

3.8. Tipo cadenas de caracteres

Las cadenas de caracteres, son secuencias inmutables que contienen caracteres encerrado entre comillas.

3.8.1. Cadenas cortas

Son caracteres encerrado entre comillas simples (`' '`) o dobles (`" "`).

```
>>> "Hola Mundo"
'Hola Mundo'
```

3.8.2. Cadenas largas

Son caracteres encerrados entre grupo comillas triples simples (`' '' '`) o dobles (`" "" "`), están son generalmente son referenciadas como *cadenas de triple comillas*.

```
>>> """Clase que representa una Persona"""
'Clase que representa una Persona'
>>> """Clase que representa un Supervisor"""
'Clase que representa un Supervisor'
```

3.8.3. Clases

A continuación, una lista de clases integradas Python para los tipos de cadenas de caracteres:

3.8.3.1. str

Son *secuencias inmutables* de cadenas de caracteres con soporte a caracteres `ASCII` .

```
>>> "Hola Mundo"
'Hola Mundo'
>>> "Hola Mundo"
'Hola Mundo'
```

3.8.4. Prefijo de cadenas

Una cadena puede estar precedida por el carácter:

 v: 3.7 ▼

- `r / R`, el cual indica, que se trata de una cadena `raw` (del inglés, cruda). Las cadenas `raw` se distinguen de las normales en que los caracteres escapados mediante la barra invertida (`\`) no se sustituyen por sus contrapartidas. Esto es especialmente útil, por ejemplo, para usar las expresiones regulares.

```
>>> raw = r"\t\nHola Plone\n"
>>> type(raw)
<type 'str'>
```


- Python 3.11 soporta cadena que utiliza codificación **Unicode**.

```
>>> saber_mas = "Atüjaa oo'omüin..."
>>> type(saber_mas)
<type 'str'>
>>> vocales = "äóè"
>>> type(vocales)
<type 'str'>
```

3.8.5. Cadenas de escape

Para escapar caracteres dentro de cadenas de caracteres se usa el carácter `\` seguido de cualquier carácter ASCII.

Secuencia Escape	Significado
<code>\newline</code>	Ignorado
<code>\\</code>	Backslash (<code>\</code>)
<code>\'</code>	Comillas simple (<code>'</code>)
<code>\"</code>	Comillas doble (<code>"</code>)
<code>\a</code>	Bell ASCII (BEL)
<code>\b</code>	Backspace ASCII (BS)
<code>\f</code>	Formfeed ASCII (FF)

Secuencia Escape	Significado
<code>\n</code>	Linefeed ASCII (LF)  v: 3.7 ▼
<code>\N{name}</code>	Carácter llamado <i>name</i> en base de datos Unicode (Solo Unicode)
<code>\r</code>	Carriage Return ASCII (CR)
<code>\t</code>	Tabulación Horizontal ASCII (TAB)
<code>\uxxxx</code>	Carácter con valor hex 16-bit <i>xxxx</i> (Solamente Unicode). Ver hex .
<code>\Uxxxxxxxx</code>	Carácter con valor hex 32-bit <i>xxxxxxxx</i> (Solamente Unicode). Ver hex .
<code>\v</code>	Tabulación Vertical ASCII (VT)
<code>\ooo</code>	Carácter con valor octal <i>ooo</i> . Ver octal .
<code>\xhh</code>	Carácter con valor hex <i>hh</i> . Ver hex .

También es posible encerrar una cadena entre triples comillas (simples o dobles). De esta forma puede escribir el texto en varias líneas, y al imprimir la cadena, se respetarán los saltos de línea que se introdujeron sin tener que recurrir a los caracteres escapados y las comillas como los anteriores.

3.8.6. Operaciones

Las cadenas también admiten operadores aritméticos como los siguientes:

- El operador [suma](#) para realizar concatenación de cadenas de caracteres:

```
>>> a, b = "uno", "dos"
>>> a + b
'unodos'
```

- El operador [multiplicación](#) para repetir la cadena de caracteres por N veces definidas en la multiplicación:

```
>>> c = "tres"
>>> c * 3
'trestrestres'
```

- El operador `modulo` usado la técnica de interpolación variables dentro de una cadena de caracteres. Más información consulte la sección [formateo %](#).

 v: 3.7 ▼

3.8.7. Comentarios

Son cadenas de caracteres las cuales constituyen una ayuda esencial tanto para quien está desarrollando el programa, como para otras personas que lean el código.

