

Desarrollo de contenido

Unidad 3

Arreglos

Fundamentos de programación
Ingeniería mecatrónica

Unidad 3. Arreglos

Introducción a la Unidad 3

En las unidades anteriores aprendimos como utilizar variables para almacenar datos o cadenas de caracteres y diferentes tipos de ciclos para resolver problemas básicos de programación desde su interpretación hasta su representación algorítmica codificada; cuando hablamos de arreglos nos referimos a un lugar de almacenamiento continuo de datos ordenados de una manera particular, estos son importantes dado que nos permiten realizar soluciones computacionales en casos donde las variables por sí solas no son suficientes y se requieren utilizar colecciones de datos del mismo tipo conocidas como arreglos o de diferentes tipos de datos conocidas como estructuras. Los conocimientos que utilizaremos en dichas unidades nos serán de gran ayuda para aplicar los conceptos de matrices, vectores y dimensionalidad que abordaremos más adelante para realizar implementaciones algorítmicas un poco complejas que nos permitan gestionar masivamente datos.

Resultados de aprendizaje de la Unidad 3

- Interpretar los diferentes tipos de datos operadores y controladores de flujo.
- Profundizar en el diseño y uso de funciones como elementos de optimización de algoritmos codificados.
- Conocer y utilizar las herramientas que se pueden usar en programación, para recorrer y buscar información en vectores y matrices.
- Unificar los conocimientos adquiridos para analizar, resolver, interpretar y codificar problemas básicos de programación.

Cronograma de actividades 1

Cronograma de actividades Unidad 3			
Actividad de aprendizaje*	Evidencia de aprendizaje**	Semana***	Ponderación
AA3. Arreglos	EA3. Proyecto integrador parte 3	Semana 7	25%
EA4 Proyecto integrador entrega final		Semana 8	30%
Total			55%

Actividad de aprendizaje 3: Arreglos

En esta Actividad de Aprendizaje seguirás profundizando en herramientas y soluciones para que puedas continuar mejorando tus habilidades como programador, de manera que tengas los elementos necesarios para abordar el análisis y la solución de los diferentes problemas que se puedan presentar, teniendo en cuenta la eficiencia en la creación y el diseño de tus programas.

Los temas que estudiarás son los siguientes:

- Funciones: estructuras y llamados
- Conceptos y uso de vectores
- Conceptos y uso de matrices
- Ejercicios prácticos

1. FUNCIONES: ESTRUCTURAS Y LLAMADOS.

Las funciones en programación brindan la posibilidad de optimizar el costo computacional de tu programa, en la medida que ayudan a disminuir el tamaño en cuanto a codificación, ayudan también a tener menos variables en operación, ya que los campos creados en la función pueden tomar valores diferentes en cada llamado de la función.

Cuando decimos llamado de la función, hacemos referencia a las ocasiones en que la ejecutamos, puede ser en diferentes partes del mismo programa.

¿Qué hacer para analizar la forma en que podemos crear una función o subprograma?

- Primero debemos ahondar más en algunos temas vistos anteriormente, con el objetivo de potenciar al máximo el diseño y el uso de nuestras funciones como: los tipos de datos, los sistemas de numeración y los operadores.
- Ahora vamos a llevar estos conocimientos al uso en el intérprete que estamos estudiando.

Dentro de los tipos de datos, tenemos, booleanos, numéricos, cadenas de caracteres o string, listas, tuplas, diccionarios, entre otros.

En esta unidad, retomaremos con un poco más de profundidad algunos de los temas ya vistos, para entrar en detalle en el diseño de funciones, su estructura y los diferentes usos y aplicaciones que estas tienen.

Comencemos...

Sistemas de numeración en Python

En los sistemas de numeración vimos que existe el sistema decimal, como su nombre lo indica, es en base 10. El sistema octal, como su nombre lo indica, en base 8. El sistema hexadecimal, en base 16 y el sistema binario, en base 2.

Ya aprendimos cómo realizar conversiones entre los sistemas de numeración, ahora veremos cómo usarlos en Python.

Tabla 1.
Información sistemas de numeración

Sistemas de numeración en Python		
Nombre	Intervalo	Notación
Octal	De 0 a 7	0o
Binario	0 y 1	0x
Hexadecimal	De 0 a 9 y de A hasta F	0b

Fuente: Elaboración propia

A continuación, comenzaremos con el análisis de un ejemplo completo para que comprendamos mejor. Para ello, tenemos el número 56 en decimal.

Figura 1.
Ejemplo sistemas de numeración

Decimal 56₁₀						
Binario 111000₁₀	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰
	32	16	8	4	2	1
	1	1	1	0	0	0
Octal 70₈	1	1	1	0	0	0
		7			0	
Hexadecimal 38₁₆	1	1	1	0	0	0
		3			8	

Fuente: Elaboración propia.

Como observamos en la imagen, lo representamos en los sistemas de numeración: decimal, binario, octal y hexadecimal.

En la imagen se observa también una de las formas que tenemos para realizar dichas conversiones. En nuestro ejemplo, tenemos como decimal el número 56, el cual se lee cincuenta y seis, pero en el caso de los sistemas octal y hexadecimal, no se leen en conjunto, si no dígito a dígito, especificando el sistema de numeración. En el caso del 7 0 en octal, se lee siete cero en octal y el 3 8 en hexadecimal, se lee tres ocho en hexadecimal.

Es importante tener en cuenta lo anterior, para que logres ubicarte de manera clara y concreta con el sistema de numeración que estás trabajando, ya sea que estés comunicando algo relacionado con el tema o estés recibiendo alguna información asociada.

A continuación, en la figura 2, vamos a observar cómo podemos ingresar o darle algún tipo de instrucción a python, en la cual se haga necesario usar los diferentes sistemas de numeración.

Figura 2.
Sistemas de numeración en python.

```

1 decimal=56 #Número en decimal
2 binario=0b111000 #Número en binario
3 octal=0o70 #Número en octal
4 hexadecimal=0x38 #Número en hexadecimal
5
6 print(decimal) #se muestran en pantalla los números.
7 print(binario)
8 print(octal)
9 print(hexadecimal)

```

input

56
56
56
56

Fuente: Elaboración propia

Para denotar un dato de tipo binario en python, lo debes hacer antecedido del 0b, lo que le dirá al intérprete que el dato que ingresaste es un dato en base 2, También hay una forma de denotar especial para el octal (0o) y para el hexadecimal (0x). Todo número que ingreses sin antes decirle a python que el dato es un sistema de numeración, definiéndolo con lo que acabamos de ver, él lo tomará como si fuera un decimal o una cadena de string al momento de usar las letras propias del hexadecimal. En ese caso no conseguirás aquello que estabas buscando.

Operadores booleanos

Los operadores booleanos, también conocidos como operadores lógicos, son aquellos que basan su comportamiento o su funcionalidad en compuertas AND y OR, donde las tablas de verdades de estas nos ayudan a delimitar decisiones y marcar condiciones en nuestro programa.

Vamos a tener en cuenta básicamente las dos compuertas más significativas y de las cuáles se desprenden o se obtienen las otras. Como mencionamos anteriormente son AND y OR. Ahora vamos a recordarlas y a llevar su aplicabilidad al contexto de programación.

Primero recordemos que, en una AND, cuando tenemos por lo menos una condición falsa o entrada baja (0), la salida es falsa o 0.

En una OR, cuando tenemos por lo menos una condición verdadera o entrada alta (1), la salida es Verdadera o 1.

Vamos a ilustrarlo con un ejemplo, tomando algunos elementos de un contexto industrial. En una compañía de producción y clasificación de cajas, tenemos cuatro bandas transportadoras. La banda 1 (B1) y la banda 2 (B2), están en constante funcionamiento. La banda 3 y la 4, se encienden según sea el tamaño de la caja que está pasando en el momento.

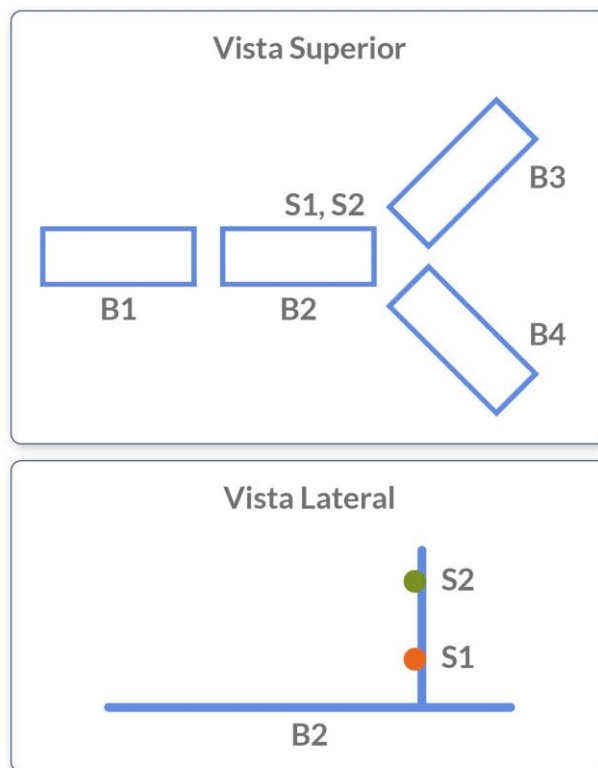
Tenemos dos tipos de cajas. Cajas grandes y cajas pequeñas. Necesitamos almacenarlas. Las pequeñas con las pequeñas y las grandes con las grandes.

