



---

# **SOFTWARE PARA INGENIEROS**

## **Unidad 1 – Fundamentos de programación en Python**

## LOGRO DE APRENDIZAJE



Al finalizar la unidad, el estudiante aplica los fundamentos de la programación Python en un contexto real o simulado, demostrando una actitud analítica y organizada.

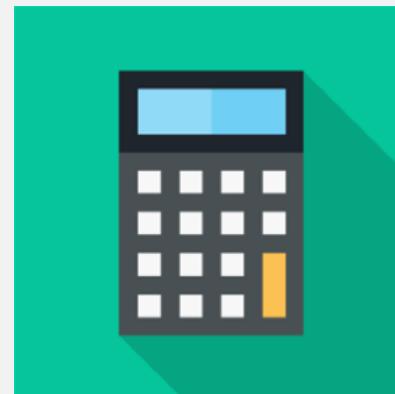
# TEMARIO



## TEMA 1

---

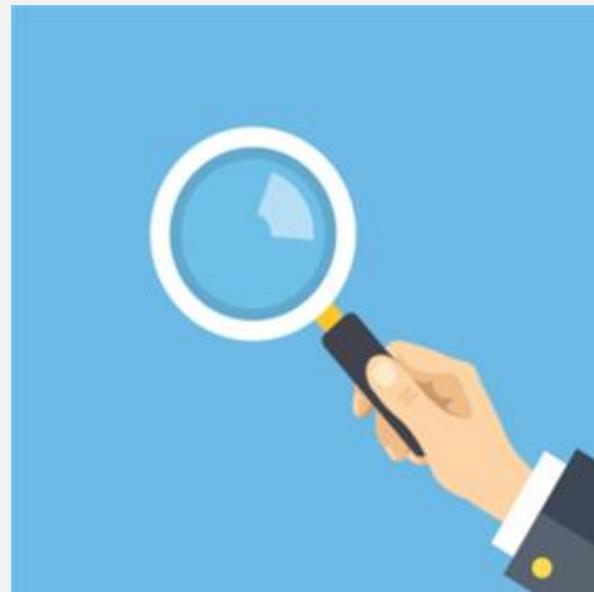
Introducción y Tipo de Datos



## TEMA 2

---

Operadores y expresiones



## TEMA 1

---

Introducción y Tipo de Datos

# 1.1 Introducción y Tipos de Datos



## Nombre de variables

Una variable puede tener nombres cortos como X e Y, o tener nombres mas descriptivos como Edad, Nombre, Volumen\_Total.

Reglas para las variables de Python:

- Una variable debe iniciar con una letra o una raya abajo ( \_ underscore character)
- Una variable no puede iniciar con numero
- Una variable puede contener solo caracteres alfa numéricos o raya abajo (A-z, 0-9, and \_ )
- Los nombres de las variables son sensibles a las mayúsculas (p.e. edad, Edad, EDAD son tres variables diferentes)

## Declaración de variables

Python como muchos lenguajes, declara la variable con el primer valor asignado a este. **NO REQUIERE UN COMANDO**

- `x = 5`
- `y = "Hello, World!"`

## Asignación de valores a múltiples variables

Python permite asignar valores a múltiples variables en una sola línea.

- `x, y, z = "Orange", "Banana", "Cherry"`
- `x = y = z = "Orange"`

# 1.1 Introducción y Tipos de Datos



## Comentarios

Python con el propósito de documentar sus códigos de línea, permite comentarlas mediante el carácter # al inicio de la línea:

- `#Este es un comentario`  
`print("Hola, mundo!")`
- `print("Hola, mundo!") #Este es un comentario`

Se pueden documentar varios códigos de línea:

- `#Este es un comentario`  
`# escrito en mas`  
`# de una línea`  
`print("Hola, mundo!")`

## 1.1 Introducción y Tipos de Datos



### Data type

Text Type:

str

Numeric Types:

int, float, complex

Sequence Types:

list, tuple, range

Mapping Type:

dict

Set Types:

set, frozenset

Boolean Type:

bool

Binary Types:

bytes, bytearray, memoryview

# 1.1 Introducción y Tipos de Datos



## Data type

Asignación de variables

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

# 1.1 Introducción y Tipos de Datos



## Data type

Para asignar a una variable un tipo de dato específico, se deben utilizar las siguientes funciones:

Example	Data Type
x = str("Hello World")	str
x = int(20)	int
x = float(20.5)	float
x = complex(1j)	complex
x = list(("apple", "banana", "cherry"))	list
x = tuple(["apple", "banana", "cherry"])	tuple
x = range(6)	range
x = dict(name="John", age=36)	dict
x = set(("apple", "banana", "cherry"))	set
x = frozenset(("apple", "banana", "cherry"))	frozenset
x = bool(5)	bool
x = bytes(5)	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

# 1.1 Introducción y Tipos de Datos



## Numbers (números)

**Enteros** – Int, or integer, es un numero entero, positivo o negativo, sin decimales, de longitud indefinida

```
x = 1  
y = 35656222554887711  
z = -3255522  
print( type(x) )  
print( type(y) )  
print( type(z) )
```

**Flotantes** – float, o numero punto flotante, positivo o negativo, contiene uno o mas decimales

```
x = 1.10  
y = 1.0e10  
z = -325.5522  
print( type(x) )  
print( type(y) )  
print( type(z) )
```

**Complejos** – Que se escriben como un j como parte imaginaria

```
x = 3 + 5j  
y = 8j  
z = 10 -3j  
print( type(x) )  
print( type(y) )  
print( type(z) )
```

### Tipo de conversiones

x = 25	# int
y = 24.85	# float
z = -3j	# complex

a = float (x)	# convierte un entero en un flotante
b = int (y)	# convierte un flotante a un entero
c = complex (x)	# convierte un entero a un complejo

## 1.2 Operadores y expresiones



### TEMA 2

---

Operadores y expresiones

## 1.2 Operadores y expresiones



- Python tiene los siguientes operadores
  - Aritméticos
  - De asignación
  - De comparación
  - Lógicos
  - Identidad
  - Miembros
  - Bitwise (Binarios)

## 1.2 Operadores y expresiones



Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	$x / y$
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor división (división entera)	$x // y$

# Operadores de asignación

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x  = 3	x = x   3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3



# Operadores de comparación



Operator	Name	Example
<code>==</code>	Equal	<code>x == y</code>
<code>!=</code>	Not equal	<code>x != y</code>
<code>&gt;</code>	Greater than	<code>x &gt; y</code>
<code>&lt;</code>	Less than	<code>x &lt; y</code>
<code>&gt;=</code>	Greater than or equal to	<code>x &gt;= y</code>
<code>&lt;=</code>	Less than or equal to	<code>x &lt;= y</code>

## 1.2 Operadores y expresiones



### Operadores lógicos

Operator	Description	Example
and	Returns True if both statements are true	<code>x &lt; 5 and x &lt; 10</code>
or	Returns True if one of the statements is true	<code>x &lt; 5 or x &lt; 4</code>
not	Reverse the result, returns False if the result is true	<code>not(x &lt; 5 and x &lt; 10)</code>

### Operadores de identidad

Los operadores de identidad se usan para comparar los objetos, no si son iguales, sino si en realidad son el mismo objeto, con la misma ubicación de memoria:

Operator	Description	Example
is	Returns True if both variables are the same object	<code>x is y</code>
is not	Returns True if both variables are not the same object	<code>x is not y</code>



# Operadores de miembros

Los operadores de membresía se utilizan para probar si una secuencia se presenta en un objeto:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

# Operadores Bitwise

Los operadores bit a bit se usan para comparar números (binarios):

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

# INSERTAR NOMBRE DE ACTIVIDAD



## PASOS DE ACTIVIDAD individual /grupal:

- Ingresar al colab personal con su cuenta gmail
  
- Copia el enunciado del problema señalado por el profesor y péguelo en el colab
  
- Desarrolle el ejercicio junto con un compañero sea en el laboratorio o en la sala grupal creada por el docente.

# Resolución de ejercicios

## ACTIVIDAD

Debe resolver los ejercicios señalados por el profesor que se encuentran en el documento:

**01 - 02.2 Fundamentos de Programación en Python  
- Guía laboratorio y Casos propuestos.pdf**

Este documento se encuentra en el aula virtual en la unidad 1



# CONCLUSIONES

**01** Python estándar ha sido desarrolladas con la arquitectura orientada a objetos.

**02** Python estándar ha sido desarrolladas con la arquitectura orientada a objetos.

**03**

**04**

# BIBLIOGRAFÍA

- Direcciones electrónicas (hipervínculos en las diapositivas)
- LUTZ, MARK (2013) Learning Python. 5th Edition. California: O'Reilly.
- RAMALHO, LUCIANO (2015) Fluent Python: Clear, Concise, and Effective Programming (inglés) 1st Edición. California: O'Reilly.
-

# **Continúa con las actividades propuestas**

---



Material producido por la  
Universidad Peruana de  
Ciencias Aplicadas

Autor:

Norman Reyes Morales

COPYRIGHT © UPC  
2020 – Todos los  
derechos reservados



---

# **SOFTWARE PARA INGENIEROS**

## **Unidad 1 – Fundamentos de programación en Python**

## LOGRO DE APRENDIZAJE



Al finalizar la unidad, el estudiante aplica los fundamentos de la programación Python en un contexto real o simulado, demostrando una actitud analítica y organizada.

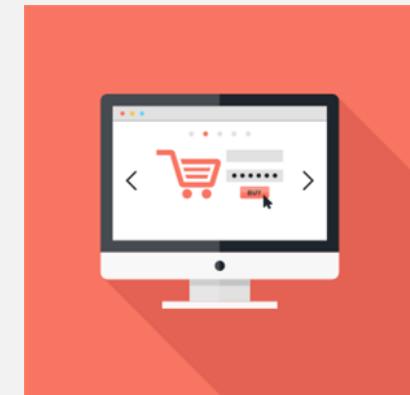
# TEMARIO



## TEMA 1

---

Introducción y Tipo  
de Datos – Cadenas y  
Listas



## TEMA 3

---

Entradas y Salidas



## TEMA 1

---

Introducción y Tipo de Datos – Cadenas y Listas

# 1.1 Introducción y Tipos de Datos

## Cadenas (string)

Las cadenas en Python, deben estar encerradas entre comillas “ ” o apóstrofos ‘ ’

“hola” es lo mismo a ‘hola’



Longitud de las cadenas en Python: función len()

x='Nada es imposible'

len(x) #el resultado es 17

Se puede usar parte de una cadena, usando **slicing [pos inicial: pos final: pasos]** la posición final no se incluye

x='Hola todos'

print(x[2:6]) #el resultado es: la t

0	1	2	3	4	5	6	7	8	9
H	O	L	A		T	O	D	O	S
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Los índices también pueden ser negativos **slicing [pos inicial: pos final:pasos]**

x='Hola todos'

print(x[-8: -2]) #el resultado es: la tod

print(x[-5:]) #el resultado es: todos

## 1.1 Introducción y Tipos de Datos

# 1.1 Introducción y Tipos de Datos



## Cadenas (string)

Las cadenas se pueden concatenar es decir unir con otras cadenas para formar otras cadenas. No se puede concatenar con números enteros o flotantes

```
x="hola "
y="mundo"
print(x+y)           #el resultado es: hola mundo
print(x+y+3)        #el resultado es: error
print((x+y)*3)      #el resultado es: hola mundohola mundohola mundo
```

Python puede comparar cadenas: `==` y `!=` (igual o diferente). La comparación se hace carácter por carácter. Si estas son iguales o de lo contrario serán diferentes

```
x="hola "
y="hola "
x==y                  #el resultado es: True (verdadero)
x="hola "
y="Hola "
x==y                  #el resultado es: False (falso)
```

Python también compara cadenas: `<` o `>` (menor o mayor). La comparación se hace carácter por carácter. Pero considerando su código ASCII (128 caracteres u 8 bits) pero ahora su nuevo estándar [Unicode \(65536 u 16 bits\)](#)

```
x="hola "
y="Hola "
x<y                  #el resultado es: False (falso)
ord('h')              #el resultado es: 104  La función ORD() devuelve la posición del carácter en la tabla UNICODE
ord('H')              #el resultado es: 72    La función chr() devuelve el carácter de la posición en la tabla UNICODE
```

## 1.1 Introducción y Tipos de Datos



### Cadenas (string)

**String Format.**- También se requieren Combinar números con letras hacer tabulados

```
x=2021  
y="Hola mundo "  
print(x+y) #el resultado es: error
```

```
x=2021  
y="Hola mundo {}"  
print(y.format(x)) #el resultado es: Hola mundo 2021
```

```
Cantidad = 3  
Articulo = 365  
Precio = 49.95  
miorden = "Yo deseo {} piezas del artículo {} por {} soles."  
print(miorden.format(Cantidad, Articulo, Precio)) #el resultado es: Yo deseo 3 piezas del artículo 365 por 49.95 soles.
```

```
miorden = "Deseo pagar {2} soles, por {0} piezas del artículo {1}."  
print(miorden.format(Cantidad, Articulo, Precio)) #el resultado es: Deseo pagar 49.95 soles, por 3 piezas del artículo 365.
```

### Recorridos de cadenas (string)

```
x= "Por un mundo sostenible"  
For i in x:  
    print(i)
```

## 1.1 Introducción y Tipos de Datos

# Cadenas (string)

**Scape characters.**- algunos efectos para imprimir las cadenas.

Carácter \. \n nueva línea

```
miorden = "Deseo pagar {2} soles \n por {0} piezas \n del artículo {1}."
```

```
print(miorden.format(Cantidad, Articulo, Precio))
```

#el resultado es: Deseo pagar 49.95 soles

por 3 piezas

del artículo 365.

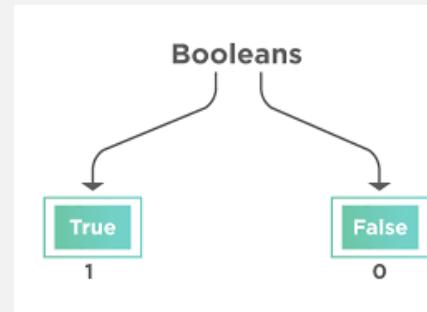
Code	Result
\'	Single Quote
\\"	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value

# 1.1 Introducción y Tipos de Datos

## Cadenas ([string](#))

Method	Description
<a href="#">capitalize()</a>	Converts the first character to upper case
<a href="#">casefold()</a>	Converts string into lower case
<a href="#">center()</a>	Returns a centered string
<a href="#">count()</a>	Returns the number of times a specified value occurs in a string
<a href="#">encode()</a>	Returns an encoded version of the string
<a href="#">endswith()</a>	Returns true if the string ends with the specified value
<a href="#">expandtabs()</a>	Sets the tab size of the string
<a href="#">find()</a>	Searches the string for a specified value and returns the position of where it was found
<a href="#">format()</a>	Formats specified values in a string
<a href="#">format_map()</a>	Formats specified values in a string
<a href="#">index()</a>	Searches the string for a specified value and returns the position of where it was found
<a href="#">isalnum()</a>	Returns True if all characters in the string are alphanumeric
<a href="#">isalpha()</a>	Returns True if all characters in the string are in the alphabet
<a href="#">isdecimal()</a>	Returns True if all characters in the string are decimals
<a href="#">isdigit()</a>	Returns True if all characters in the string are digits
<a href="#">isidentifier()</a>	Returns True if the string is an identifier
<a href="#">islower()</a>	Returns True if all characters in the string are lower case
<a href="#">isnumeric()</a>	Returns True if all characters in the string are numeric
<a href="#">isprintable()</a>	Returns True if all characters in the string are printable
<a href="#">isspace()</a>	Returns True if all characters in the string are whitespaces
<a href="#">istitle()</a>	Returns True if the string follows the rules of a title
<a href="#">isupper()</a>	Returns True if all characters in the string are upper case
<a href="#">join()</a>	Joins the elements of an iterable to the end of the string
<a href="#">ljust()</a>	Returns a left justified version of the string
<a href="#">lower()</a>	Converts a string into lower case
<a href="#">lstrip()</a>	Returns a left trim version of the string
<a href="#">maketrans()</a>	Returns a translation table to be used in translations
<a href="#">partition()</a>	Returns a tuple where the string is parted into three parts
<a href="#">replace()</a>	Returns a string where a specified value is replaced with a specified value
<a href="#">rfind()</a>	Searches the string for a specified value and returns the last position of where it was found
<a href="#">rindex()</a>	Searches the string for a specified value and returns the last position of where it was found
<a href="#">rjust()</a>	Returns a right justified version of the string
<a href="#">rpartition()</a>	Returns a tuple where the string is parted into three parts
<a href="#">rsplit()</a>	Splits the string at the specified separator, and returns a list
<a href="#">rstrip()</a>	Returns a right trim version of the string
<a href="#">split()</a>	Splits the string at the specified separator, and returns a list
<a href="#">splitlines()</a>	Splits the string at line breaks and returns a list
<a href="#">startswith()</a>	Returns true if the string starts with the specified value
<a href="#">strip()</a>	Returns a trimmed version of the string
<a href="#">swapcase()</a>	Swaps cases, lower case becomes upper case and vice versa
<a href="#">title()</a>	Converts the first character of each word to upper case
<a href="#">translate()</a>	Returns a translated string
<a href="#">upper()</a>	Converts a string into upper case
<a href="#">zfill()</a>	Fills the string with a specified number of 0 values at the beginning

## 1.1 Introducción y Tipos de Datos



# Booleans (lógicos)

Lógicos – verdadero o falso. Se puede tener la necesidad de saber si una expresión es verdadera o falsa

```
print (10 > 9 )  
print (10 == 9 )  
print (10 < 9 )
```

## Evaluación de variables y valores

x="hola"

y = 5

```
print ( bool(x) )          # la respuesta es true  
print ( bool(y) )          # la respuesta es true
```

```
print ( bool("") )          # la respuesta es false  
print ( bool(0) )           # la respuesta es false
```

# 1.1 Introducción y Tipos de Datos

## Listas (list)

Es una colección ordenada y modificable. Permite miembros duplicados.

En Python las listas se escriben entre corchetes rectos.

```
Estalista = ["apple", "banana", "cherry"] [pos inicial: pos final:pasos] la posición final no se incluye  
print(Estalista) # la respuesta es ['apple', 'banana', 'cherry']
```



0	1	2	3	4	5	6
apple	banana	cherry	orange	kiwi	melon	mango
-7	-6	-5	-4	-3	-2	-1

### ➤ Acceso a los elementos

Por ejemplo: imprimir el segundo elemento de la lista anterior

```
print(Estalista[1]) # la respuesta es banana
```

### ➤ Permite índice negativo.

```
print(Estalista[-1]) # la respuesta es cherry
```

### ➤ Permite rangos de índices.

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
print( Estalista[:4] ) # la respuesta es ['apple', 'banana', 'cherry', 'orange']
```

```
print( Estalista[-4:-1] ) # la respuesta es ['orange', 'kiwi', 'melon']
```

## 1.1 Introducción y Tipos de Datos

### Listas ([list](#))

- Cambios en elementos

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
Estalista[2] = 'lemon'
```

```
print( Estalista )          # la respuesta es ['apple', 'banana', 'lemon', 'orange', 'kiwi', 'melon', 'mango']
```

- Permite Loops o recorridos por los elementos

```
for x in Estalista:
```

```
    print (x)                  # la respuesta es apple  
                                banana  
                                lemon  
                                orange  
                                kiwi  
                                melon  
                                mango
```

- Verifica la existencia de un elemento. Por ejemplo si melon está en la lista

```
if "melon" in Estalista:
```

```
    print("Si, 'melon' está en la lista de frutas") # la respuesta es Si, 'melon' está en la lista de frutas
```



# 1.1 Introducción y Tipos de Datos



## Listas ([list](#))

- Determinar la longitud de la lista. Función len()

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
print( len(Estalista) )      # la respuesta es 7
```

- Agregar elementos. Estos se agregan al final de la lista con método append()

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
Estalista.append ("blueberry")
```

```
print(Estalista)      # la respuesta es ['apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango', 'blueberry']
```

- Insertar elementos. Estos se insertan en una posición específica de la lista con método insert()

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
Estalista.insert(2, "blueberry")
```

```
print(Estalista)      # la respuesta es ['apple', 'banana', 'blueberry', 'cherry', 'orange', 'kiwi', 'melon', 'mango']
```

- Eliminar elementos.

Se eliminan al final por defecto o en una posición específica de la lista con método pop()

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
Estalista.pop()
```

```
print(Estalista)
```

```
Estalista.pop(1)
```

```
print(Estalista)      # la respuesta es ['apple', 'cherry', 'orange', 'kiwi', 'melon']
```

# 1.1 Introducción y Tipos de Datos



## Listas ([list](#))

- Borrar la lista con comando del. Este comando borra la lista de memoria

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

del Estalista

```
print(Estalista)      # la respuesta es NameError: name 'Estalista' is not defined
```

- Vaciar la lista con método **clear()**

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
Estalista.clear()
```

```
print(Estalista)      # la respuesta es []
```

- Copiar lista

No puede copiar una lista simplemente escribiendo `list2 = list1`, porque: `list2` solo será una referencia a `list1`, y los cambios realizados en `list1` también se realizarán automáticamente en `list2`. Hay formas de hacer una copia, una es usar el método de lista incorporado **copy()**

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
Nuevalista = Estalista.copy()
```

```
print(Nuevalista)      # la respuesta es ['apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango']
```

# 1.1 Introducción y Tipos de Datos



## Listas ([list](#))

### ➤ Copiar lista

Otra forma de copiar es con el método interno `list()`

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
Nuevalista = list(Estalista)
```

```
print(Nuevalista)      # la respuesta es ['apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango']
```

### ➤ Unir dos listas

Existen varias formas de unir listas, una de ellas es la de concatenar (+)

```
list1 = ["a", "b", "c"]
```

```
list2 = [1, 2, 3]
```

```
list3 = list1 + list2
```

```
print(list3)      # la respuesta es ['a', 'b', 'c', 1, 2, 3]
```

Otra es usando el método `append()`

```
list1 = ["a", "b", "c"]
```

```
list2 = [1, 2, 3]
```

```
list1.append(list2)
```

```
print(list1)      # la respuesta es ['a', 'b', 'c', [1, 2, 3]]
```

# 1.1 Introducción y Tipos de Datos



## Listas ([list](#))

### ➤ Métodos incorporados

Python tiene un conjunto de métodos integrados que puede usar en las listas

Method	Description
<a href="#">append()</a>	Adds an element at the end of the list
<a href="#">clear()</a>	Removes all the elements from the list
<a href="#">copy()</a>	Returns a copy of the list
<a href="#">count()</a>	Returns the number of elements with the specified value
<a href="#">extend()</a>	Add the elements of a list (or any iterable), to the end of the current list
<a href="#">index()</a>	Returns the index of the first element with the specified value
<a href="#">insert()</a>	Adds an element at the specified position
<a href="#">pop()</a>	Removes the element at the specified position
<a href="#">remove()</a>	Removes the item with the specified value
<a href="#">reverse()</a>	Reverses the order of the list
<a href="#">sort()</a>	Sorts the list

## 1.3 Entradas y salidas de datos



### TEMA 3

---

Entradas y Salidas

## 1.3 Entradas y salidas de datos

### Entradas



Se usa la función `input()`. Esta función lee cadenas por lo que habrá que modificar luego el valor ingresado al tipo de dato correspondiente

➤ Sintaxis de `input()`

```
Entero = int(input("Ingrese un numero entero : "))
Decimal = float(input("Ingrese un numero decimal : "))
```

➤ Variables como argumentos en `input()`

```
numero1 = int(input("Ingrese un numero entero : "))
numero2 = int(input("Ingrese un numero entero mayor a {numero1} : "))
```



## Scripts o programa

Conjunto de sentencias de Python que se invoca desde la línea de comandos. También puede Compilarse con Spyder.

### ➤ Creación de un script

El programa se puede crear desde un editor de textos de Windows con la extensión **.py** o desde SublimeText (<https://www.sublimetext.com/> y bajar la versión correspondiente).

Desde la línea de comando y la carpeta donde se encuentra el archivo.py

### ➤ Variables como argumentos en input()

Se pasan parametros o argumentos, y para ello debe importarse en el programa la libreria SYS mediante la instrucción  
**import sys**

```
print(sys.argv)                      # se listan los argumentos remitidos  
texto = sys.argv[1]                  # se asigna el valor del primer argumento a la variable texto
```

En el siguiente ejemplo, se pasan parametros, una cadena y las veces que se deben imprimir

```
import sys  
print(sys.argv)  
if len(sys.argv) == 3:  
    texto = sys.argv[1]; repeticiones = int(sys.argv[2])
```

```
for x in range(repeticiones):  
    print(x)
```

En linea de comandos se invoca: python primer\_script.py “esta es una prueba” 5

# 1.3 Entradas y salidas de datos



## Salidas

Salida de información por diversos medios, usando el interprete sería por pantalla. Se usa la sentencia print()

### ➤ Sintaxis de print()

```
v = "otro texto"
```

```
n = 10
```

```
print("un texto", v, " y un número ", n)
```

# Se separan en comas los argumentos a imprimir

### ➤ Método format() para mayor información <https://www.python.org/dev/peps/pep-3101/>

```
c = "Un texto {} y un número {}".format(v, n)
```

```
print(c)
```

# El resultado es ‘Un texto otro texto y un número 10’

```
print("Un texto {0} y un número {1}".format(numero=n, texto=v))
```

# El 0 y 1 son los índices de los argumentos

```
print("Un texto {t} y un número {n}".format(n=n, t=v))
```

```
print("{:>30}".format("palabra"))
```

# Alineamiento a la derecha en 30 caracteres

```
print("{:30}".format("palabra"))
```

# Alineamiento a la izquierda en 30 caracteres

```
print("{:.3}".format("palabra"))
```

# truncamiento de 3 caracteres de la “palabra”

```
print("{:>30.3}".format("palabra"))
```

# truncamiento de 3 caracteres de la “palabra” alineado a la derecha

```
print("{:4d}".format(10))
```

# Alineamiento de numero a la derecha con 4 dígitos con 0 izq

```
print("{:7.3f}".format(3.14151926))
```

# Alineamiento con 3 decimales

```
print("{:7.3f}".format(153.21))
```

# Alineamiento con 3 decimales

# INSERTAR NOMBRE DE ACTIVIDAD



## PASOS DE ACTIVIDAD individual /grupal:

- Ingresar al colab personal con su cuenta gmail
  
- Copia el enunciado del problema señalado por el profesor y péguelo en el colab
  
- Desarrolle el ejercicio junto con un compañero sea en el laboratorio o en la sala grupal creada por el docente.

# Resolución de ejercicios

## ACTIVIDAD

Debe resolver los ejercicios señalados por el profesor que se encuentran en el documento:

**01 - 02.2 Fundamentos de Programación en Python  
- Guía laboratorio y Casos propuestos.pdf**

Este documento se encuentra en el aula virtual en la unidad 1



## CONCLUSIONES

**01** Python estándar ha sido desarrolladas con la arquitectura orientada a objetos.

**02**

**03**

**04**

# BIBLIOGRAFÍA

- Direcciones electrónicas (hipervínculos en las diapositivas)
- LUTZ, MARK (2013) Learning Python. 5th Edition. California: O'Reilly.
- RAMALHO, LUCIANO (2015) Fluent Python: Clear, Concise, and Effective Programming (inglés) 1st Edición. California: O'Reilly.
-

# **Continúa con las actividades propuestas**

---



Material producido por la  
Universidad Peruana de  
Ciencias Aplicadas

Autor:

Norman Reyes Morales

COPYRIGHT © UPC  
2020 – Todos los  
derechos reservados



---

# **SOFTWARE PARA INGENIEROS**

## **Unidad 1 – Fundamentos de programación en Python**

## LOGRO DE APRENDIZAJE



Al finalizar la unidad, el estudiante aplica los fundamentos de la programación Python en un contexto real o simulado, demostrando una actitud analítica y organizada.

## 1.4 Controlando el flujo



## TEMA 4

### Controlando el flujo

## 1.4 Controlando el flujo

Sentencias y Bucles en:



### If ...elif...else

Python admite las condiciones lógicas habituales de las matemáticas:

Equals:  $a == b$

Not Equals:  $a != b$

Less than:  $a < b$

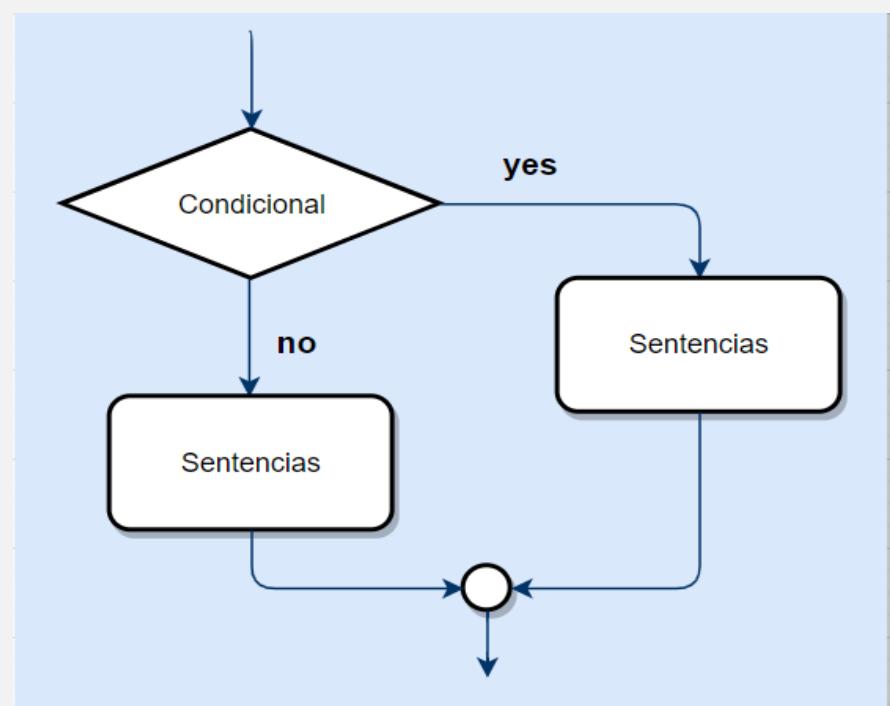
Less than or equal to:  $a <= b$

Greater than:  $a > b$

Greater than or equal to:  $a >= b$

Estas condiciones se pueden usar de varias maneras, más comúnmente en “sentencias if” y bucles. Se escribe una “sentencia if” utilizando la palabra clave **if**.

```
a = 33
b = 200
if b > a:
    print("b es mayor que a")
        # la respuesta es b es mayor que a
else:
    print("b es menor o igual que a")
        # la respuesta es b es menor o igual que a
```



## 1.4 Controlando el flujo

Sentencias y Bucles en:



### If ...elif...else

#### Indentación

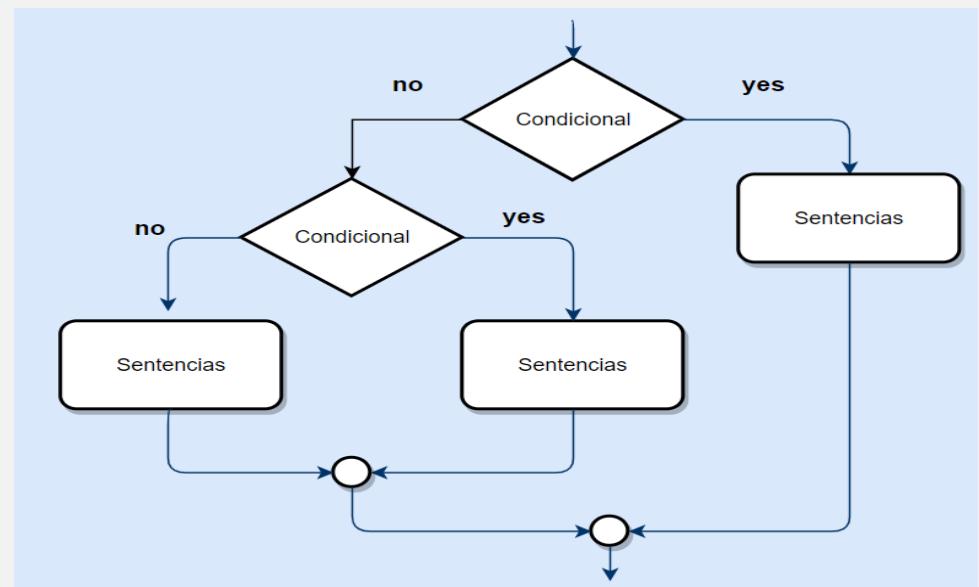
Python se basa en la sangría (espacio en blanco al comienzo de una línea) para definir el alcance en el código. Otros lenguajes de programación a menudo usan llaves para este propósito.

```
a = 33  
b = 200  
if b > a:  
  
    print("b es mayor que a") # La respuesta es error
```

#### Elif

La palabra clave **elif** es la forma de pitones de decir "si las condiciones anteriores no eran ciertas, entonces intente esta condición".

```
a = 33  
b = 200  
if b > a:  
    print("b es mayor que a")  
elif a == b:  
    print("b es igual que a")
```



## 1.4 Controlando el flujo

Sentencias y Bucles en:

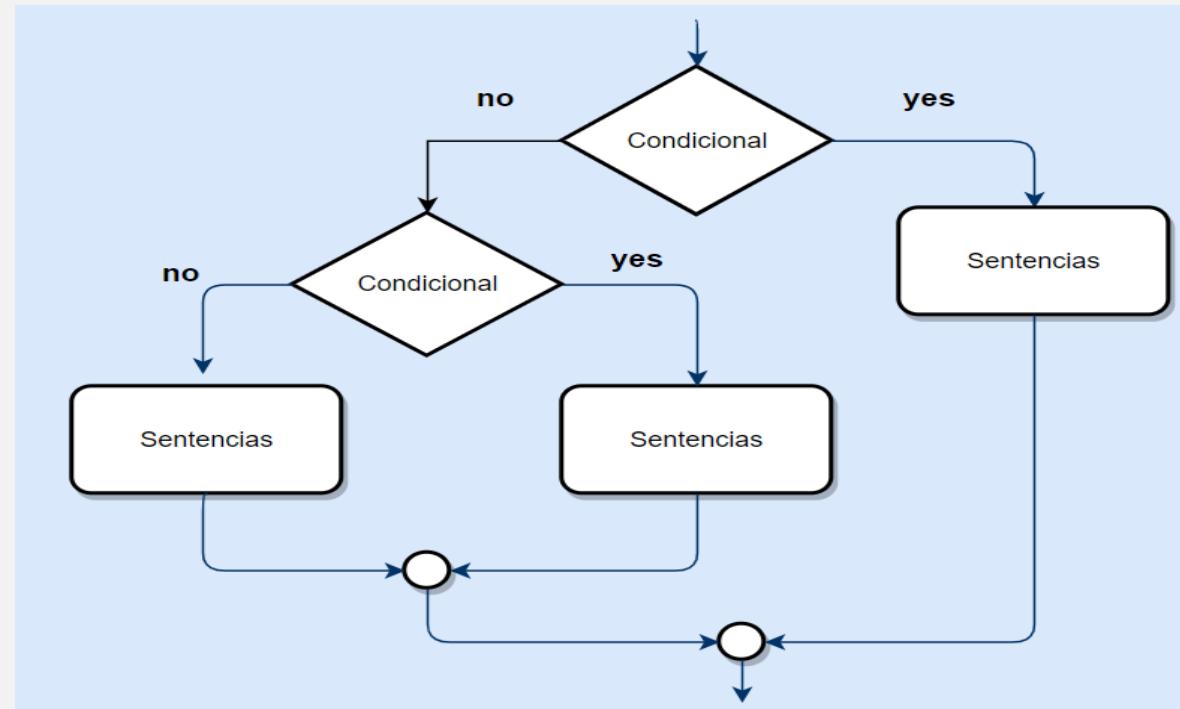


### If ...elif...else

#### Else

La palabra clave **else** captura cualquier cosa que no esté atrapada por las condiciones anteriores.

