 100

num2 = 50

num3 = 25

num4 = 10

print((num1+num2)*(num3-num4)/(num1-num4))

NOTA: En el caso que el cociente de cero, el lenguaje emitirá "*ZeroDivisorError: división by zero*"

4.6 Ejercicio 1:

$$\left(\frac{3+2}{2.5}\right)^2 = 0.25$$

Resolución (por mí):

num1 = 3

num2 = 2

num3 = 5

num4= num1+num2

num5 = num2*num3

num6 = (num4/num5)**2

print(num6)

Resolución 1:

print(((3+2) / (2*5)))**2)

Resolución 2:

calculo = ((3+2) / (2*5)))**2

print(calculo)

Resolución 3:

calculo = **pow** (((3+2) / (2*5)), 2)

print(calculo)

4.7 Ejercicio 2:

Una juguetería tiene mucho éxito en dos de sus productos: payasos y muñecas. Suele hacer venta por correo y la empresa de logística les cobra por peso de cada paquete, así que deben calcular el peso de los payasos y muñecas que saldrán en cada paquete a demanda. Cada payaso pesa 112 gr y cada muñeca 75 gr. Un cliente frecuente pide la cantidad de 23 payasos y 54 muñecas, realiza un programa que muestre el peso total de toda la venta.

Pista: Suponiendo que un cliente pide 5 payasos y 3 muñecas, la juguetería debería hacer la multiplicación de la cantidad de payasos con su peso, al igual que las muñecas; al tener ambos totales de peso, se debe sumar.

NOTA: Atl + Z para ver la consigna entera y no en una sola línea.

Resolución:

costoPayaso = (112*24)

costoMuneca = (75*54)

print("El costo total de la compra es:", costoPayaso+costoMuneca)

5. SECCIÓN 5: CADENAS DE TEXTO

5.1 Cadenas de Texto (Teoría):

Una cadena es una secuencia de caracteres.

“	M	o	n	t	y		P	y	t	h	o	n	“
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	

Dentro de las cadenas, cada uno de los caracteres ocupa un espacio, p.ej.: M [0]

Debanado de cadenas:

Consiste en mostrar contenido de una cadena de una manera segmentada.

Ejemplo:

“	M	o	n	t	y		P	y	t	h	o	n	“
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	

[0:5] = “Monty”

[:5] = “Mon”

[3:] = “hon” #Lo realizar desde atrás hacia adelante.

Reglas de las cadenas:

- Es importante escribir las cadenas con comillas dobles o con comillas simples.
- No confundir un número en comillas, a un número sin comillas.
- “4” -> 4

5.2 Cadenas de Texto (Práctica):

cadena = “Esto es un ejemplo de cadena de texto”

print(cadena)

NOTA:

****”ejemplo****” : Reconoce la palabra encerrada entre “”.

\n: Salto de línea

\t: Tabulación

\b: Quita un espacio

\f: Símbolo femenino

\v: Símbolo masculino

\r: Elimina lo que precede al comando.

5.3 Cadenas de Texto Parte 2:

Concatenación de cadenas implica que se pueda unir dos o más cadenas para formar un texto.

/**/

Ejemplo:

cadena1 = “Hola Walter”

cadena2 = “ Coto”

numero = 1

print(cadena1+cadena2)

print(“Hola usuario:”, cadena2)

print (cadena2*5)

print(**type**(**str**(numero)))

NOTA:

print (cadena2*5). Repite 5 veces la cadena2

print(**type**(**str**(numero))). Convierte numero de tipo ‘int’ a ‘string’

5.4 Debanado de Cadenas:

También, al debanado de cadenas, se lo conoce como *sub-strings*.

/**/

Ejemplo:

cadena = “Este en un ejemplo de subting (debanado de cadenas)”

print(**len**(cadena))

print(cadena[2])

print(cadena[0:9])

print(cadena[:20])

NOTA:

• **print**(**len**(cadena)). Devuelve el tamaño de la cadena de texto comenzando desde la posición 0.

• **print**(cadena[2]). Devuelve el carácter de la posición 2 de la cadena.

• **print**(cadena[0:9]). Devuelve los caracteres comprendidos desde la posición 0 hasta la posición 9 de la cadena de texto.

• **print**(cadena[:20]). Devuelve los caracteres comprendidos desde la posición 0 hasta la posición 20 de la cadena de texto.

5.5 Métodos de Cadenas:

Un método va a ser una función nueva que nos permita crear nuevas utilidades. Cuando hablamos de métodos vamos hablar sobre palabras reservadas del lenguaje.

/**/

Ejemplo:

cadena = “estoy utilizando lOs métodos de Python”

print(cadena.**lower**())

print(cadena.**upper**())

print(cadena.**capitalize**())

print(cadena.**title**())

print(cadena.**swapcase**())

NOTAS:

• **print**(cadena.**lower**()). Convierte todas las mayúsculas en minúsculas.

• **print**(cadena.**upper**()). Convierte todas las minúsculas en mayúsculas.

• **print**(cadena.**capitalize**()). Convierte la primera letra de la cadena en mayúscula.

• **print**(cadena.**title**()). Convierte todas las iniciales de las palabras en mayúsculas.

• **print**(cadena.**swapcase**()). Convierte todas las mayúsculas en minúsculas y todas las minúsculas en mayúsculas.

5.6 Ejercicio 1:

Crear un programa, que tenga una variable con la cadena “Te quiero solo como amigo”, y muestre la siguiente información:

• Imprima los dos primeros caracteres.

• Imprima los tres últimos caracteres.

• Imprima dicha cadena cada dos caracteres. Ej.: Si la cadena fuera “recta” debería imprimir “rca”.

• Dicha cadena en sentido inverso. Ej.: Si la cadena fuera “hola mundo!” debe imprimir “!odnum aloh”

• Imprima la cadena en un sentido y en sentido inverso. Ej: Si la cadena es “Reflejo” imprime “reflejoojelfer”.

/**********/

Ejercicio 1:

cad = “Te quiero solo como amigo”

print(cad[:2])

print(cad[-3:])

print(cad[: :2])

print(cad[: :-1])

print(cad+cad[: :-1])

NOTAS:

• **print**(cad[: :2]). Lee toda la cadena y luego imprime cada dos posiciones.

print(cad[: :-1]). Lee toda la cadena y luego la imprime en forma inversa.

print(cad+cad[: :-1]). Concatenación de cadenas: Original y forma inversa.

5.7 Ejercicio 2:

Crear un programa que tenga una variable con la cadena “Separado” y un carácter de coma (,); e inserte el carácter entre cada letra de la cadena. Ej: “separar” y, debería devolver “s,e,p,a,r,a,r”

Pista: Debes utilizar un método de las cadenas llamado “Replace”, recuerda que la posición de los caracteres empieza en 0.

/**********/

Ejercicio:

cad = “Separado”

print(cad.**replace**(“ “, “,”))

NOTA:

El método ***replace*** tiene como argumento dos parámetros: el primero, el carácter a sustituir y, el segundo, el carácter que sustituirá al primero.

• **print**(cad.**replace**(“ “, “,”)). Reemplaza los espacios entre caracteres por la coma.

6. SECCIÓN 6: ENTRADA POR TECLADO Y SALIDA POR PANTALLA

6.1 Entrada por Teclado:

La entrada por teclado se realiza a través de la palabra reservada ***input ()***. Para variables tipo ***string*** no es necesario colocar el tipo de dato, sin embargo, para los datos numéricos sí.

Un tipo de dato numérico *entero* presenta el siguiente formato: ***int(input())***.

/**********/

Ejemplo:

nombre = **input**(“Ingrese su nombre:”)

edad = **int**(**input**(“Ingrese su edad:”)

print(nombre)

print(edad)

NOTA:

• nombre = **input**(“Ingrese su nombre:”). Permite ingresar una cadena de caracteres sin necesidad de colocar el tipo de dato.

• edad = **int**(**input**(“Ingrese su edad:”). Permite ingresar un tipo de dato ***int***; nótese que para otro tipo de dato numérico se deberá reemplazar ***int*** por, p.ej., la palabra reservada ***float***.

6.2 Ejercicio 1:

Realizar un programa que haga el proceso de fórmula general para la resolución de ecuaciones, sabiendo que la fórmula general es la que está en la imagen, el usuario debe ingresar los valores de “a”, “b” y “c”, y el programa debe hacer el proceso para que al final muestre el mensaje: “La solución es: <solución>”

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

/**********/

Ejercicio 1:

Ejercicio pensado por mí.

```
a = float(input("Ingrese el valor de la variable a:")
b = float(input("Ingrese el valor de la variable b:")
c = float(input("Ingrese el valor de la variable c:")

dividendo = (-b+(pow((pow(b,2))-4*a*c),1/2))
divisor = 2*a
x = dividendo/divisor

print("La solución es:", x)
```

Ejemplo:

```
#3x^2-5x+2 = 0,  x1 = 1,  x2 = 2/3

from math import sqrt

a = int(input("Ingrese el valor a:")
b = int(input("Ingrese el valor b:")
c = int(input("Ingrese el valor c:")
x1 = 0
x2 = 0

if ((b**2)-(4*a*c)) < 0:
        print ("No se puede realizar la operación dado que la raiz es negativa")
else:
        x1 = (-b + sqrt((b**2)-(4*a*c)))/(2*a)
        x2 = (-b - sqrt((b**2)-(4*a*c)))/(2*a)
print("La solucion es: \nx1=", x1, "\nx2=", x2)
```

6.3 Ejercicio 2:

Se desea tener un algoritmo que permita determinar y mostrar el promedio que ha obtenido un alumno en un determinado curso, conociendo las notas de: tres prácticas, el examen parcial y el examen final.

Considere:

PP = (P1 + P2 + P3)/ 3 PROM = (PP + 2*EP + 3*EF) / 6

Donde:

P1, P2, P3: Prácticas

PP: Promedio de práctica

PROM: Promedio

EP: Examen parcial

EF: Examen final

Ejemplo:

```
/*********************************************************************************************/
```

Ejercicio 2:

Ejercicio pensado por mí.

```
P1 = float(input("Ingrese la nota de la Practica 1:")
P2 = float(input("Ingrese la nota de la Practica 2:")
P3 = float(input("Ingrese la nota de la Practica 3:")

PP = (P1+P2+P3)/3

print("El Promedio de las Practicas es:", PP)

EP = float(input("Ingrese la nota del Examen Parcial:")
EF = float(input("Ingrese la nota del Examen Final:")
```

PROM = (PP+2*EP+3*EF) / 6

print(“El Promedio total resulta:”, PROM)

Ejemplo:

practica1 = **float**(**input**(“Ingrese el valor de la practica 1:”))

practica2 = **float**(**input**(“Ingrese el valor de la practica 2:”))

practica3 = **float**(**input**(“Ingrese el valor de la practica 3:”))

ExamenParcial = **float**(**input**(“Ingrese el valor del examen parcial:”))

ExamenFinal = **float**(**input**(“Ingrese el valor del examen final:”))

PromedioPractica = (practica1 + practica2 + practica3)/3

PromedioFinal = (PromedioPractica + 2*ExamenParcial + 3*ExamenFinal)/6

print(“El promedio final del estudiante es de: \n”, PromedioFinal, “\nY el promedio practica es de: \n”, PromedioPractica)

6.4 Mostrar Datos por Pantalla:

Un método para mostrar datos por pantalla es a través de ***format*** independientemente si fueron datos cargados en el programa o digitados por el usuario.

/**/

Ejemplo:

nombre = **input**(“Ingresa tu nombre: “)

edad = **int**(**input**(“Ingresa tu edad: “)

print(“Hola **{}** tienes **{}**”, **format**(nombre,edad))

NOTA: Nótese que el método ***format*** toma como argumentos las variables que deben ser reemplazadas en los corchetes en la cadena de caracteres que debe imprimirse por pantalla.

6.5 Ejercicio 3:

Escribir un programa que solicite al usuario una vocal en minúscula, y luego, una letra en mayúscula. El programa debe convertir la letra en minúscula y la vocal en mayúscula, y al final, deben ser concatenadas ambas.

```
/*********************************************************************************************/

Ejercicio 3:

Ejercicio pensado por mí.


vocal = input(“Ingrese la vocal en minúscula: “)

letra = input(“Ingrese la letra en mayúscula: “)


vocal = vocal.upper()

letra = letra.lowe()


print(“La vocal es {} y la letra es {}”.format(vocal,letra))
```

6.6 Ejercicio 4:

Hacer un programa que pida al usuario su nombre, edad y sexo y los muestre de la siguiente forma:

Te llamas: <nombre>

Tu edad es: <edad>

Eres: <sexo>

```
/*********************************************************************************************/

Ejercicio 4:
```

Ejercicio pensado por mi.

nombre = **input**(“Ingrese su nombre: “)

edad = **int**(**input**(“Ingrese su edad: “))

sexo = **input**(“Ingrese su sexo: “)

print(“Te llamas: **{}\n**Tu edad es: **{}\n**Eres: **{}**”.**forma**(nombre,edad,sexo))

7. SECCIÓN 7: BOOLEANOS

7.1 Booleano:

Un booleano es, prácticamente, un valor que representa verdadero o falso. Un booleano sólo puede tener dos valores: true o false; 0 o 1; etc.

/**/

Ejemplo:

verdadero = **True**

falso = **False**

print(**type**(verdadero))

print(**type**(falso))

NOTA:

• verdadero = **True**. Representa una variable cuya asignación es un ***booleano*** de verdad.

• falso = **False**. Representa una variable cuya asignación es un ***booleano*** de falso.

7.2 Operadores Relacionales y Lógicos (Teoría):

1- Operadores Relacionales:

Hacen referencia a condiciones entre datos, también pueden ser vistos como comparadores, es decir, nos va a permitir comparar.

Operadores:

Operador	Descripción	Ejemplo
==	Igual que …	5 == 59 False
!=	Diferente de …	5 != 3 True
<	Menor que …	6 < 20 True
>	Mayor que …	100 > 1 True
<=	Menor o igual que …	90 <= 90 True
>=	Mayor o igual que …	100 >= 101 False

2- Operadores Lógicos:

Hacen referencia a condiciones entre datos, también pueden ser vistos como comparaciones. Permiten enlazar varias comparaciones de los operadores relacionales.

Operador	Descripción	Ejemplo
AND	y	10 > 1 and 70 <= 70 True
OR	o	50 >= 80 or 6 == 6 True
NOT	no a …	3.14 > 3 True False

AND:

Operador	Forma de Trabajo	Ejemplo
AND	True and True	True
	True and False	False
	False and True	False
	False and False	False

OR:

Operador	Forma de Trabajo	Ejemplo
OR	True and True	True
	True and False	True
	False and True	True
	False and False	False

NOT:

Operador	Forma de Trabajo	Ejemplo
NOT	True and True	False
	True and False	True
	False and True	True
	False and False	True

7.3 Operadores Relacionales:

Ejemplo:

num1 = 1000

num2 = 500

cad1 = “La noche esta linda”

cad2 = “El día es mas largo que la noche”

print(num1 == num2)

print(num1 > num2)

print(num1 < num2)

print(num1 >= num2)

print(num1 <= num2)

print(cad1 < cad2)

7.4 Operadores Lógicos:

Permite vincular varias comparaciones. Además, también pueden vincularse con los condicionales.

/**/

Ejemplo:

• **print**(99 != 99 **and** 23 == 23) #**AND**

• **print**(88 <= 45 **or** 60 > 110) #**OR**

• **print**(**not** 100 == 100) #**NOT**

7.5 Métodos Booleanos:

Utilizaremos los métodos booleanos para comparar *strings*.

/**/

Ejemplo:

cadena = “Estoy mostrando los métodos booleanos para las strings”

cadena1 = “HOLA MUNDO”

cadena2 = “cadena totalmente en minúscula”

print(cadena.**starswith**(“E”))

print(cadena.**endswith**(“s”))

print(cadena2.**isupper**())

print(cadena3.**islower**())

NOTA:

• **print**(cadena.**starswith**(“E”)). Verifica si la cadena comienza con “E”.

• **print**(cadena.**endswith**(“s”)). Verifica si la cadena termina con “s”.

• **print**(cadena2.**isupper**()). Verifica si la cadena está TOTALMENTE en mayúscula.

• **print**(cadena3.**islower**()). Verifica si la cadena está TOTALMENTE en minúscula.

8. SECCIÓN 8: CONDICIONALES

Un condicional son situaciones en las que pueden darse varios resultados dentro de un programa. A partir de la condición impuesta, el programa debe seguir hacia el camino correcto de acuerdo con el resultado obtenido de esa condición.

En el lenguaje de programación Python existen los siguientes condicionales: ***IF***, ***ELSE***, ***ELIF***.

8.1 If. Else y Elif (Teoría):

• ***If:*** Es la forma de colocar un condicional en Python, significa "Si…", pero de una condición.

• ***Else:*** Todo "If" debe llevar su ***Else***, el cual significa "Si no…", en el caso que no se cumpla la condición del ***If***.

• ***Elif:*** Se utiliza cuando se combinan varias opciones que el programa debe elegir. Es decir, permite anidar diferentes situaciones.

8.2 If y Else:

```
edad = 19

if edad >= 18:
    print("Tu eres mayor de edad. Puedes tener DNI")
else:
    print("Tu NO eres mayor de edad. NO puedes tener DNI")
```

NOTA: Nótese la notación utilizada para formular la condición, si bien la estructura es muy parecida a la de C, **NO** es igual.

8.3 Elif:

Permite crear múltilples condiciones, es decir, crear varios casos en los que se deben respetar una u otra condición.

Es muy parecido al método ***switch*** en el caso de C.

/***/

Ejemplo:

letra = “u”

```
if letra.lower() == “a”:
        print(“Esta vocal es la A”)
elif letra.lower() == “e”:
        print(“Esta vocal es la E”)
elif letra.lower() == “i”:
        print(“Esta vocal es la I”)
elif letra.lower() == “o”:
        print(“Esta vocal es la O”)
else:
        print(“Esta vocal es la U”)
```

8.4 Condicionales Anidados:

Un condicional anidado es un condicional dentro de otro condicional. Sirve para validaciones lógicas.

/**/

Ejemplo:

nombre = “Juan”

edad = 18

```
if nombre == “Juan”:
        if edad >= 18:
                print(“Tu eres Juan y eres mayor de edad”)
        else:
                print(“Tu eres Juan pero no eres mayor de edad”)
else:
        print(“Tu no eres Juan”)
```

8.5 Ejercicio 1:

Crear un programa que pida al usuario una letra, y si es vocal, muestre el mensaje “Es vocal”. Sino, decirle al usuario que no es vocal.

/**/

Ejercicio 1:

Ejercicio pensado por mí.

```
letra = input(“Ingrese una letra: “)

if letra.lower() == “a”:
    print(“La letra ingresada es la vocal A”)
elif letra.lower() == “e”:
    print(“La letra ingresada es la vocal E”)
elif letra.lower() == “i”:
    print(“La letra ingresada es la vocal I”)
elif letra.lower() == “o”:
    print(“La letra ingresada es la vocal O”)
elif letra.lower() == “u”:
    print(“La letra ingresada es la vocal U”)

else:
    print(“La letra ingresada NO es una vocal”)
```

Ejemplo:

```
letra = input(“Ingrese una letra: ”)

if letra.lower() in “aeiou”:
    print(“La letra ingresada es una vocal”)
else:
```

print(“La letra ingresada NO es una vocal”)

8.6 Ejercicio 2:

Escribir un programa que, dado un numero entero, muestre su valor absoluto. Nota: para los valores positivos el valor absoluto es igual al número (el valor absoluto de 52 es 52), mientras que, para los negativos, su valor absoluto es el número multiplicado por -1 (el valor absoluto de -52 es 52).

/**********/

Ejercicio 2:

Ejercicio pensado por mí.

```
numero = float(input(“Ingrese un numero: “))

if numero >= 0:
    print(“El valor absoluto del numero ingresado es: “, numero)
else:
    print(“El valor del numero ingresado es: “, -1*numero)
```

Ejemplo 1:

```
numero = int(input(“Ingrese un numero entero: “)

if numero > 0:
    printf(“El valor absoluto de {} es de: {}”.format(numero,numero))
else:
    printf(“El valor absoluto de {} es de: {}”.format(numero, -1*numero))
```

Ejemplo 2:

```
numero = int(input(“Ingrese un numero entero: “)
```

if numero > 0**:**

    printf(“El valor absoluto de **{}** es de: **{}**”.**format**(numero,numero))

else:

    printf(“El valor absoluto de **{}** es de: **{}**”.**format**(numero, **abs**(numero)))

NOTA:

Un método muy importante en Python para encontrar el valor absoluto de un número es utilizar:

• **abs**(numero)

8.7 Ejercicio 3:

Escribe un programa que pida dos palabras y diga si riman o no. Si coinciden las tres últimas letras tiene que decir que riman. Si coinciden sólo las dos últimas tiene que decir que riman un poco y si no, que no riman.

/***/

Ejercicio 3:

Ejercicio pensado por mí.

palabra1 = **input**(“Ingrese la primer palabra”:)

palabra2 = **input**(“Ingrese la segunda palabra”:)

if palabra1 [-3:] **==** palabra2 [-3: 0]

    print(“**{}** y **{}** riman”.**format**(palabra1, palabra2))

elif palabra1 [-2:] **==** palabra2 [-2: 0]

    print(“**{}** y **{}** riman un poco”.**format**(palabra1, palabra2))

else:

    print(“**{}** y **{}** NO riman”.**format**(palabra1, palabra2))

Ejemplo:

```
palabra1 = input(“Ingrese la primer palabra”: )
palabra2 = input(“Ingrese la segunda palabra”: )

if len(palabra1) < 3  or  len(palabra2) < 3:
        print(“NO riman, por que tienen menos de 3 caracteres”)
elif palabra1 [-3: ] == palabra2 [-3: 0]
        print(“Las palabras riman”)
elif palabra1 [-2: ] == palabra2 [-2: 0]
        print(“Las palabras riman un poco”)
else:
        print(“Las palabras no riman”)
```

8.8 Ejercicio 4:

Crear un programa que permita al usuario elegir un candidato por el cual votar. Las posibilidades son: candidato A por el partido rojo, candidato B por el partido verde, candidato C por el partido azul. Según el candidato elegido (A, B o C) se le debe imprimir el mensaje “Usted ha votado por el partido [color que corresponda al candidato elegido]”. Si el usuario ingresa una opción que no corresponde a ninguno de los candidatos disponibles, indicar “Opción errónea”.

Pista: Si la letra ingresada por el usuario es en minúscula, el programa debe convertirla en mayúscula.

```
/*********************************************************************************************/
```

Ejercicio 4:

Ejercicio pensado por mí.

```
voto = input(“Ingrese su voto: “)

if voto.upper() == “A”:
        print(“Ud. ha votado al canditato {} del partido rojo”.format(voto.upper()))
if voto.upper() == “B”:
```

```
        print("Ud. ha votado al canditato {} del partido verde".format(voto.upper()))
    if voto.upper() == "C":
        print("Ud. ha votado al canditato {} del partido azul".format(voto.upper()))
    else:
        print("Opcion errónea")
```

9. SECCIÓN 9: TEORÍA DE ESTRUCTURAS DE DATOS

Existen muchos tipos de estructuras de datos: listas, tuplas, diccionarios, colas, arreglos, etc.

9.1 Listas, Tuplas y Diccionarios (Teoría):

• ***Lista:*** Es un tipo de colección ordenada. Sería equivalente a lo que en otros lenguajes se conoce como ***arrays*** o vectores. Es decir, un conjunto de datos que están agrupados.

