# **Módulo 2**

# **Estructuras de datos secuenciales**

Una estructura de datos secuencial es aquella en la que los elementos se almacenan en un orden específico y se accede a ellos de manera secuencial. Los elementos se organizan uno tras otro en una secuencia, lo que significa que el orden en que se almacenan es significativo.

**Colecciones secuenciales:**

#### **Objetos iterables.**

* Cadenas (str)
* Listas (list)
* Tuplas (tuple)
* slice\_cortes (slice)
* Bytes (bytes) y Bytesarray (bytearray)
* Memoria compartida (memoryview)
* Index

**Manejo de string (cadenas)**

**Objetos tipo str.**

* + Concatenar Cadenas
  + Replicar Cadenas
  + slice ()
  + Slicing
  + Formato de cadenas de texto utilizando caracteres de escape:
  + Formato estilo "C" con el operador "%"
  + Formato con la función format()
  + ﻿﻿Formato con f-string

**Métodos de string**

**Estilos**

* + - capitalize()
    - casefold()
    - lower()
    - swapcase()
    - title()
    - upper()

**Ubicación en consola**

* + - center()
    - ljust()
    - rjust()

**Booleanas**

* + - isalnum(), isalpha(), isascii(), isidentifier(), islower(), isprintable(), isspace(), istitle(), isupper()
    - endswith(), startswith()
    - isdecimal(), isdigit(), isnumeric()

**Búsquedas**

* + - index()
    - rindex()
    - find()
    - rfind()

**Repeticiones**

* + - count()

**Relleno**

* + - zfill()

**Corte**

* + - split()
    - rsplit()
    - splitlines()
    - partition()
    - rpartition()

**Lista a string**

* + - join()

**Reemplazo**

* + - replace()
    - maketrans()
    - translate()

**Recorte**

* + - strip()
    - strip()
    - rstrip()
    - removeprefix()
    - removesuffix()

**Tab space**

* + - expandtadfadfbs()

**reversed()**

* + - como función no como método de listas list().reverse()

**Sorted()**

* + - como función no como método de listas list().sort()

### **range()**

**Colecciones secuenciales:**

* **list**
* **tuples**

**Métodos y atributos:**

**Agregar datos a la lista (las tuplas son inmutables)**

* append()
* insert()
* extend()

**Eliminar datos de una lista (las tuplas son inmutables)**

* pop()
* pop(I)
* remove(D)
* clear()

**Organización de datos de una lista (las tuplas son inmutables)**

* reverse()
* sort()
* lista.sort(reverse=True)

**Métodos count e index**

* count()
* index()

**Pilas y colas**

* FIFO
* LIFO

### **recorrer la estructura con bucle for**

* + Modificación de ejecución en un bucle for.
  + Break:
  + Continue
  + range
  + zip()
  + enumerate()

**Anidación**

# **Estructuras de datos secuenciales**

# **Objetos**

|  |
| --- |
| objeto = 3,14159  ¿que tipo de dato es? |

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | objeto = 3,14159  print (f"{objeto=} es del tipo {type(objeto)}") |   Salida esperada por consola   |  | | --- | | objeto=(3, 14159) es del tipo <class 'tuple'> |   · |

·

|  |
| --- |
| Es una tupla, una colección de dos (2) datos enteros.  objeto = 3,14159  |  La coma separa objetos en una colección  En una tupla los paréntesis son recomendados por la comunidad pero no obligatorios por el interprete Python  Una mejor manera de representar una tupla es la siguiente  objeto = (3,14159)  \/ \\_\_\_/  | |  | 2do dato 14159 tipo entero  1er dato 3 tipo entero |

|  |
| --- |
| Según la ley Argentina (SiMeLA - Sistema Métrico Legal Argentino) la parte entera se separa de la decimal por una coma (“,”) en programación esto no es válido el separador es el punto (“.”) |

Las estructuras de datos en Python son objetos que nos permiten almacenar y organizar datos en un orden específico. Las secuencias son muy útiles para almacenar y manipular conjuntos de datos de manera organizada.

**Python posee tres tipos principales de secuencias en Python: cadenas - strings, listas- list, tuplas – tuples, slice\_cortes – slice**.

|  |  |
| --- | --- |
| **Cadenas (str):** | Las cadenas son secuencias inmutables de caracteres. Cada caracter en una cadena tiene un índice numérico que comienza desde cero. |
| **Listas (list):** | Las listas son colecciones ordenadas y mutables de elementos. Pueden contener diferentes tipos de datos, incluidos números, cadenas, otros objetos e incluso otras listas. |
| **Tuplas (tuple):** | Las tuplas son similares a las listas, pero son inmutables, lo que significa que no pueden modificarse una vez creadas. Se definen usando paréntesis en lugar de corchetes. |
| **slice\_cortes (slice):** | \*(en realidad no es una colección, lo veremos en el punto)  Los slice\_cortes son parámetros para secuencias inmutables de números enteros.  Son útiles para generar secuencias numéricas sin ocupar mucha memoria. |
| **Bytes (bytes) y Bytesarray (bytearray):** | Estos tipos de datos se utilizan para representar secuencias de bytes. Los bytes son inmutables, mientras que bytearray es mutable.  ﻿﻿La principal diferencia entre los tipos de datos bytes y str en Python es que los bytes representan datos en formato binario en lugar de texto, mientras que los str representan datos de texto en formato Unicode.  Otra diferencia importante es la forma en que se almacenan los datos. Las cadenas (str) en Python se representan como una secuencia de caracteres Unicode, mientras que los bytes se representan como una secuencia de valores de bytes.  Por lo tanto, los datos binarios, como archivos de imagen, sonidos o vídeos, se almacenan típicamente como bytes en Python.  Además, mientras que las cadenas pueden ser diseñadas como cadenas literales utilizando comillas simples o dobles, los bytes se crean utilizando el prefijo `b` antes de la cadena literal. |
| **Memoria compartida (memoryview):** | La memoria compartida permite acceder a una secuencia de datos subyacentes en un formato optimizado para operaciones en memoria. |

Una de las grandes fortalezas de Python es su capacidad de realizar iteraciones de forma dinámica a partir de diversos tipos de objetos con la capacidad de ser iterables.

|  |
| --- |
| **Index:**  En el contexto de un string, una lista o tupla en Python, un índice se refiere a la posición numérica de un elemento dentro de la secuencia. Cada elemento en un string, lista o tupla tiene asignado un índice único que lo identifica iniciando en cero (0) y termina en el último N-1. El uso de len cuenta la cantidad de objetos desde 1 a N  En Python, los índices comienzan desde cero, lo que significa que el primer elemento de una lista o tupla tiene un índice de 0, el segundo elemento tiene un índice de 1, y así sucesivamente. Puedes acceder a un elemento específico de un string, una lista o tupla utilizando su índice correspondiente.  Aquí tienes un ejemplo de cómo acceder a elementos individuales utilizando índices:  En el ejemplo anterior, se accede a elementos individuales de la lista mi\_string, mi\_lista y la tupla mi\_tupla utilizando corchetes [] seguidos del índice correspondiente.  Se debe tener en cuenta que los índices deben estar dentro del slice\_corte válido de la lista o tupla. Si intentas acceder a un índice que está fuera de ese slice\_corte, se producirá un error de índice fuera de slice\_corte (list index out of slice). |

·

|  |
| --- |
| Para modificar elementos en listas (mutables) se puede acceder por index.  No es así en string y tuplas que son inmutables |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | lista = [1,2,3,"hola",99,3.14159,True, False]  print (f"el {lista =} es de la módulo {type (lista)=}")  print (f"el {lista[0] =} es de la módulo {type (lista[0])=}")  print (f"el {lista[1] =} es de la módulo {type (lista[1])=}")  print (f"el {lista[2] =} es de la módulo {type (lista[2])=}")  print (f"el {lista[3] =} es de la módulo {type (lista[3])=}")  print (f"el {lista[4] =} es de la módulo {type (lista[4])=}")  print (f"el {lista[5] =} es de la módulo {type (lista[5])=}")  print (f"el {lista[6] =} es de la módulo {type (lista[6])=}")  print (f"el {lista[7] =} es de la módulo {type (lista[7])=}") |   Salida esperada por consola   |  | | --- | | el lista =[1, 2, 3, 'hola', 99, 3.14159, True, False] es de la módulo type (lista)=<class 'list'>  el lista[0] =1 es de la módulo type (lista[0])=<class 'int'>  el lista[1] =2 es de la módulo type (lista[1])=<class 'int'>  el lista[2] =3 es de la módulo type (lista[2])=<class 'int'>  el lista[3] ='hola' es de la módulo type (lista[3])=<class 'str'>  el lista[4] =99 es de la módulo type (lista[4])=<class 'int'>  el lista[5] =3.14159 es de la módulo type (lista[5])=<class 'float'>  el lista[6] =True es de la módulo type (lista[6])=<class 'bool'>  el lista[7] =False es de la módulo type (lista[7])=<class 'bool'> |   · |

·

**Manejo de string (cadenas):**

En Python, los strings (cadenas de caracteres) son objetos inmutables que tienen una serie de métodos y atributos incorporados para manipular y trabajar con ellos.

**Objetos tipo str.**

Python cuenta con varios tipos de objetos relacionados con cadenas de caracteres. El más común es el tipo str, el cual puede incluir cualquier tipo de caracter, incluyendo caracteres de escape.

Los objetos de tipo str son inmutables.

