Clase 9 – Strings II

IIC1103 Sección 9 – 2019-2

Profesor: Felipe López

01-10-2019

Resumen de la clase

- **Concatenar:** Para unir dos o más *strings* distintos, se ocupa el operador "+".

```
string s1
string s2
concatenacion = s1 + s2
```

Los *strings* pueden ser literales, no solo variables. Notemos que textos de un solo caracter y con caracteres especiales (!,#,@, etc.) **también son *strings*.**

- **Secuencias de caracteres:** Un *string* es una secuencia de caracteres. Cada caracter tiene una posición específica.

Caracter →	h	o	l	a
Posición →	0	1	2	3

Para un *string* `s1`, podemos extraer cualquier carácter de la siguiente manera:

```
s1[k]
```

Siendo k una posición dentro del string.

Resumen de la clase

- **Largo de un string:** Para poder saber cuál es el largo de un *string,* ocupamos la función `len()`.

Ejemplo:

```
s1 = “hola”
len(s1) #igual a 4
```

- ***Slice*:** Es posible extraer “un pedazo” de un string mediante la operación *slice*.

Para un *string* `s1`, podemos extraer una secuencia de caracteres dentro de este string, de la siguiente manera:

```
s1[i:j]
```

Tomando los caracteres entre `i` y `j-1`.

Resumen de la clase

- **`for` sobre un *string*:** Podemos "recorrer" todos los caracteres de un string mediante un `for`.

Ejemplo:

```
s1 = "hola"
for i in s1:
    print(i) #i representa a cada uno de los caracteres de
             #s1 y en este caso se imprimen en la
             #consola.
```

Resumen de la clase

- **For sobre las posiciones de un *string*:** Podemos “recorrer” todas las posiciones de un string

```
s="ejemplo"
for i in range(0,len(s)):
    print(s[i])
```

Donde la variable i representa a todas las posiciones del string s

Ejercicio 3

Escriba una función que intercale dos strings distintos y devuelva un string con la palabra intercalada. En caso que uno sea más largo que el otro, entonces se deben intercalar las palabras hasta que se acaben los caracteres del string más corto y luego añadir lo que queda de la palabra más grande.

Ejercicio 3

```
def intercalar(s1,s2):
  str_aux = ""
  if len(s1) == len(s2):
    for i in range(0,len(s1)):
      str_aux += s1[i] + s2[i]
  elif len(s1) < len(s2):
    for i in range(0,len(s1)):
      str_aux += s1[i] + s2[i]
    str_aux += s2[i+1:len(s2)]
  elif len(s1) > len(s2):
    for i in range(0,len(s2)):
      str_aux += s1[i] + s2[i]
    str_aux += s1[i+1:len(s1)]
  return str_aux
```

Contenidos

1. Comparación de *strings*
2. Funciones de *strings*
3. Ejercicios

Comparación de Strings

- ¿Cómo podríamos comparar *strings* para poder ordenarlos de forma alfabética?
- Python permite que comparemos distintos *strings* para poder saber cuál viene antes o después según un orden alfabético. Para esto podemos ocupar los operadores < o > para comparar *strings*.
- Por ejemplo:

```
variable_a = "a"
variable_b = "b"

print(variable_a < variable_b)
print(variable_a > variable_b)
```

```
True
False
```

Comparación de Strings

- También se pueden ordenar palabras:

```
variable_a = "ejemploa"
variable_b = "ejemplob"

print(variable_a < variable_b)
print(variable_a > variable_b)
```

```
True
False
```

- ¿Y si quisiéramos comparar dos apellidos?

```
variable_a = "Rojas"
variable_b = "López"

print(variable_a < variable_b)
print(variable_a > variable_b)
```

```
False
True
```

Comparación de Strings

- Asimismo, podemos comparar si dos strings son iguales con el operador "==" o distintos con el operador "!=".

```
variable_a = "ejemploa"
variable_b = "ejemplob"
variable_c = "ejemploa"
variable_d = "Ejemploa"

print(variable_a == variable_b)
print(variable_a != variable_b)
print(variable_a == variable_c)
print(variable_a == variable_d)
```

```
False
True
True
False
```

Funciones de Strings

- Al igual que `len()`, existen varias otras funciones que nos permiten trabajar con *strings.*
- Estas funciones son sumamente importantes, ya que nos permitirá trabajar con *strings* de muchas maneras.
- Recordemos que la gran mayoría de los datos se encuentran en formato texto, por lo que conocer funcionalidades que nos permita trabajar con ellos nos dará muchas herramientas para aplicar en la vida real.

