Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Clase 09: Strings

Rodrigo Toro Icarte (rntoro@uc.cl)

IIC1103 Introducción a la Programación - Sección 5

14 de Abril, 2015

Recordatorio clases pasadas...

Tipos básicos de datos

1. Números
   - int (3)
   - float (3.0)
   - complex (3 + 0j)
2. Texto
   - str (“Texto con comillas dobles” o ‘simples’)
3. Booleano
   - bool (True, False)

Recordatorio clases pasadas...

Operaciones sobre números

Operadores básicos

$+a$, $-a$, $a+b$, $a-b$, $a*b$, a/b.

Operadores que preguntamos en pruebas

$a//b$, $a\%b$, $a**b$.

Recordatorio clases pasadas...

Operaciones sobre booleanos

Operadores que retornan bool

$a == b$, $a != b$, $a < b$, $a <= b$, $a > b$ y $a >= b$.

Operadores entre booleanos

not a, a *or* b y a *and* b.

String

1. Números
   - int (3)
   - float (3.0)
   - complex (3 + 0j)
2. **Texto**
   - **str (“Texto con comillas dobles” o ‘simples’)**
3. Booleano
   - bool (True, False)

String

Definición: Un String es una cadena de caracteres.

String

Definición: Un String es una cadena de caracteres.

¿Qué sabemos sobre ellos?

- Definirlos.

String

Strings se definen entre comillas simples o dobles.

```
1 s1 = "String 1"
2 s2 = 'String 2'
```

String

Strings se definen entre comillas simples o dobles.

```
1 s1 = "String 1"
2 s2 = 'String 2'
```

¿Qué sucede en este caso?

```
1 s = "y me dijo: "yo soy tu padre" y yo: "noooo D:""
```

a) Funciona
b) No funciona

String

Soluciones:

```
1 # Comillas simples dentro de comillas dobles
2 s= "y me dijo: 'yo soy tu padre' y yo: 'noooo D:'"
3 # Comillas dobles dentro de comillas simples
4 s= 'y me dijo: "yo soy tu padre" y yo: "noooo D:"'
5 # Caracter de escape \"
6 s= "y me dijo: \"yo soy tu padre\" y yo: \"noooo D:\""
7 # Caracter de escape \'
8 s= 'y me dijo: \'yo soy tu padre\' y yo: \'noooo D:\''
```

"\" es un caracter de escape.

String

"\" permite poner comillas, saltos de línea y tabs dentro de un String.

Secuencia	Significado
\"	Comilla doble
\'	Comilla simple
\n	Salto de línea
\t	Tabulador
\\	Backslash

String

Ejemplo:

```
1 s = "Luke... \n\t \"yo soy tu padre\""
2 print(s)
3 # >>> Luke...
4 #                "yo soy tu padre"
```

String

Ejemplo:

```
s = "Luke... \n\t \"yo soy tu padre\""
print(s)
# >>> Luke...
#                 "yo soy tu padre"
```

Obs: También se pueden usar comillas triples.

```
s = """Luke...
  "yo soy tu padre" """
print(s)
# >>> Luke...
#                 "yo soy tu padre"
```

String

Definición: Un String es una cadena de caracteres.

¿Qué sabemos sobre ellos?

- Definirlos (con comillas simples o dobles).
- Pedirlos al usuario (función *input*).
- Castearlos (funciones *int*, *float*, *complex*, *bool*).

String

Pedir y castear strings:

```
1 # pedir string
2 a = input("Ingrese input")
3
4 # castear string
5 a_int = int(a)
6 a_float = float(a)
7 a_complex = complex(a)
8 a_bool = bool(a)
```

String

¿Cómo paso un número a string con formato?

```
1 a = 3.141526535
2 a_string = str(a)
3 print(a_string) # >>> 3.141526535
```

String

Python permite mapear un número a un string en diversos formatos.

Sintaxis

"%x" % número.

