PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN
IIC1103- INTRODUCCIÓN A LA PROGRAMACIÓN

Laboratorio 6
Strings

Segundo semestre, 2015

Objetivo

Este laboratorio es para que practiques la manipulación de strings.

1. Strings como objetos

Hasta ahora, habías usado los strings (cadenas de texto) como un tipo de dato para escribir texto, sin más usos que escribir mensajes en pantalla. En clases, ya habrás conocido la verdad: un string es realmente un **objeto** de la **clase** `str`. Como se trata de una clase, los string poseen varios **métodos** que son muy útiles para manipularlos. A continuación se describen algunos de ellos, junto con algunas funciones y operadores que se aplican a strings.

1.1. Función `len` y operador `in`

La función `len` con un argumento de clase `str`, te entregará el largo del string. Por ejemplo:

```
>>> a = "hola mundo" # string de largo 10 caracteres
>>> len(a)
10
```

El operador `in` opera entre dos strings, y entrega `True` si el string de la izquierda está contenido en el string de la derecha, y `False` en caso contrario. Por ejemplo:

```
>>> "h" in "hola"
True
>>> "u" in "hola"
False
>>> "tren" in "autos, trenes y motos"
True
```

1.2. Operador `[]`

Una operación muy importante es acceder a los caracteres individuales de los strings. Para ello, se usa un operador que especifica los índices que queremos acceder dentro de corchetes `[]` después del nombre de la variable. Ejecuta el siguiente ejemplo para explorar su funcionamiento.

```
>>> a = "Hare todos los ejercicios del laboratorio :D"
>>> a[0] # primer caracter
'H'
>>> a[1] # segundo caracter
'a'
>>> a[0:10] # primeros 10 caracteres
'Hare todos'
```

Si te fijas bien en los ejemplos, el operador `[]` siempre devuelve un objeto de clase `str`. Los strings son `inmutables`, es decir, puedes leer su información, pero no modificarla, a no ser que reemplaces el string por otro. Por ejemplo, si ejecutas la instrucción `a[0] = 'h'` Python arrojará un error. Nota que los índices comienzan en cero (0), por lo que los índices válidos para un string `s` son entre `0` hasta `len(s)-1`.

Como pudiste ver en los ejemplos, al poner dentro de los corchetes dos índices separados por dos puntos (como en `a[0:10]`) se retorna una porción del string original entre las posiciones especificadas por los índices, **incluyendo al primero pero no al segundo**. Esta operación es llamada *slice*. Si escribimos `a[0:10]`, se retornará un string con los caracteres entre los índices 0 al 9 del string `a`. Esto es importante ya que no puedes usar un índice igual o mayor que el número de caracteres del string, excepto si efectúas una operación de slice, donde los índices que no corresponden no serán tomados en cuenta. También, el primer índice debe ser menor al segundo, sino retornará un string vacío. Ejecuta los siguientes ejemplos:

```
>>> a = "Hare todos los ejercicios del laboratorio :D"
>>> len(a)
44
>>> a[0:100] # Sin error, ya que retorna hasta el indice len(a)-1
'Hare todos los ejercicios del laboratorio :D'
>>> a[len(a)] #Esto da un error! NO es un slice!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

En una operación que define un *slice*, si el primer índice es `0`, o si el último es `len(a)`, pueden omitirse. También es válido usar índices negativos, con lo que puedes acceder a los caracteres desde el final. Nota que el índice negativo es sólo para indicar que estás contando desde el final hacia el principio del string, y no es que en el string existan realmente los índices negativos, básicamente es una notación rápida para acceder a la posición `len(s)-n` del string `s`, omitiendo la función `len(s)` de la notación. Analiza los siguientes ejemplos:

```
>>> a = "Hare todos los ejercicios del laboratorio :D"
>>> a[:10]
'Hare todos'
>>> a[10:]
'  los ejercicios del laboratorio :D'
>>> a[-1]
'D'
```

