

CURSO

# Python aplicado a finanzas



## MÓDULO 1

# ¿Qué es y para qué **programar**?

- Por medio del lenguaje de programación le damos tareas a las computadoras
- Cada día más dispositivos necesitan ser programados
- Mercado laboral con demanda en aumento



# ¿Qué es un **algoritmo**?

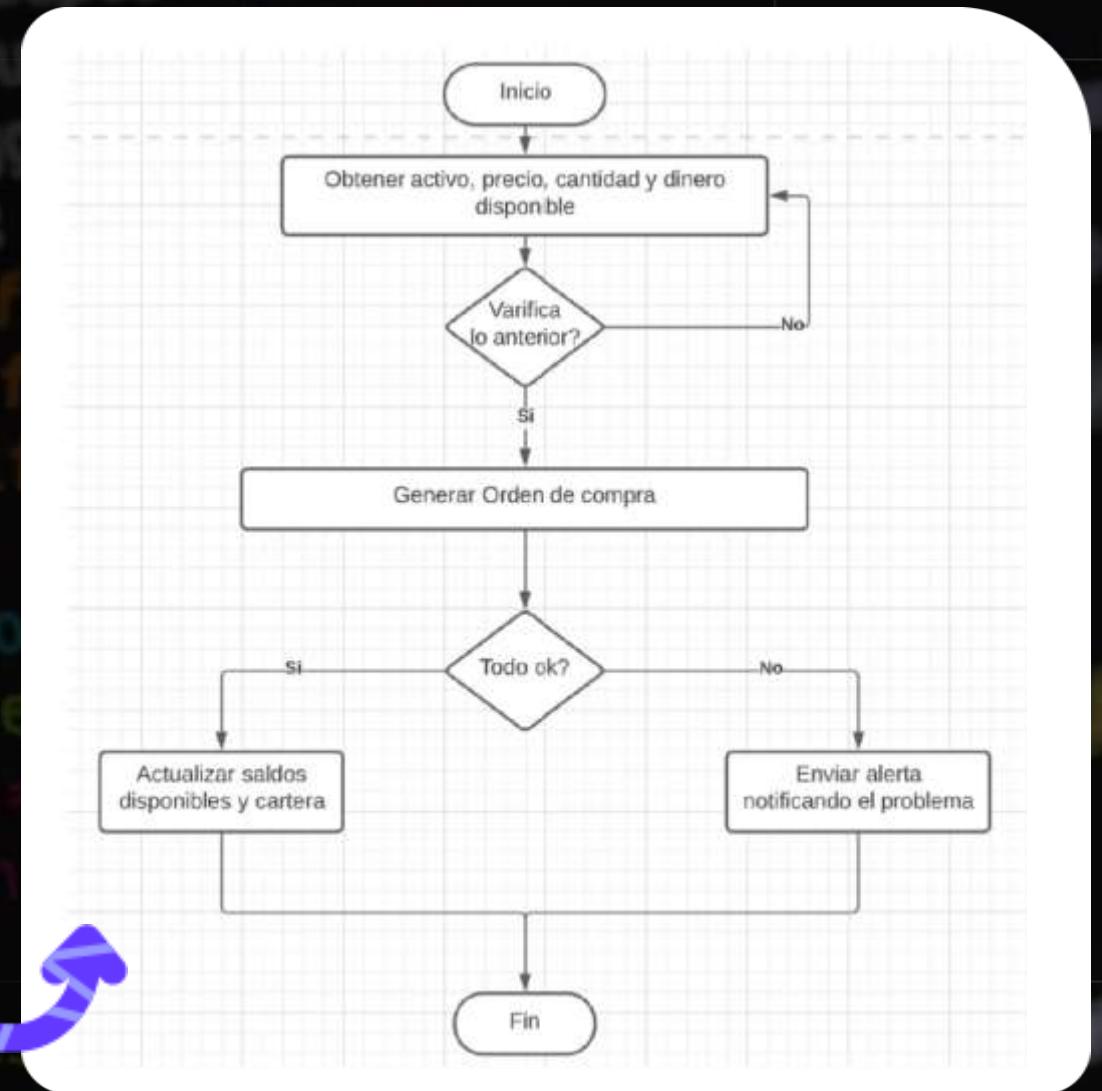
- Conjunto de instrucciones específicas y ordenadas para obtener un resultado concreto.
- Si por ejemplo quiero comprar un activo:



