

GUÍA PARA ESTUDIANTES UNIDAD 2

INTRODUCCIÓN A LOS
LENGUAJES DE
PROGRAMACIÓN (JAVA y
PYTHON)

INSTRUCTORES: BYRON GUAMÁN, KEVIN ULCUANGO
CAPACITACIONES CSOFT

PRESENTACIÓN DEL CURSO

TEMA:

- INTRODUCCIÓN A LOS LENGUAJES DE PROGRAMACIÓN (JAVA Y PYTHON)

INSTRUCTORES:

- Byron Guamán
- Kevin Ulcuango

OBJETIVO DEL CURSO:

Fortalecer el conocimiento sobre la lógica de programación en el lenguaje Java, mediante talleres, proyectos y ejercicios prácticos, los cuales les servirá para el correcto desempeño en las posteriores asignaturas de su vida académica, así como también adquirir conocimientos y destrezas necesarias en el lenguaje de programación Python.

Contenido

PRESENTACIÓN DEL CURSO	1
TEMA:	1
INSTRUCTORES:.....	1
OBJETIVO DEL CURSO:	1
¿Qué es Python?	4
Instalación.....	4
Entorno de desarrollo Visual Studio Code.....	5
Google collab	5
Indentación y codificación	6
Tipos de datos.....	7
Tipos numéricos.....	7
Conversiones	7
Tipo booleano	8
Cadenas de texto	9
Caracteres especiales	9
Variables constantes y operadores	10
Variables	10
Nombres de variables.....	11
Constantes	11
Operadores lógicos.....	12
Operadores relacionales.....	13
Operadores aritméticos.....	13
Operadores de asignación	14
Operadores Adicionales:	14
Prioridad de cálculo de los operadores	15
Entrada y salida	16
ESTRUCTURAS DE CONTROL.....	18
Estructura if	18
Estructura if-else.....	18
Estructura elif	19
BUCLES O CICLOS	19
Estructura for.....	19

Estructura while.....	21
Estructuras de datos.....	21
Listas	21
Tablas.....	23
Diccionarios	24
Bibliografía.....	28

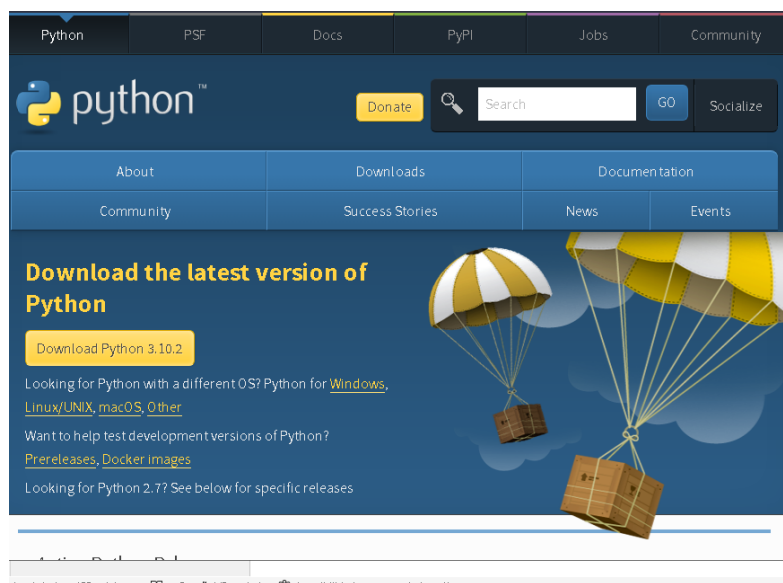
¿Qué es Python?

Python se define como un lenguaje de programación interpretado, que quiere decir que, el programa se ejecuta en tiempo real línea a línea sin pasar por un proceso de compilación. Actualmente este lenguaje de programación se ha visto usado para una amplia gama de aplicaciones gracias a ciertas características como:

- Multiplataforma: ejecutable en varios sistemas operativos
- Propósito general: no está orientado ciertos tipos de aplicaciones en particular
- Tipado dinámico: el lenguaje se adapta lo escrito en código
- Orientado a objetos: incluye funciones de reutilización de componentes mediante la herencia y el polimorfismo
- Comunidad: su comunidad lo mantiene actualizado ya sea implementando nuevas librerías, funciones, facilidades en los entornos y el soporte en general

Instalación

El primer paso para empezar a programar con Python es instalar su interprete, generalmente se debe revisar en la página web oficial, cual es la versión más reciente y estable para evitar cualquier tipo de inconvenientes.