Algunos de los métodos y atributos más comunes de los strings en Python:

|  |
| --- |
| * Un input siempre devuelve un string * Un string es una cadena de caracteres entre comillas simples , dobles o triples. * Un caracter no es solo el abecedario, sino de todo símbolo, letra, numero, etc. de los que esta dentro de las tablas unicode / utf8 según corresponde. * Esto significa que un string puede tenes caracteres en chino, cirílico, números, signos, etc. |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string="HOLA MUNDO IT"  print (string)  print (f'{string}')  print (f'{string=}')  print (f'bienvenidos a Python – {string.istitle()} - sigamos adelante') |   Salida esperada por consola   |  | | --- | | HOLA MUNDO IT  HOLA MUNDO IT  string=HOLA MUNDO IT  bienvenidos a Python – HOLA MUNDO IT - sigamos adelante' |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string="""HOLA  MUNDO  IT"""  print (f'{string}') |   Salida esperada por consola   |  | | --- | | HOLA  MUNDO  IT |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string="""HOLA  MUNDO  IT"""  print (f'{string}') |   Salida esperada por consola   |  | | --- | | HOLA  MUNDO  IT |   · |

·

**Concatenar Cadenas:**

Puedes combinar (concatenar) cadenas utilizando el operador +.

La concatenación como la suma solo se puede dar entre objetos del mismo tipo.

Por ejemplo si tenemos varios strings y queremos unirlos podemos usar el operador de + generando un nuevo objeto string con la concatenacion de los anteriores.

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string1="HOLA MUNDO IT"  string2="Sigamos adelante"  salida1 = string1+string2  print (f'{salida1}')  salida2 = string1+" "+string2  print (f'{salida2}')  salida3 = f"{string1} {string2}"  print (f'{salida3}') |   Salida esperada por consola   |  | | --- | | HOLA MUNDO ITSigamos adelante  HOLA MUNDO IT Sigamos adelante  HOLA MUNDO IT Sigamos adelante |   · |

**Replicar Cadenas:**

Puedes replicar (repetir) cadenas utilizando el operador \*.

La replicación es la repetición de un string una cantidad entera de veces (en caso de ser cero o negativo devuelve un objeto vació)

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string1=" Hola Curso /"  salida1 = string1\*5  print (f'{salida1}') |   Salida esperada por consola   |  | | --- | | Hola Curso / Hola Curso / Hola Curso / Hola Curso / Hola Curso / |   · |

·

**slice ():**

|  |
| --- |
| slice (rangos):  Si bien no es una colección, es una función que devuelve una tupla de parámetros de corte de un objeto iterable. |

·

Es una función que se utiliza para crear un objeto de rebanada (slice object) que puede ser utilizado para acceder a partes específicas de secuencias, objetos iterables (listas, cadenas, tuplas, etc). El objeto de rebanada define un slice\_corte de índices que se pueden utilizar para extraer elementos de una secuencia.

La sintaxis básica de slice() es la siguiente:

### Para iterar un objeto iterable se utiliza la siguiente sintaxis:

|  |
| --- |
| objeto\_slice = slice (inicio, limite\_final , paso\_o\_step)  ^ ^ ^  valor inicial fin paso entre un valor  (incluido) (no incluido) y el siguiente  default =0 obligatorio default =1 |

·

inicio: Índice donde comienza la rebanada. Si no se proporciona, se asume 0.

limite\_final: Índice donde finaliza la rebanada (no se incluye). Si no se proporciona, se asume el final de la secuencia.

paso\_o\_step: Paso de incremento entre elementos en la rebanada. Si no se proporciona, se asume 1.

El slice en Python es una técnica que te permite guardar una porción (subsecuencia) de un objeto iterable. Permite seleccionar un slice\_corte de elementos utilizando un inicio, un fin y un paso.

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | lista =[5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49]  nombre\_slice = slice(5,15,2)  print (f"lista ={lista}")  print (f"lista[slice\_corte] ={lista[nombre\_slice]}") |   Salida esperada por consola   |  | | --- | | lista =[5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49]  lista[slice\_corte] =[15, 19, 23, 27, 31] |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | lista ="Hola mundo IT, ¡¡¡¡¡¡¡¡Python es genial!!!!!!!!"  slice\_corte = slice(23,29,1) # slice(inicio, limite\_final, step)  print (f"slice\_corte (I,F,S) es {slice\_corte=}")  print (f"lista ={lista[slice\_corte]=}") |   Salida esperada por consola   |  | | --- | | slice\_corte (I,F,S) es slice\_corte=slice(23, 29, 1)  lista =lista[slice\_corte]='Python' |   · |

·

### **Slicing**

El slicing en Python es una técnica que te permite extraer una porción (subsecuencia) de una secuencia, como una cadena, una lista o una tupla. Permite seleccionar un slice\_corte de elementos utilizando un inicio, un fin y un paso.

La sintaxis es la siguiente:

### Para iterar un objeto iterable se utiliza la siguiente sintaxis:

|  |
| --- |
| sub\_coleccion = coleccion [inicio, limite\_final , paso\_o\_step]  ^ ^ ^  valor inicial fin paso entre un valor  (incluido) (no incluido) y el siguiente  default =0 obligatorio default =1 |

·

inicio: Índice donde comienza la rebanada. Si no se proporciona, se asume 0.

limite\_final: Índice donde finaliza la rebanada (no se incluye). Si no se proporciona, se asume el final de la secuencia.

paso\_o\_step: Paso de incremento entre elementos en la rebanada. Si no se proporciona, se asume 1.

El slicing en Python es una técnica que te permite extraer una porción (subsecuencia) de una secuencia, como una cadena, una lista o una tupla. Permite seleccionar un slice\_corte de elementos utilizando un inicio, un fin y un paso.

Aquí tienes algunos ejemplos para ilustrar cómo funciona el slicing:

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | cadena = "Python-es-Genial"  print(f"|{cadena[0:6]}| Extrae los carac.desde el índice 0 al 6 (5)") # |Python|  print(f"|{cadena[ :6]}| Extrae los carac.desde el inicio al 6 (5)")# |Python|  print(f"|{cadena[ :-10]}| Extrae los carac.desde el inicio al -10")# |Python|  print(f"|{cadena[6:16]}| Extrae los carac.desde el índice 7 hasta el 16")# |-es-Genial|  print(f"|{cadena[6:]}| Extrae los carac.desde el índice 7 hasta el final")# |-es-Genial|  print(f"|{cadena[-10:]}| Extrae los carac.desde el índice -10 hasta el final")# |-es-Genial|  print(f"|{cadena[6:10]}| Extrae los carac.desde el inicio 6 hasta el 10 (9)")# |-es-|  print(f"|{cadena[-10:-6]}| Extrae los carac.desde el inicio -10 hasta el-6(-7)")# |-es-|  print(f"|{cadena[::2]}| Extrae los carac.con un paso de 2")# |Pto-sGna|  print(f"|{cadena[::-1]}| Invierte el orden") # | laineG-se-nohtyP| |   Salida esperada por consola   |  | | --- | | |Python| Extrae los caracteres desde el índice 0 al 6 (5)  |Python| Extrae los caracteres desde el inicio al 6 (5)  |Python| Extrae los caracteres desde el inicio al -10  |-es-Genial| Extrae los caracteres desde el índice 7 hasta el 16  |-es-Genial| Extrae los caracteres desde el índice 7 hasta el final  |-es-Genial| Extrae los caracteres desde el índice -10 hasta el final  |-es-| Extrae los caracteres desde el inicio 6 hasta el 10 (9)  |-es-| Extrae los caracteres desde el inicio -10 hasta el-6(-7)  |Pto-sGna| Extrae los caracteres con un paso de 2  |laineG-se-nohtyP| Invierte el orden |   · |

·

Recuerda que el índice de inicio y fin pueden ser números negativos, lo que indica que se cuentan desde el final de la secuencia.

El slicing en Python es una herramienta muy útil para manipular secuencias y extraer partes específicas según tus necesidades.

### **Formato de cadenas de texto utilizando caracteres de escape:**

En las primeras versiones de Python, se utilizaba el formato de cadenas de texto mediante caracteres de escape. En este formato, se usaban caracteres especiales como \n para representar un salto de línea o \t para representar un tabulador.

Puedes utilizar caracteres de escape, como \n para una nueva línea o \t para una tabulación, en tus cadenas.

texto\_multilinea = "Línea 1\nLínea 2"

tabulacion = "Texto\tTabulado"

Estos son solo algunos ejemplos del uso de cadenas en Python. Las cadenas son un componente esencial en la programación y se utilizan en una amplia variedad de tareas, desde manipulación de texto hasta la comunicación con bases de datos y la creación de interfaces de usuario.

**Usa todos tus sentidos para averigua para que sirve el caracter de escape \a.**

### **Formato estilo "C" con el operador "%"**

**Este método estilo C de generar un string a patir de otros strings, numetos y demas objetos mediante concatenar, replicar y demás esta totalmente desactualizada y fuera de uso.**

﻿﻿El operador % para formatear cadenas de texto ha estado disponible en Python desde las primeras versiones del lenguaje

* %c caracter
* %s Cadena
* %i Entero Decimal con Signo
* %d Entero Decimal con Signo
* %u Entero Decimal sin Signo
* %o Entero Octal
* %x Entero Hexadecimal en minúsculas
* %X Entero Hexadecimal en mayúsculas
* %e Notación Exponencia con minúscula 'e'
* %E Notación Exponencia con mayúscula 'E'
* %f Numero Real en punto flotante
* %g Redondea la última cifra decimal a partir de .5
* %G Redondea la última cifra decimal a partir de .5

**Formato con la función format()**

﻿﻿El método `format()` de la clase `str` se utiliza para formatear cadenas de texto con valores dinámicos en tiempo de ejecución.

Básicamente, lo que hace este método es tomar los argumentos que le pasas y los introduce en una cadena de texto, reemplazando ciertas partes de la cadena de texto por valores específicos. Estos valores pueden ser números, cadenas de texto, otros objetos, etc.

Puedes usar llaves `{}` en las cadenas de texto como marcadores de posición para donde quieres agregar los argumentos, y luego usar el método `format()` para pasar esos valores en tiempo de ejecución.