Secuencia	Formato
d	Decimal
o	Octal
X	Hexadecimal
E	Notación Científica
f	Punto flotante
0.nf	Punto flotante con n decimales

String

Ejemplos:

```
1 a = 3.141526535
2 print("%d" %a)    # >>> 3              (decimal)
3 print("%E" %a)    # >>> 3.141527E+00 (científica)
4 print("%f" %a)    # >>> 3.141527       (float)
5 print("%.3f" %a) # >>> 3.142      (float 3 decimales)
```

String

Ejemplos:

```
a = 3.141526535
print("%d" %a)    # >>> 3              (decimal)
print("%E" %a)    # >>> 3.141527E+00 (científica)
print("%f" %a)    # >>> 3.141527       (float)
print("%.3f" %a) # >>> 3.142      (float 3 decimales)
```

```
print("\nDecimal \t Octal \t hexadecimal \t real\n")
i = 1
while(i <= 20):
  print("%d" %i, "\t %o" %i, "\t %X" %i, "\t %.3f" %i)
  i+=1
```

String

Definición: Un String es una cadena de caracteres.

¿Qué sabemos sobre ellos?

- Definirlos (con comillas simples o dobles).
- Pedirlos al usuario (función *input*).
- Castearlos (funciones *int*, *float*, *complex*, *bool*).
- Compararlos (==, ! =, <, <=, >, >=).

String

Comparar Strings:

```
1 "hola" == "hola"        # >>> True
2 "hola" == "oli"         # >>> False
3 "hola" != "oli"         # >>> True
4 "hola" < "oli"          # >>> True
5 "hola" > "a"            # >>> True
6 "A" < "a"               # >>> True
7 "ñ" > "z"               # >>> True
8 "á" > "z"               # >>> True
```

Orden alfabético (mayúsculas primero) ¿Por qué?

String

Definición: Un String es una cadena de *¿caracteres?*.

String

Definición: Un String es una cadena de *¿caracteres?*.

```
1 s = "yo soy tu padre"
```

y	o		s	o	y		t	u		p	a	d	r	e

String

Definición: Un String es una cadena de *¿caracteres?*.

```
1 s = "yo soy tu padre"
```

y	o		s	o	y		t	u		p	a	d	r	e

¿Qué pasa si ejecuto **print(s[1])**?

a) Muestra ‘y’
b) Muestra ‘o’
c) Muestra ‘ ’
d) Error!

String

Definición: Un String es una cadena de *¿caracteres*?.

```
1 s = "yo soy tu padre"
```

y	o		s	o	y		t	u		p	a	d	r	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

String

Definición: Un String es una cadena de *¿caracteres*?.

```
1 s = "yo soy tu padre"
```

y	o		s	o	y		t	u		p	a	d	r	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

¿Qué ocurre internamente en el computador?

String: Formato ASCII

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciicharstable.com

String: Formato ASCII

Internamente, el computador maneja Strings como una secuencias de números.

```
1 s = "yo soy tu padre"
```

y	o		s	o	y		t	u		p	a	d	r	e
121	111	32	115	111	121	32	116	117	32	112	97	100	114	101
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

String: Formato ASCII

Internamente, el computador maneja Strings como una secuencias de números.

```
1 s = "yo soy tu padre"
```

y	o		s	o	y		t	u		p	a	d	r	e
121	111	32	115	111	121	32	116	117	32	112	97	100	114	101
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Obs 1: Python compara Strings mediante ASCII.
Obs 2: Función *ord(.)* recibe caracter ASCII y retorna su id.
Obs 3: Función *chr(.)* recibe un id y retorna caracter ASCII.

String

¿Puedo modificar un String?

String

¿Puedo modificar un String?

y	o		s	o	y		t	u		p	a	d	r	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Ejemplo:

```
1 s = "yo soy tu padre"
2 s[0] = "Y"
3 print(s)
```

¿Qué muestra print?
a) yo soy tu padre
b) Yo soy tu padre
c) Error!

String

Importante: Strings son inmutables!

