



PYTHON

Manejo avanzado de cadenas de caracteres

1

MANEJO AVANZADO DE CADENAS DE CARACTERES

ENTRADA Y SALIDA DE DATOS POR CONSOLA

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Entrada y salida de datos por consola.
 - Entrada de datos por consola: función `input()`
 - Permite al usuario introducir información a través del teclado.
 - Admite como argumento un texto que se mostrará al usuario.
- Normalmente, se asigna el valor devuelto por la función a una variable:
 - `respuesta = input("¿Cómo se llama el padre de Pedro?")`

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Entrada y salida de datos por consola.
 - Entrada de datos oculta por consola:
 - Módulo `getpass`
 - función `getpass(prompt)`
 - Permite al usuario introducir información a través del teclado y que esta no sea visible.
 - Admite como argumento un texto que se mostrará al usuario.

```
>>> pin = getpass.getpass("Introduce PIN:")
Introduce PIN:
>>> print(pin)
3310
```

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Entrada y salida de datos por consola.
 - Salida por consola: función `print()`
 - Muestra los datos por consola.
 - Admite la ejecución sin parámetros.
 - Admite un número variable de argumentos separados por coma. Entre cada argumento introduce un espacio.
 - Escribe un salto de línea al final de la cadena.
 - Argumentos de palabra clave:
 - `end` → modifica el salto de línea final
 - `sep` → modifica el espacio como separador

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Entrada y salida de datos por consola.
 - Caracteres de escape. Permiten representar caracteres o elementos no representables de forma convencional.

Carácter	Significado
\\	\
\"	"
\'	'
\n	Salto de línea
\t	Tabulación
\r	Retorno de carro (devuelve el cursor al inicio de la línea)
\b	Backspace (hace retroceder el cursor un carácter)
\a	Reproduce el sonido de aviso del sistema

MANEJO AVANZADO DE CADENAS DE CARACTERES

- **Entrada y salida de datos por consola.**
 - Cadenas crudas. Permiten representar caracteres de escape con facilidad.
 - Anteponiendo `r` o `R` a la cadena a mostrar, los caracteres de escape no se procesan.

```
>>> print("Imprimir caracteres de escape \t es complicado")
Imprimir caracteres de escape      es complicado
>>> print("Imprimir caracteres de escape \\t es complicado")
Imprimir caracteres de escape \t es complicado
>>> print(r"Imprimir caracteres de escape \t es complicado")
Imprimir caracteres de escape \t es complicado
>>> print(R"Imprimir caracteres de escape \t es complicado")
Imprimir caracteres de escape \t es complicado
```

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Salida con formato:
 - Usando *formatted string literals (f-strings)*:
 - Comenzar la cadena con f o F, antes de la apertura de las comillas.
 - Permite utilizar expresiones dentro de la cadena delimitadas por { *expresión* }
 - `cadena = f"Este texto incluye una expresión {a+5}"`

https://docs.python.org/3/reference/lexical_analysis.html#f-strings

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Salida con formato:
 - Usando *formatted string literals (f-strings)*:
 - Permite especificar el número de espacios que desea que ocupe el resultado de la expresión:
 - f"texto literal {expresión:especificador-formato} texto literal"
 - Especificador-formato:
 - numero-espacios → Genérico
 - >numero-espacios → Alineado a la derecha
 - <numero-espacios → Alineado a la izquierda
 - ^numero-espacios → Centrado
 - *numero-espacios*s → Cadenas de caracteres
 - *numero-espacios*d → Números enteros
 - *numero-espacios*f → Números decimales.

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Salida con formato:
 - Usando *formatted string literals (f-strings)*:

```
print("┌",end="")
print("="*99,end="")
print("┐")
for e in empleados:
    print(f"┌ {e[0]:>10} || {e[1]:^20} || {e[2]:60} ||")
print("└",end="")
print("="*99,end="")
print("┘")
```

NASA	Peter Hughes	Chief Technologist
NASA	Bhanu Sood	Deputy Chief Technologist
NASA	Romae Young	Space Technology Mission Directorate (STMD) Program Support
ESA	Josef Aschbacher	Director ejecutivo de la Agencia Espacial Europea
ESA	Matthias Maurer	Astronauta
CNES	Philippe Baptiste	Presidente
Roscosmos	Dmitri Rogozin	Presidente

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Salida con formato:
 - Usando *formatted string literals (f-strings)*:
 - {expresion!a} → Equivale a ejecutar `ascii()`
 - {expresion!s} → Equivale a ejecutar `str()`
 - {expresion!r} → Equivale a ejecutar `repr()`
 - `ascii()` → Devuelve una versión ASCII de un string.
 - `str()` → Devuelve una cadena de caracteres. Usar para los usuarios finales.
 - `repr()` → Devuelve una cadena de caracteres (entrecomillada). Usar para depuración.

