

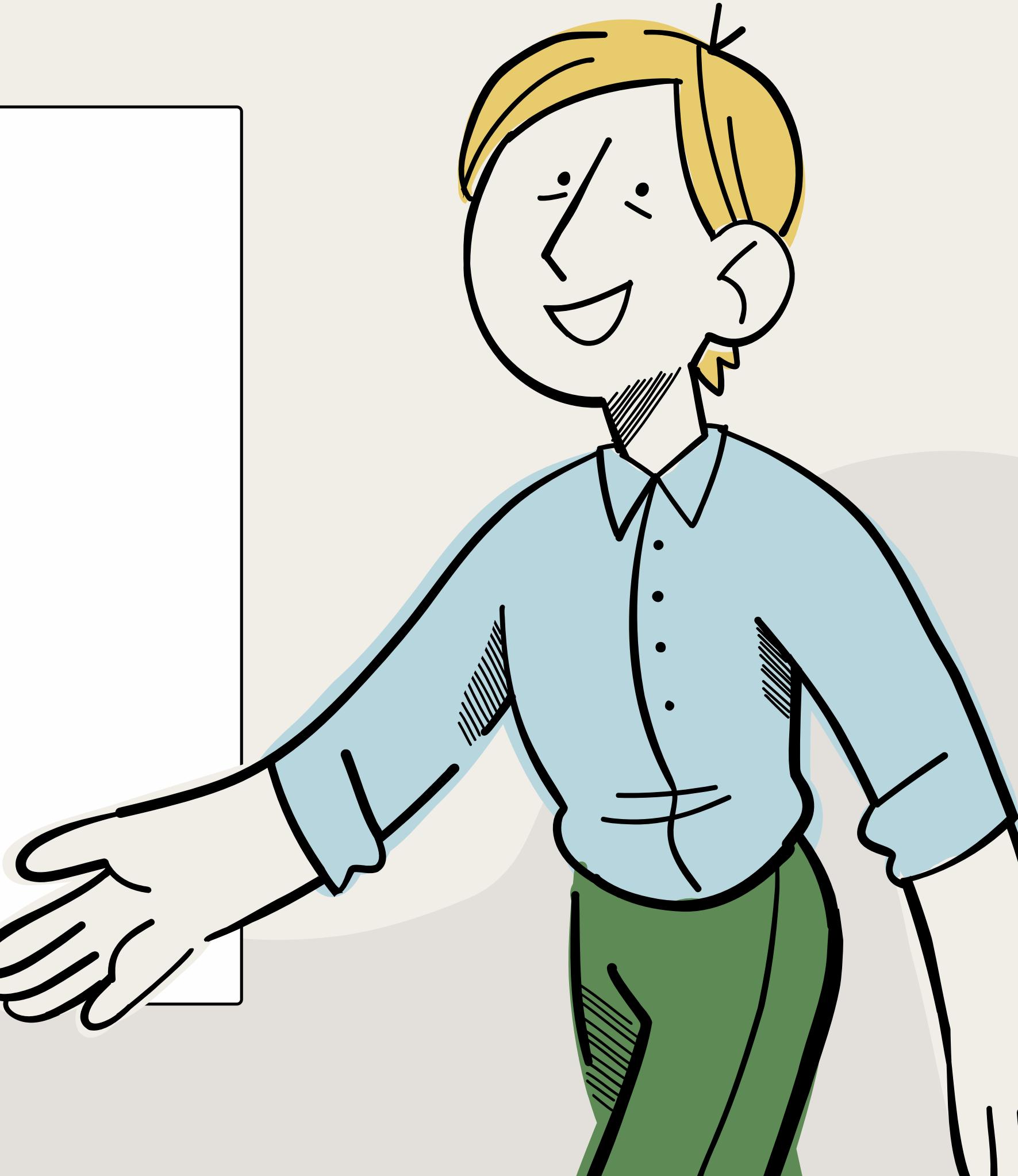
MINI PROYECTO

ALGORITMOS Y

PROGRAMACIÓN

GRUPO 2

DIANA ALEJANDRA ROSERO ROSAS



Qué es una lista en Phyton



Las listas permiten almacenar objetos mediante un orden definido y con posibilidad de duplicados. Las listas son estructuras de datos mutables, lo que significa que podemos añadir, eliminar o modificar sus elementos.



Operaciones con listas



Obtener un elemento igual que en el caso de las cadenas de texto, podemos obtener un elemento de una lista a través del índice (lugar) que ocupa.

```
>>> shopping = ['Agua', 'Huevos', 'Aceite']

>>> shopping[0]
'Agua'

>>> shopping[1]
'Huevos'

>>> shopping[2]
'Aceite'

>>> shopping[-1] # acceso con índice negativo
'Aceite'
```

Trocear una lista

El troceado de listas funciona de manera totalmente análoga al troceado de cadenas.