Tenemos también dos sensores S1 y S2, cuando se activa solamente S1, quiere decir que la caja es pequeña, por lo tanto, se enciende la banda 3 (B3), para continuar el recorrido. Cuando se activan ambos sensores, quiere decir que la caja es grande, por lo tanto, se activa la banda 4 (B4), para continuar el recorrido.

A continuación, en la figura 3 veremos una gráfica del sistema, que nos ayudará a comprender mejor.

Tengamos en cuenta que este ejemplo incluye elementos de un contexto industrial, pero que, en este caso, es ilustrativo, en este curso no lo llevaremos a un caso con entradas y salidas reales en un dispositivo programable.

Figura 3.
Ilustración ejemplo bandas transportadoras



Fuente: Elaboración propia.

En la gráfica y en la descripción anterior vemos un contexto, que puede parecernos complejo y enredado, vamos a aterrizar a lo que hemos venido trabajando y aprendiendo. Se hizo así, para que desde ahora veamos la importancia de desarrollar habilidades de programación y la aplicabilidad en nuestra cotidianidad como ingenieros.

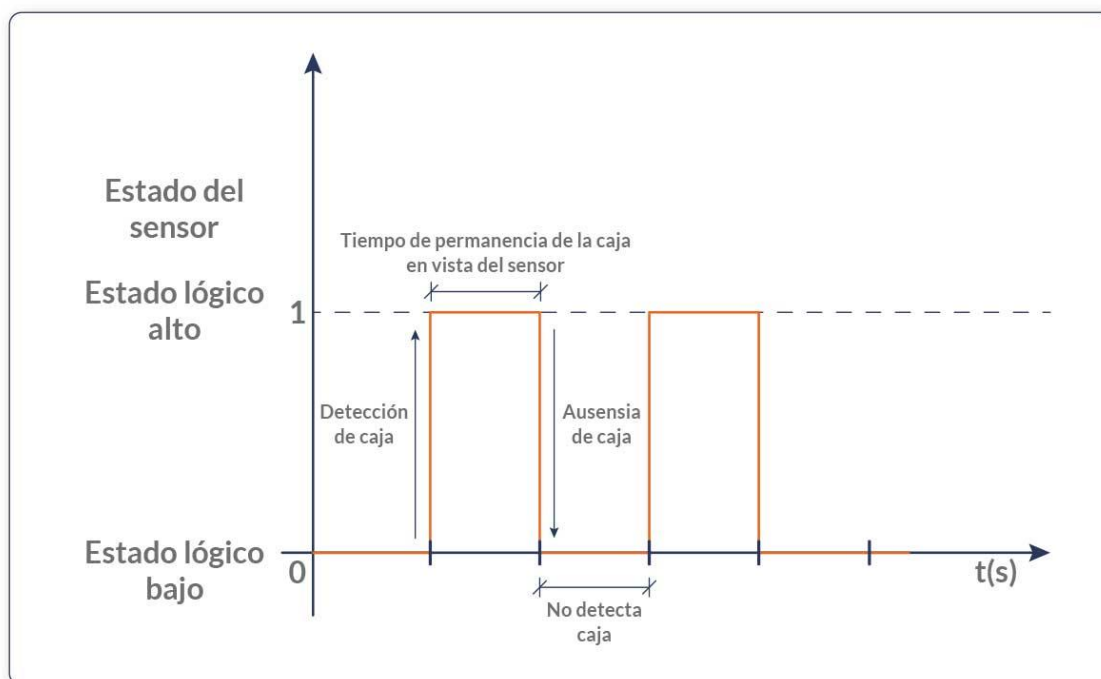
En la lógica de programación que se debe desarrollar para el uso de los sensores en este caso, es donde vamos a analizar el uso de operadores booleanos.

Este análisis parte también de recordar el uso de la instrucción if. Haciendo remembranza identificamos que puede ser una herramienta que nos sirva para ejecutar esta acción. Necesitamos construir una condición que pregunte por el estado de los dos sensores y en concordancia con esto encienda B3 o B4 según sea el caso.

Cabe aclarar que los sensores usados en este caso tienen una señal de tipo digital, o sea que solo tienen dos estados. Un estado lógico alto '1' o un estado lógico bajo '0'. No permite puntos intermedios.

En la figura 4 vamos a ver de manera gráfica este comportamiento.

Figura 4. Comportamiento de una señal de tipo digital.



Fuente: Elaboración propia

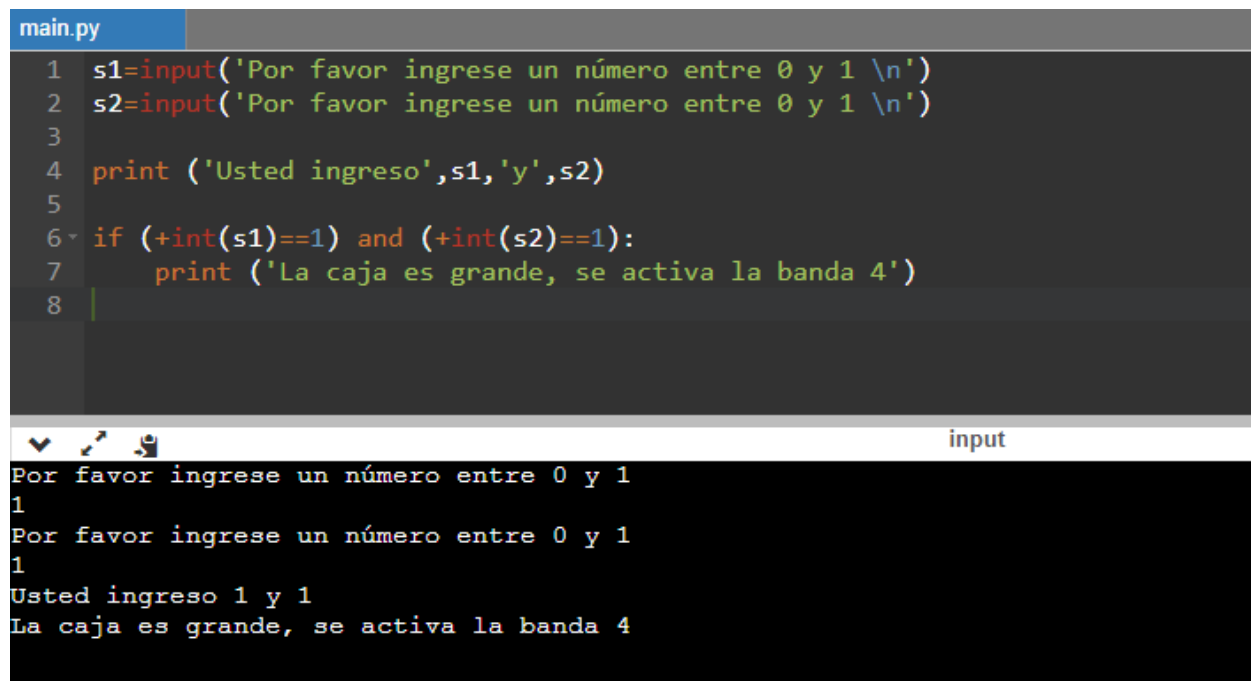
Ejercicio:

Teniendo en cuenta la explicación y la argumentación vista, el programa que vamos a diseñar, será el siguiente:

En nuestro caso, vamos a asumir que los estados del sensor los ingresa el usuario. Por lo tanto, vamos a pedirle que ingrese dos números entre 0 y 1. Los números ingresados los guardaremos en S1 y S2 respectivamente (son dos números, de acuerdo a la cantidad de sensores). Luego vamos a realizar la validación, para poder ejecutar la acción de dar aviso mediante un mensaje en pantalla de cuál banda está encendida.

En la figura 5, veremos una parte de la solución de este ejercicio, vamos a poner mucho cuidado, porque el paso a seguir es generar nuevas condiciones, para que complementes este programa.

Figura 5.
Ejercicio de las bandas transportadoras



```

main.py
1 s1=input('Por favor ingrese un número entre 0 y 1 \n')
2 s2=input('Por favor ingrese un número entre 0 y 1 \n')
3
4 print ('Usted ingreso',s1,'y',s2)
5
6 if (+int(s1)==1) and (+int(s2)==1):
7     print ('La caja es grande, se activa la banda 4')
8
input
Por favor ingrese un número entre 0 y 1
1
Por favor ingrese un número entre 0 y 1
1
Usted ingreso 1 y 1
La caja es grande, se activa la banda 4

```

Fuente: Elaboración propia.

En la figura 5 se observa una pequeña luz, para abordar la solución del ejercicio. En el siguiente cuadro de actividad, se propone un ejercicio más completo, con más condiciones, pertinente dentro del proceso de aprendizaje que estamos realizando.

Actividad

- Se le propone al estudiante que para poner en práctica los conocimientos adquiridos, y teniendo en cuenta el planteamiento que se hizo para el problema de las bandas transportadoras, realice lo siguiente.
- Hacer un programa que se llame bandas transportadoras, que pida al usuario que ingrese los valores de los sensores S1 y S2. Recordemos que los sensores que tienen una señal de tipo digital solo admiten dos estados lógicos: alto o bajo (no admite puntos intermedios). Por lo tanto, debemos validar que efectivamente el dato que ingrese sea 1 o 0, si no es así, muestra un mensaje que indique que ingresó mal el dato, que debe hacerlo según el requerimiento.
- Si tanto s1 como s2, son iguales a cero, esto quiere decir que no hay una caja que necesite ser clasificada, por lo tanto, no arranca ninguna de las bandas.
- Si $s1 = 0$ y $s2 = 1$, debe mostrar un mensaje indicando error, pues no tiene sentido que el sistema detecte presencia en s2 (sensor de arriba), pero en s1 no (sensor de abajo).
- Si $s1=1$ y $s2=0$, esta condición nos indica que la caja presente es pequeña, por lo tanto, debemos mostrar un mensaje en pantalla, que indique que la banda 3 arrancó, por presencia de caja pequeña.
- Si s1 y s2 son iguales a 1, nos indica que la caja presente es de tamaño grande, por lo tanto, se muestra en pantalla un mensaje que indica que se enciende la banda 4.
- El programa lo vamos a hacer dentro de una estructura iterativa de control. Recordemos que se ingresa a un ciclo cuando se cumple una condición y salimos de ahí cuando dicha condición deja de ser verdadera.

