

# IPI 2019 - Unidad UNO

UNNOBA



## DATOS SIMPLES

NÚMEROS, VALORES DE VERDAD  
Y OTRAS YERBAS



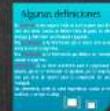
EXIT

ANDEA TE ESTÁS LEYENDO PARA RESOLVER OTROS A LA MAÑANA!



### MAIN IDEA

La idea es introducirnos en conceptos básicos de la **informática**.  
Resolver problemas mediante un **algoritmo**,  
**estructurado** y **parametrizable**.  
Aplicar el paradigma estructurado en la resolución de problemas y  
correctas **estructuras de control**.



# IPI 2019 - Unidad UNO

UNNOBA



## DATOS SIMPLES

NÚMEROS, VALORES DE VERDAD  
Y OTRAS YERBAS



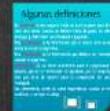
EXIT

ANDEA TE ESTÁS LEYENDO PARA RESOLVER OTROS A LA MAÑANA!



### MAIN IDEA

La idea es introducirnos en conceptos básicos de la **informática**.  
Resolver problemas mediante un **algoritmo**,  
**estructurado** y **parametrizable**.  
Aplicar el paradigma estructurado en la resolución de problemas y  
correctas **estructuras de control**.



# MAIN IDEA

La idea es introducirnos en conceptos básicos de la **programación imperativa**.

**Resolver problemas** mediante un **algoritmo**.  
**Modularizar** y **parametrizar**.

Aplicar el **paradigma estructurado** en la resolución de problemas y correctas **estructuras de control**.

## ¿QUÉ HAREMOS COMO INFORMÁTICOS?

Analizar problemas y plantear soluciones a esos problemas utilizando computadoras.



LA COMPUTADORA NO RAZONA  
NI CREA SOLUCIONES....



...sólo ejecuta una serie de órdenes que  
le proporciona el ser humano.

# ¿QUÉ HAREMOS COMO INFORMÁTICOS?

Analizar problemas y plantear soluciones a esos problemas utilizando computadoras.



# LA COMPUTADORA NO RAZONA NI CREA SOLUCIONES....



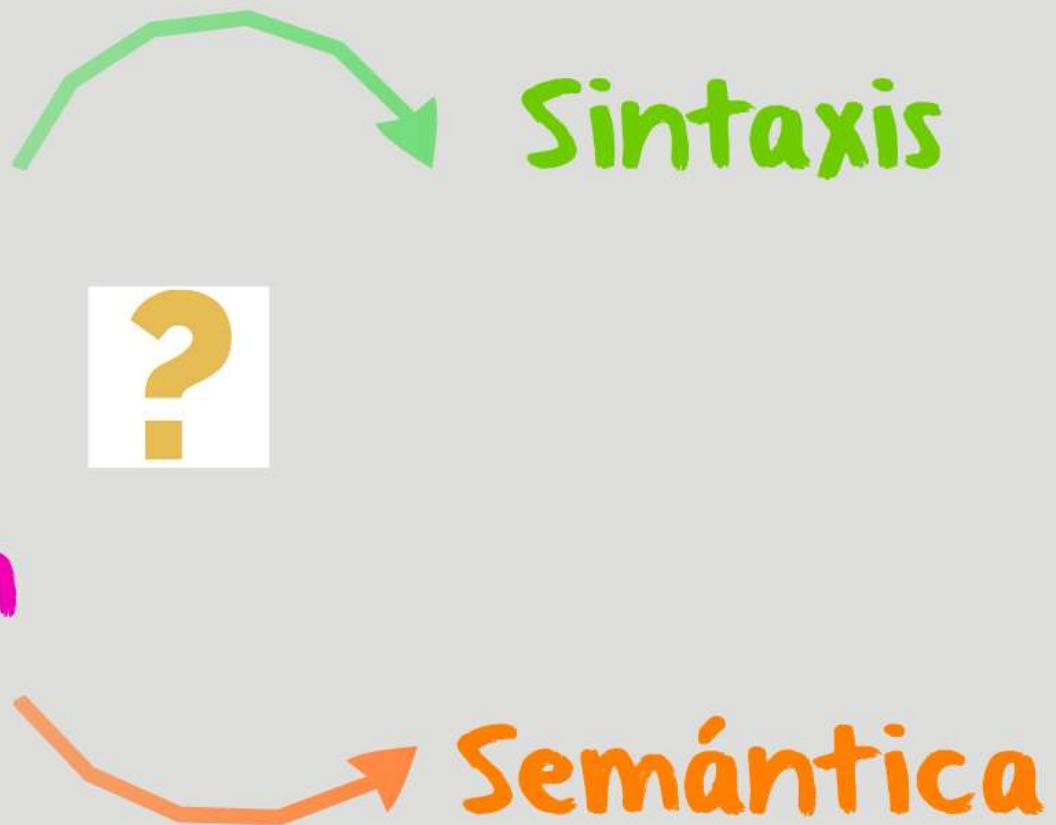
...sólo ejecuta una serie de órdenes que  
le proporciona el ser humano.

# Lenguaje de programación

Un lenguaje es básicamente un medio de comunicación. En programación un lenguaje formal es diseñado para realizar procesos que pueden ser llevados a cabo por máquinas como las computadoras.



# Lenguajes de programación



# SINTAXIS Y SEMÁNTICA

La sintaxis de un lenguaje de programación se define como el **conjunto de reglas** que deben seguirse al escribir los programas para ese lenguaje de programación.

Por ejemplo, para el lenguaje coloquial podemos verlo de la siguiente manera:

"El perro de Juan estaba mojado"

En español para el caso de la palabra "mojado" que es un adjetivo debe presentar género y número. No sería correcto decir:

"El perro de Juan estaba mojada" o  
"El perro de Juan estaba mojados"

Es una regla del lenguaje.

La semántica de un lenguaje de programación refleja el **significado, sentido** o **interpretación** de un lenguaje.

Vayamos a un ejemplo en matemáticas, semántica que si conocemos.

$$2 + 3 = 5$$

Si quisieramos explicar con palabras lo anterior podríamos decir que al número 2 se le añade el 3 y que el resultado de esa suma es el valor 5.

# SINTAXIS Y S

La sintaxis de un lenguaje de programación se define como el conjunto de reglas que deben seguirse al escribir los programas para ese lenguaje de programación.

Por ejemplo, para el lenguaje coloquial podemos verlo de la siguiente manera:

Va  
con

# para ese lenguaje de programación.

Por ejemplo, para el lenguaje coloquial podemos verlo de la siguiente manera:

"El perro de Juan estaba **mojado**"

En español para el caso de la palabra "mojado" que es un adjetivo debe presentar género y número. No sería correcto decir:

"El perro de Juan estaba **mojada**" o

"El perro de Juan estaba **mojados**"

Es una regla del lenguaje.

# Y SEMÁNTICA

de  
el  
en  
as  
ón.

La semántica de un lenguaje de programación refleja el significado, sentido o interpretación de un lenguaje.

Vayamos a un ejemplo en matemáticas, semántica que si

# lenguaje.

Vayamos a un ejemplo en matemáticas, semántica que si conocemos.

$$2 + 3 = 5$$

Si quisiéramos explicar con palabras lo anterior podríamos decir que al número 2 se le añade el 3 y que el resultado de esa suma es el valor 5.

# Algunas definiciones

Un **algoritmo** es un conjunto finito de instrucciones para llevar a cabo una tarea. Consta de número finito de pasos, no debe ser ambiguo y, debe tener una finalidad o propósito.

Una **precondición** es la información que se conoce como verdadera antes de iniciar el algoritmo.

Una **poscondición** es la información que debería ser verdadera al concluir el algoritmo.

**Comentarios.** Es un texto aclaratorio para el programador o el usuario, que no es "entendido" ni ejecutado por la computadora, sino que sirve de soporte para la comprensión del contexto trabajado.

Los comentarios serán de suma importancia cuando se intente modificar o corregir el código.



# RECORDEMOS A ADA DE LA UNIDAD INTRODUCTORIA

Enunciado  
del  
problema  
a resolver

Algoritmo

*Sentencias de nuestro pseudogódigo*



Ahora indíquemosle a Ada que camine dos cuadras, es decir que vaya hasta la esquina de la Calle 1 y Calle 6.

**ALGORITMO (Ejemplo 1):**  
Caminar una cuadra  
Cruzar la calle  
Caminar una cuadra

Ada puede entender las siguientes órdenes:

- Caminar una cuadra
- Doblar hacia la derecha
- Doblar hacia la izquierda
- Recoger un residuo
- Colocar en el cesto de residuos
- Cruzar la calle

Por último debemos saber que Ada comenzará cada algoritmo ubicada en la esquina de la Calle 1 y Calle 2, y siempre mirando hacia la Calle 4



**Secuencia:** en el ejemplo las instrucciones se ejecutan una a continuación de la otra.

En este ejemplo no hay comentarios, quizá porque es un ejemplo sencillo. Sin embargo es de mucha utilidad contar con un texto aclaratorio para el programador o el usuario como soporte para la comprensión del contexto trabajado.

