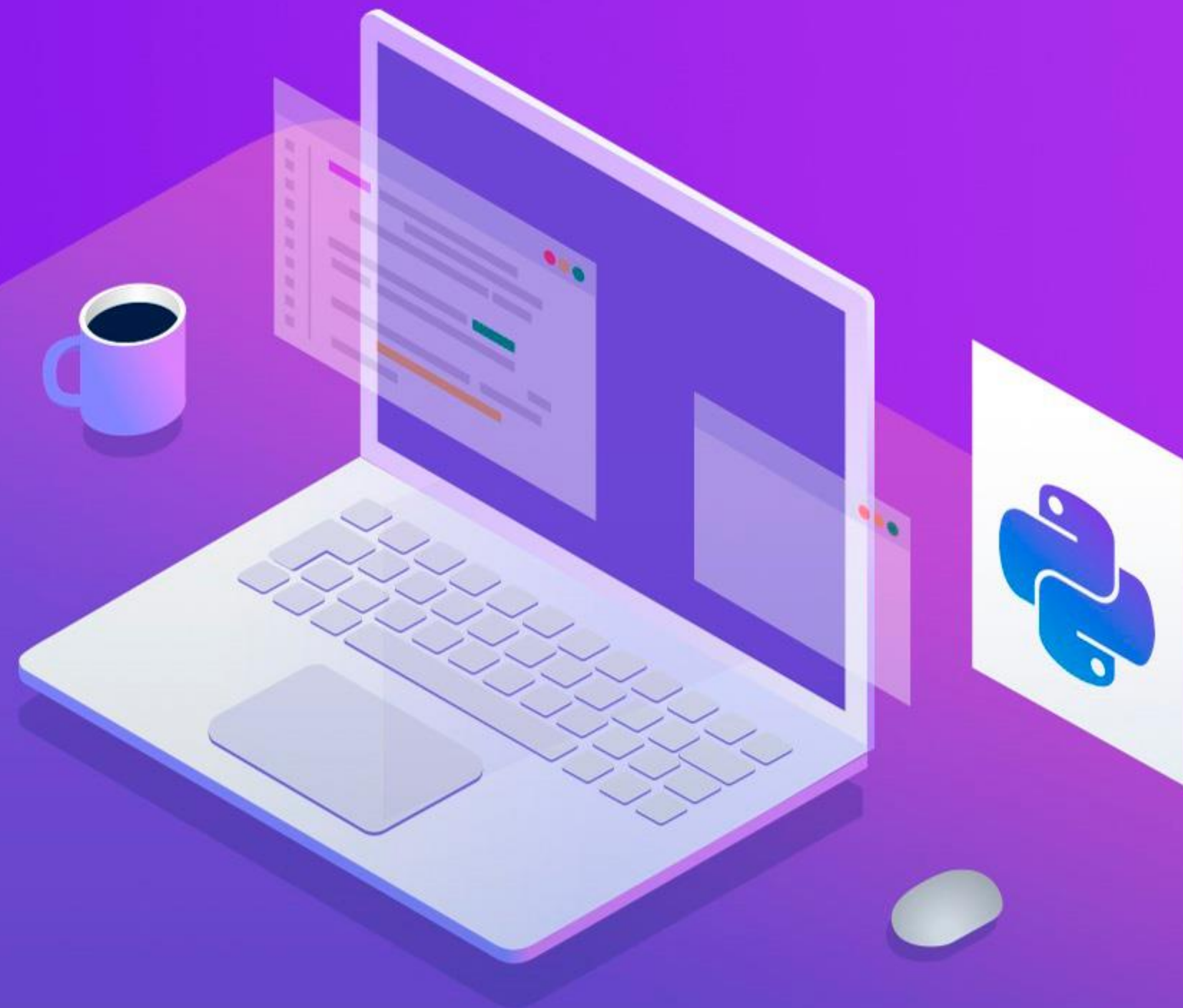




PONTIFICIA  
UNIVERSIDAD  
CATÓLICA  
DE CHILE

# **FUNCIONES DE STRING Y CREACIÓN DE LISTAS**

>>> Parte 1: Strings



# Recordemos ¿qué es un `string`?

Procesamiento de datos consiste: ser capaz de procesar texto.



La información puede obtenerse en **formato de texto**.



En las empresas manejan **muchos datos**. Es fundamental leer y editar textos.



## ¿Qué es un string?

En Python, las variables que tienen como tipo de dato texto se denominan strings.

CÓDIGO	RESULTADO
<pre>variable_texto="este es un texto de ejemplo" print(variable_texto)</pre>	<pre>este es un texto de ejemplo</pre>

# ¿Cómo podemos operar con variables de texto `strings`?

**Concatenación:** Para poder concatenar dos `strings` (unir), se puede ocupar el operador `+`.

Por ejemplo:

CÓDIGO	RESULTADO
<pre>variable_de_texto1 = "Podemos unir" variable de texto2 = "dos  conca varia variable de texto2 print(concatenacion)</pre>	<pre>Podemos unirdos textos distintos</pre>

El resultado de la concatenación es "Podemos unirdos textos distintos". Notemos que la concatenación no agrega ningún espacio o algo similar entre dos `strings` distintos.

# ¿Cómo podemos operar con variables de texto strings?

Recuerda revisar la Ruta de ejercicios.  
**Ejercicio EM2-14** →

**Concatenación:** No solo se pueden unir dos `strings` distintos, sino que más de dos.

Por ejemplo:

CÓDIGO	RESULTADO
<pre>variable_de_texto1 = "Podemos unir" variable_de_texto2 =</pre>	<pre>Podemos unir dos textos distintos</pre>
En este ejemplo, unimos tres <code>strings</code> distintos con el operador +	
<pre>concatenacion = variable_de_texto1 + " " + variable_de_texto2 print(concatenacion)</pre>	

# strings

Si quisiéramos saber **¿cuál es la primera letra de un** `string`?

¿Cómo podríamos hacerlo?

En realidad, un `string` es una secuencia de caracteres.

Por ejemplo, el `string` "hola" en realidad es una concatenación de los caracteres "h","o","l","a".