Además de simplemente insertar valores en una cadena de texto, el método `format()` también te permite especificar cómo quieres que se muestren esos valores. Por ejemplo, podrías señalar que un número en particular se muestre con un número específico de decimales, o que una cadena de texto se alinee a la izquierda o a la derecha en un cierto ancho.

nombre = "Bob"

edad = 30

mensaje = "Hola, soy {} y tengo {} años.".format(nombre, edad)

**Formato con f-string:**

En Python 3.6 se introdujeron los literales de cadena de texto formateados (f-strings). Este formato utiliza el prefijo f antes de la cadena de texto y permite incluir expresiones Python dentro de las llaves {}. También esta el prefijo r que raw-string que muestra el string obviando todo caracter de formato.

**Este método f"{}"de generar un string a patir de otros strings, numetos y demas objetos mediante concatenar, replicar y demás esta la mas actual y recomendada por la comunudad Python.**

nombre = "Ariel"

edad = 49

# O con f-strings

mensaje = f"Soy {nombre} y tengo {edad} años."

**Métodos de string:**

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string ="Ingresa su apellido: "  print (f"métodos y atributos de string :\n {**dir**(string)}") |   Salida esperada por consola   |  | | --- | | * ['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', **'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format\_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']** |   · |

·

|  |
| --- |
| Introducimos los métodos de string pero en vez de ir por orden alfabético (como esta en el dir)  lo hacemos mediante una división con respecto al resultado. |

<class 'str'> =

**Estilos**  ['capitalize', 'casefold', 'lower', 'swapcase', 'title', 'upper']

• capitalize() Convierte al primer caracter del texto en mayúscula y el resto en minúsculas

• casefold() Convierte la cadena de caracteres en minúsculas.

• lower() Convierte la cadena de caracteres en minúsculas.

• swapcase() Transforma los caracteres que están en minúsculas a mayúsculas y viceversa

• title() Convierte al primer caracter de cada palabra en mayúsculas y los otros en minúsculas.

• upper() Convierte la cadena de caracteres en mayúsculas.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string='HoLa MuNdO It. CuRsO pYtHoN'  print (f'{string.capitalize()=}')  print (f'{string.casefold()=}')  print (f'{string.lower()=}')  print (f'{string.swapcase()=}')  print (f'{string.title()=}')  print (f'{string.upper()=}') |   Salida esperada por consola   |  | | --- | | string.capitalize()='Hola mundo it. curso python'  string.casefold()='hola mundo it. curso python'  string.lower()='hola mundo it. curso python'  string.swapcase()='hOlA mUnDo iT. cUrSo PyThOn'  string.title()='Hola Mundo It. Curso Python'  string.upper()='HOLA MUNDO IT. CURSO PYTHON' |   · |

·**Ubicación en consola** ['center','ljust', 'rjust', ]

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string='datos'  print (f'|',string.center(20),'|')  print (f'|',string.ljust(20),'|')  print (f'|',string.rjust(20),'|') |   Salida esperada por consola   |  | | --- | | **| datos |**  **| datos |**  **| datos |** |   · |

**·**

**Booleanas**

['isalnum', 'isalpha', 'isascii', 'isidentifier', 'islower', 'isprintable', 'isspace', 'istitle','isupper',

'endswith','startswith',

'isdecimal', 'isdigit', 'isnumeric']

Código Python - Solo tiene una salida booleana. True o False

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string='Hola mundo Python 3'  print (f'{string.isalnum()=}')  print (f'{string.isalpha()=}')  print (f'{string.isascii()=}')  print (f'{string.isidentifier()=}')  print (f'{string.islower()=}')  print (f'{string.isupper()=}')  print (f'{string.istitle()=}')  print (f'{string.isprintable()=}')  print (f'{string.isspace()=}') |   Salida esperada por consola   |  | | --- | | string.isalnum()=False  string.isalpha()=False  string.isascii()=True  string.isidentifier()=False  string.islower()=False  string.isupper()=False  string.istitle()=False  string.isprintable()=True  string.isspace()=False |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string="HOLA MUNDO IT"  print (f'{string.islower()=}')  print (f'{string.isupper()=}')  print (f'{string.istitle()=}') |   Salida esperada por consola   |  | | --- | | string.islower()=False  string.isupper()=True  string.istitle()=False |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string=" "  print (f'{string.isspace()=}')  string="Hola Python"  print (f'{string.isspace()=}') |   Salida esperada por consola   |  | | --- | | string.isspace()=True  string.isspace()=False |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string='Hey mundo IT'  print (f'{string.startswith( "Hola")=}')  print (f'{string.endswith( "Python")=}') |   Salida esperada por consola   |  | | --- | | string.startswith( "Hola")=False  string.endswith( "Python")=False |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string='Hola mundo Python'  print (f'{string.startswith( "Hola")=}')  print (f'{string.endswith( "Python")=}') |   Salida esperada por consola   |  | | --- | | string.startswith( "Hola")=True  string.endswith( "Python")=True |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string="UNO DOS TRES"  print (f'{string.isdecimal()=}')  print (f'{string.isdigit()=}')  print (f'{string.isnumeric()=}') |   Salida esperada por consola   |  | | --- | | string.isdecimal()=False  string.isdigit()=False  string.isnumeric()=False |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string="123"  print (f'{string.isdecimal()=}')  print (f'{string.isdigit()=}')  print (f'{string.isnumeric()=}') |   Salida esperada por consola   |  | | --- | | string.isdecimal()=True  string.isdigit()=True  string.isnumeric()=True |   · |

· **Búsquedas** ['index','rindex','find','rfind']

En Python, los métodos .index() y .rindex() son similares a los métodos .find() y .rfind(), respectivamente. Ambos se utilizan para buscar una subcadena dentro de una cadena y devolver la posición en la que se encuentra.

**La diferencia clave entre ellos en términos de manejo de errores**

['index','rindex']

|  |  |  |
| --- | --- | --- |
| **index** Se utiliza para busca de izquierda a derecha una subcadena o caracter que es ingresado como argumento y regresa el índice del primero que encuentra. En caso de no encontrarlo regresará **ValueError**  **rindex** Se utiliza para busca de derecha a izquierda una subcadena o caracter que es ingresado como argumento y regresa el índice del primero que encuentra. En caso de no encontrarlo regresará **ValueError**   |  | | --- | | string="Hola alumnos, cómo están, los estaba extrañando, gracias por entrar a tiempo "  primer\_index=string.index(',')  print (f'{primer\_index=}')  ultimo\_index=string.rindex(',')  print (f'{ultimo\_index=}') |   Salida esperada por consola   |  | | --- | | primer\_index=12  ultimo\_index=47  # Hola alumnos, cómo están, los estaba extrañando, gracias por entrar a tiempo  # ^ ^  # 12 47 |   · |

· ['find','rfind']

|  |  |  |
| --- | --- | --- |
| **find** Se utiliza para busca de izquierda a derecha una subcadena o caracter que es ingresado como argumento y regresa el índice del primero que encuentra. En caso de no encontrarlo regresará el valor **-1**  **rfind** Se utiliza para busca de derecha a izquierda una subcadena o caracter que es ingresado como argumento y regresa el índice del primero que encuentra. En caso de no encontrarlo regresará el valor **-1**   |  | | --- | | string="Hola alumnos, cómo están, los estaba extrañando, gracias por entrar a tiempo "  primer\_index=string.index(',')  print (f'{primer\_index=}')  ultimo\_index=string.rindex(',')  print (f'{ultimo\_index=}') |   Salida esperada por consola   |  | | --- | | primer\_index=12  ultimo\_index=47  # Hola alumnos, cómo están, los estaba extrañando, gracias por entrar a tiempo  # ^ ^  # 12 47 |   · |

·**Repeticiones**  ['count']

|  |  |  |
| --- | --- | --- |
| Código Python  Se utiliza para contar el número de ocurrencias de una subcadena o caracter en una cadena más grande. Este método es útil cuando deseas saber cuántas veces aparece una subcadena o caracter específica en un texto.   |  | | --- | | string="""Sabe, si alguna vez tus labios rojos  quema invisible atmósfera abrasada,  que el alma que hablar puede con los ojos  también puede besar con la mirada. """  repeticiones=string.count('os')  print (f'{repeticiones=}') |   Salida esperada por consola   |  | | --- | | repeticiones=4 |   · |

·**Relleno** ['zfill']

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string="12345"  print (f'{string.zfill(10)=}') |   Salida esperada por consola   |  | | --- | | string.zfill(10)='0000012345' |   · |

·**Corte**  ['split','rsplit','splitlines','partition',rpartition']

Se utiliza para dividir una cadena en una **lista** de subcadenas, basándose en un delimitador específico. El delimitador es un caracter o una secuencia de caracteres que se utiliza para determinar dónde se deben realizar las divisiones en la cadena.

|  |  |  |
| --- | --- | --- |
| Código Python  Split() sin un argumento, lo que dividirá la cadena en espacios en blanco (espacios, tabulaciones y saltos de línea) por defecto.   |  | | --- | | string="Hola mundo IT. Python es genial"  lista=string.split()# .split(' ') sin parametros es lo mismo que ' ' espacios vacíos  print (f'{lista=}, {type (lista)}') |   Salida esperada por consola   |  | | --- | | lista=['Hola', 'mundo', 'IT.', 'Python', 'es', 'genial'], <class 'list'> |   ·En caso de no encontrar al patrón, regresará un objeto de tipo list, conteniendo al objeto str original. |

·

|  |  |  |
| --- | --- | --- |
| Código Python  Se utiliza para dividir una cadena en una **lista** de subcadenas, basándose en un delimitador específico, y la cantidad de veces que se cuenta de izquierda a derecha, el resto sigue como ultimo elemento en la lista.   |  | | --- | | string="Hola mundo IT. Python es genial"  lista=string.split(' ',3))# solo 3 ' ' espacios vacíos de izq a derecha  print (f'{lista=}, {type (lista)}') |   Salida esperada por consola   |  | | --- | | lista=['Hola', 'mundo', 'IT.', 'Python es genial'], <class 'list'>  # 1 2 3 \\_\_\_\_\_\_4\_\_\_\_\_/ |   · |

