1. DATOS "SIMPLES" CADENAS/STRING.

¿*CADENAS*?

Los datos de cadenas, son *secuencias* "**inmutables**" que contienen **caracteres** encerrados entre comillas (simples, dobles, triples comillas).

- El delimitador de inicio y final debe ser el mismo carácter.
+ Pueden contener:
  - De 0 (cadena vacía) , un carácter, una palabra, una línea o múltiples líneas.
    Una cadena en Python puede contener tantos caracteres como desees.
    El único límite son los recursos de memoria de su máquina.
  ¿*Caracteres*? Según codificación **ASCII**, **Unicode**, **bytes**.

¿*CODIFICACIÓN ASCII, UNICODE*?

Codificación **ASCII**: Están representados todos los caracteres de idiomas como el inglés.

Codificación **CP-1252**, también conocida como windows-1252 (utilizada en sistemas Windows), Para otros idiomas, como el español.

En CP-1252, los bytes 0 a 127 son iguales que en ASCII (se usan para las letras del alfabeto, los números, etc.). Los bytes 128 a 255 se utilizan para representar otros caracteres, como la ñ.
Tanto ASCII como CP-1252 utilizan un **solo byte** para representar cada carácter.

Codificación **UNICODE**: Estándar de codificación de caracteres. Permite representar los idiomas que necesitan **más** de un **byte** para representar todos los caracteres disponibles.

Las versiones de Unicode, por aparición son

- UTF-16: Codificación Unicode que usa **2 bytes** para representar todos los caracteres. Para poder representar idiomas que requieren de más de 65535 caracteres, utiliza algunos hacks un poco sucios.
- UTF-32: Codificación Unicode que usa **4 bytes** para representar todos los caracteres existentes. Cualquier idioma conocido es representable.
- **UTF-8**: Codificación Unicode de **longitud variable**. Usa la misma representación que ASCII para los 128 primeros caracteres, dos bytes para alfabeto latino extendido, 3 bytes para caracteres chinos, y 4 para el resto (no representables con menos de 4 bytes)

¿*MÚLTIPLES LÍNEAS DE SALIDA*?:

Para cadenas de múltiples líneas se utilizan las triples comillas (dobles, o simples).

Aunque, los "fin de línea" son incluidos automáticamente, para asegurar que se inserte un salto de línea después de la triples comilla, agregar una **** al final de la línea.

¿*SECUENCIAS DE ESCAPE*?:

Las secuencias de escape permiten dar otra interpretación a ciertos caracteres, para ello se utiliza la barra invertida (\ ) antes del carácter que debe ser "reinterpretado".

Las secuencias de escape más utilizadas son:

Escapar caracteres especiales:

`\'`	La comilla simple se considera como otro carácter más, NO como delimitador.
`\"`	La comilla doble se considera como otro carácter más, NO como delimitador.
`\\`	La barra invertida se considera como otro carácter más, NO para "escapar" caracteres

Para insertar "teclas de edición":

`\b`	Inserta el carácter **de retroceso** (BS) ASCII.
`\t`	Inserta el carácter **de tabulación horizontal** (TAB) ASCII.
`\v`	Inserta el carácter **de tabulación vertical** (VT) ASCII .
`\r`	Inserta el carácter de **retorno de carro** (CR) ASCII .
`\n`	Inserta el carácter de **fin de línea** (LF) ASCII .

`\enter`	Eliminar salto de línea en cadenas de líneas múltiples.
`\a`	Inserta el carácter de campana **Bell** (BEL) ASCII .
`\f`	Inserta el carácter de **Formfeed** (FF) ASCII .