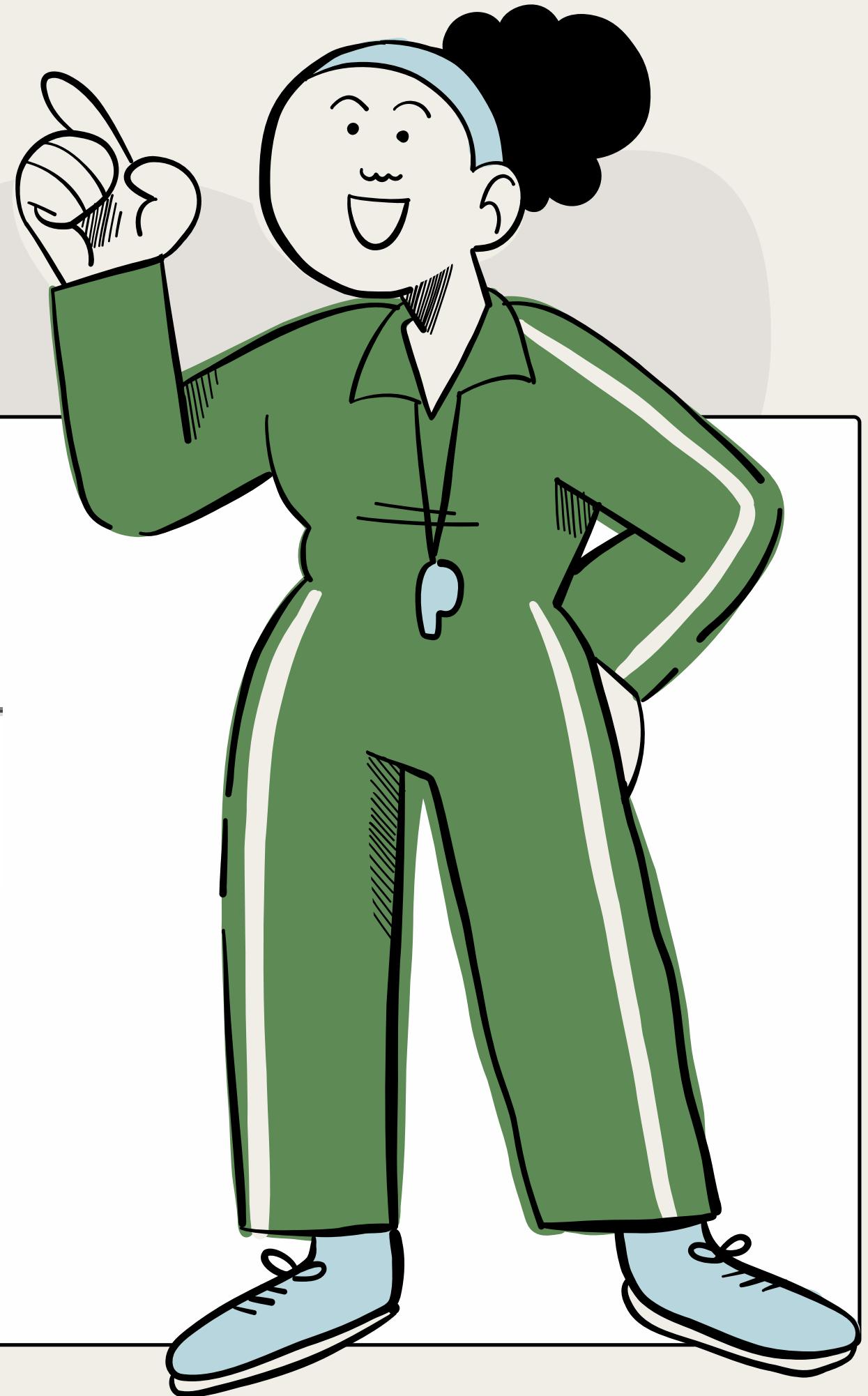
```
>>> shopping = ['Agua', 'Huevos', 'Aceite', 'Sal', 'Limón']  
>>> shopping[0:3]
```

```
['Agua', 'Huevos', 'Aceite']  
  
>>> shopping[:3]  
['Agua', 'Huevos', 'Aceite']  
  
>>> shopping[2:4]  
['Aceite', 'Sal']  
  
>>> shopping[-1:-4:-1]  
['Limón', 'Sal', 'Aceite']  
  
>>> # Equivale a invertir la lista  
>>> shopping[::-1]  
['Limón', 'Sal', 'Aceite', 'Huevos', 'Agua']
```

Invertir una lista

Python nos ofrece, al menos, tres mecanismos para invertir los elementos de una lista:

```
>>> shopping
['Agua', 'Huevos', 'Aceite', 'Sal', 'Limón']
```



Añadir al final de la lista

Una de las operaciones más utilizadas en listas es añadir elementos al final de las mismas. Para ello Python nos ofrece la función `append()`.

```
>>> shopping = ['Agua', 'Huevos', 'Aceite']

>>> shopping.append('Atún')

>>> shopping
['Agua', 'Huevos', 'Aceite', 'Atún']
```



Creando desde vacío

Una forma muy habitual de trabajar con listas es empezar con una vacía e ir añadiendo elementos poco a poco. Se podría hablar de un patrón creación.



Añadir en cualquier posición de una lista

Ya hemos visto cómo añadir elementos al final de una lista. Sin embargo, Python ofrece una función `insert()` que vendría a ser una generalización de la anterior, para incorporar elementos en cualquier posición. Simplemente debemos especificar el índice de inserción y el elemento en cuestión

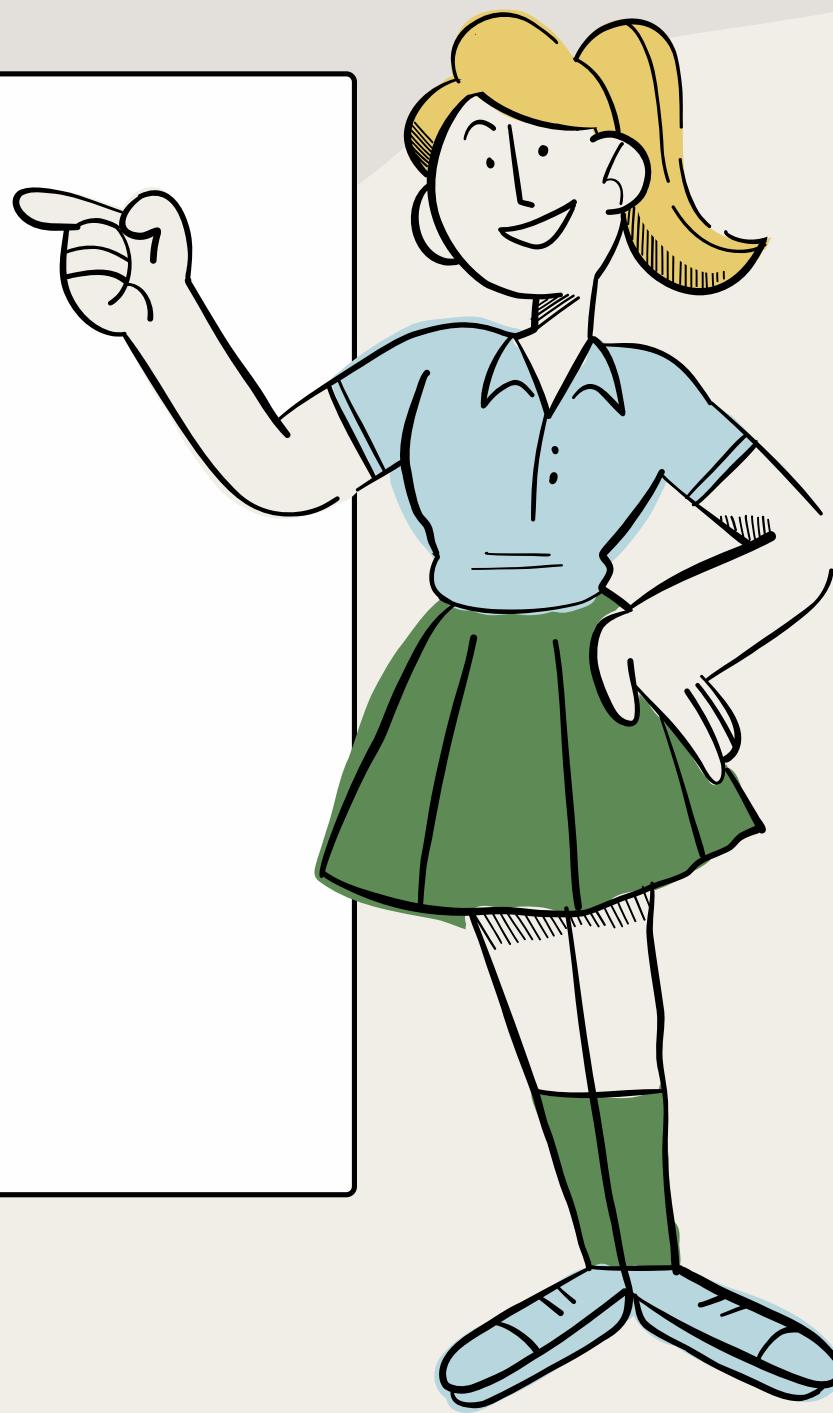
```
>>> shopping = ['Agua', 'Huevos', 'Aceite']

>>> shopping.insert(1, 'Jamón')

>>> shopping
['Agua', 'Jamón', 'Huevos', 'Aceite']

>>> shopping.insert(3, 'Queso')

>>> shopping
['Agua', 'Jamón', 'Huevos', 'Queso', 'Aceite']
```