·  ['rsplit']

|  |  |  |
| --- | --- | --- |
| Código Python  Se utiliza para dividir una cadena en una **lista** de subcadenas, basándose en un delimitador específico, y la cantidad de veces que se cuenta de derecha a izquierda, el resto sigue como `primer elemento en la lista.   |  | | --- | | string="Hola mundo IT. Python es genial"  lista=string.rsplit(' ',3))# solo 3 ' ' espacios vacíos de derecha a izq  print (f'{lista=}, {type (lista)}') |   Salida esperada por consola   |  | | --- | | lista=['Hola mundo IT.', 'Python', 'es', 'genial'], <class 'list'>9  # \\_\_\_\_\_1\_\_\_\_\_/ 2 3 4 |   · |

·Los metodos split y rsplit tienen el paramatro maxsplit=-1 que se puede modificar cuantas coincidencias buscamos, con -1 encuentra todas las posibles\*\*

|  |  |  |
| --- | --- | --- |
| Código Python  Se corta por las comas ',' y los espacios después de la coma también se eliminan en las subcadenas resultantes.   |  | | --- | | string="Hola alumnos, cómo están, los estaba extrañando, gracias por entrar a tiempo "  lista=string.split(',')  print (f'{lista=}, {type (lista)}') |   Salida esperada por consola   |  | | --- | | lista=['Hola alumnos', ' cómo están', ' los estaba extrañando', ' gracias por entrar a tiempo '], <class 'list'> |   · |

·**Corte linea nueva** ['splitlines']

|  |  |  |
| --- | --- | --- |
| Se utiliza para dividir una cadena en una lista de líneas. Esta función es útil cuando trabajas con cadenas de varias líneas y deseas separarlas en líneas individuales. Cada línea se considera como una cadena independiente en la lista resultante.   |  | | --- | | string="""Hola alumnos,  cómo están,  los estaba extrañando,  gracias por entrar a tiempo """  lista=string.splitlines()  print (f'{lista=}, {type (lista)}') |   Salida esperada por consola   |  | | --- | | lista=['Hola alumnos, ', 'cómo están,', ' los estaba extrañando, ', 'gracias por entrar a tiempo '], <class 'list'> |   · |

·**Corte por palabra** ['partition',rpartition']

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string="""Sabe, si alguna vez tus labios rojos  quema invisible atmósfera abrasada,  que el alma que hablar puede con los ojos  también puede besar con la mirada. """  lista\_corte\_left=string.partition('ojo')  print (f'{lista\_corte\_left=}')  lista\_corte\_right=string.rpartition('ojo')  print (f'{lista\_corte\_right=}') |   Salida esperada por consola   |  | | --- | | lista\_corte\_left=('Sabe, si alguna vez tus labios r', 'ojo', 's\nquema invisible atmósfera abrasada,\nque el alma que hablar puede con los ojos\n también puede besar con la mirada. ')  lista\_corte\_right=('Sabe, si alguna vez tus labios rojos\nquema invisible atmósfera abrasada,\nque el alma que hablar puede con los ', 'ojo', 's\n también puede besar con la mirada. ') |   · |

·**Lista a string** ['join']

|  |  |  |
| --- | --- | --- |
| Se utiliza para concatenar una lista de cadenas utilizando un separador específico. Este método es especialmente útil cuando tienes una lista de elementos y deseas combinarlos en un solo string o cadena.   |  | | --- | | lista=['Hola', 'mundo', 'IT.', 'Python', 'es', 'genial']  string=str(' ' ).join(lista)  print (f'{string=}') |   Salida esperada por consola   |  | | --- | | string='Hola mundo IT. Python es genial' |   · |

·**Reemplazo** ['replace', 'maketrans', 'translate']

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string="Hola Mundo Python"  print (f'{string.replace("Hola","Grande")=}') |   Salida esperada por consola   |  | | --- | | string.replace("Hola","Grande")=' Grande Mundo Python' |   · |

· [''maketrans', 'translate']

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Código Python  En Python, el método .translate() se utiliza para reemplazar caracteres individuales o conjuntos de caracteres en una cadena utilizando una tabla de traducción. Esta función es útil para realizar transformaciones de caracteres en una cadena, como cambiar letras a mayúsculas o minúsculas, eliminar caracteres no deseados, o reemplazar caracteres específicos por otros.  El método .translate() requiere el uso de una tabla de traducción, que generalmente se crea utilizando el método str.maketrans() o la función str.translate() en combinación con la función str.maketrans().   |  | | --- | | # Crear una tabla de traducción para cambiar "á" por "a" y "é" por "e", etc  tabla\_de\_traduccion = str.maketrans("áéíóú", "aeiou")  string = """El perro ladró, la vaca calló, el gato maulló y el pato, que no daba crédito, corrió todo lo que pudo. Fue así como transcurrió otro apacible día en la granja. Bueno, quizás, debí decir un día raro, y es que el león, que se había escapado de la selva, estaba apunto de hacer su entrada"""  string\_sin\_acentos = string.translate(tabla\_de\_traduccion)  print(f"{string\_sin\_acentos=}") |   Salida esperada por consola   |  | | --- | | string\_sin\_acentos='El perro ladro, la vaca callo, el gato maullo y el pato, que no daba credito, corrio todo lo que pudo. Fue asi como transcurrio otro apacible dia en la granja. Bueno, quizas, debi decir un dia raro, y es que el leon, que se habia escapado de la selva, estaba apunto de hacer su entrada' |   ·También puedes usar .translate() con str.maketrans() para eliminar caracteres específicos de la string  .   |  | | --- | | tabla\_de\_traduccion = str.maketrans("", "", "¡!@#$%^&\*()")  string = "¡Hola! ¿Cómo estás? ¡¡¡¡¡Python es genial!!!!!"  string\_limpio = string.translate(tabla\_de\_traduccion)  print(f"{string\_limpio=}") |   ·   |  | | --- | | string\_limpio='Hola ¿Cómo estás? Python es genial |   .maketrans()  Este método permite crear una tabla de caracteres, la cual consiste en un diccionario que relaciona a un par de caracteres por su número correspondiente en la codificación.  .translate()  Este método sustituye los caracteres a partir de una tabla de caracteres. |

·**Recorte**  ['strip','lstrip','rstrip','removeprefix', 'removesuffix'

· **de puntas caracteres**

['strip','lstrip','rstrip']

|  |  |  |
| --- | --- | --- |
| Código Python  Se elimina los espacios en blanco en blanco (u otros caracteres especificados) del principio y el final de una cadena. Esto es útil para limpiar y normalizar cadenas que pueden tener espacios o caracteres no deseados alrededor.   |  | | --- | | string=" Hola, mundo! "  sin\_puntas=string.strip()  print (f'{string=}')  print (f'{sin\_puntas=}') |   Salida esperada por consola   |  | | --- | | string=' Hola, mundo! '  sin\_puntas='Hola, mundo!' |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python  Se elimina los espacios en blanco en blanco (u otros caracteres especificados) del principio y el final de una cadena. Esto es útil para limpiar y normalizar cadenas que pueden tener espacios o caracteres no deseados alrededor.   |  | | --- | | string="//\*\*/\*///\*\*\*Hola, mundo!/////\*"  sin\_puntas=string.strip("/\*")  print (f'{string=}')  print (f'{sin\_puntas=}') |   Salida esperada por consola   |  | | --- | | string='//\*\*/\*///\*\*\*Hola, mundo!/////\*'  sin\_puntas='Hola, mundo!' |   · |

·**Sin\_punta izq o der** ['lstrip','rstrip']

|  |  |  |
| --- | --- | --- |
| Código Python  lstrip()  Elimina los espacios vacíos o las que este en el argumento que se encuentran a la izquierda de un texto.  rstrip()   |  | | --- | | string="\*\*\*\*\*Hola, mundo!\*\*\*\*\*"  sin\_puntas\_left=string.lstrip("\*")  sin\_puntas\_right=string.rstrip("\*")  print (f'{string=}')  print (f'{sin\_puntas\_left=}')  print (f'{sin\_puntas\_right=}') |   Salida esperada por consola   |  | | --- | | string='\*\*\*\*\*Hola, mundo!\*\*\*\*\*'  sin\_puntas\_left='Hola, mundo!\*\*\*\*\*'  sin\_puntas\_right='\*\*\*\*\*Hola, mundo!' |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python  lstrip()  Elimina los espacios vacíos o las que este en el argumento que se encuentran a la izquierda de un texto.  rstrip()  Elimina los espacios vacíos o las que este en el argumento que se encuentran a la derecha de un texto.   |  | | --- | | string="\*\*\*\*\*Hola, mundo!\*\*\*\*\*"  sin\_puntas\_left=string.lstrip("\*")  sin\_puntas\_right=string.rstrip("\*")  print (f'{string=}')  print (f'{sin\_puntas\_left=}')  print (f'{sin\_puntas\_right=}') |   Salida esperada por consola   |  | | --- | | string='\*\*\*\*\*Hola, mundo!\*\*\*\*\*'  sin\_puntas\_left='Hola, mundo!\*\*\*\*\*'  sin\_puntas\_right='\*\*\*\*\*Hola, mundo!' |   · |

·**Recorte**

· **de puntas strings**

·['removeprefix', 'removesuffix']

También se puede eliminar elementos de un string mediante .replace() y .traslate()

|  |  |  |
| --- | --- | --- |
| Elimina un prefijo o sufijo específico (si existe) en la cadena.  Código Python   |  | | --- | | string="Hola Mundo IT. Python es Genial"  print (f'{string.removeprefix("Hola")=}')  print (f'{string.removesuffix("Genial")=}')  print (f'{string.removeprefix("Genial")=}')  print (f'{string.removesuffix("Hola")=}') |   Salida esperada por consola   |  | | --- | | string.removeprefix("Hola")=' Mundo IT. Python es Genial'  string.removesuffix("Genial")='Hola Mundo IT. Python es '  string.removeprefix("Genial")='Hola Mundo IT. Python es Genial'# Nada que hacer  string.removesuffix("Hola")='Hola Mundo IT. Python es Genial'# Nada que hacer |   · |

**Tab space**

·['expandtadfadfbs']

Tamaño de tabulación en caracteres expandtadfadfbs()

Por default son 4 para la tabulación estándar

**reversed() - como función**

**no como método de listas list().reverse():**

La función incorporada que se utiliza para obtener un iterador que recorre una secuencia en orden inverso, es decir, desde el último elemento hasta el primero.

