</ Ayudantía 5: Strings />

Rolando Rojas

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Temas a tratar

{01}
Introducción a Strings

{02}
Slicing de strings

{03}
Funciones y consideraciones útiles

{04}
Comparaciones de Strings

{05}
Recorriendo Strings [while/for]

{06}
Ejercicios / Tarea

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Introducción a Strings

Los Strings (o cadenas de texto) son secuencias de caracteres, como letras, números y símbolos, que se utilizan para representar texto en programación. En Python, los Strings se encierran entre comillas simples ('') o comillas dobles (""). Puedes usar strings para almacenar y manipular palabras, frases, direcciones de correo electrónico, contraseñas y mucho más en tus programas.

1	-> No string
“1”	-> String
*(x*y)**2*	-> No string
*“(x*y)**2”*	-> String
True	-> No string
“True”	-> String

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Estructura de un string

Mediante el uso de índices y rangos podemos interactuar con los caracteres que están dentro del string.
Mediante índices podemos acceder a caracteres específicos dentro del string
***Ejemplo**: frase = "Quiero un completo"*

-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	**-8**	-7	-6	-5	-4	-3	-2	-1
Q	**u**	**i**	**e**	**r**	**o**		**u**	**n**		**c**	**o**	**m**	**p**	**l**	**e**	**t**	**o**
0	1	2	3	4	5	6	7	8	9	**10**	11	12	13	14	15	16	17

print(frase[0]) → "Q"
print(frase[12]) → "m"

print(frase[-18]) → "Q"
print(frase[-6]) → "m"

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Slicing de un string

Además de lo anterior, podemos acceder a secciones de un string:
***Ejemplo**: frase = "Quiero un completo"*

-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	**-8**	-7	-6	-5	-4	-3	-2	-1
Q	**u**	**i**	**e**	**r**	**o**		**u**	**n**		**c**	**o**	**m**	**p**	**l**	**e**	**t**	**o**
0	1	2	3	4	5	6	7	8	9	**10**	11	12	13	14	15	16	17

frase[10:16] → *"comple"*
frase[10:] → *"completo"*
frase[:6] → *"Quiero"*
frase[-11:] → *"un completo"*

1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1

</ Consideraciones (1/3): Operadores en strings

Concatenación

“Sumar” dos strings resulta en un nuevo string formado por los sumados:

```
"IWI"+"-"+"131"
'IWI-131'
"Concatenación" + " " + "de" + " " + "strings"
'Concatenación de strings'
```

Repetición

“Multiplicar” un string hace que se repita n veces:

```
5*"IWI"
'IWIIWIIWIIWIIWI'
7*"A"
'AAAAAAA'
```

Comparación

Usando operadores relacionales podemos comparar strings:

```
"A"<"B"
True
"A"<"a"
True
"a">"b"
False
"Hola" == "hola"
False
"Algo" == "Algo"
True
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Consideraciones (2/3): Nuevos operadores en strings

len() Es la cantidad de elementos en un objeto. En strings, nos sirve para ver el largo:

```
len("1234567")
7
len("Cuantos carácteres tiene este string")
36
```

in Sirve para ver si un string es parte o está dentro de otro:

```
"texto" in "String con la palabra texto"
True
```

.upper() Transforma un string a MAYÚSCULAS:

```
"string".upper()
'STRING'
```

.lower() Transforma un string a minúsculas:

```
"MAYUSCULAS".lower()
'mayusculas'
```

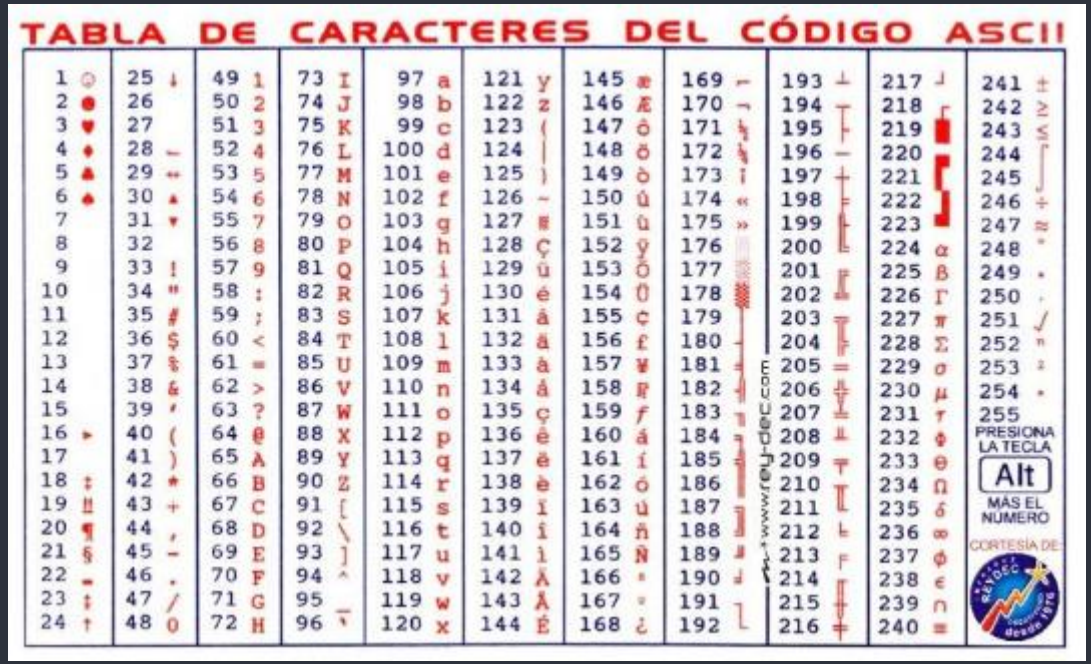
1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Consideraciones (3/3): Código ASCII... ¿Cómo comparar strings?

El código ASCII asigna un valor numérico único a cada carácter.
Con esto podemos comparar strings:

```
>>> "A"=="a"
False
>>> "A">"a"
False
>>> "A"<"a"
True
```

¿Cómo acordarse?
Primero vienen números, luego las mayúsculas y después las minúsculas



1 0 1 1 0 1 1 1 0 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1

</ Recorriendo strings: ciclo while

¿Cómo recorrer un string con un ciclo while?
Haga un programa que imprima solo las consonantes de un string:

```
texto = "ejemplo de recorrido"

indice = 0
while indice < len(texto):
    caracter = texto[indice]
    if caracter != " " and caracter not in "aeiouAEIOU":
        print(caracter)
    indice += 1
```

```
j
m
p
l
d
r
c
r
r
d
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Recorriendo strings: ciclo for [1/2]

¿Qué es el ciclo for?

*Un ciclo **for** en Python es una estructura que te permite repetir un bloque de código varias veces. Te permite recorrer cada carácter de un string, lista, entre otros, y realizar operaciones en cada uno de ellos.*
*En lugar de escribir el mismo código una y otra vez, puedes usar un ciclo **for** para automatizar el proceso.*

```
for (elemento) in (estructura):
        ...código a repetir
```

```
for (caracter) in (string):
        ...código a repetir
```

```
for (elemento) in (lista):
        ...código a repetir
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Recorriendo strings: ciclo for [2/2]

¿Cómo recorrer un string con un ciclo for?
Haga un programa que imprima solo las consonantes de un string:

```
texto = "ejemplo de recorrido"

for caracter in texto:
    if caracter != " " and caracter not in "aeiouAEIOU":
        print(caracter)
```

```
j
m
p
l
d
r
c
r
r
d
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Ciclo for v/s Ciclo while

Ambos ciclos son alternativas válidas para programar. Básicamente todo lo que elijas programar lo puedes hacer con ambos ciclos

Ejemplo:
Haga un programa que cuente cuántas veces aparece la letra "a" en una cadena de texto.

```
Usando ciclo for:
La letra 'a' aparece 3 veces

Usando ciclo while:
La letra 'a' aparece 3 veces
```

```python
texto = "banana"

# Uso de ciclo for
contador_for = 0
for caracter in texto:
    if caracter == "a":
        contador_for += 1

print("Usando ciclo for:")
print("La letra 'a' aparece", contador_for, "veces")

# Uso de ciclo while
contador_while = 0
indice = 0
while indice < len(texto):
    if texto[indice] == "a":
        contador_while += 1
    indice += 1

print("\nUsando ciclo while:")
print("La letra 'a' aparece", contador_while, "veces")
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1

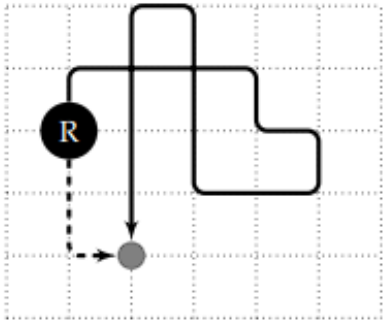
</ Ejercicio certamen: Robot

3. [40 %] El robot R ha sido diseñado para moverse a lo largo de una grilla. R recibe como entrada un string con las letras `n`, `s`, `o` y `e`, que le ordenan moverse, respectivamente, un metro hacia el norte, el sur, el oeste y el este.

En vez de obedecer ciegamente, R tiene la capacidad de optimizar la ruta, de modo de llegar al mismo destino siguiendo el camino más corto posible a lo largo de la grilla.

Escriba un programa que reciba como entrada la ruta original entregada al robot, y que muestre como salida la ruta optimizada:

```
Ruta: neeesesoonnnossss
Ruta optimizada: sse
```



Nota: En el ejemplo, la ruta optimizada también pudo ser `ess`. Ambas soluciones son correctas.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

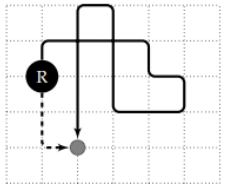
</ Ejercicio certamen: Robot

3. **[40 %]** El robot R ha sido diseñado para moverse a lo largo de una grilla. R recibe como entrada un string con las letras `n`, `s`, `o` y `e`, que le ordenan moverse, respectivamente, un metro hacia el norte, el sur, el oeste y el este.

En vez de obedecer ciegamente, R tiene la capacidad de optimizar la ruta, de modo de llegar al mismo destino siguiendo el camino más corto posible a lo largo de la grilla.

Escriba un programa que reciba como entrada la ruta original entregada al robot, y que muestre como salida la ruta optimizada:

```
Ruta: neeesesoonnnossss
Ruta optimizada: sse
```



Nota: En el ejemplo, la ruta optimizada también pudo ser `ess`. Ambas soluciones son correctas.

```python
ruta = input("Ruta: ")
norte = 0
sur = 0
este = 0
oeste = 0
ruta_optima = ""
for i in ruta:
    if i == "n":
        norte += 1
    elif i == "s":
        sur += 1
    elif i == "e":
        este += 1
    elif i == "o":
        oeste += 1
if norte >= sur:
    resultado = norte - sur
    ruta_optima += "n" * resultado
else:
    resultado = sur - norte
    ruta_optima += "s" * resultado
if este >= oeste:
    resultado = este - oeste
    ruta_optima += "e" * resultado
else:
    resultado = oeste - este
    ruta_optima += "o" * resultado
print(ruta_optima)
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1

Tarea

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Ayudantía 5:
Strings
/>

¿Dudas?

rolando.rojass@usm.cl

Rolando Rojas

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1