Sintaxis de las Listas					
[	'Pan',	40,	35.86,	'Tortilla',	]
	[0]	[1]	[2]	[3]	

• ***Tupla:*** Es una secuencia de valores similar a una lista. La principal diferencia es que las tuplas son inmutables, es decir, una "lista que no se puede modificar".

Sintaxis de las Tuplas					
(	'Jugar',	23.78,	"Juan",	"Tupla",	)
	[0]	[1]	[2]	[3]	

NOTA: A veces se suele colocar a la *Tupla* entre llaves {}.

• ***Diccionarios:*** Son colecciones que relacionan una clave y un valor. Por ejemplo, la clave va a ser un nombre que se le va a asignar a un conjunto o un solo valor.

Sintaxis de los Diccionarios							
{	"Uno":	1,	"Dos":	2,	"Tres":	3	}
	Clave	***Valor***	***Clave***	***Valor***	***Clave***	***Valor***	

10. SECCIÓN 10: ESTRUCTURAS DE DATOS: LISTAS

10.1 Listas:

Es una estructura de datos ordenada que, en otros lenguajes, se pueden llamar ***arrays*** o ***vectores***.

Para definir una lista se deben utilizar los ***corchetes*** y puedes ser igualadas a una variable. Pueden ser ***homogénea*** o ***heterogénea***, las primeras se refieren a listas con el mismo tipo de datos mientras que la heterogénea implica que se puedan combinar tipos de datos. Las listas son del tipo ***list***, mientras que los elementos que ocupen esa lista pueden ser ***string***, ***int***, ***float***, etc.

El valor de una lista puede cambiarse por otros valores.

/***/

Ejemplo:

```
lista = [‘Python’, 120, ‘Nombre’, 4.14, 6.28]


print(lista)

print(len(lista))

lista[0] = ‘python’

print(lista)
```

NOTA:

• ista[0] = ‘python’. Se ha cambiado el valor de la posición 0 de la lista original a “python”.

10.2 Debanado de Listas:

Ejemplo:

```
lista = [10, 20, 3.14, ‘Walter’, ‘Coto’, 7, “Estudiante”, ‘cuaderno’]


print(lista[5])

print(lista[0:5])
```

print(lista[:2])

print(lista[1:])

print(lista[-1])

NOTAS:

• **print**(lista[0:5]). Imprime los primeros 5 elementos de la lista.

• **print**(lista[:2]). Imprime los primeros 2 elementos de la lista.

• **print**(lista[1:]). Imprime los elementos de la lista menos el primero.

• **print**(lista[-1]). Imprime el primer elemento de la lista comenzando desde el final.

10.3 Métodos de listas – parte 1:

Es una función que agrega un comportamiento a una lista (u objeto). Para agregar más datos a una lista se emplea el método ***append***; el dato agregado ocupa la siguiente posición al último elemento de la lista.

El método ***insert*** permite agregar más datos a la lista y, a diferencia del método ***append***, permite ubicar el dato en una posición determinada de la lista.

/**********/

Ejemplo – Método ***append***:

lista = [1, 2, 3, 4, 5]

print(lista)

lista.***append***(80)

print(lista)

NOTA:

• lista.***append***(80). Agrega el dato “80” en la última posición de la lista

Ejemplo – Método ***insert***:

```
lista = [1, 2, 3, 4, 5]
print(lista)

lista.insert(2, 80)
print(lista)
```

NOTA:

• lista.***insert***(2, 80). Agrega el dato “80” en la posición 2 de la lista.

Nótese que este método recibe dos parámetros: el primero corresponde a la posición que quiero almacenar el dato y, el segundo, el dato a agregar.

10.4 Métodos de listas - parte 2:

El método ***count()*** me permite contar la cantidad de veces que se repite un dato en la lista.

El método ***index()*** permite ubicar la posición del dato que se necesita buscar. En el caso que exista el mismo dato en otra posición, este método, retornará la posición del primer dato buscado que aparece en la lista.

El método ***sort()*** permite ordenar una lista en orden ascendente.

El método ***reverse()*** permite ordenar una lista en orden descendente.

/**********/

Ejemplo – Método ***count()***:

```
lista = [1, 2, 3, 4, 5]
print(lista.count(5))
```

NOTA:

• **print**(lista.**count**(5)). Permite contar la cantidad de veces que aparece el dato “5”.

Ejemplo – Método ***index()***:

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

print(lista.**index**(4))

NOTA:

• **print**(lista.**index**(4)). Permite reconocer la posición del dato “4”.

Ejemplo – Método ***sort()***:

lista = [5, 3, 1, 4, 2]

lista.**sort()**

print(lista)

NOTA:

• lista.**sort**(). Permite ordenar de modo ascendente una lista.

Ejemplo – Método ***reserve()***:

lista = [5, 3, 1, 4, 2]

lista.**reverse()**

print(lista)

NOTA:

• lista.**reverse**(). Permite ordenar de modo descendente una lista.

10.5 Métodos de listas - parte 3:

El método ***pop()*** permite eliminar el dato de la lista indicado en el argumento; si no se le pasa argumento, elimina el último dato.

El método ***remove()*** permite eliminar el dato de la lista que le pasemos como argumento independientemente de la ubicación donde se encuentre.

/**********/

Ejemplo – Método ***pop()***:

lista = [“Python”, 24, “Rene”, “Alexander”, 12]

lista.**pop()**

print(lista)

NOTA:

• lista.**pop()**. Elimina el dato de la lista cuya posición se le indica en el argumento; en este caso, por defecto elimina el último dato.

Ejemplo – Método ***remove()***:

lista = [“Python”, 24, “Rene”, “Alexander”, 12]

lista.**remove(**“Python”**)**

print(lista)

NOTA:

• lista.**remove(**“Python”**)**. Elimina el dato de la lista indicado en el argumento independientemente de la posición que ocupe.

10.6 Llenar una lista vacía:

Ejemplo:

lista = [1, 2, 3]

lista2 = [4, 5, 6]

lista3 = lista + lista2

print(lista3)

NOTA: *lista* y *lista2* se unen para formar una única lista, *lista3.*

Ejemplo – llenar una lista:

```
lista = [ ]


edad = int(input(“Ingrese su edad: “)
lista.append(edad)
print(lista)
```

10.7 Ejercicio 1:

En la siguiente lista, debes hacer un programa que muestre los valores al usuario, a su vez, debe pedir dos datos y esos que se han ingresados deben ser sustituidos en el primer y segundo lugar:

[20 ,50, “Curso”, ‘Python’, 3.14]

/**/

Ejercicio 1:

Ejercicio pensado por mi.

```
lista = [20, 50, “Curso”, ‘Python’, 3.14]
print(lista(

lista[0] = input(“Ingrese el dato 1: “)
lista[1] = input(“Ingrese el dato 2: “)
print(lista)
```

Ejemplo:

```
lista = [20, 50, “Curso”, ‘Python’, 3.14]
print(“Estis son los valores actuales de la lista: “, lista)

palabra1 = input(“Ingrese el primer valor para sustituir en el espacio 1: “)
palabra2 = input(“Ingrese el segundo valor para sustituir en el espacio 2: “)
```

lista [0] = palabra1

lista [1] = palabra2

print(“El nuevo valor de la lista es: **{}**”.**format**(lista))

NOTA: Nótese que el código más óptimo es el que desarrollé.

10.8 Ejercicio 2:

Escribe una lista que tenga los números del 1 al 10. Luego, debes hacer que los datos que están en las posiciones 4, 7 y 9 sean multiplicados por 2; por último, mostrar la lista nueva con los nuevos datos.

/**/

Ejercicio 2:

Ejercicio pensado por mi.

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(lista)

lista[3] = lista[3]*2
lista[6] = lista[6]*2
lista[8] = lista[8]*2
print(lista)
```

Ejemplo:

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(lista)

lista[3] *= 2
lista[6] *= 2
```

```
lista[8] *= 2
print(lista)
```

NOTA: Nótese que el código óptimo es del ejemplo.

11. SECCIÓN 11: ESTRUCTURAS DE DATOS - DICCIONARIOS

11.1 Diccionarios:

Un diccionario es un espacio en memoria que almacena datos ordenados, pero, además, tiene dos elementos: la ***clave*** y el ***valor***; esa es la principal diferencia entre una lista y un diccionario. Las claves son únicas e irrepetibles.

Recordemos que un diccionario es un tipo de variable ***dict*** y se denota con llaves "{ }".

/**********/

Ejemplo:

```
diccionario = {'Usuario' : "wcoto", 'Contracenia' : 12345}
print(diccionario)
print(type(diccionario))
```

11.2 Trabajar con Diccionarios:

Ejemplo:

```
diccionario = {'Nombre': "Damian", 'Apellido': "Angelucci", 'Estatura': 1.78}
print(diccionario)

print(diccionario.keys())
print(diccionario.values())

print(diccionario["Estatura"])

diccionario["Peso"] = "58 kg"
print(diccionario)

diccionario["Nombre"] = "Walter"
print(diccionario)
```

NOTAS:

• **print**(diccionario.**keys()**). Muestra sólo las claves y NO los valores.

• **print**(diccionario.**values()**). Muestra los valores y NO las claves.

• **print**(diccionario[“Estatura”]). Muestra sólo el valor de la clave “Estatura”.

• diccionario[“Peso”] = “58 kg”. Permite agregar más claves y valores.

• diccionario[“Nombre”] = “Walter”. Modifica el valor de la clave “Nombre”.

11.3 Métodos de Diccionarios:

Ejemplo:

```
diccionario = {1: 2, 2: 3, 3: 4}
diccionario2 = {4: 5, 6: 7}
print(diccionario)

diccionario.pop(3)

diccionario.clear()

print(diccionario.get(2))

diccionario.setdefault(4,5)
print(diccionario)

diccionario.update(diccionario2)
print(diccionario)

diccionario.copy()
print(diccionario)
```

NOTAS:

• diccionario.**pop(3)**. Permite eliminar la clave 3.

• diccionario.**clear()**. Permite borrar todo el diccionario.

• **print**(diccionario.**get(**2**)**). Imprime el valor de la clave 2.

• diccionario.**setdefault**(4,5). Agrega una clave “4” con el valor “5”.

• diccionario.**update**(diccionario2). Permite adicionar el “diccionario2” al diccionario “diccionario”.

• diccionario.**copy()**. Permite copiar el diccionario.

11.4 Ejercicio 1:

En el siguiente diccionario se encuentran capitales de los países en el mundo, debes realizar un programa que pida un país al usuario, y muestre la capital de ese país, en dado caso el país no este en el diccionario, se debe mostrar un mensaje diciendo que ese país no se encuentra.

{“Guatemala”: “Ciudad de Guatemala”, “El Salvador”: “San Salvador”, “Honduras”: “Tegucigalpa”, “Nicaragua”: “Managua”, “Costa Rica”: “San Jose”, “Panama”: “Panama”, “Argentina”: “Buenos Aires”, “Colombia”: “Bogota”, “Venezuela”: “Caracas”, “Espania”: “Madrid”}

/**********/

Ejemplo 1:

```
diccionario = {
    "Guatemala": "Ciudad de Guatemala",
    "El Salvador": "San Salvador",
    "Honduras": "Tegucigalpa",
    "Nicaragua": "Managua",
    "Costa Rica": "San Jose",
    "Panama": "Panama",
    "Argentina": "Buenos Aires",
    "Colombia": "Bogota",
    "Venezuela": "Caracas",
```

```
    "Espania": "Madrid"
    }

país = input(“Ingrese el nombre de un país: “)
letra.capitalize() in diccionario

if letra == True:
        print(diccionario[país.capitalize()])
else:
        print(“El país no se encuentra en el diccionario”)
```

Ejemplo 2:

```
diccionario = {
    "Guatemala": "Ciudad de Guatemala",
    "El Salvador": "San Salvador",
    "Honduras": "Tegucigalpa",
    "Nicaragua": "Managua",
    "Costa Rica": "San Jose",
    "Panama": "Panama",
    "Argentina": "Buenos Aires",
    "Colombia": "Bogota",
    "Venezuela": "Caracas",
    "Espania": "Madrid"
    }

país = input(“Ingrese el nombre de un país: “)

if país.capitalize() in diccionario:
        print(diccionario[país.capitalize()])
```

else:

print(“El país no se encuentra en el diccionario”)

NOTAS: Nótese que el Ejemplo 2 es un código óptimo que el Ejemplo 1.

• letra.**capitalize() in** diccionario. Permite poner en mayúscula la primera letra de la palabra ingresada Y VERIFICAR que dicha palabra se encuentre en “diccionario”.

• **if** letra == **True:**. Puede abreviarse de la siguiente manera: **if** letra**:**. El lenguaje sobre entiende que es un valor ***True***.

Nótese que cuando se ingresa el país Costa Rica el programa arroja error, esto se debe al espacio que existe en la palabra. Habría que encontrar la manera de evitar que esto se produzca.

11.5 Ejercicio 2:

Con el siguiente diccionario, debes crear un programa que pregunte al usuario por un número; el programa debe imprimir el jugador al que hace referencia ese número.

{

1: “Casillas”, 15: “Ramos”,

3: “Pique”, 5: “Puyol”,

11: “Capdevila”, 14: “Xabi Alonso”,

16: “Busquets”, 8: “Xavi Hernandez”,

18: “Pedrito”, 6: “Iniesta”,

7: “Villa”

}

/**/

Ejercicio 2:

Ejercicio pensado por mí.

diccionario = {

```
        1: “Casillas”,
        15: “Ramos”,
        3: “Pique”,
        5: “Puyol”,
        11: “Capdevila”,
        14: “Xabi Alonso”,
        16: “Busquets”,
        8: “Xavi Hernandez”,
        18: “Pedrito”,
        6: “Iniesta”,
        7: “Villa”
}

clave = int(input(“Ingrese un numero: “)

if clave in diccionario:
        print(“El numero {} corresponde a: {}”.format(clave, diccionario[clave]))
else:
        print(“El numero {} no se encuentra en el diccionario”)
```

12. SECCIÓN 12: OTRAS ESTRUCTURAS DE DATOS

12.1 Conjuntos:

Un conjunto en Python es una estructura de datos que nos va a permitir, casi como una lista, agrupar datos. La diferencia entre ambos es que el conjunto no acepta datos repetidos.

La sintaxis para un conjunto pueden ser las llaves, los corchetes o los paréntesis. En el caso de que se pongan entre llaves, lo diferenciaré por que los datos se colocan como una lista.

Los conjuntos son un tipo de dato ***set***.

/***/

Ejemplo 1:

```
conjunto = {1, 2, 3, 4, 5}
print(type(conjunto))
```

Ejemplo 2:

```
conjunto = set[1, 2, 3, 4, 5]
print(type(conjunto))
```

Ejemplo 3:

```
conjunto = set((1, 2, 3, 4, 5))
print(type(conjunto))
```

NOTA: Los ejemplos representan 3 maneras diferentes de representar un conjunto. En el “Ejemplo 1”, se escribe el conjunto en forma directa; el “Ejemplo 2”, se trata de una **lista** convertida a **conjunto**; y, el “Ejemplo 3”, de una **tupla** convertida a **conjunto**.

12.2 Métodos de los Conjuntos:

Recordando algunas cuestiones, la diferencia entre una lista y un conjunto es que no se pueden repetir valores.

/**/

Ejemplo – Método ***add()***:

```
conjunto = {1, 2, 3, 4, 5}
print(conjunto)

conjunto.add(20)
print(conjunto)
```

NOTA: Añade un nuevo dato al conjunto, en este caso, el número 20.

Ejemplo – Método ***remove()***:

```
conjunto = {1, 2, 3, 4, 5}
print(conjunto)

conjunto.remove(1)
print(conjunto)
```

NOTA: Elimina un dato del conjunto, en este caso, el número 1.

Ejemplo – Método ***pop()***:

```
conjunto = {1, 2, 3, 4, 5}
print(conjunto)

conjunto.pop()
print(conjunto)
```

NOTA: Elimina el primer dato del conjunto, en este caso, el número 2 dado que previamente se eliminó el número 1.

Ejemplo – Método ***updato()***:

```
conjunto = {1, 2, 3, 4, 5}
print(conjunto)

conjunto.update([1, 2, 3])
print(conjunto)
```

NOTA: Adiciona a los datos del conjunto original, nuevos datos iterables. Los elementos repetidos NO se agregan. En este caso, se agregaron los números 1 y 2 al final del conjunto.

Ejemplo – Método ***clear()***:

```
conjunto = {1, 2, 3, 4, 5}
print(conjunto)

conjunto.clear()
print(conjunto)
```

NOTA: Borra los datos del conjunto.

12.3 Tuplas:

Es una lista que no se puede modificar, es decir, sus datos son inmutables a diferencia de una lista.

No tienen métodos dado que sus datos son inmutables.

/**/

Ejemplo:

```
tupla = 1, 2, 3, 4, 5
tupla2 = 6, 7, 8, 9, 10
```

print(tupla)

print(**type**(tupla))

print(tupla[2])

print(tupla[0:3])

print(tupla+tupla2)

NOTAS:

• **print**(tupla[2]). Imprime hasta la posición 2 de los datos de la tupla.

• **print**(tupla[0:3]). Debanado de tupla desde la posición 0 a la posición 2 INCLUSIVE.

• **print**(tupla+tupla2). Imprime la suma de “tupla” y “tupla2”.

12.4 Ejericio 1:

Escribir una tupla con los meses del año, luego, pide al usuario un número, el que haya ingresado, es el mes que debe mostrar en la tupla.

/**********/

Ejemplo:

tupla = (“Enero”, “Febrero”, “Marzo”, “Abril”, “Mayo”, “Junio”, “Julio”, “Agosto”, “Septiembre”, “Octubre”, “Noviembre”, “Diciembre”)

numero = **int**(**input**(“Ingrese el numero del mes: “)

print(“El numero **{}** corresponde al mes: **{}**”**.format**(**numero, tupla**[numero-1]))

12.5 Ejericio 2:

Escribir una tupla que tenga las letras del alfabeto. Luego, debes pedir al usuario un número, el que haya ingresado, es la letra que debe imprimir el programa.

/***/

Ejercicio 2:

Pensado por mí.

```
tupla = ("A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R",
"S", "T", "U", "V", "W", "X", "Y", "Z")

numero = int(input(“Ingrese un numero del alfabeto: “)

print(“El numero ingresado corresponde a: “, tupla[numero])
```

13. SECCIÓN 13: BUCLES

13.1 Bucles:

Un ***bucle*** es una sentencia de código que se va a repetir tantas veces, a través de una variable iteradora, hasta que la condición se cumpla.

Importante!:

Actualización de variables:

x = 0

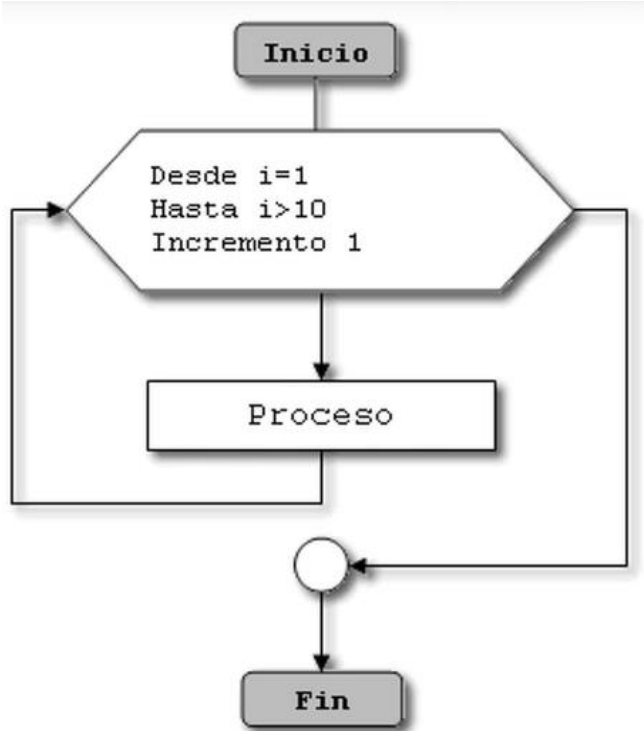
x +=1

Iteración:

Es repetir un conjunto de instrucciones las veces que sea necesario.

Bucles:

O ***loop*** son estructuras, con un segmento de código, que se repiten tantas veces que se cumpla una condición determinada.



Iterador y Break:

Un iterador es un contador que poseen los bucles, es importante que tengan uno, de otra forma… el bucle sería infinito.

La ***Break*** es una forma de detener un bucle al llegar a una condición ya definida.

WHILE:

Sintaxis:

while<condición>:

<instrucciones de código>

<incrementar o decrementar el contador>

Ejemplo - estructura:

```
n = 5
while n>0:
    print(n)
    n = n-1
print("!Despegue!")
```

FOR:

Sirve para crear una sóla vez al contedor y darle un nombre. Muchas veces se utiliza para recorrer una lista.

Sintaxis:

for <contador> **in** <nombre de lista>**:**

<segmento de código>

Ejemplo - estructura:

```
lista = [1, 2, 3, 4]
for i in lista:
    print(i)
```

13.2 While:

Un bucle *while* va a tener una condición, un contador o iterador y una sentencia que se va a ejecutar conforme pase el bucle.

/**/

Ejemplo:

```
i = 0
while i < 10:
    print("Estoy iterando, voy por el salto: ", i)
    i +=1
```

13.3 Ejercicio 1:

Escribir un programa que pida un numero al usuario y muestre las tablas de multiplicar de ese número.

/**/

Ejercicio 1:

Pensado por mí.

```
n = int(input("Ingrese un numero: ")

i = 0
while i <= 10:
    print("Tabla de multiplicación: {} x {} = {}".format(n, i, n*i))
    i += 1
```

Ejemplo:

```
numero = int(input("Ingrese un numero: ")

i = 0
```

```
multi = 0


while i <= 10:
        multi = numero * i
        print("{} x {} = {}".format(numero, i, multi))
        i += 1
```

13.4 Ejercicio 2:

Escribir un programa que pregunte al usuario su edad y muestre por pantalla todos los años que ha cumplido (desde 1 hasta su edad).

```
/*************************************************************************************************/
```

Ejercicio 2:

Pensado por mí.

```
edad = int(input("Ingrese su edad: ")


i = 0
while i <= edad:
        print("Los anios cumplidos fueron: ", i)
        i += 1
```

Ejemplo:

```
edad = int(input("Ingrese su edad: ")


i = 1
while i != edad:
        print("Haz cumplido: {} anios ".format(i))
        i += 1
```

13.5 For:

En Python, el bucle ***For*** va a cumplir dos objetivos: recorrer elementos iterables y recorrer rangos. Los elementos iterables son aquel que se le permite, por medio de un iterador, tomar todos cada uno de los valores que conforman este elemento, el más popular de todos: ***Lista***.

/**********/

Ejemplo:

```
lista = [“Uno”, “Dos”, “Tres”]

tupla = (1, 2, 3, 4, 5)


for i in lista:
    print(i)


for j in tupla:
    print(j)
```

NOTA: El iterador i toma cada posición de los elementos que contenga la lista.

13.6 Función Range:

Como hemos mencionado en la lección anterior, en Python, el segundo objetivo del bucle ***For*** es recorrer rangos; que son parte de una función. Esta función ***Range*** nos va a marcar o delimitar un punto específico en el que un bucle va a recorrerse. Esta función *Range* va a requerir dos argumentos: inicio y fin del recorrido. También es posible un tercer argumento que nos va a permitir mostrar los elementos del rango cada “x” elementos.

/**********/

Ejemplo:

```
for i in range(1, 10, 2):
    print(i)
```

NOTAS:

• **range**(1, 10, 2). Primer argumento: inicio del contador. Segundo argumento: fin del contador. Tercer elemento:

• También es posible agregar números negativos al argumento.