Esta función es útil cuando necesitas iterar sobre una secuencia en sentido contrario.

|  |
| --- |
| Devuelve un objeto reversed |

La sintaxis básica de reversed() es la siguiente:

|  |
| --- |
| reversed(sequence) |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | lista =[5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]  print (f"lista = {lista}")  print (f"reversed(string) = {reversed(lista)}")  print (f"list(reversed(string)) = {list(reversed(lista))}") |   Salida esperada por consola   |  | | --- | | lista = [5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]  reversed(string) = <list\_reverseiterator object at 0x00000223C1A4FFD0>  list(reversed(string)) = [29, 27, 25, 23, 21, 19, 17, 15, 13, 11, 9, 7, 5] |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string ="Hola mundo IT, ¡¡¡¡¡¡¡¡Python es genial!!!!!!!!"  reverso = reversed(string)  print (f"string ={str().join(reverso)=}") |   Salida esperada por consola   |  | | --- | | string =str().join(reverso)='!!!!!!!!laineg se nohtyP¡¡¡¡¡¡¡¡ ,TI odnum aloH' |   · |

·

**Sorted**

La función incorporada que se utiliza para ordenar un objeto iterable en un nuevo orden.

La función devuelve una nueva lista que contiene los elementos ordenados de la secuencia original, manteniendo la secuencia original intacta.

|  |
| --- |
| Devuelve un objeto sorted |

**La sintaxis básica de reversed() es la siguiente:**

|  |
| --- |
| sorted(iterable)  sorted(iterable, key=None, reverse=False)  iterable: La secuencia que se desea ordenar.  **key:** Opcional para determinar la clave de ordenación de cada elemento.  **default None.**  **reverse:** Booleano opcional que indica si sedebe ordenarse en inverso.  **default False.** |

La función sorted() es muy útil para ordenar datos en diferentes situaciones y proporciona flexibilidad en términos de personalización de la clave de ordenación y el orden resultante.

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | lista =[5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]  print (f"lista = {lista}")  print (f"sorted(string) = {sorted(lista)}")  print (f"list(sorted(string)) = {list(sorted(lista))}")  print (f"list(sorted(string)) = {list(sorted(lista, reverse=True))}") |   Salida esperada por consola   |  | | --- | | lista = [5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]  sorted(string) = [5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]  list(sorted(string)) = [5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]  list(sorted(string)) = [29, 27, 25, 23, 21, 19, 17, 15, 13, 11, 9, 7, 5] |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string ="Hola mundo IT, ¡¡¡¡¡¡¡¡Python es genial!!!!!!!!"  print (f"string = {string}")  print (f"{str().join(sorted(string))=}")  print (f"{str().join(sorted(string, reverse=True))=}") |   Salida esperada por consola   |  | | --- | | string = Hola mundo IT, ¡¡¡¡¡¡¡¡Python es genial!!!!!!!!  str().join(sorted(string))=' !!!!!!!!,HIPTaadeeghillmnnnooostuy¡¡¡¡¡¡¡¡'  str().join(sorted(string, reverse=True))='¡¡¡¡¡¡¡¡yutsooonnnmllihgeedaaTPIH,!!!!!!!! ' |   · |

**·**

El parámetro key se puede utilizar para proporcionar una función que determine la clave de ordenación para cada elemento.

·

|  |  |  |
| --- | --- | --- |
| Ejemplo, ordenar por el segundo elemento de cada tupla, en una lista  Código Python   |  | | --- | | tuplas = [(1, 3), (5, 1), (2, 7)]  tuplas\_ordenadas = sorted(tuplas, key=lambda x: x[1])  print (f"{tuplas\_ordenadas}") |   Salida esperada por consola   |  | | --- | | [(5, 1), (1, 3), (2, 7)] |   · |

·

### **range.**

Es una función incorporada que se utiliza para generar una secuencia de números enteros dentro de un rango específico.

Esta función es comúnmente utilizada en bucles for para generar una secuencia de valores que se utilizarán en cada iteración del bucle.

### Para iterar un objeto iterable se utiliza la siguiente sintaxis:

|  |
| --- |
| objeto\_range = range (inicio, limite\_final , paso\_o\_step)  ^ ^ ^  valor inicial fin paso entre un valor  (incluido) (no incluido) y el siguiente  default =0 obligatorio default =1 |

·

inicio: Índice donde comienza la rebanada. Si no se proporciona, se asume 0.

limite\_final: Índice donde finaliza la rebanada (no se incluye). Dato obligatorio

paso\_o\_step: Paso de incremento entre elementos en la rebanada. Si no se proporciona, se asume 1.

·

|  |
| --- |
| range (rangos):  Si bien no es una colección, es una función que devuelve una secuencia inmutable de números. |

·

|  |  |
| --- | --- |
| range (rangos):  Se utiliza principalmente en bucles para iterar sobre un rango de valores.  La sintaxis básica es la siguiente:   |  | | --- | | una range tiene 3 parámetros (dos pueden omitirse)  nombre\_range = range(inicio, limite\_final, step)  nombre\_ range = range(5,50,2)  desde 5  hasta 50 no incluido 99 real  de 2 en 2  nombre\_ range = range(0,20,1)  nombre\_ range = range(0,20)  nombre\_ range = range(20) |   **Devuelve una secuencia inmutable de números.** |

·

|  |
| --- |
| La función range es complemento fundamental para la estructura for, range nos permites generar rangos de Números.  Para definir rangos numéricos con la función range().   * range(inicio, limite\_final, step) :   rango >= inicio ,  rango < limite\_final (no pertenece al rango) ,  paso = step.   * range(inicio, limite\_final) :   rango >= inicio ,  rango < limite\_final (no pertenece al rango) ,  ~~paso = 1.~~ por default   * range(limite\_final) :   ~~rango = 0 ,~~por default  rango < limite\_final (no pertenece al rango) ,  ~~paso = 1.~~por default |

ejemplos :

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | nombre\_range = range(5,50,2)  print (f"rango ={nombre\_range}")  print (f"lista ={list(nombre\_range)}") |   Salida esperada por consola   |  | | --- | | rango =range(5, 50, 2)  lista =[5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49] |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | rango = range(0,20,1) # range(inicio, limite\_final, step)  print (f"rango (I,F,S) es {rango=}")  rango = range(0,20) # range(inicio, limite\_final,**~~step~~**)  # se omite el , step cuando es igual a 1 – None  print (f"rango (I,F) es {rango=}")  rango = range(20)# range(**~~inicio~~**, limite\_final, **~~step~~**)  # se omite el , inicio cuando es igual a 0 – None  print (f"rango (I) es {rango=}")  print (f"lista ={slice\_corte=}") |   Salida esperada por consola   |  | | --- | | rango (I,F,S) es rango=range(0, 20)  rango (I,F) es rango=range(0, 20)  rango (I) es rango=range(0, 20)  lista =[0,1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19] |   · |

·

**Colecciones secuenciales:**

Una colección en Python es un tipo de objeto que puede contener múltiples elementos u objetos. Estas colecciones son utilizadas para almacenar, organizar y manipular conjuntos de datos de manera eficiente.

Hay diferentes tipos de colecciones integradas en su biblioteca estándar, que son ampliamente utilizadas en programación.

#### **Objetos iterables (list,tuple,dic,str,df,etc).**

Algunos de estos objetos son los de tipo:

* list
* tuples

**List y Tuples (con index):**

|  |  |  |
| --- | --- | --- |
| En Python, las listas y las tuplas son dos tipos de datos usados para almacenar múltiples valores, pero tienen diferencias importantes. Las listas son mutables, lo que significa que pueden ser modificadas después de creadas, mientras que las tuplas son inmutables, lo que significa que no pueden ser modificadas después de creadas.  Las listas también permiten elementos duplicados, mientras que las tuplas también permiten elementos duplicados, pero son más adecuadas para almacenar valores que no deban ser modificados, como las dimensiones de un rectángulo o los días de la semana.  **Son colecciones desordenadas, sin index, únicos y mutables de objetos separados por comas.**  Ordenadas  Con index Tienen un index o indice.  separados por comas.  **Listas** Mutables  **Tuplas** Inmutables  Estos objetos que la componen pueden ser de cualquier tipo (str, int, float, bool y…...colecciones listas,tuplas, diccionarios, sets, frozensets,etc)  La sintaxis básica es la siguiente:   |  |  | | --- | --- | | **una lista vacía**  nombre\_lista = **list**()  o  nombre\_lista = []  **una lista con objetos int**  nombre\_ lista = [1,2,3,4,5,6,7,8,9,0] | **Una tupla vacía**  nombre\_tupla = **tuple**()  o  nombre\_tupla = ()  -------**Absurda**, luego no se puede modificar-------  **una frozenset con objetos int**  nombre\_ tupla = (1,2,3,4,5,6,7,8,9,0) |   · |