Repetir elementos

Al igual que con las cadenas de texto, el operador * nos permite repetir los elementos de una Lista



```
>>> shopping = ['Agua', 'Huevos', 'Aceite']
```

```
>>> shopping * 3
```

```
['Agua',  
 'Huevos',  
 'Aceite',
```

Combinar listas

Python nos ofrece dos aproximaciones para combinar listas:



```
>>> shopping = ['Agua', 'Huevos', 'Aceite']
```

```
>>> fruitshop = ['Naranja', 'Manzana', 'Piña']
```

```
>>> shopping + fruitshop
```

```
['Agua', 'Huevos', 'Aceite', 'Naranja', 'Manzana', 'Piña']
```

Modificar una lista

Del mismo modo que se accede a un elemento utilizando su índice, también podemos modificarlo:

```
>>> shopping = ['Agua', 'Huevos', 'Aceite']

>>> shopping[0]
'Agua'

>>> shopping[0] = 'Jugo'

>>> shopping
['Jugo', 'Huevos', 'Aceite']
```



Modificar con troceado

No sólo es posible modificar un elemento de cada vez, sino que podemos asignar valores a trozos de una lista

```
>>> shopping = ['Agua', 'Huevos', 'Aceite', 'Sal', 'Limón']

>>> shopping[1:4]
['Huevos', 'Aceite', 'Sal']

>>> shopping[1:4] = ['Atún', 'Pasta']
```



BORRAR ELEMENTOS



Borrar elementos

Python nos ofrece, al menos, cuatro formas para borrar elementos en una lista:

```
>>> shopping = ['Agua', 'Huevos', 'Aceite', 'Sal', 'Limón']
>>> del(shopping[3])
>>> shopping
['Agua', 'Huevos', 'Aceite', 'Limón']
```

Borrado completo de la lista

Python nos ofrece, al menos, dos formas para borrar una lista por completo:

```
>>> shopping = ['Agua', 'Huevos', 'Aceite', 'Sal', 'Limón']
>>> shopping.clear() # Borrado in-situ
>>> shopping
[]
```

Encontrar un elemento

Si queremos descubrir el índice que corresponde a un determinado valor dentro la lista podemos usar la función index() para ello

```
>>> shopping = ['Agua', 'Huevos', 'Aceite', 'Sal', 'Limón']  
  
>>> shopping.index('Huevos')  
1
```

