Strings y Unicode en Python

Strings

En Python, la cadenas (string) son una secuencia de caracteres, donde esta puede tener un solo caracter (no existe el tipo *char*) y se comporta como las colecciones de datos en Python (llámese tuplas o listas).

In [1]:

```
texto = "Las cadenas en Python son facil de entender"
print(texto)

# Puedo obtener la primea letra de la cadena
print("Primera letra:", texto[1])

# O puedo obtener la ultima letra
print("Ultima letra:", texto[-1])

# O puedo obtner un extracto de la cadena
print("Lenguaje de programacion:", texto[15:21])

# Y no es necesario especificar el final si es hasta el final de la cadena
print("Ultima palabra:", texto[-8:])

# Y asi tampoco especificar el inicio si es desde es desde el inicio de la cadena
print("Primera palabra:",texto[:2])

# Puedo generar una cadena concatenando cadenas previas
texto = texto + " y de manejar"
print(texto)

# Y puedo utilizar el operador * para repetir una cadena
texto = "En cambio C++ puede ser " + "Z" * 3 + "z" * 3 + "." * 3
print(texto)
```

```
Las cadenas en Python son facil de entender
Primera letra: a
Ultima letra: r
Lenguaje de programacion: Python
Ultima palabra: entender
Primera palabra: La
Las cadenas en Python son facil de entender y de manejar
En cambio C++ puede ser ZZZzzz...
```

Puedo utilizar muchos de los métodos ya conocidos en tuplas y listas, los que se pueden revisar en el sistema de ayuda:

In [1]:

```
dir(texto)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-b0cf96d2a11f> in <module>
----> 1 dir(texto)

NameError: name 'texto' is not defined
```

Como se puede observar, ¡un objeto string tiene muchas funciones disponibles! Algunas ya las hemos estado utilizando (como *split*) y al ser un string una especie de lista también puede utilizar las funciones de lista (como *len* e *index*):

In [1]:

```
print(texto)
print("Numero de letras en texto:", len(texto))
print("El primer espacio en blanco esta en la posicion", texto.index(' '))
print("El segundo espacio en blanco esta en la posicion", texto.index(' ',4))
print("La palabra 'Python' inicia en la posicion", texto.index('Python'))  #ERROR PORQUE INDEX NO ENCUE
NTRA LA FRASE
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-ae3ce6a9d992> in <module>
----> 1 print(texto)
      2 print("Numero de letras en texto:", len(texto))
      3 print("El primer espacio en blanco esta en la posicion", texto.index(' '))
      4 print("El segundo espacio en blanco esta en la posicion", texto.index(' ',4))
      5 print("La palabra 'Python' inicia en la posicion", texto.index('Python'))  #ERROR PORQUE INDEX NO ENCUENTRA LA FRASE

NameError: name 'texto' is not defined
```

Sin embargo, hay funciones especificamente diseñadas para los strings. Se muestras las mas utilizadas:

In [ ]:

```
nombre = "dionisio artemio morales renjifo"
print(nombre); print()

# Podemos poner todo el mayusculas
print(nombre.upper())

# O retorarnarlo todo a minusculas
print(nombre.lower())

# Es un nombre: hay que poner la primera letra en mayusculas
print(nombre.capitalize())
```

In [ ]:

```
texto = "tres tristes tigres comen trigo"
print(texto); print()

# Este trabalenguas es dificil por la cantidad de "t"s, "r"s y a combinacion de las dos en la frase
print("Cuantas 't' hay en la frase anterior?", texto.count('t'))
print("Cuantas 'r' hay en la frase anterior?", texto.count('r'))
print("Cuantas 'tr' hay en la frase anterior?", texto.count('tr'))

# En un string si existe la funcion "find" que retorna el equivalente de "index" en una lista o tupla
print()
print("En que indice se encuentra la palabra 'tigres':", texto.find('tigres'))
```

In [ ]:

```
texto = "        TITULO DE LA OBRA           "
print(texto); print()

# Podemos eliminar los espacios a la derecha (right)
print("|",texto.rstrip(),"|")
# O los espacios a la izquierda (left)
print("|",texto.lstrip(),"|")
# O los espacios tanto a la izquierda como a la derecha (deja un espacio en blanco a ambos lados)
print("|",texto.strip(),"|")
```