Precondiciones

Poscondición, que  
llegue a este punto

# HERRAMIENTAS QUE VAMOS A USAR EN IPI

Accedan a la pestaña "Herramientas de trabajo"

## Para trabajar junt@s

En la asignatura utilizaremos algunas herramientas, para descargarlas y utilizarlas l@s invitamos a ver la siguiente guía:



También podés acceder a los siguientes tutoriales:



[Tutorial Gráfico - Python IDLE](#)



[Tutorial Gráfico - OnLine Python Tutor](#)

# DATOS SIMPLES

NÚMEROS, VALORES DE VERDAD  
Y OTRAS YERBAS



EXIT ➔

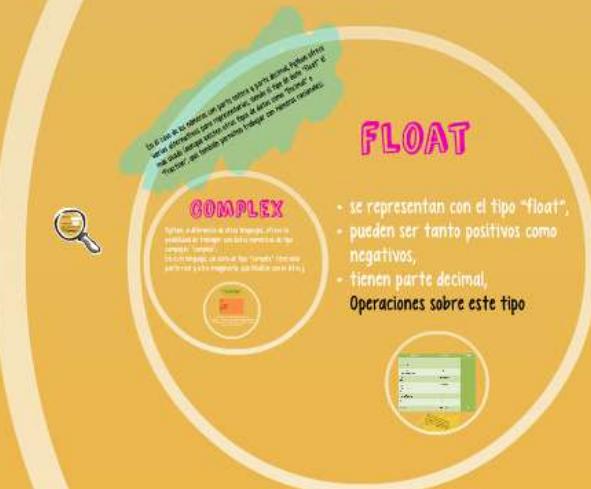
AHORA YA ESTÁS LISTO  
PARA RESOLVER EL TP 1.  
¡A TRABAJAR!

# Tipos numéricos incorporados: int, float, complex

Como ya lo comentamos en IFI  
utilizaremos  
Python para dar  
nuestras primeras pasos en programación.  
En este video veremos los  
tipos de datos numéricos  
incorporados en Python.

## ENTEROS

- se representan con el tipo “int”,
- pueden ser tanto positivos como negativos,
- no tienen parte decimal,  
**Operaciones sobre este tipo**



• Los **suma** es una de las operaciones fundamentales entre los números enteros y se realiza de la forma " $a + b$ ".  
• También existen otras que se expresan como " $a - b$ ".  
• La **resta** es una operación similar a la suma, que indica que se ha de sacar el producto de la resta de dos números.  
• La **multiplicación** es una operación que indica la repetición de un número " $a$ " por " $b$ ".  
• La **división** es una operación que indica el resultado de dividir un número " $a$ " por otro " $b$ ".  
• Podemos realizar otras dos operaciones más, siendo éstas la **potencia** de un número y la **raíz cuadrada**. La potencia de un número es la multiplicación de un mismo número por sí mismo, es decir, " $a^n$ ". La raíz cuadrada de un número " $a$ " es el resultado de dividir el número " $a$ " por el número " $b$ ".

Como ya lo comentamos en IPI  
utilizaremos  
Python para dar  
nuestros primeros pasos en programación.  
Entonces veamos los  
tipos de datos numéricos  
incorporados en Python.

# ENTEROS

## FLOAT

- se representan con el tipo "float",
- pueden ser tanto positivos como negativos,
- tienen parte decimal.

Operaciones sobre este tipo

## COMPLEX

En el caso de los números con parte entera y parte decimal, Python ofrece varias alternativas para representarlos: aparte del tipo de dato "float" más usual, existen cuatro otros tipos de datos con "decimales": "Fraction", que tienen permitido trabajar con números racionales;



- se representan con el tipo "int",
- pueden ser tanto positivos como negativos,
- no tienen parte decimal,

Operaciones sobre este tipo

- La **suma**, es una de las operaciones fundamentales entre dos números enteros y se indica de la forma " $a + b$ ".
- También está la **resta**, que se expresa como " $a - b$ ".
- La **multiplicación**, representada como " $a * b$ ", que indica que se debe calcular el producto entre los enteros " $a$ " y " $b$ ".
- La **división**, representada como " $a / b$ ", que calcula el cociente de dividir " $a$ " por " $b$ ".
- La **potencia**, expresada como " $a ** b$ ", que calcula el resultado de elevar " $a$ " a la potencia " $b$ ".

- La **suma** es una de las operaciones fundamentales entre dos números enteros y se indica de la forma “ $a + b$ ”.
- También está la **resta**, que se expresa como “ $a - b$ ”.
- La **multiplicación**, representada como “ $a * b$ ”, que indica que se debe calcular el producto entre los números “ $a$ ” y “ $b$ ”.
- La **división**, representada como “ $a / b$ ”, que calcula el cociente de dividir “ $a$ ” por “ $b$ ”.
- La **potenciación**, expresada como “ $a ** b$ ”, que calcula el resultado de elevar la base “ $a$ ” a la potencia “ $b$ ”.
- Podemos incluir otras dos operaciones más, ambas asociadas a la división: a la primera la llamaremos “**módulo**” o “**resto**”, se expresa como “ $a \% b$ ” y permite calcular el resto de dividir “ $a$ ” por “ $b$ ”. La segunda, a la cual llamaremos “**división entera**”, la escribiremos de la forma “ $a // b$ ” y se usará para calcular el cociente entero de dividir “ $a$ ” por “ $b$ ”.

En el caso de los números con parte entera y parte decimal, Python ofrece varias alternativas para representarlos, siendo el tipo de dato "float" el más usado (aunque existen otros tipos de datos como "Decimal" o "Fraction", que también permiten trabajar con números racionales).

## COMPLEX

Python, a diferencia de otros lenguajes, ofrece la posibilidad de trabajar con datos numéricos de tipo complejos: "complex".

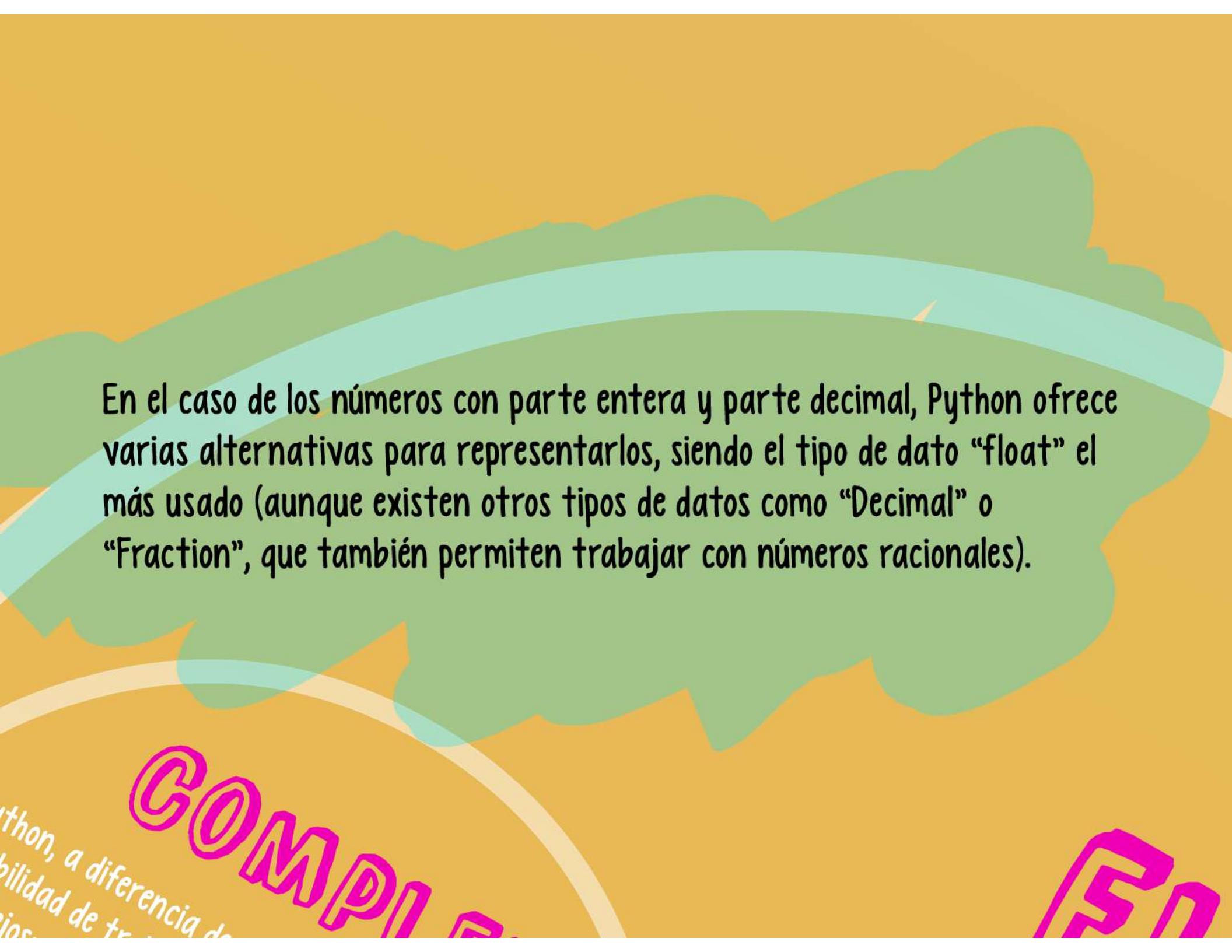
En este lenguaje, un dato de tipo "complex" tiene una parte real y otra imaginaria, que finaliza con la letra j.



