mAYO 2023



Ing. Hector Sanchez Tamayo

Miriam Pamela Andrade Martínez

Universidad Autónoma De La Laguna

Manual

Tópicos Avanzados de tecnologías de la información

Contenido

[¿Qué es Python? 5](#_Toc134037959)

[Instalación y Descargas. 5](#_Toc134037960)

[Conceptos 8](#_Toc134037961)

[Configuración en Visual Studio Code 9](#_Toc134037962)

[1. Introducción 10](#_Toc134037963)

[1.1. Markdown 10](#_Toc134037964)

[1.1.1. Encabezados 10](#_Toc134037965)

[1.1.2. Bloques de citas 10](#_Toc134037966)

[1.1.3. Sección de Código 10](#_Toc134037967)

[1.1.4. Cursiva 11](#_Toc134037968)

[1.1.5. Negritas 11](#_Toc134037969)

[1.1.6. Línea Horizontal 11](#_Toc134037970)

[1.1.7. Listas Ordenadas 12](#_Toc134037971)

[1.1.8. Listas No Ordenadas 12](#_Toc134037972)

[1.1.9. Links o Vínculos 13](#_Toc134037973)

[1.1.10. Tablas 13](#_Toc134037974)

[1.1.11. Imágenes y Videos 13](#_Toc134037975)

[2. Datos y Operadores 15](#_Toc134037976)

[2.1. Variables 15](#_Toc134037977)

[1.2.1. Métodos 15](#_Toc134037979)

[2.2. Números 16](#_Toc134037983)

[a) INT (números enteros) 17](#_Toc134037984)

[b) FLOAT (números decimales) 17](#_Toc134037986)

[c) COMPLEX (números complejos) 17](#_Toc134037989)

[2.3. Textos o Cadenas de Caracteres 19](#_Toc134037990)

[2.4. Booleanos 20](#_Toc134037991)

[2.5. Comprobación de Tipos 20](#_Toc134037992)

[2.6. Operadores 21](#_Toc134037993)

[a) Adición 21](#_Toc134037994)

[b) Sustracción 22](#_Toc134037996)

[c) Multiplicación 22](#_Toc134037997)

[d) División 22](#_Toc134037998)

[e) División Entera Truncada 22](#_Toc134037999)

[f) Módulo o Residuo de la División 22](#_Toc134038000)

[g) Exponenciación 22](#_Toc134038001)

[2.7. Operadores Lógicos 23](#_Toc134038002)

[a) AND 23](#_Toc134038003)

[b) OR 23](#_Toc134038004)

[c) NOT 24](#_Toc134038005)

[2.8. Lectura de teclado 24](#_Toc134038006)

[2.9. Formateo 25](#_Toc134038014)

[2.10. Ejercicios\2.10-Ejercicios.ipynb 25](#_Toc134038023)

[3. Estructura de datos 26](#_Toc134038026)

[3.1. Índices y Slicing 26](#_Toc134038027)

[3.1.1. Índices 26](#_Toc134038032)

[3.1.2. Slicing 26](#_Toc134038033)

[3.2. Listas 27](#_Toc134038034)

[3.3. Tuplas 28](#_Toc134038035)

[3.4. Conjuntos 29](#_Toc134038036)

[3.5. Diccionarios 30](#_Toc134038037)

[3.6. Ejercicios\3.6-Ejercicios.ipynb 31](#_Toc134038042)

[4. Sentencias Condicionales 32](#_Toc134038043)

[4.1. If/Else/Elif: 32](#_Toc134038044)

[4.2. Bucle While 33](#_Toc134038045)

[4.3. Bucle For 35](#_Toc134038046)

[4.4. Ejercicios\4.4-Ejercicios.ipynb 35](#_Toc134038047)

[5. Funciones 36](#_Toc134038048)

[5.1. Definir una Función 36](#_Toc134038054)

[5.2. Retorno de Valores 37](#_Toc134038055)

[5.3. Argumentos y Parámetros 37](#_Toc134038056)

[5.4. Argumentos Indeterminados 38](#_Toc134038057)

[5.5. Funciones Recursivas 39](#_Toc134038058)

[5.6. Funciones Integradas 40](#_Toc134038069)

[5.7. Funciones Lambda 40](#_Toc134038070)

[5.8. Funciones Open 41](#_Toc134038071)

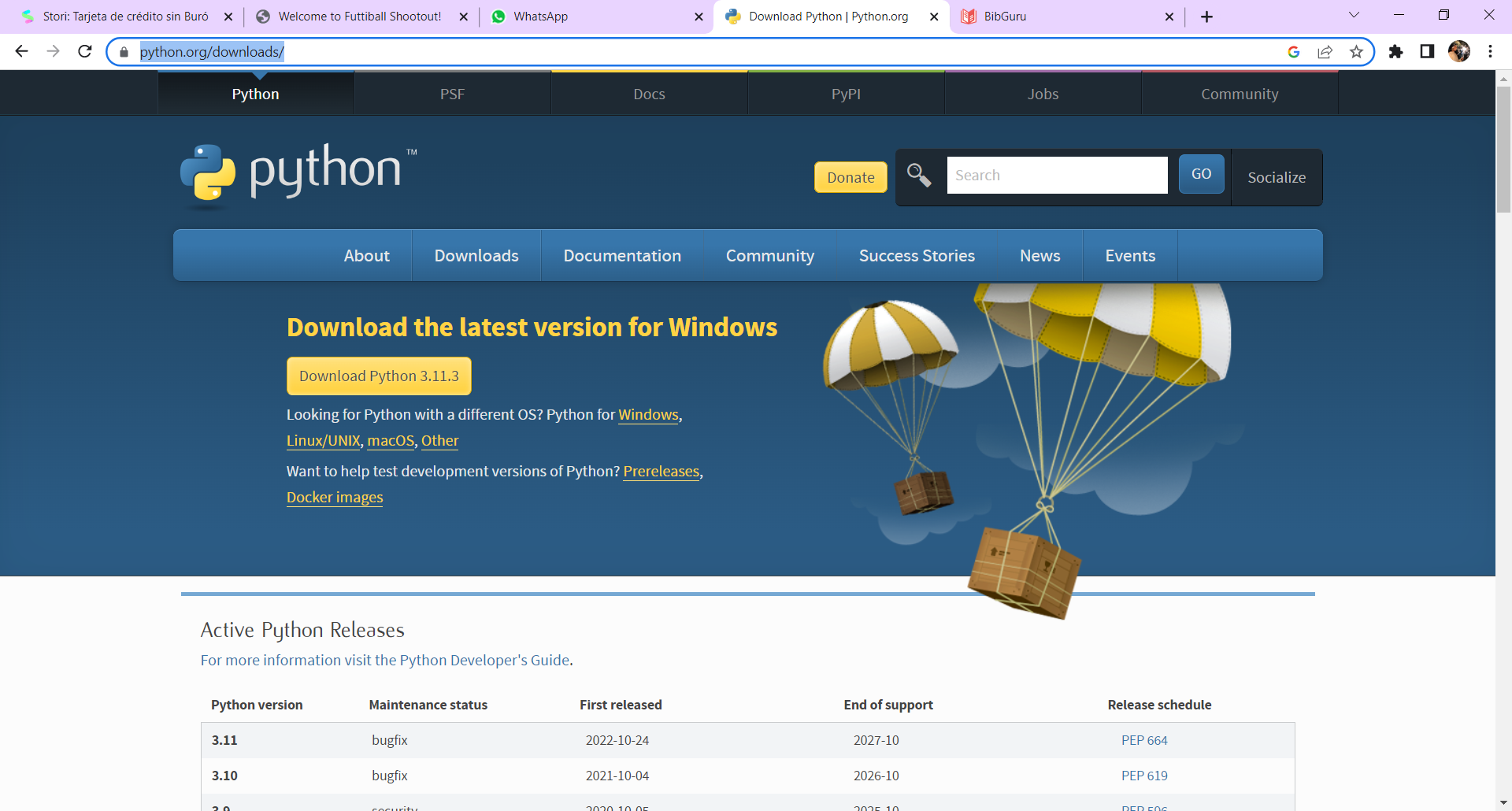
[5.9. Ejercicios\5.9-Ejercicios.ipynb 41](#_Toc134038072)

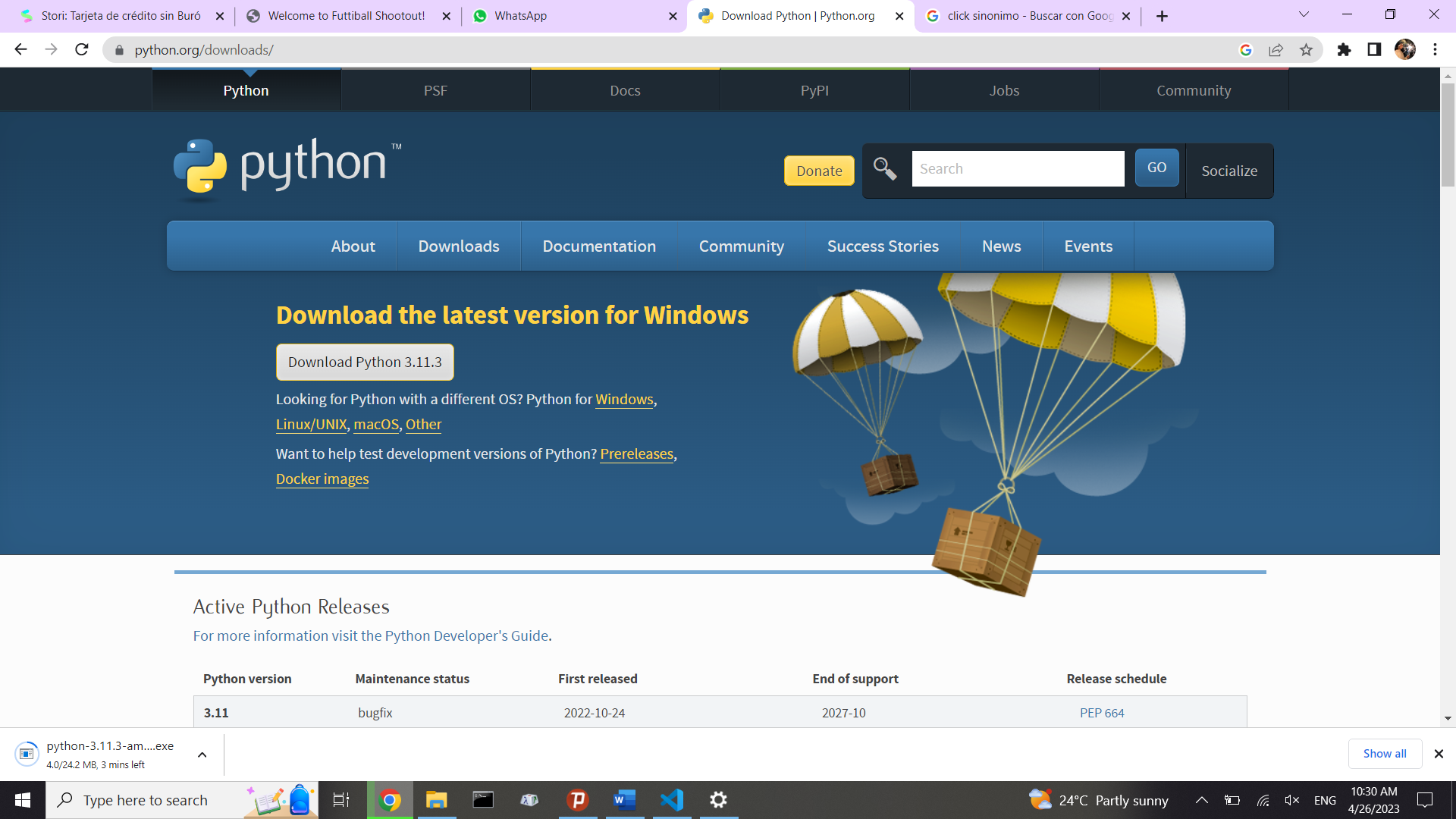
[6. Errores y Excepciones 42](#_Toc134038073)

# ¿Qué es Python?

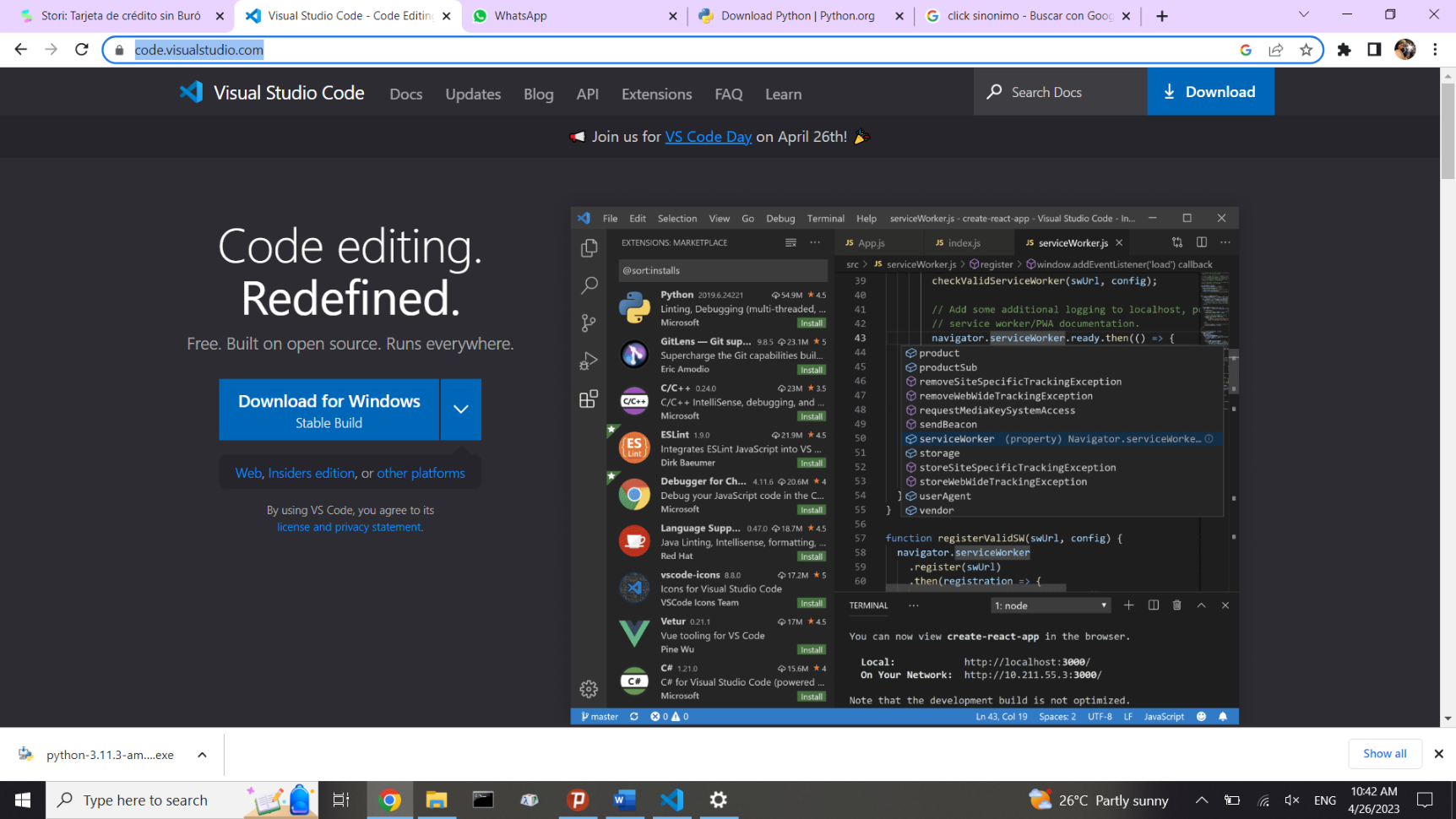
Python es un lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones de todo tipo. Es un lenguaje sencillo de leer y escribir debido a su alta similitud con el lenguaje humano. Además, se trata de un lenguaje multiplataforma de código abierto y, por lo tanto, gratuito, lo que permite desarrollar software sin límites. Con el paso del tiempo, Python ha ido ganando adeptos gracias a su sencillez y a sus amplias posibilidades, sobre todo en los últimos años, ya que facilita trabajar con inteligencia artificial, big data, machine learning y data science, entre muchos otros campos en auge.

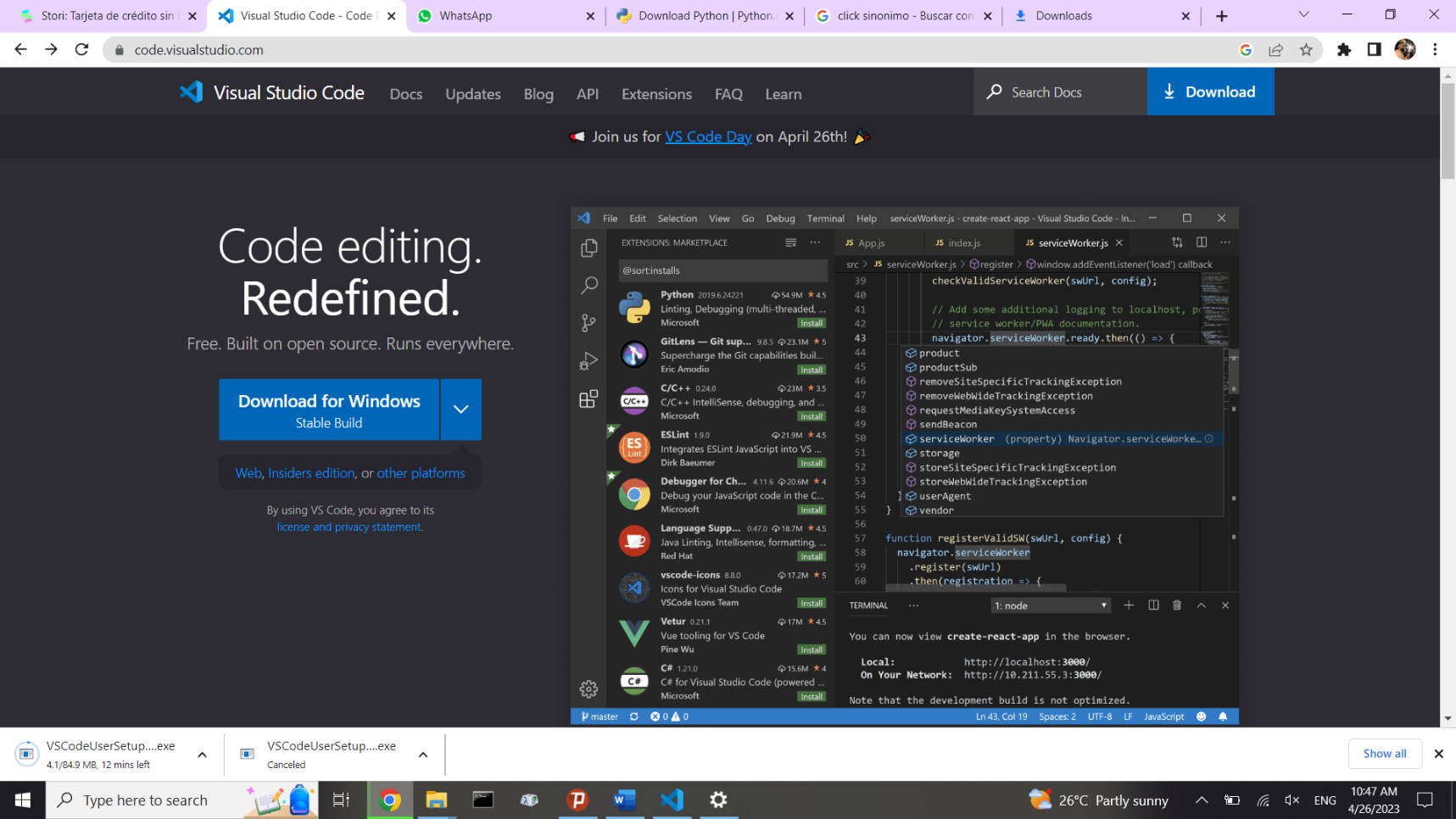
# Instalación y Descargas.

Ingresar a <https://www.python.org/downloads/> en el navegador web de su preferencia.

Se da clic en el botón que dice Download Python 3.11.3, esto hará que comience la instalación.

Al igual si no se cuenta con Visual Studio Code, se realizarían casi los mismos pasos.

Ingresar a <https://code.visualstudio.com/>

Se da clic en el botón que dice Download for Windows, esto hará que comience la instalación.

# Conceptos

IPhyton

Es un shell interactivo que añade funcionalidades extra al modo interactivo incluido con Python como resaltado de líneas y errores mediante colores, una sintaxis adicional para el shell, autocompletado mediante tabulador de variables, módulos y atributos; entre otras funcionalidades.

Jupyter Notebook

Es una interfaz web de código abierto que permite la inclusión de texto, video, audio, imágenes, así como la ejecución de código a través del navegador en múltiples lenguajes.

.py

Son archivos de programas o scripts escritos en Python, un lenguaje de programación orientado a objetos interpretado. Se pueden crear y editar con un editor de texto, sino que requieren un intérprete de Python para funcionar. A menudo se utilizan para los servidores de programación web y otros sistemas informáticos.

.ipynb

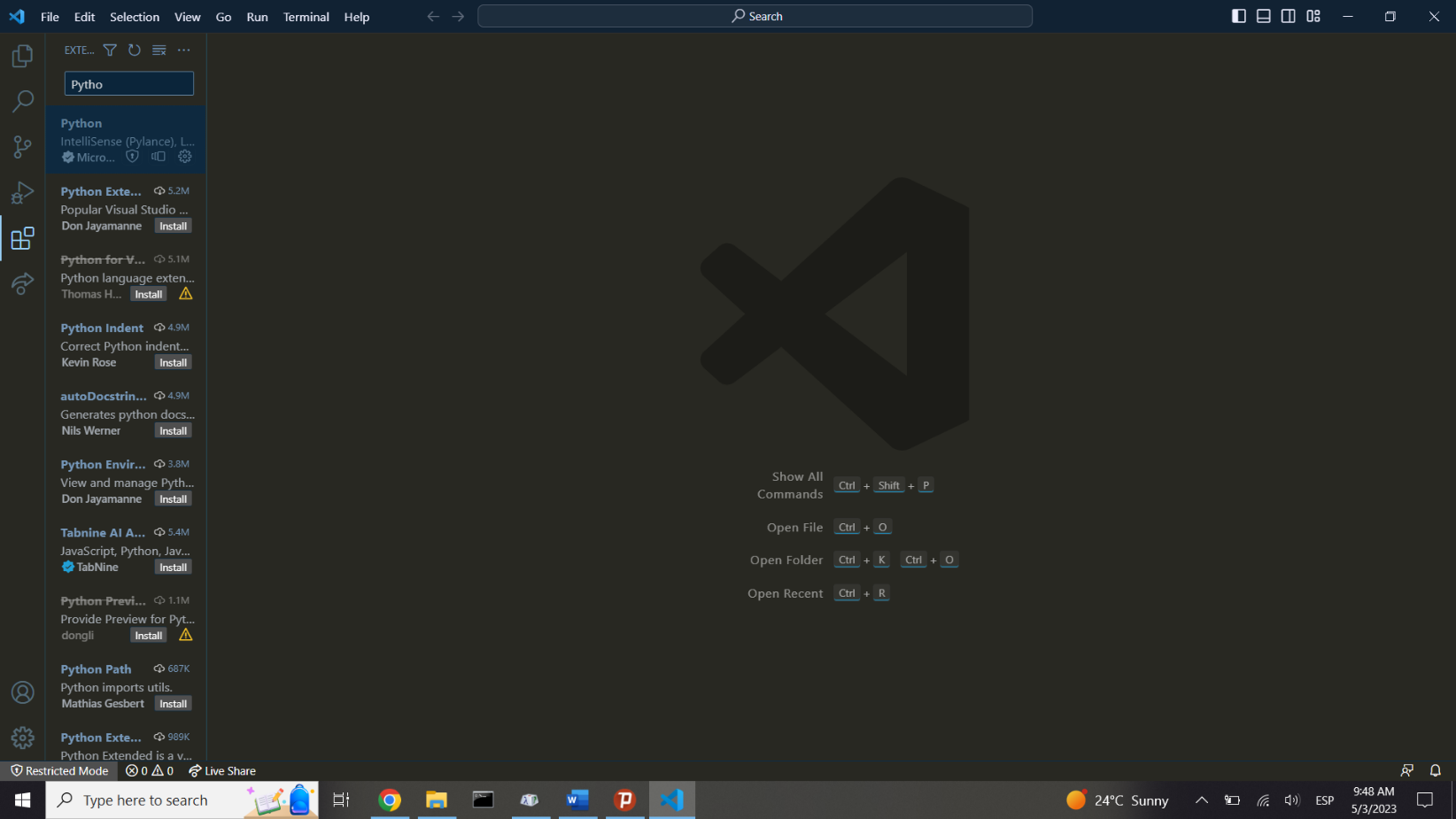
Es un documento de cuaderno utilizado por Jupyter Notebook, un entorno computacional interactivo diseñado para ayudar a los científicos a trabajar con el lenguaje Python y sus datos.

# Configuración en Visual Studio Code

Se instalan estas extensiones en Visual Studio Code:

* Jupyter
* Python
* Pylance
* Jupyter Notebook Renderer

Para instalarlas se busca en el menú de lado derecho el icono de cuadrado, que le quitan un pedazo, ese lleva las extensiones.

Al dar clic, se muestran las extensiones que se encuentran instaladas, y en la parte superior un buscador, ahí se pondrá la que se está buscando para después instalarla.

Al ser extensiones que se relacionan, es común que al instalar una, se instalen las demás, solo queda confirmar, checando una por una.

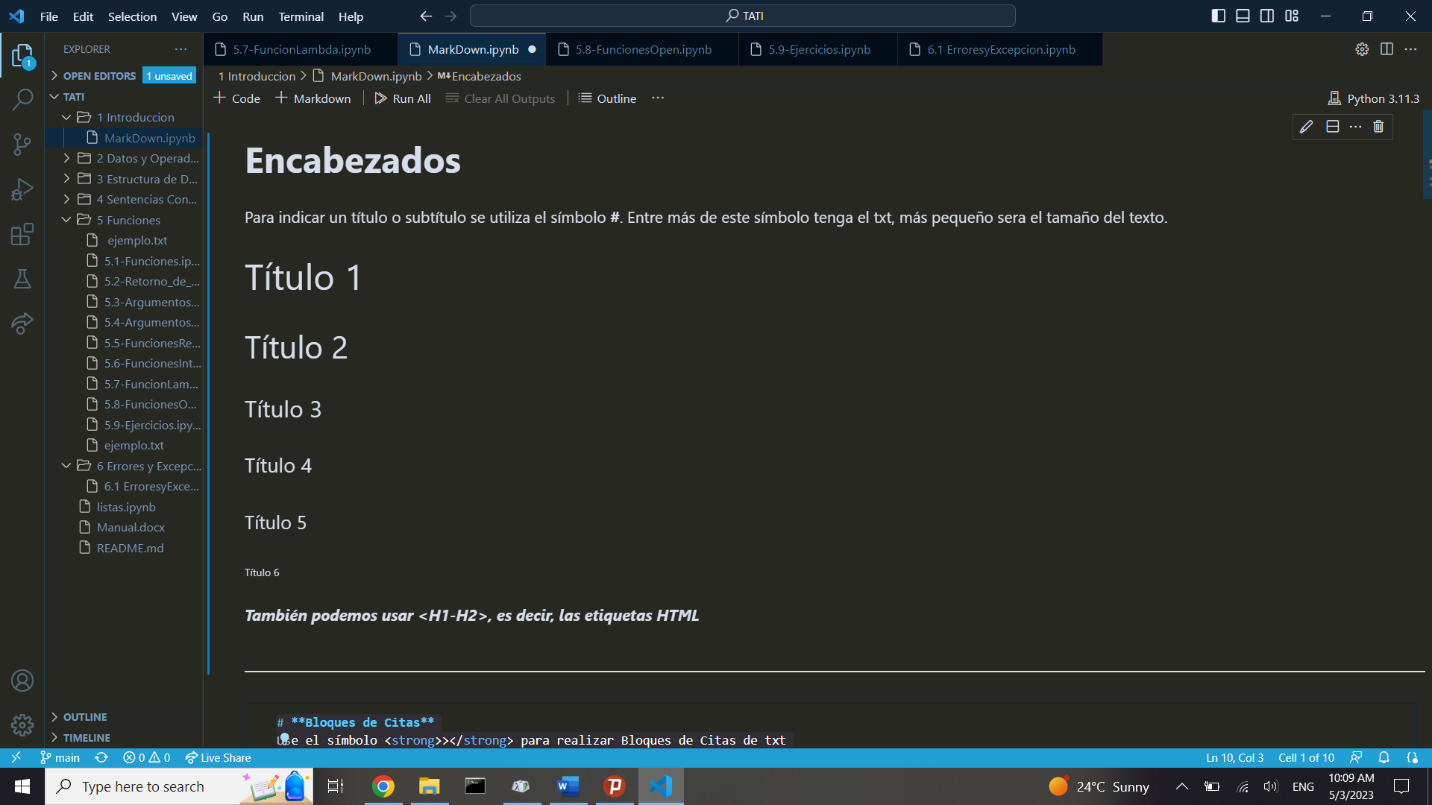
# Introducción

## Markdown

### Encabezados

 Para indicar un título o subtítulo se utiliza el símbolo #.

Entre más de este símbolo tenga el txt, más pequeño será el tamaño del texto.

# Título 1

Título 2

# Título 3

Título 4

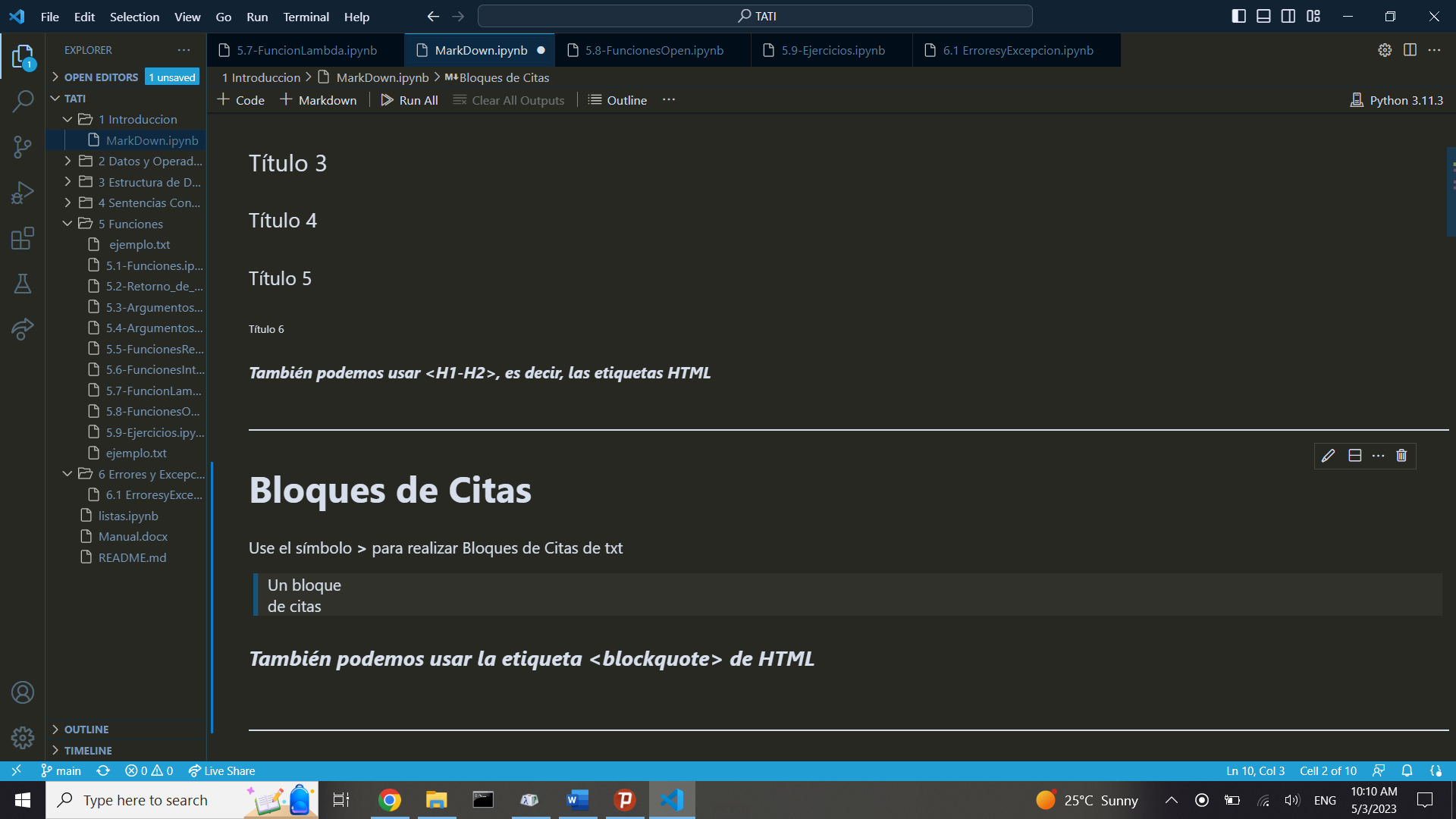
# Título 5

Título 6

También podemos usar \<H1-H2>, es decir, las etiquetas HTML

### Bloques de citas

Use el símbolo “>” para realizar Bloques de Citas de txt

> Un bloque <br> de citas

También podemos usar la etiqueta \<blockquote> de HTML

### Sección de Código

Use el símbolo \`\`\` al inicio y al final del bloque de código. Ponga el nombre del lenguaje en el que se va a codificar después de los 3 primeros símbolos.

```Python

# Este código no corre o compila

str = "Este es un bloque de código"

print(str)

```

También podemos usar la etiqueta \<code> de HTML, sin embargo, no tendrá la paleta de colores que usa el lenguaje.

<code>

str = "Este es un bloque de código"

<br>

print(str)

</code>

### Cursiva

Para cambiar el texto a cursiva, usar el símbolo \\* antes y después del texto

*\*Cursiva\**

### Negritas

Para cambiar el texto a negritas, usar los símbolos \\*\\* antes y después del texto

**Negritas**

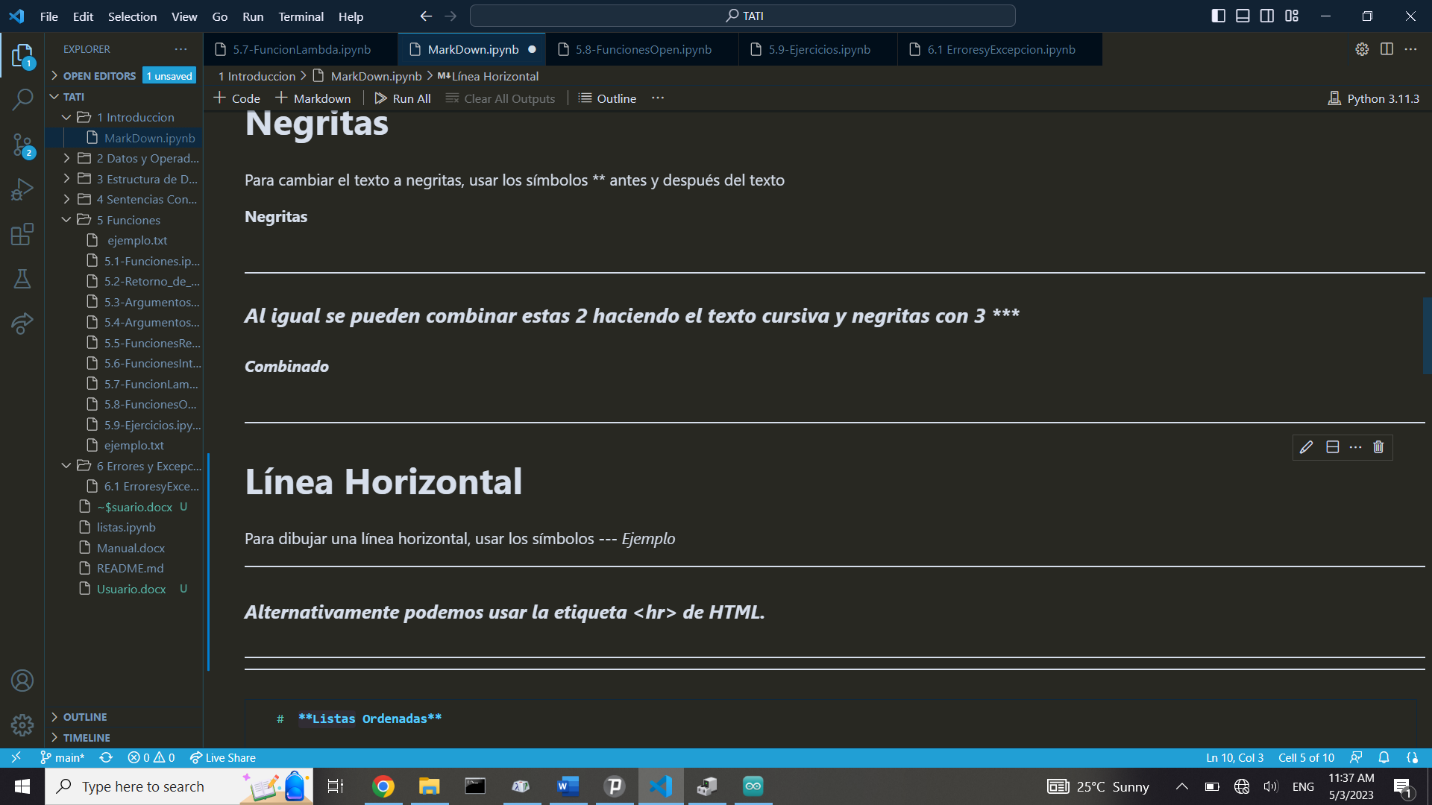
Al igual se pueden combinar estas 2 haciendo el texto cursiva y negritas con 3 \* \* \*

***\*Combinado\****

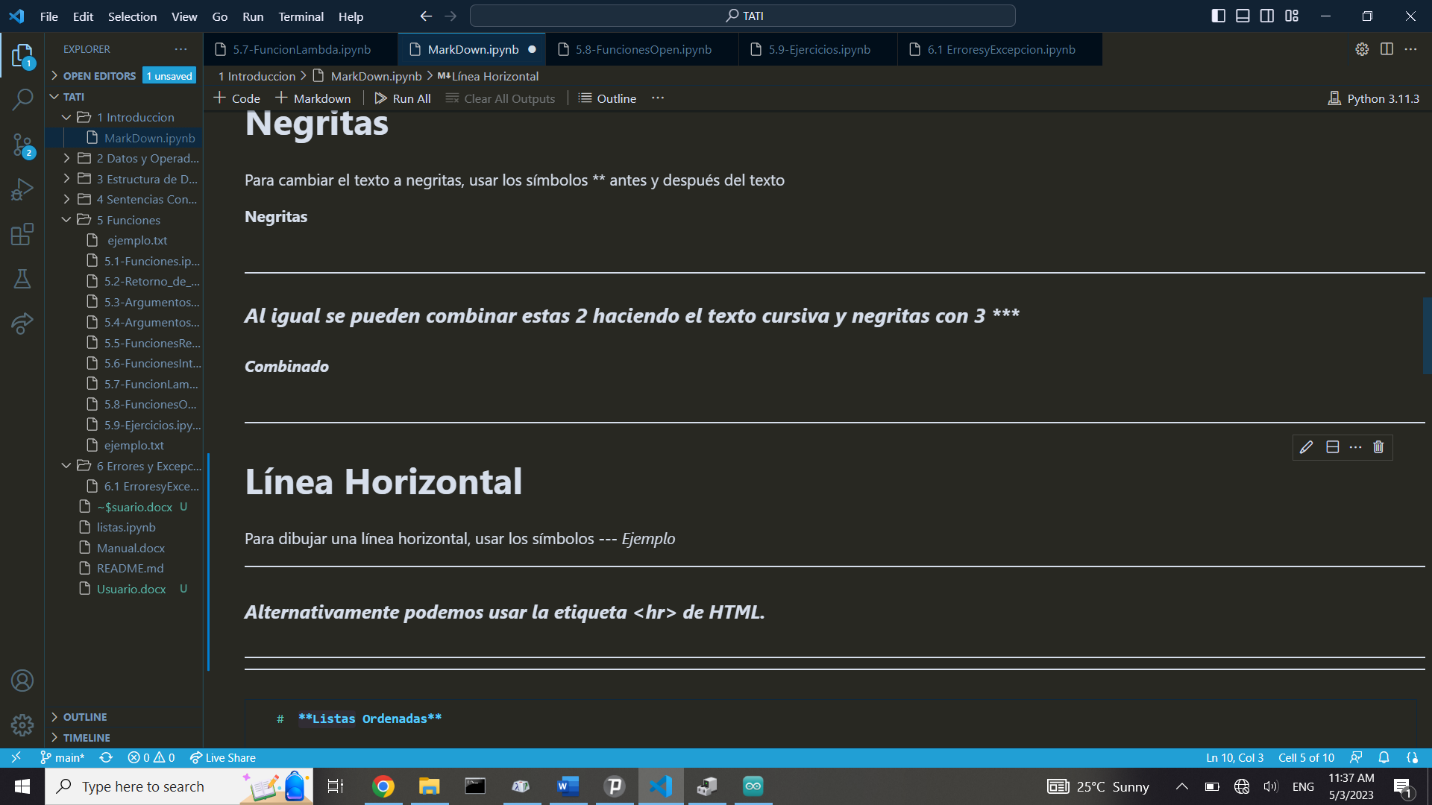
### Línea Horizontal

Para dibujar una línea horizontal, usar los símbolos **\-\-\-**

Ejemplo

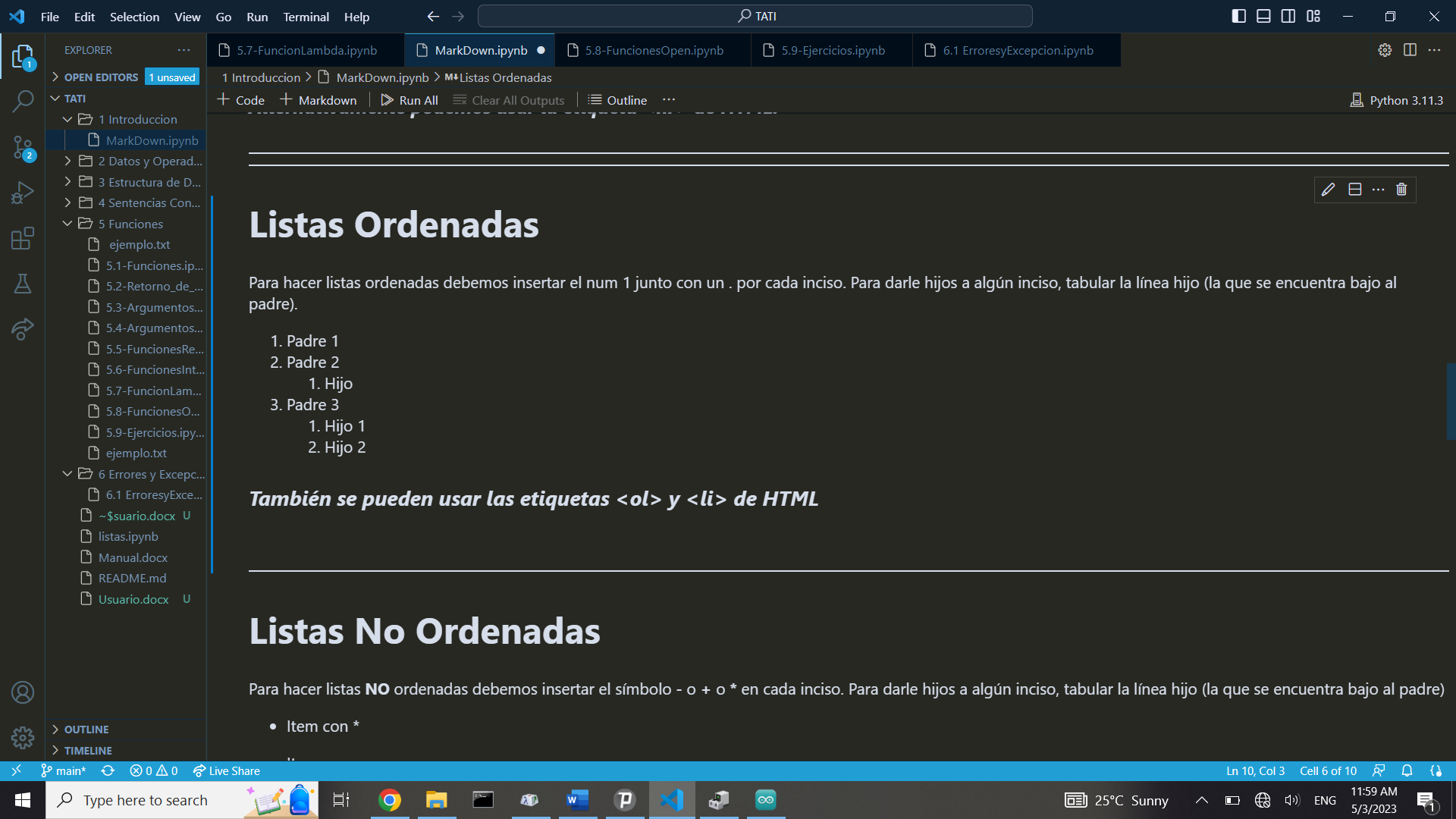
 ---

Alternativamente podemos usar la etiqueta \<hr> de HTML

<hr>

### Listas Ordenadas

Para hacer listas ordenadas debemos insertar el numero 1 junto con un \. por cada inciso. Para darle hijos a algún inciso, tabular la línea hijo (la que se encuentra bajo al padre).

1. Padre 1
2. Padre 2

1. Hijo

1. Padre 3

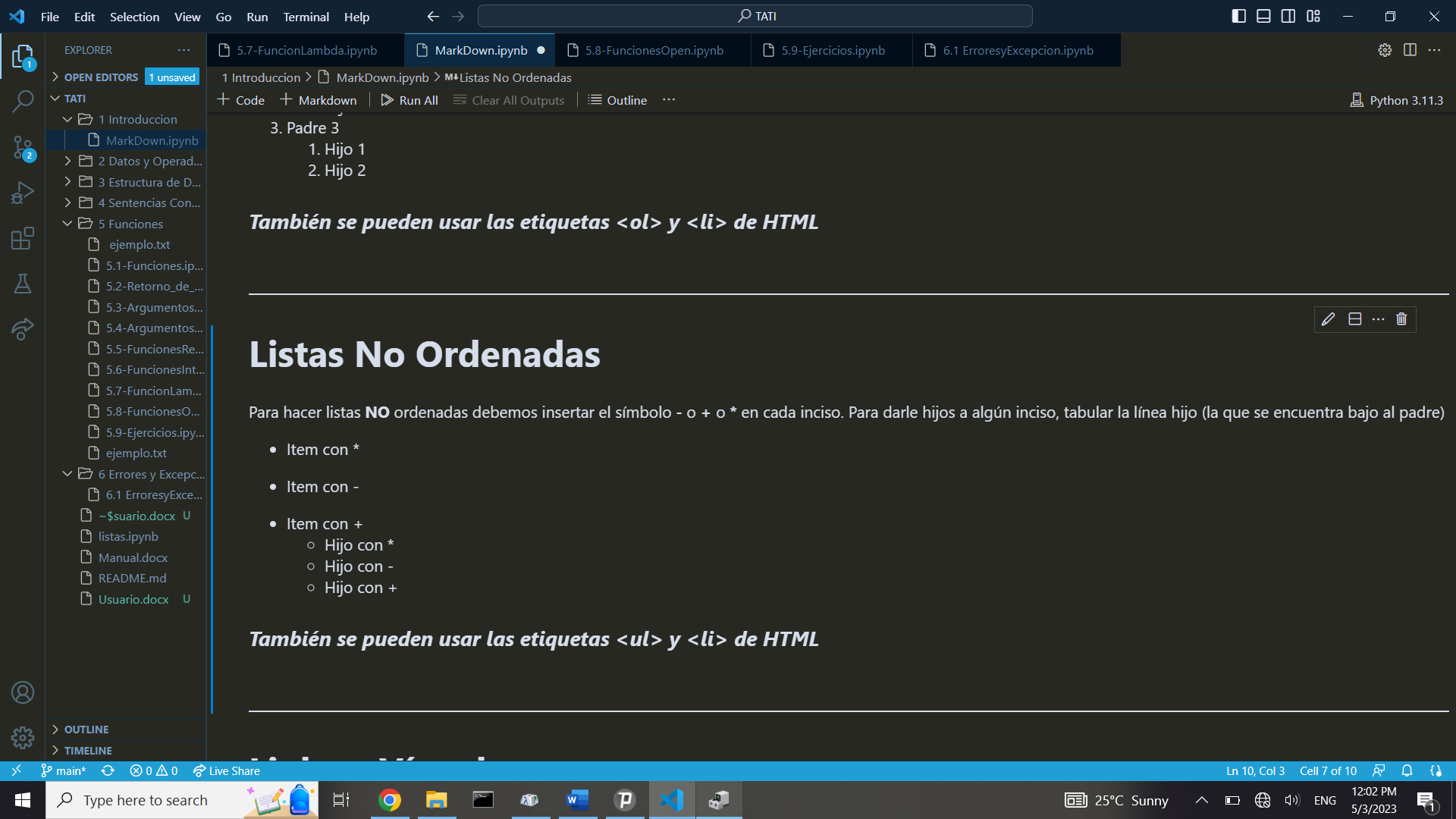
1. Hijo 1

1. Hijo 2

También se pueden usar las etiquetas \<ol> y \<li> de HTML

### Listas No Ordenadas

Para hacer listas no ordenadas debemos insertar el símbolo \***-**\* o \***+**\* o \* en cada inciso. Para darle hijos a algún inciso, tabular la línea hijo (la que se encuentra bajo al padre)

\* Ítem con \\*

- Ítem con \-

+ Ítem con \+

\* Hijo con \\*

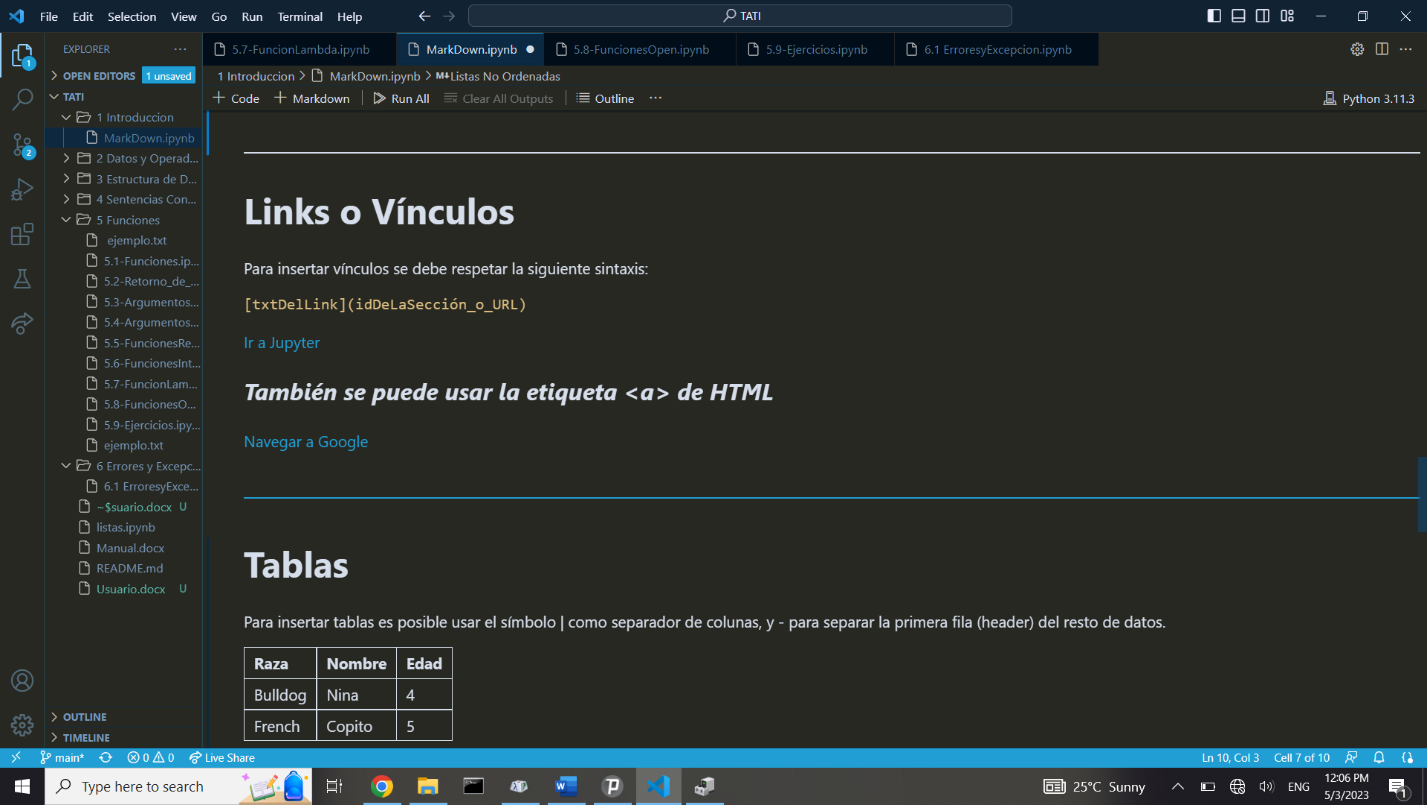
- Hijo con \-

+ Hijo con \+

También se pueden usar las etiquetas \<ul> y \<li> de HTML

### Links o Vínculos

Para insertar vínculos se debe respetar la siguiente sintaxis:

```

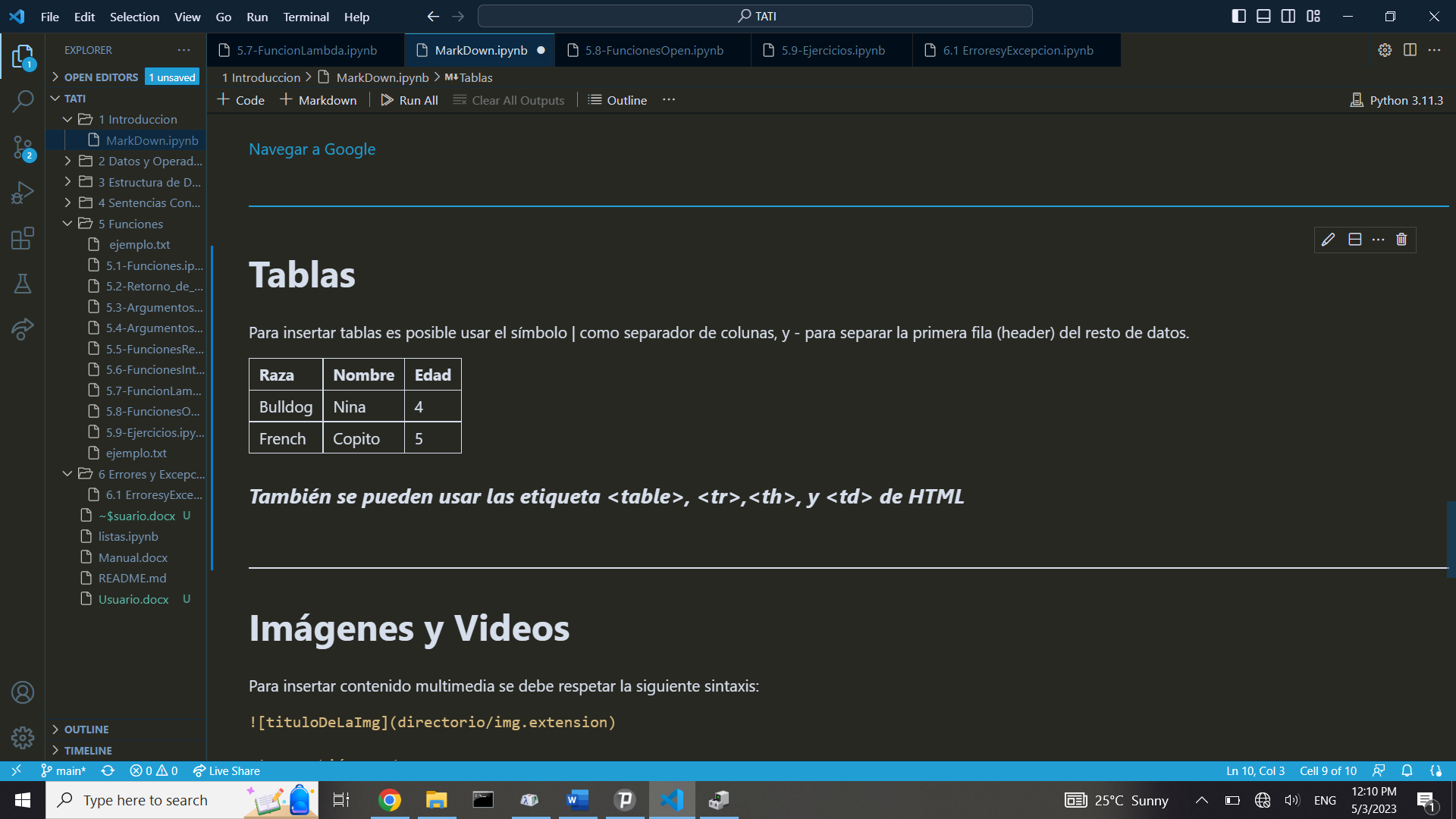
[txtDelLink](idDeLaSección\_o\_URL) ```

[Ir a Jupyter](https://jupyter.org/)

<a href="https://www.google.com">Navegar a Google<a>

### Tablas

Para insertar tablas es posible usar el símbolo | como separador de colunas, y \- para separar la primera fila (header) del resto de datos.

|Raza|Nombre|Edad|

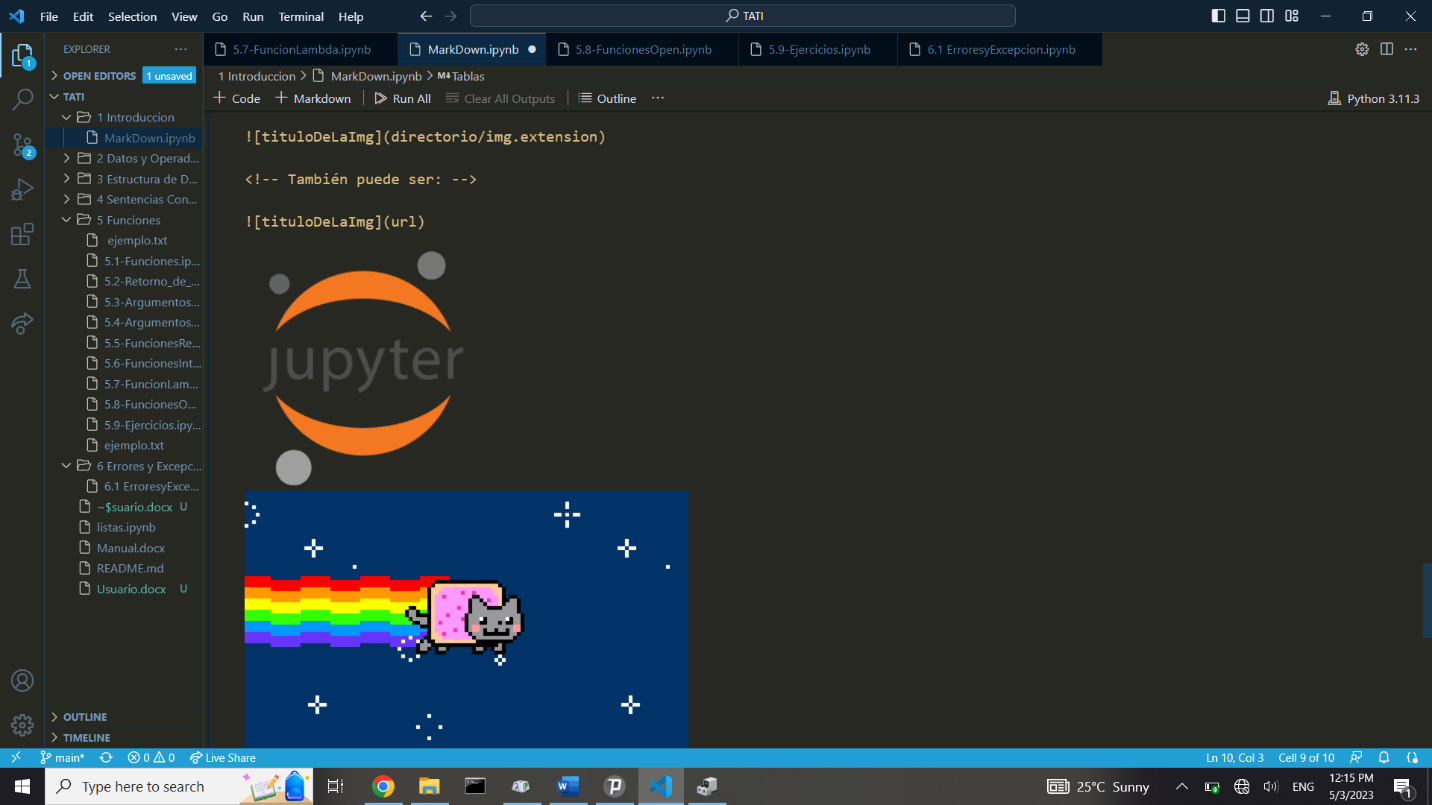
|----|------|----|

|Bulldog|Nina|4|

|French|Copito|5|

También se pueden usar las etiqueta \<table>, \<tr>,\<th>, y \<td> de HTML

### Imágenes y Videos



Para insertar contenido multimedia se debe respetar la siguiente sintaxis:

```

![tituloDeLaImg](directorio/img.extension)

![tituloDeLaImg](url)

```

![Jupyter](https://cf.appdrag.com/dashboard-openvm-clo-b2d42c/uploads/Jupyter-Notebook-EF5w-udy4.png)

<br>![Video](https://i0.wp.com/www.printmag.com/wp-content/uploads/2021/02/4cbe8d\_f1ed2800a49649848102c68fc5a66e53mv2.gif?fit=476%2C280&ssl=1)

También se puede usar la etiqueta \<img> de HTML

**DE MANERA NATIVA MARKDOWN NO CUENTA CON REDIMENSIONADO DE IMAGEN.**

# Datos y Operadores

## Variables

Python es un lenguaje de tipado dinámico y no necesita que se declare el tipo de una variable. El nombre de cada variable debe cumplir con lo siguiente:

- Solo puede incluir caracteres alfanuméricos y guiones bajos (\_)

- No puede contener espacio en blanco

- No puede empezar con un número

- En Python, el nombre de las variables es sensible a mayúsculas y minúsculas

- Existen palabras reservadas que no pueden ser utilizadas para nombrar variables

**El símbolo de igualdad (=) se utiliza para asignar valor a variables**

El conjunto de un identificador, operador y un valor se le conoce como expresión.

x *=* 30

nombre *=* 'Python para principiantes'

x

30

nombre

'Phyton para principiantes'

En Python, los tipos primitivos son: números, cadenas de caracteres (string), booleanos, tuplas, listas y diccionarios.



### Métodos

Son funciones de clase, es decir, que cada tipo de variable incluye unas funciones asociadas a ella desde su definición.

x.\_\_add\_\_(10)

70

diccionario *=* {'uno’: x, 'dos' : x.\_\_add\_\_(10)}

diccionario

{'uno': 30, 'dos': 40}

Al ser una variable tipo string, colocando. title llama al valor;

print(*f*"{nombre.title()} cuesta {diccionario['uno']} pesos \n y el otro libro {diccionario['dos']} pesos ")

Phyton Para Principiantes cuesta 30 pesos

y el otro libro 40 pesos

Se pueden revisar los métodos disponibles de una variable utilizando la función dir()

dir(x)

Output exceeds the [size limit](command:workbench.action.openSettings?%5B%22notebook.output.textLineLimit%22%5D). Open the full output data [in a text editor](command:workbench.action.openLargeOutput?103072a5-c096-4543-8714-044021582dd7)

['\_\_abs\_\_', '\_\_add\_\_', '\_\_and\_\_', '\_\_bool\_\_', '\_\_ceil\_\_', '\_\_class\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_divmod\_\_', '\_\_doc\_\_',

'\_\_eq\_\_', '\_\_float\_\_', '\_\_floor\_\_', '\_\_floordiv\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getnewargs\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_index\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_int\_\_',... 'from\_bytes', 'imag', 'numerator', 'real', 'to\_bytes']

La función help() nos muestra la información completa de cada variable

help(x)

Help on int object:

class int(object) | int([x]) -> integer | int(x, base=10) -> integer | | Convert a number or string to an integer, or return 0 if no arguments | are given. If x is a number, return x.\_\_int\_\_(). For floating point | numbers, this truncates towards zero. | | If x is not a number or if base is given, then x must be a string, | bytes, or bytearray instance representing an integer literal in the | given base. The literal can be preceded by '+' or '-' and be surrounded | by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. | Base 0 means to interpret the base from the string as an integer literal. | >>> int('0b100', base=0) | 4 | | Built-in subclasses: | bool | | Methods defined here: | | \_\_abs\_\_(self, /) | abs(self)... | | real | the real part of a complex number| Methods defined here: | | \_\_abs\_\_(self, /) | abs(self)... | | real | the real part of a complex number



## Números

Podemos asignarles valores numéricos a las variables. Python tiene 4 tipos de variables para almacenarlos.

1 *+* 2

3

2 *\** 5

10

### INT (números enteros)

x *=* 3

y *=* x *+* x.\_\_add\_\_(5)

y

11

type(x)

int

x2 *=* x.\_\_add\_\_(60)

x2

63

No se puede realizar una suma de los tipos de números int más float, pero si se puede de float más int.

x *=* 1

x2 *=* x.\_\_add\_\_(60.35)

x2

NotImplemented

f *=* 60.35

x2 *=* f.\_\_add\_\_(x)

x2

61.35



### FLOAT (números decimales)

Estos no tienen límite de decimales.

y *=* 4.5

y

4.5

type(y)

float



### COMPLEX (números complejos)

Van en formato (A + Bj) donde la unidad imaginaria está representada por "j".

c *=* 1 *+* 2*j*

c, type(c)

((1+2j), complex)

parte\_real *=* c.real

parte\_imag *=* c.imag

parte\_real, parte\_imag

(1.0, 2.0)

## Textos o Cadenas de Caracteres

Secuencias de caracteres dentro de comillas. En Python las variables de Strings se pueden declarar indistintamente usando comillas simples (' ') o dobles (" ").

"Esto es un texto"

'Esto es un texto'

'Este es otro texto'

'Este es otro texto'

nombre *=* "Miriam"

carrera *=* 'ISC'

type(nombre)

str

**PRINT()**

Esta función permite mostrar en pantalla el valor de la expresión dentro de los paréntesis() y editar su presentación con tabulaciones y saltos de línea.

print("Una cadena \tseparada por una tabulación.")

print("Una cadena en linea \nOtra cadena en linea")

Una cadena separada por una tabulación.

Una cadena en linea

Otra cadena en línea

Para que esta función nos muestre una cadena sin procesar debemos indicarle que la cadena es de tipo crudo, agregando la letra r antes de abrir comillas

print ("c\nombre\nombre\_2")

print(*r*"c\nombre\nombre\_2")

c

ombre

ombre\_2

c\nombre\nombre\_2

También podemos indicarle que deseamos mostrar el texto en varias líneas de diferente forma.

print("""Línea 1

Línea 2

Línea 3

y continuará...""")

Línea 1

Línea 2

Línea 3

y continuará...

## Booleanos

Un booleano es el tipo de dato lógico. Solo tiene dos valores: Verdadero o Falso.

1 *+* 1

1 *+* 1 *==* 2

True

1 *+* 1 *==* 3

False

variableT *=* True

variableF *=* False

variableT, variableF, type(variableF)

(True, False, bool)

True *and* True

True

(2 *==* 1 *+* 1 ) *and* (3 *==* 1 *+* 2 )

True

True *and* False

False

(3 *==* 1 *+* 1 ) *and* (3 *==* 1 *+* 2 )

False

## Comprobación de Tipos

print(*-*1, type(*-*1))

print(3.141, type(3.141))

print ("hola", type("hola"))

print (True, type(True))

print (None, type(None))

-1 <class 'int'>

3.141 <class 'float'>

hola <class 'str'>

True <class 'bool'>

None <class 'NoneType'>

**Conversión de Tipos**

x *=* 2

print(type(x))

<class 'int'>

nu\_x *=* str(x)

print(type(nu\_x))

print (nu\_x)

<class 'str'>

2

También se puede realizar la conversión sin asignar el valor a una variable.

print(int(2.1))

2

print("el entero de True es", int(True), "y el otro es", int(False))

el entero de True es 1 y el otro es 0

print(int('5'))

print(str(2))

print(str(True), str(False))

print(str(None))

5

2

True False

None

## Operadores

Los operadores aritméticos sirven principalmente para los tipos de datos numéricos, sin embargo, también pueden ser utilizados con otros tipos de datos como listas o cadenas de caracteres.

### Adición

Para las cadenas de caracteres, tuplas y listas se seguirá el formato **(a + b)**

8 *+* 30.5

38.5

"Por mi culpa" *+* " no tenemos amigos"

'Por mi culpa no tenemos amigos'



### **Sustracción**

Solo se utiliza con números el formato

49 *–* 30

19

## Multiplicación

Para las cadenas de caracteres, tuplas y listas se seguirá el formato **(a \* b)**

30 *\** 6

180

Para la multiplicación de caracteres, el resultado sera el texto el número de veces multiplicada:

"Nana" *\** 15

## División

Al igual que la sustracción solo se utiliza con números el formato **(a / b)**

26 */* 5

5.2

## División Entera Truncada

Da como resultado un valor tipo entero. Este trunca(redondea hacia abajo) los valores decimales y sigue el formato **(a // b)**

26 *//* 5

5

## Módulo o Residuo de la División

Sigue el formato (a % b), el resultado de este es lo que sobra en la división, sin tener que llegar a los decimales

26 *%* 5

1

## Exponenciación

Sigue el formato **(a\*\*b)**

4 2

16

**Operadores de Asignación (Acumuladores)**

Nos facilitan escribir el codigo. Por ejemplo:

* =
* -=
* +=
* /=
* \*=
* %=
* //=
* =

edad *=* 10

edad *=* edad *-* 3

edad

7

edad *=* 10

edad *-=* 4

edad

6

edad *+=* 10

edad

16

edad */=* 4

edad

4.0

edad *\*=* 4

edad

16.0

edad *%=* 7

edad

0.0

## Operadores Lógicos

Los operadores lógicos siguen las reglas de lógica proposicional, permiten juntar (comparar) 2 elementos lógicos con elementos relacionales

### AND

True *and* False

True *and* True

False *and* False

False

### OR

True *or* False

True *or* True

False *or* False

False

### NOT

*not* True

*not* False

True

**Operadores de Comparación**

Para comparar igualdad se utiliza **==** sin embargo podemos comparar desde el momento que se está imprimiendo el mensaje.

print(" ¿3 es igual a 3? " , 2 *==* 2)

3 es igual a 3 True

Para comparar la desigualdad utilizamos el operador !=

print("¿3 es diferente a 2? " , 3 *!=* 2)

¿3 es diferente a 2? True

Para comparar magnitudes se utilizan operadores como **<, >,>=,<=**

print ("21 es mayor que 20" , 21*>*20)

print ("21 es menor que 20" , 21*<*20)

print ("21 es mayor o igual que 20" , 21*>=*20)

print ("21 es menor o igual que 20" , 21*<=*20)

21 es mayor que 20 True

21 es menor que 20 False

21 es mayor o igual que 20 True

21 es menor o igual que 20 False

Estos operadores de comparación, tambien se pueden aplicar a las cadenas de caracteres. Estos siguen el estándar ASCII.

print ("l"*<*"m","\n","p"*>*"e" )

True

True

print ("casa"*>*"carro")

True

## Lectura de teclado

La función input() permite almacenar un valor ingresado con el teclado en una variable

valor *=* input("Hola ¿cómo estás?")

valor

''

Esta función almacena todo como variable tipo str. Pero se puede convertir en tipo

valor\_num*=*float(input("Escribe un número"))

print(valor\_num)

type(valor\_num)

2.4

float

valor\_num*=*int(input("Escribe un número"))

print(valor\_num)

type(valor\_num)

30

int



## Formateo

"{} es el primero y {} es el segundo".format(1,2)

'1 es el primero y 2 es el segundo'

"Alteramos el orden y el {1} aparece al principio, luego el {0}".format('Uno',"Dos")

'Alteramos el orden y el Dos aparece al principio, luego el Uno'

Se pueden aplicar métodos también

nombre *=* "Python para principiantes"

x *=* .76

print(*f*"El valor de {nombre} es", x , "dolares")

El valor de Phyton para principiantes es 0.76 dolares

diccionario *=* {'uno': x, 'dos':x.\_\_add\_\_(10)}

print(*f*'{nombre} cuesta {diccionario["uno"]} pesos y el otro libro cuesta {diccionario["dos"]} pesos')

Phyton para principiantes cuesta 0.76 pesos y el otro libro cuesta 10.76 pesos



## [Ejercicios\2.10-Ejercicios.ipynb](file:///C:\Users\Flor\Documents\Manual%20TATI\Ejercicios\2.10-Ejercicios.ipynb)



# Estructura de datos

## Índices y Slicing



### Índices

Los índices son números ordenados asociados a las variables que nos permiten identificar los elementos por su posición.

Por ejemplo, cada elemento de cadena tiene su propio índice.

texto *=*"Cadena de texto"

El primer elemento siempre tiene el índice 0 de manera predeterminada

print(texto[0],texto[1],texto[2],texto[3],texto[4],texto[5],texto[6])

C a d e n a

También a causa de esto, el índice del último elemento se diferencia del numero total de elementos por unidad.

[[texto[0],[texto[1],...[texto[-3],[texto[-2],[texto[-1]]

print(texto[10],texto[11],texto[12],texto[13],texto[14])

t e x t o

Para cada elemento también existe un índice inverso que realiza una numeración desde el último elemento hasta el primero, comenzando la cuenta con -1.

*# -5 \ -4 \ -3 \ -2 \ -1*

*#  T    E    X    T   O*

*# 10 \ 11 \ 12 \ 13 \ 14*

print(texto[*-*5],texto[*-*4],texto[*-*3],texto[*-*2],texto[*-*1])

### Slicing

Gracias a los índices podemos seleccionar una secuencia de elementos de una variable.

La secuencia de elementos lo definimos con el formato [a:b]. Esto es un rango de valores, el cual, nos devuelve por slicing desde el elemento a hasta el elemento previo al b dando por consecuencia que b se considere excluido del rango seleccionado.

Cuando un rango de valores excluye alguno de sus límites (siendo al límite inferior y b el límite superior), se dice que el rango es abierto:

* Abierto por la derecha(se excluye b del rango)
* Abierto por la izquierda (se excluye a del rango)

texto[0:6]

'Cadena'

texto[*-*5:*-*1]

'text'

Podemos omitir especificar alguno de los limites:

* Si omitimos a ([:b]), el rango empezará la secuencia desde el índice 0 (es igual a escribir[0:b])
* Si omitimos b ([a:]), el rango terminará la secuencia con el último elemento (es igual a escribir[a:(n-1)])

print(texto[*-*5:*-*1])

print(texto[*-*5:])

print(texto[0:6])

print(texto[:6])

text

texto

Cadena

Cadena

## Listas

Son un tipo de dato compuesto de varios elementos, estos pueden ser caracteres, numéricos o inclusive de otros tipos. Cada elemento de la lista es accesible gracias a su índice.

lista\_mandado *=* ["Huevos", "Leche","Queso","jamón"]

lista\_precios *=* ["Huevos",1.5, "Leche",2,"Queso",3.5,"Jamon",5]

type(lista\_mandado)

list

print(*f*"""Los {lista\_precios[0]} cuestan {lista\_precios[1]}, la {lista\_precios[2]} cuesta {lista\_precios[3]},

mientras que el {lista\_precios[4]} cuesta {lista\_precios[5]} y el {lista\_precios[6]} cuesta {lista\_precios[7]}""" )

Los Huevos cuestan 1.5, la Leche cuesta 2,

mientras que el Queso cuesta 3.5 y el jamón cuesta 5

**Mutabilidad**

Las listas tienen la característica de permitirnos modificar el valor de alguno de sus elementos. Esto se llama mutabilidad y es importante distinguir entre objetos mutables, como las listas y otros objetos inmutables.

lista\_de\_precios *=* [10, 20, 30]

lista\_de\_precios[1] *=* 50

print(lista\_de\_precios)

[10, 50, 30]

lista *=* list(range(1, 12, 2))

print(lista)

lista.append(8)

print(lista)

print(lista *+* [2,3])

print(len(lista))

print(2 *in* lista)

lista[:2] *=* [2, 2]

print(lista)

lista2 *=* [3, 6, 7, 8]

lista3 *=* list(range(6, 30, 3))

print([lista, lista2, lista3])

[1, 3, 5, 7, 9, 11]

[1, 3, 5, 7, 9, 11, 8]

[1, 3, 5, 7, 9, 11, 8, 2, 3]

7

False

[2, 2, 5, 7, 9, 11, 8]

[[2, 2, 5, 7, 9, 11, 8], [3, 6, 7, 8], [6, 9, 12, 15, 18, 21, 24, 27]]

## Tuplas

Es una secuencia de valores(muy parecida a listas).Son valores almacenados en una tupla pueden ser de cualquier tipo y también están indexados.

tupla *=* ('uno','dos','tres')

tupla\_mix *=* ([1,2,3],'a',23,tupla)

lista *=* list(tupla)

lista[0] *=* "cero"

**Métodos**

El método index() nos devuelve la posición en la que se encuentra el valor especificado dentro de la tupla, si el valor no es encontrado devolverá un error.

tupla.index("dos")

1

El método count() nos devuelve cuantas veces se encuentre presente el valor especificado dentro de la tupla, si el valor no es encontrado devuelve 0

tupla\_mix.count(23)

1

## Conjuntos

Son colecciones de datos que facilitan ciertas operaciones ya que solo contienen valores únicos.

Las diferencias de las tupla y listas son:

* Cada elemento debe ser unico.
* Los elementos no estan ordenados.
* Deben de estar entre llaves{} los elementos.

conjunto\_vacio *=* set()

conjunto *=* {1,2,3}

print(type(conjunto\_vacio),conjunto)

<class 'set'> {1, 2, 3}

**Métodos**

El método add() nos permite agregar elementos al conjunto

conjunto.add('Z')

conjunto.add('G')

conjunto.add(1)

conjunto.add(3)

conjunto.add(5)

conjunto.add('A')

conjunto.add('B')

conjunto

{1, 2, 3, 5, 'A', 'B', 'D', 'G', 'Z', 'd'}

Utilizando el método **In** o **Not In** se puede saber si existe el elemento buscado en el conjunto.

frutas *=* {"mango","platano","manzana"}

print('mango' *in* frutas)

print('mango' *not in* frutas)

True

False

Un conjunto elimina automáticamente los elementos duplicados.

numeros*=* {1,2,2,3,3}

numeros

{1, 2, 3}

Gracias a esta propiedad se puede convertir una lista a conjunto para eliminar duplicados.

numeros *=* [1,2,3,4,3,4,1]

print (numeros)

num\_conj *=* set(numeros)

print(num\_conj)

[1, 2, 3, 4, 3, 4, 1]

{1, 2, 3, 4}

no\_dupli *=* list(num\_conj)

print(no\_dupli)

[1, 2, 3, 4]

Cuando se aplica la conversión a una cadena, el conjunto resultante incluye solo las letras únicas de esta.

cadena *=* "le gustaba cenar un exquisito sándwich de jamón con zumo de piña y vodka fría."

set(cadena)

Output exceeds the [size limit](command:workbench.action.openSettings?%5B%22notebook.output.textLineLimit%22%5D). Open the full output data [in a text editor](command:workbench.action.openLargeOutput?bf278dcc-4613-478b-ac45-1cf69194cefd)

{' ','.', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w',... 'z', 'á', 'í', 'ñ', 'ó'}

## Diccionarios

Son colecciones de datos semejantes a las listas solo que tienen un clave : valor:

* funcionan como indice de cada elemento (no solo son numeros).

numeros *=* {1:'uno', 2:'dos', 3:'tres' }

datos *=* {"numero":"matricula","name":"nombre","lastname":"apellido"}

print(numeros)

{1: 'uno', 2: 'dos', 3: 'tres'}

Podemos escoger entre sus elementos utilizando claves.

numeros[1]

'uno'

También podemos modificar los valores de sus elementos y realizar operaciones con ellos.

numeros[2]*=*'two'

numeros

{1: 'uno', 2: 'two', 3: 'tres'}

edades *=* {

    'Miriam':21,

    'Luis':22,

    'Romina':3

}

edades['Romina'] *+=* 3

print(edades)

{'Miriam': 21, 'Luis': 22, 'Romina': 6}

edades['Luis'] *+* edades['Miriam']

43

**Métodos**

A través de un bucle **For** podemos recorrer los elementos de un diccionario.

*for* clave *in* edades:

    print(clave)

Miriam

Luis

Romina

Fernando

*for* clave, valores *in* edades.items():

    print( clave, "tiene", valores, "años")

Miriam tiene 21 años

Luis tiene 22 años

Romina tiene 6 años

Fernando tiene 28 años

El método **del()** nos permite eliminar valores de un diccionario por medio de su clave.

print(numeros)

*del*(numeros[1])

print(numeros)

{1: 'uno', 2: 'dos', 3: 'tres'}

{2: 'dos', 3: 'tres'}



## [Ejercicios\3.6-Ejercicios.ipynb](file:///C:\Users\Flor\Documents\Manual%20TATI\Ejercicios\3.6-Ejercicios.ipynb)

# Sentencias Condicionales

## If/Else/Elif:

En Python existen las sentencias condicionales, estas nos permiten dividir el flujo de un programa en diferentes caminos.

Para realizar esto, se define un bloque de instrucciones que se ejecutarán solo si la condición previa se cumple, es decir, que sea verdadera (True).

x *=* 5

*if* (x*==*5):

    print("x vale 5")

x vale 5

Podemos escribir varios \*if\* anidados respetando la identación.

a *=* 14

b *=* 6

*if* a*==*14:

    print("{} es a".format(a))

*if* b*==*6:

        print("{} es b".format(b))

14 es a

6 es b

También se pueden incluir en estas sentencias operadores lógicos para aumentar su eficiencia y legibilidad.

*if* a *==* 14 *and* b *==* 6:

    print("{} es a y {} es b".format(a,b))

14 es a y 6 es b

**Else y Elif**

Else (si no) se encadena al final del bloque para abrir una nueva lista de instrucciones. Este nos permite definir el camino que se ejecuta si la condición original no se cumple.

Elif (si no si) establece una nueva condición que se encadena a otro if o elif cuya condición resulto False(no se cumplió).

Esto nos permite establecer múltiples condiciones para lidiar con las posibles entradas que el programa pueda recibir.

z *=* 11

*if* 10 *%* 2:

    print(z,"es numero par")

*else*:

    print(z,"es numero impar")

*# Respuestas Din'amicas*

texto *=* input(¿”Que deseas a hacer?:\n").upper()

*if* texto *==* "SALUDAR":

    print("Buen Dia")

*elif* texto *==* "SALIR":

    print("Hasta Luego")

*else*:

    print("No entendí")

*# Clasificador*

puntaje *=* float(input("Introduce la calificación:\n"))

*if* puntaje *>=*9:

    print("Excelente")

*elif* puntaje *>=*7:

    print("Buen desempeño")

*elif* puntaje *>=*5:

    print("Aprobado")

*else*:

    print("Puntaje reprobado")

## Bucle While

Un bucle es una secuencia que ejecuta una acción reiteradas veces hasta que la condición asignada al bucle deja de cumplirse. Cada nueva ejecución de la secuencia se denomina iteración.

La sentencia While (Mientras) repite las operaciones dentro de un bloque mientras la condición lógica que evalúa es True (verdadera).

El programador debe planificar un momento en que la condición sea False y el bucle se detenga. De otra forma se vuelve un bucle infinito.

c *=* 0

*while* c *<=* 5:

    print("Contador vale: ", c)

    c *+=* 1

*else*:

    print("Se realizaron {} iteraciones,".format(c))

**BREAK**

Con este comando detenemos el bucle en cualquier momento.

contador *=* 0

*while* contador *<=* 5:

    contador *+=* 1

*if* contador *==* 4:

        print("Rompemos el bucle cuando c vale ", contador)

*break*

    print("c vale ", contador)

*else*:

    print("Se realizaron {} iteraciones.".format(contador))

**CONTINUE**

Con este comando saltamos a la siguiente iteración sin romper el bucle.

c *=* 0

*while* c *<=* 5:

    c *+=* 1

*if* c *==* 2 *or* c *==* 4:

        print(*f*"""Ya que c vale(c) saltamos esta iteración y continuamos con la siguiente.""")

*continue*

    print("c vale ", c)

*else*:

    print("Se realizaron {} iteraciones.".format(c))

*# Menu de Usuario*

print("Bienvenido")

*while*(True):

    opcion *=* input(*f*"""Escoge una opcion

    1) Saludar

    2) Multiplicar 2 numeros

    3) Salir""")

    print("Escogio el",opcion, "de opcion")

*if* opcion *==* "1":

        print("Hola")

*elif* opcion *==* "2":

        n1 *=* float(input("Escribe un número}}"))

        n2 *=* float(input("Escribe otro número"))

        print("La multiplicación de",n1, "por", n2, "es", n1*\**n2)

*elif* opcion *==* "3":

        print("Hasta Luego")

*break*

*else*:

        print("Opción incorrecta")

## Bucle For

For es un bucle que se repite un número determinado de veces.

Los objetos iterables pueden ser listas, cadenas de texto,etc...

print("Bucle For")

*for* i *in* nums:

    print(i)

Bucle For

1

2

3

4

5

6

7

8

9

10

**NOTA range() genera la secuencia de números enteros definida por el usuario.**

*for* i *in* range(10):

    print(i)

0

1

2

3

4

5

6

7

8

9

## [Ejercicios\4.4-Ejercicios.ipynb](file:///C:\Users\Flor\Documents\Manual%20TATI\Ejercicios\4.4-Ejercicios.ipynb)

# Funciones



## Definir una Función

Las funciones en Python, al igual que en el resto de los lenguajes de programación, permiten reutilizar código. Una función podría considerarse como una variable que encierra un conjunto de instrucciones. Por lo que tanto al llamar a una función lo que haces es ordenar el programa para que ejecute un conjunto de instrucciones.

* Se pueden crear en cualquier momento del programa
* Su palabra reservada es def
* Seguido de def viene el nombre de la función y entre paréntesis los argumentos de entrada
* No es obligatorio que la función devuelva in valor, aunque es posible hacerlo utilizando la palabra reservada return
* Las variables declaradas dentro de una función son locales a esa función
* Para declarar una variable dentro de una función, pero poder acceder a ella desde la misma función, esta se puede declarar como global(utilizando la palabra \*global\*)

*def* *saludar*():

    print(*f*"Hola")

saludar()

Hola

Scope

Cada nombre(los identificadores de variables funciones, objetos,etc) tienen su \*scope\* (alcance).Este define el área de un programa en el cual puedes acceder a un nombre.

*def* *prueba*():

    n*=*10=

prueba()

n *=* 5

*def* *prueba2*():

    print(n)

prueba2()

*def* *prueba3*():

*global* g

    g *=* 10

prueba3()

5

Una función va a dar prioridad a sus variables locales que las globales.

* 1. Retorno de Valores

Para retornar valores dentro de una función se puede utilizar la palabra reservada **return**

*def* *funcion*():

*return* "Mensaje es el retorno"

funcion()

'Mensaje es el retorno'

*def* *test*():

*return* "Mensaje",1994,[1,2,3]

test()

type(test())

tuple

txt,year,numList *=* test()

print(txt)

print(year)

print(numList)

Mensaje

1994

[1, 2, 3]

## Argumentos y Parámetros

Los argumentos son los valores que pasamos para utilizarse dentro de una función(num1,num2).

*def* *resta*(num1,num2):

*return* num1*-*num2

resta(4,2)

2

Se puede definir el valor predeterminado para nuestros parámetros.

*def* *resta2*(a*=*None,b*=*None):

*if* (a*==*None *or* b*==*None):

        print("Error, envia valores")

*return*

*else*:

*return* a *-* b

resta2(3,2)

resta2()

Error, envia valores

**Argumentos por valor**

*def* *duplicar*(num):

*return* num *\** 2

n *=* 10

duplicar(n),n

**Argumentos por Referencia**

*def* *duplicar2*(nums):

*for* i,j *in* enumerate(nums):

        nums[i] *\*=* 2

*return* nums

notas\_a *=* list(range(5))

notas\_b *=* duplicar2(notas\_a)

notas\_a,notas\_b

*# Una lista no se pasará como copia dentro de una función, mas bien será remplazada*

*# en las operaciones de dicha función.*

([0, 2, 4, 6, 8], [0, 2, 4, 6, 8])

**Nota: Para enviar una copia de una lista a una función se utiliza nombre[:]**

notas\_c *=* list(range(6))

notas\_d *=* duplicar2(notas\_c[:])

notas\_c, notas\_d

([0, 1, 2, 3, 4, 5], [0, 2, 4, 6, 8, 10])

* 1. Argumentos Indeterminados

Los argumentos indeterminados representan la forma de pasar una colección de parámetros desconocidos a nuestras funciones.

*def* *Argumentar*(*\**args):

    print(args)

Argumentar(5,"Joe",[1,2,3,4])

(5, 'Joe', [1, 2, 3, 4])

*def* *Nombrar*(*\*\**kwargs):

    print(kwargs)

Nombrar(id*=*5,nombre*=*"Kevin",notas*=*[100,100,70,84])

{'id': 5, 'nombre': 'Kevin', 'notas': [100, 10, 70, 84]}

*def* *super\_nominacion*(*\**args, *\*\**kwargs):

    suma *=* 0

*for* e *in* args:

        suma *+=* e

    print ("El promedio indeterminado es {}".format(suma*/*len(args)))

*for* clave *in* kwargs:

        print (clave,"\t", kwargs[clave])

super\_nominacion(10,10,7,8,id*=*5,nombre*=*"Kevin",edad*=*35, notas*=*[100,200,400])

El promedio indeterminado es 8.75

id 5

nombre Kevin

edad 35

notas [100, 200, 400]

## Funciones Recursivas

Es un tipo de función que se utiliza a sí misma en el mismo cuerpo de su definición.

*def* *cuenta\_atras*(num):

    num *-=* 1

*if* (num *>* 0):

        print (num)

        cuenta\_atras(num)

*else*:

        print(num,"Boom")

    print("Fin de la función {}".format(num))

cuenta\_atras(5)

4

3

2

1

0 Boom

Fin de la función 0

Fin de la función 1

Fin de la función 2

Fin de la función 3

Fin de la función 4



## Funciones Integradas

**Numéricas**

bin(19)

'0b10011'

print(hex(15),hex(10))

0xf 0xa

abs(*-*13)

13

pow(3,3)

27

round(5.5)

6

**Otras Funciones**

'para separar una cadena'.split("a")

['p', 'r', ' sep', 'r', 'r un', ' c', 'den', '']

"todo en mayúsculas".upper()

'TODO EN MAYÚSCULAS'

"ToDo en MinÚsculAs".lower()

'todo en minúsculas'

"maYúscUlas en lA inIcIal dE lAs pAlAbrAS".title()

'Mayúsculas En La Inicial De Las Palabras'

"maYúscUla Solo En La Primera palAbra".capitalize()

'Mayúscula solo en la primera palabra'

a,b,c,d *=* zip(["A","B","C","D"],[10,11,12,13,14])

dict(zip(["A","B","C","D"],[10,11,12,13,14]))

{'A': 10, 'B': 11, 'C': 12, 'D': 13}

## Funciones Lambda

Son conocidas como funciones anónimas, representan la manera de realizar una función sin tener que utilizar las palabras reservadas **def** ni asignar un nombre a la función creada.

La utilidad de las funciones Lambda se ve al asignarlas a una variable

doblar4 *=* *lambda* num: num *\** 2

doblar4(10)

20

## Funciones Open

Función que nos permite escribir o leer un archivo externo especificando la url donde se encuentra dicho archivo y el modo de acceso dependiendo de que acción quisiéramos realizar, algunas opciones son :

**w : acceso de escritura de un archivo nuevo**

**r : acceso de lectura de un archivo existente**

**a : acceso de escritura para agregar más elementos a un archivo existente**

*with* open(*r*'ejemplo.txt', 'w', encoding*=*'utf-8') *as* f:

    f.write('Escremritura en ficheros')

    f.write('\nOtra Linea ')

    f.write('Y la misma linea')

*with* open(*r*'ejemplo.txt', 'r', encoding*=*'utf-8') *as* f:

    print(f.read())

Escritura en ficheros

Otra Linea Y la misma linea

palabras2 *=* list()

*with* open(*r*'ejemplo.txt','r') *as* f:

*for* linea *in* f:

        term\_1 *=* linea.split()

*for* t *in* term\_1:

            palabras2.append(t)

    print(palabras2,type(palabras2))

['Escritura', 'en', 'ficheros', 'Otra', 'Linea', 'Y', 'la', 'misma', 'linea'] <class 'list'>

## [Ejercicios\5.9-Ejercicios.ipynb](file:///C:\Users\Flor\Documents\Manual%20TATI\Ejercicios\5.9-Ejercicios.ipynb)

# Errores y Excepciones

Los errores detienen la ejecución de un programa. Se pueden presentar distintos tipos de errores.

**Syntax Error**

Estos errores los podemos apreciar revisando el código, ya que surgen cuando por desliz nos equivocamos escribiéndolo.

print("hola"

**Cell In[1], line 1**

**print("hola"**

**^**

**SyntaxError:** incomplete input

**Name Error**

Estos errores surgen cuando nos equivocamos a declarar el nombre de algún objeto.

pint("hola")

**---------------------------------------------------------------------------**

**NameError** Traceback (most recent call last)

Cell **In[2], line 1**

**----> 1** pint("hola")

**NameError**: name 'pint' is not defined

**Index Error**

Esta clase de errores son indentificados por el editor antes de ser ejecutado el código pero existen errores que suelen pasar desapercibidos.

list *=* []

list.pop()

**---------------------------------------------------------------------------**

**IndexError** Traceback (most recent call last)

Cell **In[3], line 2**

1 list = []

**----> 2** list.pop()

**IndexError**: pop from empty list

**Type Error**

num *=* input("Ingresa un número")

print('el numero {} dividido da {}'.format(num,num*/*2))

**---------------------------------------------------------------------------**

**TypeError** Traceback (most recent call last)

Cell **In[4], line 2**

1 num = input("Ingresa un número")

**----> 2** print('el numero {} dividido da {}'.format(num,num/2))

**TypeError**: unsupported operand type(s) for /: 'str' and 'int'

**Zero Division Error**

num *=* int(input('ingresa un número'))

print('el numero {} dividido da {}'.format(num,num*/*0))

**---------------------------------------------------------------------------**

**ZeroDivisionError** Traceback (most recent call last)

Cell **In[5], line 2**

1 num = int(input('ingresa un número'))

**----> 2** print('el numero {} dividido da {}'.format(num,num/0))

**ZeroDivisionError**: division by zero

**Excepciones**

Son bloques de código que permiten continuar la ejecución del código aun cuando se presenta algún error.

**Try y Except**

continuar *=* True

*while* continuar:

*try*:

        num *=* int(input("Escribe un número"))

        print ('El numero {} dividido entre {} da {}'. format(num,num, num*/*num))

        continuar *=* False

*except*:

        print ("Intenta de nuevo")

*else*:

        print ("La ejecución fue exitosa")

*finally*:

        print("Fin del bucle")