Los comentarios en el código tienen una vital importancia en el desarrollo de todo programa, algunas de las funciones más importantes que pueden cumplir los comentarios en un programa, son:

- Brindar información general sobre el programa.
- Explicar qué hace cada una de sus partes.
- Aclarar y/o fundamentar el funcionamiento de un bloque específico de código, que no sea evidente de su propia lectura.
- Indicar cosas pendientes para agregar o mejorar.

El signo para indicar el comienzo de un comentario en Python es el carácter numeral `#`, a partir del cual y hasta el fin de la línea, todo se considera un comentario y es ignorado por el intérprete Python.

```
>>> # comentarios en línea
>>>
```

El carácter `#` puede estar al comienzo de línea (en cuyo caso toda la línea será ignorada), o después de finalizar una instrucción válida de código.

```
>>> # Programa que calcula la sucesión
... # de números Fibonacci
>>> # se definen las variables
... a, b = 0, 1
>>> while b < 100: # mientras b sea menor a 100 itere
...     print(b),
...     a, b = b, a + b # se calcula la sucesión Fibonacci
...
1 1 2 3 5 8 13 21 34 55 89
```

3.8.7.1. Comentarios multilínea

Python no dispone de un método para delimitar bloques de comentarios de varias líneas.

Al igual que los comentarios de un sola línea, son cadenas de caracteres, en este caso van entre triples comillas (simples o dobles), esto tiene el inconveniente que, aunque no genera

código ejecutable, el bloque delimitado no es ignorado por el intérprete Python, que crea el correspondiente objeto de tipo [cadena de caracteres](#).

```
>>> """comentarios en varias lineas"""  
'comentarios en varias lineas'  
>>> """comentarios en varias lineas"""  
'comentarios en varias lineas'
```

 v: 3.7 ▼

A continuación, una comparación entre comentarios multilínea y comentarios en solo una línea:

```
>>> # Calcula la sucesión  
... # de números Fibonacci  
>>> """Calcula la sucesión  
... de números Fibonacci"""  
'Calcula la sucesión \nde números Fibonacci'
```

Entonces existen al menos dos (02) alternativas para introducir comentarios multilínea son:

- Comentar cada una de las líneas con el carácter #: en general todos los editores de programación y entornos de desarrollo (IDEs) disponen de mecanismos que permiten comentar y descomentar fácilmente un conjunto de líneas.
- Utilizar triple comillas (simples o dobles) para generar una cadena multilínea: si bien este método es aceptado.



A continuación, un ejemplo de Comentarios multilínea y de solo una línea:

```
>>> """Calcula la sucesión de números Fibonacci"""  
'Calcula la sucesión de números Fibonacci'  
>>> # se definen las variables  
... a, b = 0, 1  
>>> while b < 100:  
...     print(b),  
...     # se calcula la sucesión Fibonacci  
...     a, b = b, a + b  
...  
1 1 2 3 5 8 13 21 34 55 89
```

Los comentarios multilínea usado con mucha frecuencia como en las varias sintaxis Python como [comentarios de documentación](#) a continuación se listan las sintaxis más comunes:

- [Módulos](#).
- [Funciones](#).
- [Clases](#).
- [Métodos](#).

3.8.8. Docstrings

En Python todos los objetos cuentan con una variable especial llamada `__doc__`  v: 3.7  cual puede describir para qué sirven los objetos y cómo se usan. Estas variables nombre de `docstrings`, o [cadenas de documentación](#).

Ten en cuenta, una buena documentación siempre dará respuesta a las dos preguntas:

- ¿Para qué sirve?
- ¿Cómo se utiliza?

3.8.8.1. Funciones

Python implementa un sistema muy sencillo para establecer el valor de las `docstrings` en las funciones, únicamente tiene que crear un comentario en la primera línea después de la declaración.

```
>>> def hola(arg):  
...     """El docstring de la función"""  
...     print("¡Hola", arg, "!")  
...  
>>> hola("Plone")  
¡Hola Plone !
```

Puede consultar la documentación de la función `hola()` debe utilizar la función integrada `help()` y pasarle el argumento del objeto de función `hola()`:

```
>>> help(hola)  
  
Help on function hola in module __main__:  
  
hola(arg)  
    El docstring de la función  
  
>>>  
>>> print(hola.__doc__)  
El docstring de la función
```

