

Variables y tipos de datos

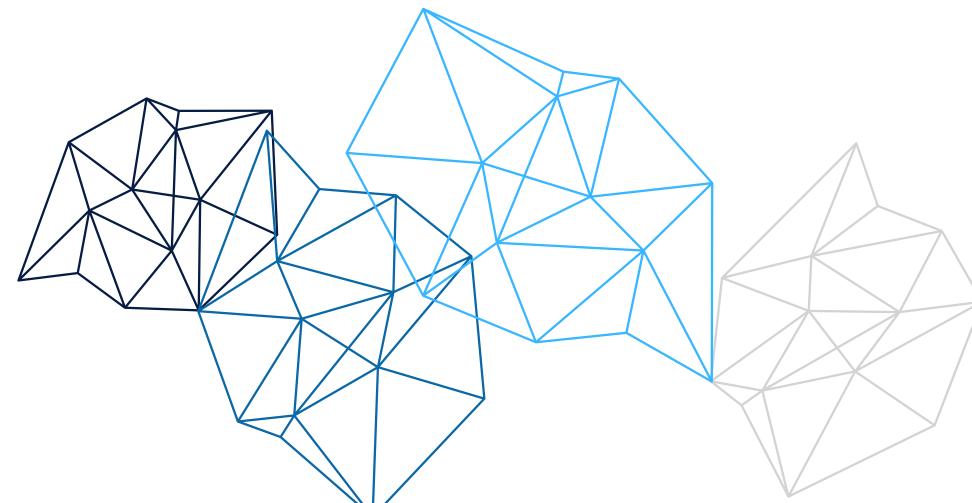
Las variables nos permiten guardar valores y reutilizarlos en distintos lugares del código, nos permite acceder fácilmente a un dato para ser manipulado y transformado a lo largo de un programa.

Para definir una variable en Python, basta con nombrarla dentro del entorno y definir un valor ya que como dijimos anteriormente es un lenguaje completamente orientado a objetos, por lo que no es necesario declarar variables o el tipo antes de usarlas como en otros lenguajes.

```
<nombre_variable> = <valor>  
    >>x = 10
```

INFORMATARIO
Hacia una mejor industria informática

En los ejercicios que desarrollemos a lo largo del curso las variables serán la forma de identificar, de forma sencilla, un dato que se encuentra almacenado en la memoria de la computadora por lo que debemos darle nombre que sean significativos.



Tipos de datos

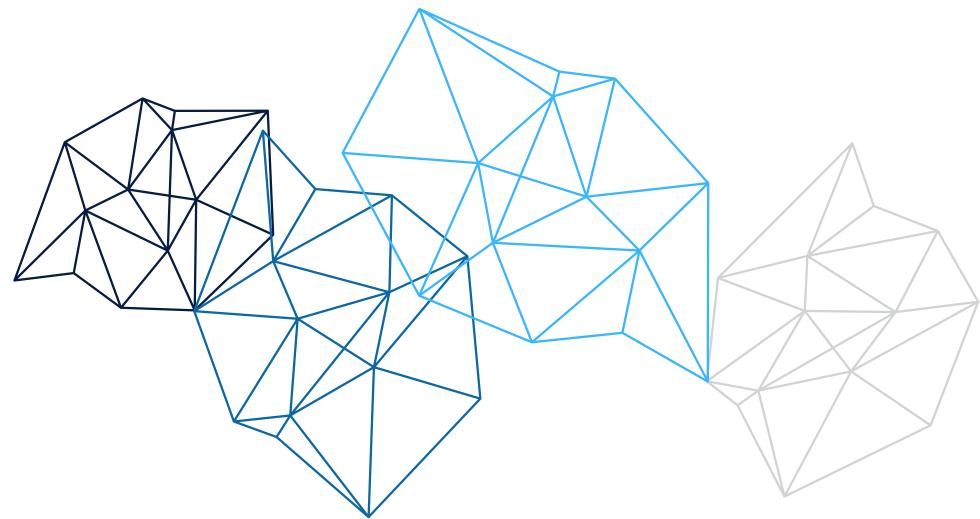
En Python, los tipos de datos básicos incluyen:

- **Números (int, float, complex)**
- **Booleanos (True, False)**
- **Cadenas de caracteres (str)**

Además de estos, existen otros tipos de datos en Python, como **listas, tuplas, diccionarios y conjuntos**.

En Python, no es necesario declarar el tipo de una variable antes de utilizarla. Cuando se asigna un valor a una variable, Python infiere el tipo de datos automáticamente.