13.7 Continue y Break:

Continue y Break sirven para detener una iteración o para saltarnos una iteración.

/**********/

Ejemplo - Break:

```
for i in range(1, 11):
    if i == 6:
        break #Detiene el bucle cuando es igual a 6.
    print(i)
```

Ejemplo - Continue:

```
for i in range(1, 11):
    if i == 6:
        continue #Salta al número 6 y continua con los siguientes elementos.
    print(i)
```

13.8 Ejercicio 1:

Imprimir por pantalla los números del 1 al 10, luego, pedir al usuario dos números y mostrar el rango de números entre ambas cifras.

/**********/

Ejercicio 1:

Pensado por mí.

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(lista)

num1 = int(input(“Ingrese el primer numero: “)
num2 = int(input(“Ingrese el segundo numero: “)

for i in range(num1, num2):
        print(i)
```

Ejemplo:

```
for i in range(1, 11):
        print(i)

num1 = int(input(“Ingrese el primer numero: “)
num2 = int(input(“Ingrese el segundo numero: “)

for i in range(num1, num2+1):
        print(i)
```

13.9 Ejercicio 2:

Pedir al usuario que ingrese 2 números, después, se debe mostrar el rango de esos 2 números, pero, solo imprimiendo los números que sean impares.

```
/*********************************************************************************************/
```

Ejemplo 1:

```
num1 = int(input(“Ingrese el primer numero: “))
num2 = int(input(“Ingrese el segundo numero: “))

for i in range(num1, num2+1):
        if i %2 == 0:
```

```
            continue
        print(i)
```

NOTA: La clave para resolver este problema está en la condición ***if***.

Ejemplo 2:

```
num1 = int(input("Ingrese el primer numero: "))
num2 = int(input("Ingrese el segundo numero: "))

for i in range(num1, num2+1, 2):
        print(i)
```

NOTA: Esta es una forma de resolver el problema, aunque sea incorrecto, dado que le estamos pidiendo al rango que vaya de 2 en 2; en el caso que se ingrese un número impar, la solución es correcta pero si se ingresa un número par, la solución es errónea.

La mejor solución es la del "Ejemplo 1".

14. SECCIÓN 14: FUNCIONES

14.1 Funciones (Teoría):

En el contexto de la programación, una función es una secuencia de sentencias que realizan una operación y que reciben un nombre. Cuando se define una función, se especifica el nombre y la secuencia de sentencias. Más adelante, se puede "llamar" a la función por ese nombre.

/**/

Ejemplo:

```
def Función():
    print("Hola Mundo")


Función()
Función()
```

```
>>>Hola Mundo
>>>Hola Mundo
```

• Parámetro:

Son variables que se definen al declarar funciones, las cuales van a ser utilizadas para enviar valores al hacer un llamado a la función

Ejemplo:

```
def Función(mensaje):
    print(mensaje)
```

• Argumentos:

Son los valores que van a tomar cada uno de los parámetros que se han definido en las funciones.

Ejemplo:

```
def Funcion("Hola Mundo")
    print(mensaje)
```

Funcion(“Hola mundo”)

>>>Hola Mundo

14.2 Funciones integradas de Python:

Ejemplo:

num = “70”

lista = [12, 60, 70, 1, -2]

print(**type**(num))

print(**type**(**int**(num)))

print(**type**(**float**(num)))

print(**len**(lista))

print(**max**(lista))

print(**min**(lista))

NOTAS:

• **print**(**len**(lista)). Reconoce la cantidad de elementos de “lista”.

• **print**(**max**(lista)). Reconoce el mayor valor de “lista”.

• **print**(**min**(lista)). Reconoce el mínimo valor de “lista”.

14.3 Funciones:

Para crear nuestras propias funciones, hay que utilizar la palabra reservada propia del lenguaje, en este caso, ***def***. Con esta palabra, le estamos indicando a Python que vamos a crear una función.

Sintaxis:

def <nombre de la función>()**:**

<sentencia>

/***/

Ejemplo 1:

```
def saludo():
    print("Hola Walter Coto")
    print("Esta es otra sentencia")

saludo()
```

Ejemplo 2:

```
def tabla7():
    for i in range(11):
        print("7 x {} = {}".format(i, i*7))

tabla7()
```

NOTA: Nótese que se crea la función con su sentencia y, fuera de estas instrucciones, se muestra nuevamente la función creada que permitirá resolver el set de instrucciones; sin ella, dichas instrucciones no serán efectuadas.

14.4 Ejercicio 1:

Crear un programa que tenga una lista, luego crear una función con la cual se van a pedir números al usuario para agregar a la lista. Debes crear una segunda función en donde se ordenen los números pares e impares dentro de dos listas nuevas.

/***/

Ejemplo 1:

```
lista = []
num = 0
```

```
def pedir():
    i = 0
    while i <= 5:
        num = float(input("Ingrese un numero: "))
        lista.append()
        i +=1

def ordenar():
    lista.sort()
    pares = []
    impares = []
    for i in lista:
        if I %2 == 0:
            pares.append(i)
        else:
            impares.append(i)
    print(pares)
    print(impares)

pedir()
ordenar()
```

14.5 Ejercicio 2:

Escribir una función que reciba un número entero positivo y devuelva su factorial.

/***/

Ejemplo 1:

```
def factorial():
```

```
    num = int(input("Ingrese un numero entero positivo: "))
    if num > 0:
        factorial = 1
        for i in range(1, num+1):
            factorial = factorial * i
        print(factorial)
    else:
        print("El numero es negativo y no podemos proceder")

factorial()
```

Ejemplo 2:

```
def factorial():
    num = int(input("Ingrese un numero entero positivo: "))
    if num > 0:
        print(factorial(num))
    else:
        print("El numero es negativo y no podemos proceder")

factorial()
```

NOTA: El "Ejemplo 2" resulta ser más óptimo que el "Ejemplo 1" dado que nos ahorramos líneas de código, es decir, espacio en memoria.

14.6 Funciones Matemáticas:

Son funciones matemáticas cargadas por defecto en Python, más bien, es una librería de funciones matemáticas que utiliza Python.

/**********/

Ejemplo:

import math *#Libería de Matemática.*

import random *#Libreria de métodos aleatorios*

print(**math.pow**(10,2)) *#Potencia de un número.*

print(**pow**(10,2)) *#Idem anterior. No requiere importar la librería math*

print(**math.sqrt**(100)) *#Raiz cuadrada de un número. Resultado en float.*

print(**math.isqrt**(100)) *# Raiz cuadrada de un número. Resultado en int*

print(**math.sin**(90)) *#Seno de un número.*

print(**math.cos**(90)) *#Coseno de un número.*

print(**math.tan**(90)) *#Tangente de un número.*

print(**math.factorial**(5)) *#Factorial de un número.*

print(**random.randint**(1, 100)) *#Elije, al azar, un numero entre 1 y 100.*

14.7 Return de Funciones:

En algunas funciones, podemos determinar que valor va a regresar. Algunas funciones lo hacen de manera predeterminada y otras, debemos configurarla nosotros.

El valor que va a enviar la función se da a partir de la palabra reservada ***return***.

Sintaxis:

def <nombre de la función>()**:**

<sentencias>

return

/**/

Ejemplo – Return con valor predefinido:

def entero()**:**

print(“Este es un dato entero: “)

```
    return 10

def decimal():
    print("Este es un dato decimal: ")
    return 90.99

def frase():
    return "Mi nombre es Damian Angelucci"

print(entero())
print(decimal())
print(frase())
```

NOTA: Nótese que para poder ver el valor del ***return***, se debe imprimir a la función creada.

Ejemplo – Return con asignaciones:

```
def asignacion():
    return 1, 2, 3, 4, 5

a, b, c, d, e = asignacion()

print(a)
print(b)
print(c)
print(d)
print(e)
```

14.8 Ejercicio 3:

Crear una función que pida dos números. Si el primero es mayor al segundo, el programa debe retornar el valor de 1; si el segundo es mayor al primero, debe retornar -1; si ambos son iguales, debe retornar 0.

/**/

Ejercicio:

Pensado por mí.

```
def funcion():
    num = float(input("Ingrese el primer numero: "))
    num2 = float(input("Ingrese el segundo numero: "))
    if num > num2:
        return 1
    elif num2 > num:
        return -1
    else:
        return 0

print(funcion())
```

Ejemplo:

```
def numeros():
    num1 = float(input("Ingrese el primer numero: "))
    num2 = float(input("Ingrese el segundo numero: "))
    if num1 > num2:
        return 1
    elif num2 > num1:
        return -1
    else:
        return 0

print(numeros())
```

14.9 Ejercicio 4:

Escribir una función que calcule el total de una factura tras aplicarle el IVA. La función debe recibir la cantidad sin IVA y el porcentaje de IVA a aplicar, y devolver el total de la factura. Si se invoca la función sin pasarle el porcentaje de IVA, deberá aplicar un 21%.

/***/

Ejemplo:

```
def total():
    monto = float(input("Ingrese el valor total del producto que estás pagando: "))
    iva = int(input("Ingrese el valor del IVA: "))

    if iva != 0:
        if iva >0:
            totalPagar = (monto * iva) / 100 + monto
            return totalPagar
        else:
            return "El monto de IVA es negativo. No podemos avanzar"

    else:
        totalPagar = (monto*1.21)
        return totalPagar

print("El total de su monto es: ", total())
```

14.10 Parámetros y Argumentos:

La forma más común de utilizar variables es a través de ***parámetros*** y ***argumentos***; un ***parámetro***, es una variable que se va a utilizar cuando se crea la

función; un ***argumento***, es el valor que se le va asignar a un *parámetro* cuando llamamos a la función.

/**/

Ejemplo 1– Parámetros y argumentos:

```
def suma(num1, num2):
    suma = num1 + num2
    return suma

print(suma(10, 35))
print(suma(100, 50))
```

Ejemplo 2:

```
def sumar():
    num1 = 20
    num2 = 30
    suma = num1 + num2
    return suma

print(sumar())
```

NOTA: Nótese la diferencia entre el Ejemplo 1 y 2. En el primero, colocamos argumento en la función "*num1*" y "*num2*", la cual nos permite reutilizarla colocando al final sus valores respectivos 10 y 35 y, en su reutilización, 100 y 50. El Ejemplo 2, no nos permite reutilizar el código, por lo tanto, en conveniente utilizar el código del Ejemplo 1.

14.11 Variables Globales:

Son variables definidas dentro de una función a través de la palabra reservada ***global*** y nos permiten utilizar dichas variables en sentencias escritas fuera de dicha función.

Sintaxis:

global <variable>

/**/

Ejemplo:

```
def valores():
    global num1
    global num2
    num1 = 110
    num2 = 40
    resultado = num1 + num2
    return resultado

print(valores())

resta = num1 – num2
print(resta)
```

NOTA: La forma de expresar una variable global y que, por ejemplo, pueda utilizarse en la sentencia de "resta" es como se menciona en el ejemplo. No puede ubicarse antes y fuera de la función ya que dicha sentencia no reconocerá las variables producto de la resta.

14.12 Argumentos indefinidos:

Hay parámetros que pueden ser *tuplas* y *listas* pero para ellos hay que tener cierto cuidado en la declaración del mismo.

Un valor indefinido es cuando se desconoce la cantidad de valores que se le van a colocar a la función, o, en algún punto del código, se desconoce cuantos datos

van a ingresar en la función. En este caso, lo que hay que hacer es colocar un ***argumento indefinido*** y, luego, se podrá conocer la cantidad de datos ingresados.

/**/

Ejemplo – **int**, **float** y **str**:

```
• def argumento (num):
        return type(num)

print(argumento(10))  #Valor entero

• def argumento (num):
        return type(num)

print(argumento(10,545))  #Valor flotante

• def argumento (num):
        return type(num)

print(argumento("Cadena"))  #Cadena de texto
```

Ejemplo – **tuple**:

```
• def argumento (*num):
        return type(num)

print(argumento(10, 20, 30, 40, 50))  #Tupla
```

NOTA: Nótese que, en el argumento de la función, está el símbolo "*" que representa que los valores otorgados al argumento se almacenan en una ***tupla***.

14.13 Ejercicio 5:

Crear un programa que tenga dos funciones, una que contenga el área de un cuadrado con argumentos de base y altura; y la otra el área de un círculo con argumento de radio.

/***/

Ejericio:

Pensado por mí.

```
import math

def cuadrado(base,altura)):
    área = base * altura
    return (“El área del cuadrado es: “, área)

print(cuadrado(20,20))

def circulo(radio)):
    área = pow(radio,2) * 3.14
    return (“El área del circulo es: “, área)

print(circulo(10))
```

Ejemplo:

```
def areaCuadrado(base,altura)):
    return (base * altura)

print(areaCuadrado(10,10))
```

```
def areaCirculo(radio)):
    return pow(radio, 2) * 3.14

print(areaCirculo(10))
```

NOTA: Los ejemplos resultan más óptimos que el ejercicio pensado por mí dado que utilizan menos código.

14.14 Ejercicio 6:

Escribir una función que reciba una muestra de números en una lista y devuelva su medida.

```
/*****************************************************************************/
```

Ejercicio:

Pensado por mí.

```
def muestra(*lista):
    return print("La longitud de la lista es: "len(lista))

muestra(1, 2, 3, 4, 5)
```

Ejemplo 1:

```
def medida(*tupla):
    print(len(tupla))

medida(2, 3, 4, 10, 20)
```

Ejemplo 2:

```
def medida(*tupla):
```

```
        return len(tupla)

print(medida(2, 3, 4, 10, 20))
```

15. SECCIÓN 15: MANEJO DE ERRORES

15.1 Errores y Excepciones:

Permiten al usuario ingresar nuevamente datos que fueron erróneamente mal ingresados ya sea por el tipo de dato, etc.

Estos métodos de errores y excepciones se dan a partir de las siguientes palabras reservadas: ***try***, ***except*** y ***finally***.

/***/

Ejemplo:

```
while True:
    try:
        edad = int(input("Ingrese su edad: "))
        print("Tu edad es: ", edad)
        break
    except:
        print("Ingresaste un valor erroneo")
    finally:
        print("La ejecución ha finalizado")
```

NOTAS:

• ***try:*** Da la posibilidad de ingresar un dato.

• ***except:*** Da la posibilidad de ingresar un dato nuevamente debido de un error producido.

• ***finally:*** Da por finalizada la ejecución del programa.

Nótese que, el ***while True***, genera un bucle que me permite ingresar tantas veces un dato hasta éste se haya colocado de forma correcta, en este caso, un dato tipo ***int*** y no ***str***.

15.2 Excepciones Múltiples:

Ejemplo – ***ZeroDivisionError***:

```
while True:
    try:
        num1 = int(input("Ingrese un numero: "))
        resultado = 100/num1
        print(resultado)
        break
    except ZeroDivisionError:
        print("No se puede dividir entre cero")
```

NOTA:

• **except ZeroDivisionError:** Error de dividir un número por cero.

Ejemplo – ValueError:

```
while True:
    try:
        edad = int(input("Ingrese su edad: "))
        print("Tu edad es: ", edad)
        break
    except ValueError:
        print("Has colocado un valor erroneo")
```

NOTA:

• **except ValueError:** Error de ingresar un tipo de dato no correspondiente al programa.

Ejemplo – KeyboardInterrupt:

```
while True:
```

```
try:
    edad = int(input("Ingrese su edad: "))
    print("Tu edad es: ", edad)
    break
except KeyboardInterrupt:
    print("\nHas cancelado la ejecución")
    break
```

NOTA:

• **except KeyboardInterrupt:** Error por cancelación de ejecución del programa. Esto se debe a que el usuario ha decidido abortar la ejecución a través del comando del teclado ***CTRL + C***.

16. SECCIÓN 16: PROGRAMACIÓN ORIENTADA A OBJETOS

16.1 Clases y Objetos (Teoría):

La POO permite hacer desarrollo web, inteligencia artificial, machine learning, etc. Permite desarrollar programas que simule lo más posible a la vida cotidiana.

Clases:

Básicamente, es una entidad que define una serie de elementos que determinan un estado (datos) y un comportamiento (operaciones sobre los datos que modifican su estado).

Ejemplo: **class** Galleta():

pass

Objetos:

En un nivel elemental, un objeto es simplemente un trazo de código más estructurado de datos, más pequeños que un programa completo.

Ejemplo: galleta = Galleta()

Un ejemplo más claro de ***clase*** puede ser un molde que se utiliza para darle forma a una galleta; mientras que el ***objeto*** es la galleta.

16.2 Clases y Objetos (Práctica):

Cuando se crea una clase, la primera letra de su nombre siempre debe colocarse en mayúscula.

La clase es del tipo ***type***. Los objetos son del tipo ***__main__.NombreClase***

Sintaxis - Clases:

class <Nombre de la Clase>():

pass

Objetos:

En un nivel elemental, un objeto es simplemente un trazo de código más estructurado de datos, más pequeños que un programa completo.

Ejemplo: galleta = Galleta()

Un ejemplo más claro de ***clase*** puede ser un molde que se utiliza para darle forma a una galleta; mientras que el ***objeto*** es la galleta.

/**/

Ejemplo:

class FabricaTelefonos()**:**

pass

print(**type**(FabricaTelefonos))

celular = FabricaTelefonos()

celular2 = FabricaTelefonos()

print(**type**(celular))

print(**type**(celular2))

NOTA: celular = FabricaTelefonos(). Es un objeto creado en la clase FabricaTelefonos().

16.3 Métodos y Atributos (Teoría):

Los atributos son las *cualidades* que dispone un ***objeto***. Así, por ejemplo, un celular cuenta con las siguientes cualidades: SO, marca, si es táctil, etc.; mientras

que los ***métodos*** son todas las acciones que puede realizar ese celular, por ejemplo, llamar, enviar mensajes, contar con un calendario, calculadora, etc.

Métodos:

Los métodos son funciones, pero que en POO pasan a llamarse ***métodos***, los cuales hacen los objetos. Por ejemplo: Si una clase crea objetos Personas, entonces un método puede ser el hablar, porque las personas hablan.

Sintaxis:

```
class Persona(): #Clase
    def Hablar(): #Método
        print("Hola")
```

Atributos:

Los atributos, son características, cualidades o descripciones que los objetos tienen. Aunque, desde un punto de vista de Código, en realidad son datos que pueden tener los objetos.

Sintaxis:

```
class Carro(): #Clase
    color #Atributo
    llantas #Atributo
    marca #Atributo
```

En definitiva, los ***métodos*** van a las ***clases*** y los ***atributos*** a los ***objetos***. Los *objetos* pueden ejecutar *métodos* y los *métodos* se pueden ligar a los *objetos*.

16.4 Métodos y Atributos (Práctica):

Un ***método*** no es más que una función dentro de una ***clase***. Los métodos de instancias utilizan ***self*** en su argumento.

/***/

Ejemplo:

```
class FabricaTelefonos():
        marca = “Huawei”
        color = “Negro”
        memoriaRam = 32
        almacenamiento = 128

        def llamar(self, mensaje):
                return mensaje

        def escucharMusica(self):
                print(“Estas escuchando musica”)

teléfono = FabricaTelefonos()
print(teléfono.marca)
print(teléfono.color)
print(teléfono.memoriaRam)
print(teléfono.memoria)

print(telefono.llamar(“Hola, con quien hablo?”)
telefono.escucharMusica()
```

16.5 Self e Init:

El método ***init*** es un equivalente al método constructor en otros lenguajes; es decir, es el primer método que se ejecuta al crearse un objeto. ***self*** es una forma de equipar los objetos llamándolos a una instancia. Cada objeto es necesario que tenga atributos que no necesariamente van a ser atributos de una clase sino de métodos de instancias; aquellos métodos creados por el programador.

El método ***self*** nos va a permitir englobar atributos a toda la clase y a todo el programa en general, por ejemplo, **self**.marca = “Huawei”

/**/

Ejemplo - ***self***:

```
class FabricaTelefonos():
    marca = “Samsung”

    def ElaborarHuawei(self):
        self.marca = “Huawei”

teléfono = FabricaTelefonos()
teléfono.ElaborarHuawei()
print(teléfono.marca)
```

NOTA:

• **self.**marca = “Huawei”. Permite considerar el atributo creado en el ***método de instancia*** en todo el programa, de tal manera que en la última línea, pueda imprimirse el valor de este atributo (“Huawei”).

Ejemplo - ***Init***:

```
class FabricaTelefonos():
    def __init__(self, marca, color):
        self.marca = marca
        self.color = color

telefono = FabricaTelefonos(“Huawei”, “Azul”)
print(telefono.marca)
```

16.6 Métodos Espaciales:

Así como existe un método constructor (***init***) que se ejecuta al inicializar el programa, existe un método destructor llamado ***del*** que se ejecuta al finalizar la ejecución de un programa que tiene clases y objetos. El destructor se encarga de destruir o eliminar todos los objetos creados en el programa.

/**/

Ejemplo – ***init, str*** & ***del***:

```
class FabricaTelefonos():
    def __init__(self, marca, color):
        self.marca = marca
        self.color = color
        print("El objeto {} ha sido creado".format(self,marca)

    def __str__(self):
        return "El objeto es {}".format(self.marca)

    def __del__(self):
        print("El objeto {} ha sido destruido".format(self.marca))

teléfono = FabricaTelefonos("Nokia", "Negro")
print(teléfono.marca)
print(teléfono)
```

NOTAS:

• ***Método str:*** Reemplaza la dirección en código de máquina donde se ubica el objeto creado por una descripción más precisa para que el usuario pueda entender.

• ***Método del:*** Permite eliminar todos los objetos creados en el programa al finalizar la ejecución.

16.7 Profundizando en POO:

En Python, hemos visto el método ***self***. Este método NO necesariamente debe llevar la palabra reservada ***self*** como tal, si no que se puede utilizar otra en su lugar, como ***this*** (utilizado en C++), y Python la reconocerá de la misma manera. Las buenas prácticas y el manual del lenguaje indican que se debe utilizar la palabra reservada ***self***.

/***/

Ejemplo:

```
class FabricaTelefonos():
    def __init__(self, marca, *colores, **modelos):
        self.marca = marcas
        self.colores = colores
        self.modelos = modelos

telefono = FabricaTelefonos("Alcatel", "Negro", "Azul", "Rojo", m1 = 500, m2= 1000)
print(teléfono.marca)
print(teléfono.colores)
print(teléfono.modelo)

teléfono.memoria = 500
print(teléfono.memoria)
```

NOTAS:

• marca, *colores, **modelos. Indican que son ***atributos***, ***tuplas*** y ***diccionarios*** respectivamente. Nótese la manera que tiene Python para reconocer qué valores son tuplas ("Negro") y cuáles son diccionarios (m1 = 500).

• teléfono.memoria = 500. Esta es una forma de crear un ***atributo temporal*** que sólo estará almacenado en ese espacio de memoria.

17. SECCIÓN 17: PILARES DE POO

Los pilares de la Programación Orientada a Objetos son 4:

1. Encapsulamiento
2. Abstracción
3. Herencia
4. Polimorfismo

17.1 Encapsulamiento:

Es aplicar sobre los atributos cierto alcance para que estos solo sean utilizados sólo dentro de la clase. Esto sirve para evitar desbordamientos de memoria, evitar ejecuciones innecesarias y demás beneficios que se verán con la práctica.

Sintaxis:

```
self._contador = 0
```

El guion bajo representa el encapsulamiento.

/***/

Ejemplo:

```
class A():
    def __init__(self):
        self.contador = 0

    def incrementar (self):
        self.contador +=1

    def cuenta (self):
        return self.contador
```

```
class B():
    def __init__(self):
        self._contador = 0

    def incrementar (self):
        self._contador +=1

    def cuenta (self):
        return self._contador

print("Objeto 1")
a = A()
print(a.cuenta())
a.incrementar()
print(a.cuenta())

print("Objeto 2")
b = B()
print(b.cuenta())
b.incrementar()
print(b.cuenta())
```

NOTA: **self.**_contador. Implica que se ha creado un atributo *contador* ***encapsulado*** dentro de la clase B().

17.2 <u>Más sobre encapsulamiento</u>:

Es una buena práctica SIEMPRE encapsular los atributos de una clase. Recordemos que estos se encapsulan a través del guion bajo.

/**/

Ejemplo:

```
class A():
    def __init__(self):
        self._contador = 0
        self._cuenta = 0

    def incrementar (self):
        self._contador +=1

    def cuenta (self):
        return self._contador

a = A()
print(a.cuenta())
a.incrementar()
print(a.cuenta())
```

NOTAS:

• Para encapsular un atributo se utiliza el guion bajo, p. ej: **self.**_contador

• Es una MALA PRÁCTICA cuando se encapsula un atributo, escribir estas líneas de código:

```
print(a.cuenta)
```

a.incrementar

print(a.cuenta)

En el ejemplo, estas líneas están bien escritas. Ver los paréntesis.

Existen los métodos ***get*** y ***set*** para esto en POO.

17.3 Método Get:

En español, *get* significa *obtener*. Cuando deseamos obtener los valores de un atributo encapsulado debemos crear el método ***get*** por cada atributo. El método ***get*** se identifica el inicio de cada atributo a través de la palabra reservada @***property***.

/***/

Ejemplo:

```
class A():
    def __init__(self):
        self._cuenta = 0
        self._contador = 0

    @property
    def cuenta (self):
        return self._cuenta

    @property
    def contador (self):
        return self._contador
```

```
a = A()
print(a.cuenta)
print(a.contador)
```

NOTAS:

• **@property**. Implica la creación del ***método get*** para el atributo que se sucede, en este caso, “cuenta” y “contador”.

• Diferencias entre:

```
print(a.cuenta)
print(a.cuenta())
```

Ambas sintaxis son correctas. La diferencia radica cuando se usa una y cuando la otra. La primera, se utiliza SOLO cuando se encuentra la palabra reservada ***@property*** en el atributo, es decir, en POO cuando invocamos al ***método get***. Cabe aclarar que, en principio, el atributo era un ***método de instancia*** que, luego de colocar la palabra reservada ***@property*** es considerado como un atributo. La segunda, sólo se escribe cuando es un ***método de instancia*** o una función.

Las buenas prácticas en POO indican que la primera forma es la correcta para realizar dicha acción.

17.4 Método Set:

En español, *set* significa *colocar*. Esto permite ver o cambiar el valor de cada atributo. El método ***set*** se identifica al final de cada atributo a partir de la palabra reservada @<*atributo*>***.setter***.

Cuando un atributo sólo tiene el ***método get*** y NO el ***método set***, se les conoce como ***atributo ruidoneo***, es decir, sólo lectura.

Sintaxis:

```
@<atributo>.setter
def <atributo> (self, <atributo>):
    self._<atributo> = <atributo modificado>
```

/***/

Ejemplo:

```
class A():
    def __init__(self):
        self._cuenta = 0
        self._contador = 0

    @property
    def cuenta (self):
        return self._cuenta

    @cuenta.setter
    def cuenta (self, cuenta):
        self._cuenta = cuenta

    @property
    def contador (self):
        return self._contador

    @contador.setter
    def contador (self, contador):
        self._contador = contador
```

```
a = A()
print(a.cuenta)
a.cuenta = 20
print(a.cuenta)
print(a.contador)
a.contador = 10
print(a.contador)
```

NOTA:

• a.cuenta = 20. El valor del atributo cuenta ha sido cambiado por 20 (antes era 0). Sólo se puede hacer de esta manera en presencia del método ***set***.

17.5 Herencia:

Es tener dos o más clases y que unas hereden a otras, existen clases padres y clases hijas. Los métodos de la clase padre pasan a la clase hija sin ser colocados nuevamente.

/**/

Ejemplo:

```
class Animales():
    def habla(self):
        print("Yo soy un Animal")

    def descripcion(self):
```

```
        print("Yo soy un {}".format(self.animal))

class Perro(Animales):
    pass

class Abeja(Animales):
    def __init__(self, animal):
        self.animal = animal

animal = Animales()
animal.habla()

perro = Pero()
perro.habla()

abeja = Abeja("Abeja")
abeja.descripcion()
```

NOTAS:

• La manera de crear una **herencia** es de la siguiente manera:

class Perro(Animales)**:**

Es una **clase** denominada "Perro" que se encuentra dentro de la clase "Animales" (en su argumento).

• La palabra reservada ***pass*** le indica a Pythoon que no se va agregar nada dentro de la clase.

17.6 Más sobre Herencia:

Tipos de errores que se pueden producir por el uso de la ***herencia*** porque hay ciertos métodos que no se van a heredar completamente sino que hay que hacer ciertos procedimientos extras para que se hereden completamente, por ejemplo, sucede con el ***método init*** cuando se utiliza en varias clases.

Para lograr heredar completamente una clase, en estos casos, es necesario utilizar la palabra reservada ***super()***.

/**********/

Ejemplo – ***super()***:

```
class Animales():
    def __init__(self, nombre):
        self.nombre = nombre

class Perro (Animales):
    def __init__(self, nombre, sonido):
        super().__init__(nombre)
        self.sonido = sonido

perro = Perro ("Firulais", "Guuaaoo")
print(perro.nombre)
print(perro.sonido)
```

NOTAS:

• Herencia completa: Para que la clase hija disponga los mismos métodos y atributos que la clase padre, es necesario colocar las siguientes líneas:

def **__init__**(**self**, nombre, sonido)**:**. En esta, se agrega el atributo “nombre” de la clase padre.

super().__init__(nombre). Se utiliza la palabra reservada y se convoca el ***init*** de la clase padre y el/los atributo/s de dicha clase.

• Los dos últimos ***print*** NO obedecen a la buena práctica de Python, solo se utilizaron para imprimir el mensaje.

17.7 Herencia múltiple:

Una herencia múltiple es una herencia con más de una clase que se van a heredar entre sí o a una tercera.

/***/

Ejemplo:

```
class A():
    def primera (self):
        return “Esta es la clase A”

class B():
    def segunda(self):
        return “Esta es la clase B”

class C(A, B):
    pass

c = C()
print(c.primera())
```

print(c.segunda())

NOTA: Nótese que la clase C hereda los métodos de las clases A y B; esto se indica por medio de esta notación: **class** C(A, B)**:**

17.8 Polimorfismo:

El polimorfismo es la modificación de métodos cuando se heredan de otras clases o crear objetos que apunten o usen el mismo método pero que el objeto sea diferente.

/***/

Ejemplo:

```
class Animales():
    def __init__(self, mensaje):
        self.mensaje = mensaje

    def hablar(self):
        print(self.mensaje)

class Perro(Animales):
    def hablar(self):
        print("Yo hago guau!")

class Pez(Animales):
    def hablar(self):
        print("Yo no hablo")
```

```
perro = Perro(“Guau!”)
perro.hablar()

animal = Animales(“Miau!”)
animal.hablar()

pez = Pez(“Nada”)
pez.hablar()
```

NOTA: Obsérvese que, a través de la técnica de polimorfismo, se ha cambiado el mensaje de las clases cuando originalmente, por medio de la técnica ***herencia***, debería otra cosa. Por ejemplo, en la clase “Perro” se debería imprimir “Guau!” dado el objeto creado, sin embargo, la clase ha sido particularizada y el mensaje se ha cambiado a “Yo hago guau!”.

18. SECCIÓN 18: EJERCICIOS DE POO

18.1 Ejercicio 1:

Realizar un programa que consta de una clase llamada Estudiante, que tenga como atributos el nombre y la nota del alumno. Definir los métodos para inicializar sus atributos, imprimirlos y mostrar un mensaje con el resultado de la nota y si ha aprobado o no.

```
/*****************************************************************************/
```

Ejercicio – por mí:

```
class Estudiante():
    def __init__(self, nombre, nota):
        self.nombre = nombre
        self.nota = nota

    def resultado(self):
        if self.nota >= 7:
            return "El alumno ha aprobado"
        else:
            return "El alumno ha desaprobado"

alumno = Estudiante("Pedro", 9)
print(alumno.nombre)
print(alumno.nota)
print(alumno.resultado())
```

Ejemplo 1 – con ***init***:

```
class Estudiante():
    def __init__(self, nombre, nota):
        self.nombre = nombre
        self.nota = nota

    def imprimir(self):
        print("Nombre: {} \nNota: {}".format(self.nombre, self.nota))

    def resultado(self):
        if self.nota < 7:
            print "Has REPROBADO!"
        else:
            print "Has APROBADO!"

estudiante1 = Estudiante("Daniel", 10)
estudiante1.imprimir()
estudiante1.resultado()

estudiante2 = Estudiante("Karla", 5)
estudiante2.imprimir()
estudiante2.resultado()
```

Ejemplo 2 – con método de instancia:

```
class Estudiante():
```

```
    def datos(self, nombre, nota):
        self.nombre = nombre
        self.nota = nota

    def imprimir(self):
        print("Nombre: {} \nNota: {}".format(self.nombre, self.nota))

    def resultado(self):
        if self.nota < 7:
            print "Has REPROBADO!"
        else:
            print "Has APROBADO!"

estudiante1 = Estudiante()
estudiante1.datos("Daniel", 10)
estudiante1.imprimir()
estudiante1.resultado()
```

CONCLUSIÓN: El ejemplo 1 resulta más optimo que el ejemplo 2 dado que ocupa menos líneas de código.

18.2 <u>Ejercicio 2</u>:

Realizar un programa en el cual se delaren dos valores enteros por teclado utilizando el método __**init**__. Calcular después la suma, resta, multiplicación y división. Utilizar un método para cada una e imprimir los resultados obtenidos. Llamar a la clase Calculadora.

```
/*****************************************************************************/
```

Ejercicio – por mí:

```
class Calculadora():
    def __init__ (self, x, y):
        self.x = x
        self.y = y

    def sumar(self):
        self.suma = self.x + self.y
        print(La suma es igual a: “, self.suma)

    def restar(self):
        self.resta = self.x - self.y
        print(La resta es igual a: “, self.resta)

    def multiplicar(self):
        self.mult = self.x * self.y
        print(La multiplicacion es igual a: “, self.mult)

    def dividir(self):
        self.division = self.x / self.y
        print(La division es igual a: “, self.division)

v1 = int(input(“Ingrese el primer valor: “))
v2 = int(input(“Ingrese el segundo valor: “))

calcular = Calculadora(v1, v2)
print(calcular.sumar())
```

```
print(calcular.restar())
print(calcular.multiplicar())
print(calcular.dividir())
```

Ejemplo:

```
class Calculadora():
    def __init__ (self):
        self.x = int(input("Ingrese el primer valor: "))
        self.y = int(input("Ingrese el segundo valor: "))

    def sumar(self):
        self.suma = self.x + self.y
        print(La suma es igual a: ", self.suma)

    def restar(self):
        self.resta = self.x - self.y
        print(La resta es igual a: ", self.resta)

    def multiplicar(self):
        self.mult = self.x * self.y
        print(La multiplicacion es igual a: ", self.mult)

    def dividir(self):
        self.division = self.x / self.y
        print(La division es igual a: ", self.division)
```

```
calcular = Calculadora()
print(calcular.sumar())
print(calcular.restar())
print(calcular.multiplicar())
print(calcular.dividir())
```

NOTA: Nótese que es más óptimo el código del Ejemplo.

18.3 Ejercicio 3:

Crear una clase Fabrica que tenga los atributos de Llantas, Color y Precio; luego crear dos clases más que hereden de Fabrica, las cuales son Moto y Carro. Crear los métodos que muestren la cantidad de llantas, color y precio de ambos transportes. Por último, crear objetos para cada clase y mostrar por pantalla los atributos de cada uno.

```
/*****************************************************************************/
Ejercicio – por mí:
class Fabrica():
    def __init__(self, llantas, color, precio):
        self.llantas = llantas
        self.color = color
        self. precio = precio

class Moto(Fabrica):
    def Llantas(self):
        print("Llantas: {}".format(self.llantas))
    def Color(self):
        print("Color: {}".format(self.color))
```

```
        def Precio(self):
                print("Precio: {}".format(self.precio))

class Carro(Fabrica):
        def Llantas(self):
                print("Llantas: {}".format(self.llantas))
        def Color(self):
                print("Color: {}".format(self.color))
        def Precio(self):
                print("Precio: {}".format(self.precio))

moto = Moto(2, "Negro", 22000)
print("La moto tiene las siguientes características: ")
moto.Llantas()
moto.Color()
moto.Precio()

carro = Carro(4, "Gris", 3000000)
print("El carro tiene las siguientes características: ")
carro.Llantas()
carro.Color()
carro.Precio()
```

Ejemplo:

```
class Fabrica():
        def __init__(self, llantas, color, precio):
```

```
            self.llantas = llantas
            self.color = color
            self. precio = precio

class Moto(Fabrica):
    def Datos(self):
            print("Llantas: {}".format(self.llantas))
            print("Color: {}".format(self.color))
            print("Precio: ${}".format(self.precio))

class Carro(Fabrica):
    def Datos(self):
            print("Llantas: {}".format(self.llantas))
            print("Color: {}".format(self.color))
            print("Precio: ${}".format(self.precio))

moto = Moto(2, "Negro", 4000)
moto.Datos()

carro = Carro(4, "Blanco", 5000)
carro.Datos()
```

NOTA: Nótese que es más óptimo el código del Ejemplo.

18.4 <u>Ejercicio 4</u>:

Crear una clase llamada Marino(), con un método hablar, en donde muestre un mensaje que diga “Hola…”. Luego, crear una clase Pulpo() que herede Marino, pero modificar el mensaje de hablar por “Soy un Pulpo”. Por último, crear una clase Foca(), heredada de Marino, pero que tenga un atributo nuevo llamado mensaje y que muestre ese mensaje como parámetro.

/**********/

Ejercicio – por mí:

```
class Marino():
    def __init__(self, mensaje):
        self.mensaje = mensaje

    def hablar (self):
        print(self.mensaje)

class Pulpo():
    def hablar(self):
        print(“Yo soy un Pulpo”)

class Foca():
    def hablar(self):
        print(self.mensaje)

marino = Marino(“Hola…”)
marino.hablar()

pulpo = Pulpo(“Hola…”)
pulpo.hablar()
```

```
foca = Foca("Yo soy una foca")
foca.hablar()
```

Ejemplo:

```
class Marino():
    def hablar (self):
        print("Hola…")

class Pulpo():
    def hablar(self):
        print("Soy un Pulpo")

class Foca():
    def hablar(self, mensaje):
        self.mensaje = mensaje
        print(self.mensaje)

marino = Marino()
marino.hablar()

pulpo = Pulpo()
pulpo.hablar()

foca = Foca()
foca.hablar("Hola, soy una foca")
```

18.5 Ejercicio 5:

Crear un programa con tres clases Universidad, con atributos nombre (Donde se almacena el nombre de la Universidad). Otra llamada Carrera, con los atributos especialidad (En donde me guarda la especialidad de un estudiante). Una última llamada Estudiante, que tenga como atributos su nombre y edad. El programa debe imprimir la especialidad, edad, nombre y universidad de dicho estudiante con un objeto llamado persona.

/**********/

Ejercicio – por mí:

Ejemplo:

```
class Universidad ():
    def __init__(self, Nombre):
        self.Nombre = Nombre

class Carrera():
    def carrera(self, especialidad):
        self.especialidad = especialidad

class Estudiante(Universidad, Carrera):
    def datos(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
```

```
        print("Mi nombre es {}, tengo {} anios, mi especialidad {} en
{}".format(self.nombre, self.edad, self.especialidad, self.Nombre))



persona = Estudiante("Don Bosco")

persona.carrera("Sistemas")

persona.datos("Carlos", 20)
```

NOTA: “Don Bosco se ha puesto en el argumento de Estudiante dado que el método constructor ***init*** sólo se encuentra en la clase Universidad donde requiere un argumento.

19. SECCIÓN 19: HTML

19.1 HTML (Teoría):

HTML por sus siglas significa ***HyperText Markup Languaje*** (*Lenguaje de Marcado de Hipertexto*), NO representa un lenguaje de programación dado que este no ocupa variables, en lugar de variables, va a tener ***viñetas***.

Un lenguaje de marcados usa ***etiquetas*** para identificar su contenido; esto es HTML; son viñetas que identifican el contenido de una página web.

Historia:

en 1989, Tim Burners Lee, logró ocupar el protocolo SPIP para enlazar documentos electrónicos entre dos dispositivos. También fue el creador de ***www*** (***World Wide Web***).

Estructura Web:

19.2 Conceptos Importantes (Teoría):

Etiquetas o **Viñetas:**

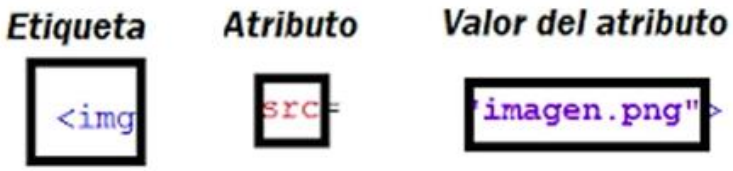
Es un fragmento que indica al navegador realizar una función o una tarea en específico. Estas etiquetas determinan la estructura y la funcionalidad que va a tener la página web.

Una estructura básica podemos ver a continuación:

Atributo:

Cualidad que puede agregarse a una etiqueta. Su función es modificar el contenido de la etiqueta o agregar mejor estructura a una página web.

La estructura de un atributo es la siguiente:



Nombre de la etiqueta + Atributo + Valor del atributo.

Estructura de una viñeta:

Etiqueta de Apertura	*Contenido*	*Etiqueta de Cierre*
<p class=”fondo”**>**	Texto del contenido	**<\p>**

<p : Nombre

class=”fondo : Atributo

Contenedores:

Son unos recuadros o espacios rectangulares que pueden colocarse en cualquier parte de la página. En ellas, podemos insertar contenido HTML. Su utilidad principal es visual, permitiendo organizar y dar estructura y diseño a las páginas HTML.

Dentro de cada contenedor hay más herramientas disponibles para ser utilizadas.

19.3 <u>Estructura Básica HTML</u>:

La estructura básica de una página web se muestra en el ejemplo.

Viñetas html:

Su función es definir una función para que el navegador la lea, y a su vez, cumpla dicha función y se la muestre al usuario.

<html>

<\html>

Cabecera (**head**)**:**

Se almacena información que se le da al navegador pero que no se le muestra al usuario. Dentro del ***head*** se almacena toda documentación necesaria para que el navegador lea el documento completo y, a su vez, pueda mostrar al usuario.

<head>

<\head>

Cuerpo (**body**)**:**

Se va almacenar toda documentación que se le va a mostrar al usuario.

<body>

<\body>

Título (**title**)**:**

Permite cambiar el título a la página web.

<title>”Titulo”<\title>

/**********/

Ejercicio:

<html>

<head>

<title>Primera Página Web**<\title>**

<\head>

<body>

Linea de texto para verificar

<\body>

<\html>

19.4 Viñeta <p> para Párrafos:

Permite reconocer un texto. Va dentro de la viñeta ***body***.

/**********/

Ejercicio:

<html>

<head>

<title>Primera Página Web**<\title>**

<\head>

<body>

<p>Primer ejemplo de párrafos**<\p>**

<p>Segundo ejemplo de párrafos**<\p>**

<p>Lorem….asdklajsdlajsdlasjdlajdsaljsaldjasdjsdjias

kasjdkasjdklasjdklajsdklasxmlksamxksamdksamdkad klsajdklsajdksajdklsajdklas**
mklxsamxksamxkmsalxkls sadkasldkñsadñl,x<\p>**

<\body>

<\html>

NOTAS:

• ** **Lorem….** **. Arroja un texto por defecto para probarlo en la pagina web.

• **
**. Provoca un salto de línea en el texto.

19.5 <u>Viñeta de Encabezados y Títulos</u>:

La diferencia es el tamaño del texto. Las viñetas de los encabezados se simbolizan como **<h1></h1>** y van hasta el **<h6></h6>**, éstos representan los títulos del encabezado; a medida que su numeración crece, decrece el tamaño del título.

/**/

<u>Ejercicio</u>:

```
<html>
    <head>
        <title>Títulos<\title>
    <\head>
    <body>
        <h1>Titulo h1<\h1>
        <h2>Titulo h2<\h2>
        <h3>Titulo h3<\h3>
        <h4>Titulo h4<\h4>
        <h5>Titulo h5<\h5>
        <h6>Titulo h6<\h6>
    <\body>
<\html>
```

19.6 Viñetas para colocar Formato al Texto:

Algunas viñetas utilizadas en html (NO html5) para dar formato al texto son:

** **. Da color negrita al párrafo.

** **. Pone al párrafo en forma cursiva.

<big> </big>. Incrementa el tamaño del párrafo.

<small> </small>. Decrementa el tamaño del párrafo.

** **. Da color negro al párrafo.

****. Pone en subíndice al párrafo.

****. Pone en superíndice al párrafo.

<ins> </ins>. Subraya el párrafo.

<del> </del>. Tacha el párrafo.

/**********/

Ejercicio:

```
<html>
    <head>
        <title>Formate para Textos<\title>
    <\head>
    <body>
        <p>Texto de ejemplo<\p>
        <p><b>Texto de ejemplo</b><\p>
        <p><i>Texto de ejemplo</i><\p>
        <p><big>Texto de ejemplo</big><\p>
        <p><small>Texto de ejemplo</small><\p>
        <p><strong>Texto de ejemplo</strong><\p>
        <p><sub>Texto de ejemplo</sub><\p>
```

<p>^{**Texto de ejemplo**}<\p>

<p><ins>Texto de ejemplo**</ins><\p>**

<p>~~Texto de ejemplo**~~<\p>**

<\body>

<\html>

NOTA: Estas viñetas solo se han aplicado en los párrafos.

19.7 Viñetas <img> para Imágenes:

Lo recomendable para utilizar imágenes en páginas web es disponer estas fotos en la carpeta del archivo.

Sintaxis:

• Si se encuentra en la misma carpeta que el archivo html:

****

• Si se encuentra en la misma carpeta que el archivo html:

****

Si la dirección de la imagen no coincide con su dirección real, la imagen no se podrá ver en la página web.

Atributos en imágenes:

Los atributos son parámetros que, en este caso, van a modificar la imagen. Algunos de ellos:

width = “150px”. Ancho de la imagen

height = “150px”. Alto de la imagen

/**/

Ejercicio:

<html>

<head>

<title>Imagenes**<\title>**

<\head>

<body>

****

<p>Este es un lenguaje de viñetas llamado HTML**<\p>**

<\body>

<\html>

19.8 Viñetas <a> para Enlaces:

Permite redireccionar una página web a través de un enlace.

Sintaxis:

****Google****

El atributto **target=**”_blank” permite redireccionar la página web a una pestaña diferente.

/**/

Ejercicio:

```
<html>
    <head>
        <title>Imagenes<\title>
    <\head>
    <body>
        < a target=”_blank”
        href="https://www.google.com">Google</a>
    <\body>
<\html>
```

19.9 Viñetas <ol> para Listas Ordenadas:

Hay tres tipos de listas: las listas ordenadas, las listas desordenadas y las listas de definición.

Sintaxis:

```
<ol start=”10”>
    <li>”Elemento 1 de la lista” </li>
    <li>”Elemento 2 de la lista” </li>
    <li>”Elemento 3 de la lista” </li>
</ol>
```

**
**. Permite definir un elemento de la lista, p.e: Papas.

Atributos:

El atributo **start=**”10” permite comenzar a la lista a partir del número 10.

El atributo **type=**”A” permite comenzar a la lista a partir de la letra A.

El atributo **type=**”a” permite comenzar a la lista a partir de la letra a.

El atributo **type=**”I” permite comenzar a la lista con el numero romano I.

El atributo **type=**”i” permite comenzar a la lista con el numero romano i.

El atributo **reversed** permite comenzar a la lista en forma inversa.

/**/

Ejercicio:

```
<html>
    <head>
        <title>Imagenes<\title>
    <\head>
    <body>
        < a target=”_blank”
        href="https://www.google.com">Google</a>
    <\body>
<\html>
```

19.10 Viñetas <ul> para Listas Desordenadas:

Sintaxis:

```
<ul>
    <li>”Elemento 1 de la lista” </li>
    <li>”Elemento 2 de la lista” </li>
    <li>”Elemento 3 de la lista” </li>
</ul>
```

**
**. Permite definir un elemento de la lista, p.e: Papas.

Listas Anidadas:

```
<ol>
    <li>Animales</li>
    <ul>
        <li>Gallina</li>
        <li>Serpiente</li>
        <li>Perro</li>
    </ul>
    <li>Paises</li>
    <ul>
        <li>Guatemala</li>
        <li>El Salvador</li>
        <li>Honduras</li>
    </ul>
</ol>
```

Nótese que permite combinar el tipo de Listas Ordenadas con Listas Desordenadas.

/***/

Ejemplo:

```
<html>
  <head>
    <title>Listas Desordenadas</title>
  </head>
  <body>
```

```
        <ol>
            <li>Animales</li>
            <ul>
                <li>Gallina</li>
                <li>Serpiente</li>
                <li>Perro</li>
                <li>Cocodrilo</li>
            </ul>
            <li>Paises</li>
            <ul>
                <li>Guatemala</li>
                <li>El salvador</li>
                <li>Honduras</li>
                <li>Costa Rica</li>
                <li>Panama</li>
            </ul>
        </ol>
    </body>
</html>
```

19.11 Viñetas <dl> para Listas de Definiciones:

Es como un diccionario, tiene una palabra y su definición.

Sintaxis:

```
<dl>
        <dt>”Palabra 1”</dt>
```

<dd>”Definición Palabra 1”**</dd>**

<dt>”Palabra 2”**</dt>**

<dd>”Definición Palabra 2”**</dd>**

</dl>

<dt></dt>. Permite crear una palabra.

<dd></dd>. Permite definir la palabra creada.

/**/

Ejemplo:

```
<html>
   <head>
      <title>Lista de Definiciones</title>
   </head>
   <body>
      <dl>
         <dt>Rojo</dt>
         <dd>Color básico</dd>
         <dt>Aguila</dt>
         <dd>Animal tipo de ave</dd>
      </dl>
   </body>
</html>
```

19.12 Viñetas <table> para Crear Tablas:

Es como un diccionario, tiene una palabra y su definición.

Sintaxis:

```
<table border="1" align="center">
   <tr bgcolor="blue">
      <td>Animales</td>
      <td>Perro</td>
      <td>Gato</td>
   </tr>
</table>
```

<table></table>. Permite crear una tabla.

<tr></tr>. Permite ubicar un elemento en una columna.

<td></td>. Permite ubicar un elemento en una fila.

Atributos:

• **border=**"1". Permite dar bordes a toda la tabla.

• **align=**"center". Permite alinear la tabla en el centro de la página.

• **bgcolor=**"blue". Pinta una celda, fila o columna de azul.

/**********/

Ejemplo:

```
<html>
   <head>
      <title>Tablas</title>
   </head>
   <body>
```

```
        <table border="1" align="center">
            <tr bgcolor="blue">
                <td>Animales</td>
                <td>Perro</td>
                <td>Gato</td>
            </tr>
            <tr>
                <td>Plantas</td>
                <td>Girasoles</td>
                <td>Rosas</td>
            </tr>
            <tr>
                <td>Comida</td>
                <td>Carne</td>
                <td>Vegetales</td>
            </tr>
        </table>
    </body>
</html>
```

19.13 Viñetas <form> para Crear Formularios:

Sintaxis:

```
        <form action="">
            <p>Selecciona tu sexo:</p>
            <input type="radio" name="respuesta" />Hombre
```

```
        <input type="radio" name="respuesta"/>Mujer
        <br/>
        <p>Seleccione un numero: </p>
        <input type="checkbox">1
        <input type="checkbox">2
        <br/>
        <input type="submit" value="enviar">
        <br/>
        <br/>
        <p>Usuario: </p>
        <input type="text">
        <p>Contraseña</p>
        <input type="Password">
    </form>
```

Atributos:

• **type=**"radio". Permite crear círculos de selección simple para la respuesta.

• **type=**"checkbox". Permite crear cuadrados de selección múltiple para la respuesta.

• **name=**"respuesta". Permite seleccionar la respuesta.

/**/

Ejemplo:

```
<html>
  <head>
    <title>Formularios</title>
  </head>
```

```
    <body>
        <form action="">
            <p>Selecciona tu sexo:</p>
            <input type="radio" name="respuesta" />Hombre
            <input type="radio" name="respuesta"/>Mujer
            <br/>
            <p>Seleccione un numero: </p>
            <input type="checkbox">1
            <input type="checkbox">2
            <br/>
            <input type="submit" value="enviar">
            <br/>
            <br/>
            <p>Usuario: </p>
            <input type="text">
            <p>Contraseña</p>
            <input type="Password">
        </form>
    </body>
</html>
```

19.14 Contenedores en HTML:

Un contenedor es una viñeta que, a simple vista no modifica nada. pero sirve para dar estilos o más funcionalidad a la página web.

Existen dos tipos: Contenedores de Líneas, aquellos que una viñeta mantiene una misma línea y Contenedores en Bloques, aquellas viñetas que salta de línea.

/***/

Ejemplo:

```
<html>
   <head>
      <title>Contenedor</title>
   </head>
   <body>
      <div style="color: red;">
      <p>Este es un ejemplo de contendor de bloque</p>
      <p>Este es un ejemplo <b>de salto de bloque</b></p>
      </div>
       <p style="color: yellow;">Este es un ejemplo fuera del bloque
      div</p>
   </body>
</html>
```

20. SECCIÓN 20: HTML5

20.1 HTML5 (Teoría):

Es la quinta versión del Lenguaje de Viñetas. Defino los nuevos estándares de desarrollo web, rediseñando el código para resolver problemas y actualizándolo así a nuevas necesidades. No se limita a crear nuevas herramientas o atributos sino que incorpora nuevas características y, a su vez, proporciona plataformas de desarrollo denominadas ***APIs*** para hacer aplicaciones web.

Nuevas características:

• Nuevas etiquetas semánticas para estructurar los documentos HTML, destinadas a reemplazar la necesidad de tener una etiqueta que identifica cada bloque de la página.

• Los nuevos elementos multimedio como ***<audio>*** y ***<video>***.

• APIs nuevas.

• Almacenamiento local en el lado del cliente, es decir, no es necesario disponer un servidor web o *hosting*.

Estructuras:

<header>

<nav>

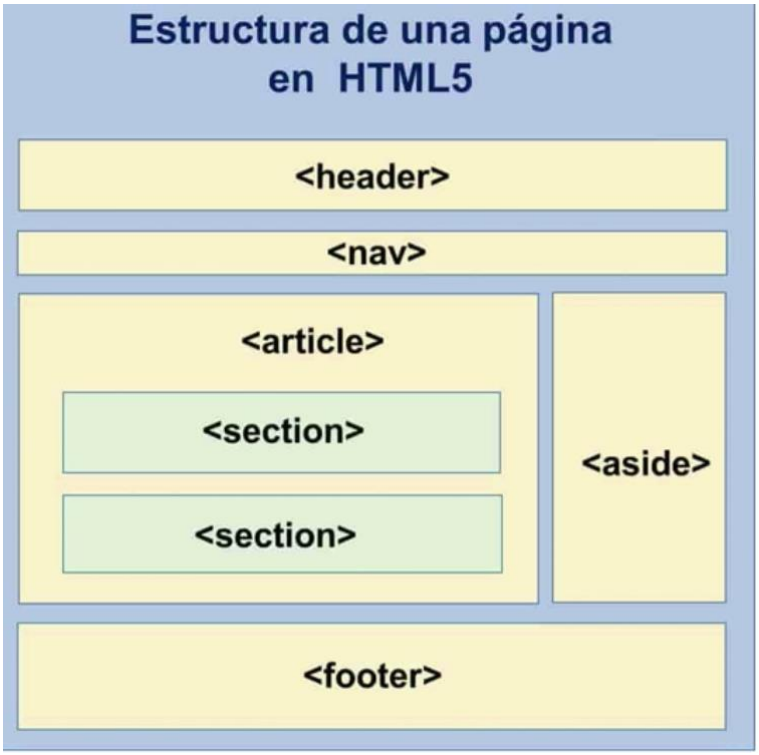
<article>

<section>

<aside>

<section>

<footer>



<header>. Algo parecido a la Cabecera. Se almacena lo principal de la página web, es decir, aquello que la distingue de lo demás.

<nav>. Algo parecido a la navegación.

<article>. Espacio que dispone de secciones o **<section>** que son contenidos extras que se le puede colocar a la página web.

<aside>. Algo parecido a un contenido que se encuentra al borde de la página web.

<footer>. Es el pie de página. Proporciona un espacio de contenido al pie de página.

20.2 Header:

Ejemplo – HTML5:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Header</title>
    <link rel="stylesheet" href="Style.css">
</head>
<header>
    <h1>Header</h1>
</header>
<body>
</body>
</html>
```

NOTAS:

• **<link rel=**"stylesheet" **href=**"Style.css"**>**. Permite vincular un archivo HTML con un archivo.css cuya dirección **href** es solicitada por el Lenguaje de Viñetas.

• **<header></header>**. Se coloca a continuación del **head** antes el **body**.

Ejemplo – CSS:

```
header{
    background-color: gray; #Color de Fondo
    border: 5px chartreuse; #Bordes
    width: 770px; #Ancho
    height: 200px; #Altura
    margin: 5px; #Margen
    border-radius: 10px; #Redondeo en el fondo.
```

}

NOTAS:

• Nótese que representa un formato de estilos al **header** del archivo HTML.

20.3 Main:

Es la parte más importante dentro del **body**. Dentro del **main**, va la parte más importante de la página web.

/***/

Ejemplo – HTML5:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Header</title>
   <link rel="stylesheet" href="Style.css">
</head>
<header>
   <h1>Header</h1>
</header>
<body>
```

```
        <main>
        </main>
</body>
</html>
```

NOTAS:

• **<main></main>**. Se coloca dentro del **body**.

Ejemplo – CSS:

```
header{
   background-color: gray; #Color de Fondo
   border: 5px chartreuse; #Bordes
   width: 770px; #Ancho
   height: 200px; #Altura
   margin: 5px; #Margen
   border-radius: 10px; #Redondeo en el fondo.
}

main{
   width: 780px;
   height: 400px;
}
```

NOTAS:

• Nótese que representa un formato de estilos al **main** del archivo HTML.

20.4 Nav:

El **nav** como tal, contiene enlaces o links hacia otros lugares ya sea dentro o fuera de la página web. En él se almacenan *inicio*, *contactos*, *entradas*, etc. típicas de ver en un sitio web.

En dispositivos móviles, al ser la pantalla más pequeña, se puede utilizar como medio de navegación para mejorar la experiencia del usuario. Por último, el **nav** viene a sustituir al **div** que se utilizaba en HTML.

/***/

Ejemplo – HTML5:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Header</title>
   <link rel="stylesheet" href="Style.css">
</head>
<header>
   <h1>Header</h1>
</header>
<body>
      <main>
```

```
            <nav>
                <h2>Nav</h2>
            </nav>
        </main>
</body>
</html>
```

NOTAS:

• **<nav></nav>**. Se coloca dentro del **main**.

Ejemplo – CSS:

```
header{
   background-color: gray; #Color de Fondo
   border: 5px chartreuse; #Bordes
   width: 770px; #Ancho
   height: 200px; #Altura
   margin: 5px; #Margen
   border-radius: 10px; #Redondeo en el fondo.
}

main{
   width: 780px;
   height: 400px;
}

nav{
```

```
    background-color: gray;
    width: 150px;
    height: 390px;
    float: left;
    margin: 5px;
    border-radius: 10px;
}
```

20.4 Section y Article:

El **section** es una sección, se encarga de agrupar elementos por contenido. Por lo general tiene diferentes características, la primera, divide un listado por noticias, información de contacto, etc., asegura que todo tenga relación y, por último, se utiliza para contenidos independientes dentro de un documento, es decir, con contenidos que no tengan que enlazarse con nada fuera del **section**, p.e., el contenido del **section** no tiene por qué enlazarse con el contenido del **nav**.

Un **article** es un artículo dentro de una **section**.

/***/

Ejemplo – HTML5:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Header</title>
```

```
    <link rel="stylesheet" href="Style.css">
</head>
<header>
    <h1>Header</h1>
</header>
<body>
        <main>
            <nav>
                <h2>Nav</h2>
            </nav>
            <section>
                <article id=”primero”>Noticia 1</article>
                <article id=”segundo”>Noticia 2</article>
                <article id=”tercero”>Noticia 3</article>
            </section>
        </main>
</body>
</html>
```

Ejemplo – CSS:

```
header{
    background-color: gray; #Color de Fondo
    border: 5px chartreuse; #Bordes
    width: 770px; #Ancho
    height: 200px; #Altura
    margin: 5px; #Margen
    border-radius: 10px; #Redondeo en el fondo.
```

```
}

main{
    width: 780px;
    height: 400px;
}

nav{
    background-color: gray;
    width: 150px;
    height: 390px;
    float: left;
    margin: 5px;
    border-radius: 10px;
}

section{
    background-color: gray;
    width: 450px;
    height: 390px;
    float: left; #Disposición de section dentro de la página web.
    margin: 5px;
    border-radius: 10px;
}

#primero{ #Es la manera de modificar el estilo a las viñetas article.
    background-color: burlywood;
```

width: 440px**;**

height: 120px**;**

float: left**;**

margin: 5px**;**

}

#segundo**{** *#Es la manera de modificar el estilo a las viñetas* ***article***.

background-color: burlywood**;**

width: 440px**;**

height: 120px**;**

float: left**;**

margin: 5px**;**

}

#tercero**{** *#Es la manera de modificar el estilo a las viñetas* ***article***.

background-color: burlywood**;**

width: 440px**;**

height: 120px**;**

float: left**;**

margin: 5px**;**

}

20.6 Aside:

Es lo mismo que un **section** pero el **aside** va a un costado de la página web. Generalmente, en el *aside* se colocan publicidad, noticias, entre otras cosas. Un *aside* también podría ser el chat que utilizaba *Facebook* donde mostraba a los amigos.

/***/

Ejemplo – HTML5:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Header</title>
   <link rel="stylesheet" href="Style.css">
</head>
<header>
   <h1>Header</h1>
</header>
<body>
        <main>
                <nav>
                        <h2>Nav</h2>
                </nav>
                <section>
                        <article id=”primero”>Noticia 1</article>
                        <article id=”segundo”>Noticia 2</article>
                        <article id=”tercero”>Noticia 3</article>
                </section>
                <aside><h2>Aside</h2></aside>
```

```
        </main>
</body>
</html>
```

Ejemplo – CSS:

```
header{
    background-color: gray; #Color de Fondo
    border: 5px chartreuse; #Bordes
    width: 770px; #Ancho
    height: 200px; #Altura
    margin: 5px; #Margen
    border-radius: 10px; #Redondeo en el fondo.
}

main{
    width: 780px;
    height: 400px;
}

nav{
    background-color: gray;
    width: 150px;
    height: 390px;
    float: left;
    margin: 5px;
    border-radius: 10px;
}
```

```
section{
    background-color: gray;
    width: 450px;
    height: 390px;
    float: left; #Disposición de section dentro de la página web.
    margin: 5px;
    border-radius: 10px;
}

#primero{ #Es la manera de modificar el estilo a las viñetas article.
    background-color: burlywood;
    width: 440px;
    height: 120px;
    float: left;
    margin: 5px;
}

#segundo{ #Es la manera de modificar el estilo a las viñetas article.
    background-color: burlywood;
    width: 440px;
    height: 120px;
    float: left;
    margin: 5px;
}

#tercero{ #Es la manera de modificar el estilo a las viñetas article.
```

```
    background-color: burlywood;
    width: 440px;
    height: 120px;
    float: left;
    margin: 5px;
}

aside{
    background-color: gray;
    width: 150px;
    height: 390px;
    float: left;
    margin: 5px;
    border-radius: 10px;
}
```

20.7 Footer:

Es el pie de página del sitio web. En él se pueden almacenar: nombre del autor, enlaces relacionados con el documento, copyright, sitios de contactos, etc.

Se coloca fuera del **body**.

```
/*******************************************************************************/
```

Ejemplo – HTML5:

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Header</title>
    <link rel="stylesheet" href="Style.css">
</head>
<header>
    <h1>Header</h1>
</header>
<body>
        <main>
                <nav>
                        <h2>Nav</h2>
                </nav>
                <section>
                        <article id=”primero”>Noticia 1</article>
                        <article id=”segundo”>Noticia 2</article>
                        <article id=”tercero”>Noticia 3</article>
                </section>
                <aside><h2>Aside</h2></aside>
        </main>
</body>
<footer><h2>Pie de página</h2></footer>
</html>
```

Ejemplo – CSS:

```
header{
    background-color: gray; #Color de Fondo
    border: 5px chartreuse; #Bordes
    width: 770px; #Ancho
    height: 200px; #Altura
    margin: 5px; #Margen
    border-radius: 10px; #Redondeo en el fondo.
}

main{
    width: 780px;
    height: 400px;
}

nav{
    background-color: gray;
    width: 150px;
    height: 390px;
    float: left;
    margin: 5px;
    border-radius: 10px;
}

section{
    background-color: gray;
    width: 450px;
```

```
    height: 390px;
    float: left; #Disposición de section dentro de la página web.
    margin: 5px;
    border-radius: 10px;
}

#primero{ #Es la manera de modificar el estilo a las viñetas article.
    background-color: burlywood;
    width: 440px;
    height: 120px;
    float: left;
    margin: 5px;
}

#segundo{ #Es la manera de modificar el estilo a las viñetas article.
    background-color: burlywood;
    width: 440px;
    height: 120px;
    float: left;
    margin: 5px;
}

#tercero{ #Es la manera de modificar el estilo a las viñetas article.
    background-color: burlywood;
    width: 440px;
    height: 120px;
    float: left;
```

```
    margin: 5px;
}

aside{
    background-color: gray;
    width: 150px;
    height: 390px;
    float: left;
    margin: 5px;
    border-radius: 10px;
}

footer{
    background-color: gray;
    width: 770px;
    height: 50px;
    margin: 5px;
    border-radius: 10px;
}
```

21. SECCIÓN 21: ATRIBUTOS HTML5

Un atributo es una cualidad o característica que cumple una función o, a su vez, un método.

21.1 Atributo Style:

Este atributo tiene mucha analogía con el Style que definimos previamente en CSS aunque más adelante profundizaremos en el tema.

/***/

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
</head>
<body>
   <h1 style="color: green;">Título</h1>
   <p style="font-size: 30px;">Lorem… </p>
   <a href=https://www.google.com target="_blank" style="background-color: grey;" >Google</a>
</body>
</html>
```

NOTA: Nótese como **h1**, **p** y **a** es afectado con el atributo **style**.

21.2 Atributo Class:

Este atributo me permite identificar la clase, por ejemplo, si tengo dos párrafos **p**, con el atributo **class**, puedo identificar cada uno de ellos (p1 y p2) para agregarle estilos.

/***/

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Class</title>
   <style>
      .primero{color: red;}
      .segundo{color: greenyellow;}
   </style>
</head>
<body>
   <p class="primero">Lorem… </p>
   <p class="segundo">Lorem... </p>
</body>
</html>
```

NOTA: Nótese como se utiliza el atributo **class** en los párrafos **p**.

21.3 Atributo ID:

Este atributo tiene muchas similitudes con el atributo **class**. El atributo **ID**, a diferencia del **class**, sobresalta el texto que lleve este atributo por sobre los demás, es decir, la más importante. Además, permite trabajar con otros lenguajes como *JavaScript* o *PHP*. Otra funcionalidad que tiene este atributo es que, cuando, por ejemplo, un contenido está fuera de mi visión en la página web, haciendo click sobre el elemento citado en el **ID** (en la página web), me redirecciona a este contenido.

/***/

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>ID</title>
   <style>
      #primero{ color: green;}
   </style>
</head>
<body>
   <a href=#tercero>Parrafo 3</a>
   <h1>Titulo de la pagina</h1>
   <h2 id="primero">Subtitulo 1</h2>
```

<h2 class="segundo">Subtitulo 2</h2>

Lorem…


```
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <br>
    <h2 id="tercero">Subtitulo 3</h2>
    <p class="segun">Lorem...</p>
</body>
</html>
```

NOTAS: • Nótese como se utiliza el atributo **ID** en el subtitulo 1 **h1** y la manera de convocarlo cuando quiero darle estilos.

• Obsérvese que, en el subtitulo 3 **h2**, el atributo **ID**, se utiliza para redireccionar dentro de la página web, el contenido de dicho subtitulo.

21.4 Atributo Accesskey:

Este atributo me permite crear combinaciones de teclas o “atajos” en mi página web. Este atributo **accesskey**, se encuentra dentro de las etiquetas **a** que define los enlaces.

/***/

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Accesskey</title>
</head>
<body>
   <a href="../21.3 Atributo ID/Atributo ID.html" target="_blank" accesskey="h">Referencia</a>
</body>
</html>
```

NOTAS: • Nótese como se designa el atributo **accesskey** dentro de la etiqueta **a**.

• Nótese como se debe colocar la dirección a la página que queremos redirigir utilizando el atributo en cuestión; esto puede verse en la etiqueta **a**.

• Desde mi página web, para acceder a este atributo, debo presionar las teclas ALT + h y me redireccionará a “Atributo ID.html”

• Los atajos creados no deben coincidir con los que viene de, por ejemplo, Google Chrome por defecto.

21.5 Atributos Contenteditable y Spellcheck:

Estos atributos permiten al usuario corregir el texto en términos de ortografía. Se pueden traducir como “*Contenido editable*” y “*Corrector ortográfico*” respectivamente.

/***/

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Contenteditable y Spellcheck</title>
</head>
<body>
```

<p contenteditable="true" **spellcheck=**"true"**>**Lorem ipsum dolor sit amet, consectetur adipisicing elit. Molestiae, error impedit itaque, nobis a quisquam atque aliquam deserunt, sequi nemo dolores nisi totam amet tenetur officia corrupti exercitationem. Quam, soluta.**</p>**

</body>

</html>

NOTA: • Nótese la notación de los atributos **contenteditable** y **spellcheck** y los valores que pueden adoptar: *true* o *false*.

21.6 Atributos Dir:

Este atributo permite dar dirección al texto; por defecto se coloca en la parte izquierda del navegador aunque con atributos se puede cambiar.

/**/

Ejemplo - HTML:

<!DOCTYPE html>

<html lang="en"**>**

<head>

<meta charset="UTF-8"**>**

<meta http-equiv="X-UA-Compatible" **content=**"IE=edge"**>**

<meta name="viewport" **content=**"width=device-width, initial-scale=1.0"**>**

<title>Dir**</title>**

</head>

<body>

<p dir="ltr"**>**Izquierda**</p>** *#Alinea el texto a la izquierda.*

<p dir="rtl"**>**Derecha**</p>** *#Alinea el texto a la derecha.*

<p dir="auto"**>**Auto**</p>** *#Alinea el texto automáticamente.*

</body>

</html>

NOTA: Nótese la forma de alinear los párrafos **p** hacia la izquierda, derecha y automáticamente a la izquierda.