	<ul style="list-style-type: none">• Cuando se presenta la condición ($s1=0$ y $s2=1$), en ese momento salimos de la estructura iterativa. De lo contrario, vamos a estar dentro del ciclo, volviendo a iniciar el programa, sin necesidad de ejecutarlo de nuevo.• No olvides documentar el programa.
--	--

Si realizas el ejercicio anterior, tendrás muy buenos elementos de valor, para seguir el desarrollo del curso y lograr conocimientos importantes para tu futuro como programador.

Gracias a las temáticas vistas, podemos ahora adentrarnos en la manera cómo podemos crear y usar funciones.

Las funciones o subprogramas son tramos o partes de programas reutilizables, que realizan una acción determinada, pero que deben estar dentro de un programa principal, para poderse ejecutar e interactuar con otras instrucciones y lograr la solución a una necesidad específica.

Estructura de una función

Para iniciar con la creación de una función, debemos emplear la palabra *def*, seguido del nombre de la función, unos paréntesis, dentro de los cuales puedes definir los parámetros de entrada, estos son opcionales, luego dos puntos.

En el renglón siguiente, comienza el diseño instruccional para la utilidad de la función. Este renglón no comienza en el mismo punto del anterior. Para definir lo que está dentro de la función, debe quedar un tanto desplazado hacia la derecha. Al finalizar la función, la escritura vuelve desde el comienzo, al lado izquierdo.

Los ejemplos nos ayudan siempre a vislumbrar de una forma más clara lo que vemos en teoría, de esta manera logramos afianzar nuestros conocimientos.

Ejemplo:

En una universidad, se tienen 3 grupos de estudiantes en el curso de robótica. Cada grupo tiene un total de 4 estudiantes. A nivel académico se crea una competencia, donde los alumnos de cada curso deben presentar un robot móvil con unos requerimientos que les hicieron los docentes. Al final de la presentación, debemos conocer el promedio de notas de cada grupo, y saber cuál es el grupo que tiene la nota más alta.

Para realizar la comparación de mayor que o menor que, es importante conocer los operadores que aparecen en la tabla 2.

Tabla 2.
Operadores de comparación.

Operador de comparación	Explicación
$>$	Mayor que. Si el operando de la izquierda es mayor que el de la derecha, la condición es verdadera. En caso contrario es falsa.
$>=$	Mayor o igual que. Si el operando de la izquierda es mayor o igual que el de la derecha, la condición es verdadera. En caso contrario es falsa.
$<$	Menor que. Si el operando de la izquierda es menor que el de la derecha, la condición es verdadera. En caso contrario es falsa.
$<=$	Menor o igual que. Si el operando de la izquierda es menor o igual que el de la derecha, la condición es verdadera. En caso contrario es falsa.
$==$	Igual que. Si el operando de la izquierda es igual al de la derecha, la condición es verdadera. En caso contrario es falsa.
$!=$	Diferente. Si los operandos son diferentes, la condición es verdadera. En caso contrario es falsa.

Fuente: Elaboración propia.

Después de ver los operadores de comparación, ahora si procedemos a dar solución al ejercicio propuesto.

Lo haremos con funciones.

En la figura 6, se observa la codificación en python para la solución del ejercicio propuesto.

Figura 6.
Solución ejercicio con funciones

```
main.py
1  #Programa ejemplo para estudiar creación y estructura de funciones
2  #Se crea la función, teniendo en cuenta la estructura vista
3  def promediorobotica(est1, est2, est3, est4): #def Nombrefuncion (parametros de entrada):
4      sumaprom=est1+est2+est3+est4 #Instrucciones que conforman la función
5      prom=sumaprom / 4
6      return prom #se retorna el valor promedio, para luego compararlo.
7
8  #Se llama la función, entre paréntesis se dan las notas de los estudiantes
9  #Se asigna a un campo llamado promgrupo1, de manera que el
10 #resultado que retorna la función quede almacenado ahí.
11 promgrupo1=promediorobotica(5,4,3,4)
12 promgrupo2=promediorobotica(4.3,3.6,4.5,4)
13 promgrupo3=promediorobotica(3.2,3.8,5,4.2)
14
15 #Condiciones para establecer el mayor valor
16 if ((promgrupo1>promgrupo2) and (promgrupo1>promgrupo3)):
17     print ("El grupo ganador es el número 1, con un promedio de:",promgrupo1)
18
19 elif ((promgrupo2>promgrupo1) and (promgrupo2>promgrupo3)):
20     print ("El grupo ganador es el número 2, con un promedio de:",promgrupo2)
21
22 elif ((promgrupo3>promgrupo1) and (promgrupo3>promgrupo2)):
23     print ("El grupo ganador es el número 3, con un promedio de:",promgrupo3)
input
El grupo ganador es el número 2, con un promedio de: 4.1
```

Fuente: Elaboración propia.

Importante



Cuando una función retorna algo, se debe ubicar o llamar en una instrucción de asignación o un imprima (print).

Siempre retorna el dato, únicamente en el lugar donde fue llamada la función.

La solución que acabamos de ver en la figura 6, es básica.

¿Qué pasa cuando hay uno o varios promedios iguales?

En el ejercicio que acabamos de realizar, en algún momento dado se puede presentar que dos de los promedios o los 3, sean iguales. Al presentarse este caso, el programa no muestra ningún mensaje.

Por lo tanto, a continuación, te presentamos una actividad, para que continúes practicando.

Actividad	<p>De acuerdo con la solución presentada en la figura 6, y teniendo en cuenta que cuando se presenta un caso en el que hay dos o tres promedios iguales, no se realiza ninguna acción, debes retomar lo que vimos sobre operadores de comparación y complementar la solución del ejercicio.</p> <p>Fuera de realizar la operación y mostrar cuál promedio es mayor, debes mostrar también si se presenta igualdad en algunos de los promedios.</p> <p>Mostrando el mensaje que indique esto y además que muestre entre cuáles grupos se presenta el empate y los promedios iguales.</p>
------------------	---

Algo que puedes hacer, es tener un archivo en el cual guardes tus funciones, en algún momento las volverás a usar y como sabemos, son de gran importancia y utilidad en labores de programación.

En los siguientes videos, encontrarás una explicación adicional para el tema de funciones, que nos ayudará a profundizar en la manera de crearlas, diseñarlas, usarlas, aprovechando los beneficios que traen y la gran utilidad en el entorno de programación.



Video

- **Autor:** Programación ATS
- **Título:** 61. Programación en Python | Funciones | Funciones sin retorno de valor
- **URL:** <https://www.youtube.com/watch?v=v0TMIZOsZls>



Video

- **Autor:** Programación ATS
- **Título:** 62. Programación en Python | Funciones | Funciones con retorno de valor
- **URL:** <https://www.youtube.com/watch?v=vVFw1xO6fHU>



Video

- **Autor:** Programación ATS
- **Título:** 63. Programación en Python | Funciones | Argumentos y parámetros
- **URL:** <https://www.youtube.com/watch?v=B6PhYbmnIVQ>

La información que viste en los videos anteriores es de gran valor. Aunque en tu vida como programador, seguramente vas a trabajar también con otros lenguajes de programación, la esencia de las funciones y en general todo lo que hemos visto, será la misma, así se presenten variaciones en la sintaxis u otros aspectos, la funcionalidad y el contenido y la aplicabilidad serán siempre las mismas. Como ya lo hemos hablado, las funciones nos ayudan a optimizar, organizar y potenciar el desarrollo de nuestros programas.

Ejercicios de aplicación de funciones:

- En varias ocasiones hemos aplicado restricciones a nuestros programas.

Las restricciones son aquellas condiciones que nos ayudan a asegurarnos que, en caso de solicitar información de un usuario, éste sólo tenga la posibilidad de ingresar el dato que necesitamos. Si son números, nos aseguramos de que efectivamente sean datos numéricos lo que el usuario ingresó, establecerlos dentro de rangos determinados, etc.

Vamos a hacer una función que se llame número entero y la usaremos en un programa, que le pida al usuario que ingrese un número entero. En caso de que no ingrese el tipo de dato que necesitamos, debe mostrar un mensaje de advertencia.

- Hacer una función que retorne un número ingresado dentro de un rango recibido.
- Hacer una función que se llame portada, que muestre en pantalla la portada del trabajo y que como dato de entrada recibe el nombre del trabajo.
- La fórmula de la velocidad es: $V=d/t$.
- Hacer un programa, con una función propia, o sea de tu autoría, que ayude a los viajeros a calcular el tiempo que les falta para llegar a su destino.
- Se debe ingresar la distancia y una velocidad promedio. Nosotros debemos calcular el tiempo y mostrarlo con sus respectivas unidades de medida, ya sean horas, minutos o segundos.
- Hacer un programa, con una función propia, o sea de tu autoría, que pida al usuario que ingrese un número, (recuerda validar que el dato ingresado efectivamente sea un número. De lo contrario, debe mostrar un mensaje de advertencia) y mostrar si el número ingresado es par o impar y si es un número primo.
- La ley de Ohm nos dice que $V=R*I$. Voltaje es igual a resistencia por corriente.
- Hacer un programa, con una función propia, o sea de tu autoría, que pida al usuario que ingrese el valor del voltaje (v) y la resistencia (ohmios) y calcule la corriente (A) de un circuito eléctrico.
- En esta unidad, un poco más atrás, creaste un programa que se llamó bandas transportadoras, ahora vas a tomarlo de nuevo, pero en esta ocasión vas a crear una función que evalúe los valores de S1 y S2, y realice las mismas acciones que ya se hicieron. Ahora todo dentro de una función que pida los valores de los sensores. Recuerda que solo pueden ser 1 o 0.