<https://docs.python.org/es/3/reference/index.html>



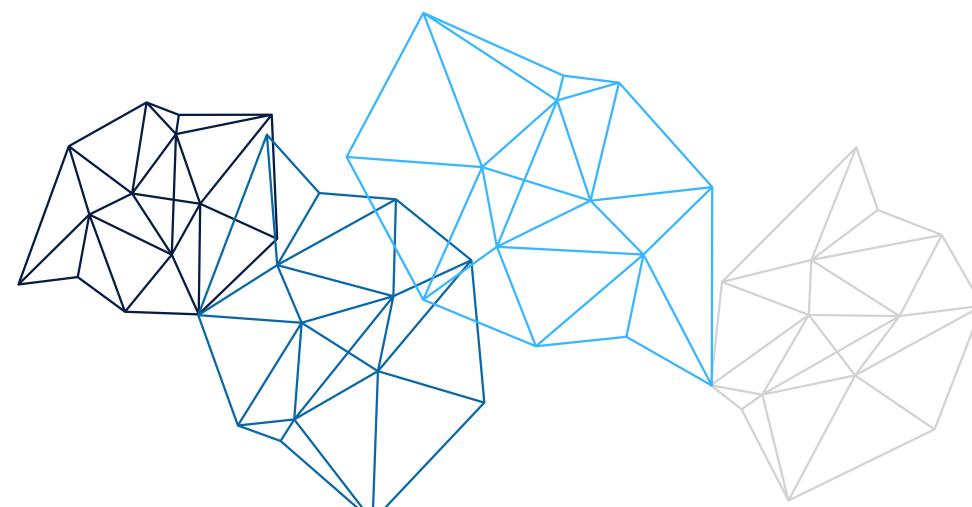
Enteros o int

Los tipos enteros o int en Python permiten almacenar un valor numérico no decimal ya sea positivo o negativo de cualquier valor. La función type() nos devuelve el tipo de la variable, y podemos ver como efectivamente es de la clase int.

```
>>i = 12  
>>print(i)          #12  
>>print(type(i))   #<class 'int'>  
>>x = 250**250  
>>print(type(x))   #<class 'int'>
```



INFORMATORIO
Hacia una mejor industria informática

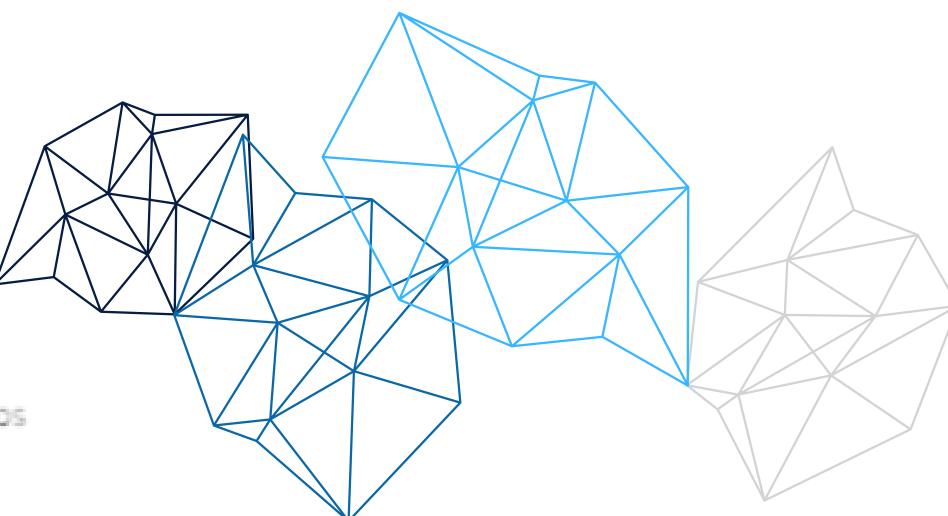


Convertir a int

El posible convertir a int otro tipo. Como hemos explicado, el tipo int no puedo contener decimales, por lo que si intentamos convertir un número decimal, se truncará todo lo que tengamos a la derecha de la coma.(Truncar significa cortar o acortar algo. En matemáticas, truncar un número significa eliminar sus cifras decimales y dejar solo el número entero más cercano.)

```
>>b = int(1.6)  
>>print(b)    #1
```

En otros lenguajes de programación, los int tenían un valor máximo que pueden representar. Una gran ventaja de Python es que ya no nos tenemos que preocupar de esto, ya que por debajo se encarga de asignar más o menos memoria al número, y podemos representar prácticamente cualquier número.



