Iteración sobre secuencias mediante `for` 3

(0)

```
def mistery(elements):
    qq = []
    for element in elements:
        qq.append(element + 42)
    return qq
```

1. ¿Qué hace esa función? No lo evalúes en Python, evalúalo en tu cabeza.
2. ¿Qué caraterísticas tiene la salida y qué tiene que ver con la entrada?
3. ¿Cómo la llamarías? ¿Y a la variable `qq` ?

(1)

Construye una función llamada `make_lengths` que recibe una lista de cadenas y devuelve: una *lista de números* , de la *misma longitud* de la lista de entrada. Cada uno de esos números es la longitud de cada una de las cadenas de entrada.

Por ejemplo:

```
["hey", "hola", "mundo"] --> [3, 4, 5]  [] --> []
```

(2)

Considerando que ya tenemos la función `sum_all` , ¿qué hace esta función y cómo la llamarías?:

```
def mistery(words):
    return sum_all(make_lengths(words))
```

(3)

Crea la función `make_inverses` que recibe una lista de números, y devuelve otra lista de igual longitud con los inversos de cada uno de los números de entrada. Respeta el orden.

(4)

Crea una función llamada `elevate` que recibe dos parámetros:

- una lista de números
- un número llamado exponent

Devuelve una nueva lista, con el resultado de elevar cada uno de los elementos originales a exponent.

(5)

Crea la función `to_strings` que recibe una lista de números y devuelve una nueva lista que contiene esos números convertidos en cadenas: 3 --> "3".

Para la conversión de `int o float a str` se usa el tipo de destino (en este caso `str`) y se le llama como si fuese una función. Por ejemplo: `str(8)` .

(6)

Crea la función `to_floats` que recibe una lista de números y devuelve una nueva lista que contiene esos números convertidos en números decimales (`float`).

Por ejemplo:

```
[1.01, 8, 42] --> [1.01, 8.0, 42.0]
```

Recuerda que el tipo de destino es `float` .

(7)

Crea la función `round_list` que recibe una lista de números (enteros o decimales) y devuelve una lista de decimales redondeados a dos posiciones.

```
[1.01, 8, 42.009] --> [1.01, 8.00, 42.01]
```

1. Averigua cómo funciona `round` de Python
2. Aprovecha la función `to_floats`

(8)

Necesitas una función que recib euna lista de números y los redondea a varias posiciones decimales (1, 2, 4, etc).

Modifica `round_list` para que pueda hacer eso.

(9)

Lo más probable es que al enterarte de que parte de la función `round_list` tendrá que variar de una llamada a otra (el número de posiciones decimales) hayas sacado eso a un parámetro.

¡Enhorabuena!

Sin embargo, ahora tienes un problema. Antes la función se llamaba así:

`round_list([2,45.9, 432.896])` con un solo parámetro

Ahora tiene dos, y tu código ya no compila. Lo que es peor, tu función es parte de un módulo de python que es usado por millones de programadores. No sólo tú, sino todos ellos van a tener que cambiar su código.

Para evitar eso, vamos a usar una cosa llamada *parámetros por defecto*.

Parámetros por defecto

Hasta ahora, todos los parámetros de nuestras funciones reciben su valor cunado la llamas. Sin embargo, también les puedes dar un valor por defecto.

Por ejemplo, tenemos una función llamada `greet` (saluda) que recibe un nombre y devuelve una cadena con un saludo para ese nombre:

```
def greet(name):
    return 'Hello " + name + "!"
```

Ahora necesitamos poder especificar el saludo para que se pueda usar en varios idiomas.

Sacamos algo que está metido con calzador en el cuerpo de la función y lo convertimos en un parámetro.

> Este es el procedimiento estandar para hacer que neustro software sea más flexible

```
def greet(name, greeting):
    return greeting + name + "!"
```

Ahora ya podemos saludar en Klingon (https://www.youtube.com/watch?v=auqS6FR_RDE) o en Japonés:

```
greet("Manolo", "nuqneH")
greet("Lucas", "Konnichiwa")
```

Es más flexible, pero también algo más coñazo, porque resulta que en la inmensa mayoría de los casos, vas a saludar en Klingon.