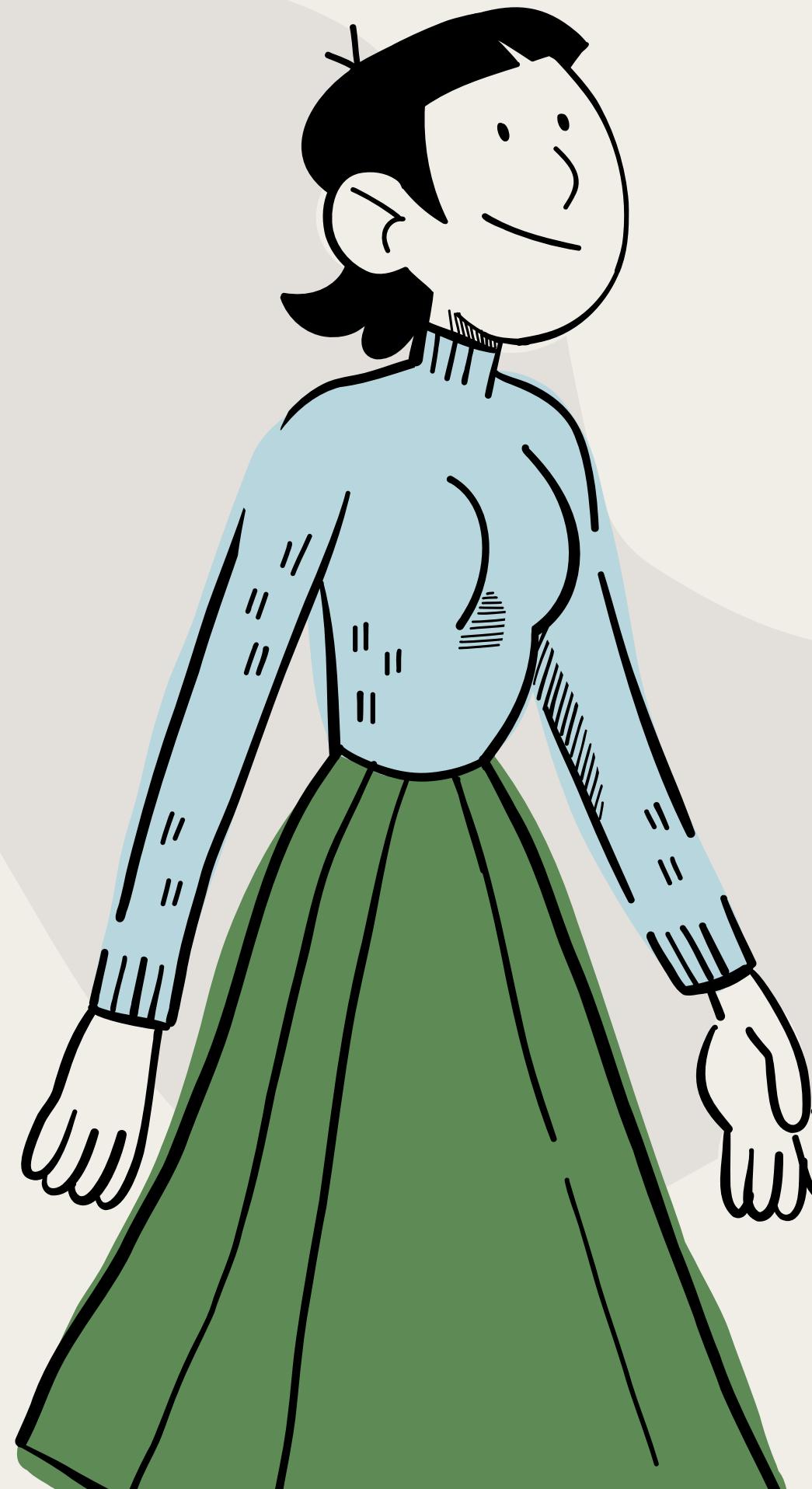


Pertenencia de un elemento

Si queremos comprobar la existencia de un determinado elemento en una lista, podríamos buscar su índice, pero la forma pitónica de hacerlo es utilizar el operador in:

```
>>> shopping = ['Agua', 'Huevos', 'Aceite', 'Sal', 'Limón']  
  
>>> 'Aceite' in shopping  
True  
  
>>> 'Pollo' in shopping  
False
```





Número de ocurrencias

Para contar cuántas veces aparece un determinado valor dentro de una lista podemos usar la función `count()`:

```
> sheldon_greeting = ['Penny', 'Penny', 'Penny']
```

```
> sheldon_greeting.count('Howard')
```

```
>>> sheldon_greeting.count('Penny')
```

```
3
```

Convertir lista a cadena de texto
Dada una lista, podemos convertirla
a una cadena de texto, uniendo
todos sus elementos
mediante algún separador. Para ello
hacemos uso de la función join()
con la siguiente
estructura:



Figura 1: Estructura de llamada a la función join()

```
>>> shopping = ['Agua', 'Huevos', 'Aceite', 'Sal', 'Limón']

>>> ', '.join(shopping)
'Agua,Huevos,Aceite,Sal,Limón'

>>> ' '.join(shopping)
'Agua Huevos Aceite Sal Limón'

>>> '|'.join(shopping)
'Agua|Huevos|Aceite|Sal|Limón'
```

Ordenar una lista

Python proporciona, al menos, dos formas de ordenar los elementos de una lista



```
>>> shopping = ['Agua', 'Huevos', 'Aceite', 'Sal', 'Limón']

>>> sorted(shopping)
['Aceite', 'Agua', 'Huevos', 'Limón', 'Sal']
```

Longitud de una lista

Podemos conocer el número de elementos que tiene una lista con la función len()



```
>>> shopping = ['Agua', 'Huevos', 'Aceite', 'Sal', 'Limón']

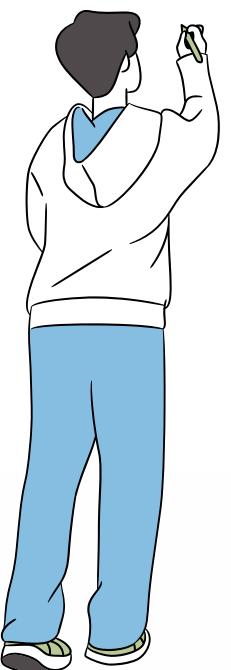
>>> len(shopping)
5
```

Iterar sobre una lista

Al igual que hemos visto con las cadenas de texto, también podemos iterar sobre los elementos de una lista utilizando la sentencia for

```
>>> shopping = ['Agua', 'Huevos', 'Aceite', 'Sal', 'Limón']

>>> for i, product in enumerate(shopping):
...     print(i, product)
...
0 Agua
1 Huevos
2 Aceite
```



```
>>> shopping = ['Agua', 'Aceite', 'Arroz']
>>> details = ['mineral natural', 'de oliva virgen', 'basmati']

>>> for product, detail in zip(shopping, details):
...     print(product, detail)
...
Agua mineral natural
Aceite de oliva virgen
Arroz basmati
```

Iterar usando enumeración

Hay veces que no sólo nos interesa «visitar» cada uno de los elementos de una lista, sino que también queremos saber su índice dentro de la misma. Para ello Python nos ofrece la función enumerate():

```
>>> shopping = ['Agua', 'Huevos', 'Aceite', 'Sal', 'Limón']

>>> for i, product in enumerate(shopping):
...     print(i, product)
...
0 Agua
1 Huevos
2 Aceite
```



Iterar sobre múltiples listas

Python ofrece la posibilidad de iterar sobre múltiples listas en paralelo utilizando la función zip():



Cadena de caracteres

Las cadenas de texto son secuencias de caracteres. También se les conoce como «strings»

y nos permiten almacenar información textual de forma muy cómoda.¹

Es importante destacar que Python 3 almacena los caracteres codificados en el estándar

Unicode, lo que es una gran ventaja con respecto a versiones antiguas del lenguaje.