Adicionalmente es recomendable usar un IDE(Integrated Development Enviroment) que integren herramientas para facilitar el desarrollo con Python, los más populares son:



Entorno de desarrollo Visual Studio Code

Visual Studio Code nos permite programar en lenguaje python mediante la instalación de una extensión que, Aprovecha todo el poder de VS Code para proporcionar autocompletado e IntelliSense, linting, depuración, junto con la capacidad de cambiar fácilmente entre entornos de Python, incluidos entornos virtuales y conda. Esta extensión esta disponible desde el “Marketplace”.

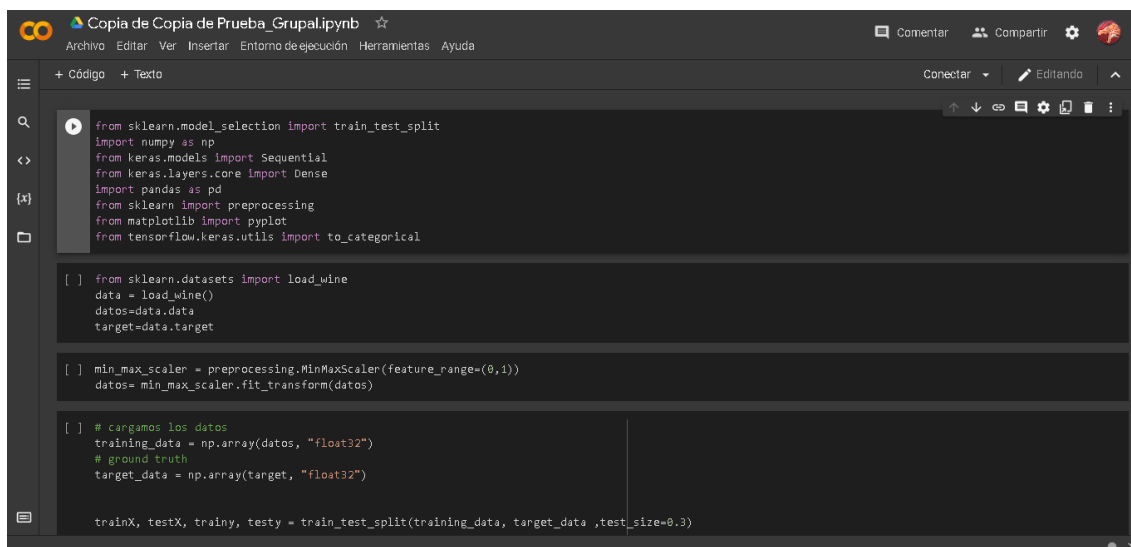


Ahora para ejecutar un script bastaría con ejecutarlo con el botón “run” del IDE, otra opción es usando el comando “py (nombre del archivo).py” en la consola.

Google collab

Colab, o "Colaboratory", te permite escribir y ejecutar código de Python en tu navegador, con

- Sin configuración requerida
- Acceso gratuito a GPU
- Facilidad para compartir



```

from sklearn.model_selection import train_test_split
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense
import pandas as pd
from sklearn import preprocessing
from matplotlib import pyplot
from tensorflow.keras.utils import to_categorical

[ ] from sklearn.datasets import load_wine
data = load_wine()
datos=data.data
target=data.target

[ ] min_max_scaler = preprocessing.MinMaxScaler(feature_range=(0,1))
datos= min_max_scaler.fit_transform(datos)

[ ] # cargamos los datos
training_data = np.array(datos, "float32")
# ground truth
target_data = np.array(target, "float32")

trainX, testX, trainy, testy = train_test_split(training_data, target_data ,test_size=0.3)

```

El manejo de los archivos y el lenguaje en Google Collab se realiza mediante el formato “.ipynb”, conocido como Jupyter Notebook. Es un entorno computacional interactivo, en el que puede combinar ejecución de código, texto enriquecido, matemáticas, gráficos y medios enriquecidos.

Indentación y codificación

En Python no existen los comandos para finalizar líneas o llaves que delimiten el código, en su lugar se usa la indentación que permite delimitar los bloques de código al estructurar el programa con comandos como if, else, for, entre otros.

Esta indentación se suele definir como 4 espacios en blanco para establecer el inicio de un nuevo bloque y para finalizar solo debemos dejar de aplicar la sangría en una nueva línea.