Para insertar "tipos de objetos":

`\u``nnnn`	Inserta carácter Unicode con valor hexadecimal de 16 bits xxxx
`\U``nnnnnnnn`	Inserta carácter Unicode con valor hexadecimal de 32 bits xxxx
`\o``nnnn`	Objeto de tipo octal
`\x``nnnn`	Objeto de tipo hexadecimal

Otras:

`\N{<name>}`	Carácter de la base de datos Unicode con <name> dado

¿*PREFIJOS EN LAS CADENAS*?:

Para anular las secuencias de escape

`R|r`	`(Cadenas crudas/sin formato)` Los caracteres escapados, con (\), no son "reinterpretados". Esto es especialmente útil, por ejemplo, para usar las expresiones regulares.

Para crear literales de cadena:

`U|u`	Para crear literales de tipo Unicode (P2).
`B|b`	Para crear literales de tipo bytes.

¿*TIPOS DE DATOS DE CADENAS*?

Es una de las mayores diferencias entre Python 2 y 3.

En P3, encontramos: **Unicode, byte** y **bytearray** .
En P2, encontramos: Unicode y str(byte).

1.1. UNICODE. <CLASS 'STR'>

¿*ALGUNAS CARACTERÍSTICAS*?

- Las cadenas de tipo **`Unicode`** <**class 'str'**>.
  + Son secuencias de **caracteres, inmutables**.
  + Su contenido está en forma **legible por el ser humano**.
  + Necesitan ser **codificadas** antes de que se pueden almacenar en el disco.
  + Pueden contener:
    - De 0 (cadena vacía) , un caracteres, una palabra, una línea o múltiples líneas.
    - Caracteres en codificación Unicode (caracteres de múltiples lenguajes)
    - Secuencias de escape.

1.1.1. CREACIÓN.

Encontramos:

- Formato **literal**, entre comillas (simples o dobles) o comillas triples si queremos que sea multilíneas.
  + Originalmente en P2, pueden seguir usándose los prefijos: `u`/`U`, para codificación Unicode.
- Función constructora: str().

`str(`*`x`***`)`** ---------- La función constructora str(). Devuelve un tipo **cadena** a partir de una amplia variedad de tipos de datos, incluidas cadenas, literales enteros y literales flotantes.

`[var_str =] srt(objeto [ ,` `encoding`**=**codificación **`,`** `errors`**=**errors **`])`**

+ **`objeto`** Cualquier objeto. Especifica el objeto a convertir en una cadena.
+ `encoding` La codificación del objeto. El valor predeterminado es UTF-8

+ `errors` Especifica qué hacer si falla la decodificación.

1.2. BYTES <CLASS 'BYTES'>.

¿*ALGUNAS CARACTERÍSTICAS*?

- Las cadenas de tipo **`Byte`** <**class 'bytes'**>.
  + Son secuencias de caracteres-bytes **inmutables**.
  + Su contenido (internamente almacenado) está en forma solo **legible por la máquina**.
  + Como los objetos Byte son legibles por máquina, pueden **almacenarse directamente en el disco**.
  + Pueden contener:
    - **Solo** permite caracteres en codificación ASCII.
    - y algunas secuencias de escape.

1.2.1. CREACIÓN.

Encontramos:

- Formato **literal**, entre comillas.
  + Entre comillas (simples o dobles) o comillas triples si queremos que sea multilíneas, pero ahora anteponiendo el prefijo: **`b`** o **`B`**.
  + Los bytes con un valor numérico de 128 o mayor deben expresarse con escapes.
  + Los literales de bytes también pueden usar el prefijo **`r`** para deshabilitar el procesamiento de secuencias de escape.
- Función constructora: bytes().

`bytes`()----------La función bytes() . Devuelve un objeto bytes. <**class 'bytes**'>.

+ Puede convertir objetos en objetos de bytes, o crear objetos de bytes vacíos del tamaño especificado.

```
[var_byte =] bytes (fuente [ ,encoding ,errors ] )
```

+**`fuente`** Especifica el elemento que se utiliza para crear un objeto de bytes.
. Si es un entero, se creará un objeto de bytes vacío del tamaño especificado.
. Si se trata de una cadena, asegúrese de especificar la `codificación` de la fuente.
. La función range().
+ `encoding` La codificación del objeto. El valor predeterminado es UTF-8.
+ `errors` Especifica qué hacer si falla la decodificación.