3.8.8.2. Clases y métodos

De la misma forma puede establecer la documentación de la clase después de la definición, y de los métodos, como si fueran funciones:

```
>>> class Clase:  
...     """El docstring de la clase"""  
...     def __init__(self):  
...         """El docstring del método constructor de clase"""
```

```

...     def metodo(self):
...         """El docstring del método de clase"""
...
>>> o = Clase()
>>> help(o)

Help on instance of Clase in module __main__:

class Clase
| El docstring de la clase
|
| Methods defined here:
|
| __init__(self)
|     El docstring del método constructor de clase
|
| metodo(self)
|     El docstring del método de clase
|
>>> o.__doc__
'El docstring de la clase'
>>> o.__init__.__doc__
'El docstring del método constructor de clase'
>>> o.metodo.__doc__
'El docstring del método de clase'

```

 v: 3.7 ▼

3.8.8.3. Scripts y módulos

Cuando tiene un script o módulo, la primera línea del mismo hará referencia al `docstrings` del módulo, en él debe explicar el funcionamiento del mismo:

En el archivo `mi_modulo.py` debe contener el siguiente código:

```

"""El docstring del módulo"""

def despedir():
    """ El docstring de la función despedir """
    print("Adiós! desde función despedir() del módulo prueba")

def saludar():
    """ El docstring de la función saludar """
    print("Hola! desde función saludar() del módulo prueba")

```

Entonces, usted debe importar el módulo anterior, para consultar la documentación del módulo `mi_modulo` debe utilizar la función integrada `help()` y pasarle el argumento el nombre de módulo `mi_modulo`, de la siguiente manera:

```

>>> import mi_modulo
>>> help(mi_modulo)

Help on module mi_modulo:

NAME

```

```
mi_modulo - El docstring del módulo
FUNCTIONS
despedir()
    El docstring de la función despedir
saludar()
    El docstring de la función saludar
```

 v: 3.7 ▼

También puede consultar la documentación de la función `despedir()` dentro del módulo `mi_modulo`, usando la función integrada `help()` y pasarle el argumento el formato *nombre_modulo.nombre_funcion*, es decir, `mi_modulo.despedir`, de la siguiente manera:

```
>>> help(mi_modulo.despedir)

Help on function despedir in module mi_modulo:

despedir()
    El docstring de la función despedir
```

Opcionalmente, usted puede listar las variables y funciones del módulo con la función `dir()`, de la siguiente manera:

```
>>> dir(mi_modulo)
['__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'despedir',
 'saludar']
```



Como puede apreciar, muchas de estas variables son especiales, puede comprobar sus valores:

```
>>> print(mi_modulo.__name__) # Nombre del módulo
'mi_modulo'
>>> print(mi_modulo.__doc__) # Docstring del módulo
'El docstring del módulo'
>>> print(mi_modulo.__package__) # Nombre del paquete del módulo
```

3.8.9. Formateo de cadenas

Python soporta múltiples formas de formatear una cadena de caracteres. A continuación se describen:

3.8.9.1. Formateo %

El carácter modulo `%` es un operador integrado en Python. Ese es conocido como el operador de interpolación. Usted necesitará proveer el `%` seguido por el tipo que necesita ser formateado o convertido. El operador `%` entonces substituye la frase “%tipodato” con cero o n  v: 3.7 ; del tipo de datos especificado:

```
>>> tipo_calculo = "raíz cuadrada de dos"
>>> valor = 2**0.5
>>> print("el resultado de %s es %f" % (tipo_calculo, valor))
el resultado de raíz cuadrada de dos es 1.414214
```

También aquí se puede controlar el formato de salida. Por ejemplo, para obtener el valor con 8 dígitos después de la coma:

```
>>> tipo_calculo = "raíz cuadrada de dos"
>>> valor = 2**0.5
>>> print("el resultado de %s es %.8f" % (tipo_calculo, valor))
el resultado de raíz cuadrada de dos es 1.41421356
```

Con esta sintaxis hay que determinar el tipo del objeto:

- `%c` = str, simple carácter.
- `%s` = str, cadena de carácter.
- `%d` = int, enteros.
- `%f` = float, coma flotante.
- `%o` = octal.
- `%x` = hexadecimal.


A continuación un ejemplo por cada tipo de datos:

```
>>> print("CMS: %s, ¿Activar S o N?: %c" % ("Plone", "S"))
CMS: Plone, ¿Activar S o N?: S
>>> print("N. factura: %d, Total a pagar: %f" % (345, 658.23))
N. factura: 345, Total a pagar: 658.230000
>>> print("Tipo Octal: %o, Tipo Hexadecimal: %x" % (027, 0x17))
Tipo Octal: 27, Tipo Hexadecimal: 17
```

3.8.9.2. Clase formatter

`formatter` es una de las clases integradas `string`. Ese provee la habilidad de hacer variable compleja de substituciones y formateo de valores usando el método `format()`. Es le permite crear y personalizar sus propios comportamientos de formatos de cadena de caracteres para reescribir los métodos públicos y contiene: `format()`, `vformat()`. Ese tiene algunos métodos que son destinado para ser remplazados por las sub-clases: `parse()`, `get_field()`, `get_value()`, `check_unused_args()`, `format_field()` y `convert_field()`.