String

Importante: Strings son inmutables!

Para pasar de “yo soy tu padre” a “Yo soy tu padre” debemos crear un nuevo string.

```
1 s = "yo soy tu padre"
2 print(s)
3 s = "Yo soy tu padre"
4 print(s)
```

String

¿Puedo usar índices negativos?

String

¿Puedo usar índices negativos?

y	o		s	o	y		t	u		p	a	d	r	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Ejemplo:

```
1 s = "yo soy tu padre"
2 print(s[-1])
```

¿Qué muestra print?

a) y

b) e

c) r

d) Error!

String

Los índices negativos cuentan en sentido inverso.

String

Los índices negativos cuentan en sentido inverso.

y	**o**		**s**	**o**	**y**		**t**	**u**		**p**	**a**	**d**	**r**	**e**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

String

Los índices negativos cuentan en sentido inverso.

y	**o**		**s**	**o**	**y**		**t**	**u**		**p**	**a**	**d**	**r**	**e**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Ejemplo:

```
1 s = "yo soy tu padre"
2 j = -15
3 while(j < 15):
4   print(s[j])
5   j += 1
```

String

Los índices negativos cuentan en sentido inverso.

y	**o**		**s**	**o**	**y**		**t**	**u**		**p**	**a**	**d**	**r**	**e**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Ejemplo:

```
1 s = "yo soy tu padre"
2 j = -15
3 while(j < 15):
4   print(s[j])
5   j += 1
```

¿Qué pasa si ejecuto **print(s[15])**?
a) y b) Error!

String

En Python si pedimos un índice inválido el programa se cae.

String

En Python si pedimos un índice inválido el programa se cae.

... luego es importante conocer el tamaño de un string.

String

En Python si pedimos un índice inválido el programa se cae.

... luego es importante conocer el tamaño de un string.

Función **len(.)** recibe un string y retorna su largo.

String

En Python si pedimos un índice inválido el programa se cae.

... luego es importante conocer el tamaño de un string.

Función **len(.)** recibe un string y retorna su largo.

Ejemplo:

```
1 s = input("Ingrese string: ")
2 j = 0
3 while(j < len(s)):
4   print(s[j])
5   j += 1
```

... este código recorre un string de tamaño arbitrario.

String

Recorrer un string es *pan de cada día*.

```
1 s = input("Ingrese string: ")
2 j = 0
3 while(j < len(s)):
4   print(s[j])
5   j += 1
```

String

Recorrer un string es *pan de cada día*.

```
1 s = input("Ingrese string: ")
2 j = 0
3 while(j < len(s)):
4   print(s[j])
5   j += 1
```

... para facilitar el día a día se crearon los **for**

String: for

for: Permite recorrer un string (y listas en general).

Sintaxis

for variable_auxiliar **in** algún_string**:**
    bloque_de_código_for
bloque_de_código_fuera_del_for

String: for

for: Permite recorrer un string (y listas en general).

Sintaxis

for variable_auxiliar **in** algún_string**:**
    bloque_de_código_for
bloque_de_código_fuera_del_for

Ejemplo

```
1 s = input("Ingrese string: ")
2 for i in s:
3   print(i)
```

String: for

for: Permite recorrer un string (y listas en general).

Sintaxis

for variable_auxiliar **in** algún_string**:**
    bloque_de_código_for
bloque_de_código_fuera_del_for

Ejemplo

```
1 s = input("Ingrese string: ")
2 for i in s:
3   print(i)
```

¿Cómo funciona?

String: Operadores básicos

4 operadores básicos

String: Operadores básicos

4 operadores básicos

Operador	Operación
$a + b$	Concatena a y b
n * a	Concatena n veces a
a **in** b	es True ssi a es parte de b
a **not in** b	es True ssi a no es parte de b

String: Operadores básicos

4 operadores básicos

Operador	Operación
$a + b$	Concatena a y b
n * a	Concatena n veces a
a **in** b	es True ssi a es parte de b
a **not in** b	es True ssi a no es parte de b

Ejemplo

```
1 a = "hola"; b = "chao"
2 print(a+b)                # >>> holachao
3 print(3*a)                # >>> holaholahola
4 print("ol" in a)          # >>> True
5 print("ol" not in b)      # >>> True
```

String: Ejercicio

"Cree una función que reciba un string y retorne el mismo string, pero sin los caracteres pares".