Tipos de datos

Los tipos de datos define a un grupo de variables con características y propiedades específicas, los valores que pueden tomar y las operaciones que se pueden realizar, por lo que en Python al no definir los tipos de datos explícitamente (lenguaje de tipado dinámico), estos se asocian a un tipo de dato de acuerdo con el valor asignado.

Tipos numéricos

Los números de Python están fuertemente relacionados con los números matemáticos, pero están sujetos a las limitaciones de la representación numérica en las computadoras.

Tipo	Definición	Ejemplos
Entero (int)	Número entero con precisión fija	# diez entero diez = 10 diez_b = 0b1010 diez_o = 0o12 diez_h = 0xa
Punto flotante (float)	Representan números reales y se escriben con un punto decimal dividiendo la parte entera y la fraccionaria	real = 1.1. real = 1/2 not_cient = 1.23E3
Complejo (complex)	Los números complejos tienen una parte real y otra imaginaria y cada una de ellas se representa como un float	complejo = 1+2j

Conversiones

Si es requerido, Python integra función de conversión de tipos numéricos para satisfacer los requerimientos de un operador o los parámetros de ciertas funciones

- `int(x)` para convertir x en un entero simple.

- `long(x)` para convertir `x` en un entero largo.
- `float(x)` para convertir `x` en un número de coma flotante.
- `complex(x)` para convertir `x` en un número complejo con parte real `x` y parte imaginaria cero.
- `complex(x, y)` para convertir `x` e `y` en un número complejo con parte real `x` y parte imaginaria `y`. `x` e `y` son expresiones numéricas

Tipo booleano

Un tipo de dato que puede almacenar dos valores, `true` para el valor verdadero y `false` para el valor falso

Tipo	Definición	Ejemplos
Booleano (bool)	Variable asignada explícitamente con el valor <i>true</i> o <i>false</i> .	<code>x = True</code> <code>y = false</code>

El método `bool()`

A cualquier variable se representará con un valor booleano (haciendo uso de la función `bool()`), generalmente las variables numéricas con un valor de 0, los elementos vacíos y nulos serán representados como *false*, y el resto serán considerados *true*.

`bool(0) → False`

`bool(0.0) → False`

`bool("") → False`

`bool(None) → False`

`bool(25) → True`

`bool(-9.5) → True`

`bool("abc") → True`

Cadenas de texto

En Python, las cadenas son matrices de bytes que representan caracteres Unicode. Python no tiene un tipo de datos de caracteres, un solo carácter es simplemente una cadena con una longitud de 1. Los corchetes se pueden usar para acceder a elementos de la cadena.

Tipo	Definición	Ejemplos
Cadena (str)	Secuencia inmutable de caracteres en formato Unicode	Text = "texto" Cadena = 'cad'

Caracteres especiales

Comillas triples

Se usan cuando se va a representar una cadena de texto de varias líneas.

```
>>> print("""
... asdhgv
... asd
... asd
... asd""")
... )
asdhgv
asd
asd
asd
>>>
```

Comillas dentro de comillas

Se usan para representar el valor de carácter de una comilla dentro de una cadena

```
>>> print("'texto'")
'texto'
>>> print('"texto"')
"texto"
```

Contrabarra (\)

\ ' y \"

Se usa para representar una comilla simple o doble dentro de una cadena

```
>>> print("esto es una comilla simple \'")
esto es una comilla simple '
>>> print("esto es una comilla doble \" ")
esto es una comilla doble "
```

\n

Representa un salto de línea dentro de una cadena

```
>>> print("Salto de texto aquí ---->\n esta es una nueva línea")
Salto de texto aquí ---->
esta es una nueva línea
>>>
```

\t

Representa una tabulación dentro de una cadena

```
>>> print("Tabulacion aquí ---->\t texto a un espacio tabulado")
Tabulacion aquí ---->      texto a un espacio tabulado
```

Variables constantes y operadores

Variables

Una variable como un nombre adjunto a un objeto en particular. En Python, las variables no necesitan ser declaradas o definidas por adelantado, como es el caso en muchos otros lenguajes de programación. Para crear una variable, simplemente asígnele un valor y luego comience a usarla.

En Python, a una variable se le puede asignar un valor de un tipo y luego reasignarle un valor de un tipo diferente, a diferencia de muchos otros lenguajes donde las variables son estáticas, de un mismo tipo durante su tiempo de vida.