Funciones de Strings

Asumamos que `s` y `x` son dos *strings* cualquiera.

- **`s.count(x)`:** Cuenta apariciones de `x` en `s`.

```
#ejemplo 1 count
s = "palabra"
print(s.count("a"))

#ejemplo 2 count
s = "papapapapape"
print(s.count("papa"))
```

```
3
2
```

Funciones de Strings

Asumamos que `s` y `x` son dos *strings* cualquiera.

- **`s.find(x)`:** Retorna la posición de `x` en `s`. Si es que `x` aparece más de una vez en `s`, devuelve la primera ocurrencia. Si `x` no aparece, entonces devuelve -1.

```
#ejemplo 1 find
s = "palabra"
print(s.find("a"))

#ejemplo 2 find
s = "papapapapape"
print(s.find("pe"))
```

```
1
10
```

Funciones de Strings

Asumamos que `s`, `x` e `y` son tres *strings* cualquiera.

- **`s.replace(x,y)`**: Reemplaza `x` por `y` en el *string* `s`.

```
#ejemplo 1 replace
s = "palabra"
print(s.replace("a","z"))

#ejemplo 2 replace
s = "papape"
print(s.replace("pa","Intro. a la progra."))
```

```
pzlzbrz
Intro. a la progra.Intro. a la progra.pe
```

Funciones de Strings

Asumamos que `s` es un *string* cualquiera.

- **`s.upper()`:** Transforma todos los caracteres del *string* a mayúsculas.

```
#ejemplo 1 upper
s = "palabra"
print(s.upper())

#ejemplo 2 upper
s = "papape"
print(s.upper())
```

```
PALABRA
PAPAPE
```

Funciones de Strings

Asumamos que `s` es un *string* cualquiera.

- **`s.lower()`:** Transforma todos los caracteres del *string* a minúsculas

```
#ejemplo 1 lower
s = "palabra"
print(s.lower())

#ejemplo 2 lower
s = "PAPAPE"
print(s.lower())
```

```
palabra
papape
```

Funciones de Strings

Asumamos que `s` y `x` son *string* cualquiera.

- **`x in s:`** Retorna True o False si es que `x` es un string que está dentro del string `s`.

```
#ejemplo 1 in
s = "palabra"
print("abra" in s)

#ejemplo 2 in
s = "papape"
print("zzz" in s)
```

```
True
False
```

Funciones de Strings

Asumamos que `s` es un *string* cualquiera.

- **`s.strip()`:** Remueve espacios y otros caracteres indeseados al principio y final de string.

```
#ejemplo 1 strip
s = "\tejemplo "
print(s.strip())

#ejemplo 2 strip
s="<!-ejemplo<!-"
print(s.strip("<!-"))
```

```
ejemplo
ejemplo
```

Funciones de Strings

Asumamos que `s` es un *string* cualquiera.

- **`s.isdigit()`**: Devuelve True si el string son solo dígitos y False en caso contrario. No considera el separador decimal.

```
#ejemplo 1 isdigit
s = "1234hola"
print(s.isdigit())

#ejemplo 2 isdigit
s = "67584023"
print(s.isdigit())

#ejemplo 3 isdigit
s = "6758.4023"
print(s.isdigit())
```

```
False
True
False
```

break

5 minutos

Ejercicio 1

Escriba una función que cuente cuántas palabras tiene un texto. Puede asumir que además de los saltos de línea el texto no tendrá ningún carácter especial.