Float

El tipo numérico **float** permite representar un número positivo o negativo con decimales, es decir, números reales.

Por lo tanto si declaramos una variable y le asignamos un valor decimal, por defecto la variable será de tipo **float**.

```
>>f = 0.10093
```

```
>>print(f)      #0.10093
```

```
>>print(type(f)) #<class 'float'>
```



INFORMATORIO
Hacia una mejor industria informática



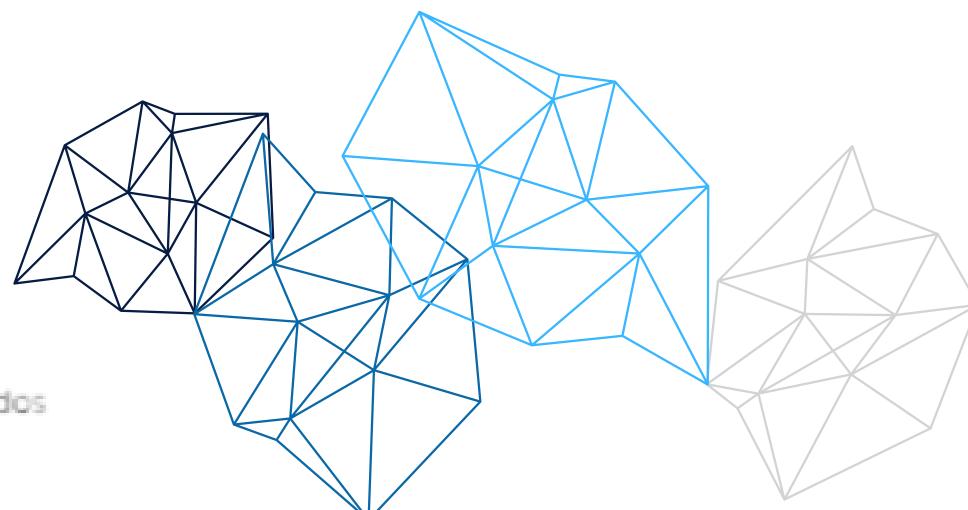
Subsecretaría de
Empleo



Ministerio de
Producción, Industria y Empleo



CHACO
Gobierno de todos



Booleanos

Al igual que en otros lenguajes de programación, en Python existe el tipo bool o booleano. Es un tipo de dato que permite almacenar dos valores True o False.

Declarar variable booleana

Se puede declarar una variable booleana de la siguiente manera.

```
>>x = True
```

```
>>y = False
```

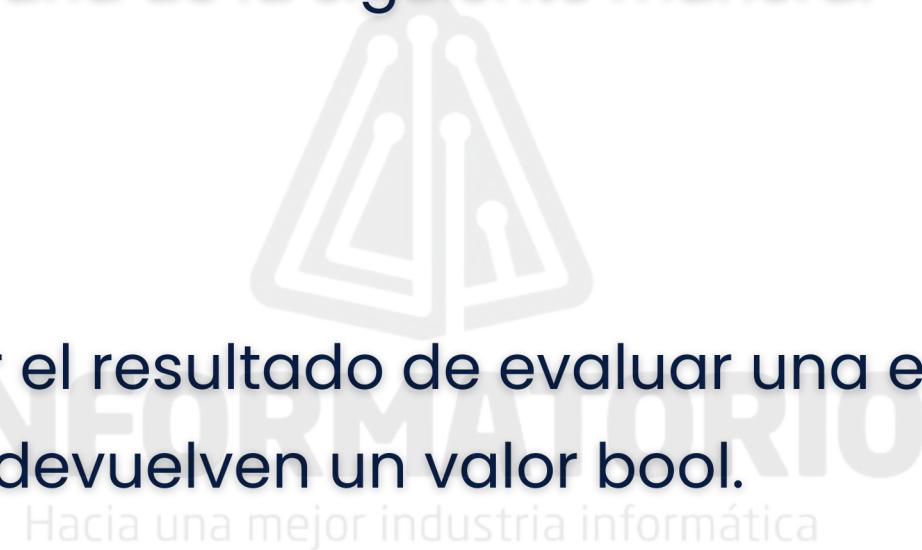
Evaluar expresiones

Un valor booleano también puede ser el resultado de evaluar una expresión. Ciertos operadores como el mayor que, menor que o igual que devuelven un valor bool.

```
>>print(1 > 0) #True
```

```
>>print(1 <= 0) #False
```

```
>>print(9 == 9) #True
```