Retomando algunos temas que en unidades anteriores sólo se mencionaron, vamos a entrar más en detalle, debido a que son pieza fundamental en el devenir de un programador. Es el caso de las listas, las tuplas, los diccionarios y varias instrucciones importantes para poder recorrer estos datos. Hacemos referencia a las estructuras iterativas de control.

Estos tipos de datos reciben el nombre de tipos de datos complejos, colecciones o arreglos. En estos casos no estamos hablando de un campo o espacio de memoria, que almacena un dato, si no que estamos hablando de campos o espacios de memoria más grandes, que almacenen varios datos, que están ubicados en posiciones específicas y se requiere de herramientas que faciliten la posibilidad de recorrer estas posiciones, con el objetivo de encontrar o realizar alguna acción con la información almacenada en algunas de estas ubicaciones.

Estos tipos de datos en otros lenguajes de programación se conocen como vectores. En nuestro caso de estudio, Python, los nombra listas, tuplas y diccionarios.

2. CONCEPTOS Y USO DE VECTORES

Vectores

Para analizar y clarificar sobre el concepto de vectores y sus aplicaciones, debemos resaltar que en Python no reciben este nombre. Este segmento de la unidad se nombró de esta manera, porque en los demás lenguajes de programación que vas a estudiar, reciben este nombre. En nuestro caso, se nombran diferente, pero su aplicabilidad y funcionalidad, son las mismas.

Para iniciar, los vectores o listas y tuplas son tipos de datos que se usan cuando se quiere agrupar elementos. Entre ambas existe una diferencia importante. En Python, las listas son mutables, o sea que se pueden modificar, ya sea toda la lista o alguno de sus elementos. Por el contrario, las tuplas son inmutables, lo que nos indica que después de creada la tupla, no se puede acceder a modificarla.

Listas

Para crear una lista debes tener en cuenta que debe tener un nombre, seguido del operador de asignación (=), luego, los datos que va a contener la lista. Estos datos van encerrados entre corchetes, usando comas para separar un dato del otro.

Como las listas contienen varios elementos, es importante mencionar que, en Python, se comienza a contar los elementos desde cero. Así, si tienes una lista con 4 elementos, puedes validarlos desde el 0 hasta el 3, completando ahí los 4 elementos, además en las listas podemos modificar los elementos que la componen. Mediante una instrucción propia de Python, podemos conocer la longitud de la lista.

La instrucción que nos permite conocer la longitud de una lista es *len*, para usarla, debemos tener en cuenta que cuando una función retorna algo, la manera de llamarla, para poder conocer ese dato que retorna, es mediante una instrucción de asignación, esto aplica para funciones propias de Python o funciones diseñadas por el programador.

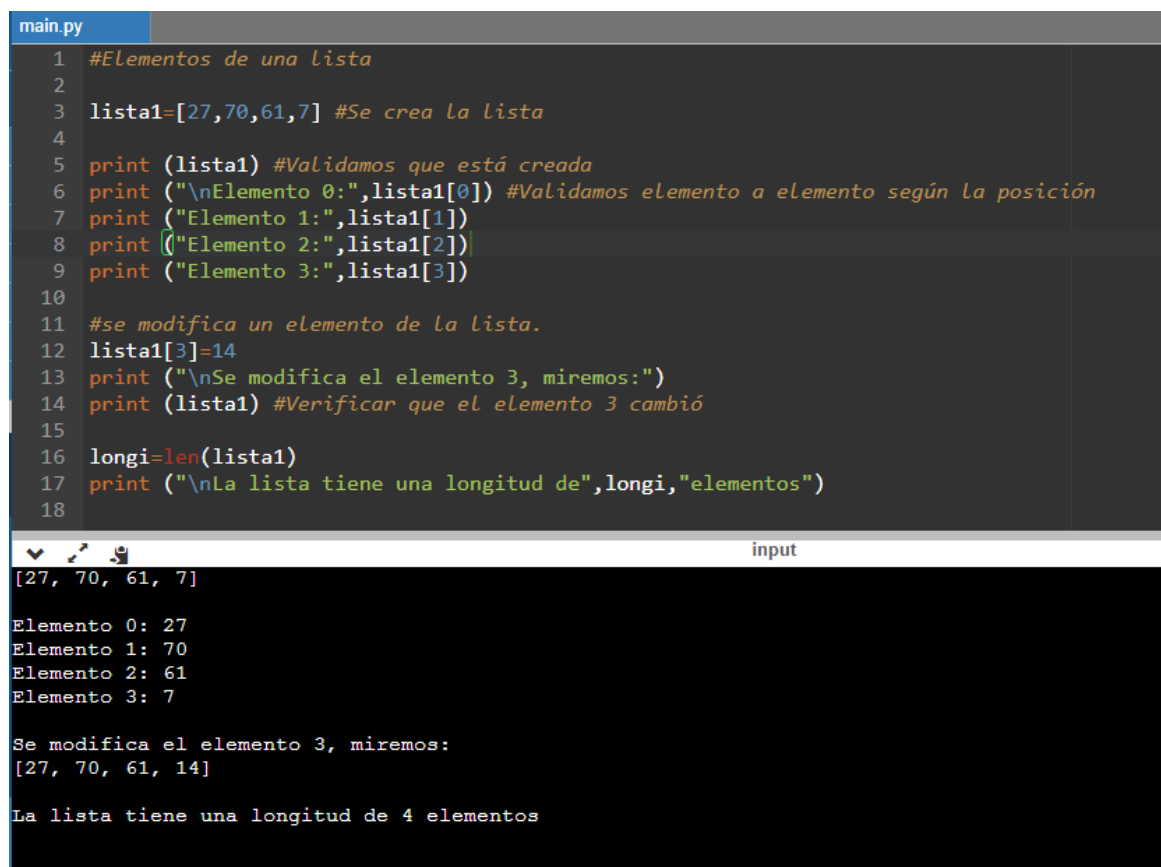
¿Para qué usamos las listas?

En la vida cotidiana, necesitamos describir o guardar información de alguna situación, acontecimiento o datos de alguna persona. Lo anterior requiere que la descripción que se haga contenga información que incluye más de un dato, debido a esto, se hace necesario el uso de tipos de datos que nos permitan almacenar y tener más detalle de la situación o acción que deseamos almacenar.

Por ejemplo, cuando se tiene el dato de una fecha, sirve poco conocer solamente el día. Sería un dato numérico que nos arroja poca información. En este caso es necesario conocer el día, el mes y el año, para poder ubicarnos mejor en el contexto que se requiera.

En la figura 7, observamos cómo se crea una lista y vamos a tomar elemento por elemento, para comprobar la numeración de estos, modificar y utilizar la instrucción *len* para conocer la longitud de la lista.

Figura 7. Elementos de una lista.



```

main.py
1  #Elementos de una lista
2
3  lista1=[27,70,61,7] #Se crea la lista
4
5  print (lista1) #Validamos que está creada
6  print ("\nElemento 0:",lista1[0]) #Validamos elemento a elemento según la posición
7  print ("Elemento 1:",lista1[1])
8  print ("Elemento 2:",lista1[2])
9  print ("Elemento 3:",lista1[3])
10
11 #se modifica un elemento de la lista.
12 lista1[3]=14
13 print ("\nSe modifica el elemento 3, miremos:")
14 print (lista1) #Verificar que el elemento 3 cambió
15
16 longi=len(lista1)
17 print ("\nLa lista tiene una longitud de",longi,"elementos")
18
input
[27, 70, 61, 7]

Elemento 0: 27
Elemento 1: 70
Elemento 2: 61
Elemento 3: 7

Se modifica el elemento 3, miremos:
[27, 70, 61, 14]

La lista tiene una longitud de 4 elementos
  
```

Fuente: Elaboración propia.

En el trabajo con listas, llegará el momento en el que surge la necesidad de verificar si un elemento se encuentra dentro de la lista y también se hará importante conocer su posición dentro del arreglo, además vamos a necesitar la posibilidad de insertar nuevos datos, al final del arreglo y en posiciones específicas y por último vamos a conocer como eliminar algún elemento del arreglo.

Las instrucciones *in*, *index*, *insert*, *append* y *remove* nos pueden ayudar en esta labor.

Todos los elementos y los temas que estamos analizando en clase, son importantes para la realización del proyecto integrador y para tener las herramientas necesarias al momento de abordar otras asignaturas. ¡¡Ánimo!!

Vamos a realizar un ejemplo, el cual podemos ver en la figura 8, para que miremos de manera codificada el uso de estas dos instrucciones.

Para la realización de este ejercicio, vamos a tener en cuenta que debemos crear una lista con los nombres de los integrantes del equipo de contabilidad de una empresa.

La compañera Jessica, entró hace poco y debemos validar si ya está incluida en la lista y además conocer la posición de su nombre dentro del arreglo.

Hay dos miembros de la compañía que estarán en esta área durante un mes en período de prueba. Estos dos compañeros se llaman Jorge y Daniel.