1. Definir activo, precio, cantidad y dinero disponible.
2. ¿Está definido el activo, el precio, la cantidad y dispongo del dinero suficiente?
  - a. Si: Crear la orden de compra.
  - b. No: Volver al punto 1.
3. ¿Todo ok?
  - a. Si: Actualizar saldos disponibles y cartera.
  - b. No: Enviar alerta avisando fallo y motivo.

# Diagrama de flujo

- Es la representación gráfica de un algoritmo



# Computador

## PARTES EXTERNAS

- Monitor
- Armazon: Carcasa o gabinete del CPU.
- Teclado.
- Mouse.
- Puertos y conectores.
- Parlantes.
- Micrófono.
- Cámara Web

## PARTES INTERNAS

- Procesador (CPU).
- Placa madre.
- Memoria RAM.
- Memoria ROM.
- Discos.
- Placa Base o placa madre.
- Tarjetas  
(Video, sonido, red)



# Computador

## a



## Vínculo entre partes (Externas e internas)

- BIOS
- Sistema Operativo (OS)
  - ✓ Windows
  - ✓ Linux
  - ✓ MacOS

# Python



- Es un lenguaje de programación interpretado, dinámico y multiplataforma.
- Su filosofía hace hincapié en la legibilidad de su código.
- Es multiparadigma ya que:
  - Soporta parcialmente la orientación a objetos.
  - Programación imperativa.
  - En menor medida, programación funcional.

# Herramientas de **trabajo**

## **Collaboratory:**

- Permite ejecutar y programar en Python en tu navegador sin requerir configuración y se guarda en la nube.
- Da acceso gratuito a GPUs y TPUs.
- Permite compartir contenido fácilmente.

## **repl.it:**

Un entorno de desarrollo online como colab.

## **Spyder:**

Entorno de desarrollo integrado multiplataforma de código abierto para programación científica en Python.

## **PyCharm:**

Entorno de desarrollo integrado para programación informática, para Python.



# Herramientas de **trabajo**

## **Jupyter Notebook:**

Entorno informático interactivo basado en la web para crear documentos de Jupyter Notebook:

- Es un documento JSON.
- Sigue un esquema versionado.
- Contiene una lista ordenada de celdas de entrada/salida que pueden contener.

Dentro de anaconda vienen instalados **Spyder** y **Jupyter Notebook**.