## FLOAT

- se representan con el tipo "float",
- pueden ser tanto positivos como negativos,
- tienen parte decimal,  
**Operaciones sobre este tipo**

operación	resultado	explicación
10/2	5.0	
10//2	5	
10%2	0	
10**2	100	
-10**2	-100	
10**0.5	3.1622776601683795	
10**0.25	2.154434690031884	
10**0.125	1.4142135623730951	



En el caso de los números con parte entera y parte decimal, Python ofrece varias alternativas para representarlos, siendo el tipo de dato “float” el más usado (aunque existen otros tipos de datos como “Decimal” o “Fraction”, que también permiten trabajar con números racionales).

Python, a diferencia de otras lenguajes, tiene la capacidad de tratar los

En

EJEMPLOS	DESCRIPCIÓN	OPERADOR
<code>&gt;&gt;&gt; 2.0 + 3.0 5.0 &gt;&gt;&gt; 2.1 + 3.1 5.2</code>	Suma	+
<code>&gt;&gt;&gt; 2.5 - 2.3 0.20000000000000018</code>	Resta	-
<code>&gt;&gt;&gt; 2.9 * 4.0 11.6</code>	Multiplicación	*
<code>&gt;&gt;&gt; 2.0 ** 4 16.0 &gt;&gt;&gt; 4.0 ** 0.5 2.0</code>	Potencia	**
<code>&gt;&gt;&gt; 12.4 / 3 4.13333333333334</code>	División	/
<code>&gt;&gt;&gt; 12.4 // 3 4.0</code>	División entera	//
<code>&gt;&gt;&gt; 12.4 % 3 0.4000000000000036</code>	Resto o módulo	%

Las operaciones son las mismas que se permiten con el tipo de datos "int". La gran diferencia es que, para cada una de ellas, el resultado siempre será de tipo "float".

EJEMPLOS	DESCRIPCIÓN	OPERADOR
>>> 2.0 + 3.0 5.0 >>> 2.1 + 3.1 5.2	Suma	+
>>> 2.5 - 2.3 0.20000000000000018	Resta	-
>>> 2.9 * 4.0 11.6	Multiplicación	*
>>> 2.0 ** 4 16.0 >>> 4.0 ** 0.5 2.0	Potencia	**
>>> 12.4 / 3 4.133333333333334	División	/
>>> 12.4 // 3 4.0	División entera	//
>>> 12.4 % 3 0.40000000000000036	Resto o módulo	%

Las operaciones son las mismas que se permiten con el tipo de datos “int”. La gran diferencia es que, para cada una de ellas, el resultado siempre será de tipo “float”.

# COMPLEX

Python, a diferencia de otros lenguajes, ofrece la posibilidad de trabajar con datos numéricos de tipo complejos: “complex”.

En este lenguaje, un dato de tipo “complex” tiene una parte real y otra imaginaria, que finaliza con la letra j.

Veamos ahora ejemplos trabajados en el shell interactivo de Python:

```
>>> 2  
2  
>>> 2j + 7.8j  
(2j + 7.8j)  
>>>
```

2 es la parte real y 7.8 es la parte imaginaria

Se le añade la “j” para representar la parte imaginaria del número.  
Sin embargo, no nos detendremos en este tipo de datos “complex”,  
ya que excede los objetivos de IPL.

Veamos ahora ejemplos trabajados en el  
shell interactivo de Python:

```
>>> 2  
2  
>>> 2.1 + 7.8j  
(2.1 + 7.8j)  
>>>
```

2.1 es la parte real y 7.8 es la parte imaginaria  
Se le añade la “j” para representar la parte imaginaria del número.  
Sin embargo, no nos detendremos en este tipo de datos “complex”,  
ya que excede los objetivos de IPI.

Con Python (además de la consola de texto) podemos:

- \* usar el intérprete, en el cual podemos ejecutar de a una sentencia a la vez, o



- \* ejecutar un conjunto de sentencias en bloques (creando un programa).



Para esta opción una vez finalizada la escritura del bloque se debe guardar el archivo, desde el menú "File" y eligiendo la opción "Save". Se deberá ingresar un nombre para el archivo, seguido de la extensión ".py" o ".pyw", de la siguiente forma: "ejemplo.py".

Por último para que el programa se ejecute, haremos click en "Run", eligiendo la opción "Run modulo f5", para que se muestre en pantalla el resultado de la ejecución del programa.

# Bool

Para poder comprender cómo se controla el **flujo de un algoritmo** se necesita la **evaluación de las condiciones** que tienen las estructuras de control.

Las condiciones son expresiones lógicas, que pueden ser simples o tener más de una expresión a evaluar.

Como ya hemos visto los valores de verdad pueden ser: “verdadero” o “falso”, “true” o “false”, 0 o 1. Y, dependiendo del valor de verdad el algoritmo tomará un rumbo u otro.

## RECORDEMOS LA UNIDAD INTRODUCTORIA

¿Se acuerdan cuando hablábamos de proposiciones como expresiones lógicas a las cuales se les puede asignar un valor de verdad?  
Bueno, estas expresiones lógicas son las **condiciones** que vamos a evaluar para controlar el flujo de los algoritmos.

# RECORDEMOS LA UNIDAD INTRODUCTORIA

¿Se acuerdan cuando hablábamos de proposiciones como expresiones lógicas a las cuales se les puede asignar un valor de verdad?

Bien, esas expresiones lógicas son las condiciones que vamos a evaluar para controlar el flujo de los algoritmos.

# Simbología

## Igualdad:

$$a = b$$

## Desigualdad:

$a \neq b$

## Identidad:

a is b

## Mayor que:

$$a > b$$

## Mayor o igual que:

$$a \geq b$$

## Menor que:

$$a < b$$

## Menor o igual que:

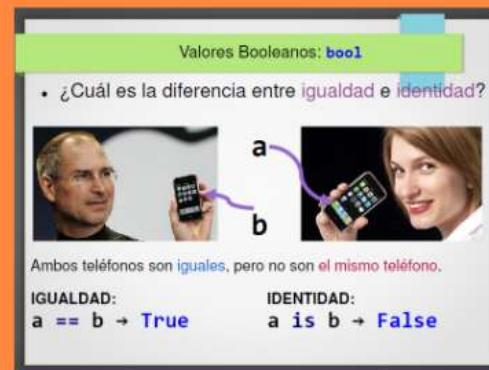
$a \leq b$



## POR EJEMPLO:

```
>>> 10 < 20  
True  
>>> 10 > 15  
False  
>>> 10 >= 25  
False  
>>> 20 <= 40  
True  
>>> 1 == 2  
False  
>>> 1 != 2  
True
```

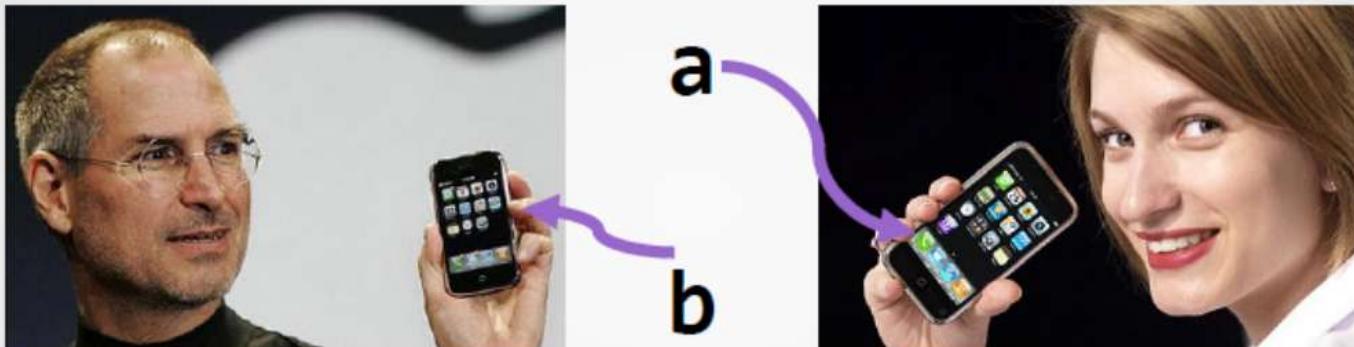
## A NO CONFUNDIR



# A NO CONFUNDIR

Valores Booleanos: `bool`

- ¿Cuál es la diferencia entre **igualdad** e **identidad**?