Python es un lenguaje altamente orientado a objetos. De hecho, prácticamente todos los elementos de datos en un programa de Python son objetos de un tipo o clase específicos.

Considere el siguiente ejemplo:

```
>>> print("cadena")
cadena
```

Para representar la sentencia ingresada `print("cadena")`, el interprete llevo a cabo los siguientes pasos por debajo:

- Crear un objeto de tipo cadena (str)
- Darle el valor de "cadena"
- Imprimirlo en la consola

Nota: Las cadenas admiten el operador "+" para concatenar y el operador "*" para multiplicar.

Nombres de variables

Los nombres de variables en Python pueden tener cualquier longitud y pueden consistir en letras mayúsculas y minúsculas (A-Z, a-z), dígitos (0-9) y el carácter de subrayado (_). Una restricción adicional es que, aunque un nombre de variable puede contener dígitos, el primer carácter de un nombre de variable no puede ser un dígito.

También se debe tomar en cuenta las palabras reservadas del lenguaje que nos permitirán usarlas como nombres de una variable:

and	exec	lambda
assert	finally	not
break	for	or
class	from	pass
continue	global	print
def	if	raise
del	import	return
elif	in	try
else	is	while
except		

Para nombrar una variable tampoco se acepta poner espacios por lo que existen varios métodos más utilizados para construir un el nombre de variable de varias palabras, los mas usados son;

Camel Case: la segunda palabra y las subsiguientes están en mayúscula para que los límites de las palabras sean más fáciles de ver. Ejemplo: *numeroDeEstudiantesInscritos*

Pascal case: Idéntico al Camel case excepto que la primera palabra también está en mayúscula. Ejemplo: *NumeroDeEstudiantesInscritos*

Snake case: las palabras están separadas por guiones bajos. Ejemplo: *numero_de_estudiantes_inscritos*

Constantes

Una constante es un tipo de variable la cual no puede ser cambiada. Eso es muy de ayuda pensar las constantes como contenedores que contienen información el cual no puede ser cambiado después.

En Python, las constantes son usualmente declaradas y asignadas en un módulo. Aquí, el módulo significa un nuevo archivo que contiene variables, funciones, etc; el cual es importada en el archivo principal. Dentro del módulo, las constantes son escritas en letras MAYÚSCULAS y separadas las palabras con el carácter underscore “_”.

Operadores

Son símbolos reservados por el propio lenguaje que se utilizan para llevar a cabo operaciones sobre uno, dos o más elementos llamados operandos. Los operandos pueden ser variables, literales, el valor devuelto por una expresión o el valor devuelto por una función.

Operadores lógicos

Los operadores lógicos son unas operaciones que trabajan con valores booleanos.

Operación	Resultado	Descripción
a or b	Si a se evalúa a falso, entonces devuelve b, si no devuelve a	Solo se evalúa el segundo operando si el primero es falso
a and b	Si a se evalúa a falso, entonces devuelve a, si no devuelve b	Solo se evalúa el segundo operando si el primero es verdadero
not a	Si a se evalúa a falso, entonces devuelve True, si no devuelve False	Tiene menos prioridad que otros operadores no booleanos

Operadores relacionales

Los operadores de comparación se utilizan, para comparar dos o más valores, dando como resultado siempre True o False.

Operador	Descripción
>	Mayor que. True si el operando de la izquierda es estrictamente mayor que el de la derecha; False en caso contrario.
>=	Mayor o igual que. True si el operando de la izquierda es mayor o igual que el de la derecha; False en caso contrario.
<	Menor que. True si el operando de la izquierda es estrictamente menor que el de la derecha; False en caso contrario.
<=	Menor o igual que. True si el operando de la izquierda es menor o igual que el de la derecha; False en caso contrario.
==	Igual. True si el operando de la izquierda es igual que el de la derecha; False en caso contrario.
!=	Distinto. True si los operandos son distintos; False en caso contrario.

Operadores aritméticos

Permiten realizar las diferentes operaciones aritméticas del álgebra: suma, resta, producto, división, entre otros.

Operador	Descripción
+	Suma dos operandos.
-	Resta al operando de la izquierda el valor del operando de la derecha. Utilizado sobre un único operando, le cambia el signo.
*	Producto/Multiplicación de dos operandos.
/	Divide el operando de la izquierda por el de la derecha (el resultado siempre es un float).
%	Operador módulo. Obtiene el resto de dividir el operando de la izquierda por el de la derecha.