<https://repl.it/languages/python3>

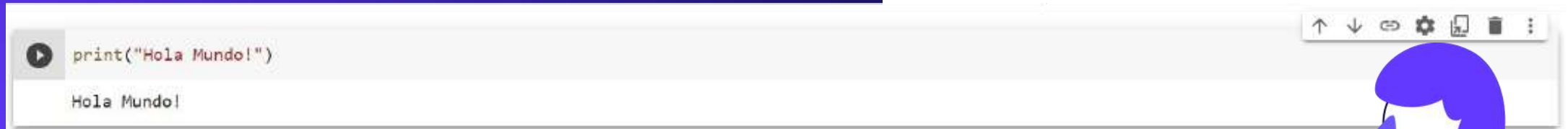
<https://colab.research.google.com/>

<https://jupyter.org/try>

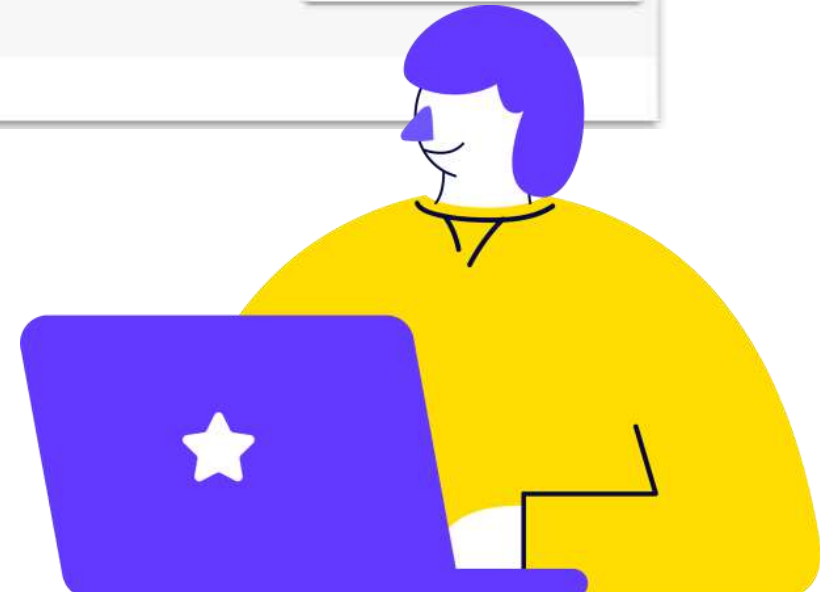
<https://www.anaconda.com/products/individual>

# Nuestro Primer Programa **Hola Mundo**

- Primer programa en Python:



A screenshot of a Python terminal window. The window has a light gray title bar with standard OS controls (minimize, maximize, close) on the right. Below the title bar, the command prompt shows the execution of the command `print("Hola Mundo!")`. The output of the command, `Hola Mundo!`, is displayed on the line immediately following the command. The terminal window is positioned horizontally across the middle of the slide.



# ¿Qué son las **variables**?

- Ingreso de valores por teclado por medio de la función **input()**

```
[2] nombre = "Ignacio Guardines"
```

```
[1] precio = 174.4  
precio
```

```
174.4
```

```
[3] cantidad_nominales_ypfd = 100
```

```
ingresenombre = input()
```



# ¿Qué son y cuáles son los tipos de **datos básicos**?

- Los tipos de datos son un conjunto de símbolos y operaciones que se pueden hacer con dichos símbolos:

## Números:

- Números enteros
- Números flotantes

## Valores de verdad (Booleanos)

- True
- False

## Cadena de caracteres o Strings (palabras)

```
[5] marketId = "NYSE"  
    ordType = "Limit"  
    side = "Buy"  
  
    price = 187.25  
    orderQty = 1000
```

```
[6] variable1, variable2 = "Hola", 2
```

# Segundo Programa en Python

- Uso de la función **print()**
- Concatenar palabras
- Otras funciones para manipular  
**Cadenas de Caracteres**

```
[ ] palabra = "hola" + " Mundo!"  
    print(palabra)
```

```
hola Mundo!
```



# Operadores y

## Operaciones

### Operadores y operaciones aritméticas:

Entorno informático interactivo basado en la web para crear documentos de **Jupyter Notebook**:

- "+" **Suma** ( $10 + 5 = 15$ )
- "-" **Resta** ( $10 - 5 = 5$ )
- "\*" **Multiplicación** ( $20 * 4 = 80$ )
- "/" **División** ( $21 / 5 = 4.2$ )
- "%" **Resto de la división**, también conocido como "módulo" ( $21 \% 5 = 1$ )
- "\*\*" **Potencia** ( $12 ** 2 = 144$ )
- "// **División entera** (Sin decimales) ( $22 // 5 = 4$ ;  $22.0 // 5.0 = 4.0$ )

```
[ ] resultado = 1000 + 0.15
```

```
[ ] resultado
```

```
1000.15
```

```
[ ] saldo_disponible = 10000 - 220  
saldo_disponible
```

```
9780
```

```
[ ] saldo_disponible = saldo_disponible - 262
```

```
[ ] saldo_disponible
```

```
9518
```

```
[ ] contador = 0  
contador = contador + 1  
contador = contador + 1  
contador = contador + 1  
contador = contador + 1  
contador
```

```
4
```

```
[ ] # Multiplicacion  
resultado = 2 * 3 * 2  
resultado
```

```
12
```

```
[ ] resultado8 = resultado / 5
```

```
[ ] resultado8
```

```
2.4
```

```
[ ] resultado // 5
```

```
2
```