Además

permite representar una cantidad ingente de símbolos incluyendo los famosos emojis

Operaciones string

Combinar cadenas

Podemos combinar dos o más cadenas de texto utilizando el operador `+`:



```
>>> proverb1 = 'Cuando el río suena'  
>>> proverb2 = 'agua lleva'  
  
>>> proverb1 + proverb2  
'Cuando el río suenaagua lleva'  
  
>>> proverb1 + ', ' + proverb2 # incluimos una coma  
'Cuando el río suena, agua lleva'
```

Repetir cadenas

Podemos repetir dos o más cadenas de texto utilizando el operador `*`:

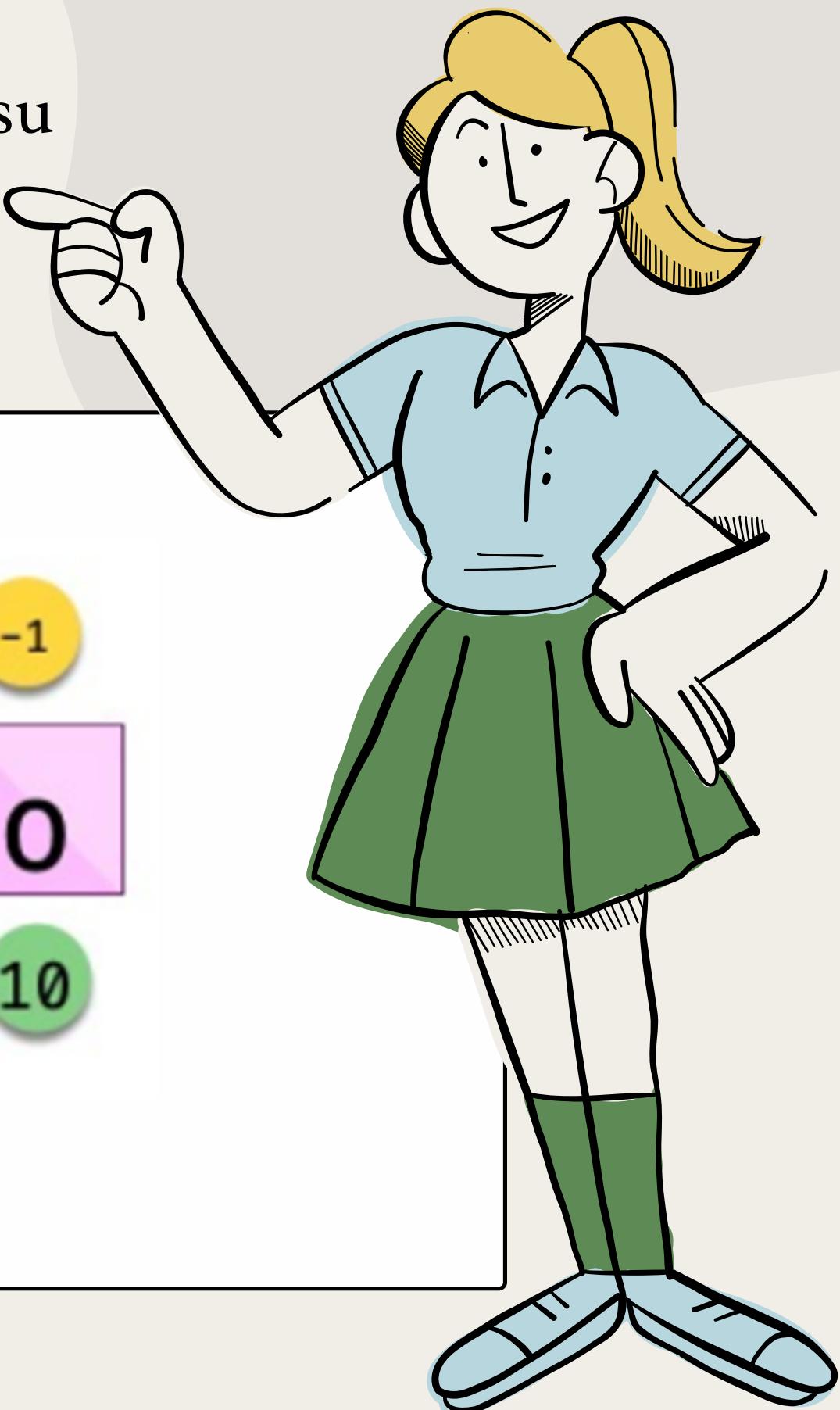
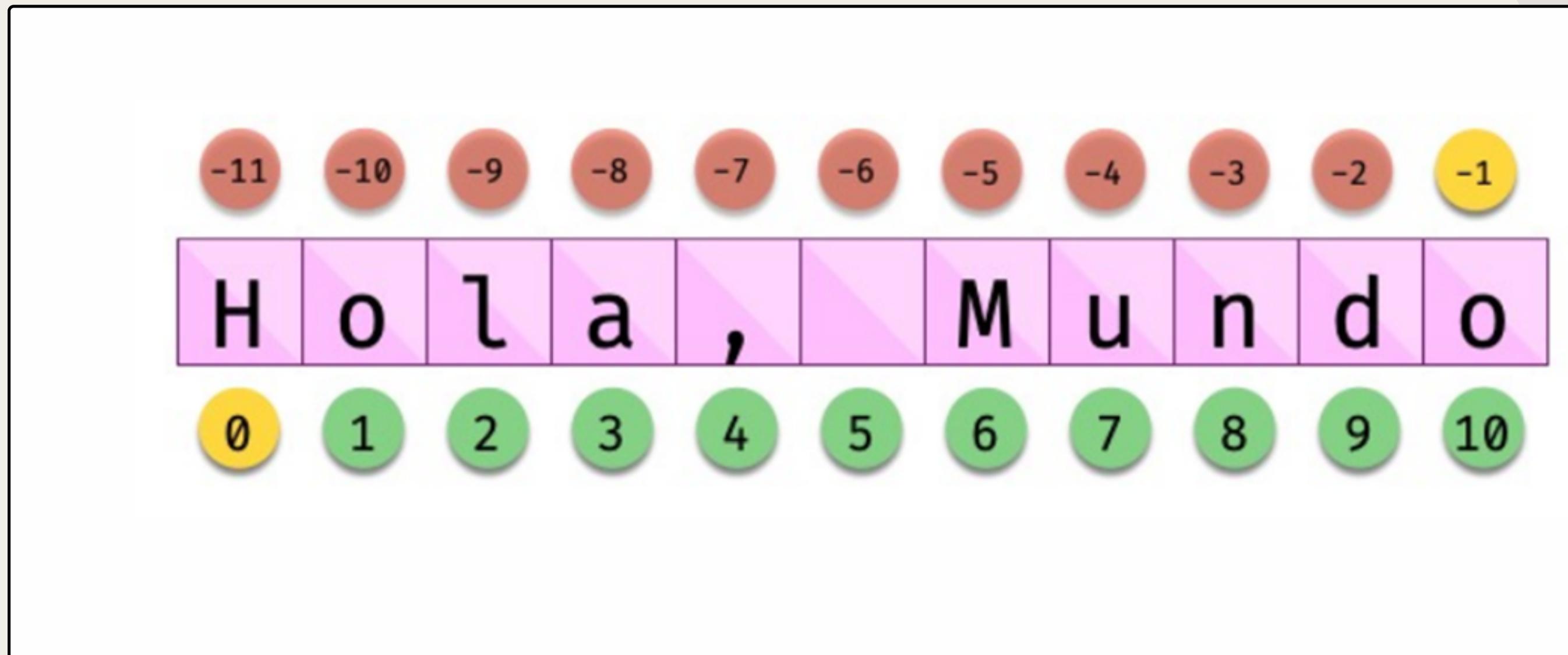


```
>>> reaction = 'Wow'  
  
>>> reaction * 4  
'WowWowWowWow'
```