```
a = 33  
b = 200  
if b > a:  
    print("b es mayor que a")  
elif a == b:  
    print("b es igual que a")  
else:  
    print("a es mayor que b")
```



#### If sentencia corta (en una línea)

Si solo tiene que ejecutar una instrucción, puede ponerla en la misma línea que la instrucción if.

```
a=42  
b=200  
if b > a: print("b es mayor que a")
```

## 1.4 Controlando el flujo

Sentencias y Bucles en:



### If ...elif...else

#### If .. else sentencia corta (en una línea)

Si solo tiene que ejecutar una instrucción, una para if y otra para más, puede poner todo en la misma línea:

a=2

b=330

```
print("A") if a > b else print("B") # La respuesta es B
```

#### If .. elif ... else sentencia corta (en una línea)

Esta técnica se conoce como operadores ternarios o expresiones condicionales.

a=330

b=330

```
print("A") if a > b else print("=") if a == b else print("B") # La respuesta es =
```

### Condiciones lógicas: and y or

Esta técnica se conoce como operadores ternarios o expresiones condicionales.

a = 200

b = 33

c = 500

if a > b or a > c:

```
print("Al menos una de las condiciones es verdadera") # La respuesta es =
```

## 1.4 Controlando el flujo

Sentencias y Bucles en:



### If Anidados

Puede tener declaraciones if dentro de declaraciones if, esto se llama instrucciones if anidadas.

```
x = 41
if x > 10:
    print("Mas de diez,")
    if x > 20:
        print("y también es mayor a 20!")
    else:
        print("pero no mas de 20")      # La respuesta es Mas de diez, y también es mayor a 20!
```

### sentencia pass

Las sentencias **if** no pueden estar vacías, pero si por alguna razón tiene una sentencia **if** sin contenido, ingrese la sentencia **pass** para evitar un error.

```
a = 33
b = 200
if b > a:
    pass
```

# While ... loops

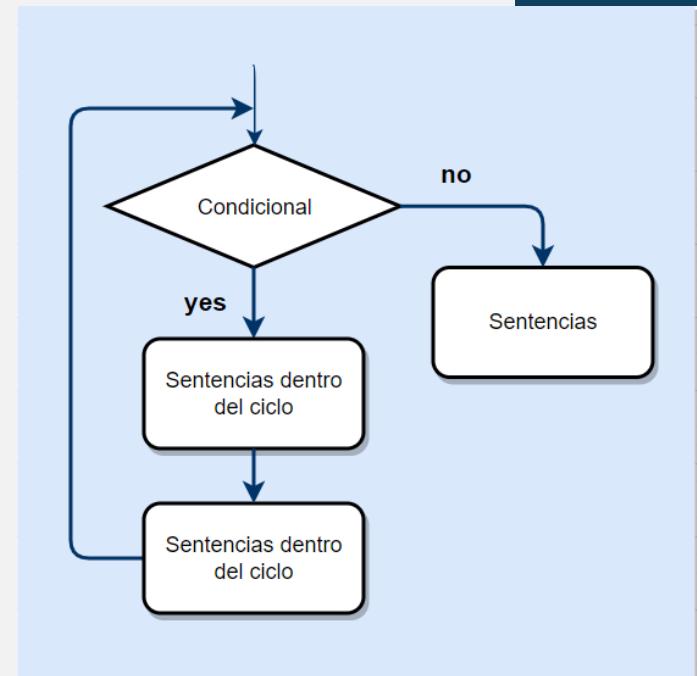
Con el ciclo **while** podemos ejecutar un conjunto de declaraciones siempre que una condición sea verdadera.

i = 1

while i < 6:

    print(i)

    i += 1



## Instrucción Break

Con la instrucción **break** podemos **detener** el ciclo incluso si la condición while es verdadera.

i = 1

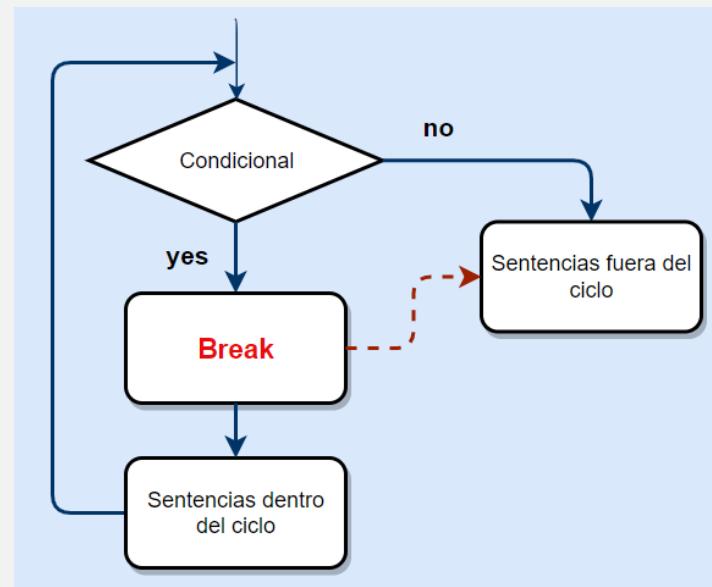
while i < 6:

    print(i)

    if i == 3:

        break

    i += 1

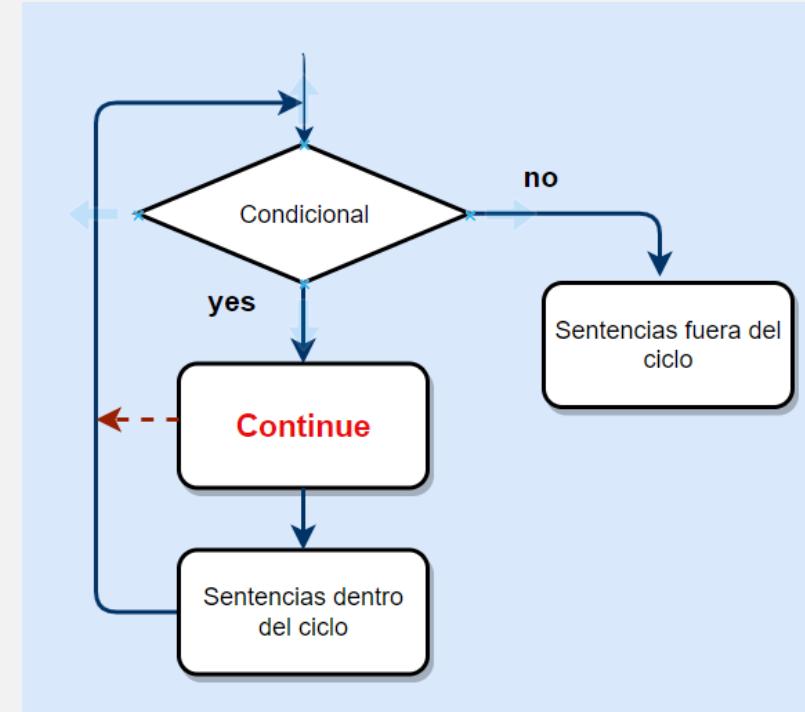


# While ... loops

## Instrucción Continue

Con la instrucción **continue** podemos detener la iteración actual y continuar con la siguiente:

```
i = 0  
while i < 6:  
    i += 1  
    if i == 3:  
        continue  
    print(i)
```



## Instrucción Else

Con la instrucción **else** podemos ejecutar un bloque de código una vez cuando la condición ya no es verdadera:

```
i = 1  
while i < 6:  
    print(i)  
    i += 1  
else:  
    print("ya no es menos de 6")
```



# For ... loops

Un bucle **for** se usa para iterar sobre una secuencia (que es una lista, una tupla, un diccionario, un conjunto o una cadena). Esto es menos como la palabra clave **for** en otros lenguajes de programación, y funciona más como un método iterador como se encuentra en otros lenguajes de programación orientada a objetos. Con el bucle **for** podemos ejecutar un conjunto de declaraciones, una vez para cada elemento de una lista, tupla, conjunto, etc.

```
frutas = ["apple", "banana", "cherry"]
for x in frutas:
    print(x)
```

El bucle for no requiere una variable de indexación para establecer de antemano

## Bucle a traves de una cadena (string)

Incluso las cadenas son objetos iterables, contienen una secuencia de caracteres:

```
for x in "banana":
    print(x)
```

# For ... loops

## Sentencia Break

Con la instrucción **break** podemos detener el ciclo antes de que haya pasado por todos los elementos:

```
fruits = ["apple", "banana", "cherry"]
```

```
for x in fruits:
```

```
    print(x)
```

```
    if x == "banana":
```

```
        break
```

## Instrucción Continue

Con la instrucción **continue** podemos detener la iteración actual del ciclo y continuar con la siguiente:

```
fruits = ["apple", "banana", "cherry"]
```

```
for x in fruits:
```

```
    if x == "banana":
```

```
        continue
```

```
    print(x)
```

# For ... loops

## La función Range()

Para recorrer un conjunto de código un número específico de veces, podemos usar la función range (), La función range () devuelve una secuencia de números, comenzando desde 0 por defecto, y se incrementa en 1 (por defecto), y termina en un número específico.

```
for x in range(6):  
    print(x)
```

Tenga en cuenta que el rango (6) no son los valores de 0 a 6, sino los valores de 0 a 5.

La función range () se establece por defecto en 0 como valor inicial, sin embargo, es posible especificar el valor inicial agregando un parámetro: range (2, 6), que significa valores de 2 a 6 (pero sin incluir 6):

```
for x in range(2, 6):  
    print(x)
```

# For ... loops

## Palabra clave Else

La palabra clave **else** en un bucle **for** especifica un bloque de código que se ejecutará cuando finalice el bucle:

```
for x in range(6):
    print(x)
else:
    print("Finally finished!")
```

## Bucles anidados (Nested Loops)

Un bucle anidado es un bucle dentro de un bucle. El "bucle interno" se ejecutará una vez por cada iteración del "bucle externo":

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
    for y in fruits:
        print(x, y)          # la respuesta es red apple
                               # red banana .....
```

## 1.4 Controlando el flujo

Sentencias y Bucles en:



### For ... loops

#### Modificar Listas

Se puede modificar listas con las siguientes instrucciones. Por ejemplo se tiene una lista de números y se quiere esta lista multiplicada por 10

```
indice = 0
```

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
for elemento in numeros:
```

```
    numeros[indice] *= 10
```

```
    indice+=1
```

```
numeros
```

# la respuesta es [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

#### Modificar Listas

Se puede modificar listas con las siguientes instrucciones. Por ejemplo se tiene una lista de números y se quiere esta lista multiplicada por 10 y ahora se hará con la función **enumerate()**

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
for indice, elemento in enumerate(numeros):
```

```
    print(indice, elemento)
```

```
    numeros[indice] *= 10
```

```
numeros
```

# la respuesta es [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

## 1.4 Controlando el flujo

### For y while en 1 linea

Sentencias y Bucles en:



```
y =0  
suma = 0  
num = int(input("Ingrese un numero:"))  
  
print()  
for x in range(int(num)):  
    suma += (x + 1 )  
print("Con (FOR) La suma es:", suma)
```

For en una linea:

```
print("Con (FOR): ", sum(x for x in range(1, num+1)))  
                                # la respuesta es [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  
nueva_lista = [print(x) for x in [1,2,3,4,5,6] if x < 5]
```

```
while y < num:  
    y += 1  
    suma += y
```

```
print("Con (WHILE):", suma)
```

# INSERTAR NOMBRE DE ACTIVIDAD



## PASOS DE ACTIVIDAD individual /grupal:

- Ingresar al colab personal con su cuenta gmail
  
- Copia el enunciado del problema señalado por el profesor y péguelo en el colab
  
- Desarrolle el ejercicio junto con un compañero sea en el laboratorio o en la sala grupal creada por el docente.

# Resolución de ejercicios

## ACTIVIDAD

Debe resolver los ejercicios señalados por el profesor que se encuentran en el documento:

**01 - 02.2 Fundamentos de Programación en Python  
- Guía laboratorio y Casos propuestos.pdf**

Este documento se encuentra en el aula virtual en la unidad 1



# CONCLUSIONES

**01** Python estándar ha sido desarrolladas con la arquitectura orientada a objetos.

**02** La arquitectura Orientada a Objetos, simplifica las instrucciones de control

**03** Los tipos de datos como Listas y Cadenas, tienen mas funcionalidades en comparación con otros lenguajes de programación

**04**

# BIBLIOGRAFÍA

- Direcciones electrónicas (hipervínculos en las diapositivas)
- LUTZ, MARK (2013) Learning Python. 5th Edition. California: O'Reilly.
- RAMALHO, LUCIANO (2015) Fluent Python: Clear, Concise, and Effective Programming (inglés) 1st Edición. California: O'Reilly.
-

# **Continúa con las actividades propuestas**

---



Material producido por la  
Universidad Peruana de  
Ciencias Aplicadas

Autor:

Norman Reyes Morales

COPYRIGHT © UPC  
2020 – Todos los  
derechos reservados



---

# **SOFTWARE PARA INGENIEROS**

## **Unidad 1 – Fundamentos de programación en Python**

## LOGRO DE APRENDIZAJE



Al finalizar la unidad, el estudiante aplica los fundamentos de la programación Python en un contexto real o simulado, demostrando una actitud analítica y organizada.

# TEMARIO



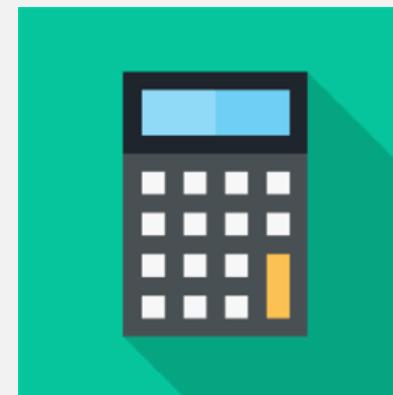
## TEMA 1

Introducción y Tipo de Datos



## TEMA 4

Controlando el flujo



## TEMA 2

Operadores y expresiones



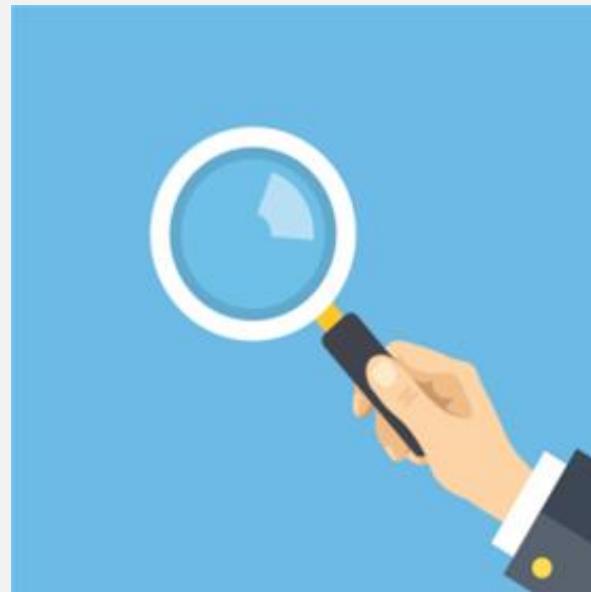
## TEMA 3

Entradas y Salidas



## TEMA 5

Matrices



## TEMA 1

---

Introducción y Tipo de Datos

# 1.1 Introducción y Tipos de Datos



## Nombre de variables

Una variable puede tener nombres cortos como X e Y, o tener nombres mas descriptivos como Edad, Nombre, Volumen\_Total.

Reglas para las variables de Python:

- Una variable debe iniciar con una letra o una raya abajo ( \_ underscore character)
- Una variable no puede iniciar con numero
- Una variable puede contener solo caracteres alfa numéricos o raya abajo (A-z, 0-9, and \_ )
- Los nombres de las variables son sensibles a las mayúsculas (p.e. edad, Edad, EDAD son tres variables diferentes)

## Declaración de variables

Python como muchos lenguajes, declara la variable con el primer valor asignado a este. **NO REQUIERE UN COMANDO**

- `x = 5`
- `y = "Hello, World!"`

## Asignación de valores a múltiples variables

Python permite asignar valores a múltiples variables en una sola línea.

- `x, y, z = "Orange", "Banana", "Cherry"`
- `x = y = z = "Orange"`

# 1.1 Introducción y Tipos de Datos



## Comentarios

Python con el propósito de documentar sus códigos de línea, permite comentarlas mediante el carácter # al inicio de la línea:

- `#Este es un comentario`  
`print("Hola, mundo!")`
- `print("Hola, mundo!") #Este es un comentario`

Se pueden documentar varios códigos de línea:

- `#Este es un comentario`  
`# escrito en mas`  
`# de una línea`  
`print("Hola, mundo!")`

## 1.1 Introducción y Tipos de Datos



### Data type

Text Type:

str

Numeric Types:

int, float, complex

Sequence Types:

list, tuple, range

Mapping Type:

dict

Set Types:

set, frozenset

Boolean Type:

bool

Binary Types:

bytes, bytearray, memoryview

# 1.1 Introducción y Tipos de Datos



## Data type

Asignación de variables

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

# 1.1 Introducción y Tipos de Datos



## Data type

Para asignar a una variable un tipo de dato específico, se deben utilizar las siguientes funciones:

Example	Data Type
x = str("Hello World")	str
x = int(20)	int
x = float(20.5)	float
x = complex(1j)	complex
x = list(("apple", "banana", "cherry"))	list
x = tuple(["apple", "banana", "cherry"])	tuple
x = range(6)	range
x = dict(name="John", age=36)	dict
x = set(("apple", "banana", "cherry"))	set
x = frozenset(("apple", "banana", "cherry"))	frozenset
x = bool(5)	bool
x = bytes(5)	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

# 1.1 Introducción y Tipos de Datos



## Numbers (números)

**Enteros** – Int, or integer, es un numero entero, positivo o negativo, sin decimales, de longitud indefinida

```
x = 1  
y = 35656222554887711  
z = -3255522  
print( type(x) )  
print( type(y) )  
print( type(z) )
```

**Flotantes** – float, o numero punto flotante, positivo o negativo, contiene uno o mas decimales

```
x = 1.10  
y = 1.0e10  
z = -325.5522  
print( type(x) )  
print( type(y) )  
print( type(z) )
```

**Complejos** – Que se escriben como un j como parte imaginaria

```
x = 3 + 5j  
y = 8j  
z = 10 -3j  
print( type(x) )  
print( type(y) )  
print( type(z) )
```

### Tipo de conversiones

x = 25	# int
y = 24.85	# float
z = -3j	# complex

a = float (x)	# convierte un entero en un flotante
b = int (y)	# convierte un flotante a un entero
c = complex (x)	# convierte un entero a un complejo

# 1.1 Introducción y Tipos de Datos

## Cadenas (string)

Las cadenas en Python, deben estar encerradas entre comillas “ ” o apóstrofos ‘ ’

“hola” es lo mismo a ‘hola’



Longitud de las cadenas en Python: función len()

```
x='Nada es imposible'
```

```
len(x) #el resultado es 17
```

Se puede usar parte de una cadena, usando **slicing [pos inicial: pos final: pasos]** la posición final no se incluye

```
x='Hola todos'
```

```
print(x[2:6]) #el resultado es: la t
```

0	1	2	3	4	5	6	7	8	9
H	O	L	A		T	O	D	O	S
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Los índices también pueden ser negativos **slicing [pos inicial: pos final:pasos]**

```
x='Hola todos'
```

```
print(x[-8: -2]) #el resultado es: la tod
```

```
print(x[-5:]) #el resultado es: todos
```

# 1.1 Introducción y Tipos de Datos

ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	\$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	-		

Extended ASCII characters

128	ç	160	á	192	ł	224	ó
129	ü	161	í	193	ł	225	ß
130	é	162	ó	194	ł	226	ö
131	â	163	ú	195	ł	227	ò
132	ä	164	ñ	196	ł	228	ö
133	à	165	ñ	197	ł	229	õ
134	à	166	º	198	ä	230	µ
135	ç	167	º	199	À	231	þ
136	ê	168	¿	200	Ł	232	Þ
137	ë	169	®	201	Ł	233	Ú
138	è	170	™	202	Ł	234	Ó
139	ï	171	½	203	Ł	235	Ù
140	î	172	¼	204	Ł	236	Ý
141	í	173	í	205	Ł	237	Ý
142	Ä	174	«	206	Ł	238	—
143	Ä	175	»	207	Ł	239	—
144	É	176	¤	208	ø	240	≡
145	æ	177	¤	209	đ	241	±
146	Æ	178	¤	210	€	242	≡
147	ô	179	—	211	€	243	¾
148	ö	180	—	212	€	244	¶
149	ò	181	À	213	—	245	§
150	ú	182	À	214	—	246	÷
151	ù	183	À	215	—	247	·
152	ÿ	184	©	216	—	248	°
153	Ö	185	—	217	—	249	—
154	Ü	186	—	218	—	250	·
155	ø	187	—	219	—	251	·
156	£	188	—	220	—	252	·
157	Ø	189	¢	221	—	253	·
158	×	190	¥	222	—	254	■
159	f	191	—	223	—	255	nbsp

## 1.1 Introducción y Tipos de Datos

# Cadenas (string)



Las cadenas se pueden concatenar es decir unir con otras cadenas para formar otras cadenas. No se puede concatenar con números enteros o flotantes

```
x="hola"  
y="mundo"  
print(x+y) #el resultado es: hola mundo
```

```
print(x+y+3) #el resultado es: error  
print((x+y)*3) #el resultado es: hola mundohola mundohola mundo
```

Python puede comparar cadenas: `==` y `!=` (igual o diferente). La comparación se hace carácter por carácter. Si estas son iguales o de lo contrario serán diferentes

```
x="hola"  
y="hola"  
x==y #el resultado es: True (verdadero)
```

```
x="hola"  
y="Hola"  
x==y #el resultado es: False (falso)
```

Python también compara cadenas: `<` o `>` (menor o mayor). La comparación se hace carácter por carácter. Pero considerando su código ASCII (128 caracteres u 8 bits) pero ahora su nuevo estándar **Unicode (65536 u 16 bits)**

```
x="hola"  
y="Hola"  
x<y #el resultado es: False (falso)  
ord('h') #el resultado es: 104  
ord('H') #el resultado es: 72
```

La función `ORD()` devuelve la posición del carácter en la tabla **UNICODE**  
La función `chr()` devuelve el carácter de la posición en la tabla **UNICODE**

# 1.1 Introducción y Tipos de Datos



## Cadenas (string)

**String Format.**- También se requieren Combinar números con letras hacer tabulados

```
x=2021  
y="Hola mundo "  
print(x+y) #el resultado es: error
```

```
x=2021  
y="Hola mundo {}"  
print(y.format(x)) #el resultado es: Hola mundo 2021
```

```
Cantidad = 3  
Articulo = 365  
Precio = 49.95  
miorden = "Yo deseo {} piezas del artículo {} por {} soles."  
print(miorden.format(Cantidad, Articulo, Precio)) #el resultado es: Yo deseo 3 piezas del artículo 365 por 49.95 soles.
```

```
miorden = "Deseo pagar {2} soles, por {0} piezas del artículo {1}."  
print(miorden.format(Cantidad, Articulo, Precio)) #el resultado es: Deseo pagar 49.95 soles, por 3 piezas del artículo 365.
```

## Recorridos de cadenas (string)

```
x= "Por un mundo sostenible"  
For i in x:  
    print(i)
```

## 1.1 Introducción y Tipos de Datos

# Cadenas (string)

**Scape characters.**- algunos efectos para imprimir las cadenas.

Carácter \. \n nueva línea

```
miorden = "Deseo pagar {2} soles \n por {0} piezas \n del artículo {1}."
```

```
print(miorden.format(Cantidad, Articulo, Precio))
```

#el resultado es: Deseo pagar 49.95 soles

por 3 piezas

del artículo 365.

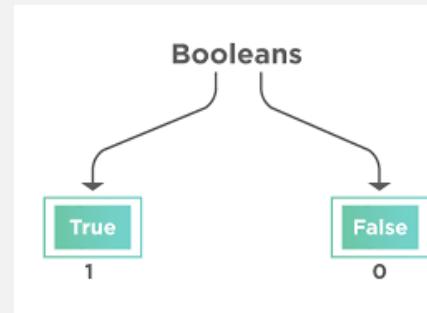
Code	Result
\'	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value

# 1.1 Introducción y Tipos de Datos

## Cadenas ([string](#))

Method	Description
<a href="#">capitalize()</a>	Converts the first character to upper case
<a href="#">casefold()</a>	Converts string into lower case
<a href="#">center()</a>	Returns a centered string
<a href="#">count()</a>	Returns the number of times a specified value occurs in a string
<a href="#">encode()</a>	Returns an encoded version of the string
<a href="#">endswith()</a>	Returns true if the string ends with the specified value
<a href="#">expandtabs()</a>	Sets the tab size of the string
<a href="#">find()</a>	Searches the string for a specified value and returns the position of where it was found
<a href="#">format()</a>	Formats specified values in a string
<a href="#">format_map()</a>	Formats specified values in a string
<a href="#">index()</a>	Searches the string for a specified value and returns the position of where it was found
<a href="#">isalnum()</a>	Returns True if all characters in the string are alphanumeric
<a href="#">isalpha()</a>	Returns True if all characters in the string are in the alphabet
<a href="#">isdecimal()</a>	Returns True if all characters in the string are decimals
<a href="#">isdigit()</a>	Returns True if all characters in the string are digits
<a href="#">isidentifier()</a>	Returns True if the string is an identifier
<a href="#">islower()</a>	Returns True if all characters in the string are lower case
<a href="#">isnumeric()</a>	Returns True if all characters in the string are numeric
<a href="#">isprintable()</a>	Returns True if all characters in the string are printable
<a href="#">isspace()</a>	Returns True if all characters in the string are whitespaces
<a href="#">istitle()</a>	Returns True if the string follows the rules of a title
<a href="#">isupper()</a>	Returns True if all characters in the string are upper case
<a href="#">join()</a>	Joins the elements of an iterable to the end of the string
<a href="#">ljust()</a>	Returns a left justified version of the string
<a href="#">lower()</a>	Converts a string into lower case
<a href="#">lstrip()</a>	Returns a left trim version of the string
<a href="#">maketrans()</a>	Returns a translation table to be used in translations
<a href="#">partition()</a>	Returns a tuple where the string is parted into three parts
<a href="#">replace()</a>	Returns a string where a specified value is replaced with a specified value
<a href="#">rfind()</a>	Searches the string for a specified value and returns the last position of where it was found
<a href="#">rindex()</a>	Searches the string for a specified value and returns the last position of where it was found
<a href="#">rjust()</a>	Returns a right justified version of the string
<a href="#">rpartition()</a>	Returns a tuple where the string is parted into three parts
<a href="#">rsplit()</a>	Splits the string at the specified separator, and returns a list
<a href="#">rstrip()</a>	Returns a right trim version of the string
<a href="#">split()</a>	Splits the string at the specified separator, and returns a list
<a href="#">splitlines()</a>	Splits the string at line breaks and returns a list
<a href="#">startswith()</a>	Returns true if the string starts with the specified value
<a href="#">strip()</a>	Returns a trimmed version of the string
<a href="#">swapcase()</a>	Swaps cases, lower case becomes upper case and vice versa
<a href="#">title()</a>	Converts the first character of each word to upper case
<a href="#">translate()</a>	Returns a translated string
<a href="#">upper()</a>	Converts a string into upper case
<a href="#">zfill()</a>	Fills the string with a specified number of 0 values at the beginning

## 1.1 Introducción y Tipos de Datos



# Booleans (lógicos)

Lógicos – verdadero o falso. Se puede tener la necesidad de saber si una expresión es verdadera o falsa

```
print (10 > 9 )  
print (10 == 9 )  
print (10 < 9 )
```

## Evaluación de variables y valores

x="hola"

y = 5

```
print ( bool(x) )          # la respuesta es true  
print ( bool(y) )          # la respuesta es true
```

```
print ( bool("") )          # la respuesta es false  
print ( bool(0) )           # la respuesta es false
```

# 1.1 Introducción y Tipos de Datos

## Listas (list)

Es una colección ordenada y modificable. Permite miembros duplicados.

En Python las listas se escriben entre corchetes rectos.

```
Estalista = ["apple", "banana", "cherry"] [pos inicial: pos final:pasos] la posición final no se incluye  
print(Estalista) # la respuesta es ['apple', 'banana', 'cherry']
```



0	1	2	3	4	5	6
apple	banana	cherry	orange	kiwi	melon	mango
-7	-6	-5	-4	-3	-2	-1

### ➤ Acceso a los elementos

Por ejemplo: imprimir el segundo elemento de la lista anterior

```
print(Estalista[1]) # la respuesta es banana
```

### ➤ Permite índice negativo.

```
print(Estalista[-1]) # la respuesta es cherry
```

### ➤ Permite rangos de índices.

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
print( Estalista[:4] ) # la respuesta es ['apple', 'banana', 'cherry', 'orange']
```

```
print( Estalista[-4:-1] ) # la respuesta es ['orange', 'kiwi', 'melon']
```

## 1.1 Introducción y Tipos de Datos

### Listas ([list](#))

- Cambios en elementos

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
Estalista[2] = 'lemon'
```

```
print( Estalista )          # la respuesta es ['apple', 'banana', 'lemon', 'orange', 'kiwi', 'melon', 'mango']
```

- Permite Loops o recorridos por los elementos

```
for x in Estalista:
```

```
    print (x)                  # la respuesta es apple  
                                banana  
                                lemon  
                                orange  
                                kiwi  
                                melon  
                                mango
```

- Verifica la existencia de un elemento. Por ejemplo si melon está en la lista

```
if "melon" in Estalista:
```

```
    print("Si, 'melon' está en la lista de frutas") # la respuesta es Si, 'melon' está en la lista de frutas
```



# 1.1 Introducción y Tipos de Datos



## Listas ([list](#))

- Determinar la longitud de la lista. Función len()

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
print( len(Estalista) )      # la respuesta es 7
```

- Agregar elementos. Estos se agregan al final de la lista con método append()

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
Estalista.append ("blueberry")
```

```
print(Estalista)      # la respuesta es ['apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango', 'blueberry']
```

- Insertar elementos. Estos se insertan en una posición específica de la lista con método insert()

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
Estalista.insert(2, "blueberry")
```

```
print(Estalista)      # la respuesta es ['apple', 'banana', 'blueberry', 'cherry', 'orange', 'kiwi', 'melon', 'mango']
```

- Eliminar elementos.

Se eliminan al final por defecto o en una posición específica de la lista con método pop()

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
Estalista.pop()
```

```
print(Estalista)
```

```
Estalista.pop(1)
```

```
print(Estalista)      # la respuesta es ['apple', 'cherry', 'orange', 'kiwi', 'melon']
```

# 1.1 Introducción y Tipos de Datos



## Listas ([list](#))

- Borrar la lista con comando del. Este comando borra la lista de memoria

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

del Estalista

```
print(Estalista)      # la respuesta es NameError: name 'Estalista' is not defined
```

- Vaciar la lista con método **clear()**

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
Estalista.clear()
```

```
print(Estalista)      # la respuesta es []
```

- Copiar lista

No puede copiar una lista simplemente escribiendo `list2 = list1`, porque: `list2` solo será una referencia a `list1`, y los cambios realizados en `list1` también se realizarán automáticamente en `list2`. Hay formas de hacer una copia, una es usar el método de lista incorporado **copy()**

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
Nuevalista = Estalista.copy()
```

```
print(Nuevalista)      # la respuesta es ['apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango']
```

# 1.1 Introducción y Tipos de Datos



## Listas ([list](#))

### ➤ Copiar lista

Otra forma de copiar es con el método interno `list()`

```
Estalista = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
Nuevalista = list(Estalista)
```

```
print(Nuevalista)      # la respuesta es ['apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango']
```

### ➤ Unir dos listas

Existen varias formas de unir listas, una de ellas es la de concatenar (+)

```
list1 = ["a", "b", "c"]
```

```
list2 = [1, 2, 3]
```

```
list3 = list1 + list2
```

```
print(list3)      # la respuesta es ['a', 'b', 'c', 1, 2, 3]
```

Otra es usando el método `append()`

```
list1 = ["a", "b", "c"]
```

```
list2 = [1, 2, 3]
```

```
list1.append(list2)
```

```
print(list1)      # la respuesta es ['a', 'b', 'c', [1, 2, 3]]
```

# 1.1 Introducción y Tipos de Datos



## Listas ([list](#))

### ➤ Métodos incorporados

Python tiene un conjunto de métodos integrados que puede usar en las listas

Method	Description
<a href="#">append()</a>	Adds an element at the end of the list
<a href="#">clear()</a>	Removes all the elements from the list
<a href="#">copy()</a>	Returns a copy of the list
<a href="#">count()</a>	Returns the number of elements with the specified value
<a href="#">extend()</a>	Add the elements of a list (or any iterable), to the end of the current list
<a href="#">index()</a>	Returns the index of the first element with the specified value
<a href="#">insert()</a>	Adds an element at the specified position
<a href="#">pop()</a>	Removes the element at the specified position
<a href="#">remove()</a>	Removes the item with the specified value
<a href="#">reverse()</a>	Reverses the order of the list
<a href="#">sort()</a>	Sorts the list

## 1.2 Operadores y expresiones



### TEMA 2

---

Operadores y expresiones

## 1.2 Operadores y expresiones



- Python tiene los siguientes operadores
  - Aritméticos
  - De asignación
  - De comparación
  - Lógicos
  - Identidad
  - Miembros
  - Bitwise (Binarios)

## 1.2 Operadores y expresiones



Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	$x / y$
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor división (división entera)	$x // y$

# Operadores de asignación

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x  = 3	x = x   3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3



# Operadores de comparación



Operator	Name	Example
<code>==</code>	Equal	<code>x == y</code>
<code>!=</code>	Not equal	<code>x != y</code>
<code>&gt;</code>	Greater than	<code>x &gt; y</code>
<code>&lt;</code>	Less than	<code>x &lt; y</code>
<code>&gt;=</code>	Greater than or equal to	<code>x &gt;= y</code>
<code>&lt;=</code>	Less than or equal to	<code>x &lt;= y</code>

## 1.2 Operadores y expresiones



### Operadores lógicos

Operator	Description	Example
and	Returns True if both statements are true	<code>x &lt; 5 and x &lt; 10</code>
or	Returns True if one of the statements is true	<code>x &lt; 5 or x &lt; 4</code>
not	Reverse the result, returns False if the result is true	<code>not(x &lt; 5 and x &lt; 10)</code>

### Operadores de identidad

Los operadores de identidad se usan para comparar los objetos, no si son iguales, sino si en realidad son el mismo objeto, con la misma ubicación de memoria:

Operator	Description	Example
is	Returns True if both variables are the same object	<code>x is y</code>
is not	Returns True if both variables are not the same object	<code>x is not y</code>



# Operadores de miembros

Los operadores de membresía se utilizan para probar si una secuencia se presenta en un objeto:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

# Operadores Bitwise

Los operadores bit a bit se usan para comparar números (binarios):

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

## 1.3 Entradas y salidas de datos



### TEMA 3

---

Entradas y Salidas

## 1.3 Entradas y salidas de datos

### Entradas



Se usa la función `input()`. Esta función lee cadenas por lo que habrá que modificar luego el valor ingresado al tipo de dato correspondiente

➤ Sintaxis de `input()`

```
Entero = int(input("Ingrese un numero entero : "))  
Decimal = float(input("Ingrese un numero decimal : "))
```

➤ Variables como argumentos en `input()`

```
numero1 = int(input("Ingrese un numero entero : "))  
numero2 = int(input("Ingrese un numero entero mayor a {numero1} : "))
```



## Scripts o programa

Conjunto de sentencias de Python que se invoca desde la línea de comandos. También puede Compilarse con Spyder.

### ➤ Creación de un script

El programa se puede crear desde un editor de textos de Windows con la extensión **.py** o desde SublimeText (<https://www.sublimetext.com/> y bajar la versión correspondiente).