Ejercicio 1

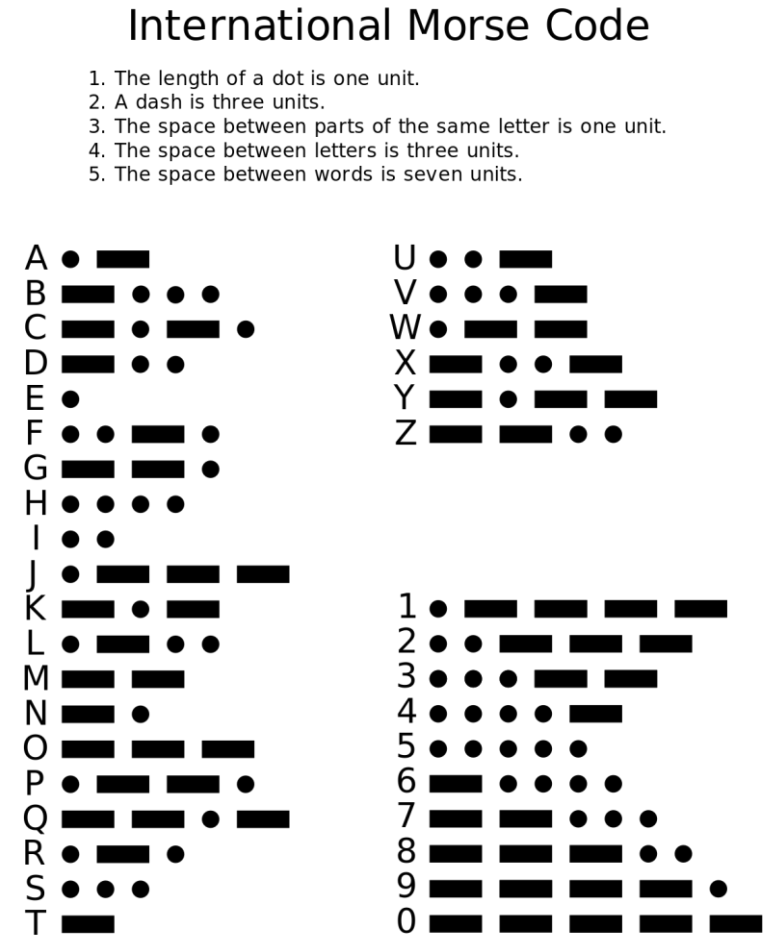
Escriba una función que cuente cuántas palabras tiene un texto. Puede asumir que además de los saltos de línea el texto no tendrá ningún carácter especial.

```
def contar_palabras(s):
  sAux = s.strip()
  sAux = sAux.replace("\n"," ")
  sAux = sAux.replace("  "," ")
  return sAux.count(" ")+1
```

Ejercicio 2

Usted es un cifrador experto y desea mejorar el actual código morse. El código morse actual es el siguiente:

- No tiene un largo constante de secuencias por lo que es difícil para un computador interpretarlo



Ejercicio 2

Para poder mejorar el código morse, usted debe implementar 3 funciones en Python:

1. Una función que pueda transformar un número binario a un decimal.

Ejercicio 2

Para poder mejorar el código morse, usted debe implementar 3 funciones en Python:

1. Una función que pueda transformar un número binario (de cualquier largo) a un decimal.

```
def binario_a_decimal(s):
  num = 0
  for i in range(0,len(s)):
    binary_num = int(s[i])
    num+=binary_num*(2**(len(s)-1-i))
  return num
```

Ejercicio 2

Para poder mejorar el código morse, usted debe implementar 3 funciones en Python:

1. Una función que pueda transformar un número binario a un decimal.
2. Una función que pueda transformar un número binario a un texto. Para esto, puede ocupar la tabla ASCII (que puede ver acá). El programa asume que cada letra en binario de la palabra que desea decodificar tiene un largo de 5 dígitos.

65	41	101	&#65;	A
66	42	102	&#66;	B
67	43	103	&#67;	C
68	44	104	&#68;	D
69	45	105	&#69;	E
70	46	106	&#70;	F
71	47	107	&#71;	G
72	48	110	&#72;	H
73	49	111	&#73;	I
74	4A	112	&#74;	J
75	4B	113	&#75;	K
76	4C	114	&#76;	L
77	4D	115	&#77;	M
78	4E	116	&#78;	N
79	4F	117	&#79;	O
80	50	120	&#80;	P
81	51	121	&#81;	Q
82	52	122	&#82;	R
83	53	123	&#83;	S
84	54	124	&#84;	T
85	55	125	&#85;	U
86	56	126	&#86;	V
87	57	127	&#87;	W
88	58	130	&#88;	X
89	59	131	&#89;	Y
90	5A	132	&#90;	Z