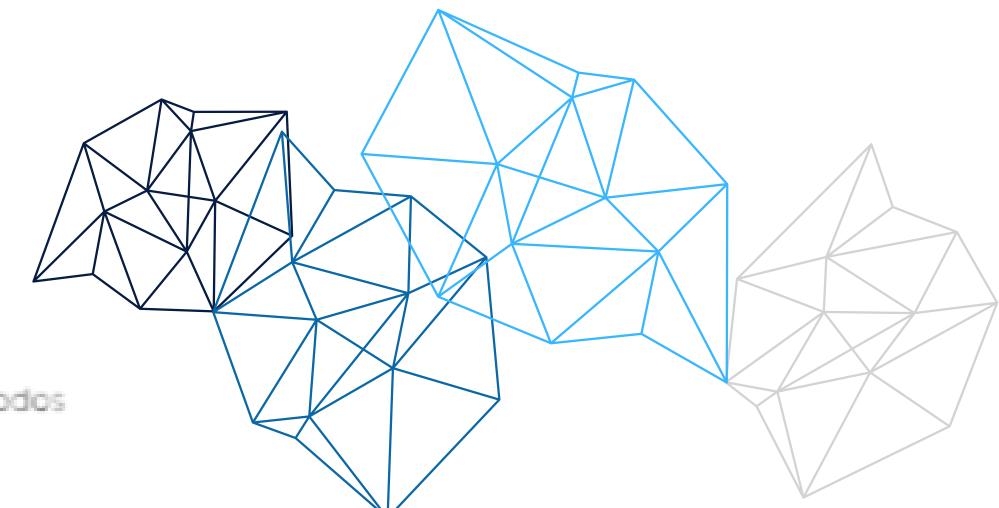
Subsecretaría de
Empleo



Ministerio de
Producción, Industria y Empleo



CHACO
Gobierno de todos



Uso con if

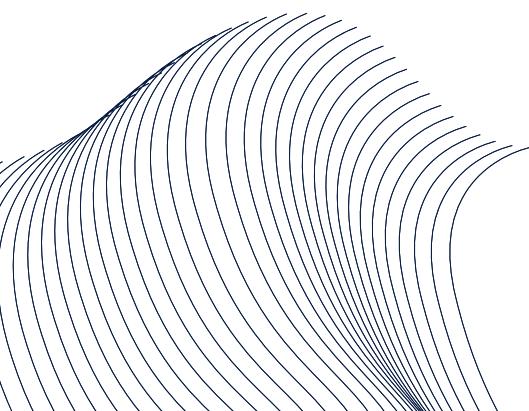
Los condicionales **if** evalúan una condición que es un valor booleano.

```
>>a = 1  
>>b = 2  
>>if b > a: (<- Si "b" es mayor que "a" ejecuta el siguiente código(imprimir "b es mayor que a"))  
    print("b es mayor que a")
```

La expresión que va después del if es siempre evaluada hasta que se da con un booleano.

```
>>if True:  
    print("Es True")
```