Ambos teléfonos son **iguales**, pero no son **el mismo teléfono**.

**IGUALDAD:**

`a == b → True`

**IDENTIDAD:**

`a is b → False`

## OPERADORES LÓGICOS

Negación:

Conjunción:

Disyunción:

not a

a and b

a or b

“Sólo apta para mayores de 18 años” edad >= 18

“Válido para personas entre 21 y 35 años inclusive” edad >= 21 and edad <= 35

“Mujeres de más de 60 años” sexo == “mujer” and edad > 60

Puede usarse cualquiera de los delimitadores,  
¡no hay diferencia!  
Pero no pueden incluirse comillas del mismo tipo  
que el usado para delimitar la cadena.

# Cadenas de caracteres. str

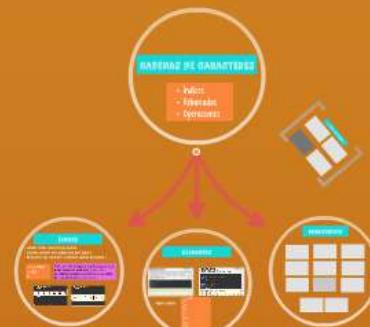
```
>>> "Hola, mundo" hola
SyntaxError: invalid syntax
>>> 'Círculo "Patatas" en mi escritorio'
'Círculo "Patatas" en mi escritorio'
>>> "Los precios de la tortilla son Muy caro"
'Los precios de la tortilla son Muy caro'
>>> 'Don't do this'
SyntaxError: invalid syntax
```

Hasta ahora hemos visto sólo tipos de datos numéricos; sin embargo, existen otros que (particularmente en Python) se llaman “cadenas”. Existe otra forma de llamar a las cadenas de caracteres y lo es a través de la palabra “string”.

¿Qué es una cadena de caracteres? Es una secuencia de caracteres, pudiendo ser éstos: letras, números, espacios en blanco, signos de puntuación o símbolos.

En Python, se escriben encerradas entre “comillas dobles” o entre ‘comillas simples’.

```
>>> cadena="hola a todos los alumnos de IPI"
>>> cadena
' hola a todos los lectores de este libro '
```



Puede usarse cualquiera de los delimitadores,  
¡no hay diferencia!

Pero no pueden incluirse comillas del mismo tipo  
que el usado para delimitar la cadena.

```
>>> "Hola "Hola" Hola"  
SyntaxError: invalid syntax  
>>> 'Ciro "Patatan" es mi mascota'  
'Ciro "Patatan" es mi mascota'  
>>> "Las profe de la teoría son María's"  
"Las profe de la teoría son María's"  
>>> 'Don't do this!'  
SyntaxError: invalid syntax
```

## CADERAS DE CARACTERES

- Índices
- Rebanadas
- Operaciones

### ÍNDICES

- Indexar: tomar caracteres por posición.
- El primer carácter de la cadena está en el índice 0.
- No existe el tipo "carácter"; se obtienen cadenas de longitud 1.

```
>>> s = "HOLA"  
>>> s[0]  
'H'
```

Existe una función que se le aplica a este tipo de datos "cadena de caracteres" y retorna la cantidad de caracteres que contiene; se escribe de la siguiente forma: len(cadena)



### REBANADAS

#### Rebanadas (índice)

• Corte una cadena usando [índice inicial] : [índice final]:[pasos].

• Si omites los tres pasos indicados se usan los valores predeterminados.

• Es recomendable usar el método slicing o str[índice]

