Compresion de listas

La *comprensión de listas* en Python es un método sintáctico para crear listas de una forma rápida de escribir, muy legible y funcionalmente eficiente.

```
languages = ["python", "c", "c++", "java"]
```

Ejemplo 1:

```
cap_languages = []
for language in languages:
    cap_languages.append(language.capitalize())
```

```
cap_languages = [language.capitalize() for language in langua
```

```
languages = [language.capitalize() for language in languages]
```

Ejemplo 2:

```
numbers = [1, 2, 3, 4, 5]
```

```
doubled_numbers = [n * 2 for n in numbers]
```

Ejemplo 3:

```
multiples = []
for n in range(1, 101):
    if n % 5 == 0:
        multiples.append(n)
```

```
multiples = [n for n in range(1, 101) if n % 5 == 0]
```

Podriamos resumir:

💡 [expresion **for** variable **in** colección **if** condición]

A través de la comprensión de listas también podemos expresar de forma compacta un conjunto de bucles anidados. Por ejemplo, el siguiente código crea una lista `points` que contiene (en forma de tuplas de dos elementos) la posición de todos los puntos bidimencionales entre las coordenadas `(0, 0)` y `(5, 10)`.

Ejemplo 4:

```
points = []
for x in range(0, 5 + 1):
    for y in range(0, 10 + 1):
        points.append((x, y))
print(points)
```

```
points = [(x, y) for x in range(0, 5 + 1) for y in range(0, 1
print(points)
```

Podríamos incluso agregar una condición usando ambas variables, por ejemplo, para mostrar solo los puntos en los que `x` es igual a `y`.

```
points = [(x, y) for x in range(0, 5 + 1) for y in range(0, 1
          if x == y]
```

Diccionarios

podemos crear un diccionario de la misma forma, pero en este caso utilizamos llaves en lugar de corchetes.

```
doubles = {n: n * 2 for n in range(1, 11)}
```

💡 {clave: valor for variable in coleccion if condicion}

Sets

```
doubles = {n * 2 for n in range(1, 11)}
```