# strings

No solo es la concatenación de estos caracteres, sino que además cada uno de ellos tiene una posición en específico.

Carácter	→	h	o	l	a
Posición	→	0	1	2	3

Notemos que los caracteres se empiezan a contar desde el 0, y no desde el 1. Esto es sumamente importante, ya que al programar en general uno empieza a contar desde el 0 y no desde el 1.

Por lo tanto, para saber cuál es la primera letra de este `string`, necesitamos saber qué letra está en la posición 0.

# strings

Para hacer esto en Python, veamos el siguiente ejemplo:

CÓDIGO	RESULTADO
<pre>s = "hola" print(s[0])</pre>	<pre>h</pre>

En este caso, nuestra variable `string` que tiene el texto "hola" se llama `s`. Podemos extraer la primera letra en la posición 0 ocupando `[0]`, y así para cualquier posición.

Notemos que para extraer la letra de cualquier posición, ocupamos la siguiente notación:

```
nombre_variable_texto[posición]
```



## Revisemos un caso

Queremos juntar la cuarta y segunda letra de dos textos distintos e imprimirlas en la consola.

Las palabras son:

“Los datos”

“Móvil”



## Revisemos un ejemplo

CÓDIGO	RESULTADO
<pre>texto1 = "Los datos" texto2 = "Móvil"  cuarta_letra_texto1 = texto1[3] segunda_letra_texto2 = texto2[1]  print(cuarta_letra_texto1 + segunda_letra_texto2)</pre>	<pre>ó</pre>

Notemos que el cuarto carácter del texto (letra) "Los datos" es un espacio, y que el segundo carácter (letra) del texto "Móvil" es una o con tilde. Por lo tanto, al juntar ambos textos queda " ó". Los espacios, así como letras con tildes, signos de puntuación y otros similares **también se cuentan como caracteres dentro de un texto.**

# Strings

También se puede resolver de esta manera:

CÓDIGO	RESULTADO
<pre>texto1 = "Los datos" texto2 = "Móvil"  print(texto1[3] + texto2[1])</pre>	ó

## Notemos dos cosas:

1. No es necesario asignar las letras a una variable para poder ocuparlas posteriormente. Se pueden ocupar directamente en alguna operación.
2. Podemos ocupar el operador + para unir ambas letras. Esto quiere decir que para Python, una letra, o un carácter, **también se considera como texto**.

# Revisemos un caso

---

Supongamos que, queremos crear un programa que tome el texto ingresado por un usuario e imprima la última letra.

El problema es que no sabemos a priori cuál es el largo del texto que ingresará el usuario (porque puede ingresar el texto que él desee), por lo tanto no sabemos cuál es la posición de la última letra.



Ingrese un texto

# Función `len()`

La función `len()` nos indica cuál es la cantidad de caracteres que contiene un texto.



Para poder saber el largo de cualquier variable de tipo `string`, ocuparemos la función `len`.



Ésta no pertenece a ningún paquete ni tampoco hay que definirla, sino que Python ya la tiene incorporada.

# Función `len()`

La función `len` se ocupa de la siguiente manera:

```
s="Texto"  
largo_del_texto = len(s)
```

Donde la variable  
`largo_del_texto`  
tendrá el valor entero 5.

Podemos ocupar la función incluyendo cualquier variable de tipo `string` dentro del paréntesis y el valor de retorno será la cantidad de caracteres de ese `string`.

## Función `len()`

Veamos un ejemplo:

CÓDIGO	RESULTADO
<pre>string_de_ejemplo = "Hola" print(len(string_de_ejemplo))</pre>	<pre>4 15</pre>
<pre>string_de_ejemplo2 = "Hola cómo estás" print(len(string_de_ejemplo2))</pre>	

La variable `string_de_ejemplo` tiene el texto "Hola" que tiene 4 caracteres, por eso se imprime el 4 en la consola.

La variable `string_de_ejemplo2` tiene el texto "Hola cómo estás" que tiene 15 caracteres, por eso se imprime un 15 en la consola.

# Retomemos el ejercicio propuesto anteriormente

Recuerda revisar la  
Ruta de ejercicios.

**Ejercicio EM2-19** →



The screenshot shows the Trinket Python IDE interface. The top bar includes the Trinket logo, a 'Run' button, and a 'Share' button. The file name 'main.py' is displayed in the top left of the editor. The code in the editor is as follows:

```
1 texto_ingresado_por_el_usuario = input("Ingrese un texto cualquier. Python imprimirá el último caracter de este texto\n")
largo_texto = len(texto_ingresado_por_el_usuario)
ultimo_caracter = texto_ingresado_por_el_usuario[largo_texto-1]
print(ultimo_caracter)
```

Below the code editor, the output of the program is shown in a light blue box:

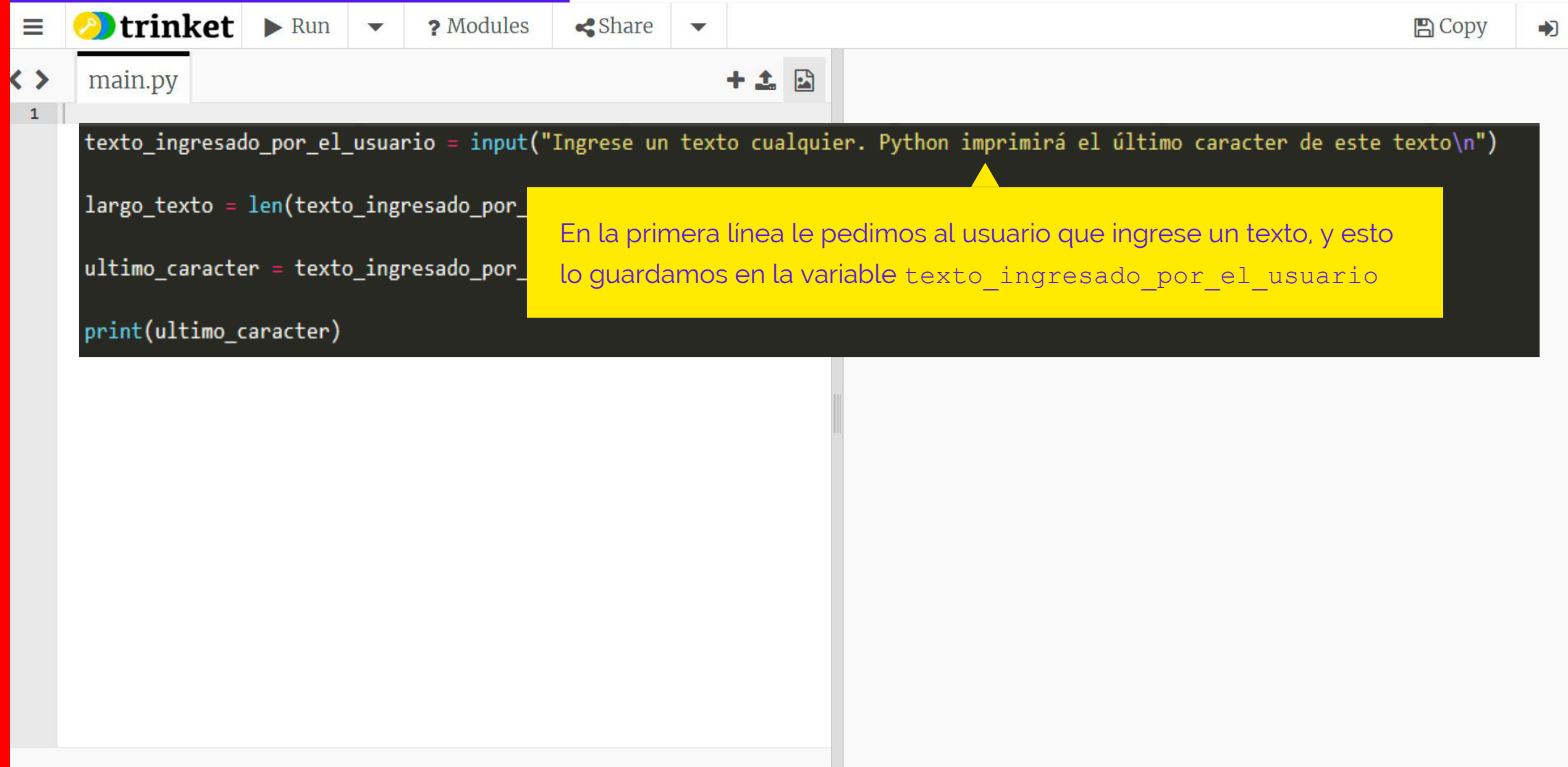
```
Ingrese un texto cualquier. Python imprimirá el último caracter de este texto
palabra
a
```

At the bottom of the image, a dark blue box contains the text:

Una solución posible sería la siguiente.



# Analicemos la respuesta



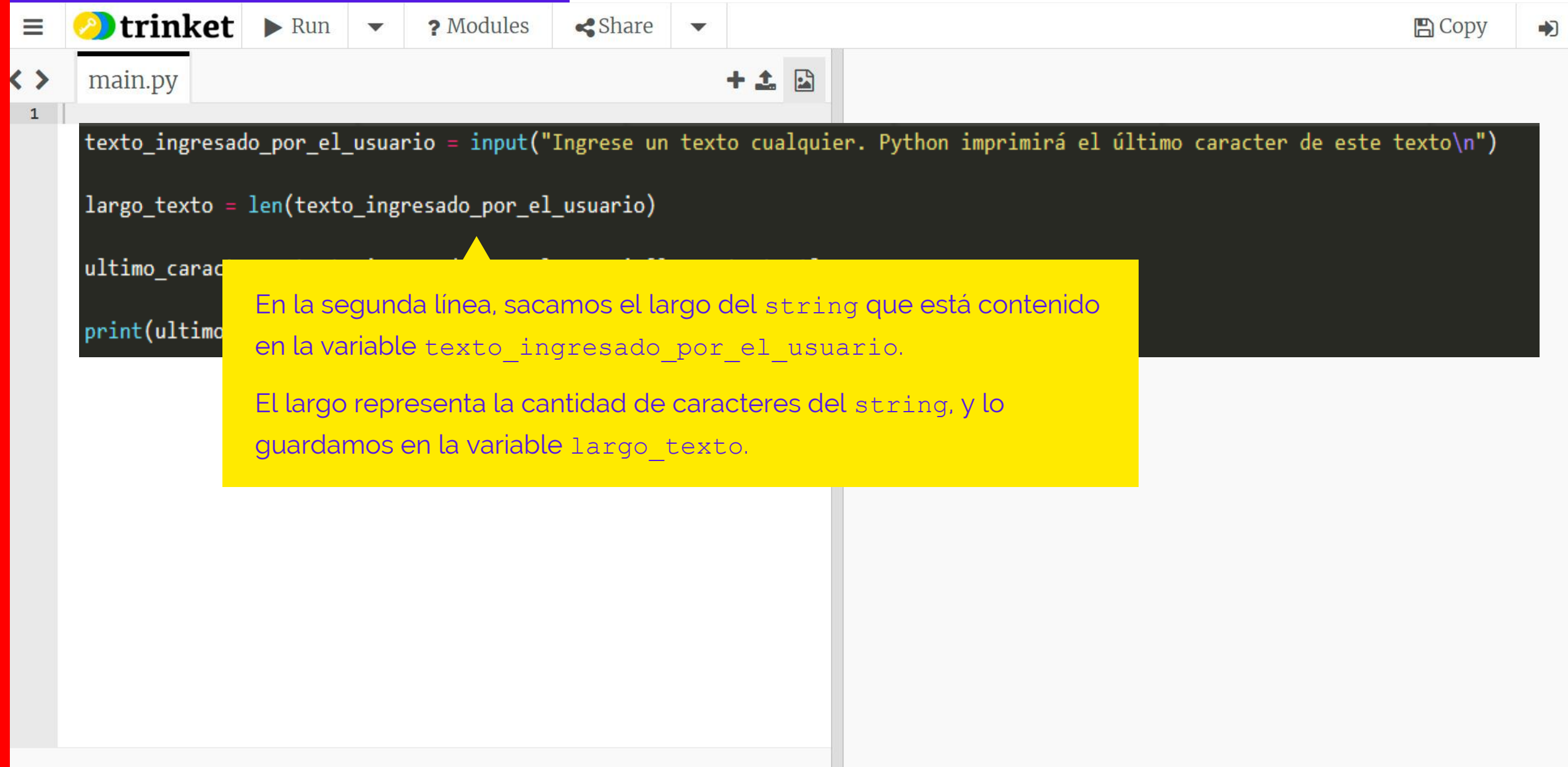
The image shows a screenshot of the Trinket Python IDE interface. The top bar includes the Trinket logo, a 'Run' button, a 'Modules' dropdown, a 'Share' button, and a 'Copy' button. The file name 'main.py' is displayed in the editor tab. The code in the editor is as follows:

```
1 texto_ingresado_por_el_usuario = input("Ingrese un texto cualquier. Python imprimirá el último caracter de este texto\n")  
largo_texto = len(texto_ingresado_por_el_usuario)  
ultimo_caracter = texto_ingresado_por_el_usuario[-1]  
print(ultimo_caracter)
```