String: Ejercicio

"Cree una función que reciba un string y retorne el mismo string, pero sin los caracteres pares".

```
1  def quitar_pares(s):
2    ret = ""
3    i = 0
4    while(i < len(s)):
5      if(i % 2 == 1):  # solo agrego posiciones impares
6        ret += s[i]
7      i += 1
8    return ret
9
10 # llamamos a la función con un string cualquiera
11 print(quitar_pares("yo soy tu padre"))
```

String: Ejercicio

"Cree una función que reciba un string y retorne el string invertido".

String: Ejercicio

"Cree una función que reciba un string y retorne el string invertido".

```
1 def invertir(s):
2   ret = ""          # String nulo!
3   for c in s:
4     ret = c + ret
5   return ret
6
7 # llamamos a la función con un string cualquiera
8 print(invertir("yo soy tu padre"))
```

String: Ejercicio

"Cree una función que reciba un string ***s*** *y retorne* ***True*** *ssi* ***s*** *es un palíndromo (una palabra que se lee igual en ambos sentidos, sin considerar espacios)".*

String: Ejercicio

```
1  def invertir(s):          # retorna "s" invertido
2    ret = ""
3    for c in s: ret = c + ret
4    return ret
5
6  def quitar_espacios(s): # retorna "s" sin espacios
7    ret = ""
8    for c in s:
9      if(c != " "): ret += c
10   return ret
11
12 def palindromo(s):
13   s = quitar_espacios(s)
14   return s == invertir(s)
15
16 # llamamos a la función con un string cualquiera
17 print(palindromo("yo soy tu padre"))
18 print(palindromo("sometamos o matemos"))
```

String: Ejercicio

"Cree una función que reciba un string y dos enteros i, j, tal que $i \leq j$; y retorne la sub-parte del string que comienza en i y termina en j-1".

String: Ejercicio

"Cree una función que reciba un string y dos enteros i, j, tal que $i \leq j$; y retorne la sub-parte del string que comienza en i y termina en j-1".

```
1 def substring(s, i, j):
2   ret = ""
3   while(i < j):
4     ret += s[i]
5     i += 1
6   return ret
7
8 # llamamos a la función con un string cualquiera
9 print(substring("yo soy tu padre",3,6))
```

Ejercicios Propuestos Parte 1

1) Cree una función que resuelve el Capicúa usando strings.

2) Cree una función que retorne el número de palabras presentes en un string (obs: considere que toda palabra válida está separada por un espacio de la anterior).

3) Cree un programa que pida párrafos al usuario hasta que él ingrese un '-1'. Guarde los párrafos en un string (considerando saltos de línea). Al finalizar el programa, muestre al usuario el texto completo ingresado.

Strings: Funciones

Importante: Con lo visto hasta ahora pueden resolver cualquier pregunta relacionada con strings.

Strings: Funciones

Importante: Con lo visto hasta ahora pueden resolver cualquier pregunta relacionada con strings.

... sin embargo python ya tiene programadas algunas operaciones comunes sobre strings.

Strings: Funciones

Ejemplo: Queremos obtener un trozo del string.

Strings: Funciones

Ejemplo: Queremos obtener un trozo del string.

```
1 s = "yo soy tu padre"
```

y	**o**		**s**	**o**	**y**		**t**	**u**		**p**	**a**	**d**	**r**	**e**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

¿Cómo obtenemos la palabra 'soy'?