INFORMATARIO
Hacia una mejor industria informática



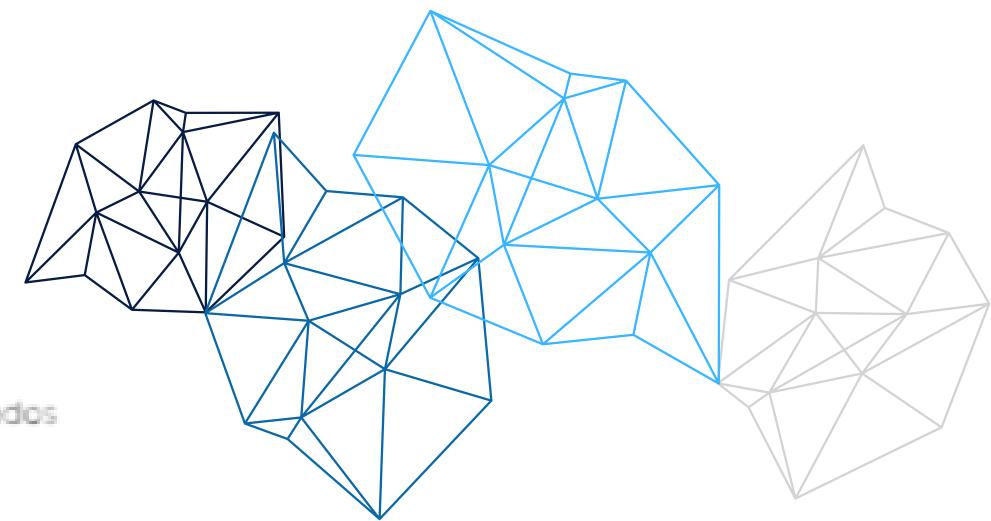
Subsecretaría de
Empleo



Ministerio de
Producción, Industria y Empleo



CHACO
Gobierno de todos



Cadena de caracteres o Strings

Para crear una variable de cadena de caracteres, simplemente asignamos una cadena de caracteres a una variable utilizando comillas simples (' ') o comillas dobles (" "):

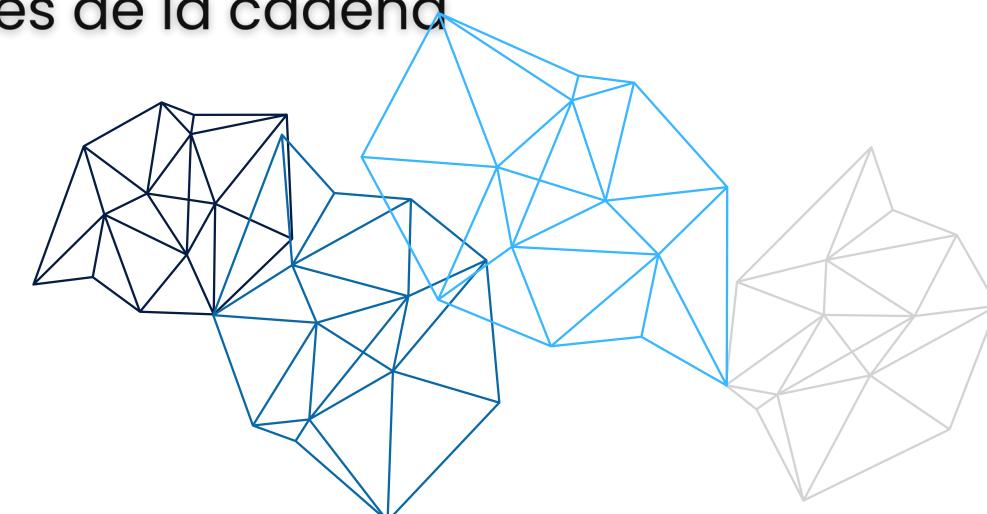
```
>>nombre = "Juan" (<- Comillas dobles ")
>>mensaje = 'Hola mundo' (<- Comillas simples ')
```

También es posible utilizar triples comillas simples o dobles para definir una cadena que abarque varias líneas:

```
>>cadena3 = "Este es un ejemplo de cadena que abarca varias líneas." (<- 3 comillas simples)
```

Las cadenas no están limitadas en tamaño, por lo que el único límite es la memoria de tu ordenador. Una cadena puede estar también vacía.

En Python, las cadenas son inmutables, lo que significa que una vez que se crea una cadena, no se puede modificar. Sin embargo, es posible crear una nueva cadena que contenga los caracteres de la cadena original y cualquier modificación que se quiera realizar.



Cadena de caracteres o Strings

Una situación que muchas veces se puede dar, es cuando queremos introducir una comilla, bien sea simple ' o doble " dentro de una cadena.

Para resolver este problema debemos recurrir a las secuencias de escape. La más importante es \", que nos permite incrustar comillas dentro de una cadena.

```
>>s = "Esto es una comilla doble \" de ejemplo"  
>>print(s)      #Esto es una comilla doble " de ejemplo
```

También podemos incluir un salto de línea dentro de una cadena, lo que significa que lo que esté después del salto, se imprimirá en una nueva línea.

```
>>s = "Primer linea\nSegunda linea"  
>>print(s)  
->#Primer linea  
->#Segunda linea
```