A yellow callout box with a purple arrow pointing to the first line of code contains the following text:

En la primera línea le pedimos al usuario que ingrese un texto, y esto lo guardamos en la variable `texto_ingresado_por_el_usuario`

# Analicemos la respuesta



The screenshot shows the Trinket Python IDE interface. The top bar includes the Trinket logo, a 'Run' button, a 'Modules' dropdown, a 'Share' button, and a 'Copy' button. The code editor displays a file named 'main.py' with the following Python code:

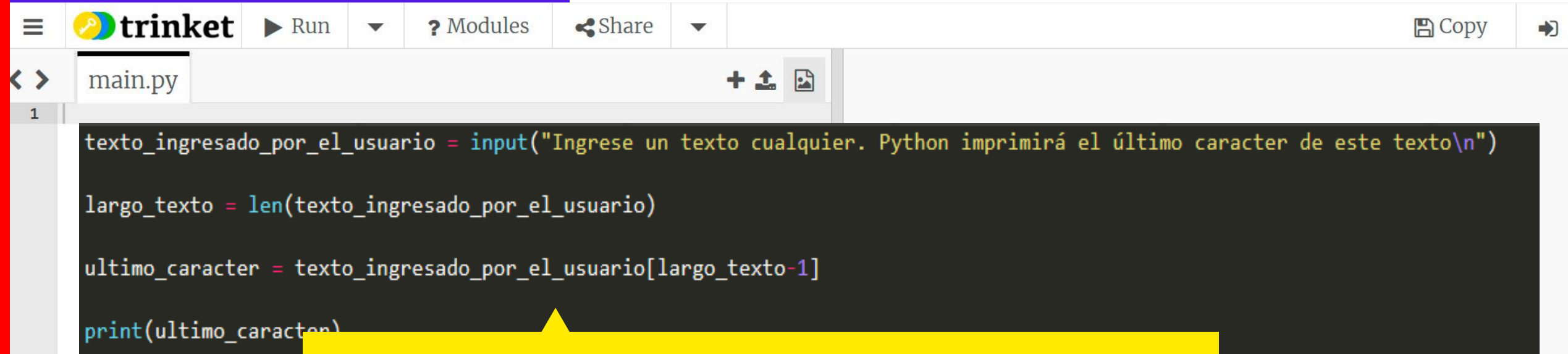
```
1 texto_ingresado_por_el_usuario = input("Ingrese un texto cualquier. Python imprimirá el último caracter de este texto\n")  
largo_texto = len(texto_ingresado_por_el_usuario)  
ultimo_carac  
print(ultimo
```

A yellow callout box points to the `len()` function in the second line of code. The text inside the box explains:

En la segunda línea, sacamos el largo del string que está contenido en la variable `texto_ingresado_por_el_usuario`.

El largo representa la cantidad de caracteres del string, y lo guardamos en la variable `largo_texto`.

# Analicemos la respuesta



The screenshot shows the Trinket Python IDE interface. At the top, there's a navigation bar with a menu icon, the 'trinket' logo, and buttons for 'Run', 'Modules', 'Share', and 'Copy'. Below this, a file explorer shows 'main.py'. The main editor area contains the following Python code:

```
1 texto_ingresado_por_el_usuario = input("Ingrese un texto cualquier. Python imprimirá el último caracter de este texto\n")  
  
largo_texto = len(texto_ingresado_por_el_usuario)  
  
ultimo_caracter = texto_ingresado_por_el_usuario[largo_texto-1]  
  
print(ultimo_caracter)
```

Finalmente, extraemos el último carácter del string que está en la variable `texto_ingresado_por_el_usuario`. Para hacerlo, ocupamos el largo del string que está en la variable `largo_texto`, y extraemos el carácter que está en la última posición del string. Notemos que la última letra está en la posición `largo_texto-1`.

## Analicemos la respuesta:

Carácter	P	A	L	A	B	R	A
Posición	0	1	2	3	4	5	6

Es importante notar que la posición del último carácter está dada por `largo_texto` menos 1. Esto es porque las posiciones se empiezan a contar desde el 0.

El largo de la palabra en Python es 7. Por lo tanto, si queremos extraer la última letra, tenemos que buscar el carácter que está en la posición 6 (largo de la palabra menos 1).

## Revisemos un caso

Digamos que tenemos el siguiente `string`, que representa el nombre completo de una persona:

"Andrés José Pérez Rojas"

No obstante, solo queremos saber el primer apellido  
¿Cómo podríamos resolverlo? Una posible solución es extraer letra a letra del apellido, pero es muy ineficiente.



# Slice

Para eso podemos ocupar, el concepto de `slice`.

Esta acción nos permite extraer una secuencia de caracteres de un `string`.

Se ocupa de la siguiente manera.

Digamos que tenemos un `string` en la variable `s`:

```
s[i:j]
```

Esta acción extrae los caracteres que se encuentran desde la posición `i` (posición inicial) hasta la posición `j - 1` (posición final).