//	Obtiene el cociente entero de dividir el operando de la izquierda por el de la derecha.
**	Potencia. El resultado es el operando de la izquierda elevado a la potencia del operando de la derecha.

Operadores de asignación

Se utiliza para asignar un valor a una variable mediante el signo =.

Operador	Ejemplo	Equivalencia
+=	x += 2	x = x + 2
-=	x -= 2	x = x - 2
*=	x *= 2	x = x * 2
/=	x /= 2	x = x / 2
%=	x %= 2	x = x % 2
//=	x //= 2	x = x // 2
**=	x **= 2	x = x ** 2
&=	x &= 2	x = x & 2
=	x = 2	x = x 2
^=	x ^= 2	x = x ^ 2
>>=	x >>= 2	x = x >> 2
<<=	x <<= 2	x = x << 2

Operadores Adicionales:

Operadores de pertenencia

Los operadores de pertenencia se utilizan para comprobar si un valor o variable se encuentran en una secuencia (list, tuple, dict, set o str).

Operador	Descripción
in	Devuelve True si el valor se encuentra en una secuencia; False en caso contrario.
not in	Devuelve True si el valor no se encuentra en una secuencia; False en caso contrario.

Operadores de identidad

Se utilizan para comprobar si dos variables son, o no, el mismo objeto

Operador	Descripción
is	Devuelve True si ambos operandos hacen referencia al mismo objeto; False en caso contrario.
is not	Devuelve True si ambos operandos no hacen referencia al mismo objeto; False en caso contrario.

Operadores a nivel de bits

Los operadores a nivel de bits actúan sobre los operandos como si fueran una cadena de dígitos binarios.

Operación	Descripción
$x \mid y$	or bit a bit de x e y.
$x \wedge y$	or exclusivo bit a bit de x e y.
$x \& y$	and bit a bit de x e y.
$x \ll n$	Desplaza x n bits a la izquierda.
$x \gg n$	Desplaza x n bits a la derecha.
$\sim x$	not x. Obtiene los bits de x invertidos.

Prioridad de cálculo de los operadores

Python tienen un orden de prioridad. Este orden es el siguiente, de más prioritario al menos prioritario:

- Aritméticos (mismo orden de prioridad que en las matemáticas)
- A nivel de bits
- Operadores de comparación, identidad y pertenencia
- Operadores booleanos
- Asignación

Entrada y salida

La función print()

Esta función nos presenta en consola un valor ya sea texto o el mismo valor de una variable.

```
>>> print("hola mundo")
hola mundo
>>> a = 56
>>> print(a)
56
>>> b = ["texto",45,6.7,False]
>>> print(b)
['texto', 45, 6.7, False]
```

De la misma manera esta función nos permite concatenar valores separados con comas para concatenar diferentes tipos de variables

```
>>> a = 56
>>> b = 4.5
>>> print("El valor de a es: ",a," y el valor de b es: ",b)
El valor de a es: 56 y el valor de b es: 4.5
```

O usando el operador "+" para concatenar texto, por lo cual se debe transformar a string otros tipos de dato:

```
>>> a = 56
>>> b = 4.5
>>> print("El valor de a es: "+str(a)+" y el valor de b es: "+str(b))
El valor de a es: 56 y el valor de b es: 4.5
```

También hay que tomar en cuenta que la función *print()* por defecto efectúa un salto de línea al final de su ejecución, y, en caso que se requiera imprimir en la misma línea se debe definir el argumento *end=""*

```
print("Esta funcion dara un salto de linea aqui ---->")
print("Y esto se mostrara en una nueva linea")
```

```
Esta funcion dara un salto de linea aqui ---->
Y esto se mostrara en una nueva linea
```

```
print('Si definimos el argumento end="" ',end="")  
print("Esta linea se presentara a continuacion")
```

```
Si definimos el argumento end="" Esta linea se presentara a continuacion
```

La función input()

Esta función es capaz de obtener datos ingresados por el usuario a través de la consola de Python

```
print("Ingrese su nombre:",end="")  
nombre = input()  
print("Bienvenido: ",nombre)
```

```
Ingrese su nombre:Usuario1  
Bienvenido: Usuario1
```

Hay que tomar en cuenta que la entrada obtenida mediante este método siempre será de tipo String y en caso de requerir otro tipo de dato se debe transformar para evitar errores

```
nombre = input("Ingrese su año de nombre:")  
print("Bienvenido: "+nombre)
```

ESTRUCTURAS DE CONTROL

Son construcciones permiten condicionar la ejecución de uno o varios bloques de sentencias al cumplimiento de una o varias condiciones.