1.3. BYTEARRAY →<CLASS 'BYTEARRAY'>.

¿*ALGUNAS CARACTERÍSTICAS*?

- Las cadenas de tipo **`Byterray`** <**class 'bytearray'**>.
  + Son secuencias de caracteres-bytes **MUTABLES**.
    La diferencia entre **bytes ()** y **bytearra**y () es que bytes () devuelve un objeto que no se puede modificar, y bytearray () devuelve un objeto que se puede modificar.

1.3.1. CREACIÓN.

- Formato **literal**: No hay una sintaxis literal para los objetos bytearray, en su lugar, siempre se crean llamando a la función constructora.
- Función constructora: bytearray().

`bytearray`(x)----------La función bytearray(). Devuelve un objeto una matriz de bytes. <**class 'bytearray**'> .

```
[var_byteArray =]bytearray (fuente [ ,encoding [ , errors ] ] )
```

+**`fuente`** Especifica el elemento que se utiliza para crear un objeto de bytes.
. Si es un entero, se creará un objeto de bytes vacío del tamaño especificado.
. Si se trata de una cadena, asegúrese de especificar la `codificación` de la fuente.
. Un objeto Copiando los datos binarios existentes a través del protocolo de búfer: bytes(obj)
. La función range().
+ `encoding` La codificación del objeto. El valor predeterminado es UTF-8
+ `errors` Especifica qué hacer si falla la decodificación. Por defecto ¿?.

2. MANIPULACIÓN OPERACIONES.

Las cadenas <str> y <bytes > de Python **no pueden ser modificadas** son **inmutable**.
Por eso, asignar a una posición indexada de la cadena resulta en un error:
Si necesitas una cadena diferente, deberías asignar una nueva cadena o crear otra variable.
Las cadenas <bytearray> si son **modificables**.

- **"Aritméticas" concatenación (+), copiar (*)**:
  -Las cadenas de texto pueden ser concatenadas (pegadas juntas) con el operador + y repetidas con *:

- **De "Secuencia"**:
  - Una cadena-Unicode es una **secuencia de caracteres**.
  - La función **`len`**() devuelve la longitud de una cadena de texto (número de caracteres).
    La longitud, de una cadena, dependerá del número de Bytes usados para representarla.
  - Las cadenas de texto se pueden **indexar** (subíndices), el primer carácter de la cadena tiene el índice 0.
  - Además de la función len() y la indexación (subíndices) y la "rebanada" utilizando corchetes.
    Los índices son usados para obtener caracteres individuales, las rebanadas para obtener sub-cadenas.

  Recorridos con for in
  Tanto para los objetos tipo str como para los objetos tipo bytes, cada carácter se vuelve un elemento iterable.

- **Algunas**, funciones utilizadas, además de len(), son: la función max(), la función min(), la función sum() (generalmente: **sólo para tipo bytes**) y la función memoryview() **SOLO para tipo byte y bytearray.**

- **Otras**, según sus métodos.

3. PASAR DE UN FORMATO A OTRO.

Se utilizan funciones incorporadas y métodos de cadena:

- **Para convertir: <str> a <byte>:**
  + Anteponer **b** a la cadena Unicode, SIEMPRE que sea un carácter del ASCII estándar
  + Si es un carácter del ASCII extendido, es necesario utilizar la función **bytes()** con la codificación correcta/deseada.
  + Con el método de **str.encode()** con la codificación correcta.