# Operadores y Operaciones lógicas

- **or**: Devuelve un valor de verdad, es verdadero si al menos uno es verdadero.
- **and**: Devuelve un valor de verdad, es verdadero si ambas variables son verdaderas.
- **not**: Devuelve un valor de verdad, invierte el valor de verdad de la variable a la que afecta.

```
[ ] esHorarioMercado = True  
    esLunes = True  
    esLunes and esHorarioMercado
```

```
True
```

```
[ ] not esLunes
```

```
False
```

```
[ ] tengoMargen = True  
    compreDolarFuturo = False  
    tengoMargen and not compreDolarFuturo
```

```
True
```

```
[ ] # Tabla de verdad  
    # Y logico True cuando todo es True.  
    # O logico True cuando al menos uno es True.  
    # Not invierte el valor de verdad sobre el que opera.
```

# Operadores de relaciones

- Son símbolos que se usan para comparar dos valores.
- Si el resultado es correcto la expresión es verdadera, caso contrario es falsa.
- ">" **Mayor estricto**, devuelve un valor de verdad resultado si es o no mayor el contenido de la primer variable respecto de la segunda, devuelve True si es así, False caso contrario.
- "<" **Menor estricto**, devuelve un valor de verdad resultado si es o no menor el contenido de la primer variable respecto de la segunda, devuelve True si es así, False caso contrario.
- ">=" **Mayor o igual**, similar al mayor estricto, agrega la igualdad.
- "<=" **Menor o igual**, similar al menor estricto, agrega la igualdad.
- "==" **Igualdad**, devuelve True si ambas variables son iguales, False caso contrario.
- "!=" **Distinto**, devuelve True si ambas variables son distintas, False caso contrario.

```
[ ] 10 < 12
```

```
True
```

```
[ ] 16 != 15
```

```
True
```