Vamos a agregar a Jorge en la posición 2 del arreglo y a Daniel al final.

Alejandra ya no estará más en la empresa, por lo tanto, debemos borrarla del arreglo.

En la solución del ejercicio que veremos en la Figura 8, vamos a tener paso a paso cada una de las acciones que nos acaban de describir, de manera que logremos identificar la funcionalidad y la diferencia entre usar cada instrucción.

En las listas, cuando vamos a ingresar datos que no sean numéricos, debemos hacerlo entre comillas, separados por comas.

Figura 8.
Instrucciones *in*, *index*, *insert*, *append* y *remove*

```

main.py
1  #Elementos de una lista
2
3  nombres=["Jose", "Camilo", "Jessica", "Alejandra", "Dany"] #Se crea la lista
4  #Validar si Jessica ya existe y conocer su posición en la lista
5  print ("Jessica" in nombres)
6  print (nombres.index ("Jessica"))
7
8  nombres.insert(2,"Jorge") #Se agrega Jorge en la posición 2
9  nombres.append("Daniel") #Se agrega Daniel al final
10 print(nombres)
11
12 nombres.remove ("Alejandra") #Se elimina Alejandra
13 print(nombres)
14

```

input

```

True
2
['Jose', 'Camilo', 'Jorge', 'Jessica', 'Alejandra', 'Dany', 'Daniel']
['Jose', 'Camilo', 'Jorge', 'Jessica', 'Dany', 'Daniel']

```

Fuente: Elaboración propia

En la parte inferior de la figura 8, donde se observa el resultado de la ejecución de cada instrucción, vemos que se da una respuesta verdadera *True* cuando preguntamos si Jessica existe dentro del arreglo, luego nos muestra que está en la posición 2. Recordemos que comienza a contar desde 0.

Luego, con la instrucción *insert*, podemos decir en qué posición necesitamos que ingrese el nuevo elemento. Posterior con la instrucción *append*, agregamos un nuevo elemento, pero al final del arreglo. Mostramos en pantalla estas modificaciones.

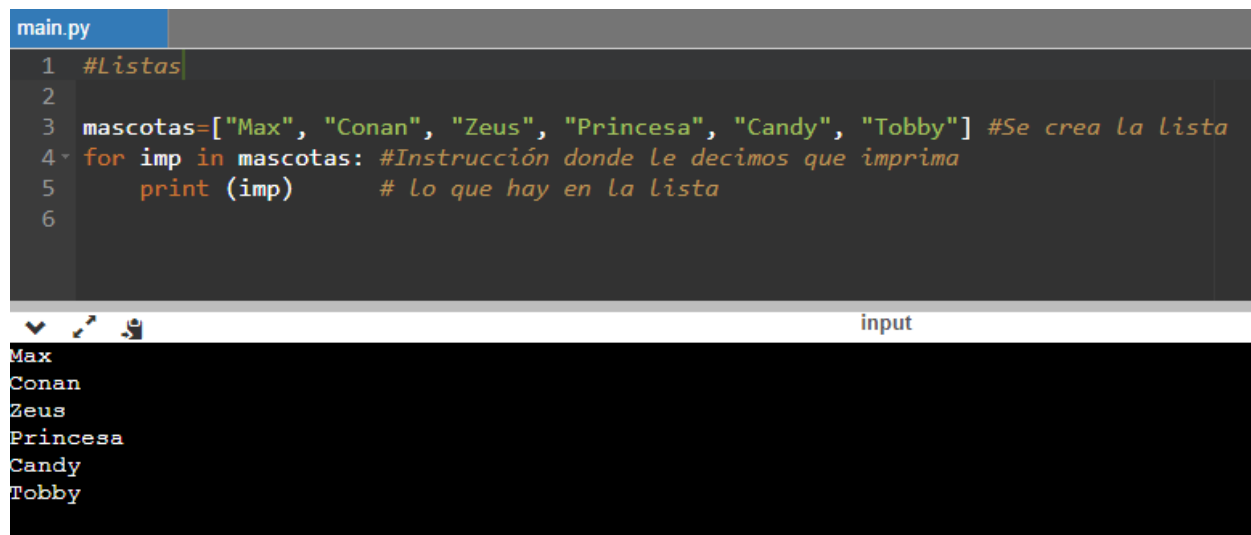
Luego nos indican que Alejandra ya no hace parte de la compañía, entonces debemos sacar su nombre del arreglo, por lo tanto, con la instrucción *remove*, mencionamos cuál elemento debemos eliminar, se realiza esta acción y de nuevo mostramos en pantalla el resultado. Dando cumplimiento a los requerimientos del planteamiento del ejercicio.

No olvidemos que siempre es importante documentar el programa.

A continuación, vamos a relacionar una de las estructuras iterativas de control, el *for*, con el tema que estamos tratando.

Ahora crearemos una lista, esta vez lo haremos con nombres de mascotas y vamos a imprimir o mostrar en pantalla uno a uno, usando una instrucción iterativa de control. En la figura 9, podemos observar la solución de este ejercicio.

Figura 9.
Listas y estructuras iterativas de control



```
main.py
1 #Listas
2
3 mascotas=["Max", "Conan", "Zeus", "Princesa", "Candy", "Tobby"] #Se crea la lista
4 for imp in mascotas: #Instrucción donde le decimos que imprima
5     print (imp)      # lo que hay en la lista
6
```

input

Max
Conan
Zeus
Princesa
Candy
Tobby

Fuente: Elaboración propia

Tuplas

Como se mencionó anteriormente, las tuplas no son modificables, o, dicho de otra manera, son inmutables. Por lo tanto, para estas no aplican varias de las funciones que vimos en las listas, que conllevan a la posibilidad de modificar el contenido de la lista.

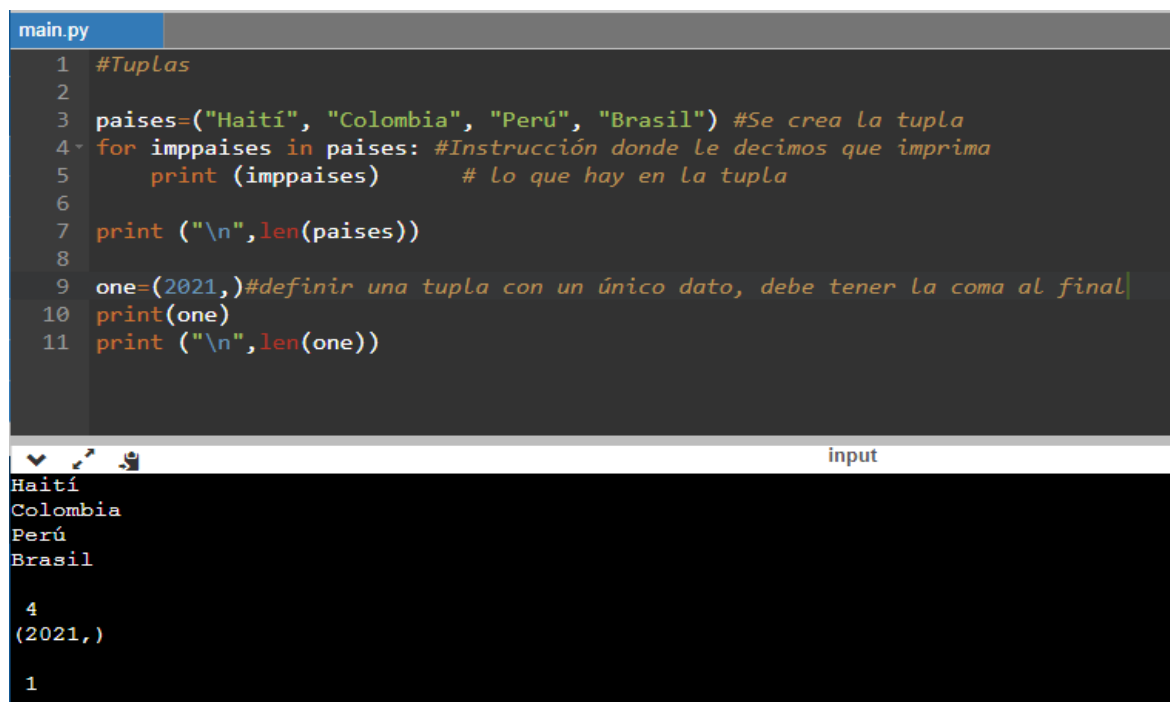
Al igual que en las listas, en las tuplas, el número de elementos se empieza a contar desde la posición 0, y podemos validar cada elemento con el nombre de la tupla y entre corchetes el número del elemento que deseamos validar.

En la notación la diferencia entre tuplas y lista, parte de que las tuplas se nombran y el contenido de la misma va entre paréntesis. Estos nos están diciendo que estamos definiendo una tupla y por lo tanto su contenido es inmutable.

Una tupla se puede conformar de un solo elemento. En ese caso es importante nombrarla de la manera correcta. En la figura 10, podemos observar cómo podemos definir una tupla de varios elementos y una que contenga un único elemento.

Cuando creas una tupla con un único dato, debes recordar que las tuplas se definen entre paréntesis y debes tener en cuenta también que debes ubicar una coma inmediatamente después del único dato que defines. Si no se ubica esta coma, al momento de ejecutar el programa y realizar alguna acción con la tupla te saldrá un error, que posiblemente puede ponerte en aprietos para encontrarlo.