·

|  |
| --- |
| Los conjuntos se utilizan para almacenar valores, permiten realizar operaciones de manipulacion de datos con index y slicing.  **características:**  **Ordenados:**  Los conjuntos mantienen un orden específico para sus elementos. Puedes acceder a los elementos mediante índices.  **Mutables:**  Las **listas** son **mutables**, lo que significa que puedes agregar y eliminar elementos después de haber creado la colección.  Las **Tuplas** son **inmutables**, lo que significa que no puedes agregar ni eliminar elementos después de haber creado la colección.  **Creación de conjuntos:**  **list:** Puedes crear una colecció usando paréntesis () o la función list().  **tuple:** Puedes crear una colecció usando corchetes [] o la función tuple(). |

·

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **tuplas**  Código Python   |  | | --- | | mi\_tupla = ("A","B","C","D","E","F")  print (f"el contenido de {mi\_tupla=}\t{type(mi\_tupla)}")  print (f"los métodos disponibles son {dir(mi\_tupla)=}") |   Salida esperada por consola   |  | | --- | | el contenido de mi\_tupla=('A', 'B', 'C', 'D', 'E', 'F') <class 'tuple'>  los métodos disponibles son dir(mi\_tupla)=['\_\_add\_\_', '\_\_class\_\_', '\_\_class\_getitem\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'count', 'index'] |   ·  **listas**  Código Python   |  | | --- | | mi\_lista = ["A","B","C","D","E","F"]  print (f"el contenido de {mi\_lista=}\t{type(mi\_lista)}")  print (f"los métodos disponibles son {dir(mi\_lista)=}") |   Salida esperada por consola   |  | | --- | | el contenido de mi\_lista=['A', 'B', 'C', 'D', 'E', 'F'] <class 'list'>  los métodos disponibles son dir(mi\_lista)=['\_\_add\_\_', '\_\_class\_\_', '\_\_class\_getitem\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_', '\_\_imul\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort'] |   ·· |

·

**Métodos y atributos:**

|  |
| --- |
| Las tuplas son más eficientes en cuanto al uso de la memoria y al rendimiento, lo que las hace más adecuadas para almacenar grandes cantidades de datos. |

·

|  |
| --- |
| Los métodos de las **tuplas** se incluyen en los de las **listas**.  Ademas las **listas** poseen los siguientes métodos ['append', 'clear', 'copy', 'extend', 'insert', 'pop', 'remove', 'reverse', 'sort'] que son todos aquellos que permiten modificar el objeto frozenset ya que es inmutable. |

**tuple (Tuplas):**

|  |
| --- |
| **una tupla con objetos int**  nombre\_tupla = (1,2,3,4,5,6,7,8,9,0)  **una tupla con objetos float**  nombre\_tupla = (3.14159 , 2.99792458 , 4.5)  **una tupla con objetos str**  nombre\_tupla = ("Hola","Mundo","IT")  **una tupla con objetos bool**  nombre\_tupla = (True,False,True,False)  **una tupla con objetos mix**  nombre\_tupla = (1, 2, 3.14159, "Hola", True). |

·

|  |
| --- |
| Len devuelve la cantidad de objetos que contienen en esta caso la colección |

**·**

|  |
| --- |
| Métodos y atributos utilizables de forma simple son los que no poseen doble guion delante y detrás. Los dejaremos par mas adelante como los constructores de módulos |

.

|  |
| --- |
| Los métodos de las tuplas se incluyen dentro de los de las lista, por lo que solo es necesarios explicarlos una vez. |

.

**Agregar datos a la lista (las tuplas son inmutables)**

* + append() agrega un solo dato al final de la lista
  + insert() agrega un solo dato en el lugar que se indique por un index
  + extend() agrega una colección de datos al final de la lista original

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | lista = [1,2,3,"hola",99,3.14159,True, False]  print (f"el contenido de original {lista=}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  # agrego dato(s)  lista.append("chau")#al final  print (f"lista.append('chau'):\n{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.insert(2,"dato insertado")# en el lugar solicitado  print (f"lista.insert(2,'dato insertado'):{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  # agrego una colección de datos  lista\_ext = ["A","B","C","D","E","F"]  lista.extend(lista\_ext)  print (f"lista.extend(lista\_ext):\n{lista}\n\tcantidad de elementos:{len(lista)}") |   Salida esperada por consola   |  | | --- | | el contenido de original lista=[1, 2, 3, 'hola', 99, 3.14159, True, False]  cantidad de elementos:8  lista.**append**('chau'):  [1, 2, 3, 'hola', 99, 3.14159, True, False,  **'chau'**]  cantidad de elementos:9  lista.**insert**(2,'dato insertado'):  [1, 2, **'dato insertado'**, 3, 'hola', 99, 3.14159, True, False,'chau']  cantidad de elementos:10  lista.**extend**(lista\_ext):  [1, 2, 'dato insertado', 3, 'hola', 99, 3.14159, True, False, 'chau', **'A', 'B', 'C', 'D', 'E', 'F'**]  cantidad de elementos:16 |   · |

**Eliminar datos de una lista (las tuplas son inmutables)**

* + pop() elimina un solo dato ubicado en el index final de la lista
  + pop(I) elimina un solo dato ubicado en el index I de la lista
  + remove(D) elimina el primer valor D que encuentra desde index 0
  + clear() elimina todo el contenido de la colección. (La colección sigue existiendo)

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | lista = [1,2,3,"hola",99,3.14159,True, False]  print (f"el contenido de original {lista=}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  # elimino un dato  lista.pop()#por index () en el lugar solicitado ultimo  print (f"lista.pop():\n{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.pop(2)#por index (2) en el lugar solicitado  print (f"lista.pop(2):\n{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.remove("hola")# por contenido  print (f"lista.remove('hola'):\n{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.remove(2)# por contenido  print (f"lista.remove(2):\n{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.clear()# vació la lista  print (f"lista.clear ():\n{lista}\n\tcantidad de elementos:{len(lista)}") |   Salida esperada por consola   |  | | --- | | el contenido de original lista=[1, 2, 3, 'hola', 99, 3.14159, True, False]  cantidad de elementos:**8**  lista.pop():  [1, 2, 3, 'hola', 99, 3.14159, True]  cantidad de elementos:**7**  lista.pop(2):  [1, 2, 'hola', 99, 3.14159, True]  cantidad de elementos:**6**  lista.remove('hola'):  [1, 2, 99, 3.14159, True]  cantidad de elementos:**5**  lista.remove(2):  [1, 99, 3.14159, True]  cantidad de elementos:**4**  lista.clear ():  []  cantidad de elementos:**0** |   · |

·

**Organización de datos de una lista (las tuplas son inmutables)**

* + reverse() invierte el orden de la lista
  + sort() ordena la lista de mayor a menor
  + lista.sort(reverse=True) elimina el primer valor D que encuentra desde index 0

·

|  |
| --- |
| Tengan en cuenta que sort ordena de mayor a menor objetos numéricos y por cantidad de caracteres en objetos string.  Si se utiliza listas con objetos de ambos tipos se obtendrá un error |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | lista = [1,2,3,4,5,6,7,8,9,8,7,6,5,6,7,8,9,11,2]  print (f"el contenido de original {lista=}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.reverse()  print (f"lista.reverse:\n{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.sort()  print (f"lista sort:\n{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.sort(reverse=True)  print (f"lista sort(reverse=True):\n{lista}\n\tcantidad de elementos:{len(lista)}") |   Salida esperada por consola   |  | | --- | | el contenido de original lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 6, 7, 8, 9, 11, 2]  cantidad de elementos:19  lista.reverse:  [2, 11, 9, 8, 7, 6, 5, 6, 7, 8, 9, 8, 7, 6, 5, 4, 3, 2, 1]  cantidad de elementos:19  lista sort:  [1, 2, 2, 3, 4, 5, 5, 6, 6, 6, 7, 7, 7, 8, 8, 8, 9, 9, 11]  cantidad de elementos:19  lista sort(reverse=True):  [11, 9, 9, 8, 8, 8, 7, 7, 7, 6, 6, 6, 5, 5, 4, 3, 2, 2, 1]  cantidad de elementos:19 |   · |

·

**Métodos count e index**

|  |  |  |
| --- | --- | --- |
| Información sobre el contenido de datos de una lista o una tupla. No modifican el contenido y son usados en listas y tuplas  ◦ count() invierte el orden de la lista  ◦ index() ordena la lista de mayor a menor  Código Python   |  | | --- | | lista = [1,2,3,4,5,6,7,8,9,8,7,6,5,6,7,8,9,11,2]  print (f"el contenido de original {lista=}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  print (f"lista.count(9) esta {lista.count(9)} veces en la lista.\n\tcantidad de elementos:{len(lista)}")  print (f"lista.count(11) esta {lista.count(11)} veces en la lista.\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  print (f"lista.index(3) esta en el index :{lista.index(3)}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------ |   Salida esperada por consola   |  | | --- | | el contenido de original lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 6, 7, 8, 9, 11, 2]  cantidad de elementos:19  lista.count(9) esta 2 veces en la lista.  cantidad de elementos:19  lista.count(11) esta 1 veces en la lista.  cantidad de elementos:19  lista.index(3) esta en el index :2  cantidad de elementos:19 |   · |

·

|  |
| --- |
| Averiguar que sucede si uso un operador + o \* con una lista .  Debes crear y llenar la lista\_1 y 2  lista\_r = lista\_1 + lista\_2  lista\_r = lista\_v\*2 |

·

**Pilas y colas**

|  |
| --- |
| FIFO First In > Fist Out  LIFO Last In > First Out –o- First In > Last Out |

·

Las pilas y las colas son estructuras de datos fundamentales utilizadas en programación para organizar y manipular conjuntos de elementos de manera específica. Son especialmente útiles para gestionar el orden de acceso y procesamiento de datos. Aquí tienes descripciones de pilas y colas en Python:

**Pilas (Stacks):**

Una pila es una estructura de datos lineal que sigue el principio "último en entrar, primero en salir" (**LIFO**, por sus siglas en inglés). Esto significa que el último elemento que se agrega a la pila es el primero en ser retirado. Las pilas se utilizan en situaciones en las que necesitas mantener un registro de elementos en un orden específico y acceder a ellos en sentido inverso.

|  |
| --- |
| **Las operaciones principales en una pila son:**  **Push:** Agregar un elemento en la parte superior de la pila.  **Pop:** Retirar el elemento superior de la pila.  **Peek (o Top):** Ver el elemento superior sin eliminarlo.  Las pilas se pueden implementar en Python utilizando listas:  stack = [] # Crear una pila vacía  stack.append(1) # Push: Agregar 1 a la pila  stack.append(2) # Push: Agregar 2 a la pila  top\_element = stack.pop() # Pop: Retirar el elemento superior (2) |

**Colas (Queues):**

Una cola es una estructura de datos lineal que sigue el principio "primero en entrar, primero en salir" (**FIFO**, por sus siglas en inglés). Esto significa que el primer elemento que se agrega a la cola es el primero en ser retirado. Las colas se utilizan para administrar elementos en un orden específico en el que se procesan en el mismo orden en el que se agregan.

|  |
| --- |
| **Las operaciones principales en una cola son:**  **Enqueue (o Push):** Agregar un elemento al final de la cola.  **Dequeue (o Pop):** Retirar el elemento del frente de la cola.  **Front:** Ver el elemento al frente de la cola sin eliminarlo.  En Python, puedes implementar colas utilizando la clase deque del módulo collections:  from collections import deque  queue = deque() # Crear una cola vacía  queue.append(1) # Enqueue: Agregar 1 al final de la cola  queue.append(2) # Enqueue: Agregar 2 al final de la cola  front\_element = queue.popleft() # Dequeue: Retirar el elemento del frente (1) |

Tanto las pilas como las colas son útiles en diversas situaciones de programación y algoritmos, y comprender sus características te ayudará a elegir la estructura de datos adecuada según tus necesidades.

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | lista = [1,2,3,4,5,6,7,8,9,10,11,112,13,14,15]  print ("append()")  lista.append(16)  print(f"{lista=}")  lista.append(17)  print(f"{lista=}")  lista.append(18)  print(f"{lista=}")  lista.append(19)  print(f"{lista=}")  lista.append(20)  print(f"{lista=}")  #-----------------------------  print ("pop() fin")  lista.pop()  print(f"{lista=}")  lista.pop()  print(f"{lista=}")  lista.pop()  print(f"{lista=}")  lista.pop()  print(f"{lista=}")  lista.pop()  print(f"{lista=}")  #-----------------------------  print ("pop(0) inicio")  lista.pop(0)  print(f"{lista=}")  lista.pop(0)  print(f"{lista=}")  lista.pop(0)  print(f"{lista=}")  lista.pop(0)  print(f"{lista=}")  lista.pop(0)  print(f"{lista=}") |   Salida esperada por consola   |  | | --- | | append()  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16, 17]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16, 17, 18]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16, 17, 18, 19]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16, 17, 18, 19, 20]  pop() fin  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16, 17, 18, 19]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16, 17, 18]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16, 17]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15]  pop(0) inicio  lista=[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15]  lista=[3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15]  lista=[4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15]  lista=[5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15]  lista=[6, 7, 8, 9, 10, 11, 112, 13, 14, 15] |   · |

## **Bucles II:**

Otras Estructuras de control de flujo son los bucles. En Python existen dos tipos de bucles, en el módulo I vimos while, ahora for :

### **Bucles for:**

Un bucle for es una estructura de control de flujo en Python que se utiliza para repetir una serie de instrucciones mientras se haya datos en cualquier **objeto iterable (OI)** (colección, string, range, etc ). Se pueden aplicar modificadores de flujo como break y continue.

|  |
| --- |
| El objeto iterable debe existir antes de entrar en el bucle y no debe modificarse mientras se utiliza. |

La sintaxis del bucle for es la siguiente:

|  |
| --- |
| <flujo de programa>  …  objeto\_iterable  for cada\_dato in objeto\_iterable:  <Bloque de código>  …  <continua flujo del programa> |

**Ejemplo de bucle for simple:**

En este ejemplo el for comprueba que el valor de la un objeto iterable (OI), si no esta vaciá ej [] o “” o {}, se inicia una iteración con cada dato del OI mientras tenga no se haya llegado al último o no se haya ejecutado un break.

**Condiciones y Bucles**

La sintaxis básica es la siguiente:

|  |
| --- |
| for cada\_dato in objeto\_iterable:  # Código que se ejecuta **PARA** cada elemento en el OI  Código que se ejecuta cuando se termina de recorrer el OI |

En Python, la función for se utiliza para crear bucles o ciclos que se ejecutan mientras una determinada condición sea verdadera. El bloque de código dentro del for se repetirá continuamente hasta que la condición se evalúe como falsa.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string = "Python"  for caracter in string:  print(f"El caracter es: {caracter}")  print("Adios...") |   Salida esperada por consola   |  | | --- | | El caracter es: P  El caracter es: y  El caracter es: t  El caracter es: h  El caracter es: o  El caracter es: n  Adios... |   · |

En el ejemplo anterior, creamos un string con el contenido “Python”, utilizamos la función for para recorrer cada caracter en el string hasta que este se termina.

### **Modificación de ejecución en un bucle for.**

En ciertas circunstancias es necesario interrumpir el flujo lógico de un programa. Python cuenta con los siguientes recursos para hacerlo.

**Break:**

### Interrupciones de ejecución de un bloque for.

La palabra reservada break termina prematuramente la ejecución del bloque de código en el que se encuentra y restablece el flujo de ejecución al bloque de código que lo precede.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | numeros =(0,1,2,3,4,5,6,7,8,9)  for cada\_numero in numeros:  if cada\_numero == 5:  print("break")  break  print (f"el valor de {cada\_numero =}") |   Salida esperada por consola   |  | | --- | | el valor de cada\_numero =0  el valor de cada\_numero =1  el valor de cada\_numero =2  el valor de cada\_numero =3  el valor de cada\_numero =4  break |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string = "Python es genial"  for caracter in string:  if caracter == " ": # " " un espacios  print ( "break")  break  print(f"El caracter es: {caracter}")  print("Adios...") |   Salida esperada por consola   |  | | --- | | El caracter es: P  El caracter es: y  El caracter es: t  El caracter es: h  El caracter es: o  El caracter es: n  break  #donde encuentra el primer espacio vació del string "Python es genial"  # ^\_primer espacio encontrado  Adios... |   · |

.

**Continue**

La palabra reservada **continue** termina de forma prematura la ejecución de un bloque dentro de un ciclo y vuelve al inicio del bucle.

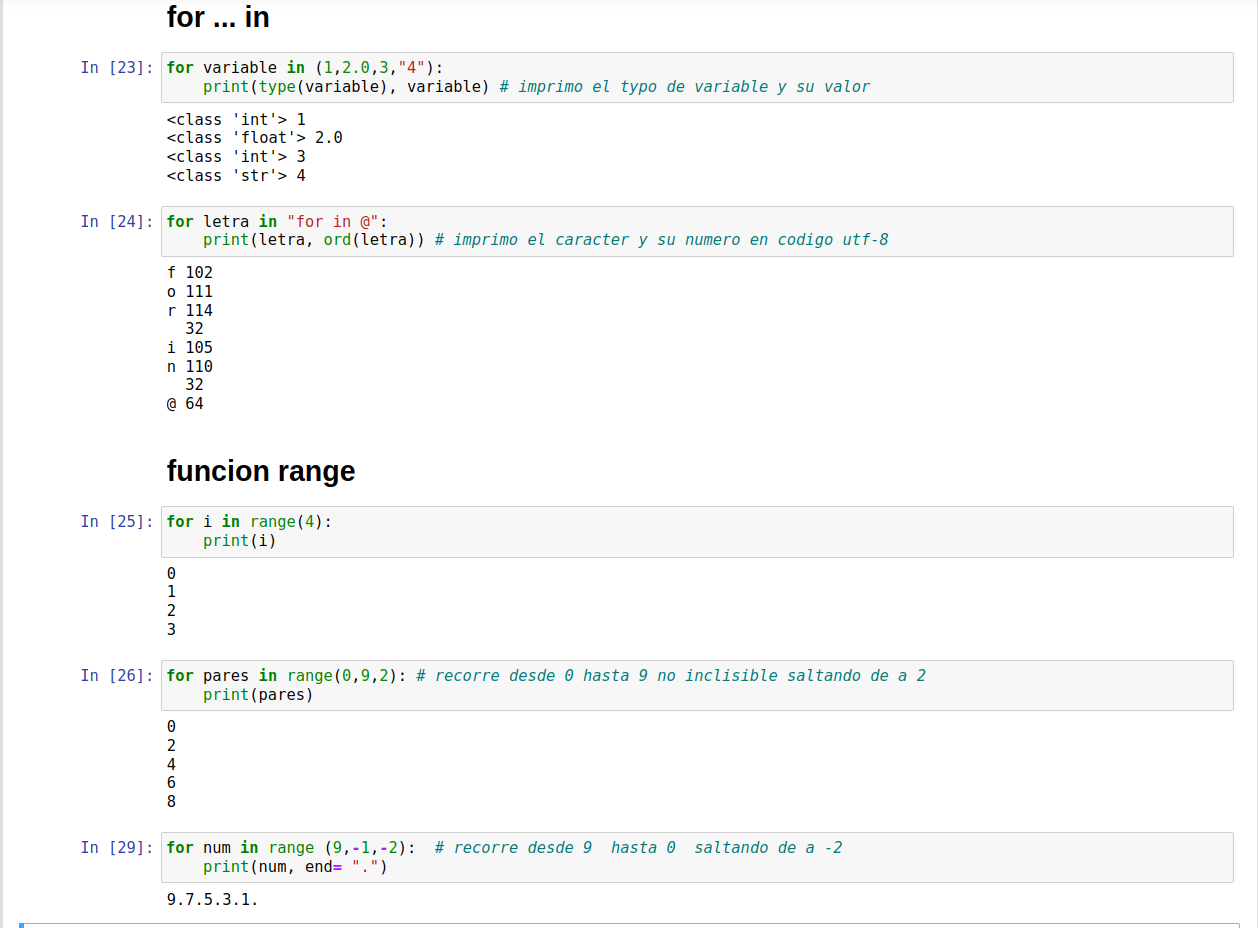
|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | numeros =(0,1,2,3,4,5,6,7,8,9)  for cada\_numero in numeros:  print (f"el valor de { cada\_numero =}")  if cada\_numero %2 == 0:  print("\t\tcontinue x par")  continue  print("\timpar") |   Salida esperada por consola   |  | | --- | | el valor de cada\_numero =0  continue x par  el valor de cada\_numero =1  impar  el valor de cada\_numero =2  continue x par  el valor de cada\_numero =3  impar  el valor de cada\_numero =4  continue x par  el valor de cada\_numero =5  impar  el valor de cada\_numero =6  continue x par  el valor de cada\_numero =7  impar  el valor de cada\_numero =8  continue x par  el valor de cada\_numero =9  impar |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | string = "Python es genial"  for caracter in string:  if caracter == " ": # " " un espacios  print ( "\*")  continue  print(f"El caracter es: {caracter}")  print("Adios...") |   Salida esperada por consola   |  | | --- | | El caracter es: P  El caracter es: y  El caracter es: t  El caracter es: h  El caracter es: o  El caracter es: n  \* donde encuentra un espacio vació imprime \*  El caracter es: e  El caracter es: s  \* donde encuentra un espacio vació imprime \*  El caracter es: g  El caracter es: e  El caracter es: n  El caracter es: i  El caracter es: a  El caracter es: l |   **·** |