# Estilos de escritura para nombres de variables de más de una palabra

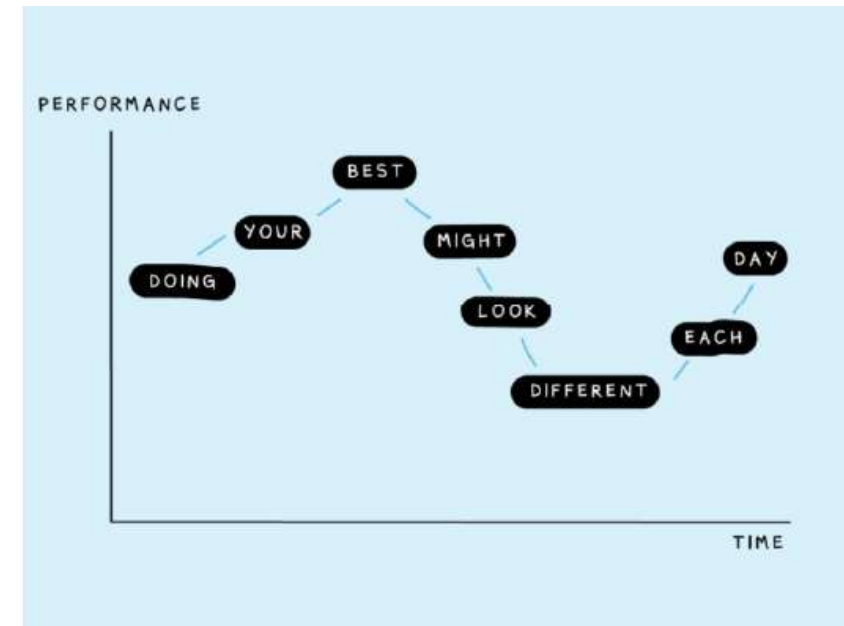


- camelCase
- snake\_case
- PascalCase
- CONSTANTS\_STYLE



# Operadores de Asignación

Operador	Ejemplo	Equivalencia
<code>+=</code>	<code>x += 2</code>	<code>x = x + 2</code>
<code>-=</code>	<code>x -= 2</code>	<code>x = x - 2</code>
<code>*=</code>	<code>x *= 2</code>	<code>x = x * 2</code>
<code>/=</code>	<code>x /= 2</code>	<code>x = x / 2</code>
<code>%=</code>	<code>x %= 2</code>	<code>x = x % 2</code>
<code>//=</code>	<code>x //= 2</code>	<code>x = x // 2</code>
<code>**=</code>	<code>x **= 2</code>	<code>x = x ** 2</code>



# Strings

- Son otro tipo de dato de Python utilizado para representar texto.
- Es una secuencia inmutable de caracteres.

- Ejemplos:

- "String válido con comillas dobles" ○ 'String válido con comillas simples'



```
self.fingerprints = set()
logdups = True
debug = debug
logger = logging.getLogger(__name__)
path:
self.file = open(os.path.join(path,
self.file.seek(0)
self.fingerprints.update(hashes)

method
om_settings(cls, settings)
bug = settings.get('debug', debug)
return cls(job_dir(settings),

request_seen(self, request):
p = self.request_fingerprint(request)
if fp in self.fingerprints:
```

# Métodos de los **Strings**

- **capitalize()** convierte el primer carácter en mayúscula.
- **count()** retorna la cantidad de veces que un carácter dado se encuentra en el string.
- **find()** busca un determinado string y devuelve la posición donde se encuentra.
- **index()** busca un determinado string y devuelve la posición donde se encuentra.
- **isdigit()** retorna True if all characters in the string are digits.
- **islower()** retorna True if all characters in the string are lower case.
- **join()** convierte los elementos de un iterable en un String.
- **lower()** convierte los caracteres de un string a minúsculas.
- **replace()** retorna un string donde un valor específico es reemplazado por otro.
- **split()** separa el string en un determinado separador, y retorna una lista.
- **title()** convierte la primera letra de cada palabra en un string.
- **upper()** convierte a mayúsculas todas las letras de una palabra.

# Ejemplos de Strings

```
[ ] "texto otro texto".capitalize()
```

```
'Texto otro texto'
```

```
[ ] " uno uno dos tres, cuatro".count("uno")
```

```
2
```

```
[ ] " uno uno dos tres, cuatro".find("uno")
```

```
1
```

```
[ ] nombre = "Ignacio"
    numero = 12.5
    mensaje = f"{nombre} les saluda y les dice el numero {numero}"
    print(mensaje)

    nuevo_mensaje = nombre + " les saluda y les dice el numero " + str(numero)
    print(nuevo_mensaje)
```

```
Ignacio les saluda y les dice el numero 12.5
```

```
Ignacio les saluda y les dice el numero 12.5
```

# Tipos de datos

## combinados

### COLECCIONES

- Una colección permite agrupar varios objetos bajo un mismo nombre.
- En Python existen 3 colecciones básicas:
  - Listas
  - Tuplas
  - Dicionarios



# Listas

- Son mutables.
- Permiten modelar conjuntos de elementos.
- Pueden ser del mismo o diferente tipo.
- Se acceden por índice, por lo que tienen un orden y cuentan con varias operaciones propias.

```
[1] mi_lista = []  
    mi_lista = list()
```

```
[3] mi_lista.append("primero")
```

```
[2] mi_lista  
  
[]
```

```
[5] mi_lista.append("segundo")
```

```
[6] mi_lista  
  
['primero', 'segundo']
```