Desde la línea de comando y la carpeta donde se encuentra el archivo.py

### ➤ Variables como argumentos en input()

Se pasan parametros o argumentos, y para ello debe importarse en el programa la libreria SYS mediante la instrucción  
**import sys**

```
print(sys.argv)                      # se listan los argumentos remitidos  
texto = sys.argv[1]                  # se asigna el valor del primer argumento a la variable texto
```

En el siguiente ejemplo, se pasan parametros, una cadena y las veces que se deben imprimir

```
import sys  
print(sys.argv)  
if len(sys.argv) == 3:  
    texto = sys.argv[1]; repeticiones = int(sys.argv[2])
```

```
for x in range(repeticiones):  
    print(x)
```

En linea de comandos se invoca: python primer\_script.py “esta es una prueba” 5

# 1.3 Entradas y salidas de datos

## Salidas



Salida de información por diversos medios, usando el interprete sería por pantalla. Se usa la sentencia print()

### ➤ Sintaxis de print()

```
v = "otro texto"
```

```
n = 10
```

```
print("un texto", v, " y un número ", n) # Se separan en comas los argumentos a imprimir
```

### ➤ Método **format()** para mayor información <https://www.python.org/dev/peps/pep-3101/>

```
c = "Un texto {} y un número {}".format(v, n)
```

```
print( c)
```

```
print("Un texto {0} y un número {1}".format(numero=n, texto=v)) # El 0 y 1 son los índices de los argumentos
```

```
print("Un texto {t} y un número {n}".format(n=n, t=v))
```

```
print( "{:>30}".format("palabra"))
```

```
# Alineamiento a la derecha en 30 caracteres
```

```
print( "{:30}".format("palabra"))
```

```
# Alineamiento a la izquierda en 30 caracteres
```

```
print( "{:.3}".format("palabra"))
```

```
# truncamiento de 3 caracteres de la "palabra"
```

```
print( "{:>30.3}".format("palabra"))
```

```
# truncamiento de 3 caracteres de la "palabra" alineado a la derecha
```

```
print( "{:4d}".format(10))
```

```
# Alineamiento de numero a la derecha con 4 dígitos con 0 izq
```

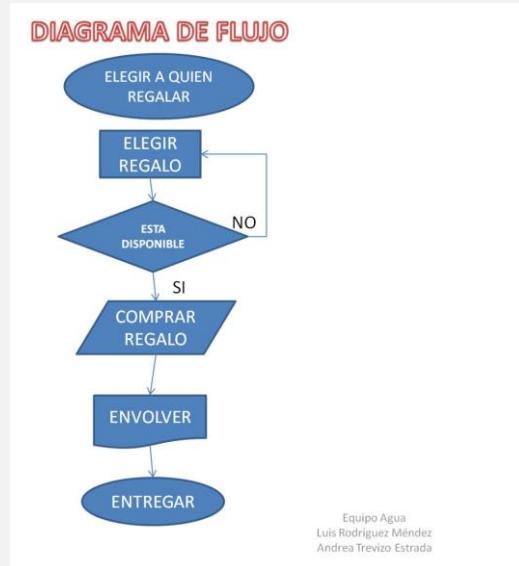
```
print( "{:7.3f}".format(3.14151926))
```

```
# Alineamiento con 3 decimales
```

```
print( "{:7.3f}".format(153.21))
```

```
# Alineamiento con 3 decimales
```

## 1.4 Controlando el flujo



## TEMA 4

### Controlando el flujo

## 1.4 Controlando el flujo

Sentencias y Bucles en:



### If ...elif...else

Python admite las condiciones lógicas habituales de las matemáticas:

Equals:  $a == b$

Not Equals:  $a != b$

Less than:  $a < b$

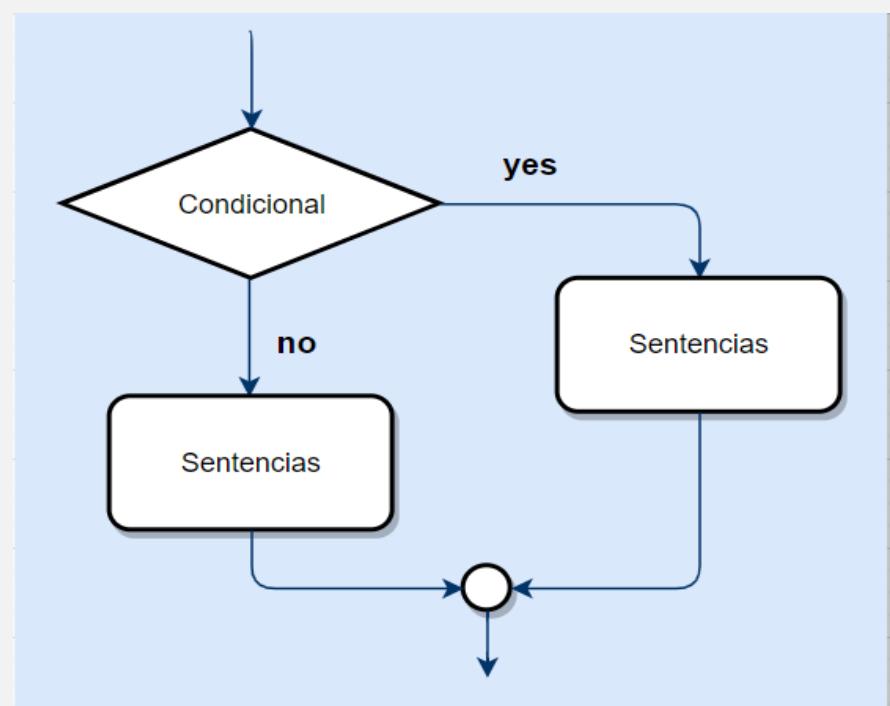
Less than or equal to:  $a <= b$

Greater than:  $a > b$

Greater than or equal to:  $a >= b$

Estas condiciones se pueden usar de varias maneras, más comúnmente en “sentencias if” y bucles. Se escribe una “sentencia if” utilizando la palabra clave **if**.

```
a = 33
b = 200
if b > a:
    print("b es mayor que a")
        # la respuesta es b es mayor que a
else:
    print("b es menor o igual que a")
        # la respuesta es b es menor o igual que a
```



## 1.4 Controlando el flujo

Sentencias y Bucles en:



### If ...elif...else

#### Indentación

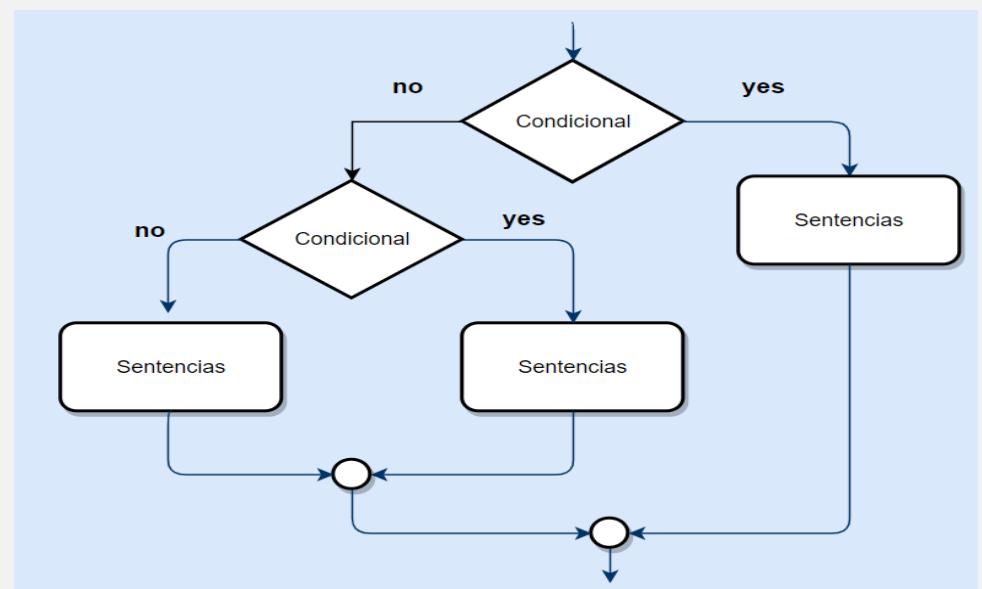
Python se basa en la sangría (espacio en blanco al comienzo de una línea) para definir el alcance en el código. Otros lenguajes de programación a menudo usan llaves para este propósito.

```
a = 33  
b = 200  
if b > a:  
  
    print("b es mayor que a") # La respuesta es error
```

#### Elif

La palabra clave **elif** es la forma de pitones de decir "si las condiciones anteriores no eran ciertas, entonces intente esta condición".

```
a = 33  
b = 200  
if b > a:  
    print("b es mayor que a")  
elif a == b:  
    print("b es igual que a")
```



## 1.4 Controlando el flujo

Sentencias y Bucles en:

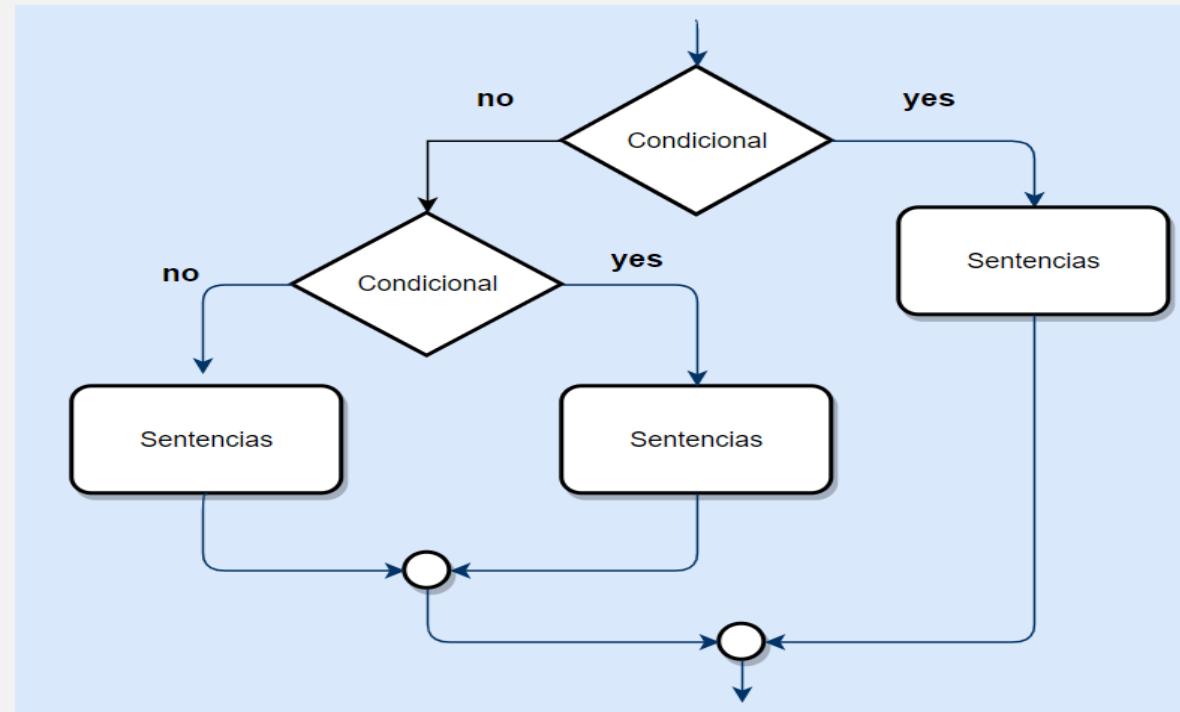


### If ...elif...else

#### Else

La palabra clave **else** captura cualquier cosa que no esté atrapada por las condiciones anteriores.

```
a = 33  
b = 200  
if b > a:  
    print("b es mayor que a")  
elif a == b:  
    print("b es igual que a")  
else:  
    print("a es mayor que b")
```



#### If sentencia corta (en una línea)

Si solo tiene que ejecutar una instrucción, puede ponerla en la misma línea que la instrucción if.

```
a=42  
b=200  
if b > a: print("b es mayor que a")
```

## 1.4 Controlando el flujo

Sentencias y Bucles en:



### If ...elif...else

#### If .. else sentencia corta (en una línea)

Si solo tiene que ejecutar una instrucción, una para if y otra para más, puede poner todo en la misma línea:

a=2

b=330

```
print("A") if a > b else print("B") # La respuesta es B
```

#### If .. elif ... else sentencia corta (en una línea)

Esta técnica se conoce como operadores ternarios o expresiones condicionales.

a=330

b=330

```
print("A") if a > b else print("=") if a == b else print("B") # La respuesta es =
```

### Condiciones lógicas: and y or

Esta técnica se conoce como operadores ternarios o expresiones condicionales.

a = 200

b = 33

c = 500

if a > b or a > c:

```
print("Al menos una de las condiciones es verdadera") # La respuesta es =
```

## 1.4 Controlando el flujo

Sentencias y Bucles en:



### If Anidados

Puede tener declaraciones if dentro de declaraciones if, esto se llama instrucciones if anidadas.

```
x = 41
if x > 10:
    print("Mas de diez,")
    if x > 20:
        print("y también es mayor a 20!")
    else:
        print("pero no mas de 20")      # La respuesta es Mas de diez, y también es mayor a 20!
```

### sentencia pass

Las sentencias **if** no pueden estar vacías, pero si por alguna razón tiene una sentencia **if** sin contenido, ingrese la sentencia **pass** para evitar un error.

```
a = 33
b = 200
if b > a:
    pass
```

# While ... loops

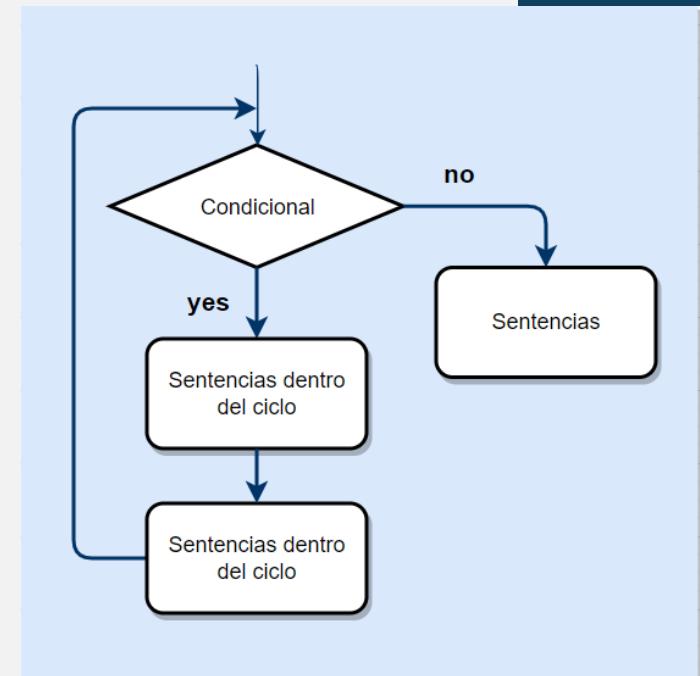
Con el ciclo **while** podemos ejecutar un conjunto de declaraciones siempre que una condición sea verdadera.

i = 1

while i < 6:

```
    print(i)
```

```
    i += 1
```



## Instrucción Break

Con la instrucción **break** podemos **detener** el ciclo incluso si la condición while es verdadera.

i = 1

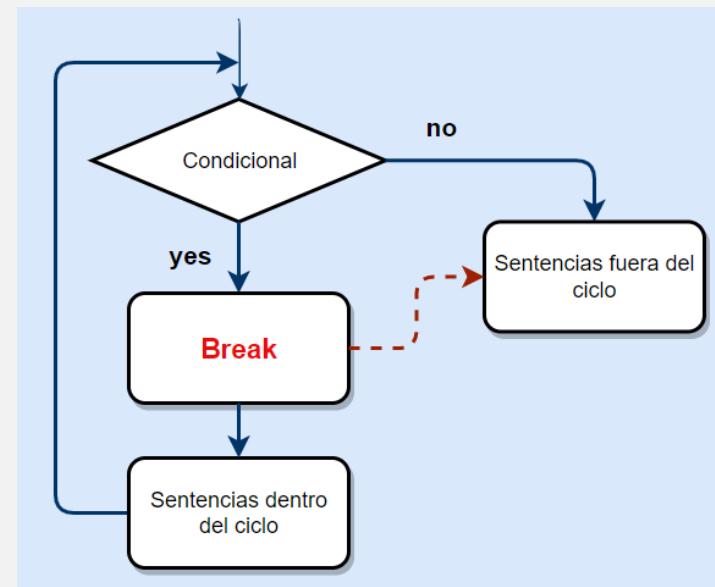
while i < 6:

```
    print(i)
```

```
    if i == 3:
```

```
        break
```

```
    i += 1
```

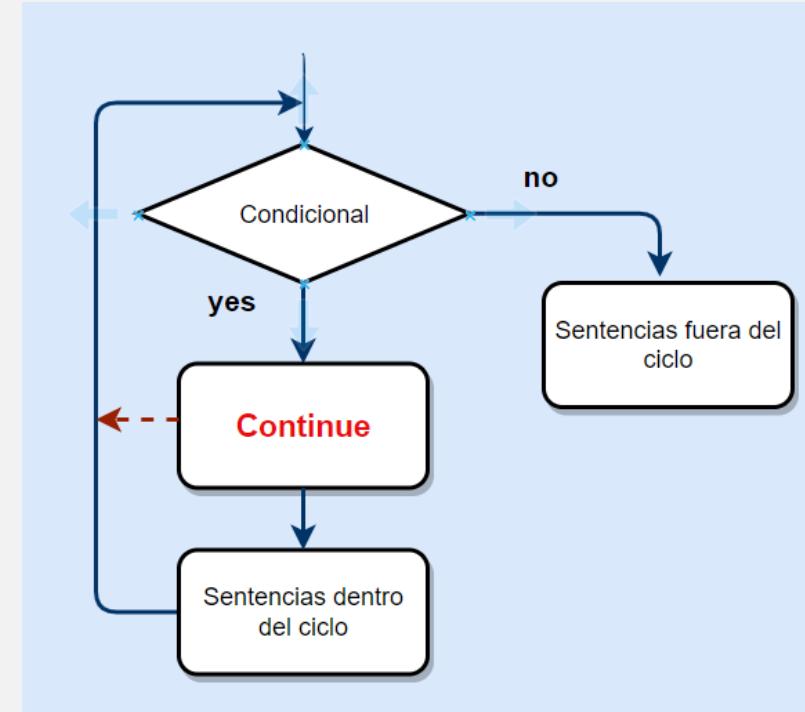


# While ... loops

## Instrucción Continue

Con la instrucción **continue** podemos detener la iteración actual y continuar con la siguiente:

```
i = 0  
while i < 6:  
    i += 1  
    if i == 3:  
        continue  
    print(i)
```



## Instrucción Else

Con la instrucción **else** podemos ejecutar un bloque de código una vez cuando la condición ya no es verdadera:

```
i = 1  
while i < 6:  
    print(i)  
    i += 1  
else:  
    print("ya no es menos de 6")
```



# For ... loops

Un bucle **for** se usa para iterar sobre una secuencia (que es una lista, una tupla, un diccionario, un conjunto o una cadena). Esto es menos como la palabra clave **for** en otros lenguajes de programación, y funciona más como un método iterador como se encuentra en otros lenguajes de programación orientada a objetos. Con el bucle **for** podemos ejecutar un conjunto de declaraciones, una vez para cada elemento de una lista, tupla, conjunto, etc.

```
frutas = ["apple", "banana", "cherry"]
for x in frutas:
    print(x)
```

El bucle for no requiere una variable de indexación para establecer de antemano

## Bucle a traves de una cadena (string)

Incluso las cadenas son objetos iterables, contienen una secuencia de caracteres:

```
for x in "banana":
    print(x)
```

# For ... loops

## Sentencia Break

Con la instrucción **break** podemos detener el ciclo antes de que haya pasado por todos los elementos:

```
fruits = ["apple", "banana", "cherry"]
```

```
for x in fruits:
```

```
    print(x)
```

```
    if x == "banana":
```

```
        break
```

## Instrucción Continue

Con la instrucción **continue** podemos detener la iteración actual del ciclo y continuar con la siguiente:

```
fruits = ["apple", "banana", "cherry"]
```

```
for x in fruits:
```

```
    if x == "banana":
```

```
        continue
```

```
    print(x)
```

# For ... loops

## La función Range()

Para recorrer un conjunto de código un número específico de veces, podemos usar la función range (), La función range () devuelve una secuencia de números, comenzando desde 0 por defecto, y se incrementa en 1 (por defecto), y termina en un número específico.

```
for x in range(6):  
    print(x)
```

Tenga en cuenta que el rango (6) no son los valores de 0 a 6, sino los valores de 0 a 5.

La función range () se establece por defecto en 0 como valor inicial, sin embargo, es posible especificar el valor inicial agregando un parámetro: range (2, 6), que significa valores de 2 a 6 (pero sin incluir 6):

```
for x in range(2, 6):  
    print(x)
```

# For ... loops

## Palabra clave Else

La palabra clave **else** en un bucle **for** especifica un bloque de código que se ejecutará cuando finalice el bucle:

```
for x in range(6):
    print(x)
else:
    print("Finally finished!")
```

## Bucles anidados (Nested Loops)

Un bucle anidado es un bucle dentro de un bucle. El "bucle interno" se ejecutará una vez por cada iteración del "bucle externo":

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
    for y in fruits:
        print(x, y)          # la respuesta es red apple
                               # red banana .....
```

## 1.4 Controlando el flujo

Sentencias y Bucles en:



### For ... loops

#### Modificar Listas

Se puede modificar listas con las siguientes instrucciones. Por ejemplo se tiene una lista de números y se quiere esta lista multiplicada por 10

```
indice = 0
```

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
for elemento in numeros:
```

```
    numeros[indice] *= 10
```

```
    indice+=1
```

```
numeros
```

# la respuesta es [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

#### Modificar Listas

Se puede modificar listas con las siguientes instrucciones. Por ejemplo se tiene una lista de números y se quiere esta lista multiplicada por 10 y ahora se hará con la función **enumerate()**

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
for indice, elemento in enumerate(numeros):
```

```
    print(indice, elemento)
```

```
    numeros[indice] *= 10
```

```
numeros
```

# la respuesta es [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

## 1.4 Controlando el flujo

### For y while en 1 linea

Sentencias y Bucles en:



```
y =0  
suma = 0  
num = int(input("Ingrese un numero:"))  
  
print()  
for x in range(int(num)):  
    suma += (x + 1 )  
print("Con (FOR) La suma es:", suma)
```

For en una linea:

```
print("Con (FOR): ", sum(x for x in range(1, num+1)))  
                                # la respuesta es [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  
nueva_lista = [print(x) for x in [1,2,3,4,5,6] if x < 5]
```

```
while y < num:  
    y += 1  
    suma += y
```

```
print("Con (WHILE):", suma)
```

## 1.5 Matrices



## TEMA 5

---

Matrices

# Matrices con Listas de Python

Las **Matrices** son disposiciones bidimensionales de valores en filas y columnas. Las **LISTAS** permiten serie de datos en una solo dimensión (se pueden emplear para **vectores**). Entonces una **lista de listas** de números pueden representar una matriz.

**Matriz**  
**3x3**

filas  
del 0  
al 2

		columnas del 0 al 2					
		-3	0	-2	1	-1	2
filas del 0 al 2	-3	0	12	1	15		
	-2	1	9	3		10	
	-1	2	14	7			6

Solo valores de la matriz

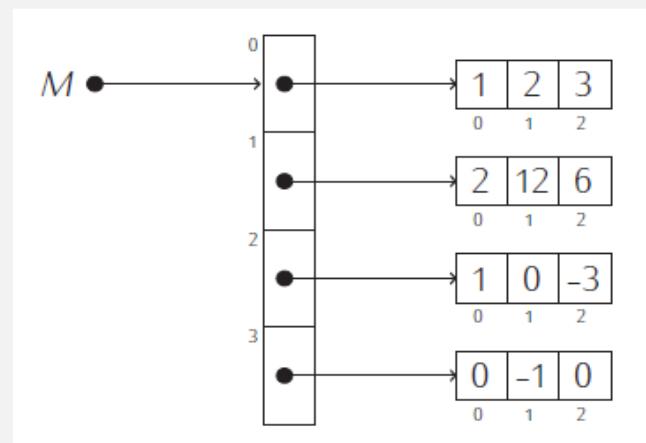
El elemento señalado de la matriz, se denota **M<sub>ij</sub>** o **M<sub>23</sub>** y en **Python** las filas y columnas **inician en cero** por lo que seria

$$\mathbf{M_{12} = 10}$$

# Matrices con Listas de Python

Ahora veremos las operaciones con matrices en Python, se tiene la siguientes matriz  $4 \times 3$  y en En Listas de Python se definiría como la figura del costado derecho

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 12 & 6 \\ 1 & 0 & -3 \\ 0 & -1 & 0 \end{pmatrix}$$



## ➤ Creación de matriz como listas.

```
M = [ [1, 2, 3], [2, 12, 6], [1, 0, -3], [0, -1, 0] ]
print(M)
print(M[0])
print(M[1][1])
```

La respuesta es [[1, 2, 3], [2, 12, 6], [1, 0, -3], [0, -1, 0]]  
 La respuesta es [1, 2, 3] es una lista o vector  
 La respuesta es 12 es un escalar

# Matrices con Listas de Python

- Recorridos de matrices

```
M=[[1, 0, 0], [0, 1, 0], [0, 0, 1]]  
print(M[-1][0]) # La respuesta es 0  
print(M[-1][-1]) # La respuesta es 1  
print('*30)  
for i in range (0, 3):  
    print(M[i])  
print('*30)  
for i in range(0, 3):  
    for j in range(0, 3):  
        print(M[i][j]) # La respuesta es cada vector
```

- Se aplican operaciones de Listas, por ejemplo para crear una matriz

```
M=[]  
for i in range (3):  
    a=[0]*6  
    M.append( a)  
print(M) # La respuesta es [ [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0] ]
```

# Matrices con Listas de Python

## ➤ Creación

```
M=[[1, 0, 0], [0, 1, 0], [0, 0, 1]]  
print(M[-1][0]) # La respuesta es 0  
print(M[-1][-1]) # La respuesta es 1  
print('-')  
for i in range (0, 3): # Con range el que varía es el índice i
```

## ➤ Longitud

```
M=[[1, 0, 0], [0, 1, 0], [0, 0, 1]]  
print(len(M)) # La respuesta es 3 porque es el numero de filas  
print(len(M[0])) # La respuesta es 3 porque es el numero de columnas
```

## ➤ Creación y carga de una matriz nxm

```
n = int(input("Ingrese el numero de filas: "))  
m = int(input("Ingrese el numero de columnas: "))  
A= []  
for i in range(n):  
    A.append( [0] * m )  
print(len(A[0]))  
  
for i in range (n):  
    for j in range(m):  
        A[i][j] = float(input("Componente ({0},{1}): ".format(i, j)))  
print(A)
```

# INSERTAR NOMBRE DE ACTIVIDAD



## PASOS DE ACTIVIDAD individual /grupal:

- Ingresar al colab personal con su cuenta gmail
  
- Copia el enunciado del problema señalado por el profesor y péguelo en el colab
  
- Desarrolle el ejercicio junto con un compañero sea en el laboratorio o en la sala grupal creada por el docente.

# Resolución de ejercicios

## ACTIVIDAD

Debe resolver los ejercicios señalados por el profesor que se encuentran en el documento:

**01 - 02.2 Fundamentos de Programación en Python  
- Guía laboratorio y Casos propuestos.pdf**

Este documento se encuentra en el aula virtual en la unidad 1



# CONCLUSIONES

**01** Python estándar ha sido desarrolladas con la arquitectura orientada a objetos.

**02** La arquitectura Orientada a Objetos, simplifica las instrucciones de control

**03** Los tipos de datos como Listas y Cadenas, tienen mas funcionalidades en comparación con otros lenguajes de programación

**04**

# BIBLIOGRAFÍA

- Direcciones electrónicas (hipervínculos en las diapositivas)
- LUTZ, MARK (2013) Learning Python. 5th Edition. California: O'Reilly.
- RAMALHO, LUCIANO (2015) Fluent Python: Clear, Concise, and Effective Programming (inglés) 1st Edición. California: O'Reilly.
-

# **Continúa con las actividades propuestas**

---



Material producido por la  
Universidad Peruana de  
Ciencias Aplicadas

Autor:

Norman Reyes Morales

COPYRIGHT © UPC  
2020 – Todos los  
derechos reservados



---

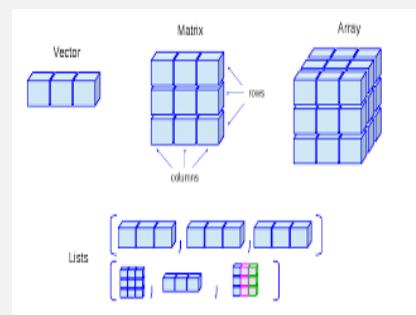
# **SOFTWARE PARA INGENIEROS**

## **UNIDAD N.º: 2 MANEJO DE DATOS Y OPTIMIZACIÓN**

## LOGRO DE APRENDIZAJE



Al finalizar la unidad, el estudiante aplica los diferentes tipos de almacenes de datos, la programación de funciones o subprogramas, y la excepción de errores de Python en un contexto real o simulado, demostrando una actitud analítica y organizada.



## TEMA 3

---

Colecciones de datos

## 2.1 Colecciones de datos

### Registros (tuplas)



Una tupla es una colección ordenada e **inmutable**. En Python, las tuplas se escriben entre paréntesis.

#### ➤ Crear una tupla

```
Estatupla = ("apple", "banana", "cherry", "orange ", "kiwi", "mango", "blueberry")
print(Estatupla) # la respuesta es ('apple', 'banana', 'cherry', 'orange', 'kiwi', 'mango', 'blueberry')
```

#### ➤ Acceso a los elementos de las tuplas

Puede acceder a los elementos de tupla haciendo referencia al número de índice, entre corchetes:

```
Estatupla = ("apple", "banana", "cherry", "orange")
print(Estatupla[1]) # la respuesta es banana
```

0	1	2	3	4	5	6
apple	banana	cherry	orange	kiwi	mango	blueberry
-7	-6	-5	-4	-3	-2	-1

#### ➤ Permite índice negativo.

```
print(Estatupla[-1]) # la respuesta es blueberry
```

#### ➤ Permite rangos de índices.

```
Estatupla = ("apple", "banana", "cherry", "orange ", "kiwi", "mango", "blueberry")
print( Estatupla[:4] ) # la respuesta es ('apple', 'banana', 'cherry', 'orange')
print( Estatupla[-4:-1] ) # la respuesta es ('orange', 'kiwi', 'mango')
```

## 2.1 Colecciones de datos



### Registros (tuplas)

Una tupla es una colección ordenada e **inmutable**. En Python, las tuplas se escriben entre paréntesis.

#### ➤ Modificar una **TUPLA**

NO SE PUEDE MODIFICAR LOS VALORES DE UNA TUPLA (INMUTABLES)

Pero hay una solución alternativa, que consiste en convertir la tupla en una lista, cambiar la lista y volver a convertir la lista en una tupla.

```
x = ("apple", "banana", "cherry")      # Se crea la tupla
y = list(x)                            # Se crea la lista a partir de la tupla
y[1] = "kiwi"                          # Se reemplaza el valor en el 2do elemento
x = tuple(y)                           # Se copia la lista modificada como tupla
print(x)                                # la respuesta es ('apple', 'kiwi', 'cherry')
```

#### ➤ Permite Loops o recorridos por los elementos

```
Estatupla = ("apple", "banana", "cherry")
for x in Estatupla:
    print (x)                            # la respuesta es apple
                                            banana
                                            Cherry
```

#### ➤ Verifica la existencia de un elemento. Por ejemplo si melon está en la lista

```
if "melon" in Estalista:
    print("Si, 'melon' está en la lista de frutas")
# la respuesta es Si, 'melon' está en la lista de frutas
```



### Registros (tuplas)

- Determinar la longitud de la tupla. Función len()

```
Estatupla = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print( len(Estatupla) )    # la respuesta es 7
```

- Agregar elementos.

No se puede, debe usarse el metodo alternativo

- Eliminar elementos.

No se puede.

## 2.1 Colecciones de datos



### Registros (tuplas)

- Determinar la longitud de la tupla. Función len()

```
Estatupla = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print( len(Estatupla) )    # la respuesta es 7
```

- Agregar y eliminar elementos.

No se puede, debe usarse el metodo alternativo en el caso de agregar

- Unir dos tuplas

Mediante la forma de concatenar (+)

```
tupla1 = ("a", "b", "c")  
tupla2 = (1, 2, 3)  
tupla3 = tupla1 + tupla2  
print(tupla3)      # la respuesta es ('a', 'b', 'c', 1, 2, 3)
```

- Función de construir una tupla con el método tuple()

```
Estatupla = tuple( ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango") )  
print(Estatupla)      # la respuesta es ('apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango')
```

## 2.1 Colecciones de datos

### Registros ([tuplas](#))



- Python cuenta con dos Métodos de tuplas, ya construidos

Method	Description
--------	-------------

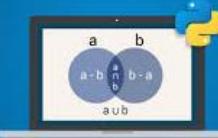
<a href="#">count()</a>	Returns the number of times a specified value occurs in a tuple
-------------------------	---

<a href="#">index()</a>	Searches the tuple for a specified value and returns the position of where it was found
-------------------------	---

## 2.1 Colecciones de datos

### Conjuntos ([sets](#))

Sets in Python



Un conjunto es una colección que no está ordenada ni indexada. En Python, los conjuntos se escriben entre llaves y hay un **elemento único**.

```
Esteset = {"apple", "banana", "cherry", "orange", "kiwi", "mango", "blueberry", "banana", "apple", "Orange"}  
print(Esteset)          # la respuesta es {'kiwi', 'orange', 'Orange', 'apple', 'banana', 'cherry', 'mango', 'blueberry'}
```

#### ➤ Acceso a los elementos de los conjuntos

No puede acceder a los elementos de un conjunto haciendo referencia a un índice, ya que los conjuntos están desordenados y los elementos no tienen índice. Pero puede recorrer los elementos del conjunto utilizando un bucle for, o preguntar si un valor especificado está presente en un conjunto, utilizando la palabra clave in.

```
Esteset = {"apple", "banana", "cherry"}  
for x in Esteset:  
    print(x)           # la respuesta es apple  
                      #                         banana  
                      #                         cherry
```

#### ➤ Preguntar si hay un elemento. Por ejemplo si banana esta en el conjunto

```
print("banana" in Esteset)          # la respuesta es True
```

## 2.1 Colecciones de datos

### Conjuntos (sets)

- Modificar una **Sets**

NO SE PUEDE MODIFICAR LOS VALORES DE UN SET

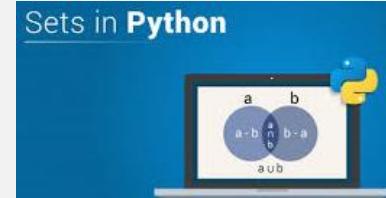
- Agregar elementos.

Para agregar un elemento a un conjunto, use el método **add()**.

```
Esteset = {"apple", "banana", "cherry"}           # Se crea el set  
Esteset.add("orange")                            # Se agrega elemento  
print(Esteset)                                  # la respuesta es {'banana', 'apple', 'orange', 'cherry'}
```

Para agregar más de un elemento a un conjunto, use el método **update()**.

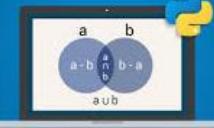
```
Esteset = {"apple", "banana", "cherry"}           # Se crea el set  
Esteset.update(["orange", "mango", "grapes"])    # Se agrega elemento  
print(Esteset)                                  # la respuesta es {'banana', 'orange', 'apple',  
                 'grapes', 'mango', 'cherry'}
```



## 2.1 Colecciones de datos

### Conjuntos ([sets](#))

Sets in Python



- Vaciar el conjunto. Se usa el método clear

```
Esteset = {"apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"}
```

```
Esteset.clear()
```

```
print( (Esteset) ) # la respuesta es set()
```

- Eliminar el set.