Sería bueno que se pudiese llamar a la función con un sólo parámetro para el caso más común (Klingon, of course) y con dos en las excepciones (Valyrio, Dothraki, etc)

Para ello le daremos al parámetro `greeting` el valor por defecto común: *nuqneH*

```
def greet(name, greeting = "nuqneH"):
    return greeting + " " + name + "!"
```

Cuando vamos a usar el valor por defecto, no tenemos que mencionar el segundo parámetro:

```
greet('Worf')
```

Para los casos específicos, pasamos el valor correcto de la siguiente manera:

```
greet('Worf', greeting= "Konnichiwa") # Japonés
greet('Worf', greeting= "Athchomar chomakea") # Dothraki
```

1. Comprueba que lo entiendes, que sabes quién es Worf, los Klingon y los Dothraki.
2. Aplica esto para que se pueda pasar el número de posiciones a `round_list` pero que el código existente no cambie cuando el número de posiciones es 1.

(10)

Crea la función `update_salaries` que recibe una lista de nóminas y crea una nueva lista con las nóminas aplicando los siguientes cambios:

- nómina < 900 se incrementa en un 30%
- 900 <= nómina < 1500, se incrementa en un 12%
- 1500 <= nómina < 6000, se queda tal cual
- nómina > 6000, se incrementa en un 100% (ventajas de ser alto cargo)

(11)

Si al resultado de `update_salaries` le aplicas un `sum_all` ¿qué te sale?

(12)

Crea la función `double_if_neg` que recibe una lista de números y devuelve una nueva lista de números con el siguiente cambio: si un número es negativo, se le multiplica por dos.

(13)

Crea la función `to_lists` que recibe una lista y devuelve otra con cada uno de los elementos de la de entrada convertido en una lista. Es decir, devolverá una lista de listas.

(14)

Crea la función `explode` . Recibe una cadena y la convierte en una lista de caracteres.

Por ejemplo:

```
"hola" --> ["h", "o", "l", "a"]  "" --> []
```

No vamos a trabajar sobre listas, sino sobre cadenas. No pasa nada, las cadenas son secuencias y se pueden manejar igual. Lo único que cambia es:

- Vas a iterar por los caracteres de la cadena
- ¿Cual es el valor inicial del acumulador que vas a usar?

1. La función `explode` es reversible?
2. ¿Te acuerdas de `implode` ?
3. ¿Qué pasa si evalúas `implode(explode("hola"))` ?

(15)

La *Lengua de la P* es un juego de niños común en Brasil en el que se alteran las palabras convirtiéndolas en una especie de trabalenguas.

Si una letra es una vocal, se le transforma en una sílaba con la p y las dos vocales:

- a -> apa
- e -> epe
- i -> ipi
- o -> opo

- u -> upu

Aplicando esto, las siguientes palabras se convertirían así:

- Basico -> Bapasipicopo
- Raro -> Raparopo
- Alamo -> Apalapamopo

Crea la función `to_p_sentence` que recibe una frase (una cadena) y la convierte a la *Lengua de la P*.

PISTAS

1. DIVIDE & VENCERÁS: lo más sencillo sería convertir un caracter a la lengua de la p. Para ello, si es una vocal, aplicas las normas de arriba, y si es cualquier otra cosa, se deja tal cual.
2. Cuando tengas eso resuelto, intenta aplicarlo a una frase.

(16)

1. ¿La función `to_p_sentence` es reversible?
2. ¿Te parece que sería una buena opción para encriptar textos secretos?

(17)

Parece que la transformación que aplica `to_p_sentence` es un poco predecible y cualquiera que reciba un texto encriptado de esa manera podría transformarlo de vuelta.

Vamos a intentar mejorarlo, haciendo que no se aplique la transformación a todas las vocales, sino sólo a aquellas que estén en una posición par de la cadena:

0	1	2	3	4	5
a	t	r	a	p	a
apa	t	r	a	p	a

atrapa --> apatrapa

Modifica la función `to_p_sentence` para que haga esto.

PISTA

Nos encontramos con una situación un poco destinta a la habitual. El bucle `for` de Python nos va dando el elemento de la secuencia (lista o cadena) sin que nos tengamos que preocupar por la posición en la que se encuentra.

La buena vida se acabó, ahora tenemos que llevar cuenta del índice:

- crear una variable para ir guardándolo
- darle un valor inicial
- ir actualizando ese valor a cada vuelta que da el bucle

```
    index = ??
    for element in elements:
```

(18)

Crea la función `foo` que recibe una lista de números y devuelve una nueva aplicando las siguientes reglas:

- si el índice es par, el valor se multiplica por 2
- si el índice es divisible por 3, el valor se divide por 4
- sino, el valor se eleva al cuadrado.