Estructura if

La estructura de control *if ...* permite que un programa ejecute unas instrucciones cuando se cumplan una condición.

if *condicion*:

 bloque de instrucciones

Ejemplo:

```
if a>b:
    print("a es mayor que b")
```

Estructura if-else

La estructura de control *if ... else ...* permite que un programa ejecute unas instrucciones cuando se cumple una condición y otras instrucciones cuando no se cumple esa condición.

if *condicion*:

 bloque de instrucciones 1

else:

 bloque de instrucciones 2

Ejemplo:

```
if a>b:
    print("a es mayor que b")
else:
    print("a no es mayor que b")
```

Estructura elif

Elif es una contracción de *else if* lo que permite evaluar más de una condición a la vez.

```
if condicion1:
    bloque de instrucciones 1
elif condicion2:
    bloque de instrucciones 2
else:
    bloque de instrucciones 3
```

Ejemplo:

```
if a>b:
    print("a es mayor que b")
elif a==b:
    print("a es igual que b")
else:
    print("a no es mayor que b")
```

BUCLES O CICLOS

Son estructuras de control que repiten la ejecución de un bloque de instrucciones

Estructura for

Un bucle *for* es una sentencia que repite el bloque de instrucciones un número predeterminado de veces.

Usualmente se aplica para recorrer una estructura de datos(elemento iterable), como lo son las listas, cadenas, diccionarios, en el caso de que se lo quiera usar con valores numéricos se puede hacer uso de la función `range()`.

```
for variable in elemento iterable (lista, cadena, range, etc.):
    cuerpo del bucle
```

Ejemplo

```
lista = [1,"texto",3.4,True]
print(lista)
for i in lista:
    print("El elemento "+str(i)+" es: " + str(type(i)))
```

```
[1, 'texto', 3.4, True]
El elemento 1 es: <class 'int'>
El elemento texto es: <class 'str'>
El elemento 3.4 es: <class 'float'>
El elemento True es: <class 'bool'>
```

La función range()

Esta función nos devuelve una secuencia de números (por defecto empieza en 0 y tiene un paso de 1), en la que podemos definir su inicio, su final y un paso.

range(inicio, final, paso)

Inicio	Opcional, numero entero que indica la posición de inicio (por defecto es 0)
Final	Obligatorio, numero entero que indica la posición de finalización (el valor declarado no se incluye)
Paso	Opcional, numero entero que define el incremento de la secuencia (por defecto es 1)

Ejemplos

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

```
for i in range(10,20,2):
    print(i)
```

```
10
12
14
16
18
```

Estructura while

Permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición, es decir mientras la condición devuelva el valor *True*.

```
while condicion:  
    cuerpo del bucle
```

Ejemplo:

```
while (i <= 5):  
    print(i)  
    i+=1
```

Estructuras de datos

Listas

Una lista es capaz de almacenar múltiples valores en una misma variable, son conjuntos ordenados de elementos (números, cadenas, listas, etc). Las listas se delimitan por corchetes ([]) y los elementos se separan por comas. Las listas pueden contener elementos del mismo tipo, o pueden contener elementos de tipos distintos.

Las listas se pueden declarar explícitamente con sus elementos, una lista vacía solo con los corchetes o explícitamente usando la función *list()*.

```
días = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes"]  
  
días2 = ["Lunes", 2, "Miércoles", 4, "Viernes"]
```

Se puede tener acceso a sus elementos individualmente mediante su posición y modificar sus elementos mediante la asignación

```
lista1 = ["a","b","c","d"]  
print(lista1[0])  
print(lista1[3])  
lista1[3] = 4  
print(lista1)
```

```
a  
d  
['a', 'b', 'c', 4]
```

Otro uso de las posiciones de una lista es el manejo de sublistas estas marcan las posiciones de inicio y fin de los elementos de una lista, si se deja vacía la expresión se tomaran todos los elementos.

```
lista1 = ["a","b","c","d"]
print(lista1)
lista2 = lista1[1:4]
print(lista2)
lista2 = lista1[: ]
print(lista2)
```

```
['a', 'b', 'c', 'd']
['b', 'c', 'd']
['a', 'b', 'c', 'd']
```

Las sublistas no sirven solo para tener acceso a los elementos de una lista, también es posible modificarlos

```
lista1 = ["a","b","c","d"]
print(lista1)
lista1[1:3]=["x"]
print(lista1)
```

```
['a', 'b', 'c', 'd']
['a', 'x', 'd']
```

Es posible concatenar listas mediante el uso del operador “+”

```
lista1 = ["a","b","c","d"]
lista2 = ["e","f","g","h"]
lista3 = lista1 + lista2
print(lista1)
print(lista2)
print(lista3)
```

```
['a', 'b', 'c', 'd']
['e', 'f', 'g', 'h']
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

Para eliminar elementos de una lista es posible usar la palabra reservada “del” ya sea con posiciones específicas o con sublistas.

