Python Programming

Módulo 2

Características y operaciones sobre cadenas

Strings, conceptos básicos

Python fue creado para desarrollar scripts (programas para automatización de tareas). El tiempo lo convirtió en un lenguaje de programación multipropósito.

Tiene incorporando todo un conjunto de características que lo hacen ideal para tareas de *scripting*. Por ejemplo, muchas de las herramientas de varios sistemas operativos basados en Linux están escritas en Python, razón por la cual las distribuciones suelen contener uno o dos intérpretes del lenguaje.

Al realizar tareas de scripting trabajaremos todo el tiempo con cadenas de caracteres.

Este trabajo implica tareas como: buscar un carácter, una palabra o frase en una cadena, eliminar caracteres innecesarios, reemplazar unos caracteres por otros, etc. Las cadenas de Python proveen de forma estándar varios métodos (funciones propias del objeto) para realizar este tipo de operaciones.

Primero, tengamos en cuenta que las cadenas en Python funcionan como las colecciones, en donde cada uno de los elementos es un carácter que se puede ubicar por un índice. De modo que las operaciones de «Slicing» y acceso a elementos a través de índices funciona sin diferencias respecto de las listas y tuplas. Hagamos algunas pruebas.

Caracteres :	P	y	t	h	o	n
Índice :	0	1	2	3	4	5
Índice inverso :	-6	-5	-4	-3	-2	-1

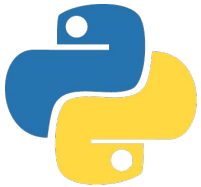
```
>>> texto = "Python"
>>> texto[2]
't'

>>> texto[2:]
'thon'
```

¿Se puede recorrer con un `for`? Si.

```
for n in texto:
    print(n)
```

```
out:
P
y
t
h
o
n
```



Se pueden concatenar:

```
>>> nombre = "Luke"
>>> apellido = "Skywalker"
>>> nombre + " " + apellido
'Luke Skywalker'
```

Es posible la concatenación múltiple (copias):

```
>>> saludo = "Hola!"
>>> saludo*3
'Hola!Hola!Hola!'
```

Las cadenas son inmutables. Esto significa que una vez creadas no pueden modificarse. En efecto, si intentamos modificar una cadena el intérprete nos indica que a ésta no se pueden asignar elementos.

```
>>> texto = "python"
>>> texto[0] = "P"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item
assignment
```

Dentro de poco, vamos a ver como solucionamos esta dificultad.

Caracteres especiales

Los caracteres especiales, como el de salto de línea (`\n`) o tabulación (`\t`), son difíciles de detectar al imprimir una cadena puesto que, justamente, no se imprimen literalmente sino que toman alguna forma en particular: el carácter `\n` se traduce como nueva línea y el carácter `\t` como un segmento vacío de longitud equivalente al de cuatro espacios. Por ejemplo, este código:

```
>>> dato = "\tHola\nChau"
>>> print(dato)
        Hola
Chau
```

Para evitar que los caracteres especiales sean "traducidos" a su representación correspondiente, Python nos provee la función `repr()`.

Que imprime en pantalla, en efecto, los caracteres literalmente:

```
>>> print(repr(dato))
'\tHola\nChau'
```

Esto es especialmente útil cuando obtenemos información de un archivo o alguna otra fuente similar y debemos trabajar con la cadena literal.

Métodos de los strings

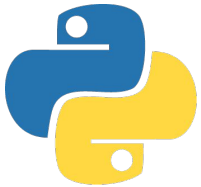
Las cadenas, nos proporcionan métodos que nos facilitan su manipulación. Por ejemplo, una tarea bastante común, es la de querer determinar si una cadena comienza o termina con determinados caracteres, tenemos los métodos `startswith()` y `endswith()` que retornan un booleano.

```
>>> frase = "¿Caminos? a donde vamos no necesitamos caminos"
>>> frase.startswith("¿Caminos?")
True

>>> frase.startswith("¿")
True

>>> frase.endswith("caminos")
True

>>> frase.endswith(".")
False
```



Otra operación bastante común es la de querer eliminar los espacios iniciales y finales de una cadena. Por ejemplo, si le solicitamos el nombre a un usuario a través de la consola (vía la función `input()`) y accidentalmente ingresa un espacio al final antes de presionar la tecla Enter. En ese caso tendríamos una cadena como la siguiente:

```
>>> nombre = "Juan "
>>> nombre.strip()
'Juan'
```

Pero también `.strip()` nos puede ser útil para quitar caracteres especiales del principio o del final de un str.

```
>>> dato = "\t\nHola\nChau"
>>> print(dato.strip())
Hola
Chau
```

Borro el `\t\n` del principio, pero no manipulo el `\n` del medio. Si hubiera estado al final, lo habría borrado también. Esto es especialmente útil al trabajar con los archivos texto (txt), pudiendo eliminar el salto de línea al leer renglones.