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Salida con formato:
 - Usando el método format:
 - `str.format()`
 - Permite construcciones con espacios delimitados con `{}` que se rellenan como parámetros de format.
 - `("El padre de {} se llama {}".format("Luke", "Darth Vader"))`
 - Los espacios pueden referenciarse por posición:
 - `("El padre de {0} se llama {1}".format("Luke", "Darth Vader"))`
 - Los espacios pueden referenciarse por nombre:
 - `("El padre de {hijo} se llama {padre}".format(padre="Darth Vader", hijo="Luke"))`

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Salida con formato:
 - Usando el método format:
 - Los espacios pueden referenciarse por posición y nombres simultáneamente (los posicionales al principio):
 - `("El padre de {hijo} se llama {0}").format("Darth Vader",hijo="Luke")`
 - `("El padre de {hijo} se llama {0}").format(hijo="Luke","Darth Vader") → ERROR`

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Salida con formato:

- Usando el método format:

- Se pueden pasar estructuras complejas:

```
>>> planetas = ["Mercurio", "Venus", "La Tierra", "Marte"]  
>>> ("El planeta más próximo a {p1} es  
{p2}").format(p1=planetas[2], p2=planetas[3])  
'El planeta más próximo a La Tierra es Marte'
```

- Otra alternativa:

```
>>> ("El planeta mas proximo a {0[2]:s} es {0[3]:s}").format(planetas)
```

Posicional

Posicional

Posicional

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Salida con formato:
 - Usando el método format:
 - Combinando estructuras complejas:

```
>>> planetas = ["Mercurio", "Venus", "La Tierra", "Marte"]
>>> distancias = {"Mercurio":222000000, "Marte":399000000}
>>> (" {0[3]:s} se encuentra a {1[Marte]:d} kilómetros de
distancia").format(planetas,distancias)
'Marte se encuentra a 399000000 kilómetros de distancia'
```

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Salida con formato:
 - Formateo manual de cadenas:
 - Métodos de str:
 - `rjust(numero)` → La salida ocupa *numero* posiciones y se justifica a la derecha.
 - `ljust(numero)` → La salida ocupa *numero* posiciones y se justifica a la izquierda.
 - `center(numero)` → La salida ocupa *numero* posiciones y se centra en dicho espacio.
 - `zfill(numero)` → La salida ocupa *numero* posiciones y el espacio a la izquierda se rellena con 0.

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Salida con formato:
 - Estilo “printf”
 - Se construye la cadena, indicando en las posiciones donde van variables el tipo de las mismas en base al siguiente marcado:
 - %s → str
 - %f → float
 - %d → int
 - y a continuación las variables en orden en el formato:
 - %(*variable1*, *variable2*, *variable3*)

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Salida con formato:

- Estilo “printf”

- Ejemplos:

```
>>> ("%s es un planeta del sistema solar" %(planetas[0]))  
'Mercurio es un planeta del sistema solar'
```

```
>>> ("%s es un planeta del sistema solar y se encuentra a  
%d kilómetros" %(planetas[0],distancias["Mercurio"]))  
'Mercurio es un planeta del sistema solar y se encuentra a  
222000000 kilómetros'
```

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Salida con formato:
 - Estilo “printf”
 - Se pueden indicar el número de decimales. Redondea si es necesario:
 - *%.**número_decimales**f*

```
>>> from math import pi
>>> ("El valor de pi es %f" %(pi))
'El valor de pi es 3.141593'
>>> ("El valor de pi es %.4f" %(pi))
'El valor de pi es 3.1416'
```

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Salida con formato:
 - Comillas triples: """
 - Respeta exactamente lo que se escriba, permitiendo introducir bloques.

```
>>> texto = """
```

```
...
```

```
... | _ \ | | |
```

```
... | | ) | | |
```

```
... | _ / | | ' \ / \ ' \
```

```
... | | | | | ( ) | | |
```

```
... | _ \ , \ _ | | \ _ / | |
```

```
... | _ / |
```

```
... | _ /
```

```
... """
```

MANEJO AVANZADO DE CADENAS DE CARACTERES

EXPRESIONES REGULARES

módulo re

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Las expresiones regulares (regex o regexp) son patrones utilizados para la búsqueda dentro de cadenas de caracteres.
- En Python se encuentran en el módulo re.
- <https://docs.python.org/es/3/howto/regex.html#regex-howto>