- **Para convertir: Byte a Unicode:**
  + Utilizando el método byte.**decode()** , para decodificar es imprescindible especificar la **codificación adecuada** con la que se obtendrá el resultado correcto.
  - Si intentamos decodificar con una codificación inadecuada podemos llevarnos una sorpresa, en **forma de error**, o bien, obtener una conversión no deseada:
  - En caso de **error** ,se puede:
    + Ignorar ('ignore' | errors='ignore') El carácter que produce el error se ignora /No se visualiza.
    +Reemplazar (**"replace"** | errors='replace') forzar un cambio de caracteres aunque no sea coherente

- **De codificación Unicode a carácter que representa y viceversa:**

chr() ---------- La función chr() . Devuelve un carácter del código Unicode especificado.

```
[variable =] chr(valor)
```

+ **valor** Un entero que representa un punto de código Unicode válido.

ord () ---------- La función ord(). Convierte un entero que representa el código Unicode del carácter especificado.

```
[variable =] ord(valor)
```

+ **valor** Un entero que representa un código Unicode válido.

4. MÉTODOS DE CADENA.

☑ Listar todas las propiedades/métodos de los objetos: **str**, **bytes** y **bytearrays**

Nota: La mayoría de los métodos son comunes a str y bytes y para bytearray
Entre los métodos más utilizados están:

4.1. PASAR DE UN FORMATO A OTRO.

- **De str a bytes:**

encode(*x*) ---------- El método encode() //`str.encode()`. Devuelve una versión codificada de la cadena como un objeto de bytes.

```
[var_bytes =] <class 'str'>.encode(encoding='utf-8',errors='strict')
```

+ `encoding` Una cadena que especifica la codificación a utilizar. El valor predeterminado es UTF-8.
+ `errors` Especifica qué hacer si falla la decodificación. Valores posibles:
  `'backslashreplace'` utiliza la barra invertida en lugar del carácter que no se pudo codificar.
  `'ignore'`. ignora los caracteres que no pueden ser codificados.
  `'namereplace'` `reemplaza` el carácter por un texto que explica el carácter.
  `'strict'`. Por **defecto**, genera un error en caso de fallo
  `'replace'`. reemplaza al personaje con un signo de interrogación.
  `'xmlcharrefreplace'` reemplaza el carácter con un carácter xml.

- **De bytes a str:**

decode(*x*) ---------- El método / `bytes.decode()`.Devuelve una cadena decodificada de los bytes dados..

```
[var_bytes =] <class 'bytes'>.decode(encoding='utf-8',errors='strict')
```

+ **encodig** La codificación predeterminada es `'utf-8'`.
+ **errors** Esquema de manejo de errores. El valor predeterminado es `'strict'`, .

- **De bytes a str en hexadecimal** (de bytes y Bytearray):

hex() ---------- El método hex() .Devuelve un objeto tipo *str* que contiene la representación en hexadecimal de cada elemento del objeto tipo *bytes*.

- **De str en hexadecimal a bytes:**

`fromhex()` ---------- El método fromhex() . un objeto tipo *bytes* a partir de un objeto *str* que contiene la representación en hexadecimal de uno o más caracteres ASCII.

4.2. PARA "PRESENTACIÓN":

Tipo de letra:

`capitalize()` ---------- El método capitalize(). Devuelve la cadena con la primera letra en mayúsculas.

`lower()` ---------- El método lower(). Devuelve la cadena con las letras en minúsculas.

`casefold()` ---------- El método casefold(). Convierte la cadena en minúsculas,

`upper()` ---------- El método upper(). Devuelve la cadena con las letras en mayúsculas.

`swapcase()` ---------- El método swapcase(). Devuelve la cadena con las letras convertidas las mayúsculas en minúsculas y viceversa.

`title()` ---------- El método title(). Devuelve la cadena con Formato Título.

Alineación Horizontal:

`center(x)` ---------- El método center(). Devuelve la cadena centrada.

`ljust(x)` ---------- El método ljust(). Devuelve la cadena alineada a la izquierda.

`rjust(x)` ---------- El método ….. Devuelve la cadena alineada a la derecha.

Relleno:

`zfill(x)` ---------- El método zfill(). Devuelve la cadena rellena con ceros a la izquierda hasta alcanzar la longitud final indicada.

4.3. PARA BÚSQUEDA:

`count(x)` ---------- El método count(). Devuelve un entero representando la cantidad de apariciones de subcadena dentro de cadena.