• Ejemplos:

```
s[0:3]
```

```
s[0:-1]
```

```
s[0::2]
```

```
s[0:-1:2]
```

```
s[0:3:-1]
```

```
s[0:-1:-1]
```

```
s[0::3:-1]
```

```
s[0:-1::3]
```

```
s[0::3:-2]
```

```
s[0:-1::3:-1]
```

```
s[0::3:-3]
```

```
s[0:-1::3:-2]
```

```
s[0::3:-4]
```

```
s[0:-1::3:-3]
```

```
s[0::3:-5]
```

```
s[0:-1::3:-4]
```

```
s[0::3:-6]
```

```
s[0:-1::3:-5]
```

```
s[0::3:-7]
```

```
s[0:-1::3:-6]
```

```
s[0::3:-8]
```

```
s[0:-1::3:-7]
```

```
s[0::3:-9]
```

```
s[0:-1::3:-8]
```

```
s[0::3:-10]
```

```
s[0:-1::3:-9]
```

```
s[0::3:-11]
```

```
s[0:-1::3:-10]
```

```
s[0::3:-12]
```

```
s[0:-1::3:-11]
```

```
s[0::3:-13]
```

```
s[0:-1::3:-12]
```

```
s[0::3:-14]
```

```
s[0:-1::3:-13]
```

```
s[0::3:-15]
```

```
s[0:-1::3:-14]
```

```
s[0::3:-16]
```

```
s[0:-1::3:-15]
```

```
s[0::3:-17]
```

```
s[0:-1::3:-16]
```

```
s[0::3:-18]
```

```
s[0:-1::3:-17]
```

```
s[0::3:-19]
```

```
s[0:-1::3:-18]
```

```
s[0::3:-20]
```

```
s[0:-1::3:-19]
```

```
s[0::3:-21]
```

```
s[0:-1::3:-20]
```

```
s[0::3:-22]
```

```
s[0:-1::3:-21]
```

```
s[0::3:-23]
```

```
s[0:-1::3:-22]
```

```
s[0::3:-24]
```

```
s[0:-1::3:-23]
```

```
s[0::3:-25]
```

```
s[0:-1::3:-24]
```

```
s[0::3:-26]
```

```
s[0:-1::3:-25]
```

```
s[0::3:-27]
```

```
s[0:-1::3:-26]
```

```
s[0::3:-28]
```

```
s[0:-1::3:-27]
```

```
s[0::3:-29]
```

```
s[0:-1::3:-28]
```

```
s[0::3:-30]
```

```
s[0:-1::3:-29]
```

```
s[0::3:-31]
```

```
s[0:-1::3:-30]
```

```
s[0::3:-32]
```

```
s[0:-1::3:-31]
```

```
s[0::3:-33]
```

```
s[0:-1::3:-32]
```

```
s[0::3:-34]
```

```
s[0:-1::3:-33]
```

```
s[0::3:-35]
```

```
s[0:-1::3:-34]
```

```
s[0::3:-36]
```

```
s[0:-1::3:-35]
```

```
s[0::3:-37]
```

```
s[0:-1::3:-36]
```

```
s[0::3:-38]
```

```
s[0:-1::3:-37]
```

```
s[0::3:-39]
```

```
s[0:-1::3:-38]
```

```
s[0::3:-40]
```

```
s[0:-1::3:-39]
```

```
s[0::3:-41]
```

```
s[0:-1::3:-40]
```

```
s[0::3:-42]
```

```
s[0:-1::3:-41]
```

```
s[0::3:-43]
```

```
s[0:-1::3:-42]
```

```
s[0::3:-44]
```

```
s[0:-1::3:-43]
```

```
s[0::3:-45]
```

```
s[0:-1::3:-44]
```

```
s[0::3:-46]
```

```
s[0:-1::3:-45]
```

```
s[0::3:-47]
```

```
s[0:-1::3:-46]
```

```
s[0::3:-48]
```

```
s[0:-1::3:-47]
```

```
s[0::3:-49]
```

```
s[0:-1::3:-48]
```

```
s[0::3:-50]
```

```
s[0:-1::3:-49]
```

```
s[0::3:-51]
```

```
s[0:-1::3:-49]
```

```
s[0::3:-52]
```

```
s[0:-1::3:-51]
```

```
s[0::3:-53]
```

```
s[0:-1::3:-52]
```

```
s[0::3:-54]
```

```
s[0:-1::3:-53]
```

```
s[0::3:-55]
```

```
s[0:-1::3:-54]
```

```
s[0::3:-56]
```

```
s[0:-1::3:-55]
```

```
s[0::3:-57]
```

```
s[0:-1::3:-56]
```

```
s[0::3:-58]
```

```
s[0:-1::3:-57]
```

```
s[0::3:-59]
```

```
s[0:-1::3:-58]
```

```
s[0::3:-60]
```

```
s[0:-1::3:-59]
```

```
s[0::3:-61]
```

```
s[0:-1::3:-59]
```

```
s[0::3:-62]
```

```
s[0:-1::3:-61]
```

```
s[0::3:-63]
```

```
s[0:-1::3:-62]
```

```
s[0::3:-64]
```

```
s[0:-1::3:-63]
```

```
s[0::3:-65]
```

```
s[0:-1::3:-64]
```

```
s[0::3:-66]
```

```
s[0:-1::3:-65]
```

```
s[0::3:-67]
```

```
s[0:-1::3:-66]
```

```
s[0::3:-68]
```

```
s[0:-1::3:-67]
```

```
s[0::3:-69]
```

```
s[0:-1::3:-68]
```

```
s[0::3:-70]
```

```
s[0:-1::3:-69]
```

```
s[0::3:-71]
```

```
s[0:-1::3:-69]
```

```
s[0::3:-72]
```

```
s[0:-1::3:-71]
```

```
s[0::3:-73]
```

```
s[0:-1::3:-72]
```

```
s[0::3:-74]
```

```
s[0:-1::3:-73]
```

```
s[0::3:-75]
```

```
s[0:-1::3:-74]
```

```
s[0::3:-76]
```

```
s[0:-1::3:-75]
```

```
s[0::3:-77]
```

```
s[0:-1::3:-76]
```

```
s[0::3:-78]
```

```
s[0:-1::3:-77]
```

```
s[0::3:-79]
```

```
s[0:-1::3:-78]
```

```
s[0::3:-80]
```

```
s[0:-1::3:-79]
```

```
s[0::3:-81]
```

```
s[0:-1::3:-79]
```

```
s[0::3:-82]
```

```
s[0:-1::3:-81]
```

```
s[0::3:-83]
```

```
s[0:-1::3:-82]
```

```
s[0::3:-84]
```

```
s[0:-1::3:-83]
```

```
s[0::3:-85]
```

```
s[0:-1::3:-84]
```

```
s[0::3:-86]
```

```
s[0:-1::3:-85]
```

```
s[0::3:-87]
```

```
s[0:-1::3:-86]
```

```
s[0::3:-88]
```

```
s[0:-1::3:-87]
```

```
s[0::3:-89]
```

```
s[0:-1::3:-88]
```

```
s[0::3:-90]
```

```
s[0:-1::3:-89]
```

```
s[0::3:-91]
```

```
s[0:-1::3:-89]
```

```
s[0::3:-92]
```

```
s[0:-1::3:-90]
```

```
s[0::3:-93]
```

```
s[0:-1::3:-91]
```

```
s[0::3:-94]
```

```
s[0:-1::3:-92]
```

```
s[0::3:-95]
```

```
s[0:-1::3:-93]
```

```
s[0::3:-96]
```

```
s[0:-1::3:-94]
```

```
s[0::3:-97]
```

```
s[0:-1::3:-95]
```

```
s[0::3:-98]
```

```
s[0:-1::3:-96]
```

```
s[0::3:-99]
```

```
s[0:-1::3:-97]
```

```
s[0::3:-100]
```

```
s[0:-1::3:-98]
```

```
s[0::3:-101]
```

```
s[0:-1::3:-99]
```

```
s[0::3:-102]
```

```
s[0:-1::3:-100]
```

```
s[0::3:-103]
```

```
s[0:-1::3:-101]
```

```
s[0::3:-104]
```

```
s[0:-1::3:-102]
```

```
s[0::3:-105]
```

```
s[0:-1::3:-103]
```

```
s[0::3:-106]
```

```
s[0:-1::3:-104]
```

```
s[0::3:-107]
```

```
s[0:-1::3:-105]
```

```
s[0::3:-108]
```

```
s[0:-1::3:-106]
```

```
s[0::3:-109]
```

```
s[0:-1::3:-107]
```

```
s[0::3:-110]
```

```
s[0:-1::3:-108]
```

```
s[0::3:-111]
```

```
s[0:-1::3:-109]
```

```
s[0::3:-112]
```

```
s[0:-1::3:-110]
```

```
s[0::3:-113]
```

```
s[0:-1::3:-111]
```

```
s[0::3:-114]
```

```
s[0:-1::3:-112]
```

```
s[0::3:-115]
```

```
s[0:-1::3:-113]
```

```
s[0::3:-116]
```

```
s[0:-1::3:-114]
```

```
s[0::3:-117]
```

```
s[0:-1::3:-115]
```

```
s[0::3:-118]
```

```
s[0:-1::3:-116]
```

```
s[0::3:-119]
```

```
s[0:-1::3:-117]
```

```
s[0::3:-120]
```

```
s[0:-1::3:-118]
```

```
s[0::3:-121]
```

```
s[0:-1::3:-119]
```

```
s[0::3:-122]</pre
```

# CADERNAS DE CARACTERES

- Índices
- Rebanadas
- Operaciones

# ÍNDICES

- Indexar: tomar caracteres por posición.
- El primer carácter de la cadena está en el índice 0.
- No existe el tipo “carácter”, se obtienen cadenas de longitud 1.

```
>>> s = "HOLA"  
>>> s[0]  
'H'
```

Existe una función que se le aplica a este tipo de datos “cadena de caracteres” y retorna la cantidad de caracteres que contiene; se escribe de la siguiente forma: len(cadena).

```
>>> cadena = 'Esta es una cadena'  
>>> len(cadena)  
18  


|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|
| E | s | t | a |   | e | s |   | u | n | a  |    | c  | a  | d  | e  | n  | a  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |

  
>>> cadena[0]  

```

```
>>> cadena = 'Esta es una cadena'  
>>> len(cadena)  
18  
>>> cadena[len(cadena) - 1]  
'a'  

```

```
>>> cadena = 'Esta es una cadena'  
>>> len(cadena)  
18
```

E	s	t	a		e	s		u	n	a		c	a	d	e	n	a
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

```
>>> cadena[0]  
'E'  
>>> cadena[4]  
'  
>>> cadena[17]  
'a'  
>>> cadena[18]  
IndexError: string index out of range
```

```
>>> cadena = 'Esta es una cadena'  
>>> len(cadena)  
18  
>>> cadena[len(cadena) - 1]  
'a'  
>>> cadena[len(cadena) - 2]  
'n'
```

-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
E	s	t	a		e	s		u	n	a		c	a	d	e	n	a
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

```
>>> cadena[-1]  
'a'  
>>> cadena[-2]  
'n'
```

# REBANADAS

## Rebanadas (*slices*)

- Cortar una cadena usando `[desde:hasta]` o `[desde:hasta:paso]`.
- El carácter en la posición indicada en `hasta` no se incluye.
- En matemáticas, es un intervalo cerrado a izquierda y abierto a derecha: `[desde, hasta)`.

```
>>> s = "Buenas Noches América"  
>>> s[0:6]  
'Buenas'  
>>> s[0:6:2]  
'Bea'
```

-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
B	u	e	n	a	s		N	o	c	h	e	s		A	m	e	r	i	c	a
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

## Rebanadas (*slices*)

- Valor por defecto para `desde:0`
- Valor por defecto para `hasta: longitud de la cadena`.
- Valor por defecto para `paso:1`

```
>>> s = "Buenas Noches América"  
>>> s[:2]      # Primeros 2 caracteres  
'Bu'  
>>> s[2:]     # Todo menos los primeros 2 caracteres  
'enas Noches América'  
>>> s[:]       # Cadena completa  
'Buenas Noches América'  
>>> s[::-2]    # Elementos en las posiciones pares  
'Bea ohsAéia'
```

-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
B	u	e	n	a	s		N	o	c	h	e	s		A	m	e	r	i	c	a
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

## Algunos ejemplos

```
>>> s = "0123456789"  
>>> s[0]  
'0'  
>>> s[0:5]  
'01234'  
>>> s[5:2]  
'024'  
>>> s[3:-1]  
'3210'  
>>> s[3:0:-1]  
'321'  
>>> s[::-1]  
'9876543210'  
>>> s[9:-1]  
'987654321'
```

## Rebanadas (*slices*)

- Cortar una cadena usando `[desde:hasta]` o `[desde:hasta:paso]`.
- El carácter en la posición indicada en `hasta` no se incluye.
- En matemáticas, es un intervalo cerrado a izquierda y abierto a derecha: `[desde, hasta)`.

```
>>> s = "Buenas Noches América"
```

```
>>> s[0:6]
```

```
'Buenas'
```

```
>>> s[0:6:2]
```

```
'Bea'
```

-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
B	u	e	n	a	s		N	o	c	h	e	s		A	m	e	r	i	c	a
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

```
>>> r = "0123456789"
```

# Rebanadas (*slices*)

- Valor por defecto para `desde`: 0
- Valor por defecto para `hasta`: longitud de la cadena.
- Valor por defecto para `paso`: 1

```
>>> s = "Buenas Noches América"  
>>> s[:2]      # Primeros 2 caracteres  
'Bu'  
  
>>> s[2:]      # Todo menos los primeros 2 caracteres  
'enas Noches América'  
  
>>> s[:]       # Cadena completa  
'Buenas Noches América'  
  
>>> s[::-2]    # Elementos en las posiciones pares  
'Bea ohsAéia'
```

-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
B	u	e	n	a	s	N	o	c	h	e	s	A	m	e	r	i	c	a		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

# Algunos ejemplos

```
>>> s = "0123456789"  
>>> s[0]  
'0'  
>>> s[0:5]  
'01234'  
>>> s[:5:2]  
'024'  

```

# OPERACIONES

## Búsqueda de subcadena: `in/not in`

`subcadena in cadena` → bool  
`subcadena not in cadena` → bool

Determina si una cadena está o no incluida en otra.

```
>>> "palabra" in "palabras espacios"
True
>>> cadena = "abcde"
>>> sub = "cd"
>>> sub in cadena
False
```

## Funcionalidad incluida: `find()`

`s.find(sub[, desde[, hasta]])` → int

Devuelve la posición en que se encuentra la primera ocurrencia de la cadena sub en s. Si no la encuentra, devuelve -1. Los argumentos `desde` y `hasta` se interpretan con la notación de rebanda (`find()` en `s[desde:hasta]`).

```
>>> "hola" in "hola hola hola"
0
>>> "mierda hola hola".find("hola", 0)
0
>>> "mierda hola hola".find("hola", 0, 4)
2
```

## Funcionalidad incluida: `index()`

`s.index(sub[, desde[, hasta]])` → int

Es como `s.find()` pero genera un error de tipo `ValueError` cuando no se puede encontrar la cadena buscada.

```
>>> "hola dulce hogar".index("hogar")
0
>>> "hola dulce hogar".index("hogar", 1)
12
>>> "hola dulce hogar".index("hogar", 1, 10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
```

## Funcionalidad incluida: `rfind()` y `rindex()`

`s.rfind(sub[, desde[, hasta]])` → int  
`s.rindex(sub[, desde[, hasta]])` → int

Son como `s.find()` y `s.index()` pero comenzando a buscar desde el final de la cadena hacia el comienzo (la `r` es de `reverso`).

## Funcionalidad incluida: `count()`

`s.count(sub[, desde[, hasta]])` → int

Devuelve la cantidad de veces en que aparece la cadena sub solaparse en `s[desde:hasta]`.

```
>>> "hola dulce hogar".count("hogar")
2
>>> "hola dulce hogar".count("dulce")
1
>>> "hola dulce hogar".count("hola")
6
```

## Funcionalidad incluida: `replace()`

`s.replace(viejo, nuevo[, count])` → str

Devuelve una copia de s con todas las ocurrencias de viejo reemplazadas por nuevo. Si se indica el argumento `count`, se reemplazan como máximo las ocurrencias indicadas.

```
>>> s = "hola mundo"
>>> s.replace("mundo", "chico")
'chico'
>>> "hola MUNDO".replace("MUNDO", "MARRA", 2)
'chico MARRA'
```

## Funcionalidad incluida: Mayúsculas y minúsculas.

`s.upper()`

Devuelve una copia de s convertida a mayúsculas.

`s.lower()`

Devuelve una copia de s convertida a minúsculas.

```
>>> s = "hola MUNDO"
>>> s.upper()
'HALA MUNDO'
>>> s.lower()
'hola munto'
```

## Funcionalidad incluida: Mayúsculas y minúsculas.

`s.capitalize()`

Devuelve una copia de s donde el primer carácter es mayúscula y el resto a minúsculas.

`s.title()`

Devuelve una copia de s en donde a cada palabra le aplica `capitalize()`.

```
>>> s = "esta es una PRUEBA"
>>> s.capitalize()
'Esta es una prueba'
>>> s.title()
'ESTA ES UNA PRUEBA'
```

## Funcionalidad incluida: Mayúsculas y minúsculas.

`s.swapcase()`

Devuelve una copia de s en donde las mayúsculas fueron cambiadas por minúsculas y viceversa.

`s.casefold()`

Devuelve una copia de s transformada para que pueda ser comparada sin tener en cuenta mayúsculas y minúsculas.

```
>>> s = "ESTO ES UNA PRUEBA"
>>> s.swapcase()
'ESTO ES UNA PRUEBA'
>>> s.casefold()
'esto es una prueba'
```

## Funcionalidad incluida: `strip()`

`s.strip([chars])` → str

Devuelve una copia de s a la que se le quitaron los blancos tanto del comienzo como del final. La cadena opcional chars indica que se quitan los caracteres contenidos en chars en lugar de quitar blancos.

```
>>> s = "    hola    "
'holo'
>>> "(Dios) A la vista".strip("(Dios")
' A la vista'
```

## Funcionalidad incluida: `lstrip()` y `rstrip()`

`s.lstrip([chars])` → str

Como `s.strip()` pero solamente afectando a los caracteres del comienzo de s (la l es de left → izquierda).

`s.rstrip([chars])` → str

Como `s.strip()` pero solamente afectando a los caracteres del final de s (la r es de right → derecha).

## Búsqueda de subcadena: **in / not in**

**subcadena in cadena -> bool**

**subcadena not in cadena -> bool**

Determina si una cadena está o no incluida en otra

```
>>> "palabra" in "palabras mayores"
```

```
True
```

```
>>> cadena = "abcde"
```

```
>>> sub = "cd"
```

```
>>> sub not in cadena
```

```
False
```

## Funcionalidad incluida: `find()`

`s.find(sub[, desde[, hasta]]) -> int`

Devuelve la posición en que se encuentra la primer ocurrencia de la cadena `sub` en `s`. Si no la encuentra, **devuelve -1**.

Los argumentos `desde` y `hasta` se interpretan con la notación de rebanada (`find()` busca en `s[desde:hasta]`).

```
>>> "hogar dulce hogar".find("hogar")
0
>>> "hogar dulce hogar".find("hogar", 3)
12
>>> "hogar dulce hogar".find("hogar", 3, 8)
-1
```

## Funcionalidad incluida: `index()`

`S.index(sub[, desde[, hasta]]) -> int`

Es como `S.find()` pero genera un error de tipo `ValueError` cuando no se puede encontrar la cadena buscada.

```
>>> "hogar dulce hogar".index("hogar")
0
>>> "hogar dulce hogar".index("hogar", 3)
12
>>> "hogar dulce hogar".index("hogar", 3, 8)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
```

## Funcionalidad incluida: **rfind()** y **rindex()**

**S.rfind(sub[, desde[, hasta]]) -> int**

**S.rindex(sub[, desde[, hasta]]) -> int**

Son como **S.find()** y **S.index()** pero comenzando a buscar desde el final de la cadena hacia el comienzo (la **r** es de reverso).

## Funcionalidad incluida: `count()`

`S.count(sub[, desde[, hasta]]) -> int`

Devuelve la cantidad de veces en la que aparece la cadena `sub` sin solaparse en `S[desde:hasta]`.

```
>>> "hogar dulce hogar".count("hogar")
2
>>> "hogar dulce hogar".count("dulce")
1
>>> "hogar dulce hogar".count("DULCE")
0
```

## Funcionalidad incluida: `replace()`

`s.replace(viejo, nuevo[, count]) -> str`

Devuelve una copia de `s` con todas las ocurrencias de `viejo` reemplazadas por `nuevo`. Si se indica el argumento opcional `count`, se reemplazan como máximo las ocurrencias indicadas.

```
>>> s = 'Hola mundo'  
>>> s.replace('Hola', 'Chau')  
'Chau Mundo'  
>>> "YA YA YA!".replace('YA', 'AHORA', 2)  
'AHORA AHORA YA!'
```

## Funcionalidad incluida: Mayúsculas y minúsculas.

### `s.upper()`

Devuelve una copia de `S` convertida a mayúsculas.