```
lista1 = ["a","b","c","d"]
print(lista1)
del(lista1[2])
print(lista1)
```

```
['a', 'b', 'c', 'd']
['a', 'b', 'd']
```

Recuerde el uso general de los objetos y su asignación, de manera que si asignamos directamente el valor de un objeto tipo lista a otro la mutación del primer objeto afectara a ambos

```
lista1 = ["a","b","c","d"]
lista2 = lista1

lista1[0] = "z"
print(lista1)
print(lista2)
```

```
['z', 'b', 'c', 'd']
['z', 'b', 'c', 'd']
```

Y si se quiere evitar este efecto se debe hacer uso de la notación de sublistas

```
lista1 = ["a","b","c","d"]
lista2 = lista1[:]

lista1[0] = "z"
print(lista1)
print(lista2)
```

```
['z', 'b', 'c', 'd']
['a', 'b', 'c', 'd']
```

Tablas

Como se revisó en el punto anterior las listas pueden almacenar cualquier tipo de dato, tomando esto en cuenta estas pueden contener otras listas dando como resultado las tablas (listas de listas)


```
dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes"]

dias2 = ["Lunes", 2, "Miércoles", 4, "Viernes"]

tabla = [dias , dias2]
```

Considerando esta particularidad las tablas serán capaces de realizar todas las funciones inherentes a las listas

Recorrido de tablas

```
tabla = [[1,2,3],[4,5,6],[7,8,9]]
print(tabla)
for i in range(len(tabla)):
    for j in range(len(tabla[i])):
        print("Elemento en la pos:"+str(i)+"["+str(j)+"]: "+str(tabla[i][j])+" ",end="")
    print()
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
Elemento en la pos:[0][0]: 1 Elemento en la pos:[0][1]: 2 Elemento en la pos:[0][2]: 3
Elemento en la pos:[1][0]: 4 Elemento en la pos:[1][1]: 5 Elemento en la pos:[1][2]: 6
Elemento en la pos:[2][0]: 7 Elemento en la pos:[2][1]: 8 Elemento en la pos:[2][2]: 9
```

```
for i in tabla:
    for j in i:
        print(j," ",end="")
    print()
```

```
1 2 3
4 5 6
7 8 9
```

Diccionarios

Permite almacenar cualquier tipo de valor como enteros, cadenas, listas e incluso otras funciones. La característica que lo diferencia de otras estructuras de datos es que estos diccionarios nos permiten identificar cada elemento por una clave (Key).

Ejemplos:

```
diccionario = {'nombre' : 'Kevin', 'edad' : 22, 'curso': ['Python','Java'] }
```

```
diccionario = {4.5:["lun","mar","mie","jue","vie"],99:[1,"martes",3,"jueves",4]}
```

Recorrido de un diccionario

Un diccionario se define como un objeto iterable por lo que podemos usar un ciclo for para recorrerlo (el elemento iterable serán las claves "keys")

```
for i in diccionario:  
    print("Key:",i," Value:",diccionario[i])
```

Funciones

Entre las principales funciones que incluye la clase diccionario para su manejo encontramos:

dict()

Recibe una representación de un diccionario mediante variables y asignaciones y si es correctamente crea un diccionario con esos valores

```
diccionario = dict(nombre="kevin",edad=22, cursos=["Java","Python"])
```

```
{'nombre': 'kevin', 'edad': 22, 'cursos': ['Java', 'Python']}
```

zip()

Recibe como parámetros dos elementos iterables de la misma longitud y los convierte en objetos iterables, el primero servirá de claves y el segundo como sus valores

```
diccionario = dict(zip("abcde",["texto",4,5.6,True,[1,2,3,4]]))
```

```
{'a': 'texto', 'b': 4, 'c': 5.6, 'd': True, 'e': [1, 2, 3, 4]}
```

keys()

Devuelve una lista con las claves del diccionario