21.7 Atributos Extra Parte 1:

En esta lección se va a ver como se puede agregar ciertos atributos a un texto o incluso a imágenes, los cuales mejoran la experiencia del usuario. A su vez, vamos a ver un atributo específico para ocultar contenido.

El atributo **draggable** permite, desde la página web, NO seleccionar el texto si no que lo puedo arrastrar.

El atributo **hidden** permite ocultar el texto y no lleva ningún argumento.

/**********/

Ejemplo - HTML:

<!DOCTYPE html>

<html lang="en"**>**

<head>

<meta charset="UTF-8"**>**

<meta http-equiv="X-UA-Compatible" **content=**"IE=edge"**>**

<meta name="viewport" **content=**"width=device-width, initial-scale=1.0"**>**

<title>Atributos extra**</title>**

</head>

<body>

<p hidden>Lorem ipsum dolor sit amet consectetur adipisicing elit. Placeat aspernatur nostrum accusamus obcaecati consequatur, modi vitae sequi quasi mollitia est amet esse quibusdam rerum! Rem quidem voluptas officiis corrupti eveniet.**</p>**

<p draggable="true"**>**Lorem ipsum dolor sit amet consectetur adipisicing elit. Impedit, eveniet amet? Molestias deleniti officiis doloribus laborum rerum officia nihil, in beatae quo! Corporis quae porro ipsa! Deleniti, dolorum! Enim, quidem.**</p>**

</body>

</html>

NOTAS: • Nótese como se emplean estos atributos en los párrafos **p**.

• El atributo **draggable** puede adoptar dos valores: *true* o *false*, mientras que **hidden** no adopta ningún valor.

21.8 Atributos Extra Parte 2:

El atributo **tabindex** permite que, desde la página web, apreto la tecla tabublación, la selección salte hacia el orden que yo elije y no por defecto que salta el orden descendente.

El atributo **title** permite ver, cuando coloco el cursor sobre un enlace sin apretarlo, una leyenda creada por mí que diga hacia donde dirige aquel enlace.

/**/

Ejemplo - HTML:

<!DOCTYPE html>

<html lang="en"**>**

<head>

<meta charset="UTF-8"**>**

<meta http-equiv="X-UA-Compatible" **content=**"IE=edge"**>**

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Atributos Extra</title>
</head>
<body>
    <a href="https://www.google.com" tabindex="1">Google</a>
    <br>
    <a href="https://www.google.es" tabindex="3">Google español</a>
    <br>
    <a href="https://www.google.com.en" tabindex="2" title="Google Inglés. Pagina de google">Google ingles</a>
</body>
</html>
```

NOTA: • Nótese como se emplean los atributos de esta lección. El orden numérico 1, 2, 3 en el atributo **tabindex** indica el orden que saltará la selección al apretar la tecla “Tab” en la página web, comenzando desde el 1.

22. SECCIÓN 22: CSS

22.1 CSS3 (Teoría):

CSS (Cascading Style Sheet) hojas de estilo en cascada. CSS es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML. Al igual que HTML, CSS NO es un lenguaje de programación dado que no utilizan variables.

Historia:

El organismo W3C (http://www.w3.org/) (World Wide Web Consortium), encargado de crear todos los estándares relacionados con la web, propuso la creación de un lenguaje de hojas de estilos específico para el lenguaje HTML. En 1996 se publica CSS.

Estructura:

Consta de 3 valores:

p **{color:** red**;}**

p = Selector. Es el elemento dentro de HTML al que se le va aplicar un estilo.

color = Propiedad. Estilo que se le va aplicar al selector

red = Valor de propiedad. Atributo que se aplica al selector.

{color: red**;}** = Declaración

Padres, hijos y hermanos:

Un hijo es una etiqueta que se encuentra dentro de otra, por ejemplo, un **h1** que se encuentra dentro del **body**.

Un hermano es una etiqueta que se encuentra “a la par” con otra etiqueta, siguiendo el ejemplo del hijo, si disponemos otra etiqueta, por ejemplo **p**, dentro del **body**, la etiqueta **p** será hermano de la etiqueta **h1**.

Por defecto, un padre, es aquella etiqueta que engloba otras etiquetas, por ejemplo, el **body** es el padre de las etiquetas **h1** y **p**.

22.2 Incorporar CSS en un documento HTML:

La manera de afectar un documento HTML a los estilos de CSS es a través de la etiqueta ***link***. A partir de ahí, puedo crear un documento.css donde puedo darle estilos al texto del documento HTML.

/***/

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="Style.css">
    <title>CSS</title>
</head>
<body>
    <h1>Titulo</h1>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Tempore minima repellat odio, dolore cupiditate ipsam, voluptatum quo quas, dignissimos mollitia delectus! Esse placeat rem mollitia corporis distinctio recusandae aliquam velit.</p>
</body>
</html>
```

NOTA: • **<link rel=**"stylesheet" **href=**"Style.css"**>**. **href** indica la dirección del documento css.

Ejemplo – CSS:

```
h1{
    background-color: blue;
}
```

NOTA: El selector h1 implica que se le está proveyendo estilos a la etiqueta **h1** en el documento HTML.

22.3 El Primer Pilar de CSS: La Cascada:

La Cascada hace referencia al orden que se le van a dar estilos al documento HTML de modo descendente.

/***/

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="Style.css">
    <title>CSS</title>
</head>
<body>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Tempore minima repellat odio, dolore cupiditate ipsam, voluptatum quo quas, dignissimos mollitia delectus! Esse placeat rem mollitia corporis distinctio recusandae aliquam velit.</p>
    <h1>Titulo</h1>
```

</body>

</html>

Ejemplo – CSS:

body**{background-color:** black**;}**

main**{background-color:** green**;}**

p**{color:** red**;}**

h1**{color:** blue**;}**

NOTA: Si se vuelven a crear los mismos selectores con diferentes Atributos de propiedad, éstos reemplazarán a los que fueron definidos previamente.

22.4 El Segundo Pilar de CSS: La Herencia:

Para indicar a CSS que se está utilizando la herencia, al selector en cuestión hay que darle el *Atributo de propiedad* **inherit**, entonces, adoptará el estilo de la etiqueta más próxima a esta definido previamente.

/**********/

Ejemplo - HTML:

<!DOCTYPE html>

<html lang="en"**>**

<head>

<meta charset="UTF-8"**>**

<meta http-equiv="X-UA-Compatible" **content=**"IE=edge"**>**

<meta name="viewport" **content=**"width=device-width, initial-scale=1.0"**>**

```
    <link rel="stylesheet" href="Style.css">
    <title>CSS</title>
</head>
<body>
    <main>
        <p>Lorem ipsum <a href="#"> sit amet, consectetur adipisicing elit. Necessitatibus adipis</a>ci, facere deserunt vitae accusamus non laboriosam magnam cum beatae quibusdam fugiat officiis itaque cupiditate amet harum odio numquam corrupti eligendi.</p>
        <h1>Titulo</h1>
    </main></body>
</html>
```

Ejemplo – CSS:

```
p{
    background-color: yellow;
    color: white;
}

a{
    color: inherit;
}
```

NOTAS:

• **<a href="#">**. El # indica que el enlace creado no será redirigido, sólo se marcará como enlace.

• **color:** inherit**;**. Esta es la manera de expresar la **herencia** en CSS.

22.5 El Tercer Pilar de CSS: La Especificidad:

Hace referencia al orden o a la prioridad en que se colocan los estilos a partir de sus selectores.

El peso de orden de importancia es:

important = 10000

style=”color: red:” = 1000. Es el estilo de línea en HTML.

id = 100

class = 10

p = 1

El orden de importancia implica los atributos de propiedad que va adoptar un texto cuando se encuentre con varios estilos definidos. Claramente, el estilo que va a predominar en el **!important**.

/***/

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet" href="Style.css">
   <title>Cascada</title>
</head>
<body>
   <main>
```

<p class="clase" **id=**"identificador" **style=**"color:greenyellow;" **>**Lorem ipsum **** sit amet, consectetur adipisicing elit. Necessitatibus adipis****ci, facere deserunt vitae accusamus non laboriosam magnam cum beatae quibusdam fugiat officiis itaque cupiditate amet harum odio numquam corrupti eligendi.**</p>**

<h1>Titulo**</h1>**

</main>

</body>

</html>

Ejemplo – CSS:

.clase**{**

color: lawngreen**;**

}

#identificador**{**

color: brown**;**

}

p**{**

color: rgb(8, 8, 8) !important**;**

}

NOTA: El color que va a predominar en el párrafo **p** va ser el categorizado como **!important**.

23. SECCIÓN 23: SELECTORES

23.1 Selector Universal:

El selector universal está representado por el **asterisco ***. Con él se puede cambiar el estilo a todo el texto de la página web.

/**/

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet" href="Style.css">
   <title>Selectores</title>
</head>
<body>
      <h1>Este es un titulo</h1>
        <p>Lorem ipsum sit amet, consectetur adipisicing elit. Necessitatibus adipisci, facere deserunt vitae accusamus non laboriosam magnam cum beatae quibusdam fugiat officiis itaque cupiditate amet harum odio numquam corrupti eligendi.</p>
        <a href="#">Link</a>
</body>
</html>
```

Ejemplo – CSS:

```
*{
   font-size: 50px;
   color: pink;
}
```

23.2 Selector de Etiqueta:

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet" href="Style.css">
   <title>Selectores</title>
</head>
<body>
      <h1>Este es un titulo</h1>
      <h2>Este es un subtitulo</h2>
       <p>Lorem ipsum sit amet, consectetur adipisicing elit. Necessitatibus adipisci, facere deserunt vitae accusamus non laboriosam magnam cum beatae quibusdam fugiat officiis itaque cupiditate amet harum odio numquam corrupti eligendi.</p>
      <a href="#">Link</a>
</body>
```

```
</html>
```

Ejemplo – CSS:

```
h1{
    background-color: indigo;
}

h2{
    font-family: Georgia, 'Times New Roman', Times, serif;
    font-size: 48px;
}

p{
    text-align: center;
}
```

23.3 Selector de Etiquetas Múltiples:

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <link rel="stylesheet" href="Style.css">
    <title>Selectores</title>
</head>
<body>
    <h1>Este en un titulo</h1>
    <h2>Este es un subtitulo</h2>
    <h3>Este es un subtitulo</h3>
    <h4>Este es un h4</h4>
    <h5>Este es un h5</h5>
    <h6>Este es un h6</h6>
    <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Veniam optio nam sint? Provident sapiente dolorum deserunt, pariatur molestias ratione dolorem tempora suscipit nam, nihil totam ab, animi quia corporis repellat!</p>
    <a href="#">Link</a>
</body>
</html>
```

Ejemplo – CSS:

```
h1,h2,h3,h4,h5,h6,p,a{
    color: red;
    background-color: black;
}
```

NOTA: Nótese cómo se pueden seleccionar varias etiquetas y agregarle es estilo que deseo para ellas.

23.4 Selectores ID y Class:

El atributo **id** a diferencia de **class** sólo puede adoptar un nombre para identificar la etiqueta a la cual se hace referencia. Por ejemplo, veremos a continuación que **class** se utiliza en la etiqueta **p** y **a** con el mismo nombre "*párrafo*"; esto no puede ocurrir con **id**.

/***/

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="Style.css">
    <title>Selectores</title>
</head>
<body>
    <h1>Este en un titulo</h1>
    <h2>Este es un subtitulo</h2>
    <h3>Este es un subtitulo</h3>
    <h4>Este es un h4</h4>
    <h5>Este es un h5</h5>
    <h6>Este es un h6</h6>
    <p class="parrafo">Lorem ipsum, dolor sit amet consectetur adipisicing elit. Veniam optio nam sint? Provident <a href="#"> dolorum deserunt, pariatur molestias ratione dolorem tempora suscipit</a> nam, nihil totam ab, animi quia corporis repellat!</p>
```

<p class="parrafo2"**>**Lorem ipsum dolor, ****sit amet consectetur adipisicing elit. Soluta enim explicabo magni quasi eos placeat deleniti nesciunt, hic ipsam accusamus, aliquam**** in amet? Fugiat a enim autem voluptate architecto corporis.**</p>**

<h3 id="identificacion"**>**Lorem, ipsum dolor sit amet consectetur adipisicing elit. Eaque obcaecati dolore deleniti? Quae autem nulla cupiditate ut molestiae facere, corrupti aspernatur amet ex illo nesciunt velit, excepturi, quidem dolor beatae?**</h3>**

****Link****

</body>

</html>

Ejemplo – CSS:

```
p.parrafo{
    color: white;
    font-size: 49px;
    background-color: black;
}

.parrafo2{
    color: white;
    font-size: 24px;
    background-color: black;
}

#identificacion{
    font-size: 30px;
    font-family: Arial, Helvetica, sans-serif;
```

```
    color: chartreuse;
}
```

NOTA: Cuando disponemos varios atributos **class** con el mismo nombre, para distinguirlos y dar estilos a esa etiqueta el selector se debe colocar, por ejemplo, **p.parrafo** o **a.parrafo**.

23.5 Selector Descendente:

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="Style.css">
    <title>Selectores</title>
</head>
<body>
    <h1>Este en un titulo</h1>
    <h2>Este es un subtitulo</h2>
    <h3>Este es un subtitulo</h3>
    <h4>Este es un h4</h4>
    <h5>Este es un h5</h5>
```

<h6>Este es un h6**</h6>**

** **Lorem ipsum, dolor sit amet consectetur adipisicing elit. Veniam optio nam sint? Provident **** dolorum deserunt, pariatur molestias ratione dolorem tempora suscipit nam, nihil totam ab, animi quia corporis repellat!** **

****Link****

</body>

</html>

Ejemplo – CSS:

p a**{**

background-color: green**;**

}

NOTA: Nótese la notación para darle estilos a la etiqueta **a** que se encuentra dentro de la etiqueta **p**; esta clase de selectores se denomina **selector descendente**.

23.6 Selector Hijo:

Ejemplo - HTML:

<!DOCTYPE html>

<html lang="en"**>**

<head>

<meta charset="UTF-8"**>**

<meta http-equiv="X-UA-Compatible" **content=**"IE=edge"**>**

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="Style.css">
    <title>Selectores</title>
</head>
<body>
    <h1>Este en un titulo <a href="#">Link</a></h1>
    <h1><p><a href="#">Link</a></p></h1>
    <h2>Este es un subtitulo</h2>
    <h3>Este es un subtitulo</h3>
    <h4>Este es un h4</h4>
    <h5>Este es un h5</h5>
    <h6>Este es un h6</h6>
    <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Veniam optio nam sint? Provident <a href="#"> dolorum deserunt, pariatur molestias ratione dolorem tempora suscipit</a> nam, nihil totam ab, animi quia corporis repellat!</p>
    <a href="#">Link</a>
</body>
</html>
```

Ejemplo – CSS:

```
h1 > a{
    color: coral;
    font-size: 48px;
    font-family: Verdana, Geneva, Tahoma, sans-serif;
    background-color: darkgoldenrod;
}
```

```
p > a{
    color: firebrick;
    font-size: 72px;
}
```

NOTAS:

• h1 > a. Implica que TODAS las etiquetas de **a** hijas directamente de **h1** van a recibir el estilo que se encuentra dentro de las llaves.

• p > a. Implica que TODAS las etiquetas de **a** hijas directamente de **p** van a recibir el estilo que se encuentra dentro de las llaves.

23.7 Selector Adyacente:

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="Style.css">
    <title>Selectores</title>
</head>
<body>
```

<h1>Este en un titulo ****Link**</h1>**

<h1> **Link </h1>**

<h2>Es hermano**</h2>**

<h3>Es adyacente**</h3>**

<h3>No es adyacente**</h3>**

<h4>Este es un h4**</h4>**

<h5>Este es un h5**</h5>**

<h6>Este es un h6**</h6>**

** **Lorem ipsum, dolor sit amet consectetur adipisicing elit. Veniam optio nam sint? Provident **** dolorum deserunt, pariatur molestias ratione dolorem tempora suscipit nam, nihil totam ab, animi quia corporis repellat!** **

****Link****

</body>

</html>

Ejemplo – CSS:

```
h2 + h3{ /*El elemento más cercano a h2 le ponemos el siguiente estilo*/
    color: fuchsia;
    font-weight: bold;
    font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida Sans', Arial, sans-serif;
}
```

NOTAS:

• h2 + h3 > a. Implica que la PRIMERA etiqueta **h3** adyacente a **h2** va a recibir los estilos definidos dentro de las llaves.

23.8 Selector Atributo:

Ejemplo - HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="Style.css">
    <title>Selectores</title>
</head>
<body>
    <h1>Este en un titulo <a href="#">Link</a></h1>
    <h1><p><a href="#">Link</a></p></h1>
    <h2>Es hermano</h2>
    <h3>Es adyacente</h3>
    <h3>No es adyacente</h3>
    <h4>Este es un h4</h4>
    <h5>Este es un h5</h5>
    <h6>Este es un h6</h6>
    <p class="parrafo">Lorem ipsum, dolor sit amet consectetur adipisicing elit. Veniam optio nam sint? Provident <a href="#"> dolorum deserunt, pariatur molestias ratione dolorem tempora suscipit</a> nam, nihil totam ab, animi quia corporis repellat!</p>
    <a href="#" class="parrafo1">Link</a>
    </body>
```

</html>

Ejemplo – CSS:

```
[class]{
    color: red;
    font-size: 48px;
    background-color: royalblue;
}

[class="parrafo"]{
    color: salmon;
    font-weight: bold;
}
```

NOTAS:

• [class]. Permite modificar los estilos de TODAS las etiquetas que contengan el atributo **class**. También se puede filtrar por el nombre del atributo como: [class=”párrafo”].

24. SECCIÓN 24: SELENIUM WEB DRIVER CON PYTHON

24.1 Qué es Selenium Web Driver?:

Es una herramienta para automatizar Testing. Precisamente, Selenium es una herramienta de código abierto que se utiliza para automatizar las pruebas realizadas en los navegadores web. Es una de las herramientas más utilizadas para hacer *testing* en las aplicaciones web.

Características:

• Utilizado con múltiples lenguajes de programación.

• Múltiplataforma. Es soportada por Windows, Linux, Mac, etc.

• Ejecutable con todos los navegadores.

• Implementa herramientas de documentación.

• Puede ser utilizado con Jenkins, Maven y Docker.

Limitaciones:

• Sólo aplicable para aplicaciones web.

• No posee soporte.

• Para testear algunos campos es necesario utilizar extensiones.

• No se instala de manera nativa. En este caso lo vamos a correr sobre un **IDE** llamado **PyCharm**.

24.2 Instalar PyCharm:

24.3 Instalar Selenium en PyCharm:

24.4 Instalación de Drivers para Selenium:

24.5 Inspección al Entorno de Testing:

24.6 Creando el Primer Script de Automatización:

/**/

Ejemplo - PyCharm:

```
from selenium import webdriver

driver = webdriver.Chrome(executable_path="Drivers/chromedriver.exe")
driver.maximize_window()

driver.get("https://www.udemy.com/join/login-popup/?locale=es_ES&response_type=html&next=https%3A%2F%2Fwww.udemy.com%2F")

driver.close()
```

NOTA: El Código es utilizado en el IDE PyCharm y explica lo siguiente:

- Importe desde selenium el driver.
- Creo una variable *driver* para ejecutar el driver de Chrome.
- Maximizo la ventana de Windows.
- Con el método **get**, abro la página web a Testear.
- Cierro la variable *driver* creada.

NO HA FUNCIONADO LA APERTURA DE LA PÁGINA WEB EN CUESTION.

24.7 Antes de comenzar:

A medida que se avanza con el *testing*, hay que tener en cuenta ciertas cosas para automatizar la página web:

- Aparece un Captcha para completar.

- Tener en cuenta lo aprendido en HTML.

- Identificar las etiquetas y los atributos de HTML que utiliza la página web.

- “Selecciona un elemento de la página para inspeccionarlo” desde el panel de “inspeccionar” en la página web.

24.8 Accediendo al Login por medio de ID:

Ejemplo:

```
from selenium import webdriver
import time

controlador = webdriver.Chrome(executable_path="Drivers/chromedriver.exe")
controlador.get("https://www.udemy.com/join/login-popup/?locale=es_ES&response_type=html&next=https%3A%2F%2Fwww.udemy.com%2F")
time.sleep(1)

usuario = controlador.find_element_by_id("email--1")
clave = controlador.find_element_by_id("id_password")
time.sleep(1)

usuario.send_keys("dfdflujogramas@gmail.com")
time.sleep(5)

clave.send_keys("12345678910")
```

```
time.sleep(5)

boton = controlador.find_element_by_id("submit-id-submit")
boton.click()
time.sleep(5)

controlador.quit()
```

NO HA FUNCIONADO LA APERTURA DE LA PÁGINA WEB EN CUESTION.

24.9 Accediendo al Login por medio de Name:

Ejemplo:

```
from selenium import webdriver
import time

controlador = webdriver.Firefox(executable_path="Drivers/geckodriver.exe")

controlador.get("https://www.udemy.com/join/login-popup/?locale=es_ES&response_type=html&next=https%3A%2F%2Fwww.udemy.com%2F")
time.sleep(1)

usuario = controlador.find_element_by_name("email")
clave = controlador.find_element_by_name("password")
time.sleep(1)

usuario.send_keys("dfdflujogramas@gmail.com")
time.sleep(5)
```

clave**.send_keys**("12345678910")

time.sleep(5)

boton = controlador**.find_element_by_name**("submit")

boton.click()

time.sleep(10)

controlador**.quit()**

NO HA FUNCIONADO LA APERTURA DE LA PÁGINA WEB EN CUESTION.

24.10 Accediendo al Login por medio de Class:

Ejemplo:

from selenium **import** webdriver

import time

driver = webdriver.Chrome(**executable_path=**"Drivers/chromedriver.exe")

driver.get("https://www.udemy.com/join/login-popup/?locale=es_ES&response_type=html&next=https%3A%2F%2Fwww.udemy.com%2F")

time.sleep(1)

email = driver**.find_element_class_name**("form-control")

clave = driver**.find_element_class_name**("textinput")

time.sleep(1)

email**.send_keys**("damian.angelucci@hotmail.com")

time.sleep(5)

clave**.send_keys**("holas231")

time.sleep(5)

boton = webdriver**.find_element_by_class_name**("btn-primary ")

boton**.click()**

time.sleep(10)

driver**.close()**

NOTA: Al inspeccionar un elemento por **clase** y me encuentro, por ejemplo, con la clase “btn btn-primary”, ese espacio significa que el elemento inspeccionado tiene dos clases: “btn” y “btn-primary”. Para **selenium** no es necesario colocar estas dos, dado que no lo va a reconocer, sólo se colocará una de ellas para su reconocimiento.

NO HA FUNCIONADO LA APERTURA DE LA PÁGINA WEB EN CUESTION.

24.11 Recuperar cuenta por medio de Link:

Ejemplo:

from selenium **import** webdriver

import time

driver = webdriver.Chrome(**executable_path=**"Drivers/chromedriver.exe")

driver.get("https://www.udemy.com/join/login-popup/?locale=en_US&response_type=html&next=https%3A%2F%2Fwww.udemy.com%2F&persist_locale=")

link_recuperacion = driver**.find_element_by_link_text**("Forgot Password")

link_recuperacion**.click()**

driver**.quit()**

NOTAS: • Se ha utilizado la página en ingles dado que en español el elemento por link resultaba tener la letra “ñ” y recordemos que los lenguajes no interpretan la letra “ñ”.