Strings: Funciones

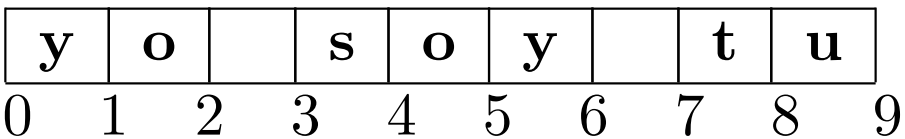
Sintaxis

String[i:j] → retorna el substring entre i y j.

Strings: Funciones

Sintaxis

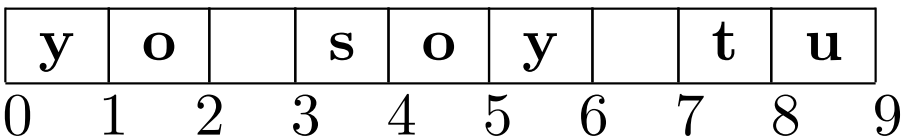
String[i:j] → retorna el substring entre i y j.



Strings: Funciones

Sintaxis

String[i:j] → retorna el substring entre i y j.



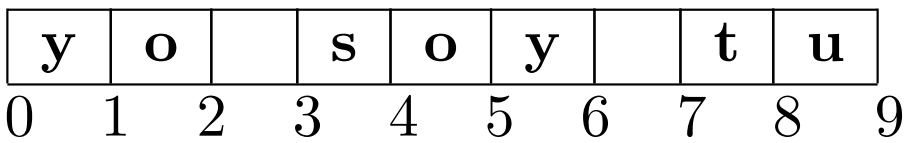

```
1 s = "yo soy tu padre"
2 print(s[3:6])              # >>> soy
```

Strings: Funciones

Observación 1: Podemos **no** especificar el inicio o el fin.

Strings: Funciones

Observación 1: Podemos **no** especificar el inicio o el fin.




```
1 s = "yo soy tu padre"
2 print(s[:6])              # >>> yo soy
3 print(s[3:])              # >>> soy tu padre
```

Strings: Funciones

Observación 2: Podemos dar tercer id para agregar saltos (ej: mostrar caracteres de n en n).

Strings: Funciones

Observación 2: Podemos dar tercer id para agregar saltos (ej: mostrar caracteres de n en n).

```
1 s = "que la curiosidad me mate"
2
3 print(s[::2])    # >>> qel uisddm ae (par)
4 print(s[1::2])   # >>> u acroia emt  (impar)
5 print(s[::-1])   # >>> etam em dadisoiruc al euq
```

Strings: Funciones

Observación 3: ¿Cómo funciona s[i:j:k]?

Strings: Funciones

Observación 3: ¿Cómo funciona s[i:j:k]?

```
1  def substring(s,i,j,k):
2    r = ""
3    while(abs(i) < abs(j)):
4      r += s[i]
5      i += k
6    return r
7
8  s = "yo soy tu padre"
9  print(s[2:7] == substring(s,2,7,1))
10 print(s[-2:-7:-1] == substring(s,-2,-7,-1))
11 print(s[::-1] == substring(s,-1,-len(s)-1,-1))
```

String: Ejercicio

"*Cree una función recursiva que reciba un string* ***s*** *y retorne* ***True*** *ssi* ***s*** *es un palíndromo*".

String: Ejercicio

"Cree una función recursiva que reciba un string ***s*** *y retorne* ***True*** *ssi* ***s*** *es un palíndromo".*

```
1 def pal(s):
2     if(len(s)<=1):
3         return True
4     return s[0]==s[len(s)-1] and pal(s[1:len(s)-1])
5
6 print(palindromo("yo soy tu padre"))
7 print(palindromo("sometamos o matemos"))
```

Strings: Funciones

s.find(c): Retorna el índice de la primera ocurrencia de **c** en **s** (-1 si no existe).

s.rfind(c): Idem pero busca de derecha a izquierda.