```
Esteset = {"apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"}
```

```
del Esteset
```

```
print( (Esteset) ) # la respuesta es error Esteset no está definido
```

- Unir dos sets

Hay varias formas de unir dos o más conjuntos en Python. Puede usar el método union () que devuelve un nuevo conjunto que contiene todos los elementos de ambos conjuntos, o el método update () que inserta todos los elementos de un conjunto en otro:

Mediante el método unión()

```
set1 = {"a", "b" , "c"}
```

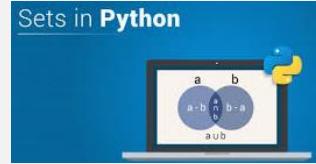
```
set2 = {1, 2, 3}
```

```
set3 = set1.union(set2)
```

```
print(set3) # la respuesta es {1, 2, 'c', 3, 'b', 'a'}
```

## 2.1 Colecciones de datos

### Conjuntos (sets)



#### ➤ Unir dos sets

Hay varias formas de unir dos o más conjuntos en Python. Puede usar el método `union()` que devuelve un nuevo conjunto que contiene todos los elementos de ambos conjuntos, o el método `update()` que inserta todos los elementos de un conjunto en otro:

Mediante el método `update()`

```
set1 = {"a", "b", "c"}  
set2 = {1, 2, 3}  
set1.update(set2)  
print(set1) # la respuesta es {1, 2, 'c', 3, 'b', 'a'}
```

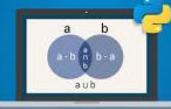
Nota: Tanto `union()` como `update()` excluirán cualquier elemento duplicado.

#### ➤ Función de construir un conjunto con el método `set()`

```
Esteset = set(("apple", "banana", "cherry"))  
print(Esteset) # la respuesta es {'banana', 'apple', 'cherry'}
```

## 2.1 Colecciones de datos

Sets in Python



# Conjuntos ([set](#))

Method	Description
<a href="#">add()</a>	Adds an element to the set
<a href="#">clear()</a>	Removes all the elements from the set
<a href="#">copy()</a>	Returns a copy of the set
<a href="#">difference()</a>	Returns a set containing the difference between two or more sets
<a href="#">difference_update()</a>	Removes the items in this set that are also included in another, specified set
<a href="#">discard()</a>	Remove the specified item
<a href="#">intersection()</a>	Returns a set, that is the intersection of two other sets
<a href="#">intersection_update()</a>	Removes the items in this set that are not present in other, specified set(s)
<a href="#">isdisjoint()</a>	Returns whether two sets have a intersection or not
<a href="#">issubset()</a>	Returns whether another set contains this set or not
<a href="#">issuperset()</a>	Returns whether this set contains another set or not
<a href="#">pop()</a>	Removes an element from the set
<a href="#">remove()</a>	Removes the specified element
<a href="#">symmetric_difference()</a>	Returns a set with the symmetric differences of two sets
<a href="#">symmetric_difference_update()</a>	inserts the symmetric differences from this set and another
<a href="#">union()</a>	Return a set containing the union of sets
<a href="#">update()</a>	Update the set with the union of this set and others

## 2.1 Colecciones de datos

### Diccionarios ([Dictionary](#))



Un diccionario es una colección que está desordenada, modificable e indexada. En Python, los diccionarios se escriben entre llaves y tienen claves y valores.

```
Estedicc = {
```

```
    "marca": "Ford",  
    "modelo": "Mustang",  
    "año": 2019  
}
```

marca	modelo	año
Ford	Mustang	2019

```
print(Estedicc)
```

# la respuesta es {'marca': 'Ford', 'modelo': 'Mustang', 'año': 2019}

#### ➤ Acceso a los elementos de los conjuntos

Puede acceder a los elementos de un diccionario haciendo referencia a su nombre clave, entre corchetes:

```
x = Estdicc["modelo"]
```

```
print(x)
```

# la respuesta es Mustang

#### ➤ También hay un método llamado get () que le dará el mismo resultado:

```
x = Estdicc.get("modelo")
```

```
print(x)
```

# la respuesta es Mustang

## 2.1 Colecciones de datos



### Diccionarios (Dictionary)

#### ➤ Modificar una Dict

Puede cambiar el valor de un elemento específico haciendo referencia a su nombre clave:

```
Estedicc = {  
    "marca": "Ford",  
    "modelo": "Mustang",  
    "año": 2019  
}
```

```
Estedicc["año"] = 2018  
print(Estedicc)
```

Marca	modelo	año
Ford	Mustang	2019

# la respuesta es {'marca': 'Ford', 'modelo': 'Mustang', 'año': 2018}

#### ➤ Se puede recorrer un diccionario un bucle For

```
for x in Estedicc:  
    print(x)
```

# la respuesta es marca  
modelo  
año

para imprimir todos los valores del diccionario  
for x in Estedicc:  
 print(Estedicc[x])

# la respuesta es Ford  
Mustang  
2018

## 2.1 Colecciones de datos

### Diccionarios (Dictionary)

- Se puede recorrer un diccionario un bucle For para que me den todos sus valores

```
for x in Estedicc.values():
    print( x) # la respuesta es
```

Ford

Mustang

2018

- Recorra tanto las claves como los valores, utilizando el método items ():

```
for x, y in Estedicc.items():
    print( x, y) # la respuesta es
```

marca Ford

modelo Mustang

año 2018

- Verificar la existencia de un elemento.

Para determinar si una **clave específica** está presente en un diccionario, use la palabra clave in:

```
Estedicc = {
    "marca": "Ford",
    "modelo": "Mustang",
    "año": 2019
}
```

```
if 'modelo' in Estedicc:
    print("Si, 'modelo' es una de las claves de este Estedicc dictionary")
```

- Longitud de un diccionario.

Para determinar cuántos elementos (pares clave-valor) tiene un diccionario, use la función len () .



## 2.1 Colecciones de datos

### Diccionarios (Dictionary)

#### ➤ Agregar elementos

La adición de un elemento al diccionario se realiza utilizando una nueva clave de índice y asignándole un valor:

```
Estedicc = {  
    "marca": "Ford",  
    "modelo": "Mustang",  
    "año": 2019  
}  
Estedicc["color"] = "rojo"  
print(Estedicc) # la respuesta es {'marca': 'Ford', 'modelo': 'Mustang', 'año': 2019, 'color': 'rojo'}
```

#### ➤ Eliminar elementos.

Existen varios métodos para eliminar elementos de un diccionario:

Empleando el método `pop()` y especificando el nombre de la clave. Por ejemplo del diccionario anterior, se quiere eliminar el modelo

```
Estedicc.pop("modelo")  
print(Estedicc) # la respuesta es {'marca': 'Ford', 'año': 2019, 'color': 'rojo'}
```



## 2.1 Colecciones de datos

### Diccionarios (Dictionary)

- Eliminar elementos.

El método `popitem()` elimina el último elemento insertado (en versiones anteriores a 3.7, se elimina un elemento aleatorio):

```
Estedicc = {  
    "marca": "Ford",  
    "modelo": "Mustang",  
    "año": 2019,  
    "color" : "rojo"  
}  
  
Estedicc.popitem()  
print(Estedicc) # la respuesta es {'marca': 'Ford', 'modelo': 'Mustang', 'año': 2019}
```

El método `del` con una clave específica

```
Estedicc = {  
    "marca": "Ford",  
    "modelo": "Mustang",  
    "año": 2019,  
    "color" : "rojo"  
}  
  
del Estedicc["modelo"]  
print(Estedicc) # la respuesta es {'marca': 'Ford', 'año': 2019, 'color': 'rojo'}
```



## 2.1 Colecciones de datos

### Diccionarios (Dictionary)

➤ Eliminar el diccionario.

El método **del** eliminando por completo el diccionario

```
Estedicc = {  
    "marca": "Ford",  
    "modelo": "Mustang",  
    "año": 2019,  
    "color" : "rojo"  
}
```

del Estedicc

```
print(Estedicc) # la respuesta es Mensaje de error
```

El método **clear** que dejara en blanco al diccionario

```
Estedicc.clear()  
print(Estedicc) # la respuesta es {}
```



## 2.1 Colecciones de datos

### Diccionarios (Dictionary)



#### ➤ Diccionarios anidados

Un diccionario también puede contener muchos diccionarios, esto se llama diccionarios anidados.:

Ejemplo de diccionario que contiene tres diccionarios

```
mifamilia = {  
    "hijo1" : {  
        "nombre" : "Emil",  
        "año" : 2004  
    },  
    "hijo2" : {  
        "nombre" : "Tobias",  
        "año" : 2007  
    },  
    "hijo3" : {  
        "nombre" : "Linus",  
        "año" : 2011  
    }  
}  
  
print(mifamilia)      # la respuesta es {'hijo1': {'nombre': 'Emil', 'año': 2004}, 'hijo2': {'nombre': 'Tobias', 'año': 2007}, 'hijo3': {'nombre': 'Linus', 'año': 2011}}
```

## 2.1 Colecciones de datos

### Diccionarios (Dictionary)

#### ➤ Diccionarios anidados

Si queremos anidar los tres diccionarios ya existentes, se puede hacer de la siguiente forma

```
hijo1 = {  
    "nombre" : "Emil",  
    "año" : 2004  
}  
  
hijo2 = {  
    "nombre" : "Tobias",  
    "año" : 2007  
}  
  
hijo3 = {  
    "nombre" : "Linus",  
    "año" : 2011  
}  
  
mifamilia = {  
    "hijo1" : hijo1,  
    "hijo2" : hijo2,  
    "hijo3" : hijo3  
} # la respuesta es {'hijo1': {'nombre': 'Emil', 'año': 2004}, 'hijo2': {'nombre': 'Tobias', 'año': 2007}, 'hijo3': {'nombre': 'Linus', 'año': 2011}}  
  
print(mifamilia)
```



# Diccionarios (Dictionary)



## ➤ Construir el diccionario

También es posible usar el constructor dict() para crear un nuevo diccionario:

```
Estedicc=dict(marca="Ford", modelo="Mustang", año= 2019, color="rojo")
```

# tenga en cuenta que las claves no son literales de cadena

# tenga en cuenta el uso de iguales en lugar de dos puntos para la asignación

```
print(Estdicc)           # la respuesta es {'marca': 'Ford', 'modelo': 'Mustang', 'año': 2019, 'color': 'rojo'}
```

## 2.1 Colecciones de datos

### Diccionarios ([Dictionary](#))



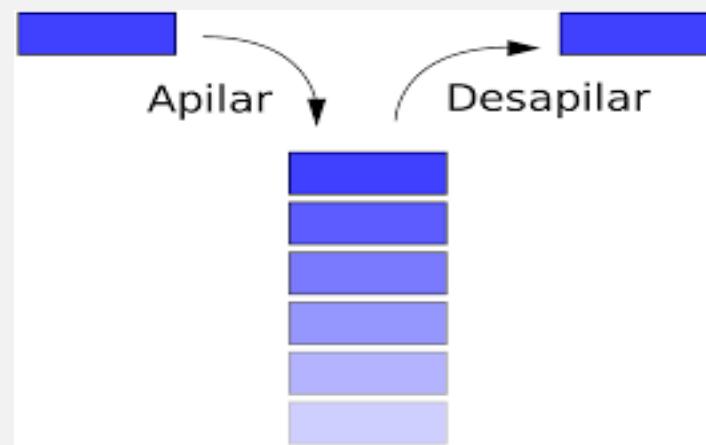
- Python cuenta con los siguientes Métodos de conjuntos, ya construidos

Method	Description
<a href="#">clear()</a>	Removes all the elements from the dictionary
<a href="#">copy()</a>	Returns a copy of the dictionary
<a href="#">fromkeys()</a>	Returns a dictionary with the specified keys and value
<a href="#">get()</a>	Returns the value of the specified key
<a href="#">items()</a>	Returns a list containing a tuple for each key value pair
<a href="#">keys()</a>	Returns a list containing the dictionary's keys
<a href="#">pop()</a>	Removes the element with the specified key
<a href="#">popitem()</a>	Removes the last inserted key-value pair
<a href="#">setdefault()</a>	Returns the value of the specified key. If the key does not exist, insert the key, with the specified value
<a href="#">update()</a>	Updates the dictionary with the specified key-value pairs
<a href="#">values()</a>	Returns a list of all the values in the dictionary

# Pilas (stacks) emulación con Listas de Python

Una **PILA** es parte de los TDA (Tipos abstractos de Datos) es una LISTA ordenada en la que el modo de acceso a sus elementos es de tipo **LIFO**(Last In First Out) que permite almacenar y recuperar datos.

## Pilas



### Operaciones básicas

- **Push (apilar)**
- **Pop (desapilar)**

### Otras Operaciones

- **Crear**
- **Tamaño**
- **Cima**
- **Vacia**

➤ Uso en informática: ¿En qué o para qué se utilizan las Pilas en informática?

- Evaluación de expresiones en notación postfija (notación polaca inversa)
- Reconocedores sintácticos de lenguajes independientes del contexto
- Implementación de recursividad
- **Solucionar problemas de búsqueda como Branch and bound, y soluciones heurísticas . Otras aplicaciones son resolver un crucigrama o jugar ajedrez**

# Pilas (stacks) emulación con Listas de Python



## ➤ Crear

Mediante una lista

```
Pila = [3, 4, 5]
```

## ➤ Añadir elementos.

El método a usar es el **append()**

```
Pila.append(6)  
Pila.append(7)  
print(Pila)
```

# la respuesta es [3, 4, 5, 6, 7]

## ➤ Extraer elementos.

El método a usar es el **pop()**

```
Pila.pop()  
print(Pila)
```

# la respuesta es 7

# la respuesta es [3, 4, 5, 6]

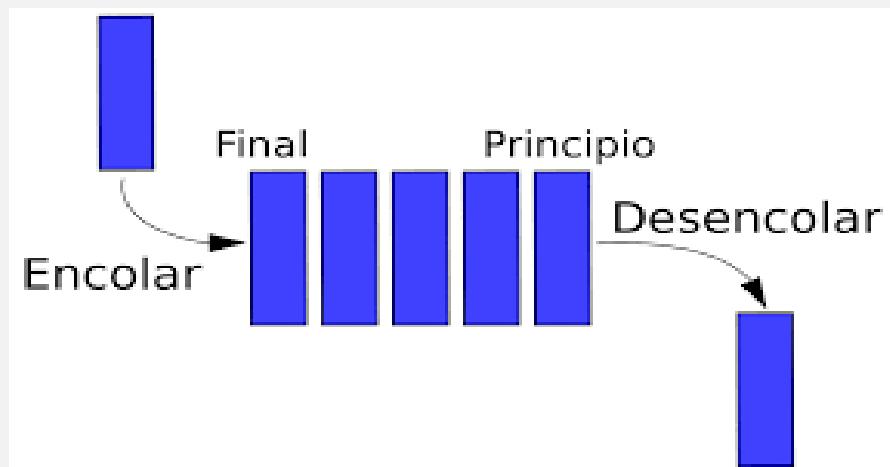
## ➤ Longitud de la pila o numero de elementos.

El método a usar es el **len()**

# Colas (queue) de Python

Una **COLA** (QUEUE) o FILA es otro TDA (Tipos abstractos de Datos), es una estructura de datos, caracterizada por ser una secuencia de elementos en la que la operación de inserción push se realizar por un extremo y la operación de extracción se realiza por otro extremo. También se le llama estructura **FIFO (First in First Out)**, debido a que el primer elemento en entrar será también el primero en salir

## Colas



### Operaciones básicas

- **Encolar (entrada)**
- **Desencolar (salida)**

### Otras Operaciones

- **Crear**
- **Tamaño**
- **frente**
- **Vacia**

#### ➤ Uso en informática:

- Se utilizan mucho en sistemas informáticos, transportes e investigación de operaciones, donde los objetos o eventos son tomados como datos que se almacenan y se guardan mediante colas para su posterior procesamiento

# Colas (queue) de Python



- Para el manejo de colas, primero debemos importar la librería **collections**.

```
from collections import deque # Debe ir en la parte superior de código antes de utilizar las colas
```

- Creación de colas.

El método a usar es el **deque()**  
cola = deque()  
print(cola)

La respuesta es **deque([])**

- Cargar datos o encolar inicialmente.

El método a usar es el **deque()**

```
cola = deque(["Hector", "Juan", "Miguel"])  
print(cola)
```

La respuesta es **deque(['Hector', 'Juan', 'Miguel'])**

- Agregar elementos.

El método a usar es el **append()**

```
cola.append("Maria")  
cola.append("Arnaldo")  
print(cola)
```

La respuesta es **deque(['Hector', 'Juan', 'Miguel', 'Maria', 'Arnaldo'])**

# INSERTAR NOMBRE DE ACTIVIDAD



## PASOS DE ACTIVIDAD individual /grupal:

- Ingresar al colab personal con su cuenta gmail
  
- Copia el enunciado del problema señalado por el profesor y péguelo en el colab
  
- Desarrolle el ejercicio junto con un compañero sea en el laboratorio o en la sala grupal creada por el docente.

# Resolución de ejercicios

## ACTIVIDAD

Debe resolver los ejercicios señalados por el profesor que se encuentran en el documento:

[02 - 01.2 Manejo de datos y optimización - Guía laboratorio y Casos propuestos.pdf](#)

Este documento se encuentra en el aula virtual en la unidad 2



## CONCLUSIONES

01

02

03

04

# BIBLIOGRAFÍA

- Direcciones electrónicas (hipervínculos en las diapositivas)
- LUTZ, MARK (2013) Learning Python. 5th Edition. California: O'Reilly.
- RAMALHO, LUCIANO (2015) Fluent Python: Clear, Concise, and Effective Programming (inglés) 1st Edición. California: O'Reilly.
-



# Continúa con las actividades propuestas

---

Material producido por la Universidad Peruana de Ciencias Aplicadas

Autor:

Colocar nombre del docente

COPYRIGHT © UPC  
2020 – Todos los derechos reservados



---

# **SOFTWARE PARA INGENIEROS**

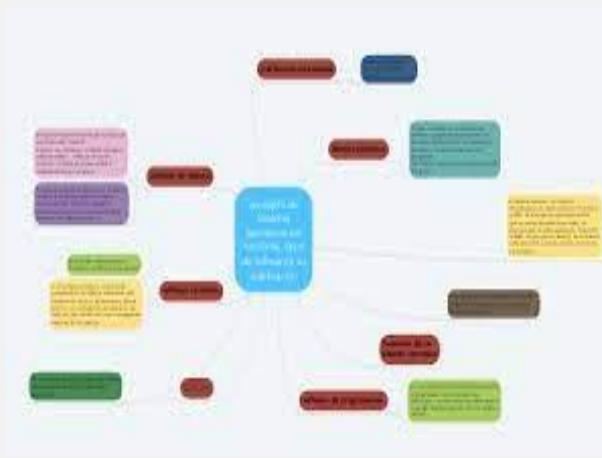
## **UNIDAD N.º: 2 MANEJO DE DATOS Y OPTIMIZACIÓN**

## LOGRO DE APRENDIZAJE



Al finalizar la unidad, el estudiante aplica los diferentes tipos de almacenes de datos, la programación de funciones o subprogramas, y la excepción de errores de Python en un contexto real o simulado, demostrando una actitud analítica y organizada.

## 2.2 Programación de funciones y/o subprogramas



# TEMA 4

# Programación de funciones y/o subprogramas

## 2.2 Programación de funciones y/o subprogramas



Una **función o subprograma** es un bloque de código que solo se ejecuta cuando se llama o invoca.

Puede pasar datos, conocidos como parámetros, a una función. Una función puede devolver datos como resultado.

- Crear y llamar una función

Una función se crea usando la palabra clave **def**

```
def mi_funcion():
    print("Hola desde una función")
```

# Acá se llama a la función mi\_funcion()

- Argumentos en una función

La información se puede pasar a las funciones como argumentos. Los **argumentos** se especifican después del nombre de la función, dentro de los paréntesis. Puede agregar tantos argumentos como desee, simplemente sepárelos con una coma.

El siguiente ejemplo tiene una función con un argumento (**fnombre**). Cuando se llama a la función, pasamos un nombre que se usa dentro de la función para imprimir el nombre completo:

```
def mi_funcion(nombre, apellido):
```

```
    print("El nombre es " + nombre + " " + apellido)
```

```
mi_funcion('Carla', 'Garcia')
```

# Acá se llama a la función mi\_funcion() con dos parámetros

o

# argumentos nombre y apellido

## 2.2 Programación de funciones y/o subprogramas

### ➤ Argumentos en una función

Si no sabe cuántos argumentos se pasarán a su función, agregue un **\*** antes del nombre del parámetro en la definición de la función. De esta forma, la función recibirá una tupla de argumentos y podrá acceder a los elementos en consecuencia:



```
def mi_funcion(*chicos):
    print("Los menores son" + chicos[2])
```

```
mi_funcion("Carla", "Mathias", "Fabiana")
```

# Acá se llama a la función `mi_funcion()`

### ➤ Argumentos claves arbitrarios

Si no sabe cuántos argumentos de palabras clave se pasarán a su función, agregue dos asteriscos: **\*\*** antes del nombre del parámetro en la definición de la función. De esta forma, la función recibirá un diccionario de argumentos y podrá acceder a los elementos en consecuencia:

```
def mi_funcion(**chicos):
    print("El apellido de los menores son " + chicos["apellido"] + ", " + chicos["nombre"])
```

```
mi_funcion(apellido="Rentería", nombre="Carla")
```

# la respuesta es **El apellido de los menores son Rentería,**

**Carla**

## 2.2 Programación de funciones y/o subprogramas

### ➤ Argumentos por defecto

El siguiente ejemplo muestra cómo usar un valor de parámetro predeterminado o por defecto.  
Si llamamos a la función sin argumento, utiliza el valor predeterminado:



```
def mi_funcion(pais="Perú"):
    print("Yo soy de" + pais)

mi_funcion()          # La respuesta es Perú
mi_funcion("Alemania") # La respuesta es Alemania
```

### ➤ Pasando una LISTA como argumentos

Puede enviar cualquier tipo de argumento de datos a una función (cadena, número, lista, diccionario, etc.), y se tratará como el mismo tipo de datos dentro de la función.

```
def mi_función(platos):
    for x in platos:
        print(x)

platos = ["arroz con pollo", "cebiche", "ají de gallina"]
mi_funcion(platos)      # la respuesta es El apellido de los menores son Rentería, Carla
```

## 2.2 Programación de funciones y/o subprogramas

- Retornar valores por la función

Se va a retornar una resultado con la sentencia **return**:

```
def mi_funcion(x):  
    return x * 5  
  
print(mi_funcion(3))                      # La respuesta es 15
```



## 2.2 Programación de funciones y/o subprogramas



### Variables locales y globales

Si se asigna o define una variable en el programa, se asumirá que es global.

- x= “Sorprendente” o ‘Sorprendente’

```
def myfunc():
    print("Python es " + x)
myfunc()
```

### Globales

Se puede crear una variable fuera y otra del mismo nombre dentro de la función. La que fue creada dentro de la función será local y tomará dicho valor en la función. La variable global mantendrá el valor inicial. Terminada la ejecución de la función, la variable local desaparecerá, y quedará la global con su valor original.

- x= ‘Sorprendente’

```
def myfunc():
    x = “genial”
    print("Python es " + x)
```

```
myfunc()
print("Python es " + x)
```

```
➤ def myfunc():
    Global x
    x=“genial” # Se declara con la palabra Global
    print("Python es " + x) # Global es palabra reservada
```

```
myfunc()
print("Python es " + x)
```

## 2.3 Gestión de errores



### **TEMA 5**

---

Gestión de errores

## 2.3 Gestión de errores



### ➤ Errores Sintácticos

El tipo de error más obvio es el **sintáctico**, que se produce cuando se escribe código de una forma no admitida por las **reglas del lenguaje**. Los errores de sintaxis son **detectados** casi siempre por el **compilador o intérprete**, que muestra un mensaje de error que informa del problema.

### ➤ Errores Semánticos

Los errores semánticos son más sutiles. Un error semántico se produce cuando la sintaxis del código es correcta, pero la semántica o significado no es el que se pretendía. La construcción obedece las reglas del lenguaje, y por ello el compilador o intérprete no detectan los errores semánticos. Los compiladores e intérpretes sólo se ocupan de la estructura del código que se escribe, y no de su significado. Un error semántico puede hacer que el programa termine de forma anormal, con o sin un mensaje de error.

Hablando en términos coloquiales, puede hacer que el equipo se quede “colgado”.

Sin embargo, no todos los errores semánticos se manifiestan de una forma tan obvia. Un programa puede continuar en ejecución después de haberse producido errores semánticos, pero su estado interno puede ser distinto del esperado. Quizá las variables no contengan los datos correctos, o bien es posible que el programa siga un camino distinto del pretendido. Eventualmente, la consecuencia será un resultado incorrecto. Estos **errores** se denominan **lógicos**, ya que aunque el programa no se bloquea, la lógica que representan contiene un error.

```
n=0; m=0
print(n/m) # división entre un cero
```

```
In [234]: n=0; m=0
           print(n/m)

-----
ZeroDivisionError
<ipython-input-234-ec32a950f4f4> in <module>
      1 n=0; m=0
      2 print(n/m)
-----
ZeroDivisionError: division by zero
```

## 2.3 Gestión de errores



### ➤ Estados de excepción

El objetivo es manejar estos errores para que no se detenga el programa. Se manejarán las siguientes instrucciones **try: , except: , else: y finally:** veamos el siguiente ejemplo

```
while(True):
    try:
        n = float(input("Ingrese un numero"))
                    # Acá le ingresamos 'aaaa', se produce un error en la conversión
        m = 4
        print("{} / {} = {}".format(n, m, n/m))
    except:
        print("Ha ocurrido un error interno, introduce bien el número")
                    # al presentar el error, automáticamente se ejecuta
                    # esta sentencia
    else:
        print("Todo ha funcionando correctamente")
                    break
                    # esta sentencia
    finally:
        print("Fin de la iteración")
```

# INSERTAR NOMBRE DE ACTIVIDAD



## PASOS DE ACTIVIDAD individual /grupal:

- Ingresar al colab personal con su cuenta gmail
  
- Copia el enunciado del problema señalado por el profesor y péguelo en el colab
  
- Desarrolle el ejercicio junto con un compañero sea en el laboratorio o en la sala grupal creada por el docente.

# Resolución de ejercicios

## ACTIVIDAD

Debe resolver los ejercicios señalados por el profesor que se encuentran en el documento:

[02 - 01.2 Manejo de datos y optimización - Guía laboratorio y Casos propuestos.pdf](#)

Este documento se encuentra en el aula virtual en la unidad 2



## CONCLUSIONES

01

02

03

04

# BIBLIOGRAFÍA

- Direcciones electrónicas (hipervínculos en las diapositivas)
- LUTZ, MARK (2013) Learning Python. 5th Edition. California: O'Reilly.
- RAMALHO, LUCIANO (2015) Fluent Python: Clear, Concise, and Effective Programming (inglés) 1st Edición. California: O'Reilly.
-

# **Continúa con las actividades propuestas**

---



Material producido por la  
Universidad Peruana de  
Ciencias Aplicadas

Autor:

Colocar nombre del docente

COPYRIGHT © UPC  
2020 – Todos los  
derechos reservados



---

# **SOFTWARE PARA INGENIEROS**

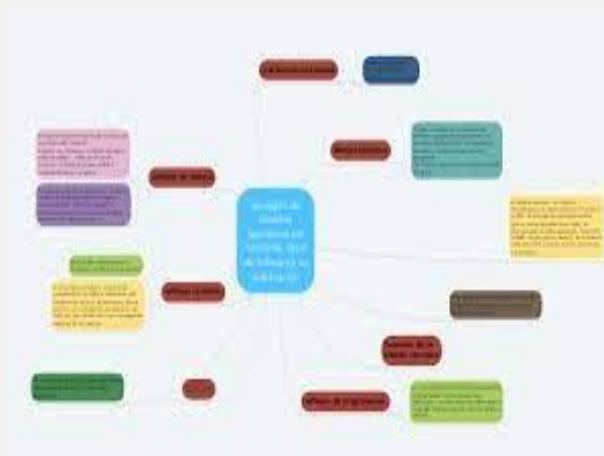
## **UNIDAD N.º: 2 MANEJO DE DATOS Y OPTIMIZACIÓN**

## LOGRO DE APRENDIZAJE



Al finalizar la unidad, el estudiante aplica los diferentes tipos de almacenes de datos, la programación de funciones o subprogramas, y la excepción de errores de Python en un contexto real o simulado, demostrando una actitud analítica y organizada.

## 2.2 Programación de funciones y/o subprogramas



# TEMA 4

## Programación de funciones y/o subprogramas

## 2.2 Programación de funciones y/o subprogramas



Una **función o subprograma** es un bloque de código que solo se ejecuta cuando se llama o invoca.

Puede pasar datos, conocidos como parámetros, a una función. Una función puede devolver datos como resultado.

- Crear y llamar una función

Una función se crea usando la palabra clave **def**

```
def mi_funcion():
    print("Hola desde una función")
```

# Acá se llama a la función mi\_funcion()

- Argumentos en una función

La información se puede pasar a las funciones como argumentos. Los **argumentos** se especifican después del nombre de la función, dentro de los paréntesis. Puede agregar tantos argumentos como desee, simplemente sepárelos con una coma.

El siguiente ejemplo tiene una función con un argumento (**fnombre**). Cuando se llama a la función, pasamos un nombre que se usa dentro de la función para imprimir el nombre completo:

```
def mi_funcion(nombre, apellido):
```

```
    print("El nombre es " + nombre + " " + apellido)
```

```
mi_funcion('Carla', 'Garcia')
```

# Acá se llama a la función mi\_funcion() con dos parámetros  
o  
# argumentos nombre y apellido

## 2.2 Programación de funciones y/o subprogramas

### ➤ Argumentos en una función

Si no sabe cuántos argumentos se pasarán a su función, agregue un **\*** antes del nombre del parámetro en la definición de la función. De esta forma, la función recibirá una tupla de argumentos y podrá acceder a los elementos en consecuencia:



```
def mi_funcion(*chicos):  
    print("Los menores son" + chicos[2])
```

```
mi_funcion("Carla", "Mathias", "Fabiana")
```

# Acá se llama a la función `mi_funcion()`

### ➤ Argumentos claves arbitrarios

Si no sabe cuántos argumentos de palabras clave se pasarán a su función, agregue dos asteriscos: **\*\*** antes del nombre del parámetro en la definición de la función. De esta forma, la función recibirá un diccionario de argumentos y podrá acceder a los elementos en consecuencia:

```
def mi_funcion(**chicos):  
    print("El apellido de los menores son " + chicos["apellido"] + ", " + chicos["nombre"])
```

```
mi_funcion(apellido="Rentería", nombre="Carla")
```

# la respuesta es **El apellido de los menores son Rentería,**

**Carla**

## 2.2 Programación de funciones y/o subprogramas

### ➤ Argumentos por defecto

El siguiente ejemplo muestra cómo usar un valor de parámetro predeterminado o por defecto.  
Si llamamos a la función sin argumento, utiliza el valor predeterminado:



```
def mi_funcion(pais="Perú"):
    print("Yo soy de" + pais)

mi_funcion()          # La respuesta es Perú
mi_funcion("Alemania") # La respuesta es Alemania
```

### ➤ Pasando una LISTA como argumentos

Puede enviar cualquier tipo de argumento de datos a una función (cadena, número, lista, diccionario, etc.), y se tratará como el mismo tipo de datos dentro de la función.

```
def mi_función(platos):
    for x in platos:
        print(x)

platos = ["arroz con pollo", "cebiche", "ají de gallina"]
mi_funcion(platos)      # la respuesta es El apellido de los menores son Rentería, Carla
```

## 2.2 Programación de funciones y/o subprogramas

- Retornar valores por la función

Se va a retornar una resultado con la sentencia **return**:

```
def mi_funcion(x):  
    return x * 5  
  
print(mi_funcion(3))                      # La respuesta es 15
```



## 2.2 Programación de funciones y/o subprogramas



### Variables locales y globales

Si se asigna o define una variable en el programa, se asumirá que es global.

- x= “Sorprendente” o ‘Sorprendente’

```
def myfunc():
    print("Python es " + x)
myfunc()
```

### Globales

Se puede crear una variable fuera y otra del mismo nombre dentro de la función. La que fue creada dentro de la función será local y tomará dicho valor en la función. La variable global mantendrá el valor inicial. Terminada la ejecución de la función, la variable local desaparecerá, y quedará la global con su valor original.

- x= ‘Sorprendente’

```
def myfunc():
    x = “genial”
    print("Python es " + x)
```

```
myfunc()
print("Python es " + x)
```

```
➤ def myfunc():
    Global x
    x=“genial” # Se declara con la palabra Global
    print("Python es " + x) # Global es palabra reservada
```

```
myfunc()
print("Python es " + x)
```

## 2.3 Gestión de errores



### **TEMA 5**

---

Gestión de errores

## 2.3 Gestión de errores



### ➤ Errores Sintácticos

El tipo de error más obvio es el **sintáctico**, que se produce cuando se escribe código de una forma no admitida por las **reglas del lenguaje**. Los errores de sintaxis son **detectados** casi siempre por el **compilador o intérprete**, que muestra un mensaje de error que informa del problema.

### ➤ Errores Semánticos

Los errores semánticos son más sutiles. Un error semántico se produce cuando la sintaxis del código es correcta, pero la semántica o significado no es el que se pretendía. La construcción obedece las reglas del lenguaje, y por ello el compilador o intérprete no detectan los errores semánticos. Los compiladores e intérpretes sólo se ocupan de la estructura del código que se escribe, y no de su significado. Un error semántico puede hacer que el programa termine de forma anormal, con o sin un mensaje de error.

Hablando en términos coloquiales, puede hacer que el equipo se quede “colgado”.

Sin embargo, no todos los errores semánticos se manifiestan de una forma tan obvia. Un programa puede continuar en ejecución después de haberse producido errores semánticos, pero su estado interno puede ser distinto del esperado. Quizá las variables no contengan los datos correctos, o bien es posible que el programa siga un camino distinto del pretendido. Eventualmente, la consecuencia será un resultado incorrecto. Estos **errores** se denominan **lógicos**, ya que aunque el programa no se bloquea, la lógica que representan contiene un error.