• Para encontrar el link, se coloca el contenido que se encuentra entre las etiquetas de apertura **a** y cierre ***/a***.

NO HA FUNCIONADO LA APERTURA DE LA PÁGINA WEB EN CUESTION.

24.12 Recuperar cuenta por medio de Partial Link:

Ejemplo:

from selenium **import** webdriver

import time

driver = webdriver.Chrome(**executable_path=**"Drivers/chromedriver.exe")

driver.get("https://www.udemy.com/join/login-popup/?locale=en_US&response_type=html&next=https%3A%2F%2Fwww.udemy.com%2F&persist_locale=")

time.sleep(1)

link_recuperacion = driver**.find_element_by_partial_link_text**("Forgot")

link_recuperacion**.click()**

time.sleep(5)

correo = driver**.find_element_by_id**("form-element--1")

correo**.send_keys**("dfdflujogramas@gmail.com")

time.sleep(1)

validar = driver**.find_element_by_class_name**("recaptcha-checkbox-border")

validar**.click()**

boton = driver**.find_element_by_class_name**("btn-primary")

boton**.click()**

time.sleep(5)

driver**.quit()**

NOTA: NO se puede automatizar una página web con CAPTCHA. Por lo tanto, las pruebas deben efectuarse en páginas webs que no tengan captcha, desactivarlo o ingresarlo manualmente.

NO HA FUNCIONADO LA APERTURA DE LA PÁGINA WEB EN CUESTION.

24.13 Acceder por medio de una sintaxis diferente:

Ejemplo:

from selenium **import** webdriver

from selenium**.webdriver.common.by import By**

import time

```
driver = webdriver.Chrome(executable_path="Drivers/chromedriver.exe")

driver.get("https://www.udemy.com/join/login-popup/?locale=es_ES&response_type=html&next=https%3A%2F%2Fwww.udemy.com%2F")

driver.maximize_window()

time.sleep(5)


correo = driver.find_element(By.ID, "email--1")

clave = driver.find_element(By.ID, "id_password")

time.sleep(5)


correo.send_keys("dfdflujogramas@gmail.com")

time.sleep(5)


clave.send_keys("12345678910")

time.sleep(5)


boton = driver.find_element(By.ID, "submit-id-submit")

boton.click()

time.sleep(5)


driver.quit()
```

NOTAS: • Esta es otra sintaxis para automatizar una página web a través del atributo **ID**. Nótese que el proceso se puede repetir para otros atributos (**Name, Class,** etc) y reemplazar el nombre de estos en el lugar respectivo del código.

• Cuando se emplea esta sintaxis, es necesario colocar en la cabecera: **from** selenium**.webdriver.common.by import By**

• driver**.maximize_window().** Permite maximizar el navegador

NO HA FUNCIONADO LA APERTURA DE LA PÁGINA WEB EN CUESTION.

24.14 XPATH:

Es otro método de selenium para identificar componentes de manera distinta a la que ya hemos visto. Es muy utilizado para identificar componentes que no poseen atributos.

XPATH es un lenguaje que permite construir expresiones que recorren y procesan un documento XML. El lenguaje XML es parecido al HTML pero permite crear, técnicamente, un lenguaje propio.

Permite buscar y seleccionar teniendo en cuenta la estructura jerárquica del XML. Por jerarquía se entiende los contenedores padres, hijos y/ hermanos.

24.15 Instalar Chropath:

24.16 Cómo utilizar Chropath?:

Dentro de la sección **chropath** dentro de la ventana **inspeccionar**, existen rutas relativas y absolutas. La primera es aquella que, posiblemente, puede cambiar mientras que las rutas absolutas no cambian. Siempre es recomendable trabajar con rutas relativas.

Sintaxis de ruta relativa:

//viñeta[@atributo=’valor’]

Por ejemplo: //input[@name='email']

Es decir, con XPATH puedo sacar componentes según su ubicación y buscarlas en toda la estructura HTML.

24.17 Accediendo al Login por medio de Chropath y Xpath:

En esta lección, vamos a trabajar con rutas relativas, aquellas que pueden cambiar de dirección dentro de la página web.

Se identifican como **Rel Xpath** dentro de **inspección** de la página web.

Ejemplo:

```
from selenium import webdriver
import time

driver = webdriver.Chrome(executable_path="Drivers/chromedriver.exe")
driver.get("https://www.udemy.com/join/login-popup/?locale=es_ES&response_type=html&next=https%3A%2F%2Fwww.udemy.com%2F")
time.sleep(1)

usuario = driver.find_element_by_xpath("//input[@id='email--1']")
usuario.send_keys("dfdflujogramas@gmail.com")
time.sleep(1)

clave = driver.find_element_by_xpath("//input[@name='password']")
clave.send_keys("12345678910")
time.sleep(1)

boton = driver.find_element_by_xpath("//input[@name='submit']")
boton.click()
time.sleep(5)
```

driver**.quit()**

NOTA: NO HA FUNCIONADO LA APERTURA DE LA PÁGINA WEB EN CUESTION.

24.18 Accediendo al Login por medio de ruta absoluta:

En la lección anterior, hemos trabajado con rutas relativas, ahora trabajaremos con rutas absolutas, es decir, aquellos elementos que no van a ser movidos en otras ubicaciones de la página web. En las cuales debemos especificar cuales son las etiquetas padres, hijos, hermanos, por lo tanto, si colocamos mal una etiqueta, no reconocerá la automatización de la página web por lo que resulta poco conveniente a la hora de automatizar.

Se identifican como **Abs Xpath** dentro de **inspección** de la página web.

Ejemplo:

```
from selenium import webdriver

import time


driver = webdriver.Chrome(executable_path="Drivers/chromedriver.exe")

driver.get("https://www.udemy.com/join/login-popup/?locale=es_ES&response_type=html&next=https%3A%2F%2Fwww.udemy.com%2F")

time.sleep(1)


usuario = driver.find_element_by_xpath("/html[1]/body[1]/div[1]/div[2]/div[1]/div[3]/form[1]/div[1]/div[1]/div[1]/input[1]")

usuario.send_keys("dfdflujogramas@gmail.com")

time.sleep(1)
```

```
clave = driver.find_element_by_xpath("/html[1]/body[1]/div[1]/div[2]/div[1]/div[3]/form[1]/div[1]/div[2]/div[1]/input[1]")

clave.send_keys("12345678910")

time.sleep(1)


boton = driver.find_element_by_xpath("//input[@name='submit']")

boton.click()

time.sleep(5)


driver.quit()
```

NOTA: NO HA FUNCIONADO LA APERTURA DE LA PÁGINA WEB EN CUESTION.

24.19 Accediendo con atributos CSS:

Para acceder con atributos CSS, hay que cambiar la solapa *Selectors* a *CSS Sel.* para identificar que vas a utilizar atributos de CSS.

Sintaxis:

viñeta[atributo='valor']

Por ejemplo: input[id='email--1']

/***/

Ejemplo:

```
from selenium import webdriver

import time
```

```
driver = webdriver.Chrome(executable_path="Drivers/chromedriver.exe")
driver.get("https://www.udemy.com/join/login-popup/?locale=es_ES&response_type=html&next=https%3A%2F%2Fwww.udemy.com%2F")
time.sleep(1)

usuario = driver.find_element_by_css_selector("input[id='email--1']")
usuario.send_keys("damian.angelucci@hotmail.com")
time.sleep(1)

clave = driver.find_element_by_css_selector("input[name='password’]”)
clave.send_keys("holas231")
time.sleep(1)

boton = driver.find_element_by_css_selector("input[name='submit']")
boton.click()
time.sleep(5)

driver.quit()
```

NOTA: NO HA FUNCIONADO LA APERTURA DE LA PÁGINA WEB EN CUESTION.

24.20 Accediendo con ID y Atributos CSS:

Esto sólo es posible cuando se quiere acceder a un atributo CSS pero que tenga un atributo ID. En estos casos, la sintaxis será como se presenta a continuación.

Sintaxis:

viñeta#valor_id

Por ejemplo: input#email--1

/***/

Ejemplo:

```
from selenium import webdriver
import time

driver = webdriver.Chrome(executable_path="Drivers/chromedriver.exe")
driver.get("https://www.udemy.com/join/login-popup/?skip_suggest=1&locale=es_ES&next=https%3A%2F%2Fwww.udemy.com%2F")
time.sleep(1)

correo = driver.find_element_by_css_selector("input#email--1")
contrasena = driver.find_element_by_css_selector("input#id_password")
time.sleep(1)

correo.send_keys("damian.angelucci@hotmail.com")
time.sleep(1)

contrasena.send_keys("holas231")
time.sleep(1)

boton = driver.find_element_by_css_selector("input#submit-id-submit")
boton.click()
time.sleep(5)

driver.quit()
```

NOTA: NO HA FUNCIONADO LA APERTURA DE LA PÁGINA WEB EN CUESTION.

24.21 Accediendo con Class y Atributos CSS:

Esto sólo es posible cuando se quiere acceder a un atributo CSS pero que tenga un atributo Class. En estos casos, la sintaxis será como se presenta a continuación.

Sintaxis:

viñeta.valor_class

Por ejemplo: input.form-control

/***/

Ejemplo:

```
from selenium import webdriver
import time

driver = webdriver.Chrome(executable_path="Drivers/chromedriver.exe")
driver.get("https://www.udemy.com/join/login-popup/?skip_suggest=1&locale=es_ES&next=https%3A%2F%2Fwww.udemy.com%2F")
time.sleep(1)

correo = driver.find_element_by_css_selector("input.form-control")
clave = driver.find_element_by_css_selector("input.textinput")
time.sleep(1)

correo.send_keys("damian.angelucci@hotmail.com")
time.sleep(1)
```

```
clave.send_keys("holas231")
time.sleep(1)

boton = driver.find_element_by_css_selector("input.btn-primary")
boton.click()
time.sleep(5)

driver.quit()
```

NOTA: NO HA FUNCIONADO LA APERTURA DE LA PÁGINA WEB EN CUESTION.

24.22 Seleccionar Varios Elementos:

Cuando se utiliza la sentencia **.find_elements** (por clase, id, o simplemente así) el valor arrojado es una LISTA de elementos que se van a recorrer a través de un bucle **for**. Por eso, no es posible enviarle ni nombre de correo o contraseña para automatizar el programa.

/**********/

Ejemplo:

```
from selenium import webdriver
import time

driver = webdriver.Chrome(executable_path="Drivers/chromedriver.exe")
driver.get("https://www.udemy.com/join/login-popup/?skip_suggest=1&locale=es_ES&next=https%3A%2F%2Fwww.udemy.com%2F")
driver.maximize_window()
```

```
varios = driver.find_elements_by_class_name("form-control")
for i in varios:
    time.sleep(1)
    i.send_keys("holas231")
```

NOTA: NO HA FUNCIONADO LA APERTURA DE LA PÁGINA WEB EN CUESTION.

25. SECCIÓN 25: INTRODUCCIÓN A TURTLE

25.1 Turtle (Teoría):

Librería:

Conjunto de módulos cuya agrupación tiene una finalidad específica y que también puede ser invocada, tal como un módulo. Pero no es un módulo, si no un conjunto de ellos con una estructura determinada para lograr una finalidad.

Módulo:

Es una porción de un programa de ordenador. De las varias tareas que debe realizar un programa para cumplir con su función y objetivos, un módulo realizará, comúnmente, una de dichas tareas (o varias, en algún caso).

Ejemplo: **import math**

Palabras claves:

Para trabajar con librerías o módulos es necesario tener en cuenta las siguientes palabras claves.

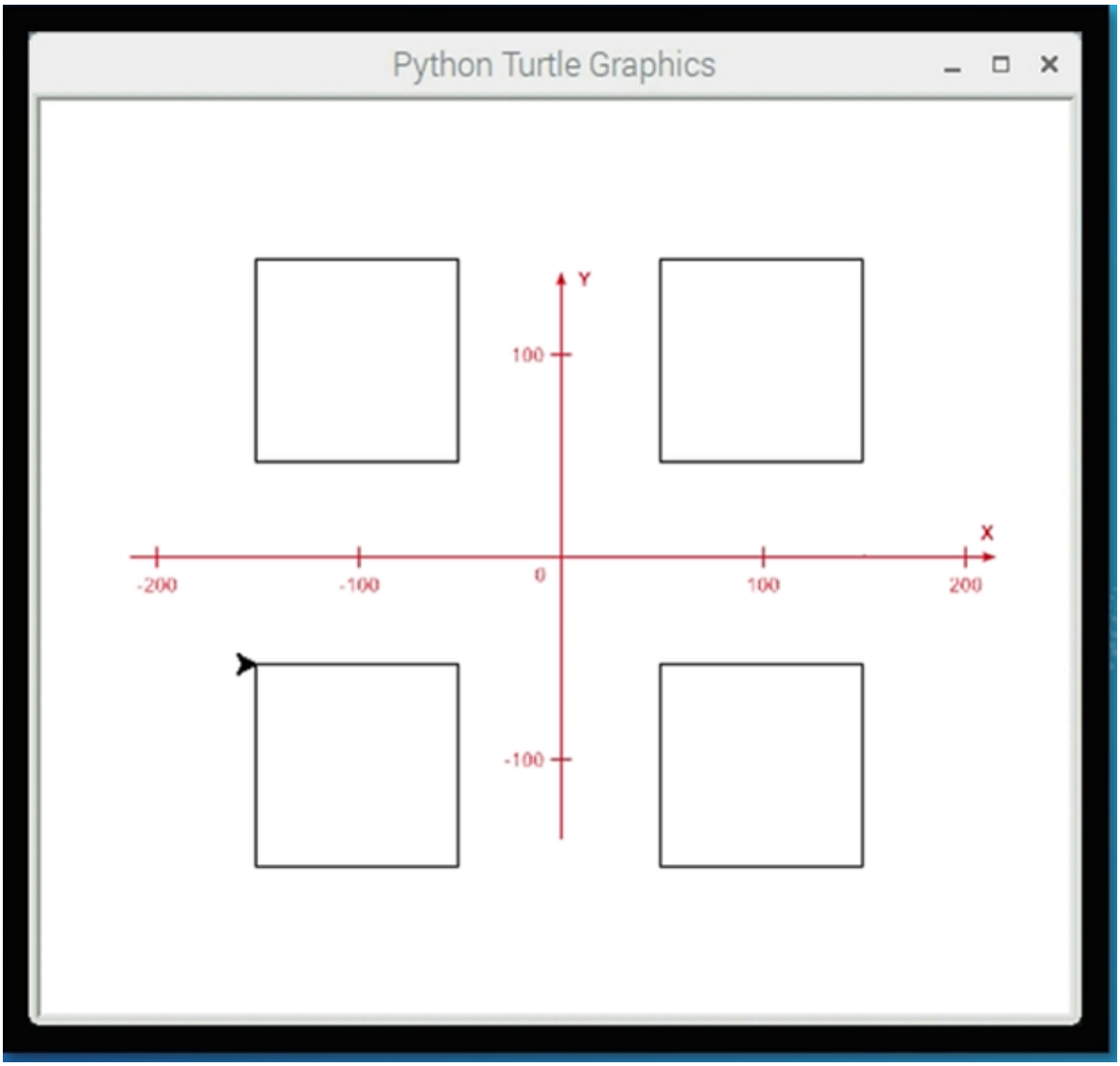
- **import**
- **import math**

- **from <módulo>import<función>**
- **from math import sqrt**
- **from math import***

Módulo (Librería) Turtle:

Python Turtle Graphics es un módulo de programación gráfica para Python utilizado como método para enseñar programación a través de coordenadas relativas. El objeto a programar recibe el nombre tortuga,

elemento clave en el lenguaje de programación Logo, creado por *Seymour Papert* a finales de la década de los 60.



25.2 Comandos Básicos:

Ejemplo 1:

```
import turtle

s = turtle.Screen()
```

```
t = turtle.Turtle()

t.backward(100) #Desplaza la tortula 100px hacia atrás
t.right(90) #Gira la tortula 90° hacia la derecha
t.forward(100) #Desplaza la tortula 100px hacia adelante
t.left(90) #Gira la tortula 90° hacia la izquierda
t.forward(100)

turtle.done()
```

Ejemplo 2:

```
import turtle

s = turtle.Screen()
t = turtle.Turtle()

t.bk(100) #Desplaza la tortula 100px hacia atrás
t.rt(90) #Gira la tortula 90° hacia la derecha
t.fd(100) #Desplaza la tortula 100px hacia adelante
t.lt(90) #Gira la tortula 90° hacia la izquierda
t.fd(100)

turtle.done()
```

NOTAS:

- s = turtle**.Screen()**. Permite visualizer una ventana gráfica.
- t = turtle**.Turtle()**. turtle es el modulo; Turtle() es una clase dentro del módulo.

• Los ejemplos muestran el mismo movimiento gráfico de la tortuga, la diferencia es que en el “Ejemplo 2”, los comandos de desplazamiento y giro de la tortuga están abreviados.

25.3 Movimiento de la Torguta en el lienzo:

Ejemplo 1:

```
import turtle

s = turtle.Screen()
t = turtle.Turtle()

t.goto(100,100) #Desplaza la tortuga hacia el punto (100,100)
t.goto(-100,100)
t.home() #Regresa a la tortuga al punto inicial (0,0)
# t.goto(0,0) Regresa a la tortuga al punto inicial (0,0)

t.fd(100)
t.rt(90)
t.fd(100)
t.rt(90)
t.fd(100)
t.rt(90)
t.fd(100)
t.rt(90)
```

```
turtle.done()
```

25.4 Comandos Especiales:

Ejemplo:

```
import turtle

s = turtle.Screen()
t = turtle.Turtle()

t.speed(10) #Permite darle velocidad al desplazamiento de la tortuga
t.circle(10) #La tortuga efectúa un círculo de radio 10px.
t.speed(10)
t.circle(50)
t.dot(30) #La tortuga efectúa un punto de diámetro 30px

t.hideturtle() #Permite ocultar la tortuga
t.speed(1)
t.circle(40)
t.showturtle() #Permite mostrar la tortuga
t.circle(100)

t.setx(100) #Mantiene el eje Y desplaza en 100px sobre el eje X
t.sety(-100) #Mantiene el eje X desplaza en -100px sobre el eje Y

turtle.done()
```

NOTAS:

• t**.speed**(10). Permite darle velocidad al desplazamiento de la tortuga; un valor de 1 es la velocidad mínima de movimiento.

25.5 Personalizar Turtle:

Ejemplo:

```
import turtle

s = turtle.Screen()
t = turtle.Turtle()

s.bgcolor("red") #Permite cambiar de color el fondo de la pantalla
s.title("Proyecto 1") #Permite cambiar el nombre del lienzo

t.shapesize(3,3,3) #Permite modificar el ancho, largo y borde de la tortuga respectivamente
t.fillcolor("orange") #Permite cambiar el color de la tortuga
t.fd(100)

t.pencolor("white") #Permite cambiar el color del borde de la tortuga y su desplazamiento
t.fd(100)

t.color("green", "blue") #Otra manera de cambiar el borde y trazo y color de la tortuga
t.rt(90)
```

t**.fd**(100)

t**.pensize**(5) *#Permite cambiar el grosor del desplazamiento de la tortuga*
t**.fd**(100)

turtle**.done()**

25.6 Otros Atributos:

Ejemplo:
import turtle

s = turtle**.Screen()**
t = turtle**.Turtle()**

'''t**.begin_fill()** *#Permite sombrear el círculo*
t**.circle**(100)
t**.end_fill()** *#Permite finalizer el sombreado*

t**.begin_fill()**
t**.color**("white", "white")
t**.circle**(50)
t**.end_fill()'''**

t**.shape**("turtle") *#Permite dar forma de tortuga al lapiz*

t**.fd**(100)

t**.penup()** *#Levanta la tortuga para que no trace*

t**.fd**(50)

t**.pendown()** *#Baja la tortuga para que comience a trazar nuevamente*

t**.fd**(100)

t**.undo()** *#Retrocede la última acción de la tortuga*

t**.clear()** *#Limpia todo el trazo realizado por la tortuga*

t**.reset()** *#Resetea todo lo efectuado por la tortuga*

t**.fd**(100)

t**.stamp()** *#Deja un sello de los movimientos efectuados*

t**.fd**(100)

turtle**.done()**

25.7 <u>Automatizando Procesos:</u>

<u>Ejemplo</u>:

import turtle

s = turtle**.Screen()**

t = turtle**.Turtle()**

'''t**.fd**(100)

t**.rt**(90)

```
t.fd(100)
t.rt(90)
t.fd(100)
t.rt(90)
t.fd(100)'''

'''for i in range(4):
    t.fd(100)
    t.rt(90)'''

'''t.circle(100)
t.circle(90)
t.circle(80)
t.circle(70)
t.circle(60)
t.circle(50)
t.circle(40)
t.circle(30)
t.circle(20)
t.circle(10)'''

i = 0
resultado = input ("Quieres imprimir una figura?")
if resultado == "si":
    while i <= 100:
        t.circle(i)
        i +=10
```

```
else:
    print("Oka")

turtle.done()
```

NOTA: Utilizando los ciclos **for** y **while** nos permiten automatizar el movimiento de la tortuga además de optimizar el código dado que requiere menor cantidad de líneas.

25.8 Descripción del Proyecto 1:

Carrera de tortugas. Se crean dos tortugas, una verde y la otra azul, las cuales resultan ser los jugadores. Parten de un estado inicial en la misma coordenada respecto al eje “x” y, a partir de la “tirada” de dados en forma aleatoria, las tortugas van a ir avanzando sobre ese mismo eje hacia un punto final, la cual es la meta. Quien primero llega a la meta, gana.

26. SECCIÓN 26: PRIMER PROYECTO CON TURTLE: CARRERA DE TORTUGAS

26.1 Creando nuestros jugadores:

Ejemplo:

```
import turtle
import random #Ya que vamos a "tirar los dados"

s = turtle.Screen()
s.title("Primero Proyecto")

jugador1 = turtle.Turtle()
jugador2 = turtle.Turtle()

jugador1.shape("turtle")
jugador1.color("green", "green")

jugador2.shape("turtle")
jugador2.color("blue", "blue")
```

[CONTINUA EN LA SIGUIENTE LECCIÓN]

```
turtle.done()
```

26.2 Dibujando las Metas:

Ejemplo:

[CONTINUA EN LA SIGUIENTE LECCIÓN]

```
import turtle
import random #Ya que vamos a "tirar los dados"

s = turtle.Screen()
s.title("Primero Proyecto")

jugador1 = turtle.Turtle()
jugador2 = turtle.Turtle()

jugador1.shape("turtle")
jugador1.color("green", "green")

jugador2.shape("turtle")
jugador2.color("blue", "blue")

jugador1.penup()
jugador1.goto(200,200)
jugador1.pendown()
jugador1.circle(40)

jugador1.penup()
jugador1.goto(-250, 225)

jugador2.penup()
```

jugador2**.goto**(200,-200)

jugador2**.pendown()**

jugador2**.circle**(40)

jugador2**.penup()**

jugador2**.goto**(-250, -170)

turtle**.done()**

26.3 Personalizando estilos de juego:

Ejemplo:

[CONTINUA EN LA LECCIÓN 26.5]

import turtle

import random *#Ya que vamos a "tirar los dados"*

s = turtle**.Screen()**

s**.title**("Primero Proyecto")

s**.bgcolor**("gray")

jugador1 = turtle**.Turtle()**

jugador2 = turtle**.Turtle()**

jugador1**.hideturtle()**

jugador1**.shape**("turtle")

jugador1**.color**(“green”, “green”)

jugador1**.shapesize**(2,2,2)

jugador1**.pensize**(3)

jugador2**.hideturtle()**

jugador2**.shape**(“turtle”)

jugador2**.color**(“blue”, “blue”)

jugador2**.shapesize**(2,2,2)

jugador2**.pensize**(3)

jugador1**.penup()**

jugador1**.goto**(200,200)

jugador1**.pendown()**

jugador1**.circle**(40)

jugador1**.penup()**

jugador1**.goto**(-250, 225)

jugador1**.showturtle()**

jugador2**.penup()**

jugador2**.goto**(200,-200)

jugador2**.pendown()**

jugador2**.circle**(40)

jugador2**.penup()**

```
jugador2.goto(-250, -170)
jugador2.showturtle()

turtle.done()
```

26.4 Método Pos():

Ejemplo:

```
import turtle

s = turtle.Screen()
t = turtle.Turtle()

t.fd(100)
t.rt(90)
t.fd(100)
print(t.pos())

turtle.done()
```

NOTA: Es de extrema importancia el método **pos()** dado que permite identificar en consola la posición en “x” e “y” de la tortuga.