3.8.9.2.1. format()

Este método devuelve una versión formateada de una cadena de caracteres, usando substituciones desde argumentos `args` y `kwargs`. Las substituciones son ident  v: 3.7 ▾ llaves `{ }` dentro de la cadena de caracteres (llamados campos de formato), y son sustituidos en el orden con que aparecen como argumentos de `format()`, contando a partir de cero (*argumentos posicionales*).

Esto es una forma más clara y elegante es referenciar objetos dentro de la misma cadena, y usar este *método* para sustituirlos con los objetos que se le pasan como argumentos.

```
>>> tipo_calculo = "raíz cuadrada de dos"
>>> valor = 2**0.5
>>> print("el resultado de {} es {}".format(tipo_calculo, valor))
el resultado de raíz cuadrada de dos es 1.41421356237
```

También se puede referenciar a partir de la posición de los valores utilizando índices:

```
>>> tipo_calculo = "raíz cuadrada de dos"
>>> valor = 2**0.5
>>> print("el resultado de {0} es {1}".format(tipo_calculo, valor))
el resultado de raíz cuadrada de dos es 1.41421356237
```

Los objetos también pueden ser referenciados utilizando un identificador con una clave y luego pasarla como argumento al método:

```
>>> tipo_calculo = "raíz cuadrada de dos"
>>> print(
...     "el resultado de {nombre} es {resultado}".format(
...         nombre=tipo_calculo, resultado=2**0.5
...     )
... )
el resultado de raíz cuadrada de dos es 1.41421356237
```

Formateo avanzado

Este método soporta muchas técnicas de formateo, aquí algunos ejemplos:

Alinear una cadena de caracteres a la derecha en 30 caracteres, con la siguiente sentencia:

```
>>> print("{:>30}".format("raíz cuadrada de dos"))
raíz cuadrada de dos
```

Alinear una cadena de caracteres a la izquierda en 30 caracteres (crea espacios a la derecha), con la siguiente sentencia:

```
>>> print("{:30}".format("raíz cuadrada de dos"))
raíz cuadrada de dos
```

Alinear una cadena de caracteres al centro en 30 caracteres, con la siguiente sentencia:

```
>>> print("{:^30}".format("raíz cuadrada de dos"))
raíz cuadrada de dos
```

 v: 3.7 ▼

Truncamiento a 9 caracteres, con la siguiente sentencia:

```
>>> print("{:.9}".format("raíz cuadrada de dos"))
raíz cua
```

Alinear una cadena de caracteres a la derecha en 30 caracteres con truncamiento de 9, con la siguiente sentencia:

```
>>> print("{:>30.9}".format("raíz cuadrada de dos"))
raíz cua
```

Formateo por tipo

Opcionalmente se puede poner el signo de dos puntos después del número o nombre, y explicitar el tipo del objeto:

- `s` para cadenas de caracteres (tipo `str`).
- `d` para números enteros (tipo `int`).
- `f` para números de coma flotante (tipo `float`).

Esto permite controlar el formato de impresión del objeto. Por ejemplo, usted puede utilizar la expresión `.4f` para determinar que un número de coma flotante (`f`) se imprima con cuatro dígitos después de la coma (`.4`).

```
>>> tipo_calculo = "raíz cuadrada de dos"
>>> valor = 2**0.5
>>> print("el resultado de {0} es {resultado:.4f}".format(tipo_calculo,
resultado=valor))
el resultado de raíz cuadrada de dos es 1.4142
```

Formateo de **números** enteros, rellenos con espacios, con las siguientes sentencias:

```
>>> print("{:4d}".format(10))
10
>>> print("{:4d}".format(100))
100
>>> print("{:4d}".format(1000))
1000
```

Formateo de **números** enteros, rellenos con ceros, con las siguientes sentencias:

```
>>> print("{:04d}".format(10))
0010
>>> print("{:04d}".format(100))
0100
>>> print("{:04d}".format(1000))
1000
```

 v: 3.7 ▼

Formateo de números flotantes, rellenos con espacios, con las siguientes sentencias:

```
>>> print("{:7.3f}".format(3.1415926))
3.142
>>> print("{:7.3f}".format(153.21))
153.210
```

Formateo de números flotantes, rellenos con ceros, con las siguientes sentencias:

```
>>> print("{:07.3f}".format(3.1415926))
003.142
>>> print("{:07.3f}".format(153.21))
153.210
```

3.8.10. Convertir a cadenas de caracteres

Para convertir a *tipos cadenas de caracteres* debe usar la función `str()` la cual [esta integrada](#) en el interprete Python.