```
n=0; m=0  
print(n/m) # división entre un cero
```

```
In [234]: n=0; m=0  
          print(n/m)  
  
-----  
ZeroDivisionError  
<ipython-input-234-ec32a950f4f4> in <module>  
      1 n=0; m=0  
----> 2 print(n/m)  
  
ZeroDivisionError: division by zero
```

## 2.3 Gestión de errores



### ➤ Estados de excepción

El objetivo es manejar estos errores para que no se detenga el programa. Se manejarán las siguientes instrucciones **try: , except: , else: y finally:** veamos el siguiente ejemplo

```
while(True):
    try:
        n = float(input("Ingrese un numero"))
                    # Acá le ingresamos 'aaaa', se produce un error en la conversión
        m = 4
        print("{} / {} = {}".format(n, m, n/m))
    except:
        print("Ha ocurrido un error interno, introduce bien el número")
                    # al presentar el error, automáticamente se ejecuta
                    # esta sentencia
    else:
        print("Todo ha funcionando correctamente")
                    break
                    # esta sentencia
    finally:
        print("Fin de la iteración")
```

# INSERTAR NOMBRE DE ACTIVIDAD



## PASOS DE ACTIVIDAD individual /grupal:

- Ingresar al colab personal con su cuenta gmail
  
- Copia el enunciado del problema señalado por el profesor y péguelo en el colab
  
- Desarrolle el ejercicio junto con un compañero sea en el laboratorio o en la sala grupal creada por el docente.

# Resolución de ejercicios

## ACTIVIDAD

Debe resolver los ejercicios señalados por el profesor que se encuentran en el documento:

[02 - 01.2 Manejo de datos y optimización - Guía laboratorio y Casos propuestos.pdf](#)

Este documento se encuentra en el aula virtual en la unidad 2



## CONCLUSIONES

01

02

03

04

# BIBLIOGRAFÍA

- Direcciones electrónicas (hipervínculos en las diapositivas)
- LUTZ, MARK (2013) Learning Python. 5th Edition. California: O'Reilly.
- RAMALHO, LUCIANO (2015) Fluent Python: Clear, Concise, and Effective Programming (inglés) 1st Edición. California: O'Reilly.
-

# **Continúa con las actividades propuestas**

---



Material producido por la  
Universidad Peruana de  
Ciencias Aplicadas

Autor:

Colocar nombre del docente

COPYRIGHT © UPC  
2020 – Todos los  
derechos reservados



---

## **UNIDAD Nº: 4 PROGRAMACIÓN ORIENTADA A OBJETOS - Librerías**

## LOGRO DE APRENDIZAJE

Al finalizar la unidad, el estudiante aplica los conocimientos de las unidades anteriores y de las librerías enseñadas en clases en un contexto real o simulado. Analiza y desarrolla todos los elementos del componente a desarrollar empleando el lenguaje de programación Python, demostrando una actitud analítica, organizada y participativa en su equipo de trabajo.





## TEMA 6

---

Librerías

### 3.1.1 Librería Matplotlib

## Librería MatPlotLib en Python



La Matplotlib es una librería para gráficos en 2D de alta calidad.

➤ Instalación de Matplotlib

```
python -m pip install --U pip
```

```
python -m pip install --U matplotlib
```

➤ Instalación de Matplotlib en ANACONDA

```
conda install matplotlib
```

➤ Manual de la librería

<https://matplotlib.org/stable/contents.html>

➤ Donde **matplotlib** es la biblioteca, **pyplot** es un paquete que incluye todas funciones de MATLAB para utilizar las funciones de MATLAB en Python. Finalmente, podemos usar **plt** para llamar a las funciones dentro del archivo Python.

```
import matplotlib.pyplot as plt
```

### 3.1.1 Librería Matplotlib

## Módulo o Librería MatPlotLib en Python



### Dibujando gráficos simples

Supongamos que queremos dibujar una línea en el siguiente conjunto de puntos:  $x=(4, 8, 13, 17, 20)$  e  $y=(54, 67, 98, 78, 45)$

```
import matplotlib.pyplot as plt  
plt.plot([4, 8, 13, 17, 20] , [54, 67, 98, 78, 45])  
plt.show() # Debe dibujarle el grafico
```

Se puede personalizar la grafica: p.e. que la línea sea punteada y verde con marcadores de diamante, para lo cual se le agrega el parámetro 'g--d'

```
import matplotlib.pyplot as plt  
plt.plot([4, 8, 13, 17, 20] , [54, 67, 98, 78, 45] , 'g--d')  
plt.title("Ejemplo de Gráfica")  
plt.xlabel("Eje X")  
plt.ylabel("Eje Y")  
plt.show() # Debe dibujarle el grafico con las características personalizadas
```

### 3.1.1 Librería Matplotlib

## Módulo o Librería Matplotlib en Python



### Dibujando gráficos simples

Se puede realizar un Histograma de frecuencia que datos de un vector:

```
x=[2,4,6,5,42,543,5,3,73,64,42,97,63,76,63,8,73,97,23,45,56,89,45,3,23,2,5,78,23,56,67,78,8,3,78,34,67,23,324,234,43,544,54,33,223,443,444,234,76,432,233,23,232,243,222,221,254,222,276,300,353,354,387,364,309]
```

```
import matplotlib.pyplot as plt  
x=[2,4,6,5,42,543,5,3,73,64,42,97,63,76,63,8,73,97,23,45,56,89,45,3,23,2,5,78,23,56,67,78,8,3,78,34,67,23,324,234,43,544,54,33,223,443,444,234,76,432,233,23,232,243,222,221,254,222,276,300,353,354,387,364,309]  
num_bis = 6  
n, bins, patches = plt.hist(x, num_bis, facecolor ='green')  
plt.title("Ejemplo de Histograma")  
plt.xlabel("Eje X")  
plt.ylabel("Eje Y")  
plt.legend(prop ={'size': 10})  
plt.show() # Debe dibujarle el Histograma
```

### 3.1.2 Librería Datetime

# Módulo o Librería Datetime en Python



Una fecha en Python no es un tipo de datos propio, pero podemos importar un módulo llamado **datetime** para trabajar con fechas como objetos de fecha.

## ➤ Carga y uso de la librería Datetime

Importar el módulo Datetime y mostrar la fecha actual:

```
import datetime
x = datetime.datetime.now()
print(x)
print(x.year)
print(x.strftime("%A"))
```

# El resultado es 2021-02-20 22:15:49.974845  
# Se pueden usar métodos. El resultado es 2021  
# Se pueden usar métodos. El resultado es Tuesday

## ➤ Creación de objetos fecha

Para crear una fecha, podemos usar la clase `datetime()` (constructor) del módulo `datetime`. The `datetime()` class requires three parameters to create a date: year, month, day

### 3.1.2 Librería Datetime

## Módulo o Librería Datetime en Python



### ➤ Método strftime()

El objeto datetime tiene un método para formatear objetos de fecha en cadenas legibles. El método se llama **strftime ()** y toma un parámetro, que es el formato y sirve para especificar el formato de la cadena devuelta:

```
import datetime  
x = datetime.datetime.now()  
print(x.strftime("%A"))  
print(x.strftime("%B"))
```

# Se pueden usar métodos. El resultado es Thursday  
# El resultado es August

### 3.1.2 Librería Datetime

## Módulo o Librería Datetime en Python



### ➤ Método strftime() - operadores

Directive	Description	Example	Try it
%a	Weekday, short version	Wed	<a href="#">Try it »</a>
%A	Weekday, full version	Wednesday	<a href="#">Try it »</a>
%w	Weekday as a number 0-6, 0 is Sunday	3	<a href="#">Try it »</a>
%d	Day of month 01-31	31	<a href="#">Try it »</a>
%b	Month name, short version	Dec	<a href="#">Try it »</a>
%B	Month name, full version	December	<a href="#">Try it »</a>
%m	Month as a number 01-12	12	<a href="#">Try it »</a>
%y	Year, short version, without century	18	<a href="#">Try it »</a>
%Y	Year, full version	2018	<a href="#">Try it »</a>
%H	Hour 00-23	17	<a href="#">Try it »</a>
%I	Hour 00-12	05	<a href="#">Try it »</a>
%p	AM/PM	PM	<a href="#">Try it »</a>
%M	Minute 00-59	41	<a href="#">Try it »</a>
%S	Second 00-59	08	<a href="#">Try it »</a>
%f	Microsecond 000000-999999	548513	<a href="#">Try it »</a>
%z	UTC offset	+0100	
%Z	Timezone	CST	
%j	Day number of year 001-366	365	<a href="#">Try it »</a>
%U	Week number of year, Sunday as the first day of week, 00-53	52	<a href="#">Try it »</a>
%W	Week number of year, Monday as the first day of week, 00-53	52	<a href="#">Try it »</a>
%c	Local version of date and time	Mon Dec 31 17:41:00 2018	<a href="#">Try it »</a>
%x	Local version of date	12/31/18	<a href="#">Try it »</a>
%X	Local version of time	17:41:00	<a href="#">Try it »</a>
%%	A % character	%	<a href="#">Try it »</a>

### 3.1.4Librería Math

## Módulo o Librería Math en Python



Python tiene un conjunto de funciones matemáticas integradas, incluido un extenso módulo matemático, que le permite realizar tareas matemáticas en números.

#### ➤ Funciones matemáticas incorporadas

Las funciones min() y max() se pueden usar para encontrar el valor más bajo o más alto en una colección:

```
import math
x = min(5, 10, 25)
y = max(5, 10, 25)
print("El valor mínimo es ", x)           # El resultado es 5
print("El valor maximo es ",y)            # El resultado es 25
x = math.sqrt(64)
print("La raiz de 64 es ", x)             # El resultado es 8.0
```

#### ➤ Métodos de redondeo

El método math.ceil() redondea un número hacia arriba a su número entero más cercano, y el método math.floor() redondea un número hacia abajo a su número entero más cercano y devuelve el resultado:

```
import math
x = math.ceil(1.4)
y = math.floor(1.4)
print(x)                                # El resultado es 2
print(y)                                # El resultado es 1
```

### 3.1.4Librería Math

## Módulo o Librería Math en Python



Otros métodos de esta librería <https://docs.python.org/3/library/numeric.html>

Method	Description
<a href="#">math.acos(x)</a>	Returns the arc cosine value of x
<a href="#">math.acosh(x)</a>	Returns the hyperbolic arc cosine of x
<a href="#">math.asin(x)</a>	Returns the arc sine of x
<a href="#">math.asinh(x)</a>	Returns the hyperbolic arc sine of x
<a href="#">math.atan(x)</a>	Returns the arc tangent value of x
<a href="#">math.atan2(y, x)</a>	Returns the arc tangent of y/x in radians
<a href="#">math.atanh(x)</a>	Returns the hyperbolic arctangent value of x
<a href="#">math.ceil(x)</a>	Rounds a number upwards to the nearest integer, and returns the result
<a href="#">math.comb(n, k)</a>	Returns the number of ways to choose k items from n items without repetition and order
<a href="#">math.copysign(x, y)</a>	Returns a float consisting of the value of the first parameter and the sign of the second parameter
<a href="#">math.cos(x)</a>	Returns the cosine of x
<a href="#">math.cosh(x)</a>	Returns the hyperbolic cosine of x
<a href="#">math.degrees(x)</a>	Converts an angle from radians to degrees
<a href="#">math.dist(p, q)</a>	Calculates the euclidean distance between two specified points (p and q), where p and q are the coordinates of that point
<a href="#">math.erf(x)</a>	Returns the error function of x
<a href="#">math.erfc(x)</a>	Returns the complementary error function of x
<a href="#">math.exp(x)</a>	Returns the value of $E^x$ , where E is Euler's number (approximately 2.718281...), and x is the number passed to it
<a href="#">math.expm1(x)</a>	Returns the value of $E^x - 1$ , where E is Euler's number (approximately 2.718281...), and x is the number passed to it
<a href="#">math.fabs(x)</a>	Returns the absolute value of a number
<a href="#">math.factorial()</a>	Returns the factorial of a number
<a href="#">math.floor(x)</a>	Rounds a number downwards to the nearest integer, and returns the result
<a href="#">math.fmod(x, y)</a>	Returns the remainder of specified numbers when a number is divided by another number
<a href="#">math.frexp()</a>	Returns the mantissa and the exponent, of a specified value
<a href="#">math.fsum(iterable)</a>	Returns the sum of all items in an iterable (tuples, arrays, lists, etc.)
<a href="#">math.gamma(x)</a>	Returns the gamma value of x
<a href="#">math.gcd()</a>	Returns the highest value that can divide two integers
<a href="#">math.hypot()</a>	Find the Euclidean distance from the origin for n inputs
<a href="#">math.isclose()</a>	Checks whether two values are close, or not
<a href="#">math.isfinite(x)</a>	Checks whether x is a finite number
<a href="#">math.isinf(x)</a>	Check whether x is a positive or negative infinity

### 3.1.4Librería Math

## Módulo o Librería Math en Python



Method	Description
<a href="#">math.isnan(x)</a>	Checks whether x is NaN (not a number)
<a href="#">math.isqrt(n)</a>	Returns the nearest integer square root of n
<a href="#">math.ldexp(x, i)</a>	Returns the expression $x * 2^i$ where x is mantissa and i is an exponent
<a href="#">math.lgamma(x)</a>	Returns the log gamma value of x
<a href="#">math.log(x, base)</a>	Returns the natural logarithm of a number, or the logarithm of number to base
<a href="#">math.log10(x)</a>	Returns the base-10 logarithm of x
<a href="#">math.log1p(x)</a>	Returns the natural logarithm of 1+x
<a href="#">math.log2(x)</a>	Returns the base-2 logarithm of x
<a href="#">math.perm(n, k)</a>	Returns the number of ways to choose k items from n items with order and without repetition
<a href="#">math.pow(x, y)</a>	Returns the value of x to the power of y
<a href="#">math.prod(iterable, *, start=1)</a>	Returns the product of an iterable (lists, array, tuples, etc.)
<a href="#">math.radians(x)</a>	Converts a degree value (x) to radians
<a href="#">math.remainder(x, y)</a>	Returns the closest value that can make numerator completely divisible by the denominator
<a href="#">math.sin(x)</a>	Returns the sine of x
<a href="#">math.sinh(x)</a>	Returns the hyperbolic sine of x
<a href="#">math.sqrt(x)</a>	Returns the square root of x
<a href="#">math.tan(x)</a>	Returns the tangent of x
<a href="#">math.tanh(x)</a>	Returns the hyperbolic tangent of x
<a href="#">math.trunc(x)</a>	Returns the truncated integer parts of x

Constant	Description
<a href="#">math.e</a>	Returns Euler's number (2.7182...)
<a href="#">math.inf</a>	Returns a floating-point positive infinity
<a href="#">math.nan</a>	Returns a floating-point NaN (Not a Number) value
<a href="#">math.pi</a>	Returns PI (3.1415...)
<a href="#">math.tau</a>	Returns tau (6.2831...)

Fuente: <https://www.w3schools.com/python>

Constantes esta librería

<https://docs.python.org/3/library/numeric.html>

# INSERTAR NOMBRE DE ACTIVIDAD



## PASOS DE ACTIVIDAD individual /grupal:

- Ingresar al colab personal con su cuenta gmail
  
- Copia el enunciado del problema señalado por el profesor y péguelo en el colab
  
- Desarrolle el ejercicio junto con un compañero, ya sea en el laboratorio o en la sala grupal creada por el docente.

# Resolución de ejercicios

## ACTIVIDAD

Debe resolver los ejercicios señalados por el profesor que se encuentran en el documento:

[03 - 01 Programación Orientada a Objetos - Guía laboratorio y Casos propuestos.pdf](#)

Este documento se encuentra en el aula virtual en la unidad 3



# CONCLUSIONES

01

Las librerías son software desarrollado por terceros con funcionalidades avanzadas, específicas y especializadas que potencian a Python estándar. Son gratuitas

02

Las diferentes librerías se compatibles con otras y se potencian entre si.

03

Las librerías enseñadas en clase son las mas populares y son la base para estudios superiores de ciencia de los datos, big data y otros tópicos de inteligencia artificial.

04

Python estándar como las librerías han sido desarrolladas con la arquitectura orientada a objetos.

# BIBLIOGRAFÍA

- Direcciones electrónicas (hipervínculos en las diapositivas)
- LUTZ, MARK (2013) Learning Python. 5th Edition. California: O'Reilly.
- RAMALHO, LUCIANO (2015) Fluent Python: Clear, Concise, and Effective Programming (inglés) 1st Edición. California: O'Reilly.
-

# **Continúa con las actividades propuestas**

---



Material producido por la  
Universidad Peruana de  
Ciencias Aplicadas

Autor:

Norman Reyes Morales

COPYRIGHT © UPC  
2023 – Todos los  
derechos reservados



---

## **UNIDAD Nº: 4 PROGRAMACIÓN ORIENTADA A OBJETOS - Librerías**

## LOGRO DE APRENDIZAJE

Al finalizar la unidad, el estudiante aplica los conocimientos de las unidades anteriores y de las librerías enseñadas en clases en un contexto real o simulado. Analiza y desarrolla todos los elementos del componente a desarrollar empleando el lenguaje de programación Python, demostrando una actitud analítica, organizada y participativa en su equipo de trabajo.





## TEMA 6

---

Librerías

## 3.1.4 Librería TKINTER

### TKINTER



**Tkinter** es la biblioteca estándar de Python para construir interfaces gráficas (GUI).

La estructura que se crea cuando se usa **tkinter** es la que se denomina **event driven** (dirigida por eventos). Esta consiste en crear la GUI y esperar a que ocurra un evento. Durante la espera, la librería Tk se encuentra en un **event loop (bucle)** vigilante a cualquier evento como una pulsación de ratón o de una tecla, una vez se produce se llama inmediatamente a un código para su ejecución.

Cada programa de tkinter consta de estas cosas:

- Ventanas, botones, barras de desplazamiento, áreas de texto y otros widgets
- Un widget es cualquier cosa que pueda ver en la pantalla de la computadora. (Generalmente, el término widget significa cualquier objeto útil; es la abreviatura de “window gadget”).
- Módulos, funciones y clases que administran los datos que se muestran en la GUI
- Un administrador de eventos que “escucha” eventos como clics del mouse y pulsaciones de teclas y reacciona a estos eventos llamando a las funciones del controlador de eventos.

#### ➤ Instalación de Tkinter en ANACONDA

```
# The actual install command  
conda install -c anaconda tk
```

Manual de la librería

<https://guia-tkinter.readthedocs.io/es/develop/>  
<https://docs.python.org/es/3/library/tk.html>

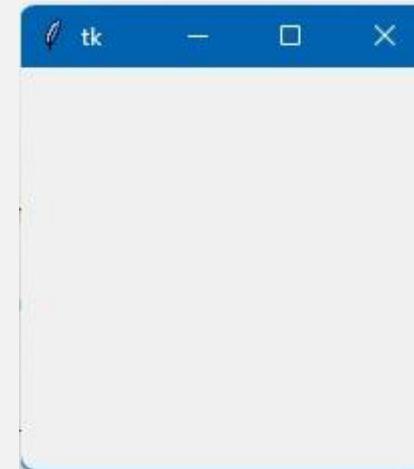
### 3.1.4 Librería TKINTER

## TKINTER



Veamos el siguiente código

```
import tkinter as tk          # importa la librería tkinter  
raiz = tk.Tk()                # crea ventana  
raiz.mainloop()              # ejecuta el event loop
```



- La primera línea importa todo el material de la GUI del módulo tkinter.
- La segunda línea crea una ventana en la pantalla, que llamamos raíz.
- La tercera línea coloca el programa en lo que es esencialmente un bucle eterno llamado event loop o bucle de eventos
- Al ejecutarlo se obtiene la pantalla de la imagen adjunta
- Si hubiera código debajo de raíz.mainloop, no se ejecutará hasta que se cierre la ventana
- Por lo general, raíz.mainloop será la última línea del código

### 3.1.4 Librería TKINTER

## TKINTER



- Existen dos métodos distintos para modificar la estética de la interfaz para diferenciarlos suele llamarlos **controles clásicos** y **controles temáticos**.
- Los **controles clásicos** son los controles originales de **Tk**, a los cuales desde Python accedemos vía el módulo **tkinter** que generalmente se abrevia **tk**.
- Los **controles temáticos** se introdujeron en la versión 8.5 de Tk y están contenidos en el submódulo de Python **ttk**.
- La diferencia entre los dos tipos de controles es principalmente estética
- Los **controles temáticos** tienen una apariencia más moderna e incluyen un sistema de personalización de esa apariencia mejor que el sistema de **controles clásicos**.
- Por ello muchas veces un programa inicia así

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()

ventana.mainloop()
```

## 3.1.4 Librería TKINTER

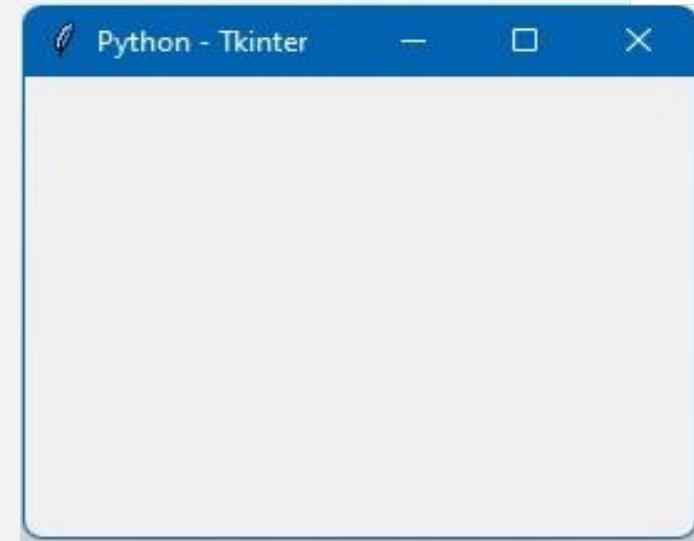
### TKINTER - Poniéndole título a la ventana



```
# Importar librería  
import tkinter as tk
```

```
# inicializar ventana  
ventana = tk.Tk()  
ventana.title("Python - Tkinter")
```

```
# Activar ventana  
ventana.mainloop()
```



### TKINTER - Tamaño de la ventana: Geometry

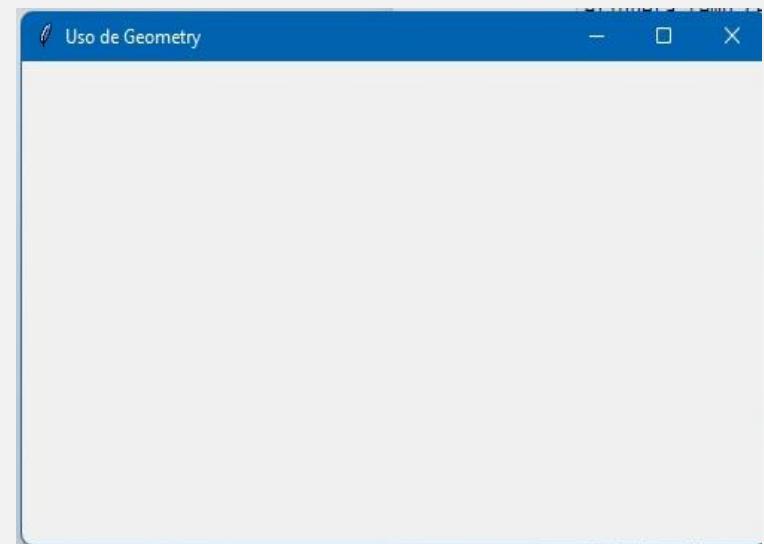
```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
#Asignar un tamaño a la ventana  
#geometry('ancho X alto + posx + posy')  
ventana.geometry('500x300+100+100')
```

```
ventana.title('Uso de Geometry')
```

```
ventana.mainloop()
```



### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con pack



```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
ventana.geometry('200x200+100+100')  
ventana.title('Uso de Etiquetas')
```

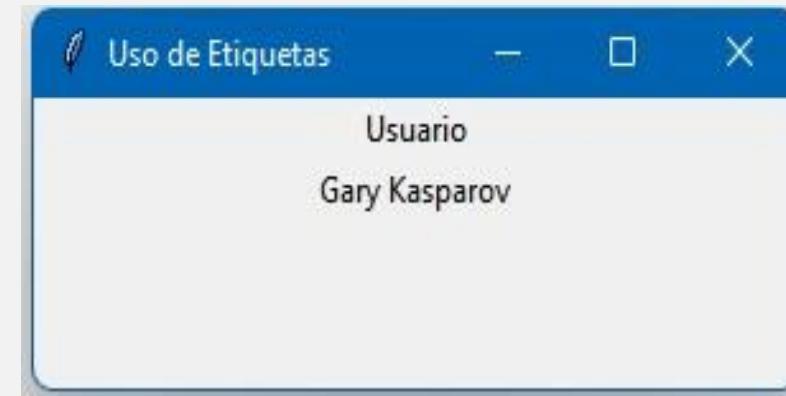
*#Creamos Las etiquetas*

```
labelUsuario = tk.Label(text='Usuario')  
labelDatos = tk.Label(text='Gary Kasparov')
```

*#Agregamos las etiquetas a la ventana*

```
labelUsuario.pack()  
labelDatos.pack()
```

```
ventana.mainloop()
```



### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con pack



```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
ventana.geometry('300x100+100+100')
```

```
ventana.title('Uso de Etiquetas')
```

*#Creamos Las etiquetas*

```
labelUsuario = tk.Label(text='Usuario')
```

```
labelDatos = tk.Label(text='Gary Kasparov')
```

```
labelProfesion = tk.Label(text='Ajedrecista')
```

```
labelTitulo = tk.Label(text='Campeón Mundial de Ajedrez')
```

*#Agregamos la etiqueta a la ventana*

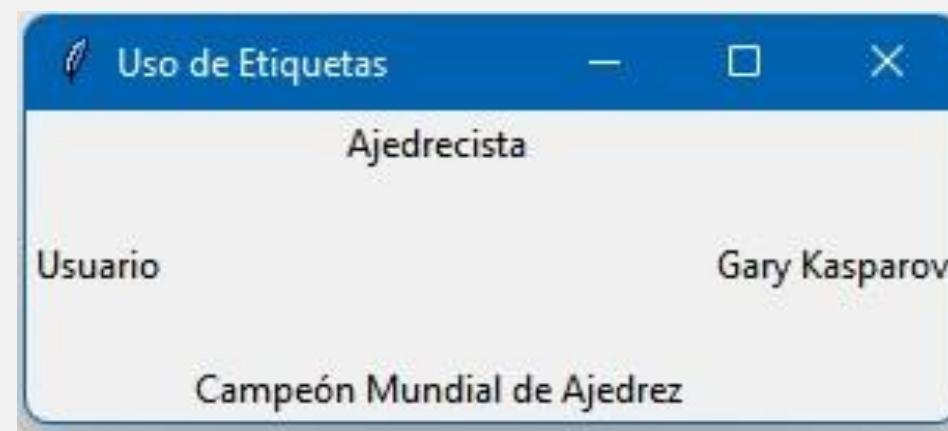
```
labelUsuario.pack(side='left')
```

```
labelDatos.pack(side='right')
```

```
labelProfesion.pack(side='top')
```

```
labelTitulo.pack(side='bottom')
```

```
ventana.mainloop()
```



### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con place



Otro método utilizado es el llamado **place** en vez de **pack**. Hay que tener cuidado ya que las medidas que se usan están en pixeles y no es caracteres por lo que muchas veces hay que probar hasta llegar a la ubicación deseada.

```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
ventana.geometry('300x100+100+100')  
ventana.title("Uso de Etiquetas")
```

*#Creamos Las etiquetas y le damos su ubicación con place*

```
labelUsuario = tk.Label(text='Usuario').place(x=10, y=10)  
labelDatos = tk.Label(text='Gary Kasparov').place(x=120, y=10)  
labelProfesion = tk.Label(text='Ajedrecista').place(x=10, y=30)  
labelTitulo = tk.Label(text='Campeón Mundial de Ajedrez').place(x=120, y=30)
```

```
ventana.mainloop()
```



## 3.1.4 Librería TKINTER

### TKINTER - Etiquetas ubicadas con grid



- Tkinter **grid** es otro y también el más importante método de geometría de diseño
- Hay que tener en cuenta que utilizaremos la ventana como una grilla.

	columna 0	columna 1	columna 2	columna 3
fila 0	(0,0)	(0,1)	(0,2)	(0,3)
fila 1	(1,0)	(1,1)	(1,2)	(1,3)
fila 2	(2,0)	(2,1)	(2,2)	(2,3)
fila 3	(3,0)	(3,1)	(3,2)	(3,3)

### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con grid



- También se podrá usar la función **sticky** para la ubicación del texto
- **sticky** determina cómo se pega el widget a la celda cuando el widget es más pequeño que la celda.

sticky	Significado
W	pegarse a la izquierda
E	mantenerse a la derecha
N	pegarse a la parte superior
S	adherirse a la parte de abajo

### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con grid



```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
ventana.geometry('300x100+100+100')
```

```
ventana.title('Uso de Etiquetas')
```

#Creamos Las etiquetas y le damos su ubicación con grid

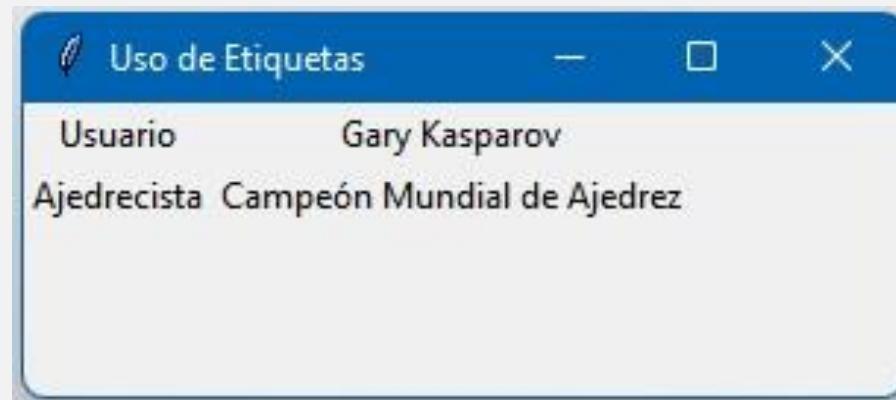
```
labelUsuario = tk.Label(text='Usuario').grid(row=0, column=0)
```

```
labelDatos = tk.Label(text='Gary Kasparov').grid(row=0, column=1)
```

```
labelProfesion = tk.Label(text='Ajedrecista').grid(row=1, column=0)
```

```
labelTitulo = tk.Label(text='Campeón Mundial de Ajedrez').grid(row=1, column=1)
```

```
ventana.mainloop()
```



### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con grid



```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
ventana.geometry('300x100+100+100')
```

```
ventana.title('Uso de Etiquetas')
```

*#Creamos Las etiquetas y le damos su ubicación con grid*

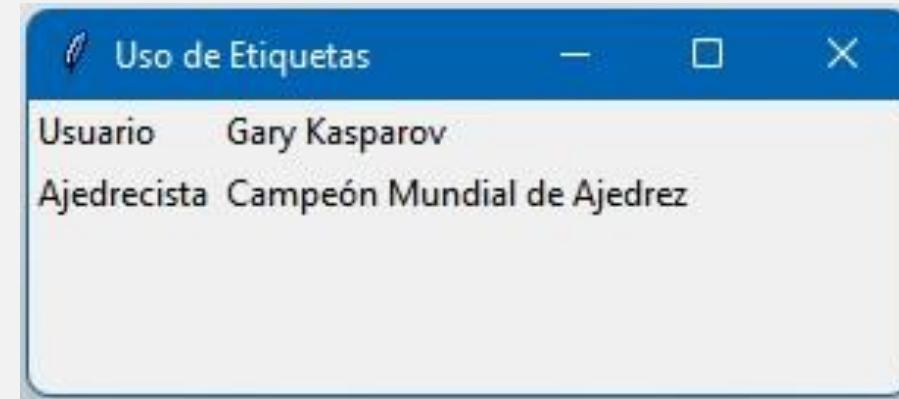
```
labelUsuario = tk.Label(text='Usuario').grid(row=0, column=0, sticky=tk.W)
```

```
labelDatos = tk.Label(text='Gary Kasparov').grid(row=0, column=1, sticky=tk.W)
```

```
labelProfesion = tk.Label(text='Ajedrecista').grid(row=1, column=0, sticky=tk.E)
```

```
labelTitulo = tk.Label(text='Campeón Mundial de Ajedrez').grid(row=1, column=1, sticky=tk.E)
```

```
ventana.mainloop()
```



### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con grid



```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
ventana.geometry('300x100+100+100')
```

```
ventana.title('Uso de Etiquetas')
```

*#Creamos Las etiquetas y le damos su ubicación con grid*

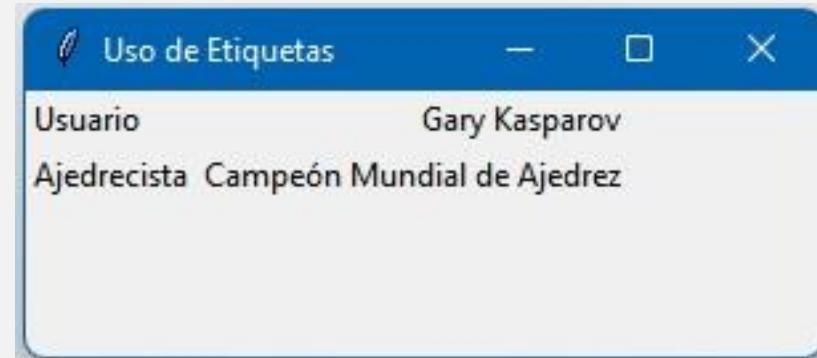
```
labelUsuario = tk.Label(text='Usuario').grid(row=0, column=0, sticky=tk.W)
```

```
labelDatos = tk.Label(text='Gary Kasparov').grid(row=0, column=1,
                                                 columnspan=2, rowspan=1,
                                                 sticky=tk.E)
```

```
labelProfesion = tk.Label(text='Ajedrecista').grid(row=1, column=0,
                                                 sticky=tk.W)
```

```
labelTitulo = tk.Label(text='Campeón Mundial de Ajedrez').grid(row=1,
                                                               column=1,
                                                               columnspan=2,
                                                               rowspan=1,
                                                               sticky=tk.E)
```

```
ventana.mainloop()
```



columnspan y  
rowspan permiten  
que el campo  
ocupe más de una  
cuadrícula de la  
grilla

### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con grid

Otra forma de escribir el código anterior

```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
ventana.geometry('300x100+100+100')
```

```
ventana.title('Uso de Etiquetas')
```

#Creamos Las etiquetas y le damos su ubicación con grid

```
labelUsuario = tk.Label(text='Usuario')
```

```
labelUsuario.grid(row=0, column=0, sticky=tk.W)
```

```
labelDatos = tk.Label(text='Gary Kasparov')
```

```
labelDatos.grid(row=0, column=1, columnspan=2, rowspan=1, sticky=tk.E)
```

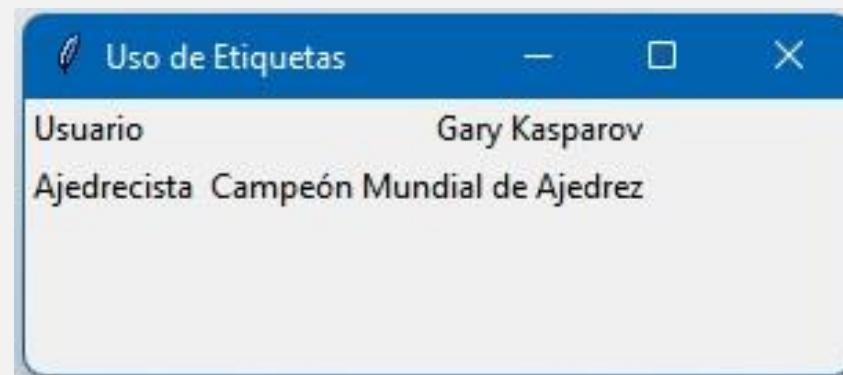
```
labelProfesion = tk.Label(text='Ajedrecista')
```

```
labelProfesion.grid(row=1, column=0, sticky=tk.W)
```

```
labelTitulo = tk.Label(text='Campeón Mundial de Ajedrez')
```

```
labelTitulo.grid(row=1, column=1, columnspan=2, rowspan=1, sticky=tk.E)
```

```
ventana.mainloop()
```



## 3.1.4 Librería TKINTER



### TKINTER - Agregar Botones

Considere el siguiente código

```
import tkinter as tk
from tkinter import ttk

def funcion_click():
    accion.configure(text='**¡Haz hecho click!**')
    etiqueta.configure(foreground='red')

ventana = tk.Tk()
ventana.geometry('300x100+100+100')
ventana.title('Python -Tkinter')

etiqueta = ttk.Label(ventana, text='Hola a todos!!!')
etiqueta.grid(row=0, column=0)

# Agrega un boton
accion = ttk.Button(ventana, text='Haz click aquí',
                    command=funcion_click)
accion.grid(column=1, row=0)

ventana.mainloop()
```

Para que el botón haga algo cuando se hace clic en él, se usa el argumento **command**. Se pone en el **nombre de una función**. Cuando se hace clic en el botón, se llama a la función, en el ejemplo, es la función llamada: **funcion\_click**. Después de haber creado una **label**, es posible que desee cambiar algo de dicha **label**. Para hacer eso, se usa el método **configure**. En el ejemplo cambia el texto del botón **acción** y el color de letra de **etiqueta**.

### 3.1.4 Librería TKINTER

## TKINTER - Agregar Caja de Texto (Entry)



```
import tkinter as tk
from tkinter import ttk

def funcion_click():
    accion.configure(text='Hola ' + nombre.get())
    # get() obtiene el valor ingresado

# Inicializar ventana
ventana = tk.Tk()
ventana.geometry('300x100+100+100')
ventana.title("Python - Tkinter")

# Agregar etiqueta por medio de un objeto
etiqueta = ttk.Label(ventana, text="Escribe tu nombre")
etiqueta.grid(column=0, row=0)

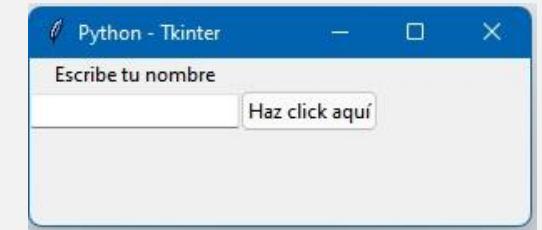
# Agregar una caja de texto
nombre = tk.StringVar() # se crea la variable para Entry con StringVar
preguntar_nombre = ttk.Entry(ventana, width=20, textvariable=nombre)
preguntar_nombre.grid(column=0, row=1)

# Agregar un botón
accion = ttk.Button(ventana, text="Haz click aquí", command=funcion_click)
accion.grid(row=1, column=1)

ventana.mainloop()
```

- **StringVar()** permite crear la variable para **Entry** la cual se usará en **textvariable**
- **Width** establece el ancho de la caja de entrada

Antes del click



Después del click



## 3.1.4 Librería TKINTER

### TKINTER - Agregar ComboBox



```
import tkinter as tk
from tkinter import ttk

def funcion_click():
    respuesta = ttk.Label(ventana, text='Seleccionado el ' + numero.get())
    respuesta.grid(column=1, row=3)

ventana = tk.Tk()
ventana.geometry('300x100+100+100')
ventana.title("Python - Tkinter")

etiqueta = ttk.Label(ventana, text="Selecciona un número")
etiqueta.grid(column=0, row=0)

# Agregar lista desplegable
numero = tk.StringVar()
seleccionar_numero = ttk.Combobox(ventana, width=12, textvariable=numero)

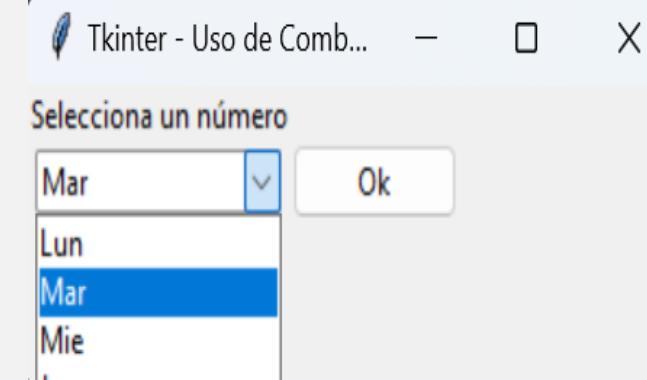
# Llenar lista desplegable
seleccionar_numero['values'] = ("Lun","Mar","Mie","Jue","Vie","Sab","Dom")

# Posicionar lista desplegable
seleccionar_numero.grid(column=0, row=1)

# Elemento de la lista seleccionado por default
seleccionar_numero.current(0)
accion = ttk.Button(ventana, text="Ok", command=funcion_click)
accion.grid(row=1, column=1)

ventana.mainloop()
```

- `ttk.Combobox` crea la lista desplegable
- `seleccionar_numero['values']`, permite asignar los valores de la lista desplegable
- `seleccionar_numero.current(0)`, establece la posición del valor por defecto



### 3.1.4 Librería TKINTER

#### TKINTER - FRAME

El frame es un contenedor de otros widgets

- Ventana raíz de la aplicación es prácticamente un frame (marco)
- Cada marco tiene su propio diseño de cuadrícula, por lo que la cuadrícula (grid) de widgets dentro de cada marco funciona de forma independiente.
- Los widgets de marco son una herramienta valiosa para hacer que su aplicación sea modular. Puede agrupar un conjunto de widgets relacionados en un widget compuesto colocándolos en un marco. Mejor aún, puede declarar una nueva clase que herede de Frame, agregándole su propia interfaz. Esta es una buena manera de ocultar los detalles de las interacciones dentro de un grupo de widgets relacionados del mundo exterior.
- Para crear un nuevo widget de marco en una ventana raíz o marco llamado parent:

```
import tkinter as tk
```

```
marco = Frame(parent, option, ...)
```

Parametros	descripción
bg or background	Color de fondo del marco
bd o borderwidth	Ancho del borde del marco. El valor predeterminado es 0 (sin borde)
cursor	El cursor utilizado cuando el mouse está dentro del widget de marco
height	La dimensión vertical del nuevo marco.
highlightbackground	Color del resaltado de enfoque cuando el marco no tiene foco
highlightcolor	Color que se muestra en el resaltado de enfoque cuando el marco tiene el foco
highlightthickness	Grosor del punto culminante del foco.
padx	Normalmente, un marco se ajusta perfectamente a su contenido. Para agregar N píxeles de espacio horizontal dentro del marco, establezca padx=N.
pady	Se utiliza para agregar espacio vertical dentro de un marco. Ver padx arriba.
relief	El relieve predeterminado para un marco es tk.FLAT, lo que significa que el marco se mezclará con su entorno. Para colocar un borde alrededor de un marco, establezca su ancho de borde en un valor positivo y establezca su relieve en uno de los tipos de relieve estándar
takefocus	Normalmente, los controles de entrada no visitan los widgets de marco. Sin embargo, puede configurar takefocus=1 si desea que el marco reciba la entrada del teclado. Para manejar dicha entrada, deberá crear enlaces para eventos de teclado
width	La dimensión horizontal del nuevo marco.

### 3.1.4 Librería TKINTER

## TKINTER - Otras opciones de tkinter



Tkinter tiene muchos otros widgets, los cuales pueden ser consultadas en la documentación. Se muestra una lista de otros widgets disponibles en tkinter

Widget	Descripción
Spinbox	Permite seleccionar un valor de un conjunto de valores. Los valores pueden ser un rango de números.
Canvas	Un área utilizada para dibujar o mostrar imágenes.
Checkbutton	Un cuadro en el que se puede hacer clic que se puede seleccionar o deseleccionar
Frame	Un contenedor para widgets
Menu	Un menú desplegable
Message	Una pantalla de varias líneas para texto
Menobutton	Un elemento en un menú desplegable
Text	Un campo de texto de varias líneas en el que el usuario puede escribir
TopLevel	Una ventana adicional
Radiobutton	Permite elegir entre una de varias opciones mutuamente excluyentes
Listbox	Permite elegir entre un conjunto de opciones o muestra una lista de elementos.
Scale	Se utiliza para implementar un control deslizante gráfico, da la opción de elegir entre un rango de valores.
Notebook	Permite dividir una parte de la ventana en distintas solapas de modo que, dependiendo de la que se encuentre seleccionada, varía el contenido de la interfaz

### 3.1.4 Librería TKINTER

## TKINTER - Programa que convierte temperaturas de Celsius a Fahrenheit.



```
import tkinter as tk

def convierte():
    temp = int(entrada.get())
    temp = 9/5*temp+32
    etiquetaSalida.configure(text = 'En Fahrenheit: {:.1f}'.format(temp))
    entrada.delete(0,tk.END)

raiz = tk.Tk()
raiz.geometry('400x100+100+100')
raiz.title("Convertir de Celsius a Fahrenheit")

etiquetaMsg = tk.Label(text='Ingrese temperatura °C', font=('Calibri', 14))
etiquetaMsg.grid(row=0, column=0)

etiquetaSalida = tk.Label(font=('Calibri', 14))
etiquetaSalida.grid(row=1, column=0, columnspan=3)

entrada = tk.Entry(font=('Calibri', 14), width=4)
entrada.grid(row=0, column=1)

botonCalc = tk.Button(text='Ok', font=('Calibri', 14), command=convierte)
botonCalc.grid(row=0, column=2)

raiz.mainloop()
```

### 3.1.4 Librería TKINTER

## TKINTER - Programa Botón presionado.



```
import tkinter as tk

alfabeto = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def btPres(x):
    etiqueta.configure(text='Botón {} presionado'.format(alfabeto[x]))

raiz = tk.Tk()
raiz.geometry('480x100+100+100')
raiz.title("Identificar botón presionado")

etiqueta = tk.Label(font=('Calibri', 12))
etiqueta.grid(row=1, column=0, columnspan=26)
botones = [0] * 26 # crea una lista de 26 botones

for i in range(26):
    botones[i] = tk.Button(text=alfabeto[i], command=lambda x=i:
btPres(x))
    botones[i].grid(row=0, column=i)

raiz.mainloop()
```

Como hemos visto en los botones mediante la opción `command` se puede invocar una función. En un primer momento aparece una restricción que es que no se pueden pasar argumentos con la función, ya que en el caso de aportarlos la función se ejecuta cuando el botón es creado y no cuando capta una pulsación.

Una forma de conseguir pasar parámetros en usando “**funciones anónimas**” con la instrucción `lambda`

La palabra clave `lambda` indica que lo que sigue será una función anónima. Luego tenemos los **argumentos de la función anónima**, seguidos de **dos puntos** y luego el **código de la función**. El código de función no puede tener más de una línea, en otras palabras, para crear una función anónima se requiere:

- La palabra clave: `lambda`
- Las variables relacionadas: `x`
- las instrucciones a aplicar

Luego, el truco básicamente consiste en diferir la llamada creando una función intermedia que pasamos en la opción `command`

### 3.1.5 Librería NUMPY



NumPy es el paquete fundamental para la computación científica en Python. Es una **biblioteca de Python** que proporciona un objeto de matriz multidimensional, varios objetos derivados (como matrices y matrices enmascaradas) y una variedad de rutinas para operaciones rápidas en matrices, que incluyen manipulación matemática, lógica, de formas, clasificación, selección, E / S., transformadas discretas de Fourier, álgebra lineal básica, operaciones estadísticas básicas, simulación aleatoria y mucho más.

➤ Instalación de Numpy en ANACONDA

```
# Best practice, use an environment rather than install in the base env
```

```
conda create -n my-env
```

```
conda activate my-env
```

```
conda config --env --add channels conda-forge # If you want to install from conda-forge
```

```
# The actual install command
```

```
conda install numpy
```

➤ Manual de la librería

<https://numpy.org/doc/stable/user/quickstart.html>

➤ Manual de clase ndarray \_ métodos

<https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html?highlight=ndarray#numpy.ndarray>

### 3.1.5 Librería NUMPY



- NumPy tiene como objetivo proporcionar un objeto de matriz que sea hasta 50 veces más rápido que las listas tradicionales de Python. El **objeto de matriz** en NumPy se llama **ndarray**, proporciona muchas funciones de apoyo que hacen que trabajar con ndarray sea muy fácil.
- Importar Numpy

Donde **Numpy** es la biblioteca, **np** es el alias para llamar a las funciones dentro del archivo Python.

```
import numpy as np
```

```
print(np.__version__)
```

# Para imprimir la versión

#### Operaciones de arreglos con Numpy

- Creando un arreglo en Numpy

```
import numpy as np  
arr = np.array([1, 2, 3, 4, 5])  
print(arr)  
print(type(arr))
```

# Debe imprimir la lista

# Debe imprimir que es una clase ndarray de Numpy

- Creando un arreglo en Numpy desde una tupla

```
import numpy as np  
arr = np.array((1, 2, 3, 4, 5))  
print(arr)
```

# Debe dibujarle el grafico con las características personalizadas

## 3.1.5 Librería NUMPY



### Operaciones de arreglos con Numpy

#### ➤ Creando un arreglo 2D en Numpy

```
import numpy as np  
arr = np.array([[9, 8, 7], [4, 5, 6]])  
print(arr)
```

# Debe imprimir la lista

#### ➤ Creando un arreglo 3D en Numpy

```
import numpy as np  
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])  
print(arr)
```

# Debe imprimir la lista de 3 filas

#### ➤ Verificando el numero de dimensiones

```
import numpy as np  
a = np.array(42)  
b = np.array([1, 2, 3, 4, 5])  
c = np.array([[1, 2, 3], [4, 5, 6]])  
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])  
  
print(a.ndim)  
print(b.ndim)  
print(c.ndim)  
print(d.ndim)
```

# Debe imprimir las dimensiones de los arreglos

## 3.1.5 Librería NUMPY

### Operaciones de arreglos con Numpy



#### ➤ Creando un arreglo con 5 dimensiones en Numpy

```
import numpy as np  
arr = np.array([1, 2, 3, 4], ndmin=5)  
print(arr)  
print('numero de dimensiones :', arr.ndim)
```

# Debe imprimirle la lista

#### ➤ Uso de arreglos con 2D en Numpy

```
import numpy as np  
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])  
print('2do elemento de la 1ra dim: ', arr[0, 1])
```

# Debe imprimir el numero 2

#### ➤ Uso de arreglos con 2D en Numpy con índices negativos

```
import numpy as np  
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])  
print("Ultimo elemento de 2da dim: ", arr[1, -1])
```

# Debe imprimir el numero 10

#### ➤ Slicing y STEP en vectores

```
import numpy as np  
arr = np.array([1, 2, 3, 4, 5, 6, 7])  
print("Los elementos pares cada dos posiciones, hasta 4: ", arr[1:5:2])  
print("Todos los elementos impares cada dos posiciones: ", arr[::-2])
```

# Debe imprimir el numero [2, 4]

# Debe imprimir el numero [2, 4]

### 3.1.5 Librería NUMPY



# Operaciones de arreglos con Numpy

## ➤ Slicing y STEP en matrices

```
import numpy as np  
arr = np.array ([[1,2,3,4,5], [6,7,8,9,10]])  
print("Los elementos de fila 2 y columnas 1 a 3 ", arr[1, 1:4])      # Debe imprimir el numero [6, 7, 8]  
print("Rangos en filas y columnas: ", arr[0:2, 1:4])                  # Debe imprimir el numero [2, 4]
```

## ➤ Uso de arreglos con 3D en Numpy

```
import numpy as np  
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])  
print(arr[0, 1, 2]) # Debe imprimir el numero 6
```

## 3.1.5 Librería NUMPY



### Operaciones de arreglos con Numpy

➤ Se muestran los tipos de datos de **NumPy** y los caracteres usados que los representan.

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type ( void )

➤ Verificar tipos de datos en Numpy

```
import numpy as np  
arr = np.array([[1, 2, 3], [4, 5, 6], [[7, 8, 9], [10, 11, 12]]])  
print(arr.dtype) # Debe imprimir integer
```

## 3.1.5 Librería NUMPY



### Operaciones de arreglos con Numpy

#### ➤ Convertir tipos de datos en Numpy

```
import numpy as np  
arr = np.array([1.1, 2.3, 3.4])  
nuevarr = arr.astype("I")  
print(nuevarr)  
print(nuevarr.dtype)
```

# El método **astype()** convierte al dato del parámetro  
# Debe imprimir el vector con enteros [1 2 3]  
# Debe imprimir int32

#### ➤ Diferencia entre copiar y ver de arreglos en Numpy

```
import numpy as np  
arr = np.array([1, 2, 3, 4, 5, 6])  
nuevarr = arr.copy()  
arr[0] = 100  
print("Vector original Arr ",arr)  
print("Vector copiado nuevo ", nuevarr)
```

# El método **copy()** crea una copia del arreglo  
# Se modifica el arreglo arr  
  
# nuevarr no se afecta con los cambios de arr

```
arr = np.array([1, 2, 3, 4, 5, 6])  
nueviewvarr = arr.view()  
arr[0] = 100  
print("Vector original Arr ",arr)  
print("Vector Vista ", nueviewvarr)
```

# El método **copy()** crea una copia del arreglo  
# Se modifica el arreglo arr  
  
# nuevarr si se afecta con los cambios de arr

## 3.1.5 Librería NUMPY



### Operaciones de arreglos con Numpy

#### ➤ Uso de shape en Numpy

```
import numpy as np  
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])  
print(arr.shape)
```

# Debe imprimir (2, 4) que dice que es un arreglo de 2 dimensiones y cada dimensión tiene 4 elementos

#### ➤ Uso de reshape en Numpy

```
import numpy as np  
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])  
nuevarr = arr.reshape(4, 3)  
print(nuevarr)
```

# Crea un nuevo arreglo de 4 filas y 3 columnas

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])  
nuevarr = arr.reshape(2, 3, 2)  
print(nuevarr)
```

# Crea un nuevo arreglo de 2 filas, 3 columnas y 2 elementos

## 3.1.5 Librería NUMPY

### Operaciones de arreglos con Numpy



#### ➤ Recorrer arreglos en Numpy

```
import numpy as np  
arr = np.array([1, 2, 3, 4])  
for x in arr:  
    print(x) # Debe imprimir cada elemento en una fila
```

#### ➤ Recorridos en matrices en Numpy

```
import numpy as np  
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])  
for x in arr:  
    for y in x:  
        print(y) # Debe imprimir cada elemento en una fila
```

#### ➤ Recorridos en matrices de altas dimensiones en Numpy

```
import numpy as np  
arr = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])  
for x in np.nditer(arr):  
    print(x) # Debe imprimir cada elemento en una fila
```

## 3.1.5 Librería NUMPY



### Operaciones de arreglos con Numpy

#### ➤ Recorridos en matrices de altas dimensiones en Numpy

```
import numpy as np  
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])  
for x in np.nditer(arr[:, ::2]):           # Debe imprimir cada elemento saltando 1  
    print(x)
```

#### ➤ Recorridos en matrices de altas dimensiones e índices en Numpy

```
import numpy as np  
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])  
for idx, x in np.ndenumerate(arr):         # Debe imprimir el índice y el elemento  
    print(idx, x)
```

#### ➤ Unir matrices en Numpy

```
import numpy as np  
arr1 = np.array([1, 2, 3, 4])  
arr2 = np.array([5, 6, 7, 8])  
arr = np.concatenate((arr1, arr2))          # Debe imprimir el índice y el elemento  
print(arr)
```

### 3.1.5 Librería NUMPY



## Operaciones de arreglos con Numpy

## ➤ Busquedas en matrices en Numpy

```
import numpy as np  
arr = np.array([1, 2, 3, 4, 5, 4, 4])  
x  = np.where(arr == 4)  
print(x) # Debe imprimir (array([3, 5, 6]),) significa los índices 3, 5 y 6
```

```
x = np.where(arr%2 == 0)
print(x) # Debe imprimir elementos pares
```

## ➤ Ordenamiento de matrices en Numpy

```
import numpy as np  
arr = np.array([4, 2, 3, 1] , [8, 6, 7, 5])  
print(np.sort(arr)) # Debe imprimir los vectores ordenados
```

## 3.1.5 Librería NUMPY



### Aleatorios con Numpy

#### ➤ Generación de aleatorios en Numpy

```
from numpy import random  
x = random.randint(100)  
print(x)
```

# Debe imprimir un numero entero entre 0 y 100

#### ➤ Generación de aleatorios en Numpy

```
from numpy import random  
x = random.rand()  
print(x)
```

# Debe imprimir un flotante entre 0 y 1

#### ➤ Generación de aleatorios en arreglos en Numpy

```
from numpy import random  
x = random.randint(100, size=(5))  
print(x)  
  
x = random.randint(100, size=(3, 5))  
print(x)  
  
x = random.rand(3, 5)  
print(x)
```

# Debe imprimir un vector de 5 elementos con enteros entre 0 y 100

# Debe imprimir una matriz de 3 x 5 elementos con enteros entre 0 y 100

# Debe imprimir una matriz de 3 x 5 elementos con flotantes entre 0 y 1

## 3.1.5 Librería NUMPY



### Aleatorios con Numpy

#### ➤ Selección aleatoria de elementos en arreglos en Numpy

```
from numpy import random  
x = random.choice([3, 5, 8, 15, 18])  
print(x)                                # Debe imprimir uno de los elementos del vector escogido aleatoriamente
```

```
from numpy import random  
x = random.choice([3, 5, 8, 15, 18], size=(3, 5))  
print(x)                                # Debe imprimir una matriz 3x5 con elementos del vector escogido aleatoriamente
```

#### ➤ Distribución aleatoria en Numpy

Una distribución aleatoria es un conjunto de números aleatorios que siguen una determinada función de densidad de probabilidad

```
from numpy import random  
x = random.choice([1, 3, 5, 7], p=[0.2, 0.3, 0.4, 0.1], size=(100))  
print(x)  
  
x = random.choice([1, 3, 5, 7], p=[0.2, 0.3, 0.4, 0.1], size=(3,5))  
print(x)
```

## 3.1.5 Librería NUMPY



### Aleatorios con Numpy

#### ➤ Permutaciones aleatorias en Numpy

```
from numpy import random  
import numpy as np  
arr = np.array( [1, 3, 5, 7, 9])
```

```
random.shuffle(arr)  
print(arr)  
  
print(random.permutation(arr))
```

# **Shuffle** hace cambios en el arreglo original  
# Debe imprimir el vector con los elementos en otra disposición

# Debe imprimir el vector con los elementos en otra disposición **no lo modifica**

## 3.1.5 Librería NUMPY



### Distribuciones con Numpy, Matplotlib y Seaborn (<https://seaborn.pydata.org/>)

#### ➤ Distribución Normal en Numpy

```
from numpy import random  
arr = random.normal(size=(2, 3))  
print(arr)
```

# Genera números aleatorios normales N(0,1)  
# Debe imprimir una matriz 2x3 con los elementos flotantes Normales

```
from numpy import random  
arr = random.normal(loc=1, scale=2, size=(2, 3)) # Genera números aleatorios normales con media=1 y desv estándar= 2  
print(arr)
```

# Debe imprimir una matriz 2x3 con los elementos flotantes Normales

```
from numpy import random  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
sns.distplot(random.normal(size=1000), hist=False)  
plt.show()
```

## 3.1.5 Librería NUMPY



### Distribuciones con Numpy, Matplotlib y Seaborn (<https://seaborn.pydata.org/>)

#### ➤ Distribución Binomial en Numpy

```
from numpy import random  
x = random.binomial(n=10, p=0.5, size=10)  
print(x) # Debe imprimir una matriz 2x3 con los elementos flotantes Normales
```

```
from numpy import random  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
sns.distplot(random.binomial(n=10, p=0.5, size=1000), hist= True, kde=False)  
plt.show()
```

#### ➤ Diferencia entre la Distribución Normal y la Binomial en Numpy

```
from numpy import random  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
sns.distplot(random.normal(loc=50, scale=5, size=1000), hist=False, label='normal')  
sns.distplot(random.binomial(n=100, p=0.5, size=1000), hist=False, label='binomial')  
plt.show()
```

## 3.1.5 Librería NUMPY



### Distribuciones con Numpy, Matplotlib y Seaborn (<https://seaborn.pydata.org/>)

#### ➤ Distribución Poisson en Numpy

```
from numpy import random  
x = random.poisson(lam=2, size=10)  
print(x) # Debe imprimir un vector de 10 elementos con Dist. Poisson  
  
from numpy import random  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
sns.distplot(random.poisson(lam=2, size=1000), kde=False)  
plt.show()
```

#### ➤ Diferencia entre la Distribución Normal y la Poisson en Numpy

```
from numpy import random  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
sns.distplot(random.normal(loc=50, scale=5, size=1000), hist=False, label='normal')  
sns.distplot(random.poisson(lam=2, size=1000), hist=False, label='poissonl')  
plt.show()
```

# INSERTAR NOMBRE DE ACTIVIDAD



## PASOS DE ACTIVIDAD individual /grupal:

- Ingresar al colab personal con su cuenta gmail
  
- Copia el enunciado del problema señalado por el profesor y péguelo en el colab
  
- Desarrolle el ejercicio junto con un compañero, ya sea en el laboratorio o en la sala grupal creada por el docente.

# Resolución de ejercicios

## ACTIVIDAD

Debe resolver los ejercicios señalados por el profesor que se encuentran en el documento:

[03 - 01 Programación Orientada a Objetos - Guía laboratorio y Casos propuestos.pdf](#)

Este documento se encuentra en el aula virtual en la unidad 3



# CONCLUSIONES

01

Las librerías son software desarrollado por terceros con funcionalidades avanzadas, específicas y especializadas que potencian a Python estándar. Son gratuitas

02

Las diferentes librerías se compatibles con otras y se potencian entre si.

03

Las librerías enseñadas en clase son las mas populares y son la base para estudios superiores de ciencia de los datos, big data y otros tópicos de inteligencia artificial.

04

Python estándar como las librerías han sido desarrolladas con la arquitectura orientada a objetos.

# BIBLIOGRAFÍA

- Direcciones electrónicas (hipervínculos en las diapositivas)
- LUTZ, MARK (2013) Learning Python. 5th Edition. California: O'Reilly.
- RAMALHO, LUCIANO (2015) Fluent Python: Clear, Concise, and Effective Programming (inglés) 1st Edición. California: O'Reilly.
-

# **Continúa con las actividades propuestas**

---



Material producido por la  
Universidad Peruana de  
Ciencias Aplicadas

Autor:

Norman Reyes Morales

COPYRIGHT © UPC  
2020 – Todos los  
derechos reservados



---

## **UNIDAD Nº: 4 PROGRAMACIÓN ORIENTADA A OBJETOS - Librerías**

## LOGRO DE APRENDIZAJE

Al finalizar la unidad, el estudiante aplica los conocimientos de las unidades anteriores y de las librerías enseñadas en clases en un contexto real o simulado. Analiza y desarrolla todos los elementos del componente a desarrollar empleando el lenguaje de programación Python, demostrando una actitud analítica, organizada y participativa en su equipo de trabajo.





## TEMA 6

---

Librerías

### 3.1.1 Librería Matplotlib

## Librería MatPlotLib en Python



La Matplotlib es una librería para gráficos en 2D de alta calidad.

➤ Instalación de Matplotlib

```
python -m pip install --U pip
```

```
python -m pip install --U matplotlib
```

➤ Instalación de Matplotlib en ANACONDA

```
conda install matplotlib
```

➤ Manual de la librería

<https://matplotlib.org/stable/contents.html>

➤ Donde **matplotlib** es la biblioteca, **pyplot** es un paquete que incluye todas funciones de MATLAB para utilizar las funciones de MATLAB en Python. Finalmente, podemos usar **plt** para llamar a las funciones dentro del archivo Python.

```
import matplotlib.pyplot as plt
```

### 3.1.1 Librería Matplotlib

## Módulo o Librería MatPlotLib en Python



### Dibujando gráficos simples

Supongamos que queremos dibujar una línea en el siguiente conjunto de puntos:  $x=(4, 8, 13, 17, 20)$  e  $y=(54, 67, 98, 78, 45)$

```
import matplotlib.pyplot as plt  
plt.plot([4, 8, 13, 17, 20] , [54, 67, 98, 78, 45])  
plt.show() # Debe dibujarle el grafico
```

Se puede personalizar la grafica: p.e. que la línea sea punteada y verde con marcadores de diamante, para lo cual se le agrega el parámetro 'g--d'

```
import matplotlib.pyplot as plt  
plt.plot([4, 8, 13, 17, 20] , [54, 67, 98, 78, 45] , 'g--d')  
plt.title("Ejemplo de Gráfica")  
plt.xlabel("Eje X")  
plt.ylabel("Eje Y")  
plt.show() # Debe dibujarle el grafico con las características personalizadas
```

### 3.1.1 Librería Matplotlib

## Módulo o Librería Matplotlib en Python



### Dibujando gráficos simples

Se puede realizar un Histograma de frecuencia que datos de un vector:

```
x=[2,4,6,5,42,543,5,3,73,64,42,97,63,76,63,8,73,97,23,45,56,89,45,3,23,2,5,78,23,56,67,78,8,3,78,34,67,23,324,234,43,544,54,33,223,443,444,234,76,432,233,23,232,243,222,221,254,222,276,300,353,354,387,364,309]
```

```
import matplotlib.pyplot as plt  
x=[2,4,6,5,42,543,5,3,73,64,42,97,63,76,63,8,73,97,23,45,56,89,45,3,23,2,5,78,23,56,67,78,8,3,78,34,67,23,324,234,43,544,54,33,223,443,444,234,76,432,233,23,232,243,222,221,254,222,276,300,353,354,387,364,309]  
num_bis = 6  
n, bins, patches = plt.hist(x, num_bis, facecolor ='green')  
plt.title("Ejemplo de Histograma")  
plt.xlabel("Eje X")  
plt.ylabel("Eje Y")  
plt.legend(prop ={'size': 10})  
plt.show() # Debe dibujarle el Histograma
```

### 3.1.2 Librería Datetime

# Módulo o Librería Datetime en Python



Una fecha en Python no es un tipo de datos propio, pero podemos importar un módulo llamado **datetime** para trabajar con fechas como objetos de fecha.

## ➤ Carga y uso de la librería Datetime

Importar el módulo Datetime y mostrar la fecha actual:

```
import datetime
x = datetime.datetime.now()
print(x)
print(x.year)
print(x.strftime("%A"))
```

# El resultado es 2021-02-20 22:15:49.974845  
# Se pueden usar métodos. El resultado es 2021  
# Se pueden usar métodos. El resultado es Tuesday

## ➤ Creación de objetos fecha

Para crear una fecha, podemos usar la clase `datetime()` (constructor) del módulo `datetime`. The `datetime()` class requires three parameters to create a date: year, month, day

### 3.1.2 Librería Datetime

## Módulo o Librería Datetime en Python



### ➤ Método strftime()

El objeto datetime tiene un método para formatear objetos de fecha en cadenas legibles. El método se llama **strftime ()** y toma un parámetro, que es el formato y sirve para especificar el formato de la cadena devuelta:

```
import datetime  
x = datetime.datetime.now()  
print(x.strftime("%A"))  
print(x.strftime("%B"))
```

# Se pueden usar métodos. El resultado es Thursday  
# El resultado es August

### 3.1.2 Librería Datetime

## Módulo o Librería Datetime en Python



### ➤ Método strftime() - operadores

Directive	Description	Example	Try it
%a	Weekday, short version	Wed	<a href="#">Try it »</a>
%A	Weekday, full version	Wednesday	<a href="#">Try it »</a>
%w	Weekday as a number 0-6, 0 is Sunday	3	<a href="#">Try it »</a>
%d	Day of month 01-31	31	<a href="#">Try it »</a>
%b	Month name, short version	Dec	<a href="#">Try it »</a>
%B	Month name, full version	December	<a href="#">Try it »</a>
%m	Month as a number 01-12	12	<a href="#">Try it »</a>
%y	Year, short version, without century	18	<a href="#">Try it »</a>
%Y	Year, full version	2018	<a href="#">Try it »</a>
%H	Hour 00-23	17	<a href="#">Try it »</a>
%I	Hour 00-12	05	<a href="#">Try it »</a>
%p	AM/PM	PM	<a href="#">Try it »</a>
%M	Minute 00-59	41	<a href="#">Try it »</a>
%S	Second 00-59	08	<a href="#">Try it »</a>
%f	Microsecond 000000-999999	548513	<a href="#">Try it »</a>
%z	UTC offset	+0100	
%Z	Timezone	CST	
%j	Day number of year 001-366	365	<a href="#">Try it »</a>
%U	Week number of year, Sunday as the first day of week, 00-53	52	<a href="#">Try it »</a>
%W	Week number of year, Monday as the first day of week, 00-53	52	<a href="#">Try it »</a>
%c	Local version of date and time	Mon Dec 31 17:41:00 2018	<a href="#">Try it »</a>
%x	Local version of date	12/31/18	<a href="#">Try it »</a>
%X	Local version of time	17:41:00	<a href="#">Try it »</a>
%%	A % character	%	<a href="#">Try it »</a>

### 3.1.4 Librería Math

## Módulo o Librería Math en Python



Python tiene un conjunto de funciones matemáticas integradas, incluido un extenso módulo matemático, que le permite realizar tareas matemáticas en números.

#### ➤ Funciones matemáticas incorporadas

Las funciones min() y max() se pueden usar para encontrar el valor más bajo o más alto en una colección:

```
import math
x = min(5, 10, 25)
y = max(5, 10, 25)
print("El valor mínimo es ", x)           # El resultado es 5
print("El valor maximo es ",y)            # El resultado es 25
x = math.sqrt(64)
print("La raiz de 64 es ", x)             # El resultado es 8.0
```

#### ➤ Métodos de redondeo

El método math.ceil() redondea un número hacia arriba a su número entero más cercano, y el método math.floor() redondea un número hacia abajo a su número entero más cercano y devuelve el resultado:

```
import math
x = math.ceil(1.4)
y = math.floor(1.4)
print(x)                                # El resultado es 2
print(y)                                # El resultado es 1
```

### 3.1.4 Librería Math

## Módulo o Librería Math en Python



Otros métodos de esta librería <https://docs.python.org/3/library/numeric.html>

Method	Description
<a href="#">math.acos(x)</a>	Returns the arc cosine value of x
<a href="#">math.acosh(x)</a>	Returns the hyperbolic arc cosine of x
<a href="#">math.asin(x)</a>	Returns the arc sine of x
<a href="#">math.asinh(x)</a>	Returns the hyperbolic arc sine of x
<a href="#">math.atan(x)</a>	Returns the arc tangent value of x
<a href="#">math.atan2(y, x)</a>	Returns the arc tangent of y/x in radians
<a href="#">math.atanh(x)</a>	Returns the hyperbolic arctangent value of x
<a href="#">math.ceil(x)</a>	Rounds a number upwards to the nearest integer, and returns the result
<a href="#">math.comb(n, k)</a>	Returns the number of ways to choose k items from n items without repetition and order
<a href="#">math.copysign(x, y)</a>	Returns a float consisting of the value of the first parameter and the sign of the second parameter
<a href="#">math.cos(x)</a>	Returns the cosine of x
<a href="#">math.cosh(x)</a>	Returns the hyperbolic cosine of x
<a href="#">math.degrees(x)</a>	Converts an angle from radians to degrees
<a href="#">math.dist(p, q)</a>	Calculates the euclidean distance between two specified points (p and q), where p and q are the coordinates of that point
<a href="#">math.erf(x)</a>	Returns the error function of x
<a href="#">math.erfc(x)</a>	Returns the complementary error function of x
<a href="#">math.exp(x)</a>	Returns the value of $E^x$ , where E is Euler's number (approximately 2.718281...), and x is the number passed to it
<a href="#">math.expm1(x)</a>	Returns the value of $E^x - 1$ , where E is Euler's number (approximately 2.718281...), and x is the number passed to it
<a href="#">math.fabs(x)</a>	Returns the absolute value of a number
<a href="#">math.factorial()</a>	Returns the factorial of a number
<a href="#">math.floor(x)</a>	Rounds a number downwards to the nearest integer, and returns the result
<a href="#">math.fmod(x, y)</a>	Returns the remainder of specified numbers when a number is divided by another number
<a href="#">math.frexp()</a>	Returns the mantissa and the exponent, of a specified value
<a href="#">math.fsum(iterable)</a>	Returns the sum of all items in an iterable (tuples, arrays, lists, etc.)
<a href="#">math.gamma(x)</a>	Returns the gamma value of x
<a href="#">math.gcd()</a>	Returns the highest value that can divide two integers
<a href="#">math.hypot()</a>	Find the Euclidean distance from the origin for n inputs
<a href="#">math.isclose()</a>	Checks whether two values are close, or not
<a href="#">math.isfinite(x)</a>	Checks whether x is a finite number
<a href="#">math.isinf(x)</a>	Check whether x is a positive or negative infinity

### 3.1.4 Librería Math

## Módulo o Librería Math en Python



Method	Description
<a href="#">math.isnan(x)</a>	Checks whether x is NaN (not a number)
<a href="#">math.isqrt(n)</a>	Returns the nearest integer square root of n
<a href="#">math.ldexp(x, i)</a>	Returns the expression $x * 2^i$ where x is mantissa and i is an exponent
<a href="#">math.lgamma(x)</a>	Returns the log gamma value of x
<a href="#">math.log(x, base)</a>	Returns the natural logarithm of a number, or the logarithm of number to base
<a href="#">math.log10(x)</a>	Returns the base-10 logarithm of x
<a href="#">math.log1p(x)</a>	Returns the natural logarithm of 1+x
<a href="#">math.log2(x)</a>	Returns the base-2 logarithm of x
<a href="#">math.perm(n, k)</a>	Returns the number of ways to choose k items from n items with order and without repetition
<a href="#">math.pow(x, y)</a>	Returns the value of x to the power of y
<a href="#">math.prod(iterable, *, start=1)</a>	Returns the product of an iterable (lists, array, tuples, etc.)
<a href="#">math.radians(x)</a>	Converts a degree value (x) to radians
<a href="#">math.remainder(x, y)</a>	Returns the closest value that can make numerator completely divisible by the denominator
<a href="#">math.sin(x)</a>	Returns the sine of x
<a href="#">math.sinh(x)</a>	Returns the hyperbolic sine of x
<a href="#">math.sqrt(x)</a>	Returns the square root of x
<a href="#">math.tan(x)</a>	Returns the tangent of x
<a href="#">math.tanh(x)</a>	Returns the hyperbolic tangent of x
<a href="#">math.trunc(x)</a>	Returns the truncated integer parts of x

Constant	Description
<a href="#">math.e</a>	Returns Euler's number (2.7182...)
<a href="#">math.inf</a>	Returns a floating-point positive infinity
<a href="#">math.nan</a>	Returns a floating-point NaN (Not a Number) value
<a href="#">math.pi</a>	Returns PI (3.1415...)
<a href="#">math.tau</a>	Returns tau (6.2831...)

Fuente: <https://www.w3schools.com/python>

Constantes esta librería

<https://docs.python.org/3/library/numeric.html>

## 3.1.4 Librería TKINTER

### TKINTER



**Tkinter** es la biblioteca estándar de Python para construir interfaces gráficas (GUI).

La estructura que se crea cuando se usa **tkinter** es la que se denomina **event driven** (dirigida por eventos). Esta consiste en crear la GUI y esperar a que ocurra un evento. Durante la espera, la librería Tk se encuentra en un **event loop (bucle)** vigilante a cualquier evento como una pulsación de ratón o de una tecla, una vez se produce se llama inmediatamente a un código para su ejecución.

Cada programa de tkinter consta de estas cosas:

- Ventanas, botones, barras de desplazamiento, áreas de texto y otros widgets
- Un widget es cualquier cosa que pueda ver en la pantalla de la computadora. (Generalmente, el término widget significa cualquier objeto útil; es la abreviatura de “window gadget”).
- Módulos, funciones y clases que administran los datos que se muestran en la GUI
- Un administrador de eventos que “escucha” eventos como clics del mouse y pulsaciones de teclas y reacciona a estos eventos llamando a las funciones del controlador de eventos.

#### ➤ Instalación de Tkinter en ANACONDA

```
# The actual install command  
conda install -c anaconda tk
```

Manual de la librería

<https://guia-tkinter.readthedocs.io/es/develop/>  
<https://docs.python.org/es/3/library/tk.html>

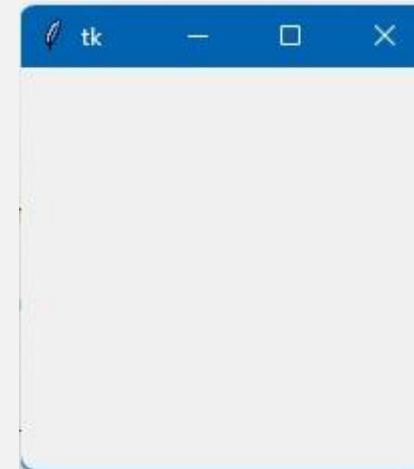
### 3.1.4 Librería TKINTER

## TKINTER



Veamos el siguiente código

```
import tkinter as tk          # importa la librería tkinter  
raiz = tk.Tk()                # crea ventana  
raiz.mainloop()              # ejecuta el event loop
```



- La primera línea importa todo el material de la GUI del módulo tkinter.
- La segunda línea crea una ventana en la pantalla, que llamamos raíz.
- La tercera línea coloca el programa en lo que es esencialmente un bucle eterno llamado event loop o bucle de eventos
- Al ejecutarlo se obtiene la pantalla de la imagen adjunta
- Si hubiera código debajo de raíz.mainloop, no se ejecutará hasta que se cierre la ventana
- Por lo general, raíz.mainloop será la última línea del código

### 3.1.4 Librería TKINTER

## TKINTER



- Existen dos métodos distintos para modificar la estética de la interfaz para diferenciarlos suele llamarlos **controles clásicos** y **controles temáticos**.
- Los **controles clásicos** son los controles originales de **Tk**, a los cuales desde Python accedemos vía el módulo **tkinter** que generalmente se abrevia **tk**.
- Los **controles temáticos** se introdujeron en la versión 8.5 de Tk y están contenidos en el submódulo de Python **ttk**.
- La diferencia entre los dos tipos de controles es principalmente estética
- Los **controles temáticos** tienen una apariencia más moderna e incluyen un sistema de personalización de esa apariencia mejor que el sistema de **controles clásicos**.
- Por ello muchas veces un programa inicia así

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()

ventana.mainloop()
```

## 3.1.4 Librería TKINTER

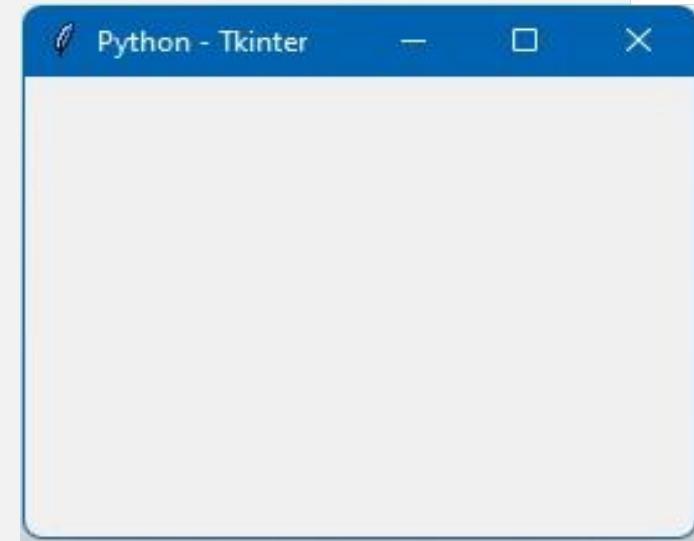
### TKINTER - Poniéndole título a la ventana



```
# Importar librería  
import tkinter as tk
```

```
# inicializar ventana  
ventana = tk.Tk()  
ventana.title("Python - Tkinter")
```

```
# Activar ventana  
ventana.mainloop()
```



### TKINTER - Tamaño de la ventana: Geometry

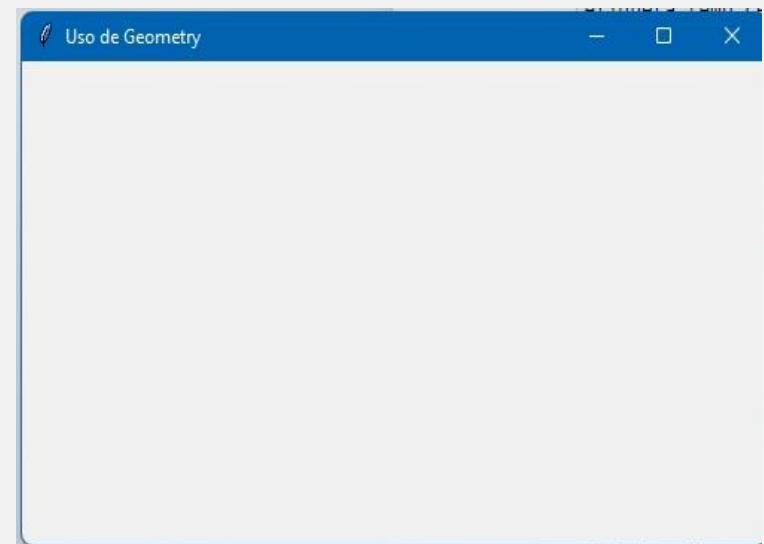
```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
#Asignar un tamaño a la ventana  
#geometry('ancho X alto + posx + posy')  
ventana.geometry('500x300+100+100')
```

```
ventana.title('Uso de Geometry')
```

```
ventana.mainloop()
```



### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con pack



```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
ventana.geometry('200x200+100+100')  
ventana.title('Uso de Etiquetas')
```

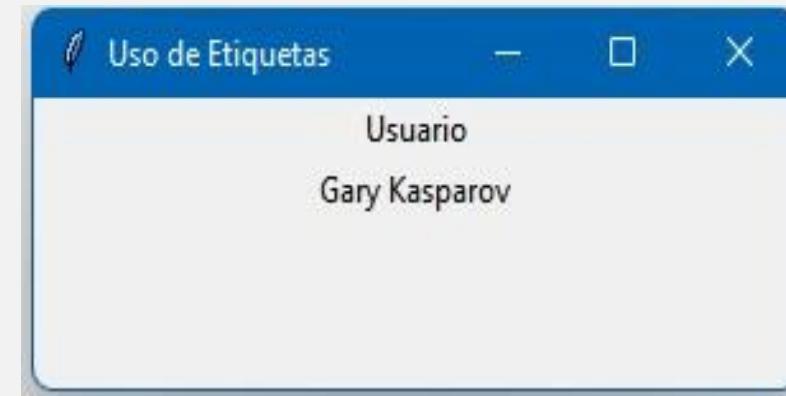
*#Creamos Las etiquetas*

```
labelUsuario = tk.Label(text='Usuario')  
labelDatos = tk.Label(text='Gary Kasparov')
```

*#Agregamos las etiquetas a la ventana*

```
labelUsuario.pack()  
labelDatos.pack()
```

```
ventana.mainloop()
```



### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con pack



```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
ventana.geometry('300x100+100+100')
```

```
ventana.title('Uso de Etiquetas')
```

*#Creamos Las etiquetas*

```
labelUsuario = tk.Label(text='Usuario')
```

```
labelDatos = tk.Label(text='Gary Kasparov')
```

```
labelProfesion = tk.Label(text='Ajedrecista')
```

```
labelTitulo = tk.Label(text='Campeón Mundial de Ajedrez')
```

*#Agregamos la etiqueta a la ventana*

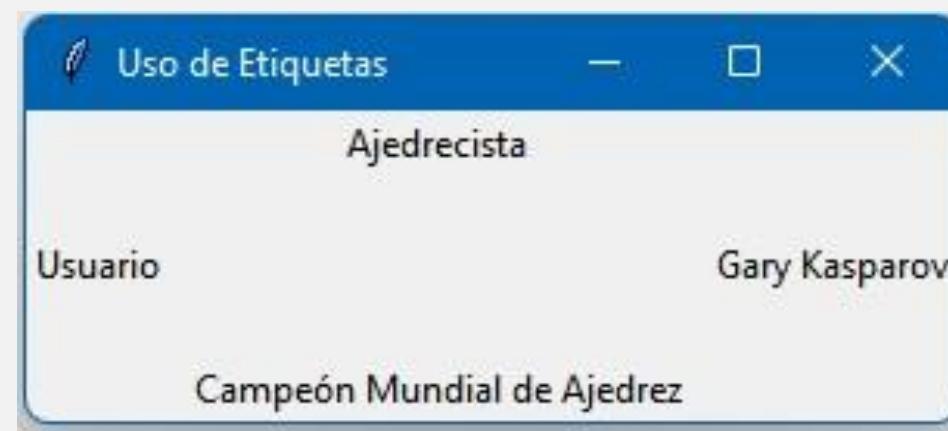
```
labelUsuario.pack(side='left')
```

```
labelDatos.pack(side='right')
```

```
labelProfesion.pack(side='top')
```

```
labelTitulo.pack(side='bottom')
```

```
ventana.mainloop()
```



### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con place



Otro método utilizado es el llamado **place** en vez de **pack**. Hay que tener cuidado ya que las medidas que se usan están en pixeles y no es caracteres por lo que muchas veces hay que probar hasta llegar a la ubicación deseada.

```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
ventana.geometry('300x100+100+100')  
ventana.title("Uso de Etiquetas")
```

*#Creamos Las etiquetas y le damos su ubicación con place*

```
labelUsuario = tk.Label(text='Usuario').place(x=10, y=10)  
labelDatos = tk.Label(text='Gary Kasparov').place(x=120, y=10)  
labelProfesion = tk.Label(text='Ajedrecista').place(x=10, y=30)  
labelTitulo = tk.Label(text='Campeón Mundial de Ajedrez').place(x=120, y=30)
```

```
ventana.mainloop()
```



### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con grid



- Tkinter **grid** es otro y también el más importante método de geometría de diseño
- Hay que tener en cuenta que utilizaremos la ventana como una grilla.

	columna 0	columna 1	columna 2	columna 3
fila 0	(0,0)	(0,1)	(0,2)	(0,3)
fila 1	(1,0)	(1,1)	(1,2)	(1,3)
fila 2	(2,0)	(2,1)	(2,2)	(2,3)
fila 3	(3,0)	(3,1)	(3,2)	(3,3)

### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con grid



- También se podrá usar la función **sticky** para la ubicación del texto
- **sticky** determina cómo se pega el widget a la celda cuando el widget es más pequeño que la celda.

sticky	Significado
W	pegarse a la izquierda
E	mantenerse a la derecha
N	pegarse a la parte superior
S	adherirse a la parte de abajo

### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con grid



```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
ventana.geometry('300x100+100+100')
```

```
ventana.title('Uso de Etiquetas')
```

#Creamos Las etiquetas y le damos su ubicación con grid

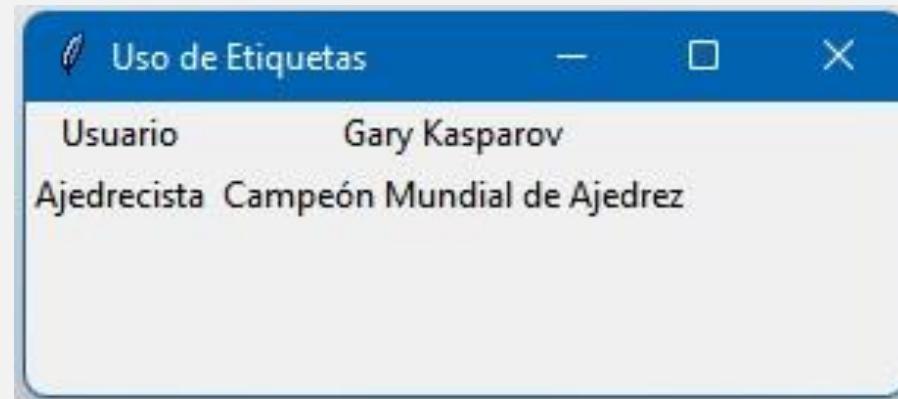
```
labelUsuario = tk.Label(text='Usuario').grid(row=0, column=0)
```

```
labelDatos = tk.Label(text='Gary Kasparov').grid(row=0, column=1)
```

```
labelProfesion = tk.Label(text='Ajedrecista').grid(row=1, column=0)
```

```
labelTitulo = tk.Label(text='Campeón Mundial de Ajedrez').grid(row=1, column=1)
```

```
ventana.mainloop()
```



### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con grid



```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
ventana.geometry('300x100+100+100')
```

```
ventana.title('Uso de Etiquetas')
```

*#Creamos Las etiquetas y le damos su ubicación con grid*

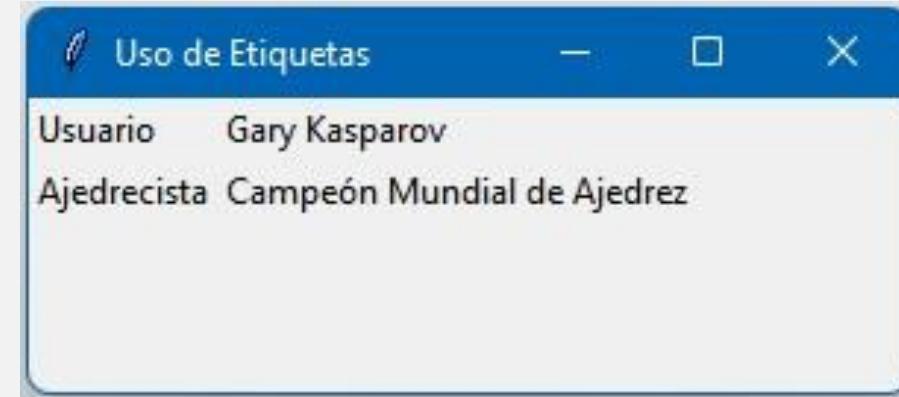
```
labelUsuario = tk.Label(text='Usuario').grid(row=0, column=0, sticky=tk.W)
```

```
labelDatos = tk.Label(text='Gary Kasparov').grid(row=0, column=1, sticky=tk.W)
```

```
labelProfesion = tk.Label(text='Ajedrecista').grid(row=1, column=0, sticky=tk.E)
```

```
labelTitulo = tk.Label(text='Campeón Mundial de Ajedrez').grid(row=1, column=1, sticky=tk.E)
```

```
ventana.mainloop()
```



### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con grid



```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
ventana.geometry('300x100+100+100')
```

```
ventana.title('Uso de Etiquetas')
```

*#Creamos Las etiquetas y le damos su ubicación con grid*

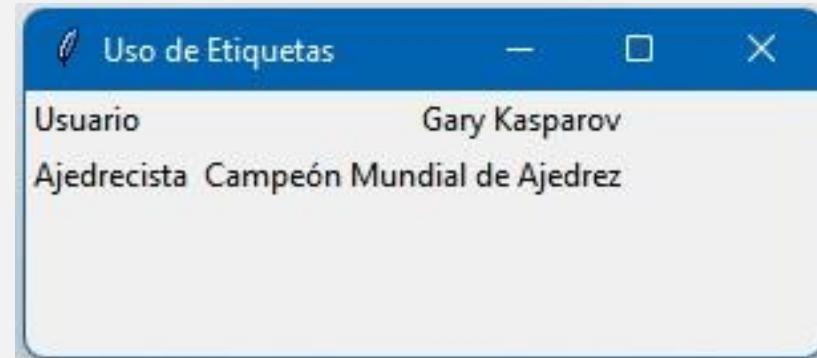
```
labelUsuario = tk.Label(text='Usuario').grid(row=0, column=0, sticky=tk.W)
```

```
labelDatos = tk.Label(text='Gary Kasparov').grid(row=0, column=1,
                                                 columnspan=2, rowspan=1,
                                                 sticky=tk.E)
```

```
labelProfesion = tk.Label(text='Ajedrecista').grid(row=1, column=0,
                                                 sticky=tk.W)
```

```
labelTitulo = tk.Label(text='Campeón Mundial de Ajedrez').grid(row=1,
                                                               column=1,
                                                               columnspan=2,
                                                               rowspan=1,
                                                               sticky=tk.E)
```

```
ventana.mainloop()
```



columnspan y  
rowspan permiten  
que el campo  
ocupe más de una  
cuadrícula de la  
grilla

### 3.1.4 Librería TKINTER

#### TKINTER - Etiquetas ubicadas con grid

Otra forma de escribir el código anterior

```
import tkinter as tk
```

```
ventana = tk.Tk()
```

```
ventana.geometry('300x100+100+100')
```

```
ventana.title('Uso de Etiquetas')
```

#Creamos Las etiquetas y le damos su ubicación con grid

```
labelUsuario = tk.Label(text='Usuario')
```

```
labelUsuario.grid(row=0, column=0, sticky=tk.W)
```

```
labelDatos = tk.Label(text='Gary Kasparov')
```

```
labelDatos.grid(row=0, column=1, columnspan=2, rowspan=1, sticky=tk.E)
```

```
labelProfesion = tk.Label(text='Ajedrecista')
```

```
labelProfesion.grid(row=1, column=0, sticky=tk.W)
```

```
labelTitulo = tk.Label(text='Campeón Mundial de Ajedrez')
```

```
labelTitulo.grid(row=1, column=1, columnspan=2, rowspan=1, sticky=tk.E)
```

```
ventana.mainloop()
```



## 3.1.4 Librería TKINTER



### TKINTER - Agregar Botones

Considere el siguiente código

```
import tkinter as tk
from tkinter import ttk

def funcion_click():
    accion.configure(text='**¡Haz hecho click!**')
    etiqueta.configure(foreground='red')

ventana = tk.Tk()
ventana.geometry('300x100+100+100')
ventana.title('Python -Tkinter')

etiqueta = ttk.Label(ventana, text='Hola a todos!!!')
etiqueta.grid(row=0, column=0)

# Agrega un boton
accion = ttk.Button(ventana, text='Haz click aquí',
                    command=funcion_click)
accion.grid(column=1, row=0)

ventana.mainloop()
```

Para que el botón haga algo cuando se hace clic en él, se usa el argumento **command**. Se pone en el **nombre de una función**. Cuando se hace clic en el botón, se llama a la función, en el ejemplo, es la función llamada: **funcion\_click**. Después de haber creado una **label**, es posible que desee cambiar algo de dicha **label**. Para hacer eso, se usa el método **configure**. En el ejemplo cambia el texto del botón **acción** y el color de letra de **etiqueta**.

## 3.1.4 Librería TKINTER

### TKINTER - Agregar Caja de Texto (Entry)



```
import tkinter as tk
from tkinter import ttk

def funcion_click():
    accion.configure(text='Hola ' + nombre.get())
    # get() obtiene el valor ingresado

# Inicializar ventana
ventana = tk.Tk()
ventana.geometry('300x100+100+100')
ventana.title("Python - Tkinter")

# Agregar etiqueta por medio de un objeto
etiqueta = ttk.Label(ventana, text="Escribe tu nombre")
etiqueta.grid(column=0, row=0)

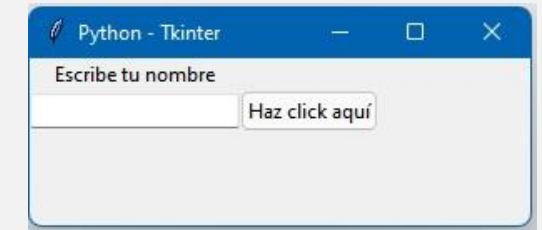
# Agregar una caja de texto
nombre = tk.StringVar() # se crea la variable para Entry con StringVar
preguntar_nombre = ttk.Entry(ventana, width=20, textvariable=nombre)
preguntar_nombre.grid(column=0, row=1)

# Agregar un botón
accion = ttk.Button(ventana, text="Haz click aquí", command=funcion_click)
accion.grid(row=1, column=1)

ventana.mainloop()
```

- **StringVar()** permite crear la variable para **Entry** la cual se usará en **textvariable**
- **Width** establece el ancho de la caja de entrada

Antes del click



Después del click



## 3.1.4 Librería TKINTER

### TKINTER - Agregar ComboBox



```
import tkinter as tk
from tkinter import ttk

def funcion_click():
    respuesta = ttk.Label(ventana, text='Seleccionado el ' + numero.get())
    respuesta.grid(column=1, row=3)

ventana = tk.Tk()
ventana.geometry('300x100+100+100')
ventana.title("Python - Tkinter")

etiqueta = ttk.Label(ventana, text="Selecciona un número")
etiqueta.grid(column=0, row=0)

# Agregar lista desplegable
numero = tk.StringVar()
seleccionar_numero = ttk.Combobox(ventana, width=12, textvariable=numero)

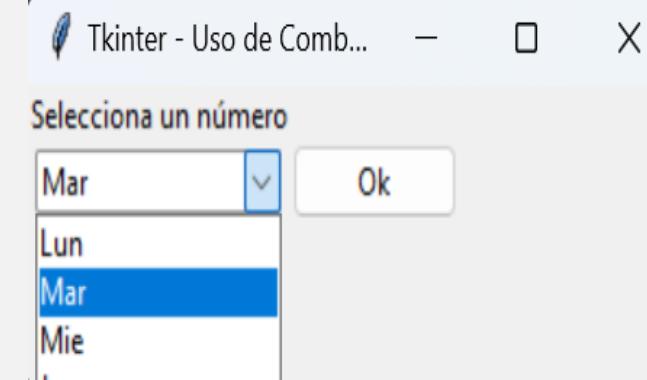
# Llenar lista desplegable
seleccionar_numero['values'] = ("Lun","Mar","Mie","Jue","Vie","Sab","Dom")

# Posicionar lista desplegable
seleccionar_numero.grid(column=0, row=1)

# Elemento de la lista seleccionado por default
seleccionar_numero.current(0)
accion = ttk.Button(ventana, text="Ok", command=funcion_click)
accion.grid(row=1, column=1)

ventana.mainloop()
```

- `ttk.Combobox` crea la lista desplegable
- `seleccionar_numero['values']`, permite asignar los valores de la lista desplegable
- `seleccionar_numero.current(0)`, establece la posición del valor por defecto



### 3.1.4 Librería TKINTER

#### TKINTER - FRAME

El frame es un contenedor de otros widgets

- Ventana raíz de la aplicación es prácticamente un frame (marco)
- Cada marco tiene su propio diseño de cuadrícula, por lo que la cuadrícula (grid) de widgets dentro de cada marco funciona de forma independiente.
- Los widgets de marco son una herramienta valiosa para hacer que su aplicación sea modular. Puede agrupar un conjunto de widgets relacionados en un widget compuesto colocándolos en un marco. Mejor aún, puede declarar una nueva clase que herede de Frame, agregándole su propia interfaz. Esta es una buena manera de ocultar los detalles de las interacciones dentro de un grupo de widgets relacionados del mundo exterior.
- Para crear un nuevo widget de marco en una ventana raíz o marco llamado parent:

```
import tkinter as tk
```

```
marco = Frame(parent, option, ...)
```

Parametros	descripción
bg or background	Color de fondo del marco
bd o borderwidth	Ancho del borde del marco. El valor predeterminado es 0 (sin borde)
cursor	El cursor utilizado cuando el mouse está dentro del widget de marco
height	La dimensión vertical del nuevo marco.
highlightbackground	Color del resaltado de enfoque cuando el marco no tiene foco
highlightcolor	Color que se muestra en el resaltado de enfoque cuando el marco tiene el foco
highlightthickness	Grosor del punto culminante del foco.
padx	Normalmente, un marco se ajusta perfectamente a su contenido. Para agregar N píxeles de espacio horizontal dentro del marco, establezca padx=N.
pady	Se utiliza para agregar espacio vertical dentro de un marco. Ver padx arriba.
relief	El relieve predeterminado para un marco es tk.FLAT, lo que significa que el marco se mezclará con su entorno. Para colocar un borde alrededor de un marco, establezca su ancho de borde en un valor positivo y establezca su relieve en uno de los tipos de relieve estándar
takefocus	Normalmente, los controles de entrada no visitan los widgets de marco. Sin embargo, puede configurar takefocus=1 si desea que el marco reciba la entrada del teclado. Para manejar dicha entrada, deberá crear enlaces para eventos de teclado
width	La dimensión horizontal del nuevo marco.

### 3.1.4 Librería TKINTER

## TKINTER - Otras opciones de tkinter



Tkinter tiene muchos otros widgets, los cuales pueden ser consultadas en la documentación. Se muestra una lista de otros widgets disponibles en tkinter

Widget	Descripción
Spinbox	Permite seleccionar un valor de un conjunto de valores. Los valores pueden ser un rango de números.
Canvas	Un área utilizada para dibujar o mostrar imágenes.
Checkbutton	Un cuadro en el que se puede hacer clic que se puede seleccionar o deseleccionar
Frame	Un contenedor para widgets
Menu	Un menú desplegable
Message	Una pantalla de varias líneas para texto
Menobutton	Un elemento en un menú desplegable
Text	Un campo de texto de varias líneas en el que el usuario puede escribir
TopLevel	Una ventana adicional
Radiobutton	Permite elegir entre una de varias opciones mutuamente excluyentes
Listbox	Permite elegir entre un conjunto de opciones o muestra una lista de elementos.
Scale	Se utiliza para implementar un control deslizante gráfico, da la opción de elegir entre un rango de valores.
Notebook	Permite dividir una parte de la ventana en distintas solapas de modo que, dependiendo de la que se encuentre seleccionada, varía el contenido de la interfaz

### 3.1.4 Librería TKINTER

## TKINTER - Programa que convierte temperaturas de Celsius a Fahrenheit.



```
import tkinter as tk

def convierte():
    temp = int(entrada.get())
    temp = 9/5*temp+32
    etiquetaSalida.configure(text = 'En Fahrenheit: {:.1f}'.format(temp))
    entrada.delete(0,tk.END)

raiz = tk.Tk()
raiz.geometry('400x100+100+100')
raiz.title("Convertir de Celsius a Fahrenheit")

etiquetaMsg = tk.Label(text='Ingrese temperatura °C', font=('Calibri', 14))
etiquetaMsg.grid(row=0, column=0)

etiquetaSalida = tk.Label(font=('Calibri', 14))
etiquetaSalida.grid(row=1, column=0, columnspan=3)

entrada = tk.Entry(font=('Calibri', 14), width=4)
entrada.grid(row=0, column=1)

botonCalc = tk.Button(text='Ok', font=('Calibri', 14), command=convierte)
botonCalc.grid(row=0, column=2)

raiz.mainloop()
```

### 3.1.4 Librería TKINTER

## TKINTER - Programa Botón presionado.



```
import tkinter as tk

alfabeto = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def btPres(x):
    etiqueta.configure(text='Botón {} presionado'.format(alfabeto[x]))

raiz = tk.Tk()
raiz.geometry('480x100+100+100')
raiz.title("Identificar botón presionado")

etiqueta = tk.Label(font=('Calibri', 12))
etiqueta.grid(row=1, column=0, columnspan=26)
botones = [0] * 26 # crea una lista de 26 botones

for i in range(26):
    botones[i] = tk.Button(text=alfabeto[i], command=lambda x=i:
btPres(x))
    botones[i].grid(row=0, column=i)

raiz.mainloop()
```

Como hemos visto en los botones mediante la opción `command` se puede invocar una función. En un primer momento aparece una restricción que es que no se pueden pasar argumentos con la función, ya que en el caso de aportarlos la función se ejecuta cuando el botón es creado y no cuando capta una pulsación.

Una forma de conseguir pasar parámetros en usando “**funciones anónimas**” con la instrucción `lambda`

La palabra clave `lambda` indica que lo que sigue será una función anónima. Luego tenemos los **argumentos de la función anónima**, seguidos de **dos puntos** y luego el **código de la función**. El código de función no puede tener más de una línea, en otras palabras, para crear una función anónima se requiere:

- La palabra clave: `lambda`
- Las variables relacionadas: `x`
- las instrucciones a aplicar

Luego, el truco básicamente consiste en diferir la llamada creando una función intermedia que pasamos en la opción `command`

### 3.1.5 Librería NUMPY



NumPy es el paquete fundamental para la computación científica en Python. Es una **biblioteca de Python** que proporciona un objeto de matriz multidimensional, varios objetos derivados (como matrices y matrices enmascaradas) y una variedad de rutinas para operaciones rápidas en matrices, que incluyen manipulación matemática, lógica, de formas, clasificación, selección, E / S., transformadas discretas de Fourier, álgebra lineal básica, operaciones estadísticas básicas, simulación aleatoria y mucho más.

➤ Instalación de Numpy en ANACONDA

```
# Best practice, use an environment rather than install in the base env
```

```
conda create -n my-env
```

```
conda activate my-env
```

```
conda config --env --add channels conda-forge # If you want to install from conda-forge
```

```
# The actual install command
```

```
conda install numpy
```

➤ Manual de la librería

<https://numpy.org/doc/stable/user/quickstart.html>

➤ Manual de clase ndarray \_ métodos

<https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html?highlight=ndarray#numpy.ndarray>

### 3.1.5 Librería NUMPY



- NumPy tiene como objetivo proporcionar un objeto de matriz que sea hasta 50 veces más rápido que las listas tradicionales de Python. El **objeto de matriz** en NumPy se llama **ndarray**, proporciona muchas funciones de apoyo que hacen que trabajar con ndarray sea muy fácil.
- Importar Numpy

Donde **Numpy** es la biblioteca, **np** es el alias para llamar a las funciones dentro del archivo Python.

```
import numpy as np
```

```
print(np.__version__)
```

# Para imprimir la versión

#### Operaciones de arreglos con Numpy

- Creando un arreglo en Numpy

```
import numpy as np  
arr = np.array([1, 2, 3, 4, 5])  
print(arr)  
print(type(arr))
```

# Debe imprimir la lista

# Debe imprimir que es una clase ndarray de Numpy

- Creando un arreglo en Numpy desde una tupla

```
import numpy as np  
arr = np.array((1, 2, 3, 4, 5))  
print(arr)
```

# Debe dibujarle el grafico con las características personalizadas

## 3.1.5 Librería NUMPY



### Operaciones de arreglos con Numpy

#### ➤ Creando un arreglo 2D en Numpy

```
import numpy as np  
arr = np.array([[9, 8, 7], [4, 5, 6]])  
print(arr)
```

# Debe imprimir la lista

#### ➤ Creando un arreglo 3D en Numpy

```
import numpy as np  
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])  
print(arr)
```

# Debe imprimir la lista de 3 filas

#### ➤ Verificando el numero de dimensiones

```
import numpy as np  
a = np.array(42)  
b = np.array([1, 2, 3, 4, 5])  
c = np.array([[1, 2, 3], [4, 5, 6]])  
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])  
  
print(a.ndim)  
print(b.ndim)  
print(c.ndim)  
print(d.ndim)
```

# Debe imprimir las dimensiones de los arreglos

## 3.1.5 Librería NUMPY

### Operaciones de arreglos con Numpy



#### ➤ Creando un arreglo con 5 dimensiones en Numpy

```
import numpy as np  
arr = np.array([1, 2, 3, 4], ndmin=5)  
print(arr)  
print('numero de dimensiones :', arr.ndim)
```

# Debe imprimirle la lista

#### ➤ Uso de arreglos con 2D en Numpy

```
import numpy as np  
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])  
print('2do elemento de la 1ra dim: ', arr[0, 1])
```

# Debe imprimir el numero 2

#### ➤ Uso de arreglos con 2D en Numpy con índices negativos

```
import numpy as np  
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])  
print("Ultimo elemento de 2da dim: ", arr[1, -1])
```

# Debe imprimir el numero 10

#### ➤ Slicing y STEP en vectores

```
import numpy as np  
arr = np.array([1, 2, 3, 4, 5, 6, 7])  
print("Los elementos pares cada dos posiciones, hasta 4: ", arr[1:5:2])  
print("Todos los elementos impares cada dos posiciones: ", arr[::-2])
```

# Debe imprimir el numero [2, 4]

# Debe imprimir el numero [2, 4]

### 3.1.5 Librería NUMPY



## Operaciones de arreglos con Numpy

## ➤ Slicing y STEP en matrices

```
import numpy as np  
arr = np.array ([[1,2,3,4,5], [6,7,8,9,10]])  
print("Los elementos de fila 2 y columnas 1 a 3 ", arr[1, 1:4])      # Debe imprimir el numero [6, 7, 8]  
print("Rangos en filas y columnas: ", arr[0:2, 1:4])                  # Debe imprimir el numero [2, 4]
```

## ➤ Uso de arreglos con 3D en Numpy

```
import numpy as np  
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])  
print(arr[0, 1, 2]) # Debe imprimir el numero 6
```

## 3.1.5 Librería NUMPY



### Operaciones de arreglos con Numpy

➤ Se muestran los tipos de datos de **NumPy** y los caracteres usados que los representan.