26.5 Implementar Método pos():

Ejemplo:

[CONTINUA EN LA SIGUIENTE LECCIÓN]

```
import turtle
import random #Ya que vamos a “tirar los dados”

s = turtle.Screen()
s.title(“Primero Proyecto”)
s.bgcolor(“gray”)

jugador1 = turtle.Turtle()
jugador2 = turtle.Turtle()

jugador1.hideturtle()
jugador1.shape(“turtle”)
jugador1.color(“green”, “green”)
jugador1.shapesize(2,2,2)
jugador1.pensize(3)

jugador2.hideturtle()
jugador2.shape(“turtle”)
jugador2.color(“blue”, “blue”)
jugador2.shapesize(2,2,2)
jugador2.pensize(3)
```

```
jugador1.penup()
jugador1.goto(200,200)
jugador1.pendown()
jugador1.circle(40)

jugador1.penup()
jugador1.goto(-250, 225)
jugador1.showturtle()



jugador2.penup()
jugador2.goto(200,-200)
jugador2.pendown()
jugador2.circle(40)

jugador2.penup()
jugador2.goto(-250, -170)
jugador2.showturtle()

dado = [1,2,3,4,5,6]

for i in range(20): #Considero que en 20 movimientos la tortuga llega
    if jugador1.pos() >= (200,200):
        print("La tortuga verde ha ganado")
        break
    elif jugador2.pos() >= (200,-200):
        print("La tortuga azul ha ganado")
```

break

turtle**.done()**

26.6 Finalizando el Proyecto:

Ejemplo:

```
import turtle
import random #Ya que vamos a "tirar los dados"

s = turtle.Screen()
s.title("Primero Proyecto")
s.bgcolor("gray")

jugador1 = turtle.Turtle()
jugador2 = turtle.Turtle()

jugador1.hideturtle()
jugador1.shape("turtle")
jugador1.color("green", "green")
jugador1.shapesize(2,2,2)
jugador1.pensize(3)

jugador2.hideturtle()
jugador2.shape("turtle")
jugador2.color("blue", "blue")
```

```
jugador2.shapesize(2,2,2)
jugador2.pensize(3)


jugador1.penup()
jugador1.goto(200,200)
jugador1.pendown()
jugador1.circle(40)

jugador1.penup()
jugador1.goto(-250, 225)
jugador1.showturtle()


jugador2.penup()
jugador2.goto(200,-200)
jugador2.pendown()
jugador2.circle(40)

jugador2.penup()
jugador2.goto(-250, -170)
jugador2.showturtle()

dado = [1,2,3,4,5,6]

for i in range(20): #Considero que en 20 movimientos la tortuga llega
        if jugador1.pos() >= (200,200):
```

```
            print("La tortuga verde ha ganado")
            break
        elif jugador2.pos() >= (200,-200):
            print("La tortuga azul ha ganado")
            break
        else:
            turno1 = input("Presiona la tecla enter para avanzar la tortuga
            verde")
            turno1 = random.choice(dado)
            print("Tu numero es: ", turno1, "\nAvanzas: ", turno1*20)
            jugador1.pendown()
            jugador1.forward(20*turno1)

            turno2 = input("Presiona la tecla enter para avanzar la tortuga
            azul")
            turno2 = random.choice(dado)
            print("Tu numero es: ", turno2, "\nAvanzas: ", turno2*20)
            jugador2.pendown()
            jugador2.forward(20*turno2)

turtle.done()
```

27. SECCIÓN 27: SEGUNDO PROYECTO CON TURTLE: VIDEJUEGO "LA CULEBRITA"

27.1 Descripción del Proyecto 2:

27.2 Creando la Serpiente y la Pantalla:

Ejemplo:

[CONTINUA EN LA SIGUIENTE LECCIÓN]

```
import turtle

s = turtle.Screen()
s.setup(650, 650)
s.bgcolor("gray")
s.title("Proyecto 2")

serpiente = turtle.Turtle()
serpiente.speed(1)
serpiente.shape("square")
serpiente.penup()
serpiente.goto(0,0)
serpiente.direction = 'stop' #Será modificada para mov. por teclado
serpiente.color("green")

turtle.done()
```

NOTA:

• s**.setup**(650, 650). Define el tamaño de la pantalla 650x650px.

27.3 <u>Agregar Movimiento a la Serpiente</u>:

<u>Ejemplo</u>:

[CONTINUA EN LA SIGUIENTE LECCIÓN]

```
import turtle
import time

retraso = 0.1

s = turtle.Screen()
s.setup(650, 650) #Define el tamaño de la pantalla en px
s.bgcolor("gray")
s.title("Proyecto 2")

serpiente = turtle.Turtle()
serpiente.speed(1)
serpiente.shape("square")
serpiente.penup()
serpiente.goto(0,0)
serpiente.direction = 'stop' #Será modificada para mov. por teclado
serpiente.color("green")

def movimiento ():
```

if serpiente**.direction ==** ‘up’**:**

y = serpiente**.ycor() #*Creación de variable sobre Y***

serpiente**.sety**(y + 20) ***#Permite desplazar 20px sobre el eje Y positivo***

if serpiente**.direction ==** ‘down’**:**

y = serpiente**.ycor() #*Creación de variable sobre Y***

serpiente**.sety**(y - 20) ***#Permite desplazar 20px sobre el eje Y negativo***

if serpiente**.direction ==** ‘right**:**

x = serpiente**.xcor() #*Creación de variable sobre X***

serpiente**.setx**(x + 20) ***#Permite desplazar 20px sobre el eje X positivo.***

if serpiente**.direction ==** ‘left**:**

x = serpiente**.xcor() #*Creación de variable sobre X***

serpiente**.setx**(x + 20) ***#Permite desplazar 20px sobre el eje X negativo.***

while True:

s.**update() #*Actualiza la pantalla constantemente***

movimiento()

time.sleep(retraso)

turtle**.done()**

NOTAS:

• y = serpiente**.ycor().** Creo una variable “y” la cual le asigno la corrdenada Y.

• serpiente**.sety**(y + 20). Permite el movimiento de la serpiente por desplazamiento de 20px sobre el eje Y positivo.

27.4 Mover la Serpiente con las flechas del teclado:

Ejemplo:

[CONTINUA EN LA SIGUIENTE LECCIÓN]

```
import turtle
import time

retraso = 0.1

s = turtle.Screen()
s.setup(650, 650)
s.bgcolor("gray")
s.title("Proyecto 2")

serpiente = turtle.Turtle()
serpiente.speed(1)
serpiente.shape("square")
serpiente.penup()
serpiente.goto(0,0)
serpiente.direction = 'stop'
serpiente.color("green")

def arriba():
```

```
        serpiente.direction = ‘up’

def abajo():
        serpiente.direction = ‘down’

def derecha():
        serpiente.direction = ‘right’

def izquierda():
        serpiente.direction = ‘left’


def movimiento ():
        if serpiente.direction == ‘up’:
                y = serpiente.ycor()
                serpiente.sety(y + 20)

        if serpiente.direction == ‘down’:
                y = serpiente.ycor()
                serpiente.sety(y - 20)

        if serpiente.direction == ‘right:
                x = serpiente.xcor()
                serpiente.setx(x + 20)

        if serpiente.direction == ‘left:
                x = serpiente.xcor()
```

```
        serpiente.setx(x + 20)

s.listen()
s.onkeypress(arriba, “Up”)
s.onkeypress(abajo, “Down”)
s.onkeypress(derecha, “Right”)
s.onkeypress(izquierda, “Left”)

while True:
    s.update()
    movimiento()
    time.sleep(retraso)

turtle.done()
```

NOTAS:

• s**.listen().** Permite que la pantalla “escuche” las teclas que presiono por teclado.

• s**.onkeypress**(arriba, “Up”). Traducción: “Al presionar una tecla”. Recibe como parámetros dos argumentos: una función “arriba” y una tecla presionada “Tecla de flecha hacia arriba”.

27.5 Creando la Comida:

Ejemplo:

[CONTINUA EN LA SIGUIENTE LECCIÓN]

```
import turtle
import time
import random

retraso = 0.1

s = turtle.Screen()
s.setup(650, 650)
s.bgcolor("gray")
s.title("Proyecto 2")

serpiente = turtle.Turtle()
serpiente.speed(1)
serpiente.shape("square")
serpiente.penup()
serpiente.goto(0,0)
serpiente.direction = 'stop'
serpiente.color("green")

comida = turtle.Turtle()
comida.shape("circle")
comida.color("orange")
comida.penup()
comida.goto(random.randint(-300,300), random.randint(-300,300))
comida.speed(0)
```

```
def arriba():
    serpiente.direction = ‘up’

def abajo():
    serpiente.direction = ‘down’

def derecha():
    serpiente.direction = ‘right’

def izquierda():
    serpiente.direction = ‘left’

def movimiento ():
    if serpiente.direction == ‘up’:
        y = serpiente.ycor()
        serpiente.sety(y + 20)

    if serpiente.direction == ‘down’:
        y = serpiente.ycor()
        serpiente.sety(y - 20)

    if serpiente.direction == ‘right:
        x = serpiente.xcor()
        serpiente.setx(x + 20)

    if serpiente.direction == ‘left:
        x = serpiente.xcor()
```

```
        serpiente.setx(x + 20)

s.listen()
s.onkeypress(arriba, "Up")
s.onkeypress(abajo, "Down")
s.onkeypress(derecha, "Right")
s.onkeypress(izquierda, "Left")

while True:
    s.update()

    if serpiente.distance(comida) <20:
        x = random.randint(-300,300)
        y = random.randint(-300,300)
        comida.goto(x,y)

    movimiento()
    time.sleep(retraso)

turtle.done()
```

NOTAS:

• comida**.goto(random.randint**(-300,300), **random.randint**(-300,300)). Posiciona el objeto "comida" en una ubicación **aleatoria** dentro de los -300 y 300px tanto en el eje "x" como en el eje "y".

• **if** serpiente**.distance**(comida) <20**:**. Detecta la distancia entre los objetos “serpiente” y comida”. En la sentencia, si “serpiente” está a una distancia menor a 20px, “comida” desaparece de ese punto y se posiciona en un punto aleatorio definido por el programa.

27.6 Alargando el cuerpo de la serpiente - Parte 1:

Ejemplo:

[CONTINUA EN LA SIGUIENTE LECCIÓN]

```
import turtle
import time
import random

retraso = 0.1

s = turtle.Screen()
s.setup(650, 650)
s.bgcolor(“gray”)
s.title(“Proyecto 2”)

serpiente = turtle.Turtle()
serpiente.speed(1)
serpiente.shape(“square”)
serpiente.penup()
serpiente.goto(0,0)
serpiente.direction = ‘stop’
```

```
serpiente.color("green")

comida = turtle.Turtle()
comida.shape("circle")
comida.color("orange")
comida.penup()
comida.goto(random.randint(-300,300), random.randint(-300,300))
comida.speed(0)

cuerpo = [ ]

def arriba():
    serpiente.direction = 'up'

def abajo():
    serpiente.direction = 'down'

def derecha():
    serpiente.direction = 'right'

def izquierda():
    serpiente.direction = 'left'

def movimiento ():
    if serpiente.direction == 'up':
        y = serpiente.ycor()
        serpiente.sety(y + 20)
```

```
        if serpiente.direction == 'down':
            y = serpiente.ycor()
            serpiente.sety(y - 20)

        if serpiente.direction == 'right:
            x = serpiente.xcor()
            serpiente.setx(x + 20)

        if serpiente.direction == 'left:
            x = serpiente.xcor()
            serpiente.setx(x + 20)

s.listen()
s.onkeypress(arriba, "Up")
s.onkeypress(abajo, "Down")
s.onkeypress(derecha, "Right")
s.onkeypress(izquierda, "Left")

while True:
        s.update()

        if serpiente.distance(comida) <20:
            x = random.randint(-300,300)
            y = random.randint(-300,300)
            comida.goto(x,y)
```

```
            nuevo_cuerpo = turtle.Turtle()
            nuevo_cuerpo.shape("circle")
            nuevo_cuerpo.color("orange")
            nuevo_cuerpo.penup()
            nuevo_cuerpo.goto(0,0)
            nuevo_cuerpo.speed(0)

        movimiento()
        time.sleep(retraso)

turtle.done()
```

NOTAS:

• Las sentencias implementadas dentro del condicional, permiten crear un nuevo cuerpo a medida que la serpiente come su comida y la ubica en el punto (0,0).

• La próxima lección veremos como ubicar ese cuerpo nuevo a la serpiente.

27.7 Alargando el cuerpo de la serpiente - Parte 2:

Ejemplo:

[CONTINUA EN LA SIGUIENTE LECCIÓN]

```
import turtle
```

```
import time
import random

retraso = 0.1

s = turtle.Screen()
s.setup(650, 650)
s.bgcolor("gray")
s.title("Proyecto 2")

serpiente = turtle.Turtle()
serpiente.speed(1)
serpiente.shape("square")
serpiente.penup()
serpiente.goto(0,0)
serpiente.direction = 'stop'
serpiente.color("green")

comida = turtle.Turtle()
comida.shape("circle")
comida.color("orange")
comida.penup()
comida.goto(random.randint(-300,300), random.randint(-300,300))
comida.speed(0)

cuerpo = [ ]
```

```
def arriba():
    serpiente.direction = ‘up’

def abajo():
    serpiente.direction = ‘down’

def derecha():
    serpiente.direction = ‘right’

def izquierda():
    serpiente.direction = ‘left’

def movimiento ():
    if serpiente.direction == ‘up’:
        y = serpiente.ycor()
        serpiente.sety(y + 20)

    if serpiente.direction == ‘down’:
        y = serpiente.ycor()
        serpiente.sety(y - 20)

    if serpiente.direction == ‘right:
        x = serpiente.xcor()
        serpiente.setx(x + 20)

    if serpiente.direction == ‘left:
        x = serpiente.xcor()
```

```
            serpiente.setx(x + 20)

s.listen()
s.onkeypress(arriba, “Up”)
s.onkeypress(abajo, “Down”)
s.onkeypress(derecha, “Right”)
s.onkeypress(izquierda, “Left”)

while True:
        s.update()

        if serpiente.distance(comida) <20:
                x = random.randint(-300,300)
                y = random.randint(-300,300)
                comida.goto(x,y)

                nuevo_cuerpo = turtle.Turtle()
                nuevo_cuerpo.shape(“circle”)
                nuevo_cuerpo.color(“orange”)
                nuevo_cuerpo.penup()
                nuevo_cuerpo.goto(0,0)
                nuevo_cuerpo.speed(0)

        total = len(cuerpo) #Saca la longitud de la lista ‘cuerpo’
        for i in range(total -1, 0, -1):
                x = cuerpo[i-1].xcor()
                y = cuerpo[i-1].ycor()
```

```
            cuerpo[i].goto(x,y)

        if total > 0:
            x = serpiente.xcor()
            y = serpiente.ycor()
            cuerpo[0].goto(x,y)

        movimiento()
        time.sleep(retraso)

turtle.done()
```

27.8 Colisiones con la Pantalla y el Cuerpo:

Ejemplo:

[CONTINUA EN LA SIGUIENTE LECCIÓN]

```
import turtle
import time
import random

retraso = 0.1

s = turtle.Screen()
s.setup(650, 650)
s.bgcolor("gray")
```

```
s.title("Proyecto 2")

serpiente = turtle.Turtle()
serpiente.speed(1)
serpiente.shape("square")
serpiente.penup()
serpiente.goto(0,0)
serpiente.direction = 'stop'
serpiente.color("green")

comida = turtle.Turtle()
comida.shape("circle")
comida.color("orange")
comida.penup()
comida.goto(random.randint(-300,300), random.randint(-300,300))
comida.speed(0)

cuerpo = [ ]

def arriba():
    serpiente.direction = 'up'

def abajo():
    serpiente.direction = 'down'

def derecha():
    serpiente.direction = 'right'
```

```
def izquierda():
    serpiente.direction = ‘left’

def movimiento ():
    if serpiente.direction == ‘up’:
        y = serpiente.ycor()
        serpiente.sety(y + 20)

    if serpiente.direction == ‘down’:
        y = serpiente.ycor()
        serpiente.sety(y - 20)

    if serpiente.direction == ‘right:
        x = serpiente.xcor()
        serpiente.setx(x + 20)

    if serpiente.direction == ‘left:
        x = serpiente.xcor()
        serpiente.setx(x + 20)

s.listen()
s.onkeypress(arriba, “Up”)
s.onkeypress(abajo, “Down”)
s.onkeypress(derecha, “Right”)
s.onkeypress(izquierda, “Left”)
```

```
while True:
    s.update()

    if serpiente.xcor() > 300 or serpiente.xcor() < -300 or serpiente.ycor() > 300 or serpiente.ycor() < -300:
        time.sleep(2)
        for i in cuerpo:
            i.clear()
            i.hideturtle()
        serpiente.home()
        serpiente.direction = 'stop'
        cuerpo.clear()

    if serpiente.distance(comida) <20:
        x = random.randint(-300,300)
        y = random.randint(-300,300)
        comida.goto(x,y)

        nuevo_cuerpo = turtle.Turtle()
        nuevo_cuerpo.shape("circle")
        nuevo_cuerpo.color("orange")
        nuevo_cuerpo.penup()
        nuevo_cuerpo.goto(0,0)
        nuevo_cuerpo.speed(0)

    total = len(cuerpo) #Saca la longitud de la lista 'cuerpo'
    for i in range(total -1, 0, -1):
        x = cuerpo[i-1].xcor()
```

```
            y = cuerpo[i-1].ycor()
            cuerpo[i].goto(x,y)

    if total > 0:
            x = serpiente.xcor()
            y = serpiente.ycor()
            cuerpo[0].goto(x,y)

    movimiento()

    for i in cuerpo:
            if i.distance(serpiente) < 20:
                    for i in cuerpo:
                    i.clear()
                    i.hideturtle()
            serpiente.home()
            serpiente.direcition = 'stop'
            cuerpo.clear()

    time.sleep(retraso)

turtle.done()
```

NOTAS:

• Las sentencias implementadas dentro del condicional, permiten crear un nuevo cuerpo a medida que la serpiente come su comida y la ubica en el punto (0,0).

• La próxima lección veremos como ubicar ese cuerpo nuevo a la serpiente.

27.9 <u>Marcador de Puntaje</u>:

[*******FIN DEL PROYECTO**********]***

<u>Ejemplo</u>:

```
import turtle
import time
import random

retraso = 0.1

marcador = 0
marcador_alto = 0

s = turtle.Screen()
s.setup(650, 650)
s.bgcolor("gray")
s.title("Proyecto 2")

serpiente = turtle.Turtle()
serpiente.speed(1)
serpiente.shape("square")
serpiente.penup()
```

```
serpiente.goto(0,0)
serpiente.direction = ‘stop’
serpiente.color(“green”)

comida = turtle.Turtle()
comida.shape(“circle”)
comida.color(“orange”)
comida.penup()
comida.goto(random.randint(-300,300), random.randint(-300,300))
comida.speed(0)

cuerpo = [ ]

texto = turtle.Turtle()
texto.speed(0)
texto.color(“black”)
texto.penup()
texto.hideturtle()
texto.goto(0,-260)
texto.write(“Marcador: 0\tMarcador mas alto: 0”, align=”center”, font=(“verdana”, 24, “normal”))

def arriba():
    serpiente.direction = ‘up’

def abajo():
    serpiente.direction = ‘down’
```

```
def derecha():
    serpiente.direction = ‘right’

def izquierda():
    serpiente.direction = ‘left’

def movimiento ():
    if serpiente.direction == ‘up’:
        y = serpiente.ycor()
        serpiente.sety(y + 20)

    if serpiente.direction == ‘down’:
        y = serpiente.ycor()
        serpiente.sety(y - 20)

    if serpiente.direction == ‘right:
        x = serpiente.xcor()
        serpiente.setx(x + 20)

    if serpiente.direction == ‘left:
        x = serpiente.xcor()
        serpiente.setx(x + 20)

s.listen()
s.onkeypress(arriba, “Up”)
s.onkeypress(abajo, “Down”)
s.onkeypress(derecha, “Right”)
```

```
s.onkeypress(izquierda, “Left”)


while True:
    s.update()


    if serpiente.xcor() > 300 or serpiente.xcor() < -300 or serpiente.ycor() > 300 or serpiente.ycor() < -300:
        time.sleep(2)
        for i in cuerpo:
            i.clear()
            i.hideturtle()
        serpiente.home()
        serpiente.direction = ‘stop’
        cuerpo.clear()


        marcador = 0
        texto.clear()
        texto.write(“Marcador: {}\tMarcador mas alto: {}”.format(marcador, marcador_alto), align=”center”, font=(“verdana”, 24, “normal”)


    if serpiente.distance(comida) <20:
        x = random.randint(-300,300)
        y = random.randint(-300,300)
        comida.goto(x,y)


        nuevo_cuerpo = turtle.Turtle()
        nuevo_cuerpo.shape(“circle”)
```

```
        nuevo_cuerpo.color("orange")
        nuevo_cuerpo.penup()
        nuevo_cuerpo.goto(0,0)
        nuevo_cuerpo.speed(0)

        marcador += 10
        if marcador > marcador_alto:
            marcador_alto = marcador
            texto.write("Marcador: {}\tMarcador mas alto: {}".format(marcador, marcador_alto), align="center", font=("verdana", 24, "normal")

total = len(cuerpo) #Saca la longitud de la lista 'cuerpo'
for i in range(total -1, 0, -1):
        x = cuerpo[i-1].xcor()
        y = cuerpo[i-1].ycor()
        cuerpo[i].goto(x,y)

if total > 0:
        x = serpiente.xcor()
        y = serpiente.ycor()
        cuerpo[0].goto(x,y)

movimiento()

for i in cuerpo:
        if i.distance(serpiente) < 20:
            for i in cuerpo:
```

```
                i.clear()
                i.hideturtle()
            serpiente.home()
            serpiente.direcition = ‘stop’
            cuerpo.clear()

        time.sleep(retraso)

turtle.done()
```

28. SECCIÓN 28: EJERCICIOS

28.1 Ejercicios a realizar:

Les dejo 2 páginas web excelentes donde vas a poder encontrar diferentes ejercicios de python con su respectiva solución

- https://pythondiario.com/ejercicios-de-programacion-python
- https://aprendeconalf.es/docencia/python/ejercicios/

29. SECCIÓN 29: DESPEDIDA DEL CURSO

30. SECCIÓN 30: BONUS EXTRA

Como agradecimiento, te comparto mis rutas de aprendizaje y todos mis cursos al precio más económico.

Rutas de Aprendizaje:

- https://achirou.com/tu-ruta-de-aprendizaje/

Libros Gratis:

- https://achirou.com/redes-sociales/