```
>>> a[-2]
':'
>>> a[-14:]
'laboratorio :D'
>>> a[len(a)-14:]
'laboratorio :D'
>>> a[:-21]
'Hare todos los ejercicios'
>>> a[-10:10] # vacio ya que es lo mismo que a[len(a)-10:10], y len(a) = 44
''
```

1.3. Algunos métodos de la clase `str`

Asumamos que tenemos una variable de tipo `str` llamada `s`. Algunos métodos implementados en la clase `str` son:

- `s.upper()`: retorna un string con todos los caracteres de `s` en mayúscula
- `s.lower()`: retorna un string con todos los caracteres de `s` en minúscula
- `s.find(b)`: retorna el índice de la primera aparición del string `b` dentro de `s`. Si no lo encuentra, retorna -1.
- `s.replace(o,a)`: retorna un string con todas las apariciones del string `o` dentro del string `s` cambiados por el string `a`.
- `s.split(b)`: retorna una lista, correspondiente a todos los substrings de `s` separados por el string `a`. Si `a` no aparece en el string `s`, la lista tiene un sólo elemento, correspondiente al string `s` completo. Sin argumentos (sin `b`), retorna una lista de las palabras contiguas, sacando todos los caracteres que separan las palabras, como espacios, tabuladores o saltos de línea.
- `s.strip()`: retorna un string con los espacios al inicio y al final eliminados.

Ejemplos de uso de los métodos de la clase `str`

```
>>> a = "Hare todos los ejercicios del laboratorio :D"
>>> print(a.lower())
hare todos los ejercicios del laboratorio :d
>>> print(a.upper())
HARE TODOS LOS EJERCICIOS DEL LABORATORIO :D
>>> print(a.replace("todos los ejercicios","ningun ejercicio"))
Hare ningun ejercicio del laboratorio :D
>>> print(a.find(":"))
42
>>> print(a.split("ejercicios"))
['Hare todos los ', ' del laboratorio :D']
>>> print(a.split(" ")) # El argumento de split es un espacio
['Hare', 'todos', 'los', 'ejercicios', 'del', 'laboratorio', ':D']
>>> print(a.split()) # split sin argumentos
```

```
['Hare', 'todos', 'los', 'ejercicios', 'del', 'laboratorio', ':D']
>>> b = '   hola mundo       ' # 3 espacios al inicio, 7 espacios al final
>>> b.strip() # retorna un string con los espacios al inicio y al final eliminados
'hola mundo'
```

1.4. Ciclo `for` en strings

Muchas veces necesitas recorrer un string de a un caracter en forma ordenada, por ejemplo si quieres buscar una letra en particular dentro de un string. Python provee de los ciclos `for`, que como verás de ahora en adelante, son una forma muy conveniente para programar ciclos cuando sabes de antemano el número de iteraciones que tu programa debe realizar. Veamos un ejemplo para entender su funcionamiento:

```
>>> cadena = "hola mundo"
>>> for c in cadena:
...     print(c)
...
h
o
l
a

m
u
n
d
o
```

En este caso, la instrucción `for c in cadena:` permite que en cada iteración la variable `c` sea asignada al siguiente caracter del string `cadena`, comenzando por el primer caracter. NO te confundas con el operador `in` descrito anteriormente.

1.5. Ejercicio resuelto

Escribe una función que reciba dos strings (`a` y `b`), que encuentre si `b` está contenido `a`, y que en ese caso retorne el índice de `a` donde se encontró el string `b`. Si no está contenido, debe retornar -1.

Solución 1