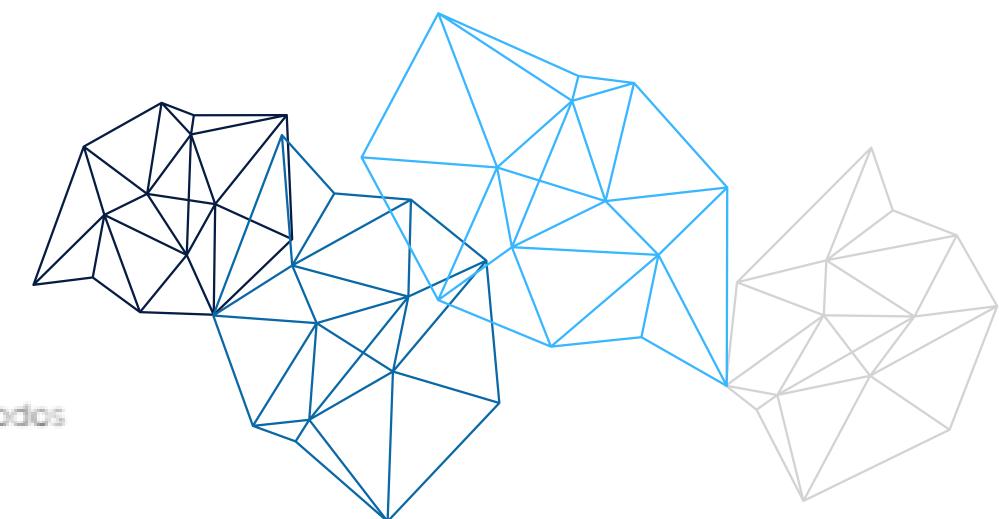
Subsecretaría de
Empleo



Ministerio de
Producción, Industria y Empleo



CHACO
Gobierno de todos



Formateo de cadenas

Tal vez queramos declarar una cadena que contenga variables en su interior, como números o incluso otras cadenas. Una forma de hacerlo sería concatenando la cadena que queremos con otra usando el operador `+`. Nótese que `str()` convierte en string lo que se pasa como parámetro.

```
>>x = 5  
>>s = "El número es: " + str(x)  
>>print(s)      #El número es: 5
```

Una mejor forma: `format()`

```
>>s = "Los números son {} y {}".format(5, 10)  
>>print(s)      #Los números son 5 y 10
```

Es posible también darle nombre a cada elemento, y `format()` se encargará de reemplazar todo.

```
>>s = "Los números son {a} y {b}".format(a=5, b=10)  
>>print(s)      #Los números son 5 y 10
```

Por si no fueran pocas ya, existe una tercera forma de hacerlo introducida en la versión 3.6 de Python. Reciben el nombre de cadenas literales o f-strings. Esta nueva característica, permite incrustar expresiones dentro de cadenas.

```
>>a = 5; b = 10  
>>s = f"Los números son {a} y {b}"  
>>print(s)      #Los números son 5 y 10
```



Subsecretaría de
Empleo



Ministerio de
Producción, Industria y Empleo

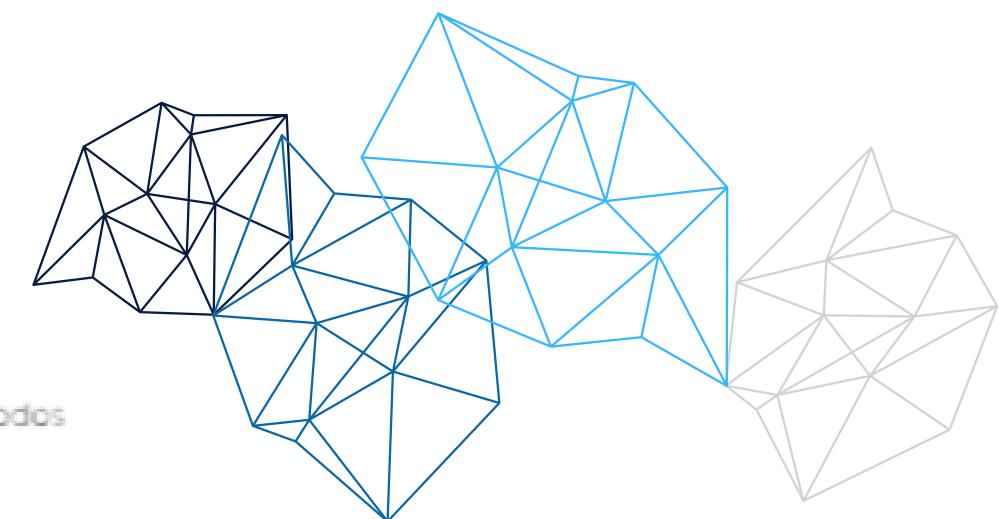


CHACO
Gobierno de todos



INFORMATARIO

Hacia una mejor industria informática



Ejemplos string

Para entender mejor la clase string, vamos a ver unos ejemplos de como se comportan. Podemos sumar dos strings con el operador +.