Truco

Para más información consulte las funciones integradas para [operaciones en cadenas de caracteres](#).

3.8.11. Ejemplos

A continuación, se presentan algunos ejemplos de su uso:

Definir cadenas de caracteres con comillas simples

```
1 cadena1 = "Texto entre comillas simples,"
2 print(cadena1, type(cadena1))
```

Definir cadenas de caracteres con comillas dobles

```
1 cadena2 = "Texto entre comillas dobles,"
```

```
2 print(cadena2, type(cadena2))
```

Definir cadenas de caracteres con código escapes

 v: 3.7 ▼

```
1 cadena3 = "Texto entre \n\tcomillas simples,"
2 print(cadena3, type(cadena3))
```

Definir cadenas de caracteres con varias lineas

```
1 cadena4 = """Texto linea 1
2 linea 2
3 linea 3
4 linea 4
5 .
6 .
7 .
8 .
9 .
10 linea N"""
11 print(cadena4 + ", ", type(cadena4))
```

Ejemplo operadores de repetición de cadenas de caracteres

```
1 cadena5 = "Cadena" * 3
2 print(cadena5 + ", ", type(cadena5))
```

Ejemplo operadores de concatenación de cadenas de caracteres

```
1 nombre, apellido = "Leonardo", "Caballero"
2 nombre_completo = nombre + " " + apellido
3 print(nombre_completo + ", ", type(nombre_completo))
```

Calcular el tamaño de la cadena con función «len()»

```
1 print("El tamaño de la cadena es:", len(nombre_completo))
```

Acceder a rango de la cadena

```
1 print("Acceso a rango de cadena: ", nombre_completo[3:13])
```

Consulta de ayuda a la función len

```
>>> help(len)

Help on built-in function len in module __builtin__:

len(...)
    len(object) -> integer
```

Return the number of items of a sequence or collection.

Consulta de ayuda a la clase int

 v: 3.7 ▼

```
>>> help(int)

Help on class int in module __builtin__:

class int(object)
|   int([x]) -> integer
|   int(x, base=10) -> integer
|
|   Convert a number or string to an integer, or return 0 if no arguments
|   are given.  If x is a number, return x.__int__().  For floating point
|   numbers, this truncates towards zero.
```

Consulta de ayuda del módulo

```
>>> import datetime
>>> help(datetime)

Help on built-in module datetime:

NAME
    datetime - Fast implementation of the datetime type.

FILE
    (built-in)

CLASSES
    __builtin__.object
        date
            datetime
```

3.8.12. Ayuda integrada

Usted puede consultar toda la documentación disponible sobre las [cadenas de caracteres](#) desde la [consola interactiva](#) de la siguiente manera:

```
>>> help(str)
```


Para salir de esa ayuda presione la tecla `q`.



Importante

Usted puede descargar el código usado en esta sección haciendo clic [aquí](#).

Truco

Para ejecutar el código `tipo_cadenas.py`, abra una consola de comando, acceda a  **v: 3.7** ▼ donde se encuentra el mismo, y ejecute el siguiente comando:

```
$ python tipo_cadenas.py
```

Ver también

Consulte la sección de [lecturas suplementarias](#) del entrenamiento para ampliar su conocimiento en esta temática.

¿Cómo puedo ayudar?

¡Mi soporte está aquí para ayudar!

Mi horario de oficina es de lunes a sábado, de 9 AM a 5 PM. [GMT-4 - Caracas, Venezuela](#).

La hora aquí es actualmente 7:35 PM GMT-4.

Mi objetivo es responder a todos los mensajes dentro de un día hábil.

[Contáctenos en la sección de soporte](#)



What do you think?

1 respuesta



Upvote



Funny



Love



Surprised



Angry

v: 3.7 ▼



Sad

0 Comentarios

1 Acceder ▼

G

Sé el primero en comentar...

INICIAR SESIÓN CON

O REGISTRARSE CON DISQUS ?

Nombre



Comparte

Mejores

Más recientes

Más antiguos

Sé el primero en comentar.

Suscríbete

Política de Privacidad

No vendan mis datos