`find(x)` ---------- El método find(). Devuelve un entero representando la posición donde inicia la subcadena dentro de cadena. Si no la encuentra, retorna -1.

`rfind(x)` ---------- El método rfind(). Busca en la cadena un valor específico y devuelve la última posición donde se encontró.

`index(x)` ---------- El método index(). Busca en la cadena un valor específico y devuelve la posición donde se encontró.

`rindex(x)` ---------- El método rindex(). Busca en la cadena un valor específico y devuelve la última posición donde se encontró.

4.4. PARA "CONFIRMACIONES" VALIDACIÓN:

Devuelven valores boolean.

`startswith(x)` ---------- El método startswith(). Devuelve un valor booleano confirmando si una cadena comienza con una subcadena determinada.

`endswith(x)` ---------- El método endswith(). Devuelve un valor booleano confirmando si una cadena finaliza con una subcadena determinada.

`isalnum()` ---------- El método isalnum(). Devuelve un valor booleano confirmando si una cadena es alfanumérica.

`isalpha()` ---------- El método isalpha(). Devuelve un valor booleano confirmando si una cadena es alfabética.

`isdigit()` ---------- El método isdigit(). Devuelve un valor booleano confirmando si una cadena es numérica.

`isdecimal`() ---------- El método isdecimal(). Devuelve True si todos los caracteres de la cadena son decimales
El método isdecimal () devuelve True si todos los caracteres son decimales (0-9).
Este método se utiliza en objetos Unicode.

`isnumeric` () ---------- El método isnumeric(). Devuelve True si todos los caracteres de la cadena son numéricos

`isupper()` ---------- El método isupper(). Devuelve un valor booleano confirmando si una cadena contiene solo mayúsculas.

`islower()` ---------- El método islower(). Devuelve un valor booleano confirmando si una cadena contiene solo minúsculas.

`isspace()` ---------- El método isspace(). Devuelve un valor booleano confirmando si una cadena contiene solo espacios en blanco.

`istitle ()` ---------- El método istitle(). Devuelve un valor booleano confirmando si una cadena tiene Formato De Título.

`isidentifier()` ---------- El método isidentifier(). Devuelve True si la cadena es un identificador.

`isprintable()` ---------- El método isprintable(). Devuelve True si todos los caracteres de la cadena son imprimibles.

4.5. PARA UNIR O DIVIDIR CADENAS.

`join(x)` ---------- El método join(). Devuelve la cadena uniéndola a un *iterable* (la cadena es separada por cada uno de los elementos del iterable).

`partition(x)` ---------- El método partition(). Devuelve una tupla con la cadena partida en varias partes, utilizando un separador.

+ una tupla de tres elementos donde el primero es el contenido de la cadena previo al separador, el segundo, el separador mismo y el ter

`rpartition(x)` ---------- El método rpartition().Devuelve una tupla en la que la cadena se divide en tres partes

`split(x)` ---------- El método split(). Devuelve una lista con la cadena partida en varias partes, utilizando un separador.

`rsplit(x)` ---------- El método rsplit(). Divide la cadena en el separador especificado y devuelve una lista.

`splitlines()` ---------- El método splitlines(). Devuelve una lista donde cada elemento es una fracción de la cadena divida en líneas.

4.6. PARA "REEMPLAZAR" LA CADENA:

`format(x)` ---------- El método *format()* . Devuelve una cadena formateada sustituyendo texto dinámicamente.

`replace(x)` ---------- El método replace(). Devuelve una cadena con texto reemplazado.

`strip(x)` ---------- El método rstrip(). Devuelve la cadena eliminando caracteres a la izquierda y derecha de una cadena. Por omisión espacios en blanco.

`lstrip(x)` ---------- El método lstrip() . Devuelve la cadena eliminando caracteres a la izquierda de una cadena. Por omisión espacios en blanco.

`rstrip(x)` ---------- El método **rstrip()** . Devuelve la cadena eliminando caracteres a la derecha de una cadena. Por omisión espacios en blanco.