Figura 10. Tuplas



```

main.py
1  #Tuplas
2
3  países=("Haití", "Colombia", "Perú", "Brasil") #Se crea la tupla
4  for impaíses in países: #Instrucción donde le decimos que imprima
5      print (impaíses)      # lo que hay en la tupla
6
7  print ("\n",len(países))
8
9  one=(2021,)#definir una tupla con un único dato, debe tener la coma al final
10 print(one)
11 print ("\n",len(one))

```

input

```

Haití
Colombia
Perú
Brasil

4
(2021,)

1

```

Fuente: Elaboración propia

Según el programa que estés haciendo, puede llegar la necesidad de agrupar en una tupla varios datos sueltos que se tengan. Por eso se tiene la opción de empaquetado y desempaqueado de tuplas. Para realizar el desempaqueado, se debe primero conocer la longitud de la tupla y asignarla a un mismo número de variables. En la figura 11, podemos ver cómo se realiza este proceso.

Figura 11. Empaquetado y desempaquetado de tuplas

```

main.py
1  #Empaquetado de tuplas.
2  nom="Camila" #Campos de un solo dato
3  edad=25
4  eps="sura"
5  empaq=nom,edad,eps #se asignan a una variable para crear una tuplas
6  print("longitud de la tupla:",len(empaq)) #Mostrar longitud de la tupla
7  print(empaq)#mostrar tupla en pantalla
8  print ("\n")
9  #Desempaquetado de tuplas
10 tup=(2021,2019,2000) #se tiene la tupla y la mostramos en pantalla
11 print (tup)
12 d1,d2,d3=tup #numero de variables igual al numero de elementos y se muestran
13 print (d1)
14 print (d2)
15 print (d3)
16

```

input

```

longitud de la tupla: 3
('Camila', 25, 'sura')

(2021, 2019, 2000)
2021
2019
2000

```

Fuente: Elaboración propia

3. CONCEPTOS Y USO DE MATRICES

Matrices

Las matrices en programación son estructuras grandes de datos que permiten almacenar información. En Python se conocen y se trabajan bajo el nombre de diccionarios.

Los diccionarios son otro tipo de dato importante en Python. a continuación.

¿Sabes cómo se construyen los diccionarios?

Entre las estructuras de datos que hemos analizado en este curso una de las más icónicas es la estructura diccionario, en esta guía conoceremos los elementos esenciales que la componen.

A continuación, analizaremos las nociones básicas de los diccionarios de Python, aprenderemos sus usos más comunes y algunos de sus métodos más importantes.

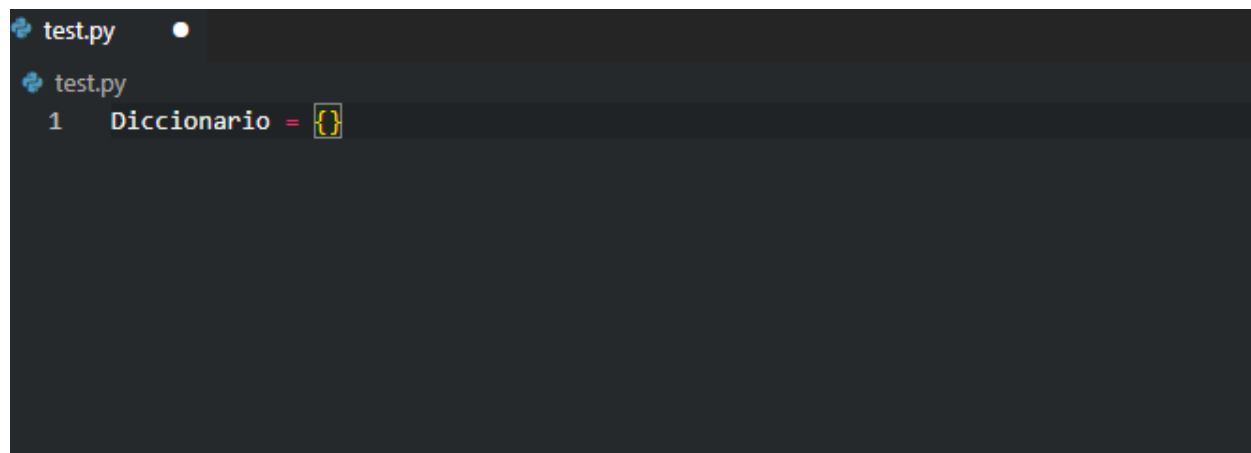
Los **diccionarios en el lenguaje de programación Python** son un tipo de estructura de datos capaz de almacenar información en un conjunto desordenado de pares clave-valor, dichas claves son únicas dentro del mismo diccionario, lo que quiere decir que cada clave es única dentro del arreglo.

La clave o key es el elemento de la estructura diccionario que nos permite identificar cada uno de los elementos que tienen nuestro diccionario. Y el valor es el dato o conjunto de datos asociados a la clave.

Para definir un diccionario se requiere encerrar el listado de valores que deseamos almacenar entre llaves. Las parejas de clave y valor se separaran con comas, y posteriormente la clave y el valor estarán separadas por dos puntos como analizaremos en el siguiente paso a paso.

Paso 1:

Escribiremos el nombre del diccionario, un signo de igual (=) y abriremos y cerraremos llaves, como se muestra en la siguiente imagen.

A screenshot of a code editor window titled 'test.py'. The editor shows a single line of code: '1 Diccionario = {}'. The text is in a light blue font on a dark background. The cursor is positioned at the end of the line, after the closing curly brace.

Fuente: Elaboración propia

Paso 2:

Dentro de las llaves escribimos el nombre de nuestra clave dentro de comillas simples, posterior el símbolo de dos puntos (":") y luego escribimos nuestros elementos con base al tipo de dato que queremos almacenar con nuestra llave, como se muestra en la siguiente imagen.

```

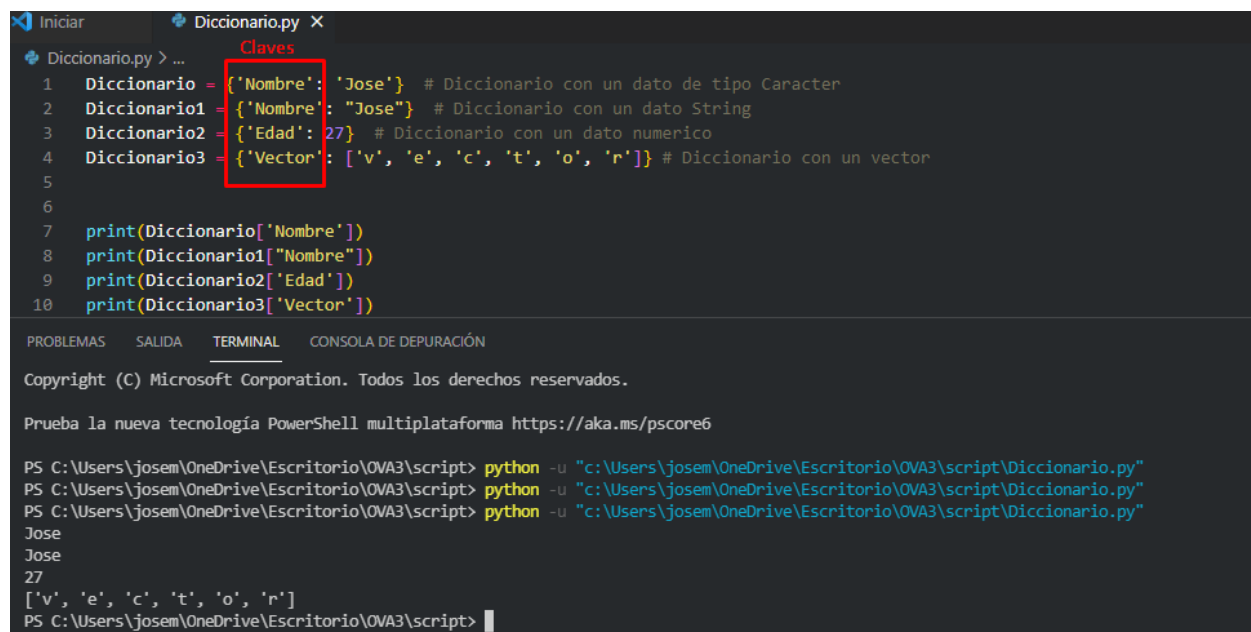
test.py
test.py > ...
1  Diccionario = {'Nombre': 'Jose'} # Diccionario con un dato de tipo Caracter
2  Diccionario1 = {'Nombre': "Jose"} # Diccionario con un dato String
3  Diccionario2 = {'Edad': 27} # Diccionario con un dato numerico
4  Diccionario2 = {'Edad': ['v','e','c','t','o','r']} #Diccionario con un vector

```

Fuente: Elaboración propia

Acceso a diccionarios:

Para acceder a los elementos de un diccionario debemos escribir la palabra reservada print y luego la clave del elemento como podremos observar a continuación.



```

Diccionario.py X
Diccionario.py > ...
1  Diccionario = {'Nombre': 'Jose'} # Diccionario con un dato de tipo Caracter
2  Diccionario1 = {'Nombre': "Jose"} # Diccionario con un dato String
3  Diccionario2 = {'Edad': 27} # Diccionario con un dato numerico
4  Diccionario3 = {'Vector': ['v','e','c','t','o','r']} # Diccionario con un vector
5
6
7  print(Diccionario['Nombre'])
8  print(Diccionario1["Nombre"])
9  print(Diccionario2['Edad'])
10 print(Diccionario3['Vector'])

