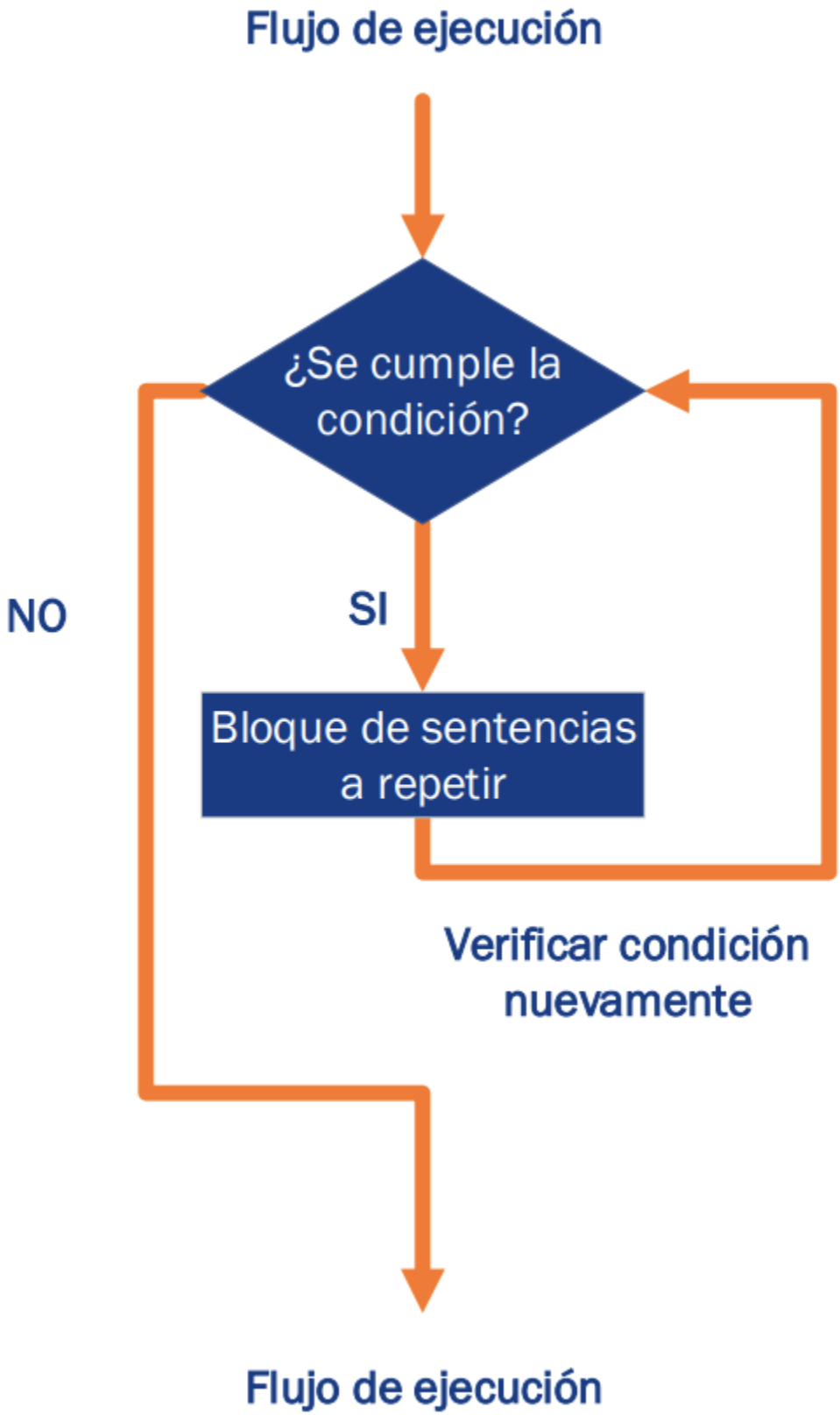
Guía 4: Iteraciones

Profesor: John Serrano - john.serrano@usach.cl

Ayudante: Vanina Correa - vanina.correa@usach.cl

1 Ciclo While

En palabras simples el concepto de "iteración" quiere referirse al concepto de **repetir algo**. En el caso de Python, podemos utilizar el ciclo While (Mientras) **para repetir una instrucción una cantidad de veces finita dependiendo de una condición que retorna un valor booleano.** Su funcionamiento es similar al de los condiciones (if, elif, else), en el sentido de que, si se cumple la condición (True) entonces entramos dentro del bloque While. Si la condición ya no se cumple (False) entonces salimos del ciclo While. La gran diferencia con los condicionales es que cada vez que ejecutemos las instrucciones del bloque While, vamos a volver a preguntar si se cumple la condición del While. En caso de que si, nuevamente se repite el proceso. Esto continua así hasta que la condición ya no se cumpla y se sale del ciclo While.