## Veamos un ejemplo para slice

CÓDIGO	RESULTADO
<pre>string_de_ejemplo = "texto de ejemplo" primera_palabra = string_de_ejemplo[0:5] print(primera_palabra)  segunda_palabra = string_de_ejemplo[6:8] print(segunda_palabra)  ultima_palabra = string_de_ejemplo[9:16] print(ultima_palabra)</pre>	<pre>texto de ejemplo</pre>

# Analicemos paso a paso:

## CÓDIGO

```
string_de_ejemplo = "texto de ejemplo"  
primera_palabra = string_de_ejemplo[0:5]  
print(primera_palabra)  
  
segunda_palabra = string_de_ejemplo[6:8]  
print(segunda_palabra)  
  
ultima_palabra = string_de_ejemplo[9:16]  
print(ultima_palabra)
```

En este diagrama podemos ver los caracteres que estamos extrayendo.

T	E	X	T	O		D	E		E	J	E	M	P	L	O
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



# Analicemos paso a paso:

## CÓDIGO

```
string_de_ejemplo = "texto de ejemplo"  
primera_palabra = string_de_ejemplo[0:5]  
print(primera_palabra)
```

```
segunda_palabra = string_de_ejemplo[6:8]  
print(segunda_palabra)
```

```
ultima_palabra = string_de_ejemplo[9:16]  
print(ultima_palabra)
```

Lo primero que se imprime en consola, es la primera palabra del string "texto de ejemplo", que está en la variable `string_de_ejemplo`.

Para poder extraerla, escribimos `[0:5]`.

El 0 es la primera posición. Se usa el 5 como la posición final, ya que para la posición final de un `slice` se considera una posición menos de la que se escribe. Como escribimos `[0:5]`, Python sabe que debe extraer de los caracteres 0 al 4.

T	E	X	T	O		D	E		E	J	E	M	P	L	O
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

# Analicemos paso a paso:

## CÓDIGO

```
string_de_ejemplo = "texto de ejemplo"  
primera_palabra = string_de_ejemplo[0:5]  
print(primera_palabra)  
  
segunda_palabra = string_de_ejemplo[6:8]  
print(segunda_palabra)  
  
ultima_palabra = string_de_ejemplo[9:16]  
print(ultima_palabra)
```

Lo segundo que se imprime en consola, es "de", que es la segunda palabra que está en la variable `string_de_ejemplo`.

Para poder extraerla, escribimos `[6:8]`. El 6 es la posición inicial. Se usa el 8 como la posición final, ya que para la posición final de un `slice` se considera una posición menos de la que se escribe. Como escribimos `[6:8]`, Python sabe que debe extraer de los caracteres 6 al 7.

T	E	X	T	O		D	E		E	J	E	M	P	L	O
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

# Analicemos paso a paso:

## CÓDIGO

```
string_de_ejemplo = "texto de ejemplo"  
primera_palabra = string_de_ejemplo[0:5]  
print(primera_palabra)  
  
segunda_palabra = string_de_ejemplo[6:8]  
print(segunda_palabra)  
  
ultima_palabra = string_de_ejemplo[9:16]  
print(ultima_palabra)
```

Lo último que se imprime en consola, es "ejemplo", que es la tercera palabra que está en la variable `string_de_ejemplo`.

Para poder extraerla, escribimos `[9:16]`. El 9 es la posición inicial. Se usa el 16 como la posición final, ya que para la posición final de un `slice` se considera una posición menos de la que se escribe. Como escribimos `[9:16]`, Python sabe que debe extraer de los caracteres 9 al 15.

T	E	X	T	O		D	E		E	J	E	M	P	L	O
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

## Retomemos el ejemplo

El siguiente `string`, que representa el nombre completo de una persona:

```
"Andrés José Pérez Rojas"
```

Queremos extraer el primer apellido ¿Cómo lo podemos hacer?

CÓDIGO	RESULTADO
<pre>nombre = "Andrés José Pérez Rojas" print(nombre[12:17])</pre>	<pre>Pérez</pre>

## Resolvamos otro problema

Si queremos imprimir todos los caracteres de un `string`. Nuevamente, podríamos imprimir carácter según su posición, pero es muy ineficiente. Para eso, podemos ocupar un `for`.

Python nos permite ocupar un `for` para poder “recorrer” todos los caracteres de un `string`.

`for`

# Resolvamos otro problema en Python

Por ejemplo, digamos que tenemos el siguiente código:

CÓDIGO	RESULTADO
<pre>s="ejemplo" for i in s:     print(i)</pre>	<pre>e j e m p l o</pre>

En el `for` de arriba, la variable `i` representa a cada uno de los caracteres del `string s`.

Por lo tanto, con `print(i)` imprimimos en consola cada uno de los caracteres del `string s`.

# Comparación de strings

¿Cómo podríamos  
comparar `strings`  
para poder  
ordenarlos de forma  
alfabética?

**Python** permite que  
comparemos  
distintos `strings`,  
para poder saber  
cuál viene antes o  
después según un  
orden alfabético.

## Comparación de strings

Para esto podemos ocupar los operadores `<` o `>` para comparar strings.

Por ejemplo:

CÓDIGO	RESULTADO
<code>variable1="a"</code>	<code>True</code>
<code>variable2="b"</code>	<code>False</code>

Vemos que en la primera línea que se imprime en consola, se imprime `True` ya que `"a"` si viene antes que `"b"` según un orden alfabético.

En la segunda línea que se imprime en consola, se imprime `False` dado que `"a"` no es mayor que `"b"` en orden alfabético.



## Comparación de strings

También se pueden ordenar palabras:

CÓDIGO	RESULTADO
<pre>variable1="ejemploa" variable2="ejemplob"</pre>	<pre>True False</pre>
<pre>print(variable1 &lt; variable2) print(variable1 &gt; variable2)</pre>	

## Comparación de strings

¿Y si quisiéramos comparar apellidos?

CÓDIGO	RESULTADO
<pre>variable1="López" variable2="Pérez"</pre>	
<pre>print(variable1 &lt; variable2) print(variable1 &gt; variable2)</pre>	<pre>True False</pre>

# Funciones de strings

---

Al igual que `len()`, existen varias otras funciones que nos permiten trabajar con `strings`.