### `s.lower()`

Devuelve una copia de `S` convertida a minúsculas.

```
>>> s = "esta es UNA PRUEBA"  
  
>>> s.upper()  
'ESTA ES UNA PRUEBA'  
  
>>> s.lower()  
'esta es una prueba'
```

## Funcionalidad incluida: Mayúsculas y minúsculas.

### `s.capitalize()`

Devuelve una copia de `s` convierte el primer carácter a mayúscula y el resto a minúsculas.

### `s.title()`

Devuelve una copia de `s` en donde a cada palabra le aplica `capitalize()`.

```
>>> s = "esta es UNA PRUEBA"
```

```
>>> s.capitalize()
```

```
'Esta es una prueba'
```

```
>>> s.title()
```

```
'Esta Es Una Prueba'
```

## Funcionalidad incluida: Mayúsculas y minúsculas.

### `s.swapcase()`

Devuelve una copia de `s` en donde las mayúsculas fueron cambiadas por minúsculas y viceversa.

### `s.casefold()`

Devuelve una copia de `s` transformada para que pueda ser comparada sin tener en cuenta mayúsculas y minúsculas.

```
>>> s = "esta es UNA PRUEBA"
```

```
>>> s.swapcase()
```

```
'ESTO ES una prueba'
```

```
>>> s.casefold()
```

```
'esto es una prueba'
```

## Funcionalidad incluida: `strip()`

`S.strip([chars]) -> str`

Devuelve una copia de `s` a la que se le quitaron los blancos tanto del comienzo como del final. La cadena opcional `chars` indica que se quitan los caracteres contenidos en `chars` en lugar de quitar blancos.

```
>>> "      Hola      ".strip()
'Hola'
>>> "((>>> 4 > 3 <<<))".strip("(>>> )<<< ")
'4 > 3'
>>> "    ABC    ".strip(".")
'ABC'
```

## Funcionalidad incluida: `lstrip()` y `rstrip()`

`S.lstrip([chars]) -> str`

Como `S.strip()` pero solamente afectando a los caracteres del comienzo de `S` (la `l` es de *left* → izquierda)

`S.rstrip([chars]) -> str`

Como `S.strip()` pero solamente afectando a los caracteres del final de `S` (la `r` es de *right* → derecha)

# A TENER EN CUENTA

## Concatenación implícita

Dos o más cadenas de texto juntas se concatenan en una única cadena.

```
>>> "esta es una cadena " "que continúa " "y termina"  
'esta es una cadena que continúa y termina'  
>>> s = "esta cadena " 'termina acá'  
>>> s  
'esta cadena termina acá'
```

## Concatenación (usando el operador +)

Dos cadenas de texto se pueden concatenar produciendo una única cadena nueva.

**cadena\_1 + cadena\_2 → cadena\_nueva**

```
>>> "Introducción a la " + 'Programación Imperativa'  
'Introducción a la Programación Imperativa'  
>>> s = "esto es una cadena" + " y " + "esta es otra"  
>>> s  
'esta es una cadena y esta es otra'
```

## Repetición (usando el operador \*)

Se construye una nueva cadena concatenando repeticiones de la misma cadena de caracteres.

**cadena \* veces → cadena\_repetida**

**veces \* cadena → cadena\_repetida**

```
>>> "Palabra" * 3  
'PalabraPalabraPalabra'  
>>> "{" + " repito " * 4 + "}"  
{ repito repito repito repito }'  
>>> "texto" * 0  
''
```

## Comparación entre cadenas:

- Se utilizan los operadores de igualdad (==), desigualdad (!=) y relaciones (>, <, >=, <=).
- Se compara carácter a carácter hasta encontrar uno que defina la comparación o se acaben los caracteres de una de las cadenas.

```
>>> s = "casa"; t = "casado"; u = "cada"; v = "barco"
```

>>> s == "casa"	>>> s < u
True	False
>>> s == "cAsa"	>>> s > u
False	True
>>> v < s	>>> s == t
True	False
>>> v == s	>>> t > s
False	True
>>> s == u	>>> s > t
False	False

## Concatenación implícita

Dos o más cadenas de texto juntas se concatenan en una única cadena.

```
>>> "esta es una cadena " "que continúa " "y termina"  
'esta es una cadena que continúa y termina'  
>>> s = "esta cadena " 'termina acá'  
>>> s  
'esta cadena termina acá'
```

## Concatenación (usando el operador **+**)

Dos cadenas de texto se pueden concatenar produciendo una única cadena nueva.

**cadena\_1 + cadena\_2 → cadena\_nueva**

```
>>> "Introducción a la " + 'Programación Imperativa'  
'Introducción a la Programación Imperativa'  
>>> s = "esto es una cadena" + " y " + "esta es otra"  
>>> s  
'esta es una cadena y esta es otra'
```

## Repetición (usando el operador **\***)

Se construye una nueva cadena concatenando repeticiones de la misma cadena de caracteres.

**cadena \* veces → cadena\_repetida**

**veces \* cadena → cadena\_repetida**

```
>>> "Palabra" * 3
'PalabraPalabraPalabra'
>>> "{" + " repito " * 4 + "}"
'{ repito repito repito repito }'
>>> "texto" * 0
''
```

## Comparación entre cadenas:

- Se utilizan los operadores de igualdad (==), desigualdad (!=) y relacionales (>, <, >=, <=).
- Se compara caracter a caracter hasta encontrar uno que defina la comparación o se acaben los caracteres de una de las cadenas.

```
>>> s = "casa"; t = "casado"; u = "cada"; v = "barco"
```

```
>>> s == "casa"
```

```
True
```

```
>>> s == "cAsa"
```

```
False
```

```
>>> v < s
```

```
True
```

```
>>> v == s
```

```
False
```

```
>>> s == u
```

```
False
```

```
>>> s < u
```

```
False
```

```
>>> s > u
```

```
True
```

```
>>> s == t
```

```
False
```

```
>>> t > s
```

```
True
```

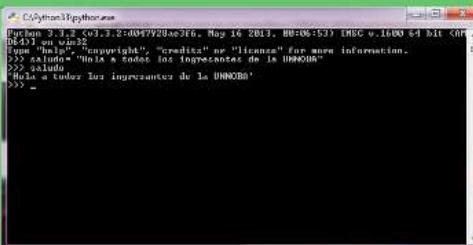
```
>>> s > t
```

```
False
```

# Entrada / salida

Durante muchos años, la **ÚNICA** forma de interactuar con una computadora era a través de una terminal (o consola de texto). La terminal permitía el ingreso de texto por teclado y la salida de información a través de texto impreso en papel o (posteriormente) texto visualizado en un monitor.

Esta forma de interactuar sigue utilizándose en muchos ámbitos (administración de servidores, redes, bases de datos, control de cambios, mantenimiento de entornos de trabajo, etc).



## Entrada

La función `input()` permite interactuar con el usuario:

```
input([prompt]) -> str
```

Esta función lee caracteres desde la entrada estándar (normalmente el teclado) hasta presionar **ENTER** (`\n`) y devuelve una cadena con los caracteres leídos. Opcionalmente, antes de pedir el ingreso de los caracteres, puede mostrar la cadena especificada en `prompt`:

```
>>> s = input("Ingrese un texto: ")
Ingrese un texto: Hola
>>> s
'Hola'
```

## Entrada

La función `input()` retorna siempre una cadena de caracteres.

¿Qué sucede si se quiere leer un número?

```
>>> n = input("Ingrese un numero: ")
Ingrese un numero: 42
>>> n
'42'
>>> n + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't convert 'int' object to str implicitly
```

Es necesario realizar la **conversión** de una cadena a un número **explícitamente**.

## Conversión de cadenas a números

Las clases `int` y `float` permiten crear valores numéricos a partir de cadenas que contengan caracteres que representen números. Es indispensable que la cadena **represente** un número válido de acuerdo al tipo al que se quiere convertir:

```
>>> int('42')
42
>>> float('42')
42.0
>>> float('23.5')
23.5
>>> int('23.5')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '23.5'
```

## Salida

La función `print()` permite realizar una salida:

```
print([value[, ...]])
```

Esta función convierte en cadenas a cada uno de los valores entre paréntesis y los escribe en la salida estándar (normalmente la pantalla), separándolos por espacios en blanco y dejando el cursor en una nueva línea (que le sigue a la última línea en la que se escribió algo):

```
>>> print("Los números primos son:", 4, 6, 8, 15, 16, 23, 42)
Los números primos son: 4 6 8 15 16 23 42
>>> print()
>>>
```

## Salida

Las cadenas de caracteres pueden contener **códigos de escape**. Estos códigos permiten realizar acciones sobre la terminal, alterando la salida normal o representar caracteres que de otra forma no podrían representarse.

Los códigos de escape comienzan con una barra invertida (`\`) seguidos de un carácter que le da significado al código. Un código comúnmente empleado es `\n`, que representa un salto a una nueva línea.

```
>>> print("\nUn texto\nOtro")
Un texto
Otro
>>>
```

## Salida

Si todos los códigos de escape comienzan con una barra invertida (`\`), ¿cómo se puede escribir una barra invertida en una cadena? Usando `\\\\"`, que se interpreta como una única `\`:

```
>>> "\\\\""
'\\'
>>> print("\\\\")
\\
```

Hay que **tener cuidado** con los códigos de escape porque a veces nos pueden sorprender:

```
>>> print("C:\\nuevo")
C:\nuevo
```

## Salida

Otros códigos de escape son `\t` (tabulador horizontal) y `\r` (retorno de carro).

`\t` completa con espacios en blanco hasta llegar a la próxima columna que sea múltiplo de 8.

```
>>> print("Hola\tMundo\tOtro")
Hola      Mundo      Otro
```

`\r` hace que el cursor regrese al comienzo de la linea actual.

```
>>> print("0123456789\r0000")
0000
```

## Salida

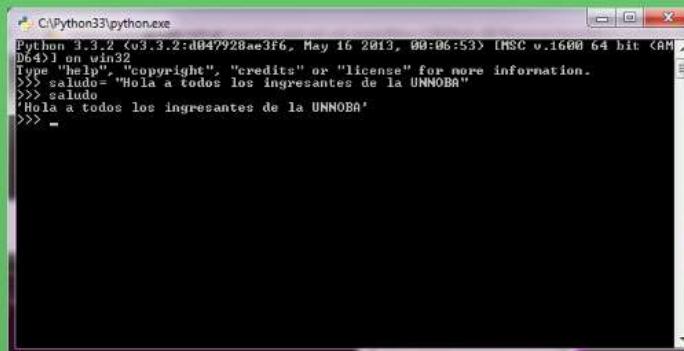
¿Cómo podría escribirse comillas simples y dobles dentro de una misma cadena? Simple, usando códigos de escape!

```
Existen "\'" (comillas simples) y "\\"" (comillas dobles).
>>> print('C:\nue \"complicado\" que \'complejo\'')
Es más "complicado" que 'complejo'.
>>> print('Es más \'complicado\" que \"complejo\'')
Es más "complicado" que 'complejo'.
```

Existen más códigos de escape que no veremos.

# Entrada / salida

Durante muchos años, la **ÚNICA** forma de interactuar con una computadora era a través de una terminal (o consola de texto). La terminal permitía el ingreso de texto por teclado y la salida de información a través de texto impreso en papel o (posteriormente) texto visualizado en un monitor. Esta forma de interactuar sigue utilizándose en muchos ámbitos (administración de servidores, redes, bases de datos, control de cambios, mantenimiento de entornos de trabajo, etc).



## Entrada

La función `input()` permite interactuar con el usuario:

```
input([prompt]) → str
```

Esta función lee caracteres desde la entrada estándar (normalmente el teclado) hasta presionar **ENTER** (→) y devuelve una cadena con los caracteres leídos. Opcionalmente, antes de pedir el ingreso de los caracteres, puede mostrar la cadena especificada en `prompt`.

```
>>> s = input("Ingrese un texto: ")
Ingrese un texto: Hola<
>>> s
'Hola'
```

## Entrada

La función `input()` retorna siempre una cadena de caracteres. ¿Qué sucede si se quiere leer un número?

```
>>> n = input("Ingrese un número: ")
Ingrese un número: 42<
>>> n
'42'
>>> n + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Es necesario realizar la **conversión** de una cadena a un número explicativamente.

## Conversión de cadenas a números

Las clases `int` y `float` permiten crear valores numéricos a partir de cadenas que contengan caracteres que representen números. Es indispensable que la cadena **represente un número válido** de acuerdo al tipo al que se quiere convertir.

```
>>> int('42')
42
>>> float('42')
42.0
>>> float('23.5')
23.5
>>> int('23.5')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '23.5'
```

## Salida

## Salida

## Salida

## Entrada

La función `input()` permite interactuar con el usuario:

**`input([prompt]) → str`**

Esta función lee caracteres desde la entrada estándar (normalmente el teclado) hasta presionar **ENTER** ( $\leftarrow$ ) y devuelve una cadena con los caracteres leídos. Opcionalmente, antes de pedir el ingreso de los caracteres, puede mostrar la cadena especificada en `prompt`.

```
>>> s = input("Ingrese un texto: ")
Ingrese un texto: Hola←
>>> s
'Hola'
```

## Entrada

La función `input()` retorna siempre una cadena de caracteres.  
¿Qué sucede si se quiere leer un número?

```
>>> n = input("Ingrese un número: ")  
Ingrese un número: 42←  
>>> n  
'42'  
>>> n + 1  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: Can't convert 'int' object to str implicitly
```

Es necesario realizar **la conversión** de una cadena a un número explícitamente.

## Conversión de cadenas a números

Las clases `int` y `float` permiten crear valores numéricos a partir de cadenas que contengan caracteres que representen números. Es indispensable que la cadena **represente un número válido** de acuerdo al tipo al que se quiere convertir.

```
>>> int('42')
42
>>> float('42')
42.0
>>> float('23.5')
23.5
>>> int('23.5')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '23.5'
```

## Salida

La función `print()` permite realizar una salida:

```
print([value[, ...]])
```

Esta función convierte en cadenas a cada uno de los valores entre paréntesis y los escribe en la salida estándar (normalmente la pantalla), separándolos por espacios en blanco y dejando el cursor en una nueva línea (que le sigue a la última línea en la que se escribió algo).

```
>>> print("Los números ganadores:", 4, 8, 15, 16, 23, 42)
Los números ganadores: 4 8 15 16 23 42
>>> print()

>>>
```

## Salida

Las cadenas de caracteres pueden contener **códigos de escape**. Estos códigos permiten realizar acciones sobre la terminal, alternando la salida normal o representar caracteres que de otra forma no podrían representarse.

Los códigos de escape comienzan con una barra invertida (\) seguidos de un carácter que le da significado al código.

Un código comúnmente empleado es "**\n**", que representa un salto a una nueva línea.

```
>>> print("\nUn texto\nOtro")
```

```
Un texto
```

```
Otro
```

```
>>>
```

## Salida

Si todos los códigos de escape comienzan con una barra invertida (\), ¿cómo se puede escribir una barra invertida en una cadena? Usando "`\\"`", que se interpreta como una única \.

```
>>> "\\"  
'\\'  
>>> print("\\")  
\
```

Hay que **tener cuidado** con los códigos de escape porque a veces nos pueden sorprender:

```
>>> print("C:\\nuevo")  
C:  
uevo
```

## Salida

Otros códigos de escape son "`\t`" (tabulador horizontal) y "`\r`" (retorno de carro).

"`\t`" completa con espacios en blanco hasta llegar a la próxima columna que sea múltiplo de 8.

```
>>> print('\tHola\tHola de nuevo\tChau')
      Hola      Hola de nuevo    Chau
```

"`\r`" hace que el cursor regrese al comienzo de la línea actual.

```
>>> print('12345678\rABC')
ABC45678
```

## Salida

¿Cómo podría escribirse comillas simples y dobles dentro de una misma cadena? Simple, ¡usando códigos de escape!

Existen "`\'`" (comillas simples) y "`\\"`" (comillas dobles).

```
>>> print('Es más "complicado" que \'complejo\' .')
Es más "complicado" que 'complejo'.
>>> print("Es más \"complicado\" que 'complejo' .")
Es más "complicado" que 'complejo'.
```

Existen más códigos de escape que no veremos.

AHORA YA ESTÁS LISTO  
PARA RESOLVER EL TP 1.

¡A TRABAJAR!

EXIT

Esta presentación fue diseñada por el siguiente equipo docente:

Mg. Claudia Russo  
Lic. Paula Lencina  
AC María Lanzillotta  
Prog. Trinidad Picco  
AC. Patricia Miguel  
AC M. del Carmen Muller  
Lic. Mariana Adó  
Lic. Cecilia Rastelli



**Atribución – No Comercial – Compartir Igual (by-nc-sa):**  
No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original. Esta licencia no es una licencia libre.



**Atribución (Attribution):** En cualquier explotación de la obra autorizada por la licencia será necesario reconocer la autoría (obligatoria en todos los casos).



**No Comercial (Non commercial):** La explotación de la obra queda limitada a usos no comerciales.



**Compartir Igual (Share alike):** La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Por lo que las autoras agradecen el reconocimiento de la autoría correspondiente.  
Para mayor información se recomienda acceder a :

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

# IPI 2019 - Unidad UNO

## UNNOBA



La idea es i

Aplicar el pa