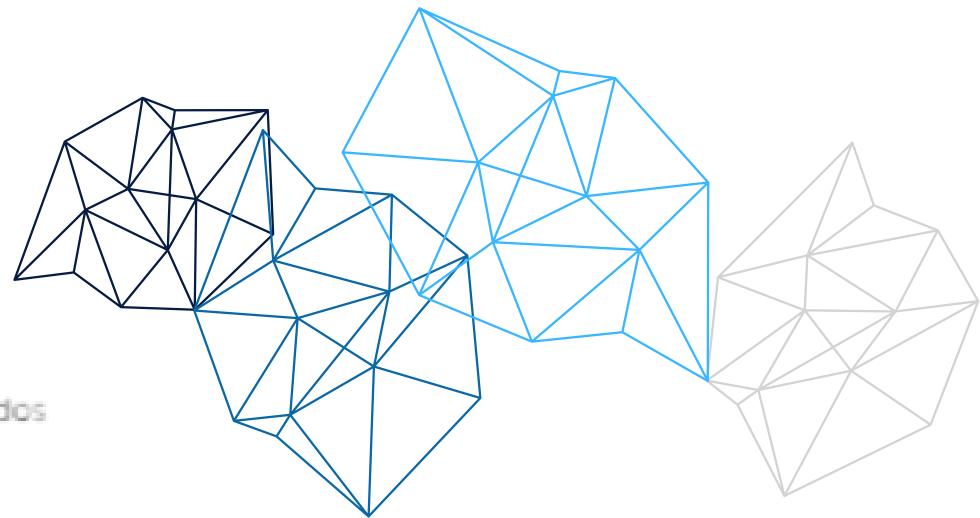
```
>>s1 = "Parte 1"  
>>s2 = "Parte 2"  
>>print(s1 + " " + s2) #Parte 1 Parte 2
```

Se puede multiplicar un string por un int. Su resultado es replicarlo tantas veces como el valor del entero.

```
>>s = "Hola "  
>>print(s*3)      #Hola Hola Hola
```

La longitud de una cadena viene determinada por su número de caracteres, y se puede consultar con la función len().

```
>>print(len("Esta es mi cadena")) #17
```



Datos mutables e inmutables

Los datos inmutables son aquellos que no se pueden modificar una vez que se han creado. Esto significa que cualquier intento de cambiar el valor de un dato inmutable resultará en la creación de un nuevo objeto. Los siguientes son algunos ejemplos de datos inmutables en Python:

1. Números enteros (int) y números de punto flotante (float).
2. Valores booleanos (bool): True y False.
3. Tuplas (tuple): una secuencia ordenada e inmutable de elementos.
4. Cadenas (str): una secuencia de caracteres inmutable.

etc.

Por otro lado, los datos mutables son aquellos que pueden ser modificados después de haber sido creados. Ejemplos de datos mutables en Python incluyen listas (list), diccionarios (dict), conjuntos (set) y objetos personalizados.



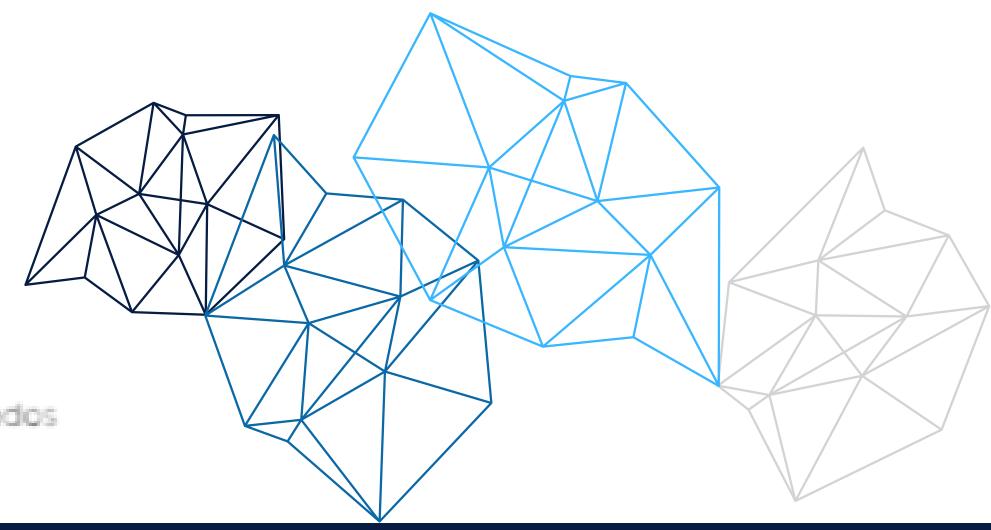
Subsecretaría de
Empleo



Ministerio de
Producción, Industria y Empleo



CHACO
Gobierno de todos



Listas

Las listas en Python son un tipo de dato que permite almacenar datos de cualquier tipo. Son mutables y dinámicas, lo cual es la principal diferencia con los sets y las tuplas.