```
def buscar(a,b):
    i=0
    while i<len(a):
        if a[i:i+len(b)]==b:
            return i
        i += 1
        print(i)
    return -1
```

Otra solución usando los métodos de la clase `str` es simplemente retornar `a.find(b)`.

2. Ejercicios

1. Programa una función `buscarTodas(a,b)`, que encuentre todas las apariciones del string `b` en el string `a`, y retorne un string que representa a una lista de índices separada por espacios. Por ejemplo, al ejecutar `buscarTodas('tres tristes tigres','t')`, debería retornar el string `'0 5 9 13'`.

2. Programa tu versión del método `replace` como una función que recibe tres strings (`a`, `b` y `c`), y retorna un string con las apariciones de `b` en el string `a` cambiadas por el string `c`. Por ejemplo, si llamas a la función `myreplace('voy a estudiar mucho', 'estudiar','dormir')`, debe retornar `'voy a dormir mucho'`. Por supuesto, no debes usar el método `replace` de la clase `str`.

3. Programa tu versión de la funcion `float`, para convertir un string a un número real. Por ejemplo, tu función `myfloat('3.14159')`, cuyo argumento es un string, debe entregar un número real igual a lo que representa el string, en este caso 3.14159. No debes usar la función `float`.

4. Programa una función que reciba un string, y que retorne el número de palabras, el número total de caracteres, el largo promedio de las palabras, el número de espacios, y el número de caracteres de puntuación (que no sean letras o espacio). Recuerda que puedes usar las operaciones de comparación entre strings. Para escribir strings largos, en python puedes usar el triple apóstrofe en vez de un apóstrofe simple o comillas. Por ejemplo:

   ```
   >>> a = '''Lorem ipsum ad his scripta blandit partiendo, eum fastidii
   accumsan euripidis in, eum liber hendrerit an. Qui ut wisi vocibus
   suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis refor
   midans eu, legimus senserit definiebas an eos. Eu sit tincidunt incor
   rupte definitionem, vis mutat affert percipit cu, eirmod consectetu
   er signiferumque eu per. In usu latine equidem dolores. Quo no fall
   i viris intellegam, ut fugit veritus placerat per.'''
   ```

   Recuerda que puedes retornar varias variables a la vez separándolas por comas en `return`.

5. Programa una clase `Codificador`, que tenga como atributos un string escrito correctamente, otro string que es la codificación del string escrito correctamente, y un tercer string que es una contraseña. El mensaje correcto sólo debe contener letras minúsculas, espacio y signos de puntuación, sin usar tildes en vocales. El constructor de la clase debe recibir la contraseña. Programa los siguientes métodos:

   - `codificar(mensaje)`: codifica el string escrito correctamente en `mensaje` mediante la inversión del alfabeto, es decir, si aparece la letra `a` es cambiada por `z`, la `b` por `y`, etc., y almacena el mensaje codificado en el atributo correspondiente.
   - `imprimir(passwrd)` : imprime en pantalla el string escrito correctamente si el string `passwrd` coincide con la contraseña almacenada; si no coincide, imprime el mensaje codificado.

   Ejemplo:

   ```
   >>> cod = Codificador('1234')
   >>> cod.codificar('el agua esta fria')
   >>> cod.imprimir('4344') # password incorrecto
   ```

```
vo ztfz vhgz uirz
>>> cod.imprimir('1234') # password correcto
el agua esta fria
```

6. Escribe una clase `FechaHora`, con las siguientes características:

   - Ingresar la fecha con un método `fijarFecha` que recibe como argumento un string de formato dd/mm/aaaa o dd-mm-aaaa
   - Ingresar la hora con un método `fijarHora` que recibe como argumento un string en formato HH:MM:SS
   - Ingresar fecha y hora de una sola vez con un método `fijarFechaHora`, que recibe como argumento un string con formato dd/mm/aaaa HH:MM:SS
   - Permitir cambiar cualquier parámetro (día, hora, etc.) con un método `cambiar` que recibe como argumento un string que especifica el tipo de parámetro y su valor, y validarlo. Por ejemplo, si quiero cambiar el día a 2, debería usar como parámetro `'dd=2'`, sin importar los espacios, y debería aparecer un mensaje cuando, por ejemplo, quiera cambiar el día a 40, que sería inválido.
   - Imprimir un objeto de la clase FechaHora mediante `print`, y que imprima en pantalla en formato aaaa/mm/dd HH:MM:SS.

   Ve el siguiente ejemplo de cómo debería operar tu clase.

```
>>> fh = FechaHora()
>>> fh.fijarFechaHora('30-09-2015 17:45:00')
>>> print(fh)
2015/09/30 17:45:00
>>> fh.cambiar('mm=10')
>>> print(fh)
2015/10/30 17:45:00
>>> fh.fijarHora('18:00:00')
>>> fh.cambiar('aaaa = 2016')
>>> print(fh)
2016/10/30 18:00:00
```