Strings: Funciones

s.find(c): Retorna el índice de la primera ocurrencia de **c** en **s** (-1 si no existe).

s.rfind(c): Idem pero busca de derecha a izquierda.

```
1 s = "el mejor placer de la vida es hacer las cosas que
      la gente dice que no podemos hacer"
2
3 print(s.find("d"))                          # >>> 16
4 print(s.rfind("d"))                         # >>> 73
5 print(s.find("vida"))                       # >>> 22
6 print(s.find("paz"))                        # >>> -1
7 print(s[s.find("d"):s.rfind("d")])          # ???
```

Strings: Funciones

s.find(c): Retorna el índice de la primera ocurrencia de **c** en **s** (-1 si no existe).

s.rfind(c): Idem pero busca de derecha a izquierda.

```
1 s = "el mejor placer de la vida es hacer las cosas que
      la gente dice que no podemos hacer"
2
3 print(s.find("d"))                          # >>> 16
4 print(s.rfind("d"))                         # >>> 73
5 print(s.find("vida"))                       # >>> 22
6 print(s.find("paz"))                        # >>> -1
7 print(s[s.find("d"):s.rfind("d")])          # ???
```

find() ✓✓✓ rfind() ✓

Strings: Funciones

s.lower(): Retorna **s** en minúscula.

s.upper(): Retorna **s** en mayúscula.

s.capitalize(): Retorna **s** con primera letra en mayúscula.

Strings: Funciones

s.lower(): Retorna **s** en minúscula.

s.upper(): Retorna **s** en mayúscula.

s.capitalize(): Retorna **s** con primera letra en mayúscula.

```
1 s = "La perseverancia puede cambiar un fracaso en un
      extraordinario logro"
2
3 print(s.lower())       # >>> la perseverancia puede...
4 print(s.upper())       # >>> LA PERSEVERANCIA PUEDE...
5 print(s.capitalize()) # >>> La perseverancia puede...
```

Strings: Funciones

s.lower(): Retorna **s** en minúscula.

s.upper(): Retorna **s** en mayúscula.

s.capitalize(): Retorna **s** con primera letra en mayúscula.

```
1 s = "La perseverancia puede cambiar un fracaso en un
      extraordinario logro"
2
3 print(s.lower())       # >>> la perseverancia puede...
4 print(s.upper())       # >>> LA PERSEVERANCIA PUEDE...
5 print(s.capitalize()) # >>> La perseverancia puede...
```

lower() ✓✓✓ upper() capitalize() ✗

Strings: Funciones

s.strip(c): Retorna **s** sin **c** a ambos costados.

s.lstrip(c): Retorna **s** sin **c** a la izquierda.

s.rstrip(c): Retorna **s** sin **c** a la derecha.

Strings: Funciones

s.strip(c): Retorna **s** sin **c** a ambos costados.

s.lstrip(c): Retorna **s** sin **c** a la izquierda.

s.rstrip(c): Retorna **s** sin **c** a la derecha.

```
1 s = "\n\t      haz de tu vida un sueño, y de tu sueño
      una realidad. :):):)"
2
3 print(s.rstrip(':)')) # quita ":):):)"
4 print(s.lstrip())     # quita "\n\t     " inicial
```

Strings: Funciones

s.strip(c): Retorna **s** sin **c** a ambos costados.

s.lstrip(c): Retorna **s** sin **c** a la izquierda.

s.rstrip(c): Retorna **s** sin **c** a la derecha.

```
1 s = "\n\t      haz de tu vida un sueño, y de tu sueño
      una realidad. :):):)"
2
3 print(s.rstrip(':)')) # quita ":):):)"
4 print(s.lstrip())     # quita "\n\t     " inicial
```

Obs: Por defecto elimina espacios, tabs y saltos de línea.