Obtener un carácter

Los «strings» están indexados y cada carácter tiene su propia posición.

Para obtener un único carácter dentro de una cadena de texto es necesario especificar su índice dentro de corchetes [...]



TROCEAR UNA CADENA

ES POSIBLE EXTRAER «TROZOS» («REBANADAS») DE UNA CADENA DE TEXTO₂

. TENEMOS VARIAS

APROXIMACIONES PARA ELLO:

[:] EXTRAЕ LA SECUENCIA ENTERA DESDE EL COMIENZO HASTA EL FINAL. ES UNA
ESPECIA DE COPIA DE
TODA LA CADENA DE TEXTO.

[START:] EXTRAЕ DESDE START HASTA EL FINAL DE LA CADENA.

[:END] EXTRAЕ DESDE EL COMIENZO DE LA CADENA HASTA END MENOS 1.

[START:END] EXTRAЕ DESDE START HASTA END MENOS 1.

[START:END:STEP] EXTRAЕ DESDE START HASTA END MENOS 1 HACIENDO SALTOS DE
TAMAÑO STEP.



```
>>> proverb = 'Agua pasada no mueve molino'
```

```
>>> proverb[:]
```

```
'Agua pasada no mueve molino'
```

```
>>> proverb[12:]
```

```
'no mueve molino'
```

```
>>> proverb[:11]
```

```
'Agua pasada'
```

```
>>> proverb[5:11]
```

```
'pasada'
```

```
>>> proverb[5:11:2]
```

```
'psd'
```

Longitud de una cadena

Para obtener la longitud de una cadena podemos hacer uso de `len()`, una función común a prácticamente todos los tipos y estructuras de datos en Python:

```
>>> proverb = 'Lo cortés no quita lo valiente'  
>>> len(proverb)  
27  
  
>>> empty = ''  
>>> len(empty)  
0
```



Pertenencia de un elemento

Si queremos comprobar que una determinada subcadena se encuentra en una cadena de texto utilizamos el operador `in` para ello. Se trata de una expresión que tiene como resultado un valor «booleano» verdadero o falso:

```
>>> proverb = 'Más vale malo conocido que bueno por conocer'  
  
>>> 'malo' in proverb  
True  
  
>>> 'bueno' in proverb  
True  
  
>>> 'regular' in proverb  
False
```



Dividir una cadena

Una tarea muy común al trabajar con cadenas de texto es dividirlas por algún tipo de separador. En este sentido, Python nos ofrece la función `split()`, que debemos usar anteponiendo el «string» que queramos dividir

```
>>> proverb = 'No hay mal que por bien no venga'  
>>> proverb.split()  
['No', 'hay', 'mal', 'que', 'por', 'bien', 'no', 'venga']  
  
>>> tools = 'Martillo,Sierra,Destornillador'  
>>> tools.split(',')  
['Martillo', 'Sierra', 'Destornillador']
```



Limpiar cadenas

Cuando leemos datos del usuario o de cualquier fuente externa de información, es bastante probable que se incluyan en esas cadenas de texto, caracteres de relleno3 al comienzo y al final. Python nos ofrece la posibilidad de eliminar estos caracteres u otros que no nos interesen.