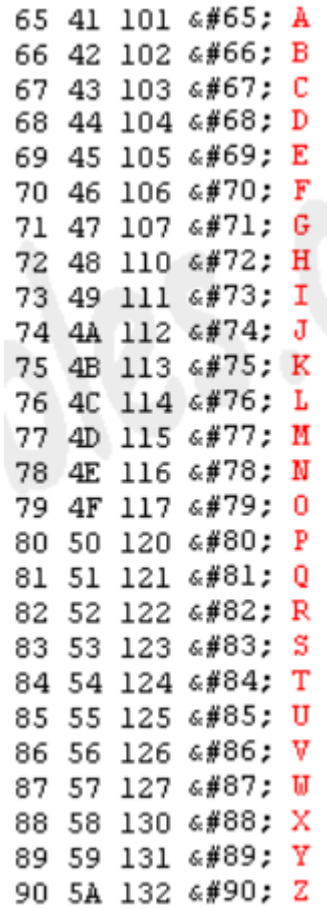
Ejercicio 2

```
def decod_bin(s):
    sAux = ""
    for i in range(0,len(s),5):
        num = binario_a_decimal(s[i:i+5])
        letra = chr(64+num)
        sAux+= letra
    return sAux
```

Ejercicio 2

Para poder mejorar el código morse, usted debe implementar 3 funciones en Python:

1. Una función que pueda transformar un número binario a un decimal.
2. Una función que pueda transformar un número binario a un texto. Para esto, puede ocupar la tabla ASCII (que puede ver acá). El programa asume que cada letra en binario de la palabra que desea decodificar tiene un largo de 5 dígitos.
3. Una función que transforme un código morse a palabras. El código morse puede incorporar ahora "," para separar palabras.



65	41	101	&#65;	A
66	42	102	&#66;	B
67	43	103	&#67;	C
68	44	104	&#68;	D
69	45	105	&#69;	E
70	46	106	&#70;	F
71	47	107	&#71;	G
72	48	110	&#72;	H
73	49	111	&#73;	I
74	4A	112	&#74;	J
75	4B	113	&#75;	K
76	4C	114	&#76;	L
77	4D	115	&#77;	M
78	4E	116	&#78;	N
79	4F	117	&#79;	O
80	50	120	&#80;	P
81	51	121	&#81;	Q
82	52	122	&#82;	R
83	53	123	&#83;	S
84	54	124	&#84;	T
85	55	125	&#85;	U
86	56	126	&#86;	V
87	57	127	&#87;	W
88	58	130	&#88;	X
89	59	131	&#89;	Y
90	5A	132	&#90;	Z

Ejercicio 2

```
def morse_modificado(s):
  sAux = s.replace(".","1").replace("-","0")
  comma_index = sAux.find(",")

  while comma_index != -1:
    bin_word = sAux[0:comma_index]
    print(decod_bin(bin_word))

    sAux = sAux[comma_index+1:len(sAux)]
    comma_index = sAux.find(",")
```

Ejercicio Propuesto

Crea una función recursiva que determine si un string s es palíndromo. La función debe retornar True o False. Deben entregarlo hasta el 1 de octubre a las 18:00 con la secretaria del DCC. Se debe entregar de forma individual.

Resumen de la clase

- **Comparación de strings:** Para comparar dos *strings* distintos, se ocupa el operador “<”,“>”,”==“ ó “!=“.
  Por ejemplo:

```
s1 = “a”
s2 = “b”
print(s1 < s2) #True
```

- Python tiene muchas funciones para operar con *strings*, a continuación veremos algunas de ellas.

Resumen de la clase

Asumamos que `s`, `x` e `y` son *strings* distintos.

- **`s.count(x):`** Retorna la cantidad de ocurrencias de `x` en `s`.
- **`s.find(x):`** Retorna la primera ocurrencia de `x` en `s`.
- **`s.replace(x,y):`** Reemplaza `x` por `y` en `s`.
- **`s.upper():`** Transforma todos los caracteres de `s` a mayúsculas.
- **`s.lower():`** Transforma todos los caracteres de `s` a mayúsculas.
- **`x in s:`** Devuelve `True` o `False` dependiendo si `x` se encuentra en `s`.
- **`s.strip():`** Elimina todos los espacios o caracteres especiales (como "\n") al principio y final de `s`.
- **`s.isdigit():`** Devuelve True si el string son solo dígitos y False en caso contrario. No considera el separador decimal.

Bibliografía

- http://runest.ing.puc.cl/string.html
- A. B. Downey. Think Python: How to think like a computer scientist. Green Tea Press, 2013 -> Capítulo 8

Links

- https://repl.it/@FelipeLopez/IIC1103StringsII que contiene todos los ejemplos de la clase.

Clase 9 – Strings II

IIC1103 Sección 9 – 2019-2

Profesor: Felipe López

01-10-2019