Strings: Funciones

s.strip(c): Retorna **s** sin **c** a ambos costados.

s.lstrip(c): Retorna **s** sin **c** a la izquierda.

s.rstrip(c): Retorna **s** sin **c** a la derecha.

```
1 s = "\n\t      haz de tu vida un sueño, y de tu sueño
      una realidad. :):):)"
2
3 print(s.rstrip(':)')) # quita ":):):)"
4 print(s.lstrip())     # quita "\n\t      " inicial
```

Obs: Por defecto elimina espacios, tabs y saltos de línea.

strip() ✓ lstrip() ✗ rstrip() ✗

Strings: Funciones

s.startswith(c): Retorna True ssi **s** comienza con **c**.

s.endswith(c): Retorna True ssi **s** termina con **c**.

Strings: Funciones

s.startswith(c): Retorna True ssi **s** comienza con **c**.

s.endswith(c): Retorna True ssi **s** termina con **c**.

```
1 s = "Mientras subía y subía, el globo lloraba al ver
      que se le escapaba el niño."
2
3 print(s.startswith('Mientras s')) # >>> True
4 print(s.endswith('niño'))         # >>> False
```

Strings: Funciones

s.startswith(c): Retorna True ssi **s** comienza con **c**.

s.endswith(c): Retorna True ssi **s** termina con **c**.

```
1 s = "Mientras subía y subía, el globo lloraba al ver
      que se le escapaba el niño."
2
3 print(s.startswith('Mientras s')) # >>> True
4 print(s.endswith('niño'))         # >>> False
```

startswith() ✓✓ endswith() ✓✓✓

Strings: Funciones

s.isalpha(): Retorna True ssi **s** sólo tiene letras.

s.isdigit(): Retorna True ssi **s** sólo tiene números.

Strings: Funciones

s.isalpha(): Retorna True ssi **s** sólo tiene letras.

s.isdigit(): Retorna True ssi **s** sólo tiene números.

```
1 s = "no he fracasado. Sólo acabo de encontrar 10.000
      maneras de no lograrlo."
2
3 print(s.isalpha())           # >>> False
4 print(s[6:15].isalpha())     # >>> True  (fracasado)
5 print(s[41:47].isdigit())    # >>> False (10.000)
6 print(s[41:43].isdigit())    # >>> True  (10)
```

Strings: Funciones

s.isalpha(): Retorna True ssi **s** sólo tiene letras.

s.isdigit(): Retorna True ssi **s** sólo tiene números.

```
1 s = "no he fracasado. Sólo acabo de encontrar 10.000
      maneras de no lograrlo."
2
3 print(s.isalpha())           # >>> False
4 print(s[6:15].isalpha())     # >>> True  (fracasado)
5 print(s[41:47].isdigit())    # >>> False (10.000)
6 print(s[41:43].isdigit())    # >>> True  (10)
```

isalpha() ✓ isdigit() ✓✓✓

Strings: Funciones

s.count(c): Retorna el número de ocurrencias de **c** en **s**.

Strings: Funciones

s.count(c): Retorna el número de ocurrencias de **c** en **s**.

```
1 s = "Fue desamor a primera vista. José Luis Zárate"
2
3 print(s.count('e'))          # >>> 4
4 print(s.count('a '))         # >>> 2
5 print(s.count('desamor'))    # >>> 1
6 print(s.count('fue'))        # >>> 0
```

Strings: Funciones

s.count(c): Retorna el número de ocurrencias de **c** en **s**.

```
1 s = "Fue desamor a primera vista. José Luis Zárate"
2
3 print(s.count('e'))          # >>> 4
4 print(s.count('a '))         # >>> 2
5 print(s.count('desamor'))    # >>> 1
6 print(s.count('fue'))        # >>> 0
```

count() ✓

Strings: Funciones

s.split(c): Divide **s** según las ocurrencias de **c** y retorna las partes en una *lista*.

c.join(l): Une una *lista* **l** mediante **c** y retorna el string formado.