# Métodos de Listas

- `index(n)` devuelve el orden dentro de la lista del elemento "n"
- `count()` cuenta la cantidad de elementos
- `sort (reverse=False)`
- `append()` agrega un valor al final
- `extend()` agrega una lista entera de valores
- `insert(x, "n")` inserta el elemento "n" en la posición "x"
- `pop ("n")` "saca" el elemento "n" y me lo devuelve como return
- `reverse()` invierte el orden dado
- `clear()` borra los elementos de la lista

# Intervalos

[desde : hasta : paso]

- "Desde" siempre es inclusive
- "Hasta" siempre es inclusive
- `[:]` todo
- `[:-1]` orden inverso



# Tuplas

- Similares a las listas pero son inmutables (no se pueden modificar)
- Se usan como conjuntos y también para modelar



```
[10] ordenCompra = (30, "buy", "DoDic20", 2, 72000)
```

```
▶ print(ordenCompra[3])
```

```
2
```

# Diccionarios

{clave : valor}

- El par clave, contenido suele llamarse elemento
- Cada bloque de información tiene asociada una palabra
- Para encontrar este bloque usamos la clave
- No pueden existir dos elementos con igual clave

y = { Co : Vo , C1 : V1 , ... }

```
self.file = ...
self.fingerprints = ...
self.logdups = True
self.debug = debug
self.logger = logging.getLogger(__name__)
if path:
    self.file = open(os.path.join(path, 'log.txt'), 'a')
    self.file.seek(0)
    self.fingerprints.update({
        path: self.file.read()
    })

@classmethod
def from_settings(cls, settings):
    debug = settings.getboolean('DEBUG', False)
    return cls(job_dir(settings))

def request_seen(self, request):
    fp = self.request_fingerprint(request)
    if fp in self.fingerprints:
```

# Librerías de tiempos

datetime, time, timedelta, date

```
[3] import datetime
```

```
[ ]
```

```
[2] import datetime as dt
```

```
[4] dt.date.today()
```

```
datetime.date(2021, 11, 10)
```

```
[5] import datetime as dt  
hoy = dt.date.today()  
type(hoy)  
print(hoy)
```

```
2021-11-10
```



# Funciones



- Bloque de código con un nombre asociado.
- Recibe cero, uno o más argumentos como entrada.
- Sigue una secuencia de sentencias, ejecuta una operación deseada y devuelve un valor y/o realiza una tarea

- Ventajas:
  - **Modularización:** Permite segmentar un programa complejo en una serie de partes o módulos más simples.
  - **Reutilización:** Permite reutilizar la misma función en distintos programas. Python dispone de funciones integradas al lenguaje y también permite crearlas.
- Se utiliza la palabra "def" para crear una función definida por el usuario, seguido por el nombre de la función y la lista de parámetros

# Bloque de Códigos

- Grupo de sentencias relacionadas bien delimitadas.
- En Python se utiliza la indentación o sangría.
- La sangría o indentación consiste en mover un bloque de texto hacia la derecha insertando espacios o tabuladores al comienzo de la línea.

```
[ ] ##  
    def esElPrimeroMayor(numero1, numero2, numero3):  
        return numero1 > numero2 and numero1 > numero3
```

```
[ ] print(esElPrimeroMayor(1,2,3)) # debe dar falso  
    print(esElPrimeroMayor(3,2,3)) # debe dar falso  
    print(esElPrimeroMayor(6,2,3)) # Aca si
```

```
False  
False  
True
```

```
[ ] def sumar(x1,x2):  
    return x1 + x2
```

```
[ ] def sumar( x1, x2 ):  
    return 0
```

```
[ ] #Definimos la funcion  
    def saludar(quien):  
        print("¡Hola!", quien)
```

```
[ ] #usamos la funcion  
    saludar("Agus")  
    saludar("Ale")  
    saludar("Moni")
```

```
¡Hola! Agus  
¡Hola! Ale  
¡Hola! Moni
```