Estas funciones son sumamente importantes, ya que nos permitirá trabajar con `strings` de muchas maneras.



Recordemos que la gran mayoría de los datos se encuentran en formato texto, por lo que conocer funcionalidades que nos permitan trabajar con ellos nos dará muchas herramientas para aplicar en la vida real.

## Funciones de strings

Asumamos que `s` y `x` son dos strings cualquiera.

```
s.count(x)
```

Cuenta apariciones de `x` en `s`.

CÓDIGO EM2-26

RESULTADO

```
s = "palabra"  
print(s.count("a"))
```

3

CÓDIGO

```
s =  
prin
```

En consola se imprime 3, ya que "a" se encuentra 3 veces en el string "palabra". No solo se puede contar un carácter, sino que también una secuencia de caracteres.

## Funciones de strings

Asumamos que `s` y `x` son dos strings cualquiera.

```
s.find(x)
```

Retorna la posición de `x` en `s`.

CÓDIGO EM2-28

RESULTADO

```
s = "palabra"  
print(s.find("a"))
```

1

CÓDIGO

```
s =  
print(s.find("pe"))
```

En consola se imprime 1, ya que "a" se encuentra por primera vez en la posición 1.

# Funciones de strings

Asumamos que `s` y `x` son dos strings cualquiera.

`s.replace(x, y)` Reemplaza `x` por `y` en el string `s`.

## CÓDIGO EM2-30

```
s = "palabra"  
print(s.replace("a", "b"))
```

## RESULTADO

pblbbrb

En el ejemplo se cambiaron todas las "a" por "b".

## CÓDIGO EM2-31

```
s = "papapapepa"  
print(s.replace("pa", "tx"))
```

txttxpetx

En el ejemplo se cambiaron todas las "pa" por "tx".

## Funciones de `strings`

Asumamos que `s` es un `string` cualquiera.

```
s.upper()
```

Transforma todos los caracteres del `string` a mayúsculas.

CÓDIGO	RESULTADO
<pre>s = "palabra" print(s.upper())</pre>	PALABRA

Esta función, no afecta a los caracteres que ya son mayúsculas.

# Funciones de `strings`

Asumamos que `s` es un `string` cualquiera.

```
s.lower()
```

Transforma todos los caracteres del string a minúsculas.

CÓDIGO

```
s = "PALABRA"  
print(s.upper())
```

RESULTADO

palabra

Esta función, no afecta a los caracteres que ya son minúsculas.



# Funciones de `strings`

Asumamos que `s` es un `string` cualquiera.

`strip()`

Remueve espacios y otros caracteres indeseados al principio y final de `string`.

CÓDIGO	RESULTADO
<pre>s = "\nPALABRA " print(s.strip())</pre>	PALABRA

Al imprimir en la consola, Python quitó el `\n` al inicio y el espacio al final. Este comando es muy importante porque muchas veces hay caracteres "basura" cuando ocupamos datos.



PONTIFICIA  
UNIVERSIDAD  
CATÓLICA  
DE CHILE

# **FUNCIONES DE STRING Y CREACIÓN DE LISTAS**

>>> Parte 2: Listas



# Listas

---

Recordemos que una variable de tipo texto o `string` (en Python), en realidad es una secuencia de caracteres.



Éstos ocupan cierta posición y permiten extraer caracteres de forma individual, solo al saber su posición.



No es solo aplicable para `strings`, sino que es abstraíble a cualquier tipo de datos.

# Listas

---

**Existen variables que** almacenan más de un dato.



Por ejemplo `strings`, donde tenemos una secuencia de caracteres con una posición determinada



Este tipo de variables permiten hacer operaciones + complejas.



Este tipo de variables en Python se denominan **listas**.



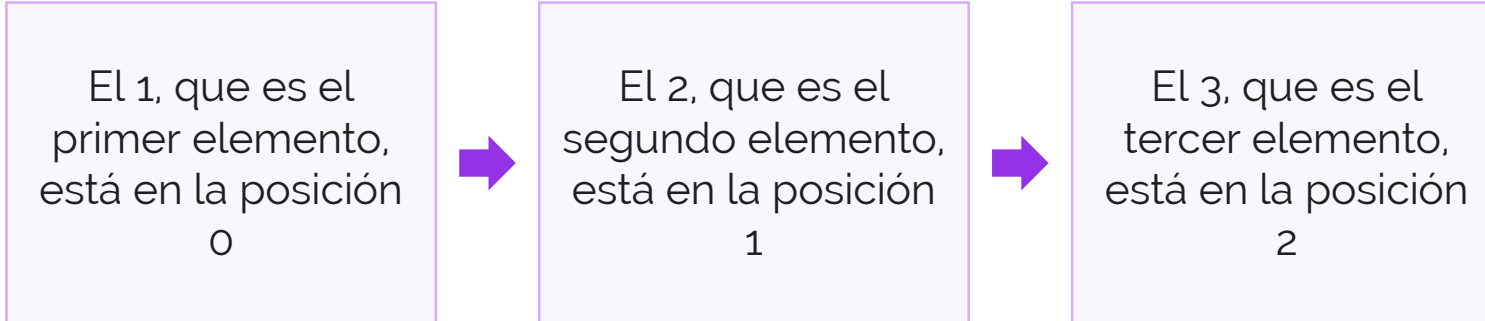
Las listas guardan una secuencia de valores denominados **elementos**.

# Listas

La forma más simple de crear una lista es de la siguiente manera:

## CÓDIGO

```
lista_de_ejemplo = [1, 2, 3]
```



En este caso, creamos una lista que tiene tres elementos de tipo `int`.

Aquí puedes visualizar esta lista:

ELEMENTO	1	2	3
POSICIÓN	0	1	2

# Listas

Además, podemos crear listas con elementos de distinto tipo.

Por ejemplo:

## CÓDIGO

```
variable_tipo_lista = ["texto", 100, -9.346, 0, 0.53, "texto2"]
```

En este caso, creamos una variable de tipo lista, se llama `variable_tipo_lista` y tiene 6 elementos:

- El primer elemento es, un `string` con valor `"texto2"`.
- El segundo elemento es, un `int` con valor `100`.
- El tercer elemento es, un `float` con valor `-9.346`.