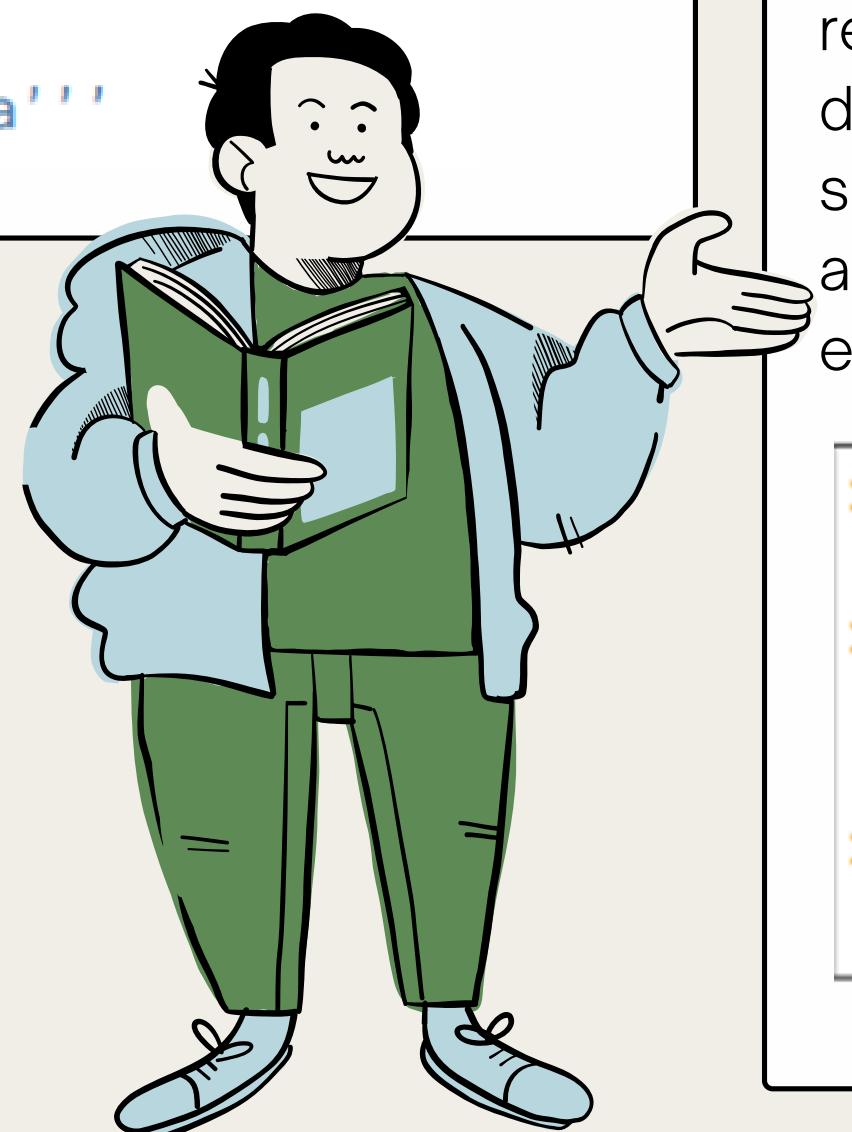
La función `strip()` se utiliza para eliminar caracteres del principio y del final de un «string». También existen variantes de esta función para aplicarla únicamente al comienzo o únicamente al final de la cadena de texto.

```
>>> serial_number = '\n\t \n 48374983274832 \n\n\t \t \n'  
  
>>> serial_number.strip()  
'48374983274832'
```

Realizar búsquedas

Aunque hemos visto que la forma pitónica de saber si una subcadena se encuentra dentro de otra es a través del operador `in`, Python nos ofrece distintas alternativas para realizar búsquedas en cadenas de texto

```
>>> lyrics = '''Quizás porque mi niñez  
... Sigue jugando en tu playa  
... Y escondido tras las cañas  
... Duerme mi primer amor  
... Llevo tu luz y tu olor  
... Por dondequiera que vaya'''
```



Reemplazar elementos

Podemos usar la función `replace()` indicando la subcadena a reemplazar, la subcadena de reemplazo y cuántas instancias se deben reemplazar. Si no se especifica este último argumento, la sustitución se hará en todas las instancias encontradas:

```
>>> proverb = 'Quien mal anda mal acaba'  
  
>>> proverb.replace('mal', 'bien')  
'Quien bien anda bien acaba'  
  
>>> proverb.replace('mal', 'bien', 1) # sólo 1 reemplazo  
'Quien bien anda mal acaba'
```

Mayúsculas y minúsculas

Python nos permite realizar variaciones en los caracteres de una cadena de texto para pasarlos a mayúsculas y/o minúsculas.

Veamos las distintas opciones disponibles

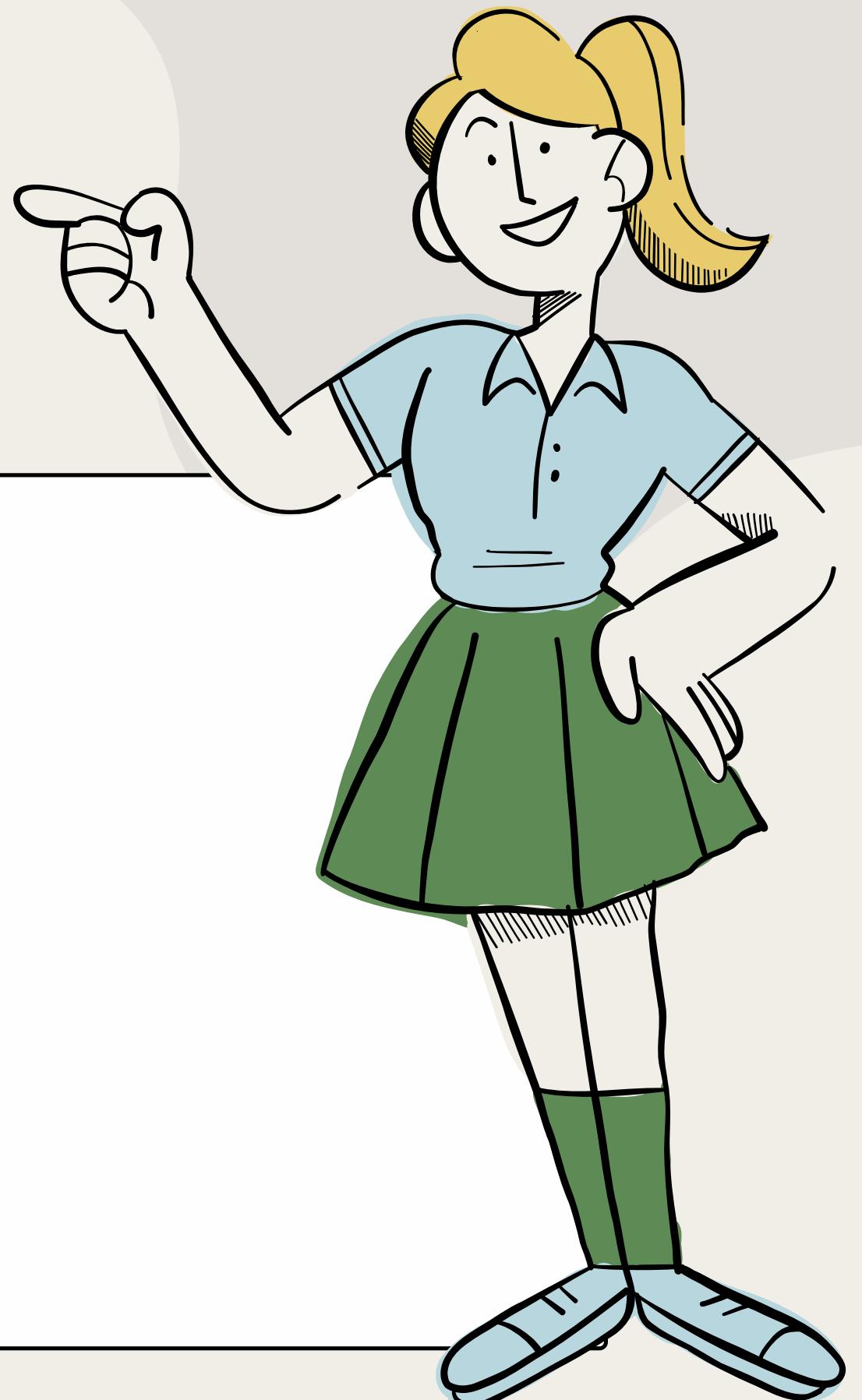


```
>>> proverb = 'quien a buen árbol se arrima Buena Sombra le cobija'  
  
>>> proverb  
'quien a buen árbol se arrima Buena Sombra le cobija'  
  
>>> proverb.capitalize()  
'Quien a buen árbol se arrima buena sombra le cobija'  
  
>>> proverb.title()  
'Quien A Buen Árbol Se Arrima Buena Sombra Le Cobija'  
  
>>> proverb.upper()  
'QUIEN A BUEN ÁRBOL SE ARRIMA BUENA SOMBRA LE COBIJA'  
  
>>> proverb.lower()  
'quien a buen árbol se arrima buena sombra le cobija'  
  
>>> proverb.swapcase()  
'QUIEN A BUEN ÁRBOL SE ARRIMA bUENA sOMBRA LE COBIJA'
```