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Símbolos:
 - $\backslash \rightarrow$ Carácter de escape
 - $\cdot \rightarrow$ Cualquier carácter.
 - \wedge y $\$ \rightarrow$ Al principio y al final de la cadena.
 - $*$ \rightarrow 0 ó más veces.
 - $+$ \rightarrow 1 ó más veces.
 - $?$ \rightarrow 0 ó 1 veces.
 - $\{m\} \rightarrow$ Cardinalidad (m es el número de veces)
 - $\{m,n\} \rightarrow$ Cardinalidad (entre m y n número de veces)
 - $\{m,n\}^? \rightarrow$ Cardinalidad (entre m y n número de veces, buscando el mínimo número de veces posible)

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Símbolos:
 - $[abc] \rightarrow$ Relación de caracteres (a, b ó c).
 - $[a-z] \rightarrow$ Rango de caracteres.
 - $() \rightarrow$ Agrupación.
 - $A|B \rightarrow A \text{ ó } B$, siendo ambas regexp.

MANEJO AVANZADO DE CADENAS DE CARACTERES

- Funciones re:
 - `match()` Determina si la RE coincide con el comienzo de la cadena de caracteres.
 - `fullmatch()` Determina si la RE coincide con la cadena de caracteres completa.
 - `search()` Escanea una cadena, buscando cualquier ubicación donde coincida este RE.
 - `findall()` Encuentra todas las subcadenas de caracteres donde coincide la RE y las retorna como una lista.
 - `finditer()` Encuentra todas las subcadenas donde la RE coincide y las retorna como un término iterado iterator.

MANEJO AVANZADO DE CADENAS DE CARACTERES

```
cadena = """
Mi dirección de correo electrónico es
fernando.paniagua.formacion@gmail.com.
Hay otras direcciones de correo electronico, pero son privadas.
¿Existe la palabra electrónico?
"""

texto_a_buscar = "correo"
texto_encontrado = re.search(texto_a_buscar, cadena)
if texto_encontrado is not None: #Devuelve None o un objeto re.Match
    print("Encontrado")
    print(texto_encontrado.start())#Posición inicio
    print(texto_encontrado.end())#Posición final
    print(texto_encontrado.span())#Tupla con posicion inicial y
posición final
else:
    print("No encontrado")
```

MANEJO AVANZADO DE CADENAS DE CARACTERES

```
texto_encontrado = re.findall(texto_a_buscar, cadena)#Devuelve una lista con
todas las apariciones del patrón
print(texto_encontrado)

print(re.findall("^Mi",cadena))#Empieza por
print(re.findall("privadas$",cadena))#Termina por

print(re.findall("[abc]",cadena))#Contiene las letras a, b ó c
print(re.findall("[@:]gmail", cadena))#Contiene @ o : y después gmail
print(re.findall("[a-d]",cadena))#Contiene las letras entre la a y la d
minúsculas
print(re.findall("[a-zA-Z]", cadena))#Todas las letras mayúsculas y minúsculas
print(re.search("[@#]",cadena))#Si contiene el carácter @ o #

print(re.findall("electr[oó]nico", cadena))#electrónico y electronico, para
evitar los errores ortográficos

print(re.findall("electr[^oó]nico", cadena))#Devuelve electrónico
```

MANEJO AVANZADO DE CADENAS DE CARACTERES

```
nombre = "Fernando"
nombre_similar = "Fisnando"
print(re.match("[A-F]", nombre)) #Match busca al comienzo de cadena de texto.
print(re.match("[a-f]", nombre))
print(re.match("[a-f]", nombre, flags=re.IGNORECASE)) #No distingue mayúsculas de minúsculas
print(re.match("F..nando", nombre)) #. hace de comodín de carácter
print(re.match("F..nando", nombre_similar)) #. hace de comodín de carácter
print(re.match("\d", nombre)) #Empieza por dígito
print(re.match("\d", "1Fernando")) #Empieza por dígito
```

MANEJO AVANZADO DE CADENAS DE CARACTERES

Posible validador de dirección de correos electrónicos:

```
def validarREGEX(cadena):  
    print("validando.....")  
    patron = "[a-z]+(\\.[a-z]+)*@[a-z]+(\\.[a-z])+"  
    if re.match(patron, cadena) is None:  
        return(f"{cadena:40} NO cumple con la expresión regular {patron}")  
    else:  
        return(f"{cadena:40} Sí cumple con la expresión regular {patron}")
```

MANEJO AVANZADO DE CADENAS DE CARACTERES

Comprobación del posible validador de dirección de correos electrónicos:

```
emails=[]
emails.append("fernando.paniagua.formacion@gmail.com")
emails.append("fernando.paniagua@gmail.com")
emails.append("fernando@gmail.com")
emails.append("fernando.gmail.com")
emails.append("fernando@gmail")

for resultado in list(map(validarREGEX, emails)):
    print(resultado)
```