**Con range:**

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | nombre\_range = range(5,25,2)  print (f"rango ={nombre\_range}")  for numero in nombre\_range:  print(f"El valor del rango es: {numero}")  print("Adios...") |   Salida esperada por consola   |  | | --- | | rango =range(5, 25, 2)  El valor del rango es: 5  El valor del rango es: 7  El valor del rango es: 9  El valor del rango es: 11  El valor del rango es: 13  El valor del rango es: 15  El valor del rango es: 17  El valor del rango es: 19  El valor del rango es: 21  El valor del rango es: 23  Adios... |   **·** |

****

**zip():**

Es una función integrada que permite combinar dos o más objetos iterables en una sola secuencia de tuplas, donde cada tupla contiene un elemento de cada objetos iterables

Si los objetos iterables tienen cantidades de objetos diferentes, la longitud del zip será siempre la más corta.

La función zip() es útil cuando necesitas combinar múltiples secuencias de manera sincronizada para procesar o analizar datos en pares o en paralelo.

La sintaxis básica de zip() es la siguiente:

|  |
| --- |
| zip(iterable1, iterable2, …) |

iterable1, iterable2, ...: Las secuencias que se desean combinar. Pueden ser listas, tuplas, cadenas u otros iterables.

A continuación, un ejemplo para entender cómo funciona zip()

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | nombres = ["Juan", "María", "Pepe"]  notas = [9, 10, 8]  salida = tuple( zip(nombres,notas)) |   Salida esperada por consola   |  | | --- | | salida (('Juan', 9), ('María', 10), ('Pepe', 8)) |   · |

.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | nombres = ["Juan", "María", "Pepe"]  notas = [9, 10, 8]  for cada\_nombre, cada\_nota in zip(nombres,notas):  print (f"el alumno {cada\_nombre} tiene {cada\_nota} puntos") |   Salida esperada por consola   |  | | --- | | el alumno Juan tiene 9 puntos  el alumno María tiene 10 puntos  el alumno Pepe tiene 8 puntos |   · |

.

**enumerate():**

Es una función incorporada que se utiliza para iterar sobre una objetos iterables al mismo tiempo que lleva un seguimiento del índice o posición actual de cada elemento.

Esto es especialmente útil cuando necesitas acceder tanto al valor del elemento como a su índice dentro del bucle.

|  |
| --- |
| **Importante:** **enumerate no da el index**, enumera los objetos iterables. No siempre se coordina el index con la enumeración. **Pero si en una gran cantidad de oportunidades.** |

La sintaxis básica de enumerate() es la siguiente:

|  |
| --- |
| enumerate(iterable, start=0)  start: El valor inicial del contador. Por defecto, es 0. |

La función enumerate() es especialmente útil cuando necesitas realizar operaciones que involucren tanto el valor de un elemento como su posición en una secuencia.

.

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | nombres = ["Juan", "María", "Pepe"]  for enum ,cada\_nombre in enumerate(nombres):  print (f"en la posición {enum} se encuentra el alumno {cada\_nombre} ") |   Salida esperada por consola   |  | | --- | | en la posición 0 se encuentra el alumno Juan  en la posición 1 se encuentra el alumno María  en la posición 2 se encuentra el alumno Pepe |   · |

**Ejemplos donde enumerate() no coinciden con el index:**

|  |  |
| --- | --- |
| Código Python  Salida esperada por consola   |  | | --- | | 0 cada\_dato='Ana'  1 cada\_dato='Lucas'  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  0 cada\_dato='Ariel'  1 cada\_dato='Luis'  2 cada\_dato='Lucas'  3 cada\_dato='Ana'  4 cada\_dato='María'  5 cada\_dato='Juan'  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  0 cada\_dato='Ariel'  1 cada\_dato='Luis'  2 cada\_dato='Lucas'  3 cada\_dato='Ana'  4 cada\_dato='María'  5 cada\_dato='Juan'  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  0 cada\_dato='Ana'  1 cada\_dato='Ariel'  2 cada\_dato='Juan'  3 cada\_dato='Lucas'  4 cada\_dato='Luis'  5 cada\_dato='María'  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |   · |

.

.**enumerate() y zip():**

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | nombres = ["Juan", "María", "Pepe"]  notas = [9, 10, 8]  for enum,[cada\_nombre, cada\_nota] in enumerate(zip(nombres,notas)):  print (f" en la posición {enum} esta el alumno {cada\_nombre} tiene {cada\_nota} ") |   Salida esperada por consola   |  | | --- | | en la posición 0 esta el alumno Juan tiene 9  en la posición 1 esta el alumno María tiene 10  en la posición 2 esta el alumno Pepe tiene 8 |   · |

.

**Anidación:**

Un bucle dentro de un bucle.

Si el bucle interior itera una **N** cantidad de veces, el bucle exterior una **M** cantidad.   
La cantidad de iteraciones totales es **N** x **M.**

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | pseudo\_matriz=[  ["0 - 0","0 - 1","0 - 2","0 - 3","0 - 4","0 - 5","0 - 6","0 - 7","0 - 8","0 - 9"],  ["1 - 0","1 - 1","1 - 2","1 - 3","1 - 4","1 - 5","1 - 6","1 - 7","1 - 8","1 - 9"],  ["2 - 0","2 - 1","2 - 2","2 - 3","2 - 4","2 - 5","2 - 6","2 - 7","2 - 8","2 - 9"],  ["3 - 0","3 - 1","3 - 2","3 - 3","3 - 4","3 - 5","3 - 6","3 - 7","3 - 8","3 - 9"],  ["4 - 0","4 - 1","4 - 2","4 - 3","4 - 4","4 - 5","4 - 6","4 - 7","4 - 8","4 - 9"],  ["5 - 0","5 - 1","5 - 2","5 - 3","5 - 4","5 - 5","5 - 6","5 - 7","5 - 8","5 - 9"]  ]  for vector in pseudo\_matriz:  for dato in vector:  print (f"{dato}|",end="")  print() |   Salida esperada por consola   |  | | --- | | 0 - 0|0 - 1|0 - 2|0 - 3|0 - 4|0 - 5|0 - 6|0 - 7|0 - 8|0 - 9|  1 - 0|1 - 1|1 - 2|1 - 3|1 - 4|1 - 5|1 - 6|1 - 7|1 - 8|1 - 9|  2 - 0|2 - 1|2 - 2|2 - 3|2 - 4|2 - 5|2 - 6|2 - 7|2 - 8|2 - 9|  3 - 0|3 - 1|3 - 2|3 - 3|3 - 4|3 - 5|3 - 6|3 - 7|3 - 8|3 - 9|  4 - 0|4 - 1|4 - 2|4 - 3|4 - 4|4 - 5|4 - 6|4 - 7|4 - 8|4 - 9|  5 - 0|5 - 1|5 - 2|5 - 3|5 - 4|5 - 5|5 - 6|5 - 7|5 - 8|5 - 9| |   · |

·

|  |
| --- |
| Python también proporciona módulos adicionales en su biblioteca estándar que ofrecen colecciones más especializadas y eficientes.  Ejemplos:  **collections:** Este módulo ofrece tipos de colecciones adicionales como deque (cola doblemente terminada), Counter (contador), OrderedDict (diccionario ordenado) y namedtuple (tupla con nombre), entre otros.  **array:** Este módulo proporciona un tipo de colección llamado array, que es una estructura de datos eficiente para almacenar elementos del mismo tipo de datos en un arreglo.  **heapq:** Este módulo implementa estructuras de datos de montículos (heaps) que se utilizan para mantener el orden en una colección de manera eficiente.  **queue:** Este módulo proporciona implementaciones de diferentes tipos de colas, como Queue, LifoQueue (pila) y PriorityQueue (cola de prioridad).  Y debemos tener muy en cuenta los arrays de la librería **numpy (Num**erical **Py**thon**)** y los DataFrames de **pandas (Pan**el **Da**ta**s)** |

.