Crear listas:

Las listas en Python son uno de los tipos o estructuras de datos más versátiles del lenguaje, ya que permiten almacenar un conjunto arbitrario de datos. Es decir, podemos guardar en ellas prácticamente lo que sea. Una lista se crea con [] separando sus elementos con comas ,. Una gran ventaja es que pueden almacenar tipos de datos distintos.

```
>>lista = [1, "Hola", 3.67, [1, 2, 3]]
```

Algunas propiedades de las listas:

- Son ordenadas, mantienen el orden en el que han sido definidas
- Pueden ser formadas por tipos arbitrarios
- Pueden ser indexadas con [i].
- Se pueden anidar, es decir, meter una dentro de la otra.
- Son mutables, ya que sus elementos pueden ser modificados.
- Son dinámicas, ya que se pueden añadir o eliminar elementos.



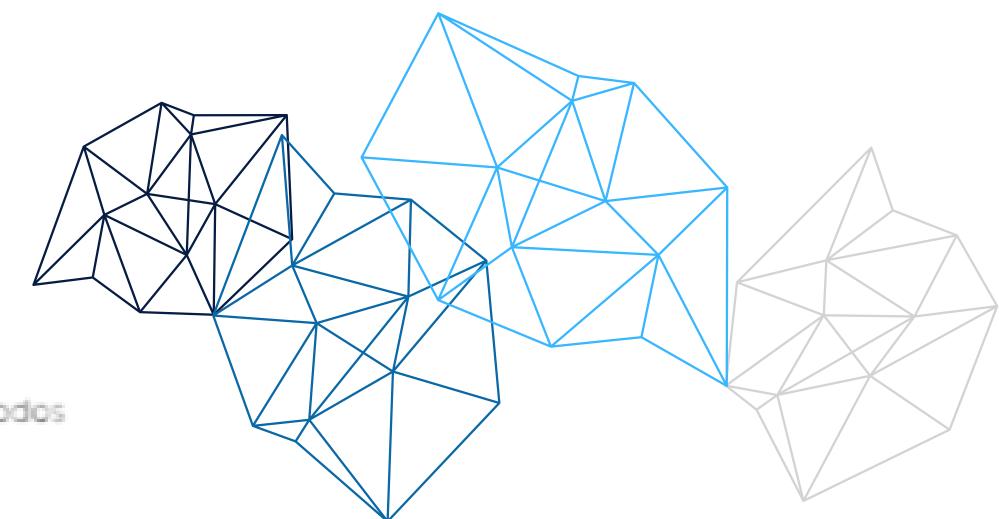
Subsecretaría de
Empleo



Ministerio de
Producción, Industria y Empleo



CHACO
Gobierno de todos



Acceder, modificar y eliminar elementos de las listas

Si tenemos una lista `a` con 3 elementos almacenados en ella, podemos acceder a los mismos usando corchetes y un índice, que va desde 0 a $n-1$ siendo n el tamaño de la lista.

```
>>a = [90, "Python", 3.87]
```

```
>>print(a[0]) #90
```

Se puede también acceder al último elemento usando el índice `[-1]`.

```
>>a = [90, "Python", 3.87]
```

```
>>print(a[-1]) #3.87
```

De la misma manera, al igual que `[-1]` es el último elemento, podemos acceder a `[-2]` que será el penúltimo.

```
>>print(a[-2]) #Python
```

Y si queremos modificar un elemento de la lista, basta con asignar con el operador `=` el nuevo valor.

```
>>a[2] = 1
```

```
>>print(a) #[90, 'Python', 1]
```

Un elemento puede ser eliminado con `del` y la lista con el índice a eliminar.

```
>>l = [1, 2, 3, 4, 5]
```

```
>>del l[1]
```

```
>>print(l) #[1, 3, 4, 5]
```



Subsecretaría de
Empleo



Ministerio de
Producción, Industria y Empleo



CHACO
Gobierno de todos