Identificando caracteres

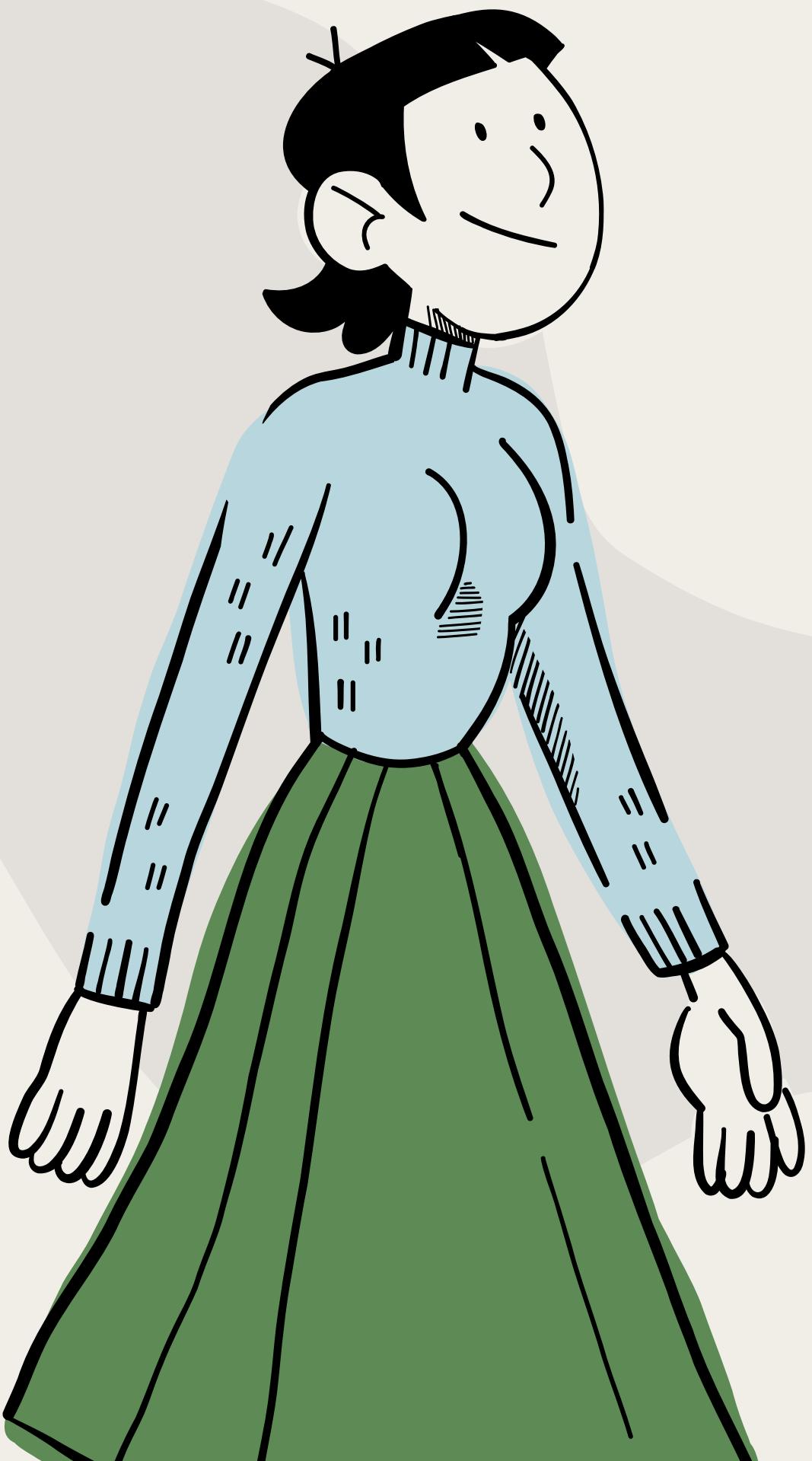
Hay veces que recibimos información textual de distintas fuentes de las que necesitamos identificar qué tipo de caracteres contienen. Para ello Python nos ofrece un grupo de funciones.

```
>>> 'R2D2'.isalnum()
True
>>> 'C3-PO'.isalnum()
False
```



¡Muchas gracias!





BIBLIOGRAFÍA

Listas en Python. (s. f.). El Libro De Python.
Recuperado 24 de octubre de 2022, de
<https://ellibrodepython.com/listas-en-python>

Cadenas en Python. (s. f.). El Libro De Python.
Recuperado 24 de octubre de 2022, de
<https://ellibrodepython.com/cadenas-python>