```
k = diccionario.keys()  
print(k)
```

```
{'a': 'texto', 'b': 4, 'c': 5.6, 'd': True, 'e': [1, 2, 3, 4]}  
dict_keys(['a', 'b', 'c', 'd', 'e'])
```

values()

devuelve una lista con los valores de un diccionario

```
v = diccionario.values()
print(v)
```

```
{'a': 'texto', 'b': 4, 'c': 5.6, 'd': True, 'e': [1, 2, 3, 4]}
dict_values(['texto', 4, 5.6, True, [1, 2, 3, 4]])
```

clear()

Elimina los datos de un diccionario dejándolo vacío

```
diccionario.clear()
print(diccionario)
```

```
{'a': 'texto', 'b': 4, 'c': 5.6, 'd': True, 'e': [1, 2, 3, 4]}
{}
```

copy()

Crea una copia de un diccionario

```
diccionario2 = diccionario.copy()
print(diccionario)
```

```
{'a': 'texto', 'b': 4, 'c': 5.6, 'd': True, 'e': [1, 2, 3, 4]}
{'a': 'texto', 'b': 4, 'c': 5.6, 'd': True, 'e': [1, 2, 3, 4]}
```

get()

Recibe como parámetro una clave, devuelve el valor de la clave. Si no lo encuentra, devuelve un objeto none.

```
x = diccionario.get("c")
print(x)
```

```
{'a': 'texto', 'b': 4, 'c': 5.6, 'd': True, 'e': [1, 2, 3, 4]}
5.6
```

pop()

Recibe como parámetro una clave, elimina esta y devuelve su valor. Si no lo encuentra, devuelve error.

```
print(diccionario)
x = diccionario.pop("a")
print(diccionario)
print(x)
```

```
{'a': 'texto', 'b': 4, 'c': 5.6, 'd': True, 'e': [1, 2, 3, 4]}
{'b': 4, 'c': 5.6, 'd': True, 'e': [1, 2, 3, 4]}
texto
```

setdefault()

Puede recibir como parámetro un índice funcionando de la misma manera que el método get()

```
x = diccionario.setdefault("b")
print(x)
```

```
{'a': 'texto', 'b': 4, 'c': 5.6, 'd': True, 'e': [1, 2, 3, 4]}
4
```

también puede recibir dos parámetros para agregar un nuevo índice y valor a un diccionario

```
diccionario.setdefault("f", "valoragregado")
print(diccionario)
```

```
{'a': 'texto', 'b': 4, 'c': 5.6, 'd': True, 'e': [1, 2, 3, 4]}
{'a': 'texto', 'b': 4, 'c': 5.6, 'd': True, 'e': [1, 2, 3, 4], 'f': 'valoragregado'}
```

Bibliografía

- Bill, L. (2019). *Introducing Python* (Andy Oram and Allyson MacDonald, Ed.). O'Reilly Media, Inc.
https://d1wqtxts1xzle7.cloudfront.net/59519777/introducingpython20190604-80425-peeeks-with-cover-page-v2.pdf?Expires=1648011142&Signature=UBoja2h4lV86o8UoSXZnyLu5qcc1ACweNjU1UWs8GnRIBiQ1JyxwNN7r3ulRqK7-5NzMFAOTMFrAPvfJkrHaxGm49ZwrWjd5srDLeZa51OXtvl9pnQ2Qia-LoHlcjRzKGdy6tWfVH9DPeF0-YK68Spc7iBF3bFh-dhvKKqxBSKh-uMFO5ECzLyqp~FJLO5S3DQMs4uPjYiddAxHY602fNtLPfX0Can~hTMmNuKf7PM3hEdXpaU9XEJVPb9hn56C1-rn7TgDUcYP-j67p33kkR8MwyWqwuVJCnKVLDRYgBLrVGgtC4CC8rWWCa~40b9VpgSMqbcM4uDn1xI6bLkujFA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA
- Python Paso a paso* - Ángel Pablo Hinojosa Gutiérrez - Google Libros. (n.d.). Retrieved March 21, 2022, from <https://books.google.com.ec/books?id=Uo6fDwAAQBAJ&printsec=frontcover&hl=es#v=onepage&q&f=false>
- Repositorio Institucional de la Universidad Politécnica Salesiana: Codifica en Python.* (n.d.). Retrieved March 21, 2022, from <https://dspace.ups.edu.ec/handle/123456789/19346>