```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma <https://aka.ms/pscore6>

```

PS C:\Users\josem\OneDrive\Escritorio\OVA3\script> python -u "c:\Users\josem\OneDrive\Escritorio\OVA3\script\Diccionario.py"
PS C:\Users\josem\OneDrive\Escritorio\OVA3\script> python -u "c:\Users\josem\OneDrive\Escritorio\OVA3\script\Diccionario.py"
PS C:\Users\josem\OneDrive\Escritorio\OVA3\script> python -u "c:\Users\josem\OneDrive\Escritorio\OVA3\script\Diccionario.py"
Jose
Jose
27
['v', 'e', 'c', 't', 'o', 'r']
PS C:\Users\josem\OneDrive\Escritorio\OVA3\script>

```

Fuente: Elaboración propia

También podemos acceder a cada una de las posiciones de un vector contenido en un diccionario de la siguiente manera como se muestra a continuación.

```

Diccionario.py > ...
1  Diccionario = {'Nombre': 'Jose'} # Diccionario con un dato de tipo Caracter
2  Diccionario1 = {'Nombre': "Jose"} # Diccionario con un dato String
3  Diccionario2 = {'Edad': 27} # Diccionario con un dato numerico
4  Diccionario3 = {'Vector': ['v', 'e', 'c', 't', 'o', 'r']} # Diccionario con un vector
5
6
7
8  print(Diccionario3['Vector'][0]) #imprime la posicion 0 de nuestro vector en este caso 'v'

```

PROBLEMAS SALIDA **TERMINAL** CONSOLA DE DEPURACIÓN

PS C:\Users\josem\OneDrive\Escritorio\OVA3\script> & C:/Users/josem/AppData/Local/Programs/Python/Python310/python.exe c:/Users/josem/OneDrive/Escritorio/OVA3/script/Diccionario.py

v

PS C:\Users\josem\OneDrive\Escritorio\OVA3\script> █

Fuente: Elaboración propia

También podemos acceder a cada una de las posiciones de un diccionario de varios elementos de la siguiente manera como se muestra a continuación.

```

Diccionario.py > ...
1  Diccionario = {'Nombre': 'Jose', 'Edad': 3, 'Año': 2021} # Diccionario con un dato de tipo Caracter
2
3  print(Diccionario['Edad']) # Respuesta : Jose
4  print(Diccionario['Nombre']) # Respuesta : 3
5  print(Diccionario['Año']) # Respuesta : 2021
6

```

PROBLEMAS SALIDA **TERMINAL** CONSOLA DE DEPURACIÓN

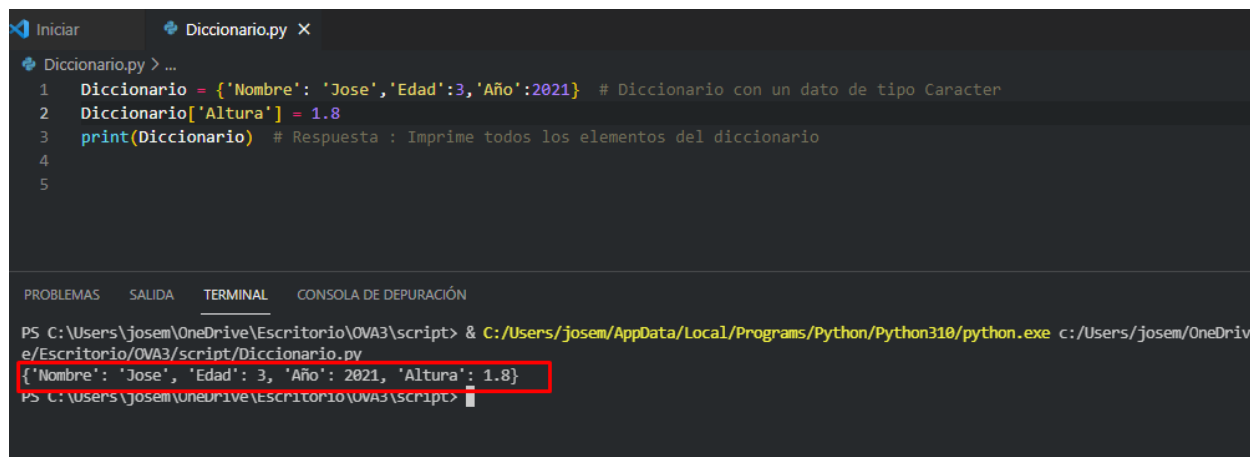
PS C:\Users\josem\OneDrive\Escritorio\OVA3\script> & C:/Users/josem/AppData/Local/Programs/Python/Python310/python.exe c:/Users/josem/OneDrive/Escritorio/OVA3/script/Diccionario.py

3
Jose
2021

PS C:\Users\josem\OneDrive\Escritorio\OVA3\script> █

Fuente: Elaboración propia

Luego podemos realizar la inserción de elementos al diccionario y la posterior visualización de los mismo como se muestra a continuación.



```

Diccionario.py > ...
1  Diccionario = {'Nombre': 'Jose', 'Edad': 3, 'Año': 2021} # Diccionario con un dato de tipo Caracter
2  Diccionario['Altura'] = 1.8
3  print(Diccionario) # Respuesta : Imprime todos los elementos del diccionario
4
5

PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

PS C:\Users\josem\OneDrive\Escritorio\OVA3\script> & C:/Users/josem/AppData/Local/Programs/Python/Python310/python.exe c:/Users/josem/OneDrive/Escritorio/OVA3/script/Diccionario.py
{'Nombre': 'Jose', 'Edad': 3, 'Año': 2021, 'Altura': 1.8}
PS C:\Users\josem\OneDrive\Escritorio\OVA3\script>

```

Fuente: Elaboración propia

Acceso a diccionarios usando la estructura For

Otra manera de imprimir el contenido de los diccionarios es usando la estructura For como se muestra a continuación.



```

Diccionario.py > ...
1  Diccionario = {'Nombre': 'Jose', 'Edad': 3, 'Año': 2021, 'Altura': 1.8}
2
3  for key in Diccionario:
4      print (key, ":", Diccionario[key])
5
6

PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

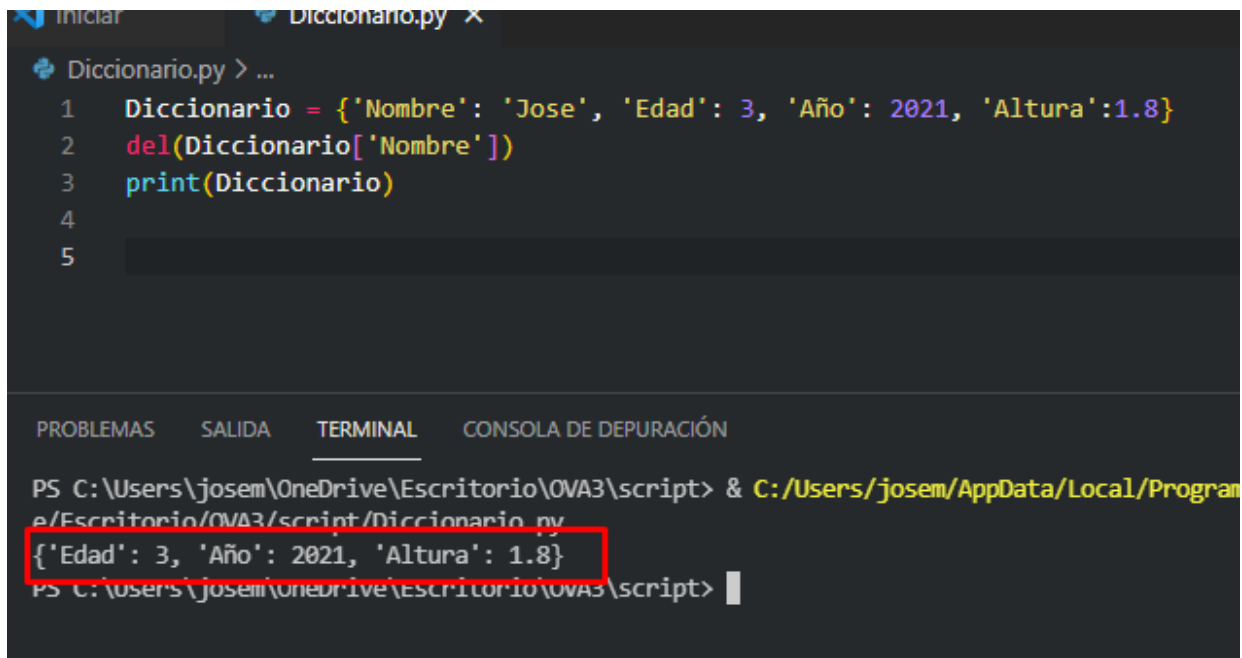
PS C:\Users\josem\OneDrive\Escritorio\OVA3\script> & C:/Users/josem/AppData/Local/Programs/Python/Python310/python.exe c:/Users/josem/OneDrive/Escritorio/OVA3/script/Diccionario.py
Nombre : Jose
Edad : 3
Año : 2021
Altura : 1.8
PS C:\Users\josem\OneDrive\Escritorio\OVA3\script>

```

Fuente: Elaboración propia

Eliminar elementos del diccionario:

Para eliminar elementos de nuestro diccionario debemos utilizar el comando “del(NombreDelDiccionario[‘NombreDeLaClave’])” de la manera como se muestra a continuación:



```

Diccionario.py > ...
1  Diccionario = {'Nombre': 'Jose', 'Edad': 3, 'Año': 2021, 'Altura':1.8}
2  del(Diccionario['Nombre'])
3  print(Diccionario)
4
5
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

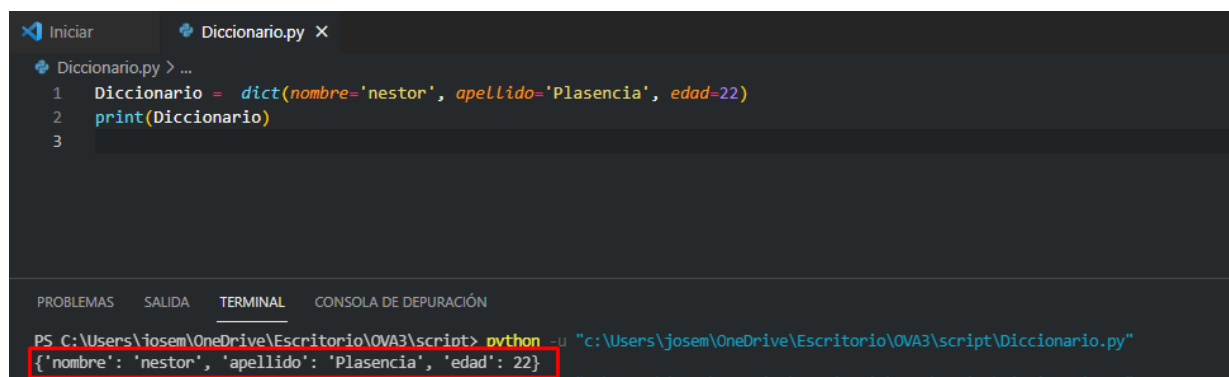
PS C:\Users\josem\OneDrive\Escritorio\OVA3\script> & C:/Users/josem/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe C:\Users\josem\OneDrive\Escritorio\OVA3\script\Diccionario.py
{'Edad': 3, 'Año': 2021, 'Altura': 1.8}
PS C:\Users\josem\OneDrive\Escritorio\OVA3\script>

```

Fuente: Elaboración propia

Método dict:

El método dict se utiliza para convertir un conjunto de datos separados por coma en un diccionario como se muestra a continuación.



```

Diccionario.py > ...
1  Diccionario = dict(nombre='nestor', apellido='Plasencia', edad=22)
2  print(Diccionario)
3
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

PS C:\Users\josem\OneDrive\Escritorio\OVA3\script> python -u "c:\Users\josem\OneDrive\Escritorio\OVA3\script\Diccionario.py"
{'nombre': 'nestor', 'apellido': 'Plasencia', 'edad': 22}
PS C:\Users\josem\OneDrive\Escritorio\OVA3\script>

```

Fuente: Elaboración propia

Método zip:

El método zip recibe dos elementos, ya sean una cadena, una lista o una tupla. Ambos parámetros deben tener el mismo número de elementos. Se devolverá un diccionario relacionando cada una de las posiciones de los elementos una a una como se muestra a continuación:

```

1  Diccionario = dict(zip('abcd',[1,2,3,4]))
2  print(Diccionario)
3

```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```

PS C:\Users\josem\OneDrive\Escritorio\OVA3\script> python -u "c:\Users\josem\OneDrive\Escritorio\OVA3\script\Diccionario.py"
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
PS C:\Users\josem\OneDrive\Escritorio\OVA3\script>

```

Fuente: Elaboración propia

Métodos items, keys y values:

- El método items devuelve una lista de duplas, en las cuales el primer valor es la llave y el segundo elemento es el dato asignado a la llave anteriormente mencionada.
- El método key devuelve todas las llaves que contienen un diccionario.
- El método values devuelve cada uno de los valores que tienen asignadas las llaves de un diccionario.

```

1  Diccionario = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
2  elementos = Diccionario.items()
3  claves = Diccionario.keys()
4  Valores = Diccionario.values()
5  print(elementos)
6  print(claves)
7  print(Valores)
8

```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```

PS C:\Users\josem\OneDrive\Escritorio\OVA3\script> python -u "c:\Users\josem\OneDrive\Escritorio\OVA3\script\Diccionario.py"
dict_items([('a', 1), ('b', 2), ('c', 3), ('d', 4)])
dict_keys(['a', 'b', 'c', 'd'])
dict_values([1, 2, 3, 4])
PS C:\Users\josem\OneDrive\Escritorio\OVA3\script>

```

Fuente: Elaboración propia

El tema de los diccionarios es un tema amplio, pero con mucha importancia dentro del contexto de la programación, pues de manera frecuente nos podemos encontrar con problemas en los que se deba incluir esta herramienta como parte de la solución.

Para profundizar más en el tema de vectores, matrices y en general ampliar el contexto de uso y conocimiento de la herramienta que es nuestro objeto de estudio, te invitamos a que ingreses y estudies con atención el siguiente documento:



Para aprender más

- Si deseas conocer más sobre vectores, matrices y diccionarios, lee el siguiente material:
- **Libro electrónico:** Algoritmos y programación I. Aprendiendo a programar usando Python como herramienta.
- **Sección a consultar:** Unidades 7, 8 y 9.
- **URL:** <http://materias.fi.uba.ar/7501/apunte%20PYTHON.pdf>

Al consultar en el libro que te acabamos de recomendar, verás que es de gran ayuda conocer y dominar mejor estos temas, ya que sus presentaciones dentro del mundo de la programación nos dan una perspectiva más amplia del manejo y el almacenamiento de la información.

4. Ejercicios de aplicación vectores y matrices:

Hacer un programa que le muestre al usuario un menú con las siguientes opciones:

1. Ingresar mascota
2. Modificar datos de una mascota
3. Modificar datos de una mascota.
4. Borrar una mascota,
5. Insertar una mascota.
6. Listar las mascotas.
7. Salir.

Para lo anterior, se debe crear un arreglo que contenga:

- Número de identificación.
- Nombre de la mascota.
- Sexo
- Edad
- Teléfono del propietario.

Los datos de las mascotas iniciales serán los siguientes:

- 101, Susy, F, 4, 12233445
- 1215, Sacha, F, 4, 300112513
- 23, Max, M, 2, 810442
- 900, Pitufu, M, 6, 151515151
- 2, Dante, M, 1, 6765440.

De acuerdo con la opción digitada por el usuario, se debe activar una función propia, o sea creada por ti, donde se pueda realizar la opción deseada.

El usuario podrá ingresar cuantas mascotas más desee. No pueden ser menos de 3 mascotas nuevas ingresadas.

Recuerda que la opción para que se deje de ejecutar el programa, es la opción 7. Mientras el usuario no digite esta opción, el programa debe continuar en ejecución. Cuando el usuario digite esta opción, se debe mostrar en pantalla un mensaje de agradecimiento por usar nuestros servicios. y finalizar la ejecución del programa.

Cierre

¡Felicidades!

Acabas de finalizar el contenido de tu asignatura fundamentos de programación. Seguramente fueron muchos temas nuevos, pero estamos seguros de que serán de gran ayuda para tu crecimiento personal y profesional.

La programación nos muestra un enfoque diferente de la computación, donde logramos comprender de mejor manera la creación y el funcionamiento de la máquina, desde la perspectiva del *software*. Esperamos que todo lo trabajado haya sido de tu agrado y te invitamos a continuar con motivación y disciplina, a seguir trabajando por tus sueños, a movilizarte, no quedarte estático y buscar cada día ser mejor. Nunca es tarde para lograr lo que te propones. Muchas gracias por hacer parte de la familia IU Digital de Antioquia y del programa de ingeniería mecatrónica.



Lecturas y Material Complementario

Te invitamos a explorar el siguiente material para que amplíes los temas estudiados hasta este momento. Lee y analiza cada uno de ellos, pues contienen información que te será de gran ayuda en el desarrollo del curso.

Lecturas:

- **Título:** Algoritmos y programación I. Aprendiendo a programar usando Python como herramienta.
- **Autor:** Rosita Wachenchauzer
- **Sección para consultar:** Unidades 7, 8 y 9.
- **URL:** <http://materias.fi.uba.ar/7501/apunte%20PYTHON.pdf>


Videos

- **Título:** 61. Programación en Python | Funciones | Funciones sin retorno de valor
- **Autor:** Programación ATS
- **URL:** <https://www.youtube.com/watch?v=v0TMIZOsZls>
- **Título:** 62. Programación en Python | Funciones | Funciones con retorno de valor
- **Autor:** Programación ATS
- **URL:** <https://www.youtube.com/watch?v=vVFw1xO6fHU>
- **Título:** 63. Programación en Python | Funciones | Argumentos y parámetros
- **Autor:** Programación ATS
- **URL:** <https://www.youtube.com/watch?v=B6PhYbmnlVQ>



Bibliografía

- *Objetos Diccionarios — documentación de Python - 3.8.12.* (s. f.). Python.org. Recuperado 8 de noviembre de 2021, de <https://docs.python.org/es/3.8/c-api/dict.html>
- Wachenchauzer, R., Manterola, M., Curia, M., Medrano, M., & Paez, N. (2013, 27 mayo). *Algoritmos y Programación I: Aprendiendo a programar usando Python como herramienta*. UBA. Recuperado 8 de noviembre de 2021, de <http://materias.fi.uba.ar/7501/apunte%20PYTHON.pdf>
- Wachenchauzer, R. (2012). *Algoritmos y programación I. Aprendiendo a programar usando Python como herramienta*. Disponible en: <http://materias.fi.uba.ar/7501/apunte%20PYTHON.pdf>



Esta licencia permite a otros distribuir, remezclar, retocar, y crear a partir de esta obra de manera no comercial y, a pesar que sus nuevas obras deben siempre mencionar a la IU Digital y mantenerse sin fines comerciales, no están obligados a licenciar obras derivadas bajo las mismas condiciones.



IUDigital
de Antioquia
INSTITUCIÓN UNIVERSITARIA
DIGITAL DE ANTIOQUIA