# Listas

Además, podemos crear listas con elementos de distinto tipo.

Por ejemplo:

## CÓDIGO

```
variable_tipo_lista = ["texto", 100, 9.346, 0, 0.53, "texto2"]
```

No obstante, en general uno  
debiese ocupar listas con  
elementos del mismo tipo.

En este caso, creamos una variable de tipo lista, se llama `variable_tipo_lista` y tiene 6 elementos:

- El cuarto elemento es, un `int` con valor 0.
- El quinto elemento es, un `float` con valor 0.53.
- El sexto elemento es, un `string` con valor "texto2".

# Listas

Recordemos de `strings` que podemos obtener el carácter en cualquier posición.

CÓDIGO	RESULTADO
<pre>s = "hola" print(s[0])</pre>	<pre>h</pre>

En este caso, lo que se imprime en consola es la letra "h", ya que ésta se en la posición 0 del texto que tiene la variable `s`.



**¿Cómo podríamos hacer lo mismo en listas?**

En realidad, un `string` no es más que una lista de caracteres.



Por lo tanto, muchas de las características que vimos en `strings` anteriormente aplican a las listas.



# Obtener elementos

Para poder obtener algún elemento, de una posición determinada en una lista, se hace de la siguiente forma:

```
elemento_en_posición_x = variable_de_tipo_lista[x]
```

Donde `elemento_en_posición_x`

Es la variable que almacenará el elemento que está en la posición `x` de la lista que está almacenada en la `variable_de_tipo_lista`.

## Veamos un ejemplo

Queremos obtener el tercer elemento de una lista:

CÓDIGO	RESULTADO
<pre>lista1 = ["administración", "ventas", "operaciones", "post-venta"] elemento = lista1[2] print(elemento)</pre>	<pre>operaciones</pre>

En este caso, como extraemos el elemento en la posición 2 y lo almacenamos en la variable `elemento`, al imprimir en consola la variable `elemento` obtenemos el `string` "operaciones".

Si no queda claro, veamos esta imagen que representa a la lista "lista1":

"administración"	"ventas"	"operaciones"	"post-venta"
0	1	2	3

# Obtener elementos

---

También podemos ocupar el concepto de `slice` en listas.



En este caso, nos serviría para obtener una nueva lista desde nuestra lista original.



La nueva lista tendrá ciertos elementos que dependerá de cómo hicimos el `slice`.

# Obtener elementos

Digamos que tenemos una lista `l`, para obtener los elementos desde la posición `i` a la `j` podemos hacer lo siguiente:

```
l[i:j]
```

Por ejemplo, queremos obtener los dos primeros elementos de la misma lista del ejemplo anterior:

CÓDIGO	RESULTADO
<pre>lista1 = ["administración", "ventas" ,"operaciones", "post- venta"] elementos = lista1[0:2] print(elementos)</pre>	<pre>['administración', 'ventas']</pre>

# Editar elementos

Digamos que tenemos la misma lista del ejemplo anterior, que representa los nombres de los distintos departamentos de una empresa. No obstante, el departamento de "post-venta" cambió el nombre a "servicio al cliente".

¿Cómo podríamos editarlo?

**administración**

**ventas**

**operaciones**

~~**post-venta**~~



**servicio al cliente**

# Editar elementos

Veamos un ejemplo:

CÓDIGO	RESULTADO
<pre>lista1 = ["administración", "ventas", "operaciones", "post- venta"] print(lista1) lista1[3] = "servicio al cliente" print(lista1)</pre>	<pre>['administración', 'ventas', 'operaciones', 'post-venta']  ['administración', 'ventas', 'operaciones', 'servicio al cliente']</pre>

En este caso, al elemento en la posición 3 le asignamos el texto "servicio al cliente". El texto ingresado sobrescribe el valor que estaba anteriormente. El elemento que estaba en la posición 3, pasó de tener el valor "post-venta" al valor "servicio al cliente".

# Añadir elementos

¿Cómo podríamos agregar nuevos elementos a nuestra lista?

Sigamos con el mismo ejemplo. Digamos que en esta empresa se quiere agregar un nuevo departamento de marketing. Para eso, a nuestra lista que contiene los nombres de los departamentos de la empresa, ahora queremos agregar un nuevo elemento con el valor "marketing".

**administración**

**ventas**

**operaciones**

**servicio al cliente**

**marketing**

NEW

# Añadir elementos

Para poder agregar elementos a una lista, ocupamos la función `append()`. Digamos que tenemos a una lista `l`. Si queremos agregar un elemento `x`, entonces escribimos:

```
l.append(x)
```

Veamos cómo se vería en nuestro ejemplo:

CÓDIGO	RESULTADO
<pre>lista1 = ["administración", "ventas", "operaciones", "servicio al cliente"] print(lista1) lista1.append("marketing") print(lista1)</pre>	<pre>['administración', 'ventas', 'operaciones', 'servicio al cliente']  ['administración', 'ventas', 'operaciones', 'servicio al cliente', 'marketing']</pre>



## ¿Cómo podríamos quitar elementos de nuestra lista?

---

Sigamos con el mismo ejemplo. Digamos que en esta empresa se quiere eliminar el recientemente agregado departamento de marketing. Para eso, en nuestra lista que contiene los nombres de los departamentos de la empresa, eliminaremos el elemento con el valor "marketing".

**administración**

**ventas**

**operaciones**

**servicio al cliente**

~~**marketing**~~

# Quitar elementos

Para poder eliminar elementos a una lista, ocupamos la función `remove()`. Digamos que tenemos a una lista `l`. Si queremos quitar un elemento `x` que ya está en la lista, entonces escribimos:

```
l.remove(x)
```

Veamos cómo se vería en nuestro ejemplo:

CÓDIGO	RESULTADO
<pre>lista1 = ["administración", "ventas", "operaciones", "servicio al cliente", "marketing"] print(lista1) lista1.remove("marketing") print(lista1)</pre>	<pre>['administración', 'ventas', 'operaciones', 'servicio al cliente', 'marketing']  ['administración', 'ventas', 'operaciones', 'servicio al cliente']</pre>

Si se intenta eliminar un elemento, que no está en la lista, entonces Python arrojará un error.