Strings: Funciones

s.split(c): Divide **s** según las ocurrencias de **c** y retorna las partes en una *lista*.

c.join(l): Une una *lista* **l** mediante **c** y retorna el string formado.

```
1 s = "Ella era poesía. Él, puro cuento. Susana
      Landazuri."
2
3 l = s.split(" ")      # separo s en los " "
4 print(l)
5 # >>> ['Ella', 'era', 'poesía.', 'Él,'...]
6 print(l[3])           # >>> Él,
7 print(l[-4])          # >>> puro
8 s2 = "\n".join(l)     # unimos 'l' mediante "\n"
9 print(s2)             # ???
```

Strings: Funciones

s.split(c): Divide **s** según las ocurrencias de **c** y retorna las partes en una *lista*.

c.join(l): Une una *lista* **l** mediante **c** y retorna el string formado.

```
1 s = "Ella era poesía. Él, puro cuento. Susana
      Landazuri."
2
3 l = s.split(" ")      # separo s en los " "
4 print(l)
5 # >>> ['Ella', 'era', 'poesía.', 'Él,'...]
6 print(l[3])           # >>> Él,
7 print(l[-4])          # >>> puro
8 s2 = "\n".join(l)     # unimos 'l' mediante "\n"
9 print(s2)             # ???
```

split() ✓✓✓ join() ✓✓

Strings: Funciones

s.replace(c_1,c_2): Retorna un string igual a **s**, pero reemplazando cada ocurrencia de c_1 por c_2.

Strings: Funciones

s.replace(c_1,c_2): Retorna un string igual a **s**, pero reemplazando cada ocurrencia de c_1 por c_2.

```
1  s = "el éxito llega cuando tus sueños superan tus
       excusas."
2
3  # cambio una frase por otra
4  print(s.replace("tus sueños superan", "superas"))
5
6  # quito los espacios
7  print(s.replace(" ", ""))
8
9  # quito signos de puntuación
10 s = s.replace(".","").replace(",","").replace(":","")
11 s = s.replace(";","").replace("!","").replace("?","")
12 print(s)
```

Strings: Funciones

s.replace(c_1,c_2): Retorna un string igual a **s**, pero reemplazando cada ocurrencia de c_1 por c_2.

```
1  s = "el éxito llega cuando tus sueños superan tus
       excusas."
2
3  # cambio una frase por otra
4  print(s.replace("tus sueños superan", "superas"))
5
6  # quito los espacios
7  print(s.replace(" ", ""))
8
9  # quito signos de puntuación
10 s = s.replace(".","").replace(",","").replace(":","")
11 s = s.replace(";","").replace("!","").replace("?","")
12 print(s)
```

replace() ✓✓✓

Ejercicios Propuestos Parte 2

1) Cree un método que reciba un string y lo retorne en minúscula y sin signos de puntuación (.,;-/¡!¿?) ni caracteres espaciales (\n, \t, \", \', \\).

2) Cree un reconocedor de palíndromos que no considere ni espacios ni signos de puntuación.

3) Implemente la función **mayor_que(s,k)** tal que retorne True ssi **s** tiene una palabra de tamaño mayor o igual a **k**. Notar que el método creado en **1)** le será útil.

4) Cree un método que reciba un string **s** y retorne True ssi **s** es un correo electrónico.

Ejercicios Propuestos Parte 2

5) Cree un método que reciba un string **s** y retorne True ssi **s** puede ser convertido a float.

6) Cree un método que reciba un string **s** y retorne la cantidad de números enteros que en él aparecen.

7) Cree un método que reciba un string **s** y retorne su traducción a jerigonzio (ej: rodrigo → ropodripigopo).

8) Cree una función que reciba **s** y retorne True ssi **s** es un pangram (contiene todas las letras del alfabeto sin considerar eñes ni tíldes).

Ejercicios Propuestos Parte 2

9) Cree el codificador ROT-n. Este codificador mapea cada letra del string a la letra que está n posiciones delante de ella en el alfabeto (sin considerar la eñe). Ej: En ROT-13: a $\rightarrow$ n; b $\rightarrow$ o, ..., m $\rightarrow$ z, n $\rightarrow$ a, ..., z $\rightarrow$ m. Para esto, use las propiedades del formato ASCII.

Pycharm

Página oficial: link
Descargar: Link