In [ ]:

```
texto = "lo que no te mata te hace mas fuerte"
print(texto); print()

# Podemos reemplazar parte de una cadena por otra
print(texto.replace("hace", "hara"))
# O podemos extraer las palabras de una cadenna y almacenarlas en una lista
palabras = texto.split()
print(palabras)
# Y utilizar una lista de palabras para crear una cadena
print(" ".join(palabras))
print("-".join(palabras))
```

Una cosa que se suele pasar por alto es que las strings son *inmutables*, al igual que las tuplas. En los ejemplo se estan generando nuevas cadenas y se estan imprimiendo pero las cadenas originales no estan siendo modificadas (revise el ejemplo anterior: para la impresión se cambio la palabra "hace" por "hara", pero al utilizar nuevamente la cadena para generar la lista de palabras aun continua "hace").

In [ ]:

```
cadena = "esta es una frase de prueba"

#Probemos ssi es inmutable
cadena[0] = 'E'
```

Una cadena por lo tanto no puede cambiar. Todas las operaciones anteriores, si se quieren guardar, se tienen que almacenar en una cadena nueva.

Existen ciertas funciones que permiten validar si un caracter es de cierto tipo:

In [ ]:

```
letra = 'A'
numero = '5'
simbolo = "!"
espacio = " "
enter = "\n"

print(letra.isalpha())
print(numero.isdigit())
print(letra.isalnum())
print(numero.isalnum())
print(simbolo.isalpha())
print(simbolo.isprintable())
print(espacio.isspace())
print(enter.isprintable())
```

Unicode

Como ya se sabe de los estudios previos en lenguajes de programacion, las letras de una cadena se almacenan en Python en forma de código Python utiliza un codigo de caracteres extendido al ASCII original que solo soportaba 127 caracteres para soportar hasta 100,730 caracteres en la version 5.1 del estándar Unicode. Esto le permite a Python manejar simbolos en otros idiomas (acentos, ñ, carcteres cirilos, runas, chinos, coreanos... hasta emojis).

Para conocer el codigo de un caracter, se utiliza la funcion *ord* y para conocer a que caracter e correspode un código se utliza la funcion *chr*

In [ ]:

```
print("La 'A' tiene el codigo Unicode", ord('A'))
print("El codigo Unicode 65 le corresponde a la letra", chr(65))
```

Esto permite manipular caracteres como si fueran numeros, lo que es muy útil para cifrar texto o para hacer modificaciones en una cadena. Por ejemplo el cifrado Cesar, donde cada letra es desplazada hacia la derecha (o la izquierda) un numero de espacios. Digamos que queremos construir un cifrador Cesar de despazamiento 5 a la derecha:

In [3]:

```
texto = "Este es un mensaje super secreto! Mandar el mensaje lineas abajo..."
print("Mensaje:", texto)

# Se necesita desplazar todos los caracteres del texto cinco caracteres a la derecha
# de forma que 'a' -> 'f', 'b' -> 'g', ..., 'z' -> 'e'
texto_cifrado = ''
for letra in texto:
    # Si el caracter es una letra...
    if letra.isalpha():
        # Se desplaza este caracter 5 espacios a la derecha
        code = ord(letra) + 5
        # Y si son las letras "v...z" que se salen del la alfabeto, hay que retornarlas a "a...e"
        # Esto es: 26 caracteres mas abajo
        if letra.islower() and code > ord('z'):
            code -= 26
        elif letra.isupper() and code > ord('Z'):
            code -= 26
        # Se concatena el texto cifrado
        texto_cifrado += chr(code)
    # De lo contrario, si es un numero, espacio en blanco o simbolo de puntuacion...
    else:
        # Se concatena el texto cifrado tal y como esta en el mensjae original
        texto_cifrado += letra

print("Cifrado:",texto_cifrado)
```

```
Mensaje: Este es un mensaje super secreto! Mandar el mensaje lineas abajo...
Cifrado: Jxyj jx zs rjsxfoj xzujw xjhwjyt! Rfsifw jq rjsxfoj qnsjfx fgfot...
```

¿Puede convertir este programa en una funcion? Por ejemplo **cifrado_cesar(cadena, despz)** donde *cadena* sera el agumento con el texto y *despz* el numero de letras a desplazar (sea positivo o negativo).