# Quitar elementos

También podemos remover elementos de la lista por su posición. Para eso ocuparemos la función `pop()`, que en listas se ocupa de la siguiente manera:

```
l.pop(x)
```

Donde `x` es la posición de un elemento dentro de la lista.

CÓDIGO	RESULTADO
<pre>lista1 = ["administración", "ventas", "operaciones", "servicio al cliente", "marketing"] print(lista1) lista1.pop(2) print(lista1)</pre>	<pre>['administración', 'ventas', 'operaciones', 'servicio al cliente', 'marketing']  ['administración', 'ventas', 'servicio al cliente', 'marketing']</pre>

Si se intenta eliminar un elemento, que no está en la lista, entonces Python arrojará un error.

## Otras funciones de listas

`l.extend(m)`    Agrega todos los elementos de la lista `m` al final de la lista `l`.

CÓDIGO	RESULTADO
<pre>lista1 = ["administración", "ventas", "operaciones", "servicio al cliente", "marketing"] lista2 = ["finanzas", "contabilidad"] print(lista1) lista1.extend(lista2) print(lista1)</pre>	<pre>['administración', 'ventas', 'operaciones', 'servicio al cliente', 'marketing']  ['administración', 'ventas', 'operaciones', 'servicio al cliente', 'marketing', 'finanzas', 'contabilidad']</pre>

## Otras funciones de listas

`l.count(x)`      Retorna la cantidad de veces que el elemento `x` se encuentra dentro de `l`.

CÓDIGO	RESULTADO
<pre>lista1 = ["administración", "ventas", "ventas", "ventas", "marketing"] print(lista1.count("ventas"))</pre>	3

## Otras funciones de listas

`l.index(x)`

Retorna el índice de la primera vez que el elemento `x` se encuentra dentro de `l`. Sino se encuentra, genera un error.

CÓDIGO	RESULTADO
<pre>lista1 = ["administración", "ventas", "operac iones", "servicio al cliente"] print(lista1.index("operaciones"))</pre>	2

## Otras funciones de listas

`l.sort()`      Ordena los elementos de l.

CÓDIGO	RESULTADO
<pre>lista1 = ["administración", "ventas", "operaciones", "servicio al cliente"] print(lista1) lista1.sort() print(lista1)</pre>	<pre>['administración', 'ventas', 'operaciones', 'servicio al cliente']  ['administración', 'operaciones', 'servicio al cliente', 'ventas']</pre>

En este caso, se ordenan los elementos de la lista por orden alfabético. Si fueran números, se ordenarían por orden numérico.

# **Operaciones sobre listas**

Sigamos trabajando con nuestro ejemplo con listas. Recordemos que tenemos una lista que tiene los nombres de los departamentos de una empresa. Ahora, queremos imprimir todos los nombres de los departamentos en la consola.

**administración**

**ventas**

**operaciones**

**servicio al cliente**



# Operaciones sobre listas

Para poder hacerlo, podemos ocupar el ciclo `for`. Este comando viene incorporado para ser usado con listas.

Se puede ocupar de la siguiente manera:

```
for i in lista:  
    #en cada iteración del for la variable i  
    #representará a cada uno de los elementos de lista.  
    #Parte desde el elemento que está en la posición 0  
    #hasta llegar al último elemento
```

# Operaciones sobre listas

Entonces, para poder imprimir en consola todos los departamentos de la empresa, lo podemos hacer de la siguiente manera:

CÓDIGO	RESULTADO
<pre>lista1 = ["administración", "ventas", "operaciones", "servicio al cliente"] for i in lista1:     print(i)</pre>	<pre>administración ventas operaciones servicio al cliente</pre>

# Operaciones sobre listas

Para cerrar este apartado, aprenderemos cómo transformar strings a listas.

Digamos que una persona ingresa el nombre de algunos trabajadores.

## CÓDIGO

```
nombres_trabajadores =  
input("Ingrese el nombre de  
los trabajadores del  
departamento de ventas\n")  
print(nombres_trabajadores)
```

## RESULTADO

```
pedro, andrea, daniela, felipe  
, pamela, victoria, juan
```

Podemos ver que en la variable "nombres\_trabajadores", se almacenó el valor "pedro, andrea, daniela, felipe, pamela, victoria, juan".

No obstante, no podemos ocupar cada nombre por separado. Para que fuera información más útil, sería mucho mejor que los nombres se almacenaran como elementos de una lista.

# Operaciones sobre listas

Para hacer operaciones sobre listas, podemos ocupar la función `split()`. Esta función se aplica sobre un `string`, separa a ese `string` en elementos y devuelve una lista.

Veamos como se aplica:

CÓDIGO	RESULTADO
<pre>nombres_trabajadores = input("Ingrese el nombre de los trabajadores del departamento de ventas\n") print(nombres_trabajadores) print(nombres_trabajadores.split(","))</pre>	<pre>Ingrese el nombre de los trabajadores del departamento de ventas  pedro, andrea, daniela, felipe, pamela, victo ria, juan  pedro, andrea, daniela, felipe, pamela, victo ria, juan ['pedro', 'andrea', 'daniela', 'felipe', 'pamela', 'victoria', 'juan\n']</pre>

# Operaciones sobre listas

En este ejemplo, el usuario ingresa el `string`  
"pedro, andrea, daniela, felipe, pamela, victoria, juan".



Este `string` se almacena en la variable `nombres_trabajadores` (se puede observar cuando imprimimos esta variable en consola).



Luego, se aplica la función `split` a esta variable con el `string` ",".



Python, busca cada vez que aparece el `string` "," dentro de la variable `nombres_trabajadores`, y, cuando lo encuentra separa la variable.



Finalmente, podemos ver cómo se separó el `string` inicial en distintos elementos de una lista.

## >>> Cierre

Has finalizado la revisión de los contenidos de esta clase.

A continuación, te invitamos a realizar las actividades y a revisar los recursos del módulo que encontrarás en plataforma.