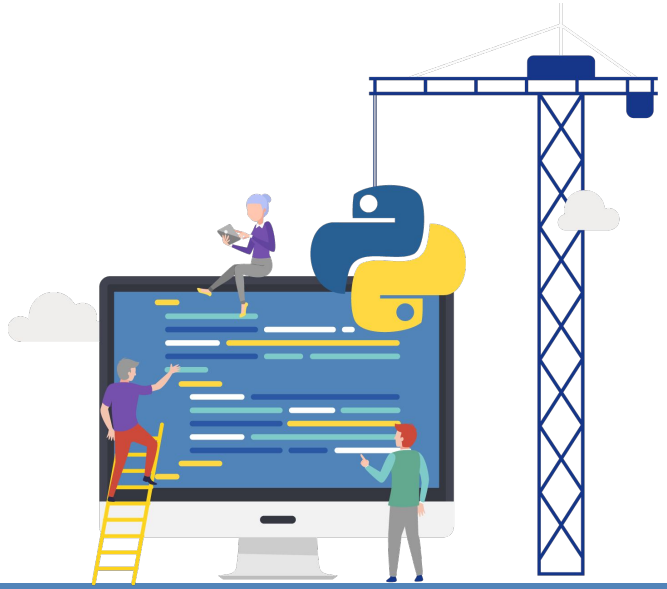
Para reemplazar un (conjunto de) carácter(es) por otro tenemos a disposición la función `replace()`.

```
>>> frase = "¡Nuestro gato se llamaba Indiana!"
>>> frase.replace("gato","perro")
'¡Nuestro perro se llamaba Indiana!'
```

El primer argumento corresponde a la cadena buscada y el segundo, a la cadena que se quiere poner en su lugar. Esto ocurre para todas las apariciones del primer argumento en la cadena original.

El `replace()` retorna una nueva cadena con el reemplazo correspondiente; no modifica la cadena original. Si quisiéramos modificar la original se debería de hacer así:

```
>>> frase = frase.replace("gato","perro")
```



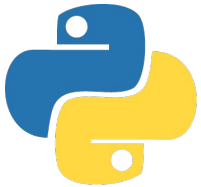
Cuando queremos cortar una cadena en función de un delimitador (como una coma o un espacio vacío), usamos la función `split()`, que retorna una lista.

```
>>> nombres = "Lautaro José Sofía"
>>> nombres.split()
['Lautaro', 'José', 'Sofía']
```

El **`.split()`** sin argumento, utiliza por defecto el carácter espacio vacío como separador. Ahora si deseamos usar otro delimitador podemos pasarlo como parámetro:

```
>>> datos = "Juan,38,Tierra del Fuego"
>>> datos.split(",")
['Juan', '38', 'Tierra del Fuego']
```

Al finalizar los elementos de la lista resultante no contienen el delimitador. Si el delimitador no es encontrado en la cadena, el resultado es una lista de un solo elemento.



El método **`find()`** toma como argumento una cadena y retorna la posición en la que se encuentra dentro de la cadena desde donde fue invocada la función, o -1 en caso de no ser encontrada.

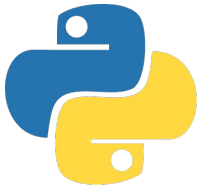
```
>>> cita = "lo esencial es invisible a los ojos"
>>> cita.find("invisible")
15
>>> cita.find("principio")
-1
```

El **`count()`** retorna el número de veces que se repite un conjunto de caracteres especificado:

```
>>> cancion = "...hacer cosas imposibles, cosas
imposibles, imposibles..."
>>> cancion.count("imposibles")
3
```

`.upper()` y **`.lower()`** convierten los caracteres de la cadena a mayúsculas y minúsculas, respectivamente.

```
>>> saludo = "Hola Mundo"
>>> saludo.upper()
'HOLA MUNDO'
>>> saludo.lower()
'hola mundo'
```



Otros métodos

- **`str.isdecimal()`** devuelve True si todos los caracteres de la cadena son decimales y si hay al menos un carácter. Falso en caso contrario. Los caracteres decimales son aquellos que se pueden usar para formar números en base 10
- **`str.swapcase()`** devuelve una copia de la cadena con caracteres en mayúscula convertidos a minúsculas y viceversa.
- **`str.title()`** devuelve una versión en formato título. Las palabras comienzan con un carácter en mayúscula y el resto de caracteres en minúsculas.
- **`str.capitalize()`** devuelve una copia de la cadena con su primer carácter en mayúscula y el resto en minúsculas.

> Puedes encontrar más en la documentación oficial: "métodos de los strings"

¡Sigamos trabajando!