(19)

Crea la función `fizzbuzz` que recibe una lista de números y devuelve otra lista, de la misma longitud, aplicando la siguiente transformación a cada uno de los elementos originales:

- Si el elemento es *divisible por 3*, se le reemplaza por la cadena `Fizz`.
- Si el elemento es *divisible por 5*, se le reemplaza por la cadena `Buzz`
- Si el elemento es divisible *tanto por 3 como por 5*, se le reemplaza por la cadena `Fizz Buzz`.
- Sino, se deja tal como está

`[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]` se debería de convertir en `[1, 2, 'Fizz', 4, 'Buzz', 'Fizz', 7, 8, 'Fizz', 'Buzz', 11, 'Fizz', 13, 14, 'Fizz Buzz', 16, 17, 'Fizz', 19, 'Buzz']`

(20)

¿Lo has conseguido? ¡Enhorabuena! Acabas de superar una [pregunta clásica de entrevista de trabajo (https://blog.codinghorror.com/why-cant-programmers-program/)](https://blog.codinghorror.com/why-cant-programmers-program/).

Sal al rellano de la escalera y pega el [grito de Tarzán (https://www.youtube.com/watch?v=MwHWbsvgQUE)](https://www.youtube.com/watch?v=MwHWbsvgQUE) para que se entere todo el vecindario. Parece que esto marcha.

(21)

Ahora crea la función `tarzan`. Recibe una lista de números y devuelve otra lista, de la misma longitud, aplicando la siguiente transformación a cada uno de los elementos originales:

- Si **el índice** del elemento es *divisible por 3*, se le reemplaza por la cadena `ooo`.
- Si **el índice** del elemento es *divisible por 5*, se le reemplaza por la cadena `uuu`
- Si **el índice** del elemento es divisible *tanto por 3 como por 5*, se le reemplaza por la cadena `aaa`.
- Sino, se deja el elemento tal como está

(22)

1. ¿En qué se parecen todas estas funciones?
2. ¿Qué tiene cada una de particular?
3. ¿Identificas algún patrón?

Repaso y Conclusiones

NO LEAS ESTO ANTES DE HACER LOS EJERCICIOS

Describe con tus palabras aquello que has visto de común que tienen todas estas funciones.

NO SIGAS ANTES DE HACER ESO

Transformadores

Todas estas funciones reciben una secuencia y devuelven otra secuencia de la misma longitud. La secuencia de salida tiene los elementos de la primera transformados de alguna forma, a veces cambiando el tipo, otras veces no.

Todas ellas *transforman* a todos y cada uno de los elementos.

Todas ellas van *construyendo una nueva secuencia* con los elementos transformados.

Para ello, se usa una variable que empieza con el elemento nulo que corresponde (`[]` para listas y `""` para cadenas) y a ella se van añadiendo los elementos transformados.

Cuando llegas al final de la secuencia, la nueva secuencia transformada está en la variable, list apara salir del horno.

El patrón general de un **transformador** es:

```
# para listas
def transfrom(elements):
    transformed = []
    for element in elements:
        transformed.append(<transformation>(element))
    return transformed

# para cadenas
def transfrom(elements):
    transformed = ""
    for element in elements:
        transformed = transformed + <transformation>(element)
    return transformed
```

Esqueleto común y partes específicas

Más arriba hemvos sito el esqueleto común a todo transformador. Lo que es específico de cada uno está entre <>:

- la transformación que se aplica a cada elemento

Todas estas funciones se parecen demasiado.

> DRY: Do not repeat yourself

Estamos cometiendo un grave pecado contra el principio DRY y deberíamos buscar una solución.

¿Qué solución se te ocurre a tí?

Más adelante resolveremos esto.

¿Reversible?

Todo depende de la operación de transformación. Si logramos dar con una buena, podríamos tener un **transformador irreversible**. En criptografía esto es esencial, y hay mucha investigación para encontrar transformadores irreversibles.

Las funciones criptográficas más comunes, como AES (https://en.wikipedia.org/wiki/Advanced_Encryption_Standard), son transformadores que actúan sobre *secuencias de bits* (ceros y unos).

Dichas funciones son muy complicadas y de bajo nivel, y se salen por completo del alcance de este curso, pero la idea general y ala tienes.

Destrucción

Ninguna de estas funciones altera o destruye sus parámetros. Por ejemplo, la lista de entrada queda tal cual después de llamar a la función.

> Las funciones que alteran sus parámetros se llaman destructivas. Son más peligrosas que una piraña en un bidé

Nunca implementes funciones destructivas a no ser que sea absolutamente indispensable. Si lo haces, documéntalo muy bien.