Flujo de ejecución
¿Se cumple la condición?
NO
SI
Bloque de sentencias a repetir
Verificar condición nuevamente
Flujo de ejecución

```
<Sentencias previas>
while <condición>:
    # Se ejecuta si la condición se cumple
    <Bloque de sentencias a repetir>
<Sentencias después del ciclo>
```

Supongamos que queremos hacer la suma de los números del 1 al 10:

- Podemos escribir las sumas necesarias para llegar al resultado:

[1]:
```
print("La suma de los primeros 10 números es:", 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 +␣
↪9 + 10)
```

```
La suma de los primeros 10 números es: 55
```

- Podemos utilizar la fórmula de la sumatoria de n números:

[2]:
```
n = 10
suma = n * (n + 1) / 2
print("La suma de los primeros", n, "numeros es:", suma)
```

```
La suma de los primeros 10 numeros es: 55.0
```

Pero sabemos que la primera solución es demasiado engorrosa para números grandes y la segunda requiere recordar una formula, la cual no nos sirve para todos los casos. **Entonces, ahora entra la tercera opción, que es construir un programa iterativo (que use un ciclo While):**

[3]:
```
# ENTRADA
numero = int(input("Ingrese un número entero positivo: "))

# PROCESAMIENTO
i = 0                   # Se crea un iterador (variable que se utilizará en la␣
↪condición de un ciclo While)
suma = 0                # Se crea una variable para guardar el resultado de la suma
while i <= numero:      # Mientras el valor de i (inicialmente 0) sea menor o␣
↪igual al valor de numero (entrada)
    suma = suma + i     # Se suma el valor de suma junto con el valor de i. Es␣
↪igual a escribir suma += i
    i = i + 1           # Se aumenta el iterador. Es importante hacer esto, de lo␣
↪contrario el While se quedará ejecutandose infinitas veces.

# SALIDA
print("La suma de los primeros", numero, "numeros es:", suma)
```

```
Ingrese un número entero positivo:  10

La suma de los primeros 10 numeros es: 55
```

Es importante considerar que la condición de un ciclo While NO debe ser una tautología, es decir, no debe ser siempre verdadera. En algún punto la condición debe dejar de ser falsa, o de lo contrario el programa se quedará atascado en un bucle infinito.

2 Ciclo For-in

El ciclo for-in nos permite realizar iteración donde una variable irá tomando distintos valores para cada ciclo. Es similar a While, pero existen diferencias notables a considerar. La sintaxis de For-in es la siguiente:

for **Identificador** in **Elemento Iterable:**

```
operaciones a realizar en el ciclo
```

De lo anterior:

- Un **elemento iterable** es algún dato que esté compuesto por varios elementos. Por ejemplo, los strings, archivos y las lista.
- El **identificador** es una variable que **está siendo definida**, la cual tomará un valor del elemento iterable por cada ciclo. Idealmente, debería tener un nombre representativo (buenas prácticas).

Supongamos que tenemos un texto (string) y queremos obtener cada elemento (letras o caracteres) del texto.

- Con ciclo While:

[4]:
```
texto = input("Ingrese un texto: ") # Se ingresa un texto cualquiera (string)
i = 0                               # Se define un iterador
while i < len(texto):               # Mientras el valor de i sea menor al largo␣
 ↪del texto
    print(texto[i])                 # Voy a acceder a la posición i de texto (i␣
 ↪= 0, 1, 2, 3, etc)
    i = i + 1                       # Se aumenta i en 1 para seguir accediendo a␣
 ↪las otras posiciones
```

```
Ingrese un texto:  Ustedes deberian estar leyendo esto para aprobar

U
s
t
e
d
e
s

d
e
```

b
e
r
i
a
n

e
s
t
a
r

l
e
y
e
n
d
o

e
s
t
o

p
a
r
a

a
p
r
o
b
a
r

- Con ciclo For-in:

[5]:
```
texto = input("Ingrese un texto: ") # Se ingresa un texto cualquiera (string)
for caracter in texto:              # Para cada caracter en el texto (NOTESE QUE␣
 ↪LA VARIABLE caracter SE DEFINE AHI MISMO)
    print(caracter)
```

```
Ingrese un texto:  Ustedes deberian estar leyendo esto para aprobar
U
```

s
t
e
d
e
s

d
e
b
e
r
i
a
n

e
s
t
a
r

l
e
y
e
n
d
o

e
s
t
o

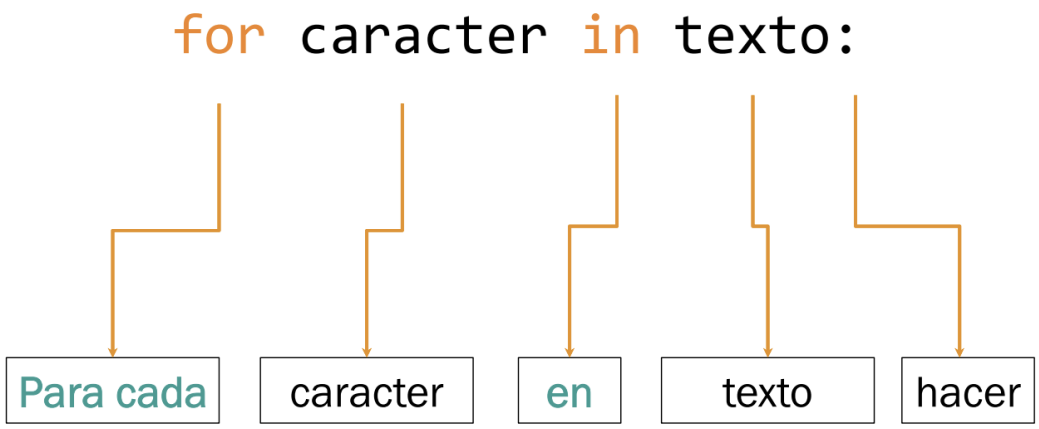
p
a
r
a

a
p
r
o
b
a
r

MUCHO CUIDADO!: La variable caracter tiene ese nombre ya que es representativo a los elementos de texto (un string está compuesto de caracteres), pero "caracter" es simplemente un nombre más. Podría ser cualquiera, pero NO cambia el funcionamiento del ciclo For-in. Es decir, si le coloco a esa variable el nombre "palabras":

[6]:
```
texto = input("Ingrese un texto: ") # Se ingresa un texto cualquiera (string)
for palabra in texto:               # Para cada palabra (elemento) en texto
    print(palabra)                  # Se imprime la variable caracter, la cual␣
 ↪toma el valor de cada caracter de texto
```

```
Ingrese un texto:  Ustedes van a aprobar la asignatura
U
s
t
e
d
e
s

v
a
n

a

a
p
r
o
b
a
r

l
a

a
s
i
g
n
a
t
u
r
a
```



Lo anterior significa que caracter tomará un valor que haya en texto y ejecutará el ciclo. El ciclo repetirá esta acción para cada valor que se encuentre en texto. Dar un nombre representativo al identificador ayuda la lectura del ciclo (texto está compuesto de caracteres). Una vez realizado esto, el ciclo termina y continúa el resto del código.

El tipo de dato en el que estoy iterando determina qué será el elemento a iterar, es decir:

- Si hago un for-in a una lista, el elemento a iterar serán los elementos que la compone.
- Si hago un for-in a un string, el elemento a iterar serán los caractereres.
- Si hago un for-in a un archivo, el elemento a iterar serán las líneas del archivos.

3 Función range()

La función range recibe tres atributos:

- Un valor int que representa el valor del inicio del intervalo que se quiere generar. Este valor es opcional y por defecto es 0.
- Un valor int que representa el valor (final - salto) del intervalo que se quiere generar.
- Un valor int que representa el salto del intervalo que se quiere generar. Este valor es opcional y por defecto es 1.

Por ejemplo, si colocamos solo el valor final al que queremos llegar:

[7]:
```
for numero in range(9):
    print(numero)
```

```
0
1
2
3
4
5
6
```

```
7
8
```

Se puede ver que se llegue hasta el valor final (9) - salto (1), lo que nos da 8, el valor de inicio es 0 y va de 1 en 1. Ahora, si colocamos los tres valores:

[8]:
```
for numero in range(5, 100, 5):
    print(numero)
```

```
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
```

El ejemplo anterior parte desde 5 y llega hasta 95, pues el salto es de 5 en 5 y el valor final colocado es 100. Dado que 100 - 5 = 95, entonces el intervalo llega hasta ese valor.

range() suele ser util para algunos ejercicio de listas y se suele combinar con el ciclo For-in.