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type ( void )

➤ Verificar tipos de datos en Numpy

```
import numpy as np  
arr = np.array([[1, 2, 3], [4, 5, 6], [[7, 8, 9], [10, 11, 12]]])  
print(arr.dtype) # Debe imprimir integer
```

## 3.1.5 Librería NUMPY



### Operaciones de arreglos con Numpy

#### ➤ Convertir tipos de datos en Numpy

```
import numpy as np  
arr = np.array([1.1, 2.3, 3.4])  
nuevarr = arr.astype("I")  
print(nuevarr)  
print(nuevarr.dtype)
```

# El método **astype()** convierte al dato del parámetro  
# Debe imprimir el vector con enteros [1 2 3]  
# Debe imprimir int32

#### ➤ Diferencia entre copiar y ver de arreglos en Numpy

```
import numpy as np  
arr = np.array([1, 2, 3, 4, 5, 6])  
nuevarr = arr.copy()  
arr[0] = 100  
print("Vector original Arr ",arr)  
print("Vector copiado nuevo ", nuevarr)
```

# El método **copy()** crea una copia del arreglo  
# Se modifica el arreglo arr  
  
# nuevarr no se afecta con los cambios de arr

```
arr = np.array([1, 2, 3, 4, 5, 6])  
nueviewvarr = arr.view()  
arr[0] = 100  
print("Vector original Arr ",arr)  
print("Vector Vista ", nueviewvarr)
```

# El método **copy()** crea una copia del arreglo  
# Se modifica el arreglo arr  
  
# nuevarr si se afecta con los cambios de arr

## 3.1.5 Librería NUMPY



### Operaciones de arreglos con Numpy

#### ➤ Uso de shape en Numpy

```
import numpy as np  
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])  
print(arr.shape)
```

# Debe imprimir (2, 4) que dice que es un arreglo de 2 dimensiones y cada dimensión tiene 4 elementos

#### ➤ Uso de reshape en Numpy

```
import numpy as np  
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])  
nuevarr = arr.reshape(4, 3)  
print(nuevarr)
```

# Crea un nuevo arreglo de 4 filas y 3 columnas

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])  
nuevarr = arr.reshape(2, 3, 2)  
print(nuevarr)
```

# Crea un nuevo arreglo de 2 filas, 3 columnas y 2 elementos

## 3.1.5 Librería NUMPY

### Operaciones de arreglos con Numpy



#### ➤ Recorrer arreglos en Numpy

```
import numpy as np  
arr = np.array([1, 2, 3, 4])  
for x in arr:  
    print(x) # Debe imprimir cada elemento en una fila
```

#### ➤ Recorridos en matrices en Numpy

```
import numpy as np  
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])  
for x in arr:  
    for y in x:  
        print(y) # Debe imprimir cada elemento en una fila
```

#### ➤ Recorridos en matrices de altas dimensiones en Numpy

```
import numpy as np  
arr = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])  
for x in np.nditer(arr):  
    print(x) # Debe imprimir cada elemento en una fila
```

## 3.1.5 Librería NUMPY



### Operaciones de arreglos con Numpy

#### ➤ Recorridos en matrices de altas dimensiones en Numpy

```
import numpy as np  
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])  
for x in np.nditer(arr[:, ::2]):           # Debe imprimir cada elemento saltando 1  
    print(x)
```

#### ➤ Recorridos en matrices de altas dimensiones e índices en Numpy

```
import numpy as np  
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])  
for idx, x in np.ndenumerate(arr):         # Debe imprimir el índice y el elemento  
    print(idx, x)
```

#### ➤ Unir matrices en Numpy

```
import numpy as np  
arr1 = np.array([1, 2, 3, 4])  
arr2 = np.array([5, 6, 7, 8])  
arr = np.concatenate((arr1, arr2))          # Debe imprimir el índice y el elemento  
print(arr)
```

### 3.1.5 Librería NUMPY



## Operaciones de arreglos con Numpy

## ➤ Busquedas en matrices en Numpy

```
import numpy as np  
arr = np.array([1, 2, 3, 4, 5, 4, 4])  
x  = np.where(arr == 4)  
print(x) # Debe imprimir (array([3, 5, 6]),) significa los índices 3, 5 y 6
```

```
x = np.where(arr%2 == 0)  
print(x) # Debe imprimir elementos pares
```

## ➤ Ordenamiento de matrices en Numpy

```
import numpy as np  
arr = np.array([4, 2, 3, 1] , [8, 6, 7, 5])  
print(np.sort(arr)) # Debe imprimir los vectores ordenados
```

## 3.1.5 Librería NUMPY



### Aleatorios con Numpy

#### ➤ Generación de aleatorios en Numpy

```
from numpy import random  
x = random.randint(100)  
print(x)
```

# Debe imprimir un numero entero entre 0 y 100

#### ➤ Generación de aleatorios en Numpy

```
from numpy import random  
x = random.rand()  
print(x)
```

# Debe imprimir un flotante entre 0 y 1

#### ➤ Generación de aleatorios en arreglos en Numpy

```
from numpy import random  
x = random.randint(100, size=(5))  
print(x)  
  
x = random.randint(100, size=(3, 5))  
print(x)  
  
x = random.rand(3, 5)  
print(x)
```

# Debe imprimir un vector de 5 elementos con enteros entre 0 y 100

# Debe imprimir una matriz de 3 x 5 elementos con enteros entre 0 y 100

# Debe imprimir una matriz de 3 x 5 elementos con flotantes entre 0 y 1

## 3.1.5 Librería NUMPY



### Aleatorios con Numpy

#### ➤ Selección aleatoria de elementos en arreglos en Numpy

```
from numpy import random  
x = random.choice([3, 5, 8, 15, 18])  
print(x)                                # Debe imprimir uno de los elementos del vector escogido aleatoriamente
```

```
from numpy import random  
x = random.choice([3, 5, 8, 15, 18], size=(3, 5))  
print(x)                                # Debe imprimir una matriz 3x5 con elementos del vector escogido aleatoriamente
```

#### ➤ Distribución aleatoria en Numpy

Una distribución aleatoria es un conjunto de números aleatorios que siguen una determinada función de densidad de probabilidad

```
from numpy import random  
x = random.choice([1, 3, 5, 7], p=[0.2, 0.3, 0.4, 0.1], size=(100))  
print(x)  
  
x = random.choice([1, 3, 5, 7], p=[0.2, 0.3, 0.4, 0.1], size=(3,5))  
print(x)
```

## 3.1.5 Librería NUMPY



### Aleatorios con Numpy

#### ➤ Permutaciones aleatorias en Numpy

```
from numpy import random  
import numpy as np  
arr = np.array([1, 3, 5, 7, 9])
```

```
random.shuffle(arr)  
print(arr)  
  
print(random.permutation(arr))
```

# **Shuffle** hace cambios en el arreglo original  
# Debe imprimir el vector con los elementos en otra disposición

# Debe imprimir el vector con los elementos en otra disposición **no lo modifica**

## 3.1.5 Librería NUMPY



### Distribuciones con Numpy, Matplotlib y Seaborn (<https://seaborn.pydata.org/>)

#### ➤ Distribución Normal en Numpy

```
from numpy import random  
arr = random.normal(size=(2, 3))  
print(arr)
```

# Genera números aleatorios normales N(0,1)  
# Debe imprimir una matriz 2x3 con los elementos flotantes Normales

```
from numpy import random  
arr = random.normal(loc=1, scale=2, size=(2, 3)) # Genera números aleatorios normales con media=1 y desv estándar= 2  
print(arr)
```

# Debe imprimir una matriz 2x3 con los elementos flotantes Normales

```
from numpy import random  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
sns.distplot(random.normal(size=1000), hist=False)  
plt.show()
```

## 3.1.5 Librería NUMPY



### Distribuciones con Numpy, Matplotlib y Seaborn (<https://seaborn.pydata.org/>)

#### ➤ Distribución Binomial en Numpy

```
from numpy import random  
x = random.binomial(n=10, p=0.5, size=10)  
print(x) # Debe imprimir una matriz 2x3 con los elementos flotantes Normales
```

```
from numpy import random  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
sns.distplot(random.binomial(n=10, p=0.5, size=1000), hist= True, kde=False)  
plt.show()
```

#### ➤ Diferencia entre la Distribución Normal y la Binomial en Numpy

```
from numpy import random  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
sns.distplot(random.normal(loc=50, scale=5, size=1000), hist=False, label='normal')  
sns.distplot(random.binomial(n=100, p=0.5, size=1000), hist=False, label='binomial')  
plt.show()
```

## 3.1.5 Librería NUMPY



### Distribuciones con Numpy, Matplotlib y Seaborn (<https://seaborn.pydata.org/>)

#### ➤ Distribución Poisson en Numpy

```
from numpy import random  
x = random.poisson(lam=2, size=10)  
print(x) # Debe imprimir un vector de 10 elementos con Dist. Poisson  
  
from numpy import random  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
sns.distplot(random.poisson(lam=2, size=1000), kde=False)  
plt.show()
```

#### ➤ Diferencia entre la Distribución Normal y la Poisson en Numpy

```
from numpy import random  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
sns.distplot(random.normal(loc=50, scale=5, size=1000), hist=False, label='normal')  
sns.distplot(random.poisson(lam=2, size=1000), hist=False, label='poissonl')  
plt.show()
```

### 3.1.6 Librería PANDAS



Pandas es una librería de Python especializada en el manejo y análisis de estructuras de datos.

➤ Las principales características de esta librería son:

- Define nuevas estructuras de datos basadas en los arrays de la librería NumPy pero con nuevas funcionalidades.
- Permite leer y escribir fácilmente ficheros en formato CSV, Excel y bases de datos SQL.
- Permite acceder a los datos mediante índices o nombres para filas y columnas.
- Ofrece métodos para reordenar, dividir y combinar conjuntos de datos.
- Permite trabajar con series temporales.
- Realiza todas estas operaciones de manera muy eficiente.

Pandas dispone de Tipos de datos de Pandas. Las estructuras de datos son:

- Series: Estructura de una dimensión.
- DataFrame: Estructura de dos dimensiones (tablas).
- Panel: Estructura de tres dimensiones (cubos). Usado mayormente en Bigdata, Machine Learning, etc

➤ Documentación de la librería

<https://pandas.pydata.org/>

## 3.1.6 Librería PANDAS

### Estructuras de Pandas



#### ➤ La clase de objetos Series

Son estructuras similares a los arrays de una dimensión(**Vector**). Son homogéneas, es decir, sus elementos tienen que ser del mismo tipo, y su tamaño **es inmutable**, es decir, no se puede cambiar, aunque si su contenido. Dispone de un índice que asocia un nombre a cada elemento de la serie, a través de la cual se accede al elemento.

Ejemplo: Se tiene la siguiente serie con asignaturas del curso

Indices	I1	I2	I3	I4
Valores	Calculo Diferencial	Calculo Integral	Calculo Vectorial	Ecuaciones diferenciales

#### ➤ Creación de la serie a partir de una lista

```
import pandas as pd  
s1 = pd.Series(["Calculo Diferencial", "Calculo Integral", "Calculo Vectorial", "Ecuaciones diferenciales"])  
print(s1) # Debe imprimir índice y curso en una fila cada uno
```

#### ➤ Creación de la serie a partir de un diccionario

```
import pandas as pd  
s1 = pd.Series({"Calculo Diferencial": 16.0, "Calculo Integral": 13.5, "Calculo Vectorial":12.5, "Ecuaciones  
diferenciales":18.5})  
print(s1) # Debe imprimir curso y nota por cada fila
```

# Estructuras de Pandas

## ➤ Métodos de una serie

**s.size** : Devuelve el número de elementos de la serie s.

**s.index** : Devuelve una lista con los nombres de las filas del DataFrame s.

**s.dtype** : Devuelve el tipo de datos de los elementos de la serie s.

**s.count()** : Devuelve el número de elementos que no son nulos ni NaN en la serie s.

**s.sum()** : Devuelve la suma de los datos de la serie s cuando los datos son de un tipo numérico, o la concatenación de ellos cuando son del tipo cadena str.

**s.cumsum()** : Devuelve una serie con la suma acumulada de los datos de la serie s cuando los datos son de un tipo numérico.

**s.value\_counts()** : Devuelve una serie con la frecuencia (número de repeticiones) de cada valor de la serie s.

**s.min()** : Devuelve el menor de los datos de la serie s.

**s.max()** : Devuelve el mayor de los datos de la serie s.

**s.mean()** : Devuelve la media de los datos de la serie s cuando los datos son de un tipo numérico.

**s.std()** : Devuelve la desviación típica de los datos de la serie s cuando los datos son de un tipo numérico.

**s.concat()** : Combina o concatena dos series y entrega el valor a una nueva serie.

**s.describe()**: Devuelve una serie con un resumen descriptivo que incluye el número de datos, su suma, el mínimo, el máximo, la media, la desviación típica y los cuartiles.

**s.sort\_values(ascending=boolano)** : Devuelve la serie que resulta de ordenar los valores la serie s. Si argumento del parámetro ascending es True el orden es creciente y si es False decreciente.

**s.sort\_index(ascending=boolano)** : Devuelve la serie que resulta de ordenar el índice de la serie s. Si el argumento del parámetro ascending es True el orden es creciente y si es False decreciente.

## 3.1.6 Librería PANDAS



### Estructuras de Pandas

➤ La clase de objetos Series

➤ Aplicar funciones a una serie

También es posible aplicar una función a cada elemento de la serie mediante el siguiente método:

`s.apply(f)` : Devuelve una serie con el resultado de aplicar la función f a cada uno de los elementos de la serie s.

```
import pandas as pd
from math import log
s1 = pd.Series([1, 2, 3, 4])
s1.apply(log)
```

```
s = pd.Series(['a', 'b', 'c'])
s.apply(str.upper)
```

## 3.1.6 Librería PANDAS

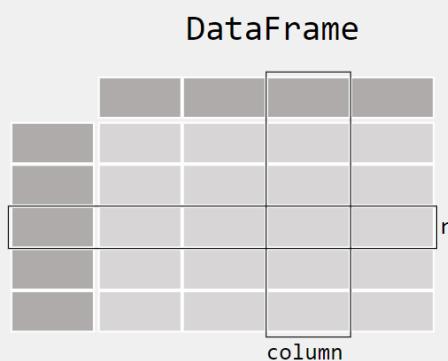


### Estructuras de Pandas

#### ➤ La clase de objetos DataFrames

Un objeto del tipo DataFrame define un conjunto de datos estructurado en forma de tabla donde cada columna es un objeto de tipo Series, es decir, todos los datos de una misma columna son del mismo tipo, y las filas son registros que pueden contener datos de distintos tipos.

Un DataFrame contiene dos índices, uno para las filas y otro para las columnas, y se puede acceder a sus elementos mediante los nombres de las filas y las columnas.



	Nombre	correo	Curso	Nota	Veces
1	Juan Pérez	<a href="mailto:Jperez@gmail.com">Jperez@gmail.com</a>	Calculo Integral	13.75	1
2	Francisco Quiroz	<a href="mailto:Fquiroz@gmail.com">Fquiroz@gmail.com</a>	Calculo Diferencial	18.5	2
3	Enrique Panta	<a href="mailto:Epanta@gmail.com">Epanta@gmail.com</a>	Calculo Vectorial	17.2	1

## 3.1.6 Librería PANDAS

### Estructuras de Pandas



#### ➤ Los métodos de DataFrames

- ✓ **df.info()** : Devuelve información (número de filas, número de columnas, índices, tipo de las columnas y memoria usado) sobre el DataFrame df.
- ✓ **df.shape** : Devuelve una tupla con el número de filas y columnas del DataFrame df.
- ✓ **df.size** : Devuelve el número de elementos del DataFrame.
- ✓ **df.columns** : Devuelve una lista con los nombres de las columnas del DataFrame df.
- ✓ **df.index** : Devuelve una lista con los nombres de las filas del DataFrame df.
- ✓ **df.dtypes** : Devuelve una serie con los tipos de datos de las columnas del DataFrame df.
- ✓ **df.head(n)** : Devuelve las n primeras filas del DataFrame df.
- ✓ **df.tail(n)** : Devuelve las n últimas filas del DataFrame df.
- ✓ **df.count()** : Devuelve una serie número de elementos que no son nulos ni NaN en cada columna del DataFrame df.
- ✓ **df.sum()** : Devuelve una serie con la suma de los datos de las columnas del DataFrame df cuando los datos son de un tipo numérico, o la concatenación de ellos cuando son del tipo cadena str.
- ✓ **df.concat()** : Concatena una serie a un DF.
- ✓ **df.cumsum()** : Devuelve un DataFrame con la suma acumulada de los datos de las columnas del DataFrame df cuando los datos son de un tipo numérico.

## 3.1.6 Librería PANDAS



### Estructuras de Pandas

#### ➤ Los métodos de DataFrames

- ✓ **df.min()** : Devuelve una serie con los menores de los datos de las columnas del DataFrame df.
- ✓ **df.max()** : Devuelve una serie con los mayores de los datos de las columnas del DataFrame df.
- ✓ **df.mean()** : Devuelve una serie con las media de los datos de las columnas del DataFrame df cuando los datos son de un tipo numérico.
- ✓ **df.std()** : Devuelve una serie con las desviaciones típicas de los datos de las columnas del DataFrame df cuando los datos son de un tipo numérico.
- ✓ **df.var()** : Devuelve una serie con las varianzas de los datos de las columnas numéricas del DataFrame df.
- ✓ **df.cov()** : Devuelve un DataFrame con las covarianzas de los datos de las columnas numéricas del DataFrame df.
- ✓ **df.corr()** : Devuelve un DataFrame con los coeficientes de correlación de Pearson de los datos de las columnas numéricas del DataFrame df.
- ✓ **df.describe(include = tipo)** : Devuelve un DataFrame con un resumen estadístico de las columnas del DataFrame df del tipo tipo. Para los datos numéricos (number) se calcula la media, la desviación típica, el mínimo, el máximo y los cuartiles de las columnas numéricas. Para los datos no numéricos (object) se calcula el número de valores, el número de valores distintos, la moda y su frecuencia. Si no se indica el tipo solo se consideran las columnas numéricas.

## 3.1.6 Librería PANDAS

### Estructuras de Pandas



#### ➤ Los métodos de DataFrames

✓ **df.append(serie, ignore\_index=True)** : Devuelve el DataFrame que resulta de añadir una fila al DataFrame df con los valores de la serie serie. Los nombres del índice de la serie deben corresponderse con los nombres de las columnas de df. Si no se pasa el parámetro ignore\_index entonces debe pasarse el parámetro name a la serie, donde su argumento será el nombre de la nueva fila.

```
df = df.append(pd.Series(['Carlos Rivas', 28, 'H', 89.0, 1.78, 245.0],  
index=['nombre','edad','sexo','peso','altura','colesterol']), ignore_index=True)
```

✓ **df.drop(filas)** : Devuelve el DataFrame que resulta de eliminar las filas con los nombres indicados en la lista filas del DataFrame df.

✓ **df[condicion]** : Devuelve un DataFrame con las filas del DataFrame df que se corresponden con el valor True de la lista booleana condicion. condicion debe ser una lista de valores booleanos de la misma longitud que el número de filas del DataFrame.

✓ **df.sort\_values(columna, ascending=booleano)** : Devuelve el DataFrame que resulta de ordenar las filas del DataFrame df según los valores de la columna con nombre columna. Si argumento del parámetro ascending es True el orden es creciente y si es False decreciente.

✓ **df.sort\_index(ascending=booleano)** : Devuelve el DataFrame que resulta de ordenar las filas del DataFrame df según los nombres de las filas. Si el argumento del parámetro ascending es True el orden es creciente y si es False decreciente.

## 3.1.6 Librería PANDAS



### Estructuras de Pandas

#### ➤ Los métodos de DataFrames

- ✓ **df.groupby(columnas).groups** : Devuelve un diccionario con cuyas claves son las tuplas que resultan de todas las combinaciones de los valores de las columnas con nombres en la lista columnas, y valores las listas de los nombres de las filas que contienen esos valores en las correspondientes columnas del DataFrame df.
- ✓ **df.groupby(columnas).get\_group(valores)** : Devuelve un DataFrame con las filas del DataFrame df que cumplen que las columnas de la lista columnas presentan los valores de la tupla valores. La lista columnas y la tupla valores deben tener el mismo tamaño.
- ✓ **df.groupby(columnas).agg(funciones)** : Devuelve un DataFrame con el resultado de aplicar las funciones de agregación de la lista funciones a cada uno de los DataFrames que resultan de dividir el DataFrame según las columnas de la lista columnas.

<https://pandas.pydata.org/pandas-docs/stable/reference/groupby.html>

## 3.1.6 Librería PANDAS

### Estructuras de Pandas



#### ➤ La clase de objetos DataFrames

##### ➤ Creación de un dataframe desde un diccionario de listas

**DataFrame(data=diccionario, index=filas, columns=columnas, dtype=tipos)** : Devuelve un objeto del tipo DataFrame cuyas columnas son las listas contenidas en los valores del diccionario, los nombres de filas indicados en la lista filas, los nombres de columnas indicados en la lista columnas y los tipos indicados en la lista tipos. La lista filas tiene que tener el mismo tamaño que las listas del diccionario, mientras que las listas columnas y tipos tienen que tener el mismo tamaño que el diccionario. Si no se pasa la lista de filas se utilizan como nombres los enteros empezando en 0. Si no se pasa la lista de columnas se utilizan como nombres las claves del diccionario. Si no se pasa la lista de tipos, se infiere.

**Nota.- Los valores asociados a las claves del diccionario deben ser listas del mismo tamaño.**

```
import pandas as pd
import numpy as np
df = pd.DataFrame( np.array([ ["Juan Pérez","Jperez@gmail.com", "Calculo Integral", 13.75, 1],
                            ["Francisco Quiroz", "FQuiroz@gmail.com", "Calculo Diferencial", 18.5, 2],
                            ["Enrique Panta", "Epanta@gmail.com", "Calculo Vectorial", 17.2, 1] ]),
                    columns=[ "nombre", "correo", "curso", "nota", "veces" ] )
print(df)
```

## 3.1.6 Librería PANDAS



### Estructuras de Pandas

#### ➤ La clase de objetos DataFrames

##### ➤ Creación de un DataFrame a partir de un fichero CSV o Excel

**read\_csv(fichero.csv, sep=separador, header=n, index\_col=m, na\_values=no-validos, decimal=separador-decimal)** : Devuelve un objeto del tipo DataFrame con los datos del fichero CSV fichero.csv usando como separador de los datos la cadena separador. Como nombres de columnas se utiliza los valores de la fila n y como nombres de filas los valores de la columna m. Si no se indica m se utilizan como nombres de filas los enteros empezando en 0. Los valores incluidos en la lista no-validos se convierten en NaN. Para los datos numéricos se utiliza como separador de decimales el carácter indicado en separador-decimal.

**read\_excel(fichero.xlsx, sheet\_name=hoja, header=n, index\_col=m, na\_values=no-validos, decimal=separador-decimal)** : Devuelve un objeto del tipo DataFrame con los datos de la hoja de cálculo hoja del fichero Excel fichero.xlsx. Como nombres de columnas se utiliza los valores de la fila n y como nombres de filas los valores de la columna m. Si no se indica m se utilizan como nombres de filas los enteros empezando en 0. Los valores incluidos en la lista no-validos se convierten en NaN. Para los datos numéricos se utiliza como separador de decimales el carácter indicado en separador-decimal

```
import pandas as pd  
df = pd.read_csv(  
    'https://raw.githubusercontent.com/asalber/manual-python/master/datos/colesteroles.csv', sep=';', decimal=',')  
print(df.head())
```

# Importación del fichero datos-colesteroles.csv

## 3.1.6 Librería PANDAS



### Estructuras de Pandas

- La clase de objetos DataFrames
  - Leer desde un DataFrame

```
import pandas as pd
df = pd.read_excel(io = "fNotSem.xlsx", sheet_name="Notas")
print(df)
Prom=df["nota"].mean()
Mini=df["nota"].min()
Maxi=df["nota"].max()
print("El promedio es", Prom)
print("La nota minima es ", Mini)
print("La nota maxima es ", Maxi)
```

```
1 import pandas as pd
2 df = pd.read_excel(io = "fNotSem.xlsx", sheet_name="Notas")
3 print(df)
4 Prom=df["nota"].mean()
5 Mini=df["nota"].min()
6 Maxi=df["nota"].max()
7 print("El promedio es", Prom)
8 print("La nota minima es ", Mini)
9 print("La nota maxima es ", Maxi)
```

```
↳      Unnamed: 0          nombre ...     nota    veces
0           0       Juan Pérez ...  13.75      1
1           1   Francisco Quiroz ...  18.50      2
2           2       Enrique Panta ...  17.20      1
3           3        Luis Quiroz ...  17.00      2
```

```
[4 rows x 6 columns]
El promedio es 16.6125
La nota minima es 13.75
La nota maxima es 18.5
```

## 3.1.6 Librería PANDAS



### Estructuras de Pandas

- La clase de objetos DataFrames
  - Lectura de un DataFrame y grabar un fichero CSV o Excel

**df.to\_csv(fichero.csv, sep=separador, columns=booleano, index=booleano) :**

Exporta el DataFrame **df** al fichero **fichero.csv** en formato CSV usando como separador de los datos la cadena **separador**. Si se pasa True al parámetro **columns** se exporta también la fila con los nombres de columnas y si se pasa True al parámetro **index** se exporta también la columna con los nombres de las filas.

**df.to\_excel(fichero.xlsx, sheet\_name = hoja, columns=booleano, index=booleano) :**

Exporta el DataFrame **df** a la hoja de cálculo **hoja** del fichero **fichero.xlsx** en formato Excel. Si se pasa True al parámetro **columns** se exporta también la fila con los nombres de columnas y si se pasa True al parámetro **index** se exporta también la columna con los nombres de las filas.

```
import pandas as pd # Importación del fichero datos-colesteroles.csv
df = pd.DataFrame( np.array([["Juan Pérez", "Jperez@gmail.com", "Calculo Integral", 13.75, 1],
                           ["Francisco Quiroz", "FQuiroz@gmail.com", "Calculo Diferencial", 18.5, 2],
                           ["Enrique Panta", "Epanta@gmail.com", "Calculo Vectorial", 17.2, 1]]),
                    columns=[ "nombre", "correo", "curso", "nota", "veces" ] )  
  
df.to_excel('IN318.xlsx', sheet_name='Notas')
```

### 3.1.7 Librería OPENPYXL

## Librería OpenPyXL en Python



Openpyxl es una de esas bibliotecas que nos permite usar Excel para realizar pruebas basadas en datos e incluso para preparar informes de resultados de pruebas, etc.

- Instalación: desde línea de comandos de Anaconda escribir: `pip install openpyxl`

<https://openpyxl.readthedocs.io/en/stable/>

```
C:\Users\Admin\AppData\Local\Programs\Python\Python38-32>pip install openpyxl
Collecting openpyxl
  Downloading https://files.pythonhosted.org/packages/f9/d8/be9dc2b17ba47f1db9032ed7e19915/openpyxl-3.0.4-py2.py3-none-any.whl (241kB)
    |████████| 245kB 1.3MB/s
Collecting et-xmlfile (from openpyxl)
  Downloading https://files.pythonhosted.org/packages/22/28/a99c42aea746e18382ad9fb36f64c1c/et_xmlfile-1.0.1.tar.gz
Collecting jdcal (from openpyxl)
  Downloading https://files.pythonhosted.org/packages/f0/da/572cbc0bc582390480bbd7c4e93d14/jdcal-1.4.1-py2.py3-none-any.whl
Installing collected packages: et-xmlfile, jdcal, openpyxl
  Running setup.py install for et-xmlfile ... done
Successfully installed et-xmlfile-1.0.1 jdcal-1.4.1 openpyxl-3.0.4
WARNING: You are using pip version 19.2.3, however version 20.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Users\Admin\AppData\Local\Programs\Python\Python38-32>
```

- Verificación:

```
C:\Users\Admin\AppData\Local\Programs\Python\Python38-32>pip list
Package      Version
-----
et-xmlfile   1.0.1
jdcal        1.4.1
openpyxl     3.0.4
pip          19.2.3
selenium     3.141.0
setuptools   41.2.0
urllib3      1.25.9
WARNING: You are using pip version 19.2.3, however version 20.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Users\Admin\AppData\Local\Programs\Python\Python38-32>
```

### 3.1.7 Librería OPENPYXL

## Librería OpenPyXL en Python



Openpyxl es una de esas bibliotecas que nos permite usar Excel para realizar pruebas basadas en datos e incluso para preparar informes de resultados de pruebas, etc.

#### ➤ Instalación Anaconda:

En línea de comandos de Anaconda, escribir

```
conda install -c anaconda openpyxl
```

```
C:\WINDOWS\system32\cmd. > + - X

(base) C:\Users\jesws>conda install -c anaconda openpyxl
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

environment location: C:\Users\jesws\anaconda3

added / updated specs:
- openpyxl

The following packages will be downloaded:

  package          |      build
  --> ca-certificates-2022.4.26 | haa95532_0    163 KB  anaconda
  certifi-2022.6.15   | py39haa95532_0 157 KB  anaconda
  et_xmlfile-1.1.0   | py39haa95532_0   9 KB   anaconda
  openpyxl-3.0.9     | pyhd3eb1b0_0    159 KB  anaconda
  Total:           488 KB

The following NEW packages will be INSTALLED:

  et_xmlfile        anaconda/win-64::et_xmlfile-1.1.0-py39haa95532_0 None
  openpyxl          anaconda/noarch::openpyxl-3.0.9-pyhd3eb1b0_0 None

The following packages will be SUPERSEDED by a higher-priority channel:
```

### 3.1.7 Librería OPENPYXL

## Librería OpenPyXL en Python



Openpyxl es una de esas bibliotecas que nos permite usar Excel para realizar pruebas basadas en datos e incluso para preparar informes de resultados de pruebas, etc.

#### ➤ Instalación:

- Para poder introducir datos en nuestra hoja de calculo, debemos utilizar `load_workbook`. Creamos una variable que contendrá la ruta de nuestro archivo creado anteriormente. Creamos el objeto `load_workbook` y le pasamos como argumento la ruta del archivo en el cual queremos escribir.
- Aplicamos `active` sobre el objeto `load_workbook`, guardándolo en una variable (`sheet`). La variable `sheet` ahora es un diccionario cuyas llaves son las celdas de la hoja de calculo. Así que para escribir en una celda utilizamos: `sheet[CELDA]` y asignamos los valores que contendrá esa celda. Por ultimo guardamos los cambios.

### 3.1.7 Librería OPENPYXL

## Librería OpenPyXL en Python

- Código de Ejemplo:

```
# importar modulo Workbook desde la librería openpyxl
from openpyxl import Workbook
wb = Workbook()

# activar el objeto workbook creado (wb)
ws = wb.active

# Se puede asignar un valor a una celda, p.e. 42 a la celda A1
ws['A1'] = 42

# Se puede agregar una fila
ws.append([1, 2, 3])

# Python convierte automáticamente un objeto fecha de la librería datetime
ws['A2'] = datetime.datetime.now()

# Salvar el archivo
wb.save("sample.xlsx")
```



### 3.1.7 Librería OPENPYXL

## Librería OpenPyXL en Python



Openpyxl es una de esas bibliotecas que nos permite usar Excel para realizar pruebas basadas en datos e incluso para preparar informes de resultados de pruebas, etc.

```
# importamos load_workbook  
from openpyxl import load_workbook
```

```
# ruta de nuestro archivo  
filesheet = "./demosheet.xlsx"
```

```
# creamos el objeto load_workbook  
wb = load_workbook(filesheet)
```

```
# Seleccionamos el archivo  
sheet = wb.active
```

```
# Ingresamos el valor 56 en la celda 'A1'  
sheet['A1'] = 56
```

```
# Ingresamos el valor 1845 en la celda 'B3'  
sheet['B3'] = 1845
```

```
# Guardamos el archivo con los cambios  
wb.save(filesheet)
```

### 3.1.7 Librería OPENPYXL

## Librería OpenPyXL en Python



Existe un inconveniente con esta manera de ingresar datos. Y es que es muy tedioso y muy ineficiente ingresar datos celda por celda. Pero no te preocupes, existe una forma de ingresar grupos de datos. Vemos como:

```
# importamos load_workbook  
from openpyxl import load_workbook
```

```
# ruta de nuestro archivo  
filesheet = "./demosheet.xlsx"
```

```
# creamos el objeto load_workbook  
wb = load_workbook(filesheet)
```

```
# seleccionaos el archivo  
sheet = wb.active
```

```
# escribimos los datos con sus respectivas filas y columnas  
datos = [('id', 'nombre', 'edad'),  
          (0, "Jose", 35),  
          (1, "Carlos", 27),  
          (2, "Sofia", 24)]
```

```
# recorremos las columnas y escribimos los datos  
for row in datos:  
    sheet.append(row)
```

```
# guardamos los cambios  
wb.save(filesheet)
```

### 3.1.7 Librería OPENPYXL

## Librería OpenPyXL en Python



Si sabemos escribir, deberíamos saber leer ¿no es cierto?. Veamos como:

```
# importamos load_workbook  
from openpyxl import load_workbook  
  
# ruta de nuestro archivo  
filesheet = "./demosheet.xlsx"  
  
# creamos el objeto load_workbook  
wb = load_workbook(filesheet)  
  
# seleccionamos el archivo  
sheet = wb.active  
  
# Obtenemos el valor de la celda A1  
A1 = sheet['A1'].value
```

```
# Obtenemos el valor de la celda B5  
B5 = sheet['B5'].value  
  
# Obtenemos el valor de la celda C5  
C5 = sheet['C5'].value  
  
# Mostramos los valores  
celdas = [A1, B5, C5]  
for valor in celdas:  
    print(valor)
```

### 3.1.7 Librería OPENPYXL



<https://pypi.org/project/openpyxl/>

<https://openpyxl.readthedocs.io/en/latest/tutorial.html>

Video corto para ejemplos de Openpyxl -

<https://www.youtube.com/watch?v=WMPZMeAQX3Q>

[https://programacion.net/articulo/como\\_trabajar\\_con\\_archivos\\_excel\\_utilizando\\_python\\_1419](https://programacion.net/articulo/como_trabajar_con_archivos_excel_utilizando_python_1419)

<https://es.stackoverflow.com/questions/106761/como-crear-base-de-datos-y-despues-consultarlos-en-excel-a-traves-de-python?newreg=832a9ba2b7184877ae32e35aad1e44bc>

[exceltotal.com/base-de-datos-en-excel](http://exceltotal.com/base-de-datos-en-excel)

# INSERTAR NOMBRE DE ACTIVIDAD



## PASOS DE ACTIVIDAD individual /grupal:

- Ingresar al colab personal con su cuenta gmail
  
- Copia el enunciado del problema señalado por el profesor y péguelo en el colab
  
- Desarrolle el ejercicio junto con un compañero, ya sea en el laboratorio o en la sala grupal creada por el docente.

# Resolución de ejercicios

## ACTIVIDAD

Debe resolver los ejercicios señalados por el profesor que se encuentran en el documento:

[03 - 01 Programación Orientada a Objetos - Guía laboratorio y Casos propuestos.pdf](#)

Este documento se encuentra en el aula virtual en la unidad 3



# CONCLUSIONES

01

Las librerías son software desarrollado por terceros con funcionalidades avanzadas, específicas y especializadas que potencian a Python estándar. Son gratuitas

02

Las diferentes librerías se compatibles con otras y se potencian entre si.

03

Las librerías enseñadas en clase son las mas populares y son la base para estudios superiores de ciencia de los datos, big data y otros tópicos de inteligencia artificial.

04

Python estándar como las librerías han sido desarrolladas con la arquitectura orientada a objetos.

# BIBLIOGRAFÍA

- Direcciones electrónicas (hipervínculos en las diapositivas)
- LUTZ, MARK (2013) Learning Python. 5th Edition. California: O'Reilly.
- RAMALHO, LUCIANO (2015) Fluent Python: Clear, Concise, and Effective Programming (inglés) 1st Edición. California: O'Reilly.
-

# **Continúa con las actividades propuestas**

---



Material producido por la  
Universidad Peruana de  
Ciencias Aplicadas

Autor:

Norman Reyes Morales

COPYRIGHT © UPC  
2020 – Todos los  
derechos reservados