# Librerías **Random**

- Contiene una serie de funciones relacionadas con los valores aleatorios.
- El listado completo de funciones se describe en el manual de Python.

```
[7] ###primero debemos importar la libreria
import random as rd # podemos agregarle un alias

rd.randrange(10)
```

7

```
[8] import random
#from random import randrange, choice
conjunto = ["Raquel", "Gustavo", "Nacho", "hernan", "Bruno", "Sofi"]
print(random.choice(conjunto))
```

Sofi



# Estructuras de Control Condicionales

## Control de flujo:

- if
- elif
- else



```
fingerprints = set()
logdups = True
debug = debug
logger = logging.getLogger(__name__)
path:
self.file = open(os.path.join(path, 'log.txt'), 'w')
self.file.seek(0)
self.fingerprints.update([fingerprint])

def __init__(cls, settings):
    debug = settings.getbool('debug', False)
    return cls(job_dir(settings), debug)

def request_seen(self, request):
    fp = self.request_fingerprint(request)
    if fp in self.fingerprints:
```



# Estructuras de control

## Condicionales

### IF

- La sentencia **"if"**, significa que si se cumple la expresión condicional se ejecuta el bloque de sentencias seguidas

```
if True:
    print("siempre se va a ejecutar")
    print("Sigo dentro del codigo")

    print("¿Aca sigo dentro del bloque?")

if False:
    print("nunca se va a ejecutar")

if not False:
    print("esto se ejecuta?")
```

siempre se va a ejecutar  
Sigo dentro del codigo  
¿Aca sigo dentro del bloque?  
esto se ejecuta?



# Estructuras de control

## Condicionales

### ELIF

- La sentencia **"elif"**, significa que si de lo contrario se cumple la expresión condicional se ejecuta el bloque de sentencias seguidas.

```
[ ] if hay_oportunidad_compra:  
    print("comprar")  
elif hay_oportunidad_holdear:  
    print("mantener")  
else:  
    print("vende")
```

# Estructuras de control

## Condicionales


### ELSE

- La sentencia **"else"**, significa que si de lo contrario se cumple sin evaluar ninguna expresión condicional y ejecuta el bloque de secuencias seguidas.
- Es opcional.

```
[ ] if False:  
    pass  
    else:  
        print("otra cosa que se va a imprimir")
```

# Algunos Operadores y Expresiones Condicionales

- Operador **"is"**: Prueba identidad, ambos lados de la expresión condicional deben ser el mismo objeto.
- Operador **"in"**: Cualquier colección del valor del lado izquierdo que contenga el valor del lado derecho.
- Operador **"not in"**: Contrario del operador **"in"**.



```
uno = 1
uno is 1
```

True

```
[ ] 4 in [1,2,3]
```

False

```
[ ] 1 not in [3,2,0]
```

True

# Estructuras de Control Iterativas

## Ciclo definido "for"

```
▶ print("Empiezo")  
  
for _ in [0, 1, 2]:  
    print("Hola")  
  
print("Termino")
```

Empiezo  
Hola  
Hola  
Hola  
Termino

## Iterando una tupla

```
▶ listado = ("AAPL", "AMZN", "NFLX", "FB")  
for a in listado:  
    print(a)  
  
print(a)
```

AAPL  
AMZN  
NFLX  
FB  
FB



# Estructuras de Control Iterativas

- Ciclo "while"

- Bibliografía :

[https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion4/bucle\\_while.html](https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion4/bucle_while.html)

```
▶ i = 0
  while i < 5:
    ##ejecuto codigo
    print(i)
    i +=1
```

0  
1  
2  
3